Федеральное государственное бюджетное учреждение
«Национальный исследовательский центр «Курчатовский институт»

*На правах рукописи*

Рыбаков Алексей Анатольевич

# Методы и средства повышения производительности программ для моделирования объектов со сложной геометрией

Специальность 2.3.5 –
«Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей»

Диссертация на соискание ученой степени
доктора технических наук

Москва – 2026

**Оглавление**

**Введение** **6**

**Глава 1. Вычисления в условиях сложной геометрии** **21**

1.1 Расчетные сетки в условиях сложной геометрии . . . . . 21

1.2 Распараллеливание на блочно-структурированных сетках . . . . 25

1.3 Распараллеливание на поверхностных сетках . . . . . 26

1.4 Перестроение поверхностной сетки . . . . . 30

1.5 Векторизация вычислений . . . . . 33

1.6 Использование искусственного интеллекта при моделировании объектов со сложной геометрией . . . . . 37

1.7 Задача моделирования обледенения поверхности . . . . . 40

1.8 Описание границы ледяного нароста в задаче моделирования обледенения . . . . . 50

1.9 Выводы . . . . . 55

**Глава 2. Перестроение поверхностной сетки** **57**

2.1 Основные используемые геометрические примитивы . . . . . 59

2.1.1 Линейные примитивы . . . . . 59

2.1.2 Окрестности геометрических объектов . . . . . 60

2.1.3 Поверхностная неструктурированная сетка . . . . . 60

2.1.4 Заметаемые площади и объемы при движении ячеек . . . 62

2.2 Постановка задачи перестроения сетки . . . . . 66

2.3 Смещение узлов в двумерной постановке . . . . . 67

2.4 Приближенные методы перестроения . . . . . 69

2.4.1 Методы перестроения в двумерном случае . . . . . 69

2.4.1.1 Метод прямоугольников . . . . . 69

2.4.1.2 Метод трапеций . . . . . 70

2.4.1.3 Метод окрестностей в двумерном случае . . . . 71

2.4.2 Методы перестроения в трехмерном случае . . . . . 71

2.4.2.1 Метод призм . . . . . 72
2.4.2.2 Метод пирамид . . . . . 73
2.4.3 Дополнительные аспекты перестроения сетки . . . . . 74
2.4.3.1 Многослойное перестроение . . . . . 74
2.4.3.2 Перестроение со сглаживаниями . . . . . 75
2.5 Метод окрестностей перестроения поверхностной сетки в трехмерном случае . . . . . 79
2.5.1 Постановка задачи для отдельной ячейки . . . . . 79
2.5.2 Пересечения прямой с окрестностью отрезка . . . . . 82
2.5.3 Поиск нового положения узла сетки . . . . . 85
2.5.4 Алгоритм перестроения поверхностной сетки по методу окрестностей в трехмерной постановке . . . . . 88
2.5.5 Анализ метода окрестностей . . . . . 89
2.6 Оценки точности в двумерном случае . . . . . 92
2.6.1 Оценки точности для выпуклой сетки . . . . . 92
2.6.2 Оценки точности для вогнутой сетки . . . . . 96
2.6.3 Анализ оценок точности . . . . . 98
2.7 Оценки сглаживания пиков и впадин . . . . . 100
2.7.1 Получение оценок сглаживания . . . . . 101
2.7.2 Анализ оценок сглаживания пиков . . . . . 102
2.7.3 Анализ оценок сглаживания впадин . . . . . 104
2.8 Удаление самопересечений поверхностной сетки . . . . . 105
2.8.1 Адаптация сетки . . . . . 108
2.8.2 Поиск пересекающихся ячеек . . . . . 112
2.8.3 Обход внешней поверхности . . . . . 118
2.8.4 Алгоритмы метода удаления самопересечений поверхностной неструктурированной сетки, основанного на обходе ее внешней поверхности . . . . . 123
2.9 Сопряжение с объемными решателями . . . . . 127
2.9.1 Пересечение поверхностной сетки с декартовой . . . . . 128
2.9.2 Метод погруженной границы . . . . . 132

2.10 Выводы . . . . . 141

**Глава 3. Вычисления на блочно-структурированной сетке 143**

3.1 Показатели распараллеливания вычислений на распределенной памяти . . . . . 143

3.2 Архитектура блочно-структурированной сетки с поддержкой функции дробления блоков . . . . . 146

3.3 Распределение вычислительной нагрузки при проведении параллельных вычислений на блочно-структурированной сетке 154

3.3.1 Распределение блоков без дробления . . . . . 156

3.3.2 Механизм дробления блоков . . . . . 161

3.3.3 Распределение блоков с дроблением . . . . . 163

3.3.4 Метод распределения блоков сетки по вычислительным процессам с использованием дробления блоков, основанный на алгоритме распределения множества блоков по партициям с помощью дробления наибольших блоков . . 169

3.3.5 Метод распределения блоков сетки по вычислительным процессам с использованием дробления блоков, основанный на параметрическом алгоритме распределения блоков по партициям с уменьшением количества дроблений . . . . 176

3.4 Выводы . . . . . 182

**Глава 4. Вычисления на поверхностной сетке 183**

4.1 Декомпозиция поверхностной неструктурированной сетки . . . 183

4.1.1 Обзор методов декомпозиции . . . . . 183

4.1.2 Метод сглаживания границ между доменами поверхностной неструктурированной сетки . . . . . 191

4.1.3 Масштабирование вычислений . . . . . 205

4.2 Распараллеливание на общей памяти . . . . . 210

4.2.1 Распараллеливание для задачи обледенения . . . . . 210

4.2.2 Разделение ребер поверхностной неструктурированной сетки на неконфликтующие подмножества, основанное на реберной раскраске дуального графа . . . . . 212
4.3 Выводы . . . . . 218

**Глава 5. Векторизация вычислений** **219**
5.1 Особенности векторных инструкций . . . . . 220
5.2 Выделение однотипных операций для векторизации . . . . . 223
5.3 Плоский цикл . . . . . 232
5.3.1 Понятие плоского цикла . . . . . 232
5.3.2 Граф потока управления тела плоского цикла . . . . . 237
5.3.3 Семантика векторных инструкций . . . . . 238
5.3.4 Представление расчетов в виде плоских циклов . . . . . 241
5.4 Вынос маловероятных регионов . . . . . 247
5.5 Векторизация с помощью слияния путей исполнения . . . . . 250
5.6 Объединение и комбинирование векторных масок . . . . . 253
5.7 Векторизация гнезд циклов . . . . . 259
5.7.1 Внутренний цикл с постоянным количеством итераций . 262
5.7.2 Внутренний цикл с непостоянным количеством итераций 266
5.7.3 Внутренний цикл с нерегулярным количеством итераций 271
5.8 Методика векторизации вычислений . . . . . 276
5.9 Эффективность векторизации программного контекста . . . . . 279
5.10 Выводы . . . . . 281

**Заключение** **282**

**Список сокращений** **285**

**Список литературы** **288**

## Введение

Высокопроизводительные вычисления являются инструментом, широко применяемым в научных исследованиях и промышленных разработках. Развитие высокопроизводительных вычислений необходимо для обеспечения технологического лидерства Российской Федерации, вопросы создания и эффективного использования высокопроизводительных вычислительных систем актуальны [1–4] и соответствуют национальным целям и приоритетам научно-технологического развития Российской Федерации.

Высокопроизводительные вычисления являются основой для развития передовых технологий проектирования и создания высокотехнологичной продукции, они применяются в авиационной и космической промышленности, автомобилестроении, проектировании морского и железнодорожного транспорта, разработке образцов вооружения, разработке новых материалов [5–8]. В энергетической сфере высокопроизводительные вычисления применяются для моделирования объектов генерации электроэнергии и процессов внутри ядерных реакторов, а детальное моделирование месторождений углеводородного сырья позволяет повысить эффективность добычи и транспортировки [9–12]. Использование больших вычислительных систем позволяет извлекать качественно новые данные из точного моделирования крупных органических молекул, обработка больших массивов медицинских данных в совокупности с геномными исследованиями и индивидуальной настройкой параметров моделей на основе стандартных клинических данных знаменуют переход к персонализированной медицине [13–18]. Компьютерное моделирование активно используется в растениеводстве и животноводстве для повышения эффективности выработки продовольствия [19–22]. Высокопроизводительные вычисления применяются для проектирования и развития систем железнодорожного и авиасообщения, обеспечения логистики морских перевозок, создания глобальных компьютерных сетей [23–25]. В целях природосбережения высокопроизводительные вычисления применяются для решения задач экологии – исследование климата, мирового океана, экосистем, выбросов в окружающую среду [26–29].

Таким образом, сферы применения высокопроизводительных вычислений соответствуют приоритетам научно-технологического развития Российской Федерации и в частности переходу к передовым технологиям проектирования и создания высокотехнологичной продукции на основе интеллектуальных производственных решений, роботизированных и высокопроизводительных вычислительных систем, новых материалов и химических соединений, результатов обработки больших объемов данных, технологий машинного обучения и искусственного интеллекта. Вопросы создания и эффективного использования высокопроизводительных вычислительных систем вынесены на государственный уровень.

Компьютерное моделирование физических процессов в трехмерном пространстве относится к наиболее ресурсоемким научно-техническим задачам, связанным с высокопроизводительными вычислениями. В числе таких задач можно назвать моделирование процессов газовой динамики [30–32], электромагнетизма [33, 34], термодинамики [35, 36] и многих других. При этом в настоящее время актуальностью обладают вопросы разработки методов и средств мультифизического моделирования [37–39], включающего в себя одновременное моделирование целой совокупности сопряженных физических процессов, а также вопросы создания вычислительных сред и программного обеспечения для поддержки крупномасштабных научных экспериментов с использованием высокопроизводительных вычислений [40–42].

В современных расчетных задачах особый интерес представляет моделирование объектов и областей со сложной геометрией, так как это позволяет повысить точность описания свойств исследуемых объектов и систем. Объекты и области со сложной геометрией встречаются во многих научно-практических задачах. В качестве примеров таких задач можно привести численное моделирование обтекания тел со сложной поверхностью, многофазных течений с несмешивающимися фазами, течений в пористых средах, потоков с наличием взвеси твердых или жидких частиц, движения воды над неровным дном, электростатического поля и переноса излучения в сложных объектах и системах, процессов в толще грунта или льда со сложными внешними гра-

ницами и многие другие [43–52]. Эти задачи характеризуются тем, что в расчетной области присутствуют различные подобласти, разделенные границами (поверхностями), имеющими сложную форму. Эти границы могут быть поверхностью твердого тела, как в задаче обтекания, или разделять разные фазы многофазного течения, они могут перемещаться в расчетной области, как в задаче моделирования потока с наличием взвеси твердых частиц, или видоизменяться и перестраиваться (эволюционировать), как в задаче обледенения. При этом вычисления проводятся, как правило, с использованием расчетных сеток, которые могут описывать как область трехмерного пространства (объемные расчетные сетки [53, 54]), так и некоторую расположенную в трехмерном пространстве поверхность (поверхностные расчетные сетки [55, 56]). Особую сложность при проведении компьютерного моделирования представляет организация вычислений при работе с сетками, геометрия которых может изменяться в процессе проведения расчетов [57–59].

Для численного решения задач, в которых приходится иметь дело с областями со сложной геометрией, применяются различные подходы. Использование неструктурированных расчетных сеток позволяет описать расчетную область произвольной сложности, такие сетки более просты с точки зрения построения, однако они менее предпочтительны по сравнению со структурированными сетками с точки зрения построения расчетных схем повышенной точности и производительности вычислений [60–65]. Для решения широкого класса научно-практических задач применяются бессеточные методы, в которых вместо расчетных сеток используются наборы произвольно распределенных узлов или частиц [66–69]. При решении задач со сложной геометрией могут использоваться структурированные криволинейные сетки, согласующиеся с границей расчетной области, использование таких сеток требует больших вычислительных затрат для построения и поддержания ее в согласованном со сложной границей состоянии [70–74]. С точки зрения эффективности вычислений наиболее предпочтительно использование простых декартовых сеток, направленных вдоль осей координат. В этом случае необходимо дополнительно выполнять аппроксимацию краевых условий на криволинейных границах.

Такие методы обладают универсальностью, могут быть использованы для решения широкого класса задач и предпочтительны с точки зрения повышения производительности вычислений. Среди методов, позволяющих выполнить аппроксимацию граничных условий и обеспечить применение декартовых сеток в сочетании с криволинейными границами, можно назвать метод ступенчатого представления границы, метод скошенных ячеек, метод свободной границы, метод объема жидкости для аппроксимации свободной поверхности, метод погруженной границы и другие [75–87].

С помощью перечисленных выше методов возможно организовать вычисления внутри областей со сложной геометрией, используя при этом блочно-структурированные сетки, что открывает широкие возможности для повышения производительности вычислений. Организация данных внутри блочно-структурированных сеток в виде наборов многомерных массивов и инвариантность вычислительных шаблонов для ячеек таких сеток позволяют организовывать вычисления с высокой степенью параллелизма на всех уровнях: на уровне множества вычислительных узлов суперкомпьютера – на распределенной памяти, на уровне множества параллельных потоков внутри вычислительного узла – на общей памяти, на уровне отдельного вычислительного ядра или исполнительного устройства – векторизация вычислений [88–93].

Сложная граница расчетной области может быть описана через адаптацию объемной расчетной сетки, как это реализовано в пакете FlowVision [94, 95], однако более распространенным решением является ее представление в виде поверхностной сетки (ЛОГОС, ANSYS, Star-CCM+, Comsol Multiphysics) [96–100]. Для получения качественных результатов моделирования на сложных границах расчетных областей необходимо выполнять вычисления на поверхностной сетке. К таким задачам, в частности, относятся расчет высоты ледяного покрова обтекаемой воздушным потоком поверхности, расчет течения тонкого слоя жидкости по поверхности и другие задачи [101–103]. При выполнении вычислений на поверхностной неструктурированной сетке, так же как и в случае блочно-структурированных сеток, для повышения производительности расчетов необходимо использовать все доступные уровни

распараллеливания [104–109]. Для многих расчетных приложений характерно изменение геометрии области расчета, ограниченной поверхностной сеткой. При этом происходит эволюция ограничивающей поверхности – сдвиг узлов сетки в соответствии с логикой приложения [110–114]. В результате изменения геометрии поверхностной сетки возникает необходимость выполнения ее коррекции ввиду понижения качества отдельных элементов (увеличение площади ячеек, возникновение слишком малых или больших углов, коротких ребер, узлов неправильного положения) [115–117], а также других дефектов, таких как самопересечения [118–124].

Производительность расчетов при моделировании объектов со сложной геометрией тесно связана с надежностью программы перестроения сетки, то есть ее способностью выполнять свои функции в соответствии с заданными требованиями[1]. Повышение надежности программы направлено на предотвращение перехода сетки в некорректное состояние, связанное с возникновением и накоплением дефектов. При проведении имитационного моделирования (ИМ) используются три различных понятия, касающиеся времени: физическое время относится к моделируемой системе, модельное время – воспроизведение физического времени в модели, процессорное время – время выполнения модели на компьютере [125]. Если для изучения объекта требуется выполнить его ИМ на отрезке физического времени $t_{start} \leq t \leq t_{finish}$, то возникновение некорретного состояния сетки в момент $t_{err} \leq t_{finish}$ ставит под сомнение результаты моделирования на всем отрезке $t_{err} \leq t \leq t_{finish}$ либо вовсе способно привести к аварийному завершению программы, что уменьшает моделируемое физическое время объекта. Возникновение некорректного состояния сетки приводит к необходимости выполнения повторных запусков, что негативно влияет на производительность вычислений при моделировании объектов со сложной геометрией в условиях комплексного мультифизического моделирования. Функции программы, направленные на предотвращение возникновения дефектов сетки в процессе ее эволюции либо их устранение, повышают надеж-

[1]ГОСТР МЭК 62628-2021 Надежность в технике. Руководство по обеспечению надежности программного обеспечения.

ность программы перестроения и производительность вычислений, увеличивая моделируемое физическое время при моделировании объектов со сложной геометрией (в идеале исключая возникновение некорректного состояния сетки).

Развитие вычислительной техники, перспективных параллельных архитектур и средств программирования открывают широкие возможности для разработки приложений компьютерного моделирования и повышения производительности расчетов. Специфика новых появляющихся программно-аппаратных средств высокопроизводительных вычислений требует развития математического аппарата для их эффективного использования.

Таким образом, для моделирования объектов со сложной геометрией актуальны исследования, направленные на создание методов и средств организации вычислений на блочно-структурированных и неструктурированных расчетных сетках, трансформации поверхностных сеток, описывающих сложную границу расчетной области, и повышения надежности и производительности вычислений. В работе рассматриваются математические методы, архитектурные решения и программные средства организации высокопроизводительных вычислений на объемных блочно-структурированных и поверхностных неструктурированных сетках, перестроения и коррекции поверхностных неструктурированных сеток, повышения производительности вычислений на всех уровнях распараллеливания: на распределенной памяти, на общей памяти и на уровне отдельных инструкций путем векторизации. Предложенные методы и средства повышения производительности программ направлены на использование при решении широкого спектра задач компьютерного моделирования. Постоянно возрастающие требования, предъявляемые к точности и комплексности расчетов при моделировании объектов со сложной геометрией, а также развитие программно-аппаратных технических средств, используемых в расчетах, определяют актуальность исследований в этом направлении.

Настоящее исследование опирается на работы известных отечественных и зарубежных ученых, оказавших существенное влияние на развитие суперкомпьютерных технологий, их применение, включая создание методов, алгоритмов и программного обеспечения для организации и повышения производитель-

ности вычислений, в частности на труды А. Н. Тихонова, А. А. Самарского, Г. И. Марчука, С. К. Годунова, Б. Н. Четверушкина, Н. С. Бахвалова в области вычислительных методов и алгоритмов, Г. И. Савина, Е. Е. Тыртышникова, Ю. В. Василевского, W. Habashi, Ch. S. Peskin в области методов моделирования сложных процессов и систем, В. П. Иванникова, А. И. Аветисяна, В. В. Воеводина, Вл. В. Воеводина, М. В. Якобовского, И. А. Соколова, Б. М. Шабанова, А. Н. Томилина в области создания и использования высокопроизводительных вычислительных систем и организации вычислений, В. А. Мельникова, В. К. Левина, Б. А. Бабаяна, А. К. Кима, В. Ю. Волконского, J. Jeffers в области разработки и применения новых перспективных архитектур для высокопроизводительных вычислений, В. Ф. Тишкина, В. А. Гаранжи, В. Д. Лисейкина, Д. Л. Ревизникова, X. Jiao в области вычислительной геометрии и управления расчетными сетками.

**Цели и задачи работы**

Целью работы является разработка комплекса математических методов и алгоритмов для повышения производительности вычислений при моделировании объектов со сложной геометрией на современных параллельных вычислительных системах.

Для достижения поставленной цели в диссертационной работе необходимо решить следующие **задачи**:

- Разработать архитектуру и состав программного комплекса, реализующего математические методы и алгоритмы, применяемые для моделирования объектов со сложной геометрией с использованием высокопроизводительных вычислительных систем.
- Разработать методы перестроения и коррекции поверхностной неструктурированной сетки, описывающей границы расчетных подобластей, для увеличения моделируемого физического времени при моделировании объектов со сложной геометрией.
- Разработать решения по повышению производительности параллельных вычислений на объемных блочно-структурированных и поверхностных

неструктурированных сетках, проводимых при расчетах в подобластях со сложной геометрией и на их границах.

- Разработать методы векторизации программного кода для повышения производительности вычислений на современных параллельных вычислительных системах, применимые к широкому классу приложений.

**Методология и методы исследования**

Математическую основу исследования составляют методы организации и повышения эффективности высокопроизводительных вычислений, методы вычислительной геометрии, теории графов, теории алгоритмов, теории комбинаторной оптимизации, теории эквивалентных преобразований программ. Для анализа и обоснования эффективности алгоритмов использовались модельные объекты, метод статистических испытаний и вычислительные эксперименты на суперкомпьютерных системах различных архитектур.

**Научная новизна**

- Разработан и реализован в программных кодах комплекс математических методов и алгоритмов для повышения производительности вычислений при моделировании объектов со сложной геометрией, ориентированный на применение на современных параллельных вычислительных системах и объединяющий в себе решения по перестроению и коррекции поверхностной неструктурированной сетки, описывающей границы расчетных подобластей, а также по распараллеливанию вычислений, выполняемых на объемных и поверхностных сетках.
- Разработан метод окрестностей перестроения поверхностной неструктурированной сетки, основанный на поиске пересечения траектории смещения узла с границей окрестности инцидентных узлу ячеек и обладающий способностью сглаживания локальных дефектов для предотвращения их накопления.
- Разработан метод удаления самопересечений поверхностной неструктурированной сетки с треугольными ячейками, основанный на обходе ее

внешней поверхности, которая определяется направлениями внешних нормалей ячеек.

- Разработан трехмерный метод погруженной границы с фиктивными ячейками, являющийся обобщением двумерного случая и основанный на аппроксимации физических величин в фиктивной ячейке по трем точкам пространства и направлению нормали в точке на границе.
- Разработаны архитектура объемной блочно-структурированной сетки и методы распределения блоков сетки по вычислительным процессам с использованием дробления блоков для распараллеливания вычислений на распределенной памяти.
- Разработан метод сглаживания границ между доменами декомпозированной поверхностной неструктурированной сетки, позволяющий из множества локальных шаблонов сглаживания границы между парой доменов выбрать подмножество шаблонов, при одновременном применении которых обеспечивается максимальное сокращение длины границы при заданном изменении баланса ячеек между этими доменами.
- Введено понятие плоского цикла с рядом ограничений на его структуру как удобного программного контекста для векторизации вычислений, разработаны методы векторизации плоских циклов с телом произвольного вида.
- Разработана методика повышения производительности программ путем представления вычислений в виде плоских циклов, которая обеспечивает возможность объединения нескольких экземпляров подпрограммы для решения на одном процессорном ядре.

**Положения, выносимые на защиту**

- Комплекс математических методов и алгоритмов позволяет повысить производительность решения широкого класса ресурсоемких вычислительных задач, применяемых для моделирования объектов со сложной геометрией, с учетом особенностей архитектуры параллельной вычислительной системы.

- Метод окрестностей перестроения поверхностной неструктурированной сетки позволяет повысить надежность перестроения для увеличения моделируемого физического времени при моделировании объектов со сложной геометрией.
- Метод удаления самопересечений поверхностной неструктурированной сетки путем обхода ее внешней поверхности обеспечивает корректное состояние сетки, описывающей границу расчетной подобласти.
- Метод погруженной границы с фиктивными ячейками в трехмерной постановке позволяет проводить вычисления в расчетной подобласти со сложной границей без необходимости построения объемной сетки, согласованной с поверхностной сеткой, описывающей границу подобласти.
- Методы распределения блоков сетки по вычислительным процессам позволяют преодолеть проблему низкой производительности вычислений из-за наличия крупных блоков и снизить показатель неравномерности распределения вычислительной нагрузки между процессами.
- Метод сглаживания границ между доменами поверхностной неструктурированной сетки позволяет уменьшить длину границ и повысить тем самым производительность вычислений за счет сокращения объема межпроцессных обменов.
- Методы векторизации плоских циклов, направленные на сокращение количества операций передачи управления и повышение плотности векторных масок внутри тела плоского цикла, позволяют ускорить выполнение плоских циклов.
- Методика векторизации плоских циклов с телом произвольного вида позволяет повысить производительность вычислений на широком классе приложений.

**Соответствие паспорту специальности**

Диссертация соответствует следующим направлениям исследований паспорта специальности 2.3.5. «Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей»:

- 1. Модели, методы и алгоритмы проектирования, анализа, трансформации, верификации и тестирования программ и программных систем.
- 3. Модели, методы, архитектуры, алгоритмы, языки и программные инструменты организации взаимодействия программ и программных систем.
- 8. Модели и методы создания программ и программных систем для параллельной и распределенной обработки данных, языки и инструментальные средства параллельного программирования.

**Теоретическая и практическая значимость**

Метод окрестностей перестроения поверхностной неструктурированной сетки и метод удаления самопересечений вносят теоретический вклад в вычислительную геометрию и могут быть использованы для развития механизмов эволюции сетки. Алгоритмы распределения блоков сетки с использованием дробления блоков вносят теоретический вклад в решение задачи выбора наилучшего разбиения. Алгоритм сглаживания границ между доменами поверхностной неструктурированной сетки вносит теоретический вклад в решение задачи декомпозиции и может быть использован для обобщения на объемный случай. Методы векторизации программного кода вносят теоретический вклад в теорию эквивалентных преобразований программ и могут быть использованы в инструментах компиляции программ.

Комплекс решений для повышения производительности программ, включающий в себя разработанные математические методы и алгоритмы, используется при проведении расчетов по моделированию обледенения поверхностей и моделированию газодинамических течений в условиях сложных конфигураций в интересах создания новых образцов авиационной техники: моделирование обледенения фюзеляжа, несущих аэродинамических поверхностей, элементов воздухозаборных устройств летательных аппаратов, несущего и рулевого винта вертолета, моделирование газодинамических процессов в элементах силовых установок летательных аппаратов. Разработанные методы и алгоритмы реализованы в программных системах компьютерного моделирования

физических процессов и обработки расчетных сеток. На основе результатов исследования созданы программные средства, которые используются для проведения научно-исследовательских и опытно-конструкторских работ в ряде организаций: Национальный исследовательский центр «Курчатовский институт», Федеральное автономное учреждение «Центральный институт авиационного моторостроения имени П. И. Баранова», Акционерное общество «Национальный центр вертолетостроения имени М. Л. Миля и Н. И. Камова». На разработанные программные средства получено 8 свидетельств о государственной регистрации программ для ЭВМ.

**Достоверность полученных результатов**

Полученные результаты реализованы в программных комплексах, используемых в научных исследованиях и промышленных расчетах. Достоверность результатов диссертации обеспечивается строгим применением используемого математического аппарата, результатами экспериментов на суперкомпьютерах различной архитектуры Межведомственного суперкомпьютерного центра Российской академии наук и Национального исследовательского центра «Курчатовский институт», практикой применения программного комплекса компьютерного моделирования процесса обледенения элементов авиационных силовых установок «Кристалл» и согласованностью результатов диссертации с известными из научной литературы.

**Апробация диссертации**

Основные результаты диссертации представлены на конференциях:

- Международная научно-техническая конференция по авиационным двигателям (ICAM-2025), Россия, Москва, 1-3 декабря 2025 г.
- V Национальный Суперкомпьютерный Форум (НСКФ-2016), Россия, Переславль-Залесский, Институт программных систем им. А. К. Айламазяна РАН, 29 ноября – 2 декабря 2016 г.
- VII Национальный Суперкомпьютерный Форум (НСКФ-2018), Россия, Переславль-Залесский, Институт программных систем им. А. К. Айла-

мазяна РАН, 27-30 ноября 2018 г.

- X Национальный Суперкомпьютерный Форум (НСКФ-2021), Россия, Переславль-Залесский, Институт программных систем им. А. К. Айламазяна РАН, 30 ноября – 3 декабря 2021 г.
- I Международная научная конференция «Конвергентные когнитивно-информационные технологии», Россия, Москва, Московский государственный университет им. М. В. Ломоносова, 25-26 ноября 2016 г.
- III Международная научная конференция «Конвергентные когнитивно-информационные технологии», Россия, Москва, Московский государственный университет им. М. В. Ломоносова, 22-25 ноября 2018 г.
- XVI Международная конференция «Супервычисления и математическое моделирование», Россия, Саров, Российский федеральный ядерный центр – Всероссийский научно-исследовательский институт экспериментальной физики, 3-7 октября 2016 г.
- VI всероссийская конференция «Вычислительный эксперимент в аэроакустике», Россия, Светлогорск, организатор – Институт прикладной математики им. М. В. Келдыша РАН, 19-24 сентября 2016 г.
- IX профессиональный слет разработчиков отечественных CFD кодов (CFD Weekend 2022), Россия, Москва, Институт прикладной математики им. М. В. Келдыша РАН, 3-4 декабря 2022 г.
- X профессиональный слет разработчиков отечественных CFD кодов (CFD Weekend 2023), Россия, Москва, Институт прикладной математики им. М. В. Келдыша РАН, 9-10 декабря 2023 г.
- XII профессиональный слет разработчиков отечественных CFD кодов (CFD Weekend 2025), Россия, Москва, Институт прикладной математики им. М. В. Келдыша РАН, 29-30 ноября 2025 г.
- The Intel Xeon Phi users group Russia annual meeting (IXPUG Russia-2017), Joint Supercomputer Center of the Russian Academy of Sciences (JSCC RAS), Russia, Moscow, 1-2 June 2017.

**Публикации**

Основные результаты диссертации изложены в 42 печатных работах, из них 30 статей опубликованы в изданиях, рекомендованных ВАК, в том числе 20 работ — в журналах, индексируемых базой данных RSCI, 5 работ — в изданиях, отнесенных к категории К-1 Перечня ВАК, 10 работ — в журналах, индексируемых международными базами данных Web of Science и Scopus. По теме диссертации получено 8 свидетельств о государственной регистрации программ для ЭВМ [126–133].

**Структура и объем работы**

Диссертация состоит из введения, пяти глав, заключения, списка сокращений и списка литературы. Общий объем диссертации – 337 страниц, в том числе 126 рисунков, 9 таблиц и 17 листингов. Список литературы состоит из 383 наименований.

Первая глава является обзорной и посвящена проблемам проведения высокопроизводительных расчетов при моделировании объектов со сложной геометрией. В ней рассмотрены методы и инструменты для работы с поверхностными неструктурированными сетками, методы распараллеливания вычислений на объемных и поверхностных сетках на всех уровнях распараллеливания: на распределенной памяти, на общей памяти, на уровне отдельных инструкций. Рассмотрена задача моделирования обледенения поверхности в условиях обтекания внешним потоком, которая является критической для проектировании новых образцов авиационной техники, и в применении к которой были разработаны, реализованы и внедрены предложенные в работе методы и алгоритмы повышения производительности программ для моделирования объектов со сложной геометрией, в частности в рамках программного комплекса расчета процесса обледенения элементов авиационных силовых установок «Кристалл» [126, 127], а также других программных средств [128–133].

Вторая глава посвящена задаче разработки метода перестроения поверхностной неструктурированной сетки, способного сглаживать локальные дефекты, и метода удаления самопересечений такой сетки. Исследуется перестроение

поверхностной сетки в двумерном и трехмерном случаях, рассматриваются известные методы и предлагается новый метод перестроения, выводятся аналитические оценки точности и эффективности сглаживания дефектов. Исследуется корректировка поверхностной сетки после перестроения, рассматривается проблема удаления самопересечений, анализируются существующие методы, предлагается новый метод удаления самопересечений. Исследуется проблема сопряжения с объемной сеткой, предлагается трехмерный метод погруженной границы, используемый для проведения расчетов на объемной сетке без необходимости построения согласованной сетки в пространстве.

Третья глава посвящена задаче разработки методов повышения производительности параллельных вычислений на объемной блочно-структурированной сетке. Рассматривается проблема распределения вычислительной нагрузки между узлами вычислительной системы и предлагаются методы распределения вычислительной нагрузки, использующие дробление блоков сетки.

Четвертая глава посвящена задаче разработки методов повышения производительности параллельных вычислений на поверхностной неструктурированной сетке. Предлагается метод сглаживания границ между доменами сетки, направленный на уменьшение объема межпроцессных обменов и применимый совместно с произвольными методами декомпозиции сетки. Рассматривается проблема разрешения конфликтов при распараллеливании вычислений на поверхностной неструктурированной сетке на общей памяти.

Пятая глава посвящена задаче разработки методов векторизации программного кода и методики применения векторизации. Вводится понятие плоского цикла, обладающего специальными органичениями, которые позволяют применить ряд преобразований, направленных на повышение эффективности векторизации таких циклов. Предлагается ряд методов, позволяющих векторизовать плоский цикл с телом произвольного вида. Предлагается методика, объединяющая в себе рекомендации по организации кода в виде набора плоских циклов, и преобразования, направленные на эффективное применение векторизации.

В заключении формулируются основные результаты диссертации.

# Глава 1. Вычисления в условиях сложной геометрии

Области со сложной геометрией встречаются во многих научно-практических задачах. В качестве примеров таких задач можно привести численное моделирование обтекания тел со сложной поверхностью [43], многофазных течений с несмешивающимися фазами [44, 45], течений в пористых средах [46], потоков с наличием взвеси твердых или жидких частиц [47], движения воды над неровным дном [48], электростатического поля [49] и переноса излучения в сложных объектах и системах [50], процессов в толще грунта [51] или льда [52] со сложными внешними границами. Эти задачи характеризуются тем, что в расчетной области присутствуют различные подобласти, разделенные границами (поверхностями) сложной формы. Границы могут быть поверхностью твердого тела, как в задаче обтекания, или разделять фазы многофазного течения, они могут перемещаться в расчетной области, как в задаче моделирования потока с наличием взвеси твердых частиц, или видоизменяться и перестраиваться (эволюционировать), как в задаче обледенения.

Проблемы создания и обработки сложных поверхностных сеток также имеют место в других задачах, напрямую не связанных с имитационным моделированием физических процессов, например в задачах компьютерной графики [134, 135] или построения рельефа по данным сканирования [136].

## 1.1 Расчетные сетки в условиях сложной геометрии

Для численного решения задач моделирования объектов со сложной геометрией применяются подходы с использованием различных расчетных сеток.

Построение неструктурированных сеток в областях со сложной формой – это активно развивающийся раздел вычислительной математики и численной геометрии [137, 138]. Использование неструктурированных сеток позволяет описать расчетную область произвольной сложности, они могут строиться в условиях адаптации к искомому решению [139], однако их использование связано с определенными проблемами. По сравнению со структурированными сетками на неструктурированных сетках сложнее выполнять аппроксима-

цию уравнений в частных производных, а шаблоны разностной схемы для разных точек или ячеек сетки могут различаться [60]. Использование неструктурированных сеток существенно усложняет построение схем повышенной точности [61]. При решении задач газовой динамики дискретизация уравнений на неструктурированных сетках приводит к разреженной матрице коэффициентов, которая является несимметричной и не имеет диагонального преобладания, что приводит к замедлению скорости сходимости [62]. Неструктурированные сетки более требовательны к вычислительным ресурсам, так как для их обработки требуется хранить информацию об элементах и связях между ними. Навигация по неструктурированной сетке связана с косвенностью обращения в память, так как количество смежных и инцидентных объектов элементов неструктурированной сетки непостоянно, и связи, как правило, хранятся в списках. Также при использовании неструктурированных сеток возникают дополнительные сложности при попытке оптимизировать вычисления с помощью использования специализированных векторных или матричных операций микропроцессора или сопроцессора. Существует много современных подходов, позволяющих оптимизировать применение неструктурированных сеток в плане аппроксимации и решения уравнений в частных производных: многосеточные методы (MultiGrid method, MG-method) для решения задач математической физики на неструктурированных сетках [63], метод Галёркина с разрывными базисными функциями (Разрывный Метод Галёркина, РМГ) для решения гиперболических систем уравнений на неструктурированных сетках [64], конечно-объемные методы с использованием полиномиальной реконструкции переменных [65], – но сложность организации вычислений на неструктурированных сетках выше, чем на структурированных.

Для решения широкого класса задач применяются бессеточные методы, использующие вместо расчетных сеток наборы произвольно распределенных узлов или частиц. Так, метод сглаженных частиц (Smoothed Particle Hydrodynamics, SPH) [66] с использованием взаимодействующих и обладающих материальными свойствами частиц, широко распространен при решении задач гидродинамики [67], включая задачи со свободной поверхностью и

деформируемыми границами [68]. Он прост в реализации, автоматически обеспечивает сохранение массы и позволяет моделировать больших деформаций, но для получения адекватных результатов необходимо использование большого количества частиц. В методе дискретных вихрей (МДВ) движение жидкости рассматривается как движение множества взаимодействующих между собой и сохраняющих интенсивность вихрей [69].

При решении задач со сложной геометрией могут использоваться структурированные криволинейные сетки, согласующиеся с границей расчетной области [70]. Их использование требует больших вычислительных затрат для построения и поддержания в согласованном состоянии. Особенно это критично для задач, в которых границы изменяются [71]. Задача построения структурированной криволинейной сетки решается с помощью поиска взаимно-однозначного отображения вычислительного пространства на физическое (рис. 1.1). Существуют методы решения задачи отображения с помощью трансфинитной интерполяции [72], с помощью решения уравнения Пуассона [73], а в [74] предложено решение по генерации структурированных криволинейных сеток с использованием искусственной нейронной сети.

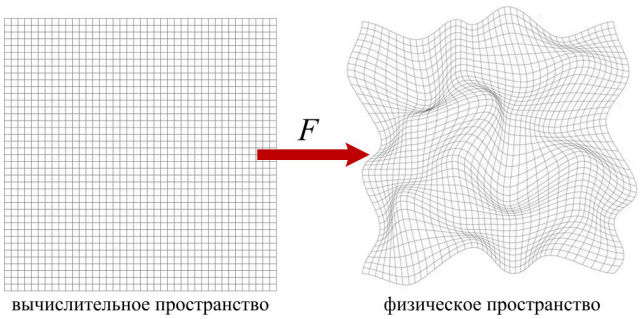


Рис. 1.1. Отображение вычислительного пространства на физическое пространство с задаче создания структурированной криволинейной сетки

С точки зрения эффективности вычислений наиболее предпочтительно использование декартовых сеток, направленных вдоль осей координат. В этом случае необходимо дополнительно выполнять аппроксимацию краевых условий на криволинейных границах. Такие методы обладают универсальностью и могут быть использованы для решения широкого класса задач [71]. Среди методов, позволяющих выполнить аппроксимацию граничных условий

и обеспечить применение декартовых сеток в сочетании с криволинейными границами, можно назвать метод ступенчатого представления границы [75], метод скошенных ячеек [76], метод свободной границы [77], метод объема жидкости (Volume Of Fluid, VOF) для аппроксимации свободной поверхности [78], метод погруженной границы (МПГ) [79]. В некоторых работах метод скошенных ячеек рассматривается как разновидность МПГ [80], а метод свободной границы имеет сходство с МПГ с использованием штрафных функций [81].

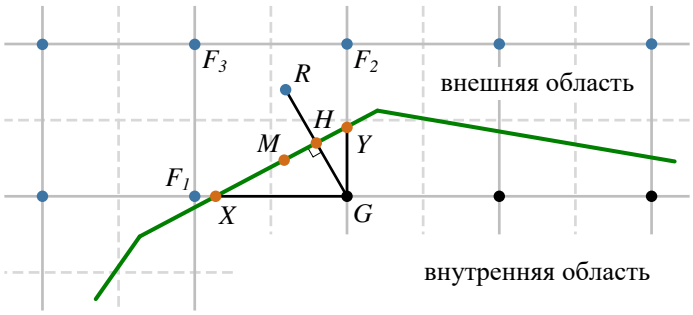


Рис. 1.2. Точки, используемые для вычисления физических величин в фиктивных ячейках в методе погруженной границы

МПГ изначально был разработанном для решения задач биологической гидродинамики [140]. В МПГ с использованием штрафных функций граница раздела двух сред моделируется путем добавления в исходное уравнение источниковых членов, учитывающих влияние границы [80]. В МПГ с использованием фиктивных ячеек, одной из областей применения которого является моделирование течений вблизи поверхности сложной формы, расчетная область разделена на внешнюю область (течение) и внутреннюю область (обтекаемое тело) (рис. 1.2). Наряду с обычными ячейками, находящимися во внешней области (внешние ячейки) для расчетов используются также ячейки из внутренней области, входящие в вычислительный шаблон хотя бы одной внешней ячейки (фиктивные ячейки). Во время расчетов параметры течения в фиктивных ячейках определяются через параметры в других точках расчетной области с помощью интерполяции с учетом условий на границе. На рис. 1.2 в двумерной иллюстрации показаны возможные точки, используемые при определении параметров для точки фиктивной ячейки $G$: точки внешних ячеек $F_i$, ближайшая точка границы $H$, середина отрезка $XY$ – точка $M$ (где

$X$, $Y$ – точки пересечения параллельных осям координат прямых $GX$, $GY$ с границей [82]), образ $R$ точки $G$ относительно границы [83], – могут быть использованы и другие точки. Известны различные подходы к интерполяции параметров в фиктивных точках, включая линейную интерполяцию [84], билинейную/трилинейную интерполяцию [85], квадратичную интерполяцию [86], разложение параметров в фиктивной точке в ряд Тейлора [87].

## 1.2 Распараллеливание на блочно-структурированных сетках

С помощью приведенных методов можно организовать вычисления внутри областей со сложной геометрией, используя блочно-структурированные сетки, что открывает широкие возможности для повышения производительности вычислений. Организация данных внутри блочно-структурированных сеток в виде наборов многомерных массивов и инвариантность вычислительных шаблонов для ячеек таких сеток позволяют организовывать вычисления с высокой степенью параллелизма на всех уровнях: на уровне множества вычислительных узлов внутри суперкомпьютера – на распределенной памяти, на уровне множества параллельных потоков в рамках одного вычислительного узла – на общей памяти, на уровне отдельных инструкций.

Использование структурированных данных позволяет упростить распараллеливание вычислений между процессами/потоками, что применяется во многих расчетных приложениях [88–90]. Наличие упорядоченных структур позволяет быстро и эффективно распределить вычисления по процессам/потокам с минимальным количеством накладных расходов, так как структурированная расчетная область допускает достаточно простую декомпозицию (рис. 1.3). Параллелизм на уровне отдельных инструкций выражен во VLIW (Very Long Instruction Word) архитектурах [91], а также в вычислениях, при организации которых одна инструкция применяется сразу к некоторому набору данных, что имеет место при использовании GPU [92] и векторных инструкций CPU [93].

Задача распределения блоков расчетной сетки на вычислительные узлы с целью балансировки вычислительной нагрузки связана с поиском опти-

мального отображения процессов параллельной программы, обрабатывающих отдельные блоки, на узлы суперкомпьютерной системы [141]. Задача сводится к задаче отображения информационного графа программы на граф вычислительной системы. Точное решение этой задачи связано с большими вычислительными затратами ввиду ее NP-полноты, поэтому для решения часто используются приближенные и эвристические подходы такие как поиск отображения с помощью алгоритма имитации отжига [142], генетических алгоритмов [143], искусственных нейронных сетей [144] и других подходов [145].

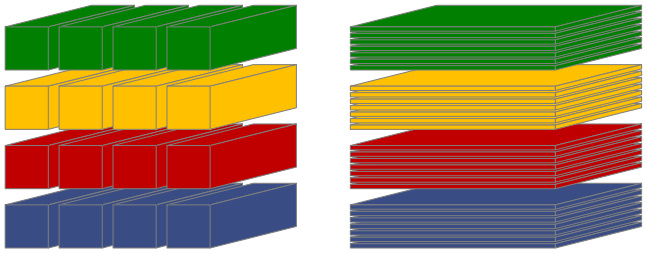

Рис. 1.3. Примеры декомпозиции структурированной расчетной области

В качестве дополнительного аспекта организации вычислений на блочно-структурированных сетках отметим адаптацию сетки (Adaptive Mesh Refinement, AMR). Блочно-структурированные сетки отличаются простотой реализации и скоростью обработки, функционал для работы с ними может быть расширен с помощью дробления ячеек на более мелкие, адаптируясь к сложной геометрии [146] или особенностям решения [147]. Локальная адаптация может применяться также на уровне блоков сетки, а не ячеек [148].

## 1.3 Распараллеливание на поверхностных сетках

При выполнении вычислений в расчетной области со сложной границей существуют разные подходы к описанию этой границы. В некоторых программных кодах описание сложной границы выполняется через адаптацию расчетной сетки, как это имеет место в программном комплексе FlowVision [94, 95]. Однако, в большинстве случаев в программных пакетах для суперкомпьютерного моделирование отдается предпочтение представлению сложной границы с

помощью поверхностной неструктурированной сетки [96]. Использование поверхностной неструктурированной сетки для представления сложной границы встречается в таких широко известных программным пакетах как ЛОГОС [97], ANSYS [98], Star-CCM+ [99], Comsol Multiphysics [100] и многих других. В общем случае для описания произвольной геометрии сложной поверхности необходимо использование расчетных сеток с треугольными ячейками, так как при наличии более трех узлов в ячейке невозможно гарантировать, что она останется плоской. Обычным методом построения таких сеток является метод подвижного фронта [149], в котором сначала строится неструктурированная сетка на плоскости, начиная с границ области и продвигаясь внутрь путем добавления новых треугольников, а затем построенная сетка переносится на трехмерную поверхность с помощью отображения. Различные усовершенствованные методы построения позволяют повысить детализацию сетки в зависимости от свойств расчетной области [150], а также ускорить построение для моделей высокой детализации и выполнить коррекцию результирующей сетки [151].

Для получения качественных результатов моделирования на сложных границах расчетных областей необходимо выполнять вычисления на поверхностной сетке. К таким задачам, в частности, относятся расчет высоты ледяного покрова обтекаемой воздушным потоком поверхности [101], расчет течения тонкого слоя жидкости по поверхности [102], расчет МГД-устойчивости двумерных плазменных конфигураций [103] и другие задачи. При выполнении вычислений на поверхностной неструктурированной сетке, так же как и в случае блочно-структурированных сеток, для повышения производительности расчетов необходимо использовать все доступные уровни распараллеливания.

При выполнении распараллеливания на распределенной памяти возникает проблема декомпозиции неструктурированной сетки и организации межпроцессных обменов. В результате декомпозиции граф разбивается на подграфы, соответствующие доменам сетки, обрабатываемым в отдельных вычислительных процессах. Декомпозиция неструктурированной сетки, или ее дуального графа, существенным образом отличается от аналогичной задачи,

решаемой для структурированной сетки. Решению задачи о декомпозиции неструктурированной сетки на близкие по размеру домены с минимизацией количества ребер, находящихся на границах между разными доменами, посвящено большое количество исследований [104]. К наиболее известным методам декомпозиции можно отнести рекурсивную геометрическую бисекцию, заполнение пространства с помощью кривой Гильберта, алгоритмы наращивания доменов, локально-корректирующие алгоритмы Кернигана-Лина и Фидуччи-Маттейсеса и их модификации, спектральную бисекцию, иерархические алгоритмы с использованием огрубления и уточнения сетки (рис. 1.4).

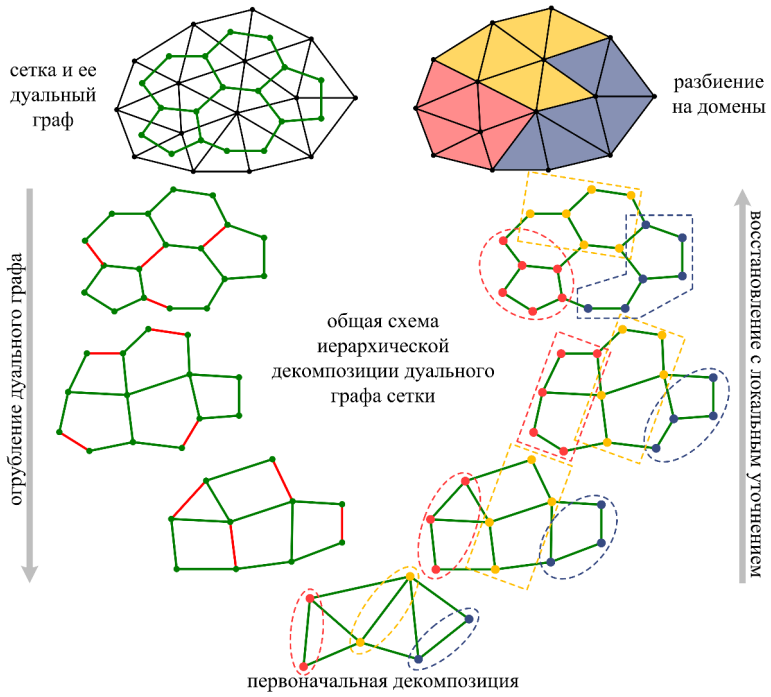


Рис. 1.4. Схема иерархической декомпозиции расчетной сетки

Геометрическая бисекция основана на разделении произвольного домена на две равные части, опираясь на геометрическое положение вершин графа – множество вершин может быт разбито как с помощью одной из плоскостей, параллельных координатным осям $OX$, $OY$, $OZ$, так с помощью плоскости общего положения, и вообще с помощью произвольного отношения порядка, введенного для вершин дуального графа сетки [105]. Декомпозиция на основе кривой Гильберта опирается на свойство локальности этой кривой – звенья кривой с близкими номерами имеют близкие координаты в пространстве, что

позволяет строить достаточно компактные домены [152]. Декомпозиция с помощью наращивания доменов представляет собой группу алгоритмов, в которых домены, инициируемые некоторыми начальными вершинами, последовательно расширяются путем добавления ближайших к ним соседей, и это расширение может выполняться последовательно, как в первых реализациях (на примере алгоритма Фархата [153]), так и одновременно для всех доменов [106]. Инкрементный метод декомпозиции основан на понятии ядра домена заданного уровня, определяемого через кратчайшее расстояние от вершины до множества граничных вершин домена, и ориентирован на формирование доменов с соблюдением связности ядер заданного уровня [154]. Алгоритм Кернигана-Лина является алгоритмом уточнения разбиения на два домена $A$ и $B$, он основан на итерационном обмене этих доменов вершинами между собой, то есть на каждой итерации алгоритма оценивается полезность обмена вершинами $a \in A$ и $b \in B$ на основании количества внутренних и внешних связей этих вершин в своих доменах [155]. Алгоритм Феддучи-Маттейсеса предлагает эвристику для нахождения перемещаемой между доменами точки с линейной сложностью, что позволяет достаточно быстро производить локальную коррекцию разбиения [156]. Метод спектральной бисекции основан на использовании спектральной матрицы графа (матрицы Лапласа), и направлен на минимизацию количества ребер, соединяющих вершины из разных доменов [157]. Иерархические алгоритмы основаны на трех основных действиях: огрубление графа – в процессе которого создается последовательность постепенно огрубленных графов меньшего размера, начальная декомпозиция – разбиение наиболее огрубленного графа на домены, восстановление графа – обратное движение по последовательности огрубленных графов с выполнением локального уточнения декомпозиции на каждом шаге [158]. В одном из первых пакетов декомпозиции графов Chaco поддержаны различные методы декомпозиции, включая спектральную, геометрическую и иерархическую [159]. Среди других классических пакетов декомпозиции можно отметить ParMETIS с реализацией иерархической декомпозиции [160], основанный на алгоритме Фархата и последующих локальных уточнениях декомпозиции пакет JOSTLE [161],

пакет с рекурсивной бисекционной декомпозицией SCOTCH [162] и многие другие [163]. В современном отечественном пакете GridSpiderPar реализован параллельный алгоритм инкрементной декомпозиции, превосходящий зарубежные аналоги по дисбалансу вершин в доменах, количеству несвязных доменов и количеству ребер, соединяющих вершины из разных доменов [164].

При выполнении вычислений с использованием многоядерных микропроцессоров возникают вопросы распараллеливания вычислений на общей памяти, когда несколько параллельных потоков одновременно имеют доступ в одной и той же области памяти. При организации вычислений неизбежно возникают конфликты по обращению к данным – когда два потока потенциально могут обраться на запись к одному и тому же элементу данных в памяти (классический пример операции `x += v`). При использовании стандарта распараллеливания программ OpenMP [165] такие конфликты могут быть разрешены с помощью использования атомарных операций или критических секций (директивы `atomic`, `critical`). Как правило, использование средств OpenMP приводит к снижению эффективности вычислений, поэтому возникает необходимость разработки алгоритмических решений к разрешению конфликтов. Так для конечно-элементных вычислений описаны подходы к разрешению конфликтов, основанные на изменении порядка суммирования данных при вычислениях [107], а также на разделении конечно-элементных сеток на отдельные слои [108]. При реализации конечно-объемных численных методов на неструктурированных сетках полностью устранить конфликты по данным при использовании общей памяти позволяет использование разбиение множества обрабатываемых элементов сетки на неконфликтующие подмножества, что выполняется с помощью раскрасок графа [109].

## 1.4 Перестроение поверхностной сетки

Для многих расчетных приложений характерно изменение геометрии области расчета, ограниченной поверхностной сеткой. При этом происходит эволюция ограничивающей поверхности – сдвиг узлов сетки в соответствии с

логикой приложения. В результате изменения геометрии поверхностной сетки возникает необходимость выполнения ее коррекции ввиду понижения качества отдельных элементов (увеличение площади ячеек, возникновение слишком малых или больших углов, коротких ребер, узлов неправильного положения), а также других дефектов, таких как самопересечения [115]. Коррекция сетки выполняется на основе определенных аспектов качества, таких как качество отдельных ячеек, сохранение топологии, регулярность узлов, избавление от артефактов и других. Качество треугольной ячейки $t$ может быть выражено в различных метриках, например $Q(t) = \frac{\sqrt{3}l^2}{4s}$ ($l$ – длина наибольшей стороны, $s$ – площадь) [166], $Q(t) = \frac{l_{max}}{l_{min}}$ ($l_{max}$ – длина наибольшей стороны, $l_{min}$ – длина наименьшей стороны) [167], $Q(t) = \max_{\alpha \in A(t)} |\alpha - \frac{\pi}{3}|$ ($A(t)$ – множество углов треугольника) [168]. Сохранение топологии подразумевает малое изменение сетки при выполнении ее корректности, величина изменения может выражаться в таких метриках, как Хаусдорфово расстояние [169], среднее расстояние или среднеквадратическое расстояние [116] между исходной и скорректированной сетками. Метрика регулярности узлов сетки выражается в доле всех узлов, для которых количество инцидентных ребер находится в заданном диапазоне. Для сеток с треугольными ячейками в качестве такого диапазона можно принять значение от 5 до 7. Среди основных подходов к коррекции сетки можно выделить упрощение сетки с требуемой степенью огрубления, локальную модификацию отдельных элементов сетки, коррекцию с помощью триангуляции Делоне (Delaunay Triangulation, DT), сглаживание сетки с использованием центроидов соседей узлов (Laplacian Smoothing, LS), а также с помощью центроидной диаграммы Вороного (Centroidal Voronoi Tesselation, CVT). К локальным модификациям сетки относятся такие операции, как стягивание сетки по ребру, изменение координат отдельного узла, разбиение ребра путем добавления новой вершины (рис. 1.5) и другие [117].

При выполнении перестроения поверхностной неструктурированной сетки в процессе моделирования не исключено возникновение самопересечений. Наличие самопересечений может привести к искажению результатов вычислений, поэтому для обеспечения продолжения корректных расчетов самопересече-

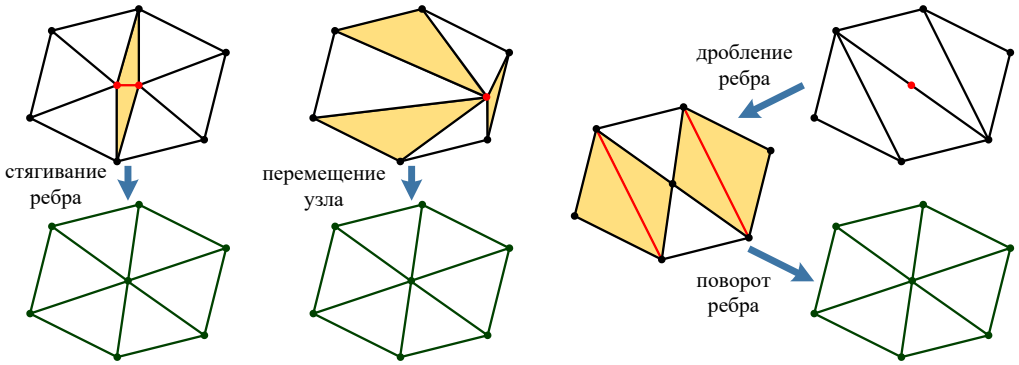


Рис. 1.5. Локальная коррекция сетки

ния должны быть удалены. В работах [118, 170, 171] представлено развитие подхода к удалению самопересечений, основанного на использовании построенной тетраэдральной объемной сетки в расчетной области, что связано с существенными вычислительными затратами. Из ранних работ по поиску и удалению самопересечений, работающих непосредственно на поверхностной сетке, можно отнести методы, базирующиеся на поиске самопересечений в виде замкнутых ломаных и обхода их с помощью алгоритма TNOIT (Tracing the Neighbours of Intersecting Triangles) [119, 172]. Узким местом этого алгоритма являются погрешности вычислений, выполняемых над числами с плавающей точкой, из-за чего может быть неверно найдено пересечение геометрических примитивов. Такие ошибки возникают, если сетка содержит вырожденные ячейки, ячейки околонулевой площади, имеет близко расположенные друг другу элементы, накладывающиеся друг на друга ячейки и другие особенности. Существуют подходы, позволяющие сгладить эти негативные эффекты, например, применение возмущений расположения узлов для устранения особенностей сетки [120] или восстановление потерянных из-за ошибок точности звеньев ломаной пересечения [121]. Среди современных работ, в которых предпринимаются попытки компенсировать ошибки точности при работе с числами с плавающей точкой, выделяется подход с использованием неявного представления точек пересечения через другие точки, чьи координаты представлены точно (предикатные методы) [122, 123]. Избежать ошибок определения пересечений, связанных с погрешностью вычислений, можно с

помощью выполнения вычислений в рациональных координатах, в которых задачи поиска пересечения элементов сетки выполняются без потери точности, но такой подход сопряжен с высокими вычислительными затратами [124].

## 1.5 Векторизация вычислений

Самым низким уровнем распараллеливания вычислений является распараллеливание на уровне отдельных инструкций. Векторизация вычислений это низкоуровневая оптимизация, применение которой способно кратно повысить производительность программного кода и сократить энергопотребление [173]. Использование векторных инструкций позволяет применять одинаковые операции к наборам данных, упакованных в векторные регистры. Векторные инструкции представлены во многих современных микропроцессорным архитектурах, но в наибольшей степени они развиты в архитектуре x86.

Архитектура x86 представлена богатым набором векторных инструкций, которые добавлялись постепенно [174]. Первый набор инструкций MMX был предназначен для обработки аудио и видеоданных и содержал операции по работе с целыми числами, упакованными в 64-битные регистры. В дальнейшем в архитектуру были добавлены несколько наборов SSE и AVX инструкций, и наконец AVX-512 с возможностью выборочной обработки элементов векторов.

В архитектуре ARM векторные инструкции поддержаны в 128-битном SIMD расширении Neon, и они являются по сути аналогом SSE [175]. Инструкции Neon поддерживают работу с целыми элементами данных размера от 8 до 64 битов, а также с вещественными значениями размера 16, 32 и 64 бита. Инструкции Neon не поддерживают выборочную обработку элементов данных векторов, но в наборе предусмотрены возможности комбинирования данных путем выполнения операций сдвига и перестановки элементов вектора. Современное расширение SVE [176] позволяет оперировать терминами векторных вычислений без привязки к длине вектора (при условии кратности 128), которая определяется аппаратной реализацией (например, в микропроцессоре Fujitsu A64FX используется набор SVE с длиной вектора 512-бит [177]).

Для поддержки векторных и матричных вычислений в архитектуре Power предусмотрено векторное расширение VMX [178]. Поддержаны векторные регистры, состоящие из целых значений размера до 32 битов, либо значения FP32. В расширение входят арифметические и логические операции, операции доступа в память, выборочная обработка элементов векторов не поддержана.

В архитектуре «Эльбрус» присутствует поддержка векторных операций для векторов размера 64 бита, что позволяет параллельно выполнять арифметические операции над их элементами [179]. Так как выборочная обработка элементов векторов в «Эльбрус» отсутствует, то для векторизации управления в цикле используются логические операции обнуления ненужных частей соответствующих векторных регистров с последующим их слиянием [180].

Можно отметить современные китайские микропроцессорные архитектуры, также поддерживающие векторизацию. Так, в оригинальной архитектуре LoongArch, на базе которой создан микропроцессор Loongson 3A6000, поддержаны векторные инструкции, работающие с упакованными векторами размера 128 и 256 бит [181]. А в архитектуре Sunway [182], на базе которой собрана одна из мощнейших линеек китайских суперкомпьютеров, в систему команд включены инструкции для работы с 512-битными векторами.

Наиболее продвинутым набором векторных инструкций является AVX-512, поддержку которого можно встретить в микропроцессорах Intel и AMD. Инструкции AVX-512 работают с векторными регистрами zmm размера 512 бит, каждый из которых вмещает в себя 8 элементов в формате вещественных чисел двойной точности, что с учетом комбинированных операций приводит к возможности выполнения 16 операций сложения и умножения двойной точности за одну векторную команду. Из этого следует, что на микропроцессорах с поддержкой AVX-512 без использования векторизации даже теоретически невозможно добиться производительности более 6,25% от пиковой. Особенности векторных инструкций AVX-512 позволяют применять их для векторизации сложного программного контекста, но оптимизирующий компилятор не всегда успешно справляется с этой задачей. Для обеспечения программисту возможности прямого использования векторных инструкций существует набор

функций-интринсиков [183], которые в дальнейшем заменяются компилятором на конкретные векторные инструкции или последовательности инструкций. Интринсики упрощают разработку векторизованного кода и позволяют вести разработку в терминах функций. Можно считать, что впервые AVX-512 был поддержан в 2016 году в микропроцессорах Intel Xeon Phi KNL [184], более раннее поколение Intel Xeon Phi Knights Corner (KNC) представляет собой ускоритель, и векторный код для него не является x86 совместимым.

В мире активно проводятся исследования, направленные на повышение производительности программных кодов с помощью векторизации. В работе [185] путем векторизации безусловных операций продемонстрировано повышение производительности газодинамического решателя на 200%. В работе [186] описывается сравнение реализации римановских решателей в применении к теории мелкой воды, одним из результатов работы является ускорение решателя с помощью инструкций AVX-512 в 16,7 раз при работе с числами FP32. В исследовании [187] в помощью векторизации достигнуто ускорение в 3,27 газодинамического решателя ADflow, работающего на структурированных расчетных сетках. В работе [188] рассмотрена реализация расчета гравитационного взаимодействия между $N$ телами, использование набора инструкций AVX-512 позволило ускорить работу приложения в 2 и более раз. В работе [189] авторы успешно применили векторные инструкции в задаче поиска сходных участков в белковых последовательностях. В работе [190] предложена векторная версия быстрой сортировки (vqsort), превышающая по эффективности реализацию из стандартной библиотеки (qsort) более чем в 10 раз. В работе [191] рассматриваются реализация бессеточного метода решения задачи газовой динамики и векторизация этого метода, использование векторных инструкций AVX-512 позволило ускорить исходный код в 6,3 раза. В статье [192] векторизация применяется для алгоритма анализа генетических вариаций, использование AVX-512 позволило добиться ускорения расчетов от 7 до 12 раз по сравнению с оригинальным алгоритмом. В работе [193] рассматривается прямое применение инструкций AVX-512 для задачи обработки временных рядов, что привело к ускорению результирующего кода в 4 раза по

сравнению с автоматической векторизацией, выполняемой оптимизирующим компилятором. Статья [194] посвящена оптимизации алгоритмов шифрования с открытым ключом, в частности, рассматривается оптимизация блочного варианта умножения Монтгомери (достигнуто ускорение основных операций в 1,5-1,9 раза). В статье [195] показана реализация алгоритма блочного шифрования на различных архитектурах, включая GPU и CPU, использование AVX-512 позволило ускорить его в 9,5 раза по сравнению с оригиналом. В работе [196] векторизация с помощью AVX-512 применяется для оптимизации быстрого преобразования Фурье, что привело к ускорению примерно в 1,5 раза. Работа [197] посвящена применению подмножества инструкций Integer Fused Multiply-Add (IFMA) для повышения эффективности реализации деления больших целых чисел, в результате оптимизации было достигнуто ускорение функционала на 25–35%. В работе [198] продемонстрирован практический подход к векторизации расчета попарного взаимодействия множества частиц, использование инструкций AVX-512 в явном виде привело к ускорению исполнения программы в 3,3 раза. В статье [199] предлагается подход к развитию средств автоматической векторизации расчетных циклов, основанный на анализе и эквивалентных преобразованиях графа зависимостей между операциями, расположенными в теле цикла. Этот подход позволяет переупорядочить операции внутри цикла, снизив их количество и укоротив критический путь исполнения [200], и сгруппировать для объединения в векторные инструкции.

В качестве практического руководства по созданию векторизованного программного кода с помощью функций-интринсиков и с помощью языка ассемблера для широкого спектра практических задач, включая задачи линейной алгебры, обработки изображений, реализацию сверток и другие, можно рассматривать работу [201]. В рассмотренных работах и рекомендациях по программированию с использованием векторных инструкций усилия по векторизации направлены в основном на наиболее вложенные циклы. В настоящей работе рассмотрены методы и предложена методика векторизации программного кода, применимая к широкому спектру расчетных приложений.

## 1.6 Использование искусственного интеллекта при моделировании объектов со сложной геометрией

Технологии искусственного интеллекта широко применяются в исследованиях и разработках, задачи моделирования физических процессов и сложных технических систем не являются исключением. Искусственные нейронные сети все чаще используются для автоматизации отдельных процессов суперкомпьютерного моделирования, миниминизации участия человека в подготовке и анализе данных, и результаты уже проведенных вычислений компьютерного моделирования используются для обучения таких сетей.

Генерация расчетных сеток является одним из самых активных направлений по использованию искусственных нейронных сетей в области суперкомпьютерного моделирования. В работе [202] предложен подход к созданию расчетной сетки для численного решения задач газовой динамики, сетка создается с помощью искусственной нейронной сети, обученной на решениях задач с похожими конфигурациями, и учитывает возможные предсказанные особенности решения, которое должно быть получено в процессе расчетов. Такое создание расчетной сетки, близкой к оптимальной для расчета конктерной конфигурации, позволяет повысить эффективность выполнения расчетов компьютерного моделирования. В работе [203] описано применение искусственной нейронной сети для создания структурированной сетки в области по заданным разбиениям на ее границе и функции потерь, основанной на заданных дифференциальных уравнениях внутри области. Известны исследования по использованию LLM (Large Language Model) для построения сеток трехмерных объектов и поверхностей на основе их текстового описания, в качестве примеров можно привести такие решения, как LLaMA-Mesh [204] и MeshLLM [205]. Создание сеток трехмерных объектов на основе текстового описания с помощью LLM на сегодняшний день только начинает развиваться, однако в дальнейшем результаты этих исследований могут развиться в решения, пригодные для применения в области суперкомпьютерного моделирования сложных мультифизических процессов.

Технологии искусственного интеллекта давно зарекомендовали себя в качестве эффективного инструмента для обнаружения аномалий и дефектов в различных приложениях [206, 207]. В частности они применяются для обнаружения и устранения аномалий цифровых трехмерных моделей [208]. В работе [209] описано решение, основанное на применении графовых нейронных сетей (Graph Neural Network, GNN), которое используется для перестроения поверхностной расчетной сетки в задаче моделирования обледенения. Для обучения сети используются решения, полученные при использовании классических алгоритмов перестроения поверхностной сетки. В работе [210] предлагается решение по автоматическому огрублению расчетной сетки с помощью GNN при моделировании физических процессов.

Инструменты для создания, обработки и корректирования расчетных сеток, пригодные для применения в суперкомпьютерных приложениях, основанные на технологиях искусственного интеллекта, демонстрируют тенденции для развития в будущем, однако для обеспечения их корректной работы и формирования обучающих наборов данных необходимо развитие и реализация классических методов и алгоритмов работы с сетками.

Отдельной областью исследований является применение ИИ к моделированию физических процессов. В работе [211] описан метод ускорения газодинамических расчетов с помощью использования сверточной нейронной сети (Convolutional Neural Network, CNN). При решении дифференциальных уравнений в частных производных (ДУЧП) эффективным оказывается применение физически информированных нейронных сетей (Physics-Informed Neural Network, PINN) [212, 213], позволяющих использовать знания о физических процессах, что ограничивает пространство допустимых решений. Активно развиваются решения, основанные на LLM, направленные на автоматизацию и оптимизацию действий, необходимых при выполнении моделирования физических процессов. В [214] описано решение CFDLLMBench, аккумулирующее в себе экспертные знания в области CFD для создания программного кода численного решения типовых уравнений и функционал по конфигурированию и организации вычислений с помощью OpenFOAM для решения задачи.

Широкое применение суперкомпьютерного моделирования в проектировании технических систем связано с задачей выбора оптимальной конфигурации при большом количестве параметров. По мере усложнения проектируемых систем и роста количества параметров возникла проблема дефицита вычислительных ресурсов, что привело к появлению концепции суррогатного моделирования [215–217], то есть использования упрощенных суррогатных моделей, обученных на результатах суперкомпьютерного моделирования на ограниченных наборах данных. Но даже использование суррогатного моделирования может лишь частично снизить потребность в вычислительных ресурсах, и не умаляет актуальность проблемы эффективного их использования.

В настоящее время LLM демонстрируют впечатляющие результаты по автоматической генерации программного кода на различных языках программирования по текстовому описанию решаемой задачи [218,219], чему в немалой степени способствовало развитие открытых репозиториев программного кода и платформ по обучению программированию, что обеспечило создание больших обучающих наборов для искусственных нейронных сетей. Применение LLM для автоматической векторизации циклов, также в последнее время развивается. В работе [220] описано примерение LLM для автоматической векторизации циклов с простым управлением при помощи функций-интринсиков для инструкций AVX2. В результате наблюдается векторизация даже тех циклов, которые не были векторизованы с помощью компиляторов gcc или icc, хотя в результате оптимизации не всегда получается эквивалентный векторный код (то есть нейронная сеть допускает ошибки при создании кода).

Технологии искусственного интеллекта демонстрируют тенденцию к расширению своего применения в задачах моделирования физических процессов, сложных технических систем, объектов со сложной геометрии в условиях мультифизических расчетов. Особенные перспективы от их примененнния ожидаются при решения рутинных однотипных задач, которые могут быть автоматизированы. К таким задачам относятся задачи построения и коррекции расчетных сеток, перебора параметров при решении оптимизационных задач, организация стадий и сценариев организации мультифизических расчетов.

Развитие методов решения задач компьютерного моделирования с помощью искусственных нейронных сетей происходит параллельно с развитием классических подходов и должно опираться на них для верификации и валидации результатов моделирования.

## 1.7 Задача моделирования обледенения поверхности

Задача моделирования обледенения поверхности является представителем задач моделирования объектов со сложной геометрией, так как в ней приходится иметь дело с моделированием ледяных наростов, имеющих сложную форму и изменяющихся во времени. Разработанные в диссертации математические методы и алгоритмы повышения производительности программ для моделирования объектов со сложной геометрией нашли свое отражение в задаче моделирования обледенения и были реализованы в программных кодах, в частности в программном комплексе расчета процесса обледенения элементов авиационных силовых установок «Кристалл» [126, 127].

Задача моделирования обледенения летательных аппаратов является критически важной для обеспечения безопасности полетов [221]. Известны случаи, когда обледенение несущих частей и силовых установок летательных аппаратов приводили к авиакатастрофам с большим количеством человеческих жертв, что свидетельствует о важности этой задачи и востребованности результатов ее решения. Требования и рекомендации к процессам проектирования, разработки и сертификации авиационных систем с учетом эксплуатации в режимах обледенения зафиксированы во многих нормативных документах[2].

Задача моделирования обледенения поверхности является комплексной. Для получения адекватной картины обледенения необходимо учитывать множество связанных между собой процессов, включая обтекание тела, выпадение на поверхность влаги и ледяных кристаллов [222], взаимодействие выпадающего вещества с поверхностью [223], течение жидкости по поверхности в виде тонкой пленки или отдельных нитей [224], теплопроводность на поверхности,

[2] 14 CFR – Part 25 (Appendix C, O), Part 29 (Appendix C), Part 33 (Appendix D); FAA AC 25-28; ARP 4754; ARP 4761; P 4761; ISO 11076:2020; Doc 9640; FAA Report RD-77-76; НЛГ БАС-СТ.

а также через слой жидкости и льда [225], и другие процессы. В процессе образования слоя льда меняется геометрия рассматриваемой поверхности, так как форма образовавшихся ледяных наростов влияет на все связанные процессы, что приводит к необходимости перестроения расчетных сеток [110, 114].

Моделирование обледенения выполняется, как правило, на поверхностных расчетных сетках и состоит из отдельных стадий, которые циклически следуют друг за другом. Основной стадией является расчет интенсивности нарастания льда в рамках отдельных ячеек расчетной сетки (скорость накопления льда в каждой ячейке). Для вычисления интенсивности ледообразования в ячейках сетки существует множество моделей обледенения [226–228], в которых учитываются различные состояния льда, течение жидкой пленки по поверхности, тепловые потоки, выпадение на поверхность влаги, ледяных кристаллов и переохлажденных капель [229, 230], различные механики плавления и срыва ледяных наростов [231] и многие другие факторы. Результатом этой стадии является информация о количестве льда, накопленного в каждой ячейке. Далее выполняется перестроение поверхности обледеневающего тела. При этом перестроение должно выполняться таким образом, чтобы изменение объема рассматриваемого тела соответствовало вычисленному объему льда. После перестроения поверхностной сетки необходимо произвести пересчет газодинамических величин вокруг нее, таких как поле скоростей или коэффициент улавливания влаги, так как изменение геометрии поверхности существенным образом влияет на картину обтекания, траектории движения капель и зоны вторичного выпадения. После пересчета газодинамических характеристик моделирование обледенения может быть продолжено.

Таким образом, можно выделить основные стадии комплексного многостадийного расчета в задаче моделирования обледенения (рис. 1.6). Стадия “Аэродинамика”, на которой выполняется расчет аэродинамического поля вокруг обледеневающего тела. Стадия “Динамика капель”, на которой выполняется расчет полета влаги и ледяных кристаллов вокруг обледеневающего тела, расчет коэффициента улавливания влаги, интенсивности выпадения влаги, в том числе и вторичного выпадения. Стадия “Обледенение поверхности” –

центральная стадия в задаче обледенения, на которой выполняется расчет структуры ледяного покрова в каждой ячейке сетки, включая состояние слоя подледной воды, течение жидкости по поверхности льда, характеристики слоя льда. Стадия “Перестроение поверхности”, на которой выполняется перестроение сетки, описывающей поверхность ледяного нароста, в соответствии с вычисленными в ячейках сетки высотами льда. Все стадии выполняются циклически, содержат свои решатели, результаты работы решателей одной стадии являются входными данными для решателей последующих стадий.

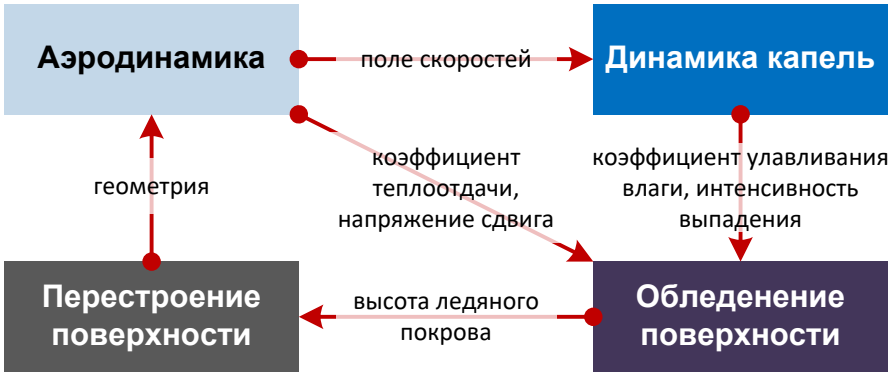


Рис. 1.6. Схема многостадийного расчета обледенения

Частота вызова стадий “Перестроение поверхности”, “Аэродинамика” и “Динамика капель” определяется продолжительностью отрезка моделируемого физического времени, на котором выполняется расчет обледенения без перестроения поверхностной сетки (и без коррекции соответствующих данных окружающего воздушного потока).

Во время расчета на стадии “Обледенение поверхности” каждая ячейка поверхностной расчетной сетки представлена в виде сложной многослойной структуры, которая может содержать следующие слои, перечисленные на рис. 1.7 снизу вверх: “система обогрева” – элементы которой находятся внутри тела, “твердое тело” – остается неизменным в процессе моделирования обледенения, “поверхность тела” – граничащая с внешней областью поверхность, “подледная вода” – слой находящейся подо льдом жидкости, “слой льда” – различающийся по плотности и другим характеристикам, “текущая вода” – текущая по поверхности льда жидкость, “окружающий воздух” – окружающее

воздушное пространство. В зависимости от модели могут использоваться либо игнорироваться отдельные слои. Так, при моделировании обледенения по модели Мессингера не используется слой “подледная вода” [232], однако его использование оправдано при моделировании систем противообледенения на основе задачи Стефана, так как в этом случае присутствует подтаивание слоя льда снизу [233]. Во время выполнения расчетов осуществляется обмен потоками вещества и тепла между отдельными слоями, а также между соседними ячейками.

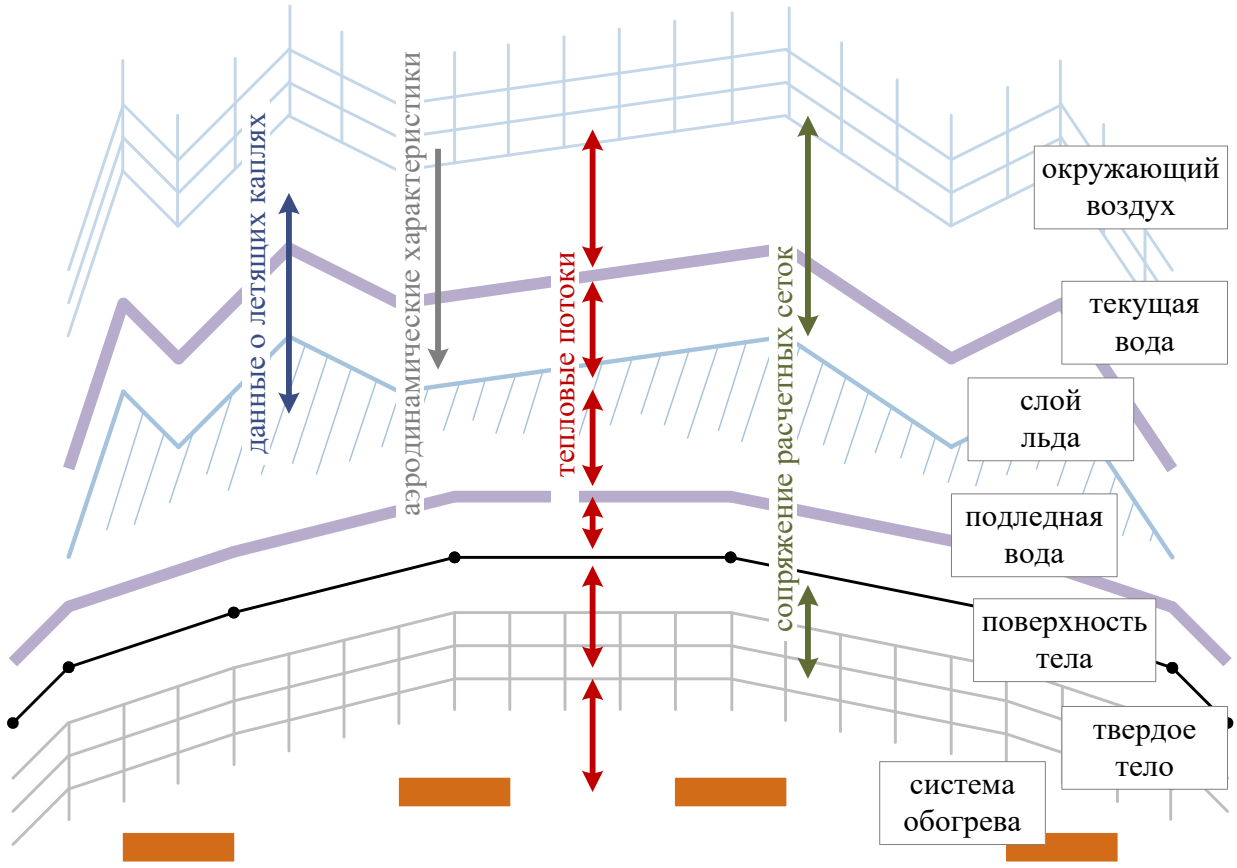


Рис. 1.7. Слои, используемые при моделировании обледенения

Перестроение сетки делится на локальное и глобальное. При локальном перестроении сетки ее узлы смещаются в направлении роста льда таким образом, чтобы заметаемый при движении каждой отдельной ячейкой объем пространства соответствовал объему накопленного в ячейке льда (целевой объем). Таким образом, перестроенная поверхность будет соответствовать поверхности ледяного покрова. Как правило, локальное перестроение выполняется

в направлении, перпендикулярном поверхностной сетке, что соответствует смещению узла в направлении, равном среднему нормалей инцидентных ячеек (такое смещение также называют смещением в направлении биссектрис, или методом биссектрис) [110]. В общем случае поиск смещений узлов при фиксированных направлениях смещения для достижения целевого объема каждой ячейкой может не иметь точного решения, а также может приводить к вырождению сетки (возникновение дефектов, шумов, самопересечений) или понижению ее качества, поэтому после некоторого количества шагов локального перестроения необходимо использовать глобальное перестроение сетки для обеспечения ее корректного состояния. Глобальное перестроение направлено на повышение качества сетки и удалению возникающих дефектов, оно состоит из различных видов сглаживания и может приводить к изменению топологии сетки вплоть до полного перестроения. Глобальное перестроение сетки является более ресурсозатратными, чем локальное перестроение, поэтому оно применяется по необходимости. Использование локального перестроения, позволяющего сглаживать возникающие локальные дефекты, позволяет использовать глобальное перестроение реже, что приводит к увеличению моделируемого физического времени процесса обледенения без глобального перестроения. Более редкое применение глобального перестроения сетки приводит к повышению производительности расчетов при моделировании обледенения поверхности.

Компьютерное моделирование обледенения является ресурсоемким и включает в себя не только выполнение вычислений на поверхностной сетке, но и постоянное ее перестроение, пересчет газодинамических характеристик в окружающем пространстве и различных характеристик поверхности, таких как коэффициент улавливания влаги или коэффициент теплоотдачи. Перестроение поверхностной сетки, адекватно описывающей геометрию образующихся ледяных наростов, должно отвечать критериям точности и надежности. Возникновение дефектов сетки, таких как острые пики и изломы, трещины и впадины, самопересечения сетки, может привести к нефизичным результатам или даже к аварийному завершению расчета и необходимости его перезапуска,

что приводит к дополнительному потреблению вычислительных ресурсов. Для моделирования обледенения на значимом промежутке времени требуется использование существенных вычислительных ресурсов и применение методов повышения производительности вычислений на поверхностных и объемных расчетных сетках на всех уровнях распараллеливания: на распределенной памяти, на общей памяти и на уровне отдельных инструкций.

В пользу актуальности и востребованности исследований и разработок в задаче моделирования обледенения свидетельствует большое количество существующих и разрабатываемых программных пакетов для решения этой проблемы. Зарубежным мировым флагманом в области моделирования обледенения считается пакет инженерного проектирования ANSYS, включающий в себя следующие модули: FENSAP – для моделирования воздушного потока вокруг тела; DROP3D – для расчета траекторий переохлажденных капель; ICE3D – для расчета состояния льда и жидкости на поверхности, для определения толщины ледяного покрова; CHT3D – для расчета сопряженного теплопереноса [234–236]. Из отечественных решений для моделирования обледенения следует отметить модуль iceFoam [58] как часть открытого программного обеспечения OpenFOAM, решение для моделирования обледенения в составе пакета инженерного анализа ЛОГОС [237, 238], методику IceVision [95] в составе пакета FlowVision, а также ряд других программных решений, таких как LEWICE [239], AIPAC [240], ONERA [241], TRAJICE [111] и другие.

Среди особенностей моделирования обледенения с помощью ANSYS (FENSAP-ICE) можно отметить многостадийный трехмерный расчет с использованием сопряженных модулей FENSAP, DROP3D, ICE3D, CHT3D [242]. Расчет ледообразования основан на термодинамической модели Мессингера [236], при этом учитываются такие факторы, влияющие на ледообразование, как взаимодействие поверхности с ледяными кристаллами, отскок и разбрызгивание крупных переохлажденных капель, вторичное выпадение влаги на поверхность, шероховатость поверхности, отрыв льда от поверхности, теплообмен между воздухом и частицами. Для расчета воздушного потока с целью повышения точности анализа возможно использование модуля ANSYS

Fluent [243]. Для локального перестроения поверхностной сетки используется смещение узлов по методу биссектрис в соответствии со скоростью нарастания льда, а пространственная сетка перестраивается с помощью подхода ALE (Arbitrary Lagrangian-Eulerian) [244, 245], базирующегося на решении уравнения Лапласа [246]. Также для улучшения качества сетки и устранения дефектов используется локальная коррекция [247] и метод глобального перестроения [248], основанный на технике обтягивания (shrink wrapping remesh [249]), так как при использовании только локального перестроения возникают дефекты сетки, препятствующие проведению дальнейших расчетов (рис. 1.8). Глобальное перестроение сетки является более ресурсозатратной процедурой, и ее применение на каждом шаге затруднительно.

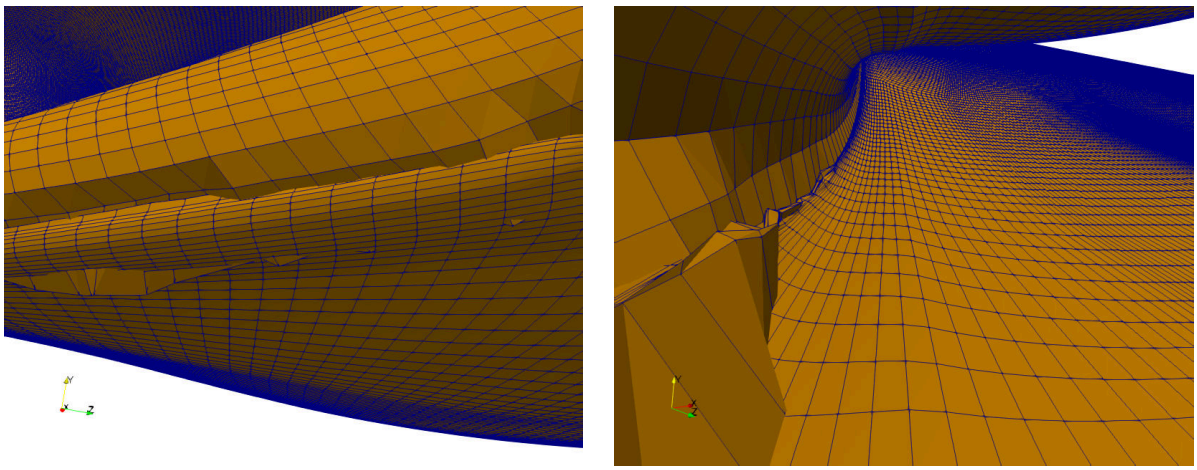

Рис. 1.8. Возникновение дефектов сетки (острые пики, самопересечения) во время локального перестроения при моделировании обледенения с помощью программного модуля FENSAP-ICE

Отечественный программный модуль iceFoam, разрабатываемый ИСП РАН на базе открытой библиотеки OpenFOAM [58], обладает широким функционалом и используется для моделирования обледенения в различных областях применения, в частности для моделирования обледенения летательных аппаратов [250] и морских судов [251]. Идеология расчета обледенения во многом схожа с используемой в ANSYS, используется расширенная модель Мессингера, а именно SWIM (Shallow Water Icing Model) [252]. Для определения состояния льда и жидкости в ячейке (обледенение в виде инея – rime ice, глад-

кий лед – glaze ice, теплая водяная пленка) решаются уравнения массового и теплового баланса, течение жидкой пленки по поверхности определяется направлением вектора напряжения сдвига. Локальное перестроение сетки выполняется с помощью смещения узлов по методу биссектрис, перестроение объемной сетки выполняется с помощью решения уравнения Лапласа [58]. Модуль поддерживает распараллеливание с помощью MPI [253].

Другое отечественное решение по моделированию обледенения представлено в пакете инженерного анализа ЛОГОС в модуле ЛОГОС Аэро-Гидро [237] и предназначено для расчета характеристик летательных аппаратов в условиях обледенения. При моделировании водяных частиц в расчетной области вокруг обтекаемого тела используется как Эйлерова многофазность (представвление водяной фазы в виде сплошной среды), так и Лагранжева (дискретное представление капель) [238]. Для моделировании обледенения используется модель Мессингера. В модуле учитывается шероховатость поверхности, состояние водяной фазы в различных состояниях, в том числе течение и замезание водяной пленки [254]. Изменение поверхностной сетки выполняется только в локальной постановке путем смещения узлов по методу биссектрис на величину, вычисляемую как среднее толщин ледяных наростов в инцидентных ячейках, при этом вычисляется взвешенное среднее по площадям ячеек [255].

От существующих решений принципиально отличается подход к моделированию обледенения, применяемый в методике IceVision, реализованной в рамках программного комплекса FlowVision [95]. Главное отличие методики IceVision от других подходов – использование технологии VOF (Volume of Fluid). С помощью этой технологии отслеживается объем жидкости в ячейке, при этом лед явно присутствует в ячейках расчетной области, и по этим данным определяется форма ледяных наростов [256]. В программном модуле реализован ряд таких особенностей как расчет сухого и влажного льда, срыв водяной пленки, отскок и разбрызгивание выпадающих на поверхность крупных переохлажденных капель, течение жидкости по поверхности в виде сплошной пленки и ручейков и другие [257]. В IceVision поверхность льда аппроксимируется с помощью декартовой объемной сетки в пространстве,

которая адаптируется к поверхности путем дробления ячеек пополам по каждому из трех направлений [258].

Предложенный в работе комплекс методов и алгоритмов повышения производительности программ для моделирования объектов со сложной геометрией реализован в программных кодах, в частности в рамках программного комплекса расчета процесса обледенения элементов авиационных силовых установок «Кристалл» [126, 127] и других программных средств [128–132].

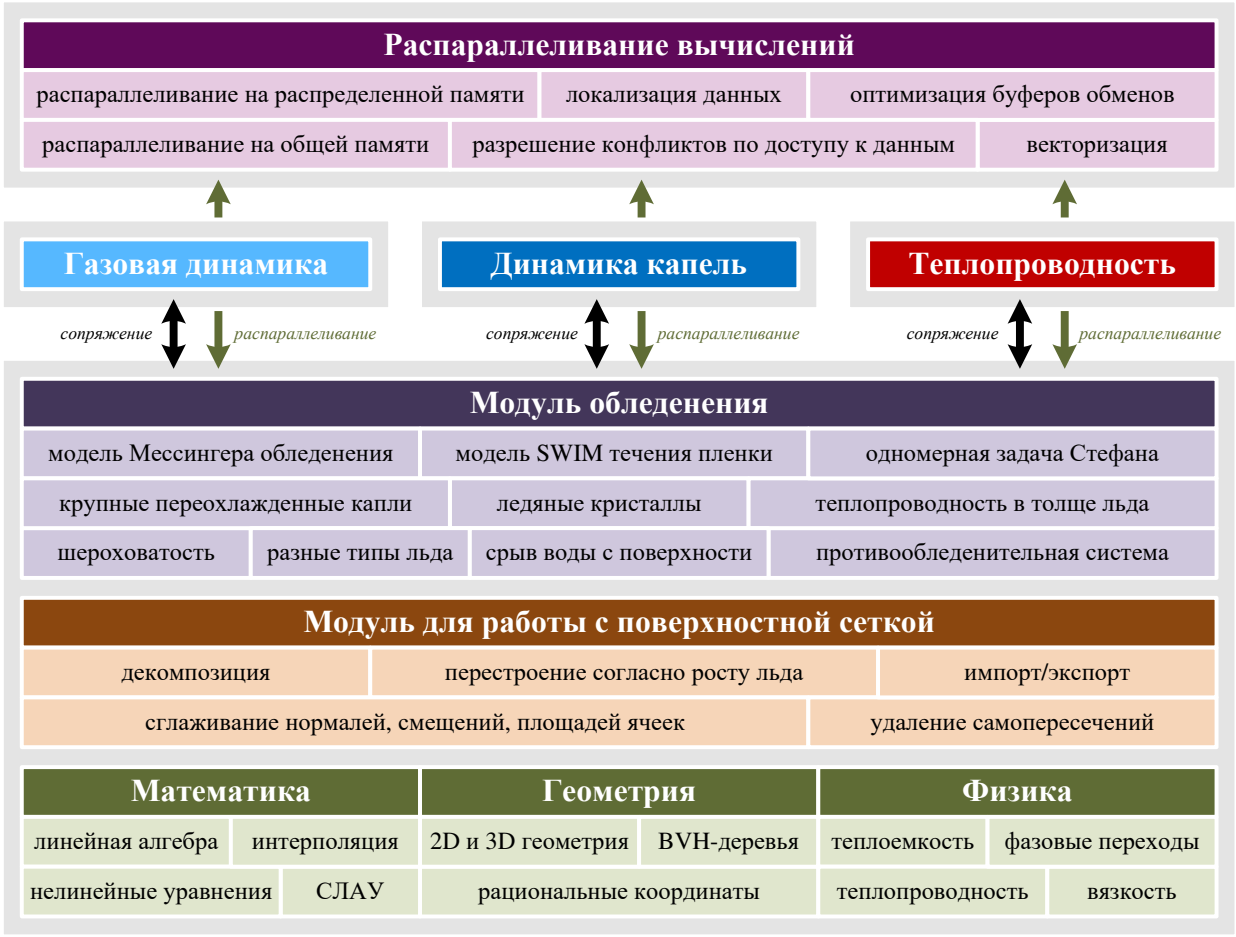


Рис. 1.9. Схема программного комплекса компьютерного моделирования процесса обледенения элементов авиационных силовых установок «Кристалл»

Программный комплекс «Кристалл» предназначен для моделирования обледенения и отличается множеством особенностей, позволяющих получать адекватную картину ледообразования. Поддержана интеграция с решателями в окружающем пространстве, из которых получаются характеристики окружающего потока (рис. 1.9, газовая динамика) и информация о выпадении на

поверхность влаги (рис. 1.9, динамика капель), а также с решателем, поставляющим информацию о поступающем тепловом потоке от поверхности (рис. 1.9, теплопроводность). В качестве внешних решателей могут выступать как зарубежные продукты (ANSYS Fluent, ANSYS FENSAP), так и отечественные решения (ПО «Лазурит» [259], ЛОГОС Аэро-Гидро).

В качестве основных особенностей программного комплекса «Кристалл» можно отметить следующие. Расчет обледенения (высоты ледяного покрова) выполняется на поверхностной неструктурированной сетке с треугольными ячейками. Для расчетов может использоваться как расширенная модель Мессингера (с моделью SWIM течения жидкой пленки по поверхности льда), так и модель с включенным слоем подледной воды, расчет при наличии которого производится с помощью решения одномерной задачи Стефана в каждой ячейке сетки [260]. Поддержан модуль для работы с крупными переохлажденными каплями, реализовано их взаимодействие с поверхностью и разбрызгивание [261]. Механизм течения жидкой пленки по поверхности реализован с учетом срыва воды и вторичного выпадения [262]. Поддержан модуль для работы с ледяными кристаллами, реализован механизм взаимодействия с поверхностью, включающий эффекты отскока и прилипания [263]. Во время расчета используется распределение температур в толще льда, вычисленная шероховатость поверхности, моделируются различные типы льда (наледь в виде инея – rime ice, гладкий лед с водяной пленкой – glaze ice, лед с впитыванием влаги – spongy ice). Поддержаны различные виды моделирования противообледенительной системы (ПОС), в том числе по расписанию и с данными, получаемыми из внешнего решателя.

Для повышения производительности и надежности вычислений реализован функционал по работе с поверхностной неструктурированной сеткой с поддержкой ее декомпозиции для распараллеливания, локального перестроения (смещение узлов по методу биссектрис), глобального сглаживания сетки по различным показателям (сглаживание нормалей, сглаживание величин высот льда для удаления поверхностного шума, сглаживание площадей ячеек), удаления самопересечений сетки [264].

В виде отдельных модулей реализован функционал математики (операции линейной алгебры, интерполяция функций, решение нелинейных уравнений для массового и теплового балансов [265], решение систем линейных уравнений), геометрии (операции с объектами на плоскости и в пространстве, работа с геометрическими примитивами с вещественными и рациональными координатами, реализация BVH-деревьев для быстрого поиска пересечения ячеек), физики.

В программном комплексе реализован ряд решений, позволяющих выполнять распараллеливание вычислений на суперкомпьютере на распределенной и общей памяти на объемной и поверхностной расчетной сетке, а также векторизовать и подключать для исполнения отдельные модули.

## 1.8 Описание границы ледяного нароста в задаче моделирования обледенения

Перестроение поверхностной сетки при моделировании объектов со сложной геометрией базируется, как правило, на смещении ее элементов в направлении, перпендикулярном описываемой поверхности. В качестве примеров можно привести перестроение двумерной сетки в задаче ледообразования [110, 111] или построения пристеночного слоя в задаче газовой динамики [112]. При этом в [112] для сглаживания локальных дефектов используется коррекция положения узлов в зависимости от угла между инцидентными ячейками, а в [113, 114] описан ряд методов, направленных на глобальное сглаживание результирующей сетки для снижения зашумленности и устранения дефектов. Появление дефектов сетки обусловлено ее дискретностью, при использовании только локального перестроения возникновение дефектов неизбежно, поэтому требуется применение различных видов глобального перестроения и сглаживания. Глобальное перестроение сетки гораздо более ресурсозатратно, применение его на каждом шаге изменения сетки связано с существенных замедлением расчетов, поэтому требуется разработка методов локального перестроения сетки, обладающих свойством предотвращения появления дефектов.

Для анализа обоснованности необходимости сглаживания возникающих локальных дефектов рассмотрим границу между подобластями не в виде дискретной расчетной сетки, а в виде поверхности в аналитическом виде и рассмотрим изменение этой поверхности при продвижении ее фронта. Перестроение поверхности будем рассматривать в применении к задаче моделирования обледенения, в которой граница представляет собой границу поверхности льда (разделяет подобласти ледяного нароста и окружающего пространства). Анализ будем проводить в предположении продвижения фронта поверхности на одну и ту же величину, что соответствует геометрическому принципу Гюйгенса, который впервые был сформулирован применительно к геометрической оптике [266]. В применении к задаче моделирования обледенения это означает равномерное наращивание льда одновременно во всех направлениях с одинаковой скоростью.

Рассмотрим подобласть, ограниченную замкнутой двусторонней поверхностью. Эта поверхность разделяет пространство на две части – саму рассматриваемую подобласть и оставшуюся внешнюю часть пространства (рис. 1.10, а). Будем считать, что поверхность является гладкой, то есть в каждой ее точке можно построить касательную плоскость и определить внешнюю нормаль (рис. 1.10, а). Выполним анализ эволюции поверхности, при которой каждая ее точка смещается в направлении своей внешней нормали с одной и той же скоростью в соответствии с геометрическим принципом Гюйгенса, к котором поверхность $G$ – это фронт, новое положение которого через малое время $\Delta t$ при скорости продвижения $v$ представлено общей огибающей семейства сфер с центрами в точках $G$ и радиусами $r = v\Delta t$.

Будем выполнять анализ развития возникающих дефектов поверхности в двумерном случае, то есть на плоскости, где поверхность представлена замкнутой кривой без самопересечений (рис. 1.10, а). В качестве анализируемых локальных дефектов будем рассматривать острые пики и острые впадины. Дополнительно рассмотрим развитие впадины в общем случае. При этом рассматривается распространение фронта только во внешнюю область. На рис. 1.10 приведены примеры распространения фронта в простейших случа-

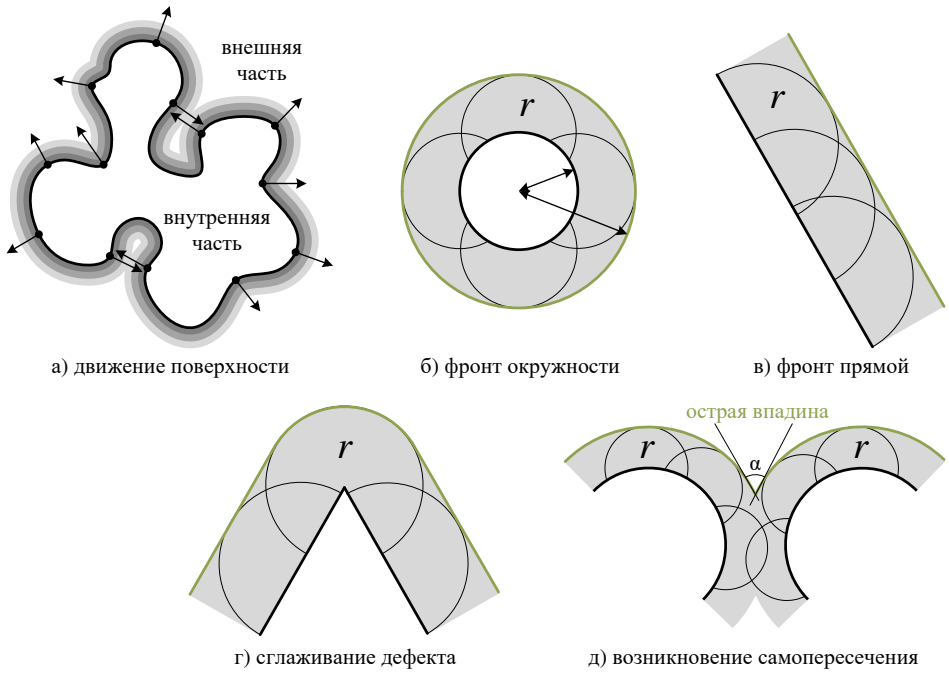


Рис. 1.10. Эволюция поверхности
согласно геометрическому принципу Гюйгенса

ях. Например, фронтом окружности является новая окружность с тем же центром и радиусом на $r$ больше (рис. 1.10, б), а фронтом прямой является параллельная ей прямая, находящаяся на расстоянии $r$ от нее (рис. 1.10, в).

Проанализируем потенциальные дефекты рассматриваемой замкнутой кривой без самопересечений. Пусть в некоторой точке $P \in G$ кривая не является гладкой, то есть невозможно провести касательную к ней. Если в малой окрестности точки $P$ площадь внешней части больше площади внутренней части, то такой дефект будем называть острым пиком, в противном случае – острой впадиной. Нетрудно видеть, что острый пик при эволюции поверхности сглаживается, и локальная кривизна фронта этого дефекта $1/r$ (рис. 1.10, г). Если рассматривается граница в виде замкнутой гладкой кривой, то острые пики не могут появиться в процессе эволюции, но при возникновении самопересечений фронта могут появиться острые впадины (рис. 1.10, д). Под углом при остром дефекте будем понимать угол между соответствующей парой касательных, проведенных к частям кривой с двух сторон от точки $P$, как показано на рис 1.10 д.

**Лемма 1.1.** *Пусть $P$ – точка острой впадины, образованная двумя частями кривой с кривизной $1/r_1$ и $1/r_2$ в этой точке соответственно, а $\alpha$ – угол при этой впадине. Тогда при продвижении фронта на малую величину $r$ сглаженный угол $\beta$ при этой острой впадине выражается как*

$$\beta = \arccos\left(\frac{\cos\alpha + 1}{(1+\delta_1)(1+\delta_2)} - 1\right), \tag{1.1}$$

*где $\delta_1 = \frac{r}{r_1}$, $\delta_2 = \frac{r}{r_2}$ (рис. 1.11).*

Для доказательства этого утверждения достаточно записать выражение длины отрезка $O_1O_2$ по теореме косинусов из треугольников $O_1PO_2$ и $O_1QO_2$ с учетом того, что $\angle O_1PO_2 = \pi - \alpha$, $\angle O_1QO_2 = \pi - \beta$:

$$\begin{cases} O_1O_2^2 = r_1^2 + r_2^2 + 2r_1r_2\cos\alpha, \\ O_1O_2^2 = (r_1+r)^2 + (r_2+r)^2 + 2(r_1+r)(r_2+r)\cos\beta, \end{cases} \tag{1.2}$$

откуда значение $\beta$ явно выражается через $\alpha$ в виде (1.1). ■

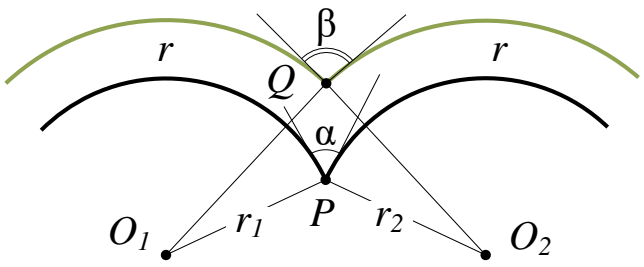


Рис. 1.11. Оценка сглаживания острой впадины

При $\delta_1 > 0$, $\delta_2 > 0$ нетрудно видеть, что $\beta > \alpha$, то есть при продвижении фронта наблюдается сглаживание угла при острой впадине. Заметим, что при нулевой кривизне $1/r_1$, $1/r_2$ из (1.1) получается $\beta = \alpha$.

Для оценки впадины в глобальном смысле без ограничения общности будем рассматривать гладкую замкнутую кривую. Рассмотрим выпуклую оболочку этой кривой, которая состоит из выпуклых во внешнюю область участков кривой и отрезков, касающихся кривой в двух точках. Рассмотрим

один из таких отрезков, введя систему координат таким образом, чтобы ось $OX$ была направлена параллельно этому отрезку, как показано на рис. 1.12. Тогда участок кривой под этим отрезком представляет собой гладкую впадину. Для анализа сглаживания этой впадины без ограничения общности будем считать, что во введенной системе координат профиль впадины может быть представлен в виде функции $y = f(x)$. Тогда может быть сформулирована следующая лемма.

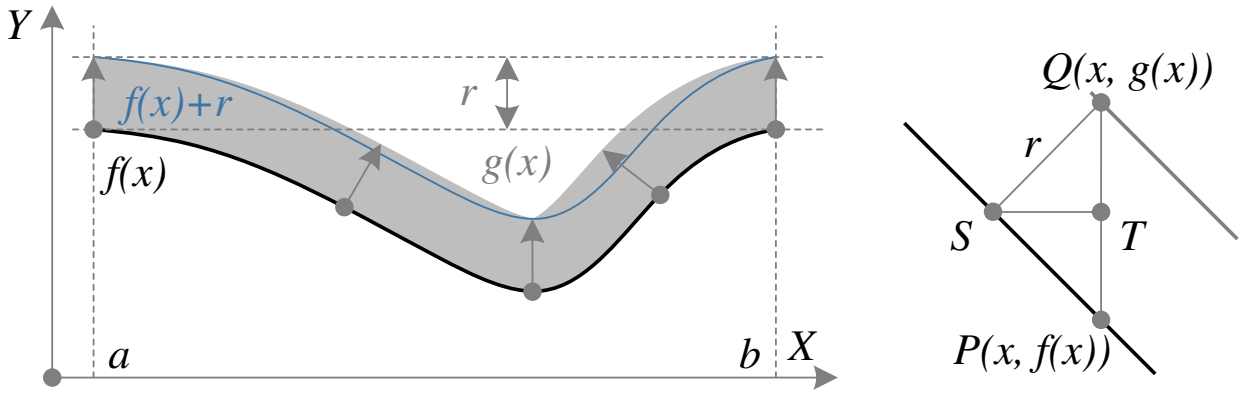


Рис. 1.12. Оценка сглаживания впадины

**Лемма 1.2.** *Пусть на сегменте* $[a, b]$ *определена гладкая функция* $f(x)$ *такая, что* $f(a) = f(b) = \max_{x\in[a,b]} f(x) = M$ *и* $f'(a) = f'(b) = 0$. *Эта функция на сегменте* $[a, b]$ *образует впадину, размер которой равен* $V_{[a,b]}(f) = \int_a^b (M - f(x))dx$. *Тогда при продвижении фронта на малую величину* $r$ *верно*

$$V_{[a,b]}(f)'_r = -\int_a^b \left(h(x)\left(\sqrt{(h(x))^2 + 1} + h(x)\right)\right)^{-1} dx, \tag{1.3}$$

*где* $h(x) = 1/|f'(x)|$.

Рассмотрим продвижение фронта на малую величину $r$. В силу малости $r$ считаем, что после продвижения фронт описывается функцией $g(x)$. В силу $f'(a) = f'(b) = 0$ имеем $g(a) = g(b) = M + r$. Рассмотрим произвольное $x \in (a, b)$. При малом $r$ можно считать, что точка $P(x, f(x))$ на графике функции $f(x)$ соотвествует точке $Q(x, g(x))$ на графике функции $g(x)$. При

этом можно считать, что в рассматриваемой окрестности точки $P$ функция $f(x)$ линейна, а точка $Q$ является образом некоторой точки $S$ с графика функции $f(x)$ такой, что $SQ = r$ и $\angle QSP = \pi/2$. Тогда $PQ = \frac{SP}{ST}r = \sqrt{1 + (\frac{TP}{ST})^2}r = \sqrt{1 + (f'(x))^2}r$, то есть $g(x) = f(x) + \sqrt{1 + (f'(x))^2}r$, откуда имеем

$$V_{[a,b]}(g) = \int_a^b ((M + r) - g(x))\,dx = \\ = \int_a^b \left((M - f(x)) - \left(\sqrt{1 + (f'(x))^2}r - r\right)\right)dx \\ = V_{[a,b]}(f) - r\int_a^b \left(\sqrt{1 + (f'(x))^2} - 1\right)dx \quad (1.4)$$

Выполнив преобразования и введя обозначение $h(x) = 1/|f'(x)|$, получим

$$V_{[a,b]}(g) - V_{[a,b]}(f) = -r\int_a^b \left(h(x)\left(\sqrt{(h(x))^2 + 1} + h(x)\right)\right)^{-1} dx, \qquad (1.5)$$

откуда получаем выражение (1.3) для $V_{[a,b]}(f)'_r$. ■

Из леммы 1.2 получаем, что величина впадины при продвижении фронта убывает, так как значение интеграла из (1.3) положительно. Таким образом, при продвижения фронта в соответствии с геометрическим принципом Гюйгенса дефекты поверхности сглаживаются.

## 1.9 Выводы

Моделирование объектов со сложной геометрией связано с комплексом отдельных вычислительных задач, которые должны быть решены и оптимизированы для обеспечения эффективных расчетов. В число таких задач входит создание сеток, описывающих расчетные подобласти и границы между ними, обработка этих сеток, перестроение и корректировка описывающих границы сеток, обеспечение синхронизации расчетов в подобластях и на границах между ними, а также распараллеливание вычислений в расчетных подобластях и границах между ними на всех уровнях распараллеливания.

К основным уровням распараллеливания, рассмотренным в работе, относятся распараллеливание на распределенной памяти для организации вычислений на множестве вычислительных узлов в рамках одного суперкомпьютерного кластера, распараллеливание на общей памяти для организации вычислений на множестве ядер и потоков внутри одного вычислительного узла и распараллеливание на уровне отдельных инструкций для повышения эффективности вычислений внутри отдельно взятого ядра с использованием его аппаратных особенностей.

В главе представлен состав программного комплекса, реализующего математические методы и алгоритмы, применяемые для моделирования объектов со сложной геометрией с использованием высокопроизводительных вычислительных систем. В качестве такого программного комплекса выступает программный комплекс расчета процесса обледенения элементов авиационных силовых установок «Кристалл».

В главе рассмотрены ресурсоемкие вычислительные задачи, возникающие при моделировании объектов со сложной геометрией с использованием высокопроизводительных вычислительных систем. В эти задачи входят перестроение и коррекция поверхностной неструктурированной сетки, описывающей границы расчетных подобластей, а также выполнение параллельных вычислений на объемных блочно-структурированных и поверхностных неструктурированных сетках.

Исследование модели границы расчетной подобласти, применяемой при моделировании объектов со сложной геометрией, в виде замкнутой двусторонней поверхности с определенным направлением внешней нормали в каждой точке поверхности показало, что при эволюции этой границы в соответствии с геометрическим принципом Гюйгенса происходит сглаживание локальных дефектов (острых пиков и впадин). Указанное свойство сглаживания локальных дефектов границы при ее эволюции является важным, и его следует учитывать при разработке методов перестроения поверхностной неструктурированной сетки для повышения надежности перестроения.

## Глава 2. Перестроение поверхностной сетки

В главе рассматривается задача перестроения поверхностной неструктурированной сетки, которая разделяет области со сложной геометрией и которая должна изменяться по мере проведения расчетов (эволюционировать). Проблема перестроения сетки рассматривается в двух постановках: в двумерной постановке на плоскости – сетка представлена ломаной с ячейками-отрезками, в трехмерной постановке в пространстве – сетка состоит из треугольных ячеек.

Эволюция поверхностной сетки определяется изменением положения ее узлов, которые в свою очередь определяются через условия, накладываемые на изменение положения ее ячеек. Так, в задаче моделирования обледенения поверхностная сетка описывает поверхность обледеневающего тела и разделяет область льда и область окружающего пространства. При эволюции поверхностной сетки на ее ячейки накладывается следующее условие – заметаемая при движении ячейки площадь (в двумерном случае) или объем (в трехмерном случае) должен соответствовать количеству накопленного в ячейке льда [110, 264]. В задаче создания пристеночного слоя при моделировании течений для тел со сложной геометрией на движение ячеек накладываются условия параллельного смещения относительно исходного положения на одну и ту же фиксированную величину (ширину пристеночного слоя) [112].

Одной из проблем, связанных с эволюцией поверхностной сетки, является накапливание локальных дефектов, развитие которых приводит к ненадежному поведению программы и необходимости выполнять повторные запуски для получения результатов на протяженном времени моделирования. Такие перезапуски программы приводят к потере производительности при проведении расчетов. В различных приложениях проблема накапливания локальных дефектов сетки преодолевается с помощью дополнительных эвристических алгоритмов сглаживания сетки, как это описано в [113, 114], локальной корректировки для сглаживания отдельных острых углов и впадин с помощью вспомогательного зависящего от значения угла коэффициента, как приведено в [112], и другими искусственными методами.

В главе предлагается общий подход к реализации локального перестроения поверхностной неструктурированной сетки по методу биссектрис, позволяющий выполнить перестроение с требуемой точностью, а также обладающий свойством сглаживания локальных дефектов сетки. Рассматриваются известные методы перестроения, дополнительные аспекты повышения точности перестроения, различные виды сглаживания, выполняемые на поверхностных сетках. Предлагается новый метод перестроения поверхностной неструктурированной сетки, допускающий параллельную реализацию, приводятся аналитические формулы для нахождения новых положений узлов.

Также в главе исследуются вопросы пересечения расчетных сеток: самопересечение поверхностной неструктурированной сетки и пересечение поверхностной неструктурированной сетки с объемной декартовой сеткой.

Во время эволюции поверхностной сетки применение сглаживаний может отложить возникновение самопересечений, но полностью их исключить нельзя, поэтому возникает необходимость обработки этого глобального дефекта для поддержания сетки в корректном состоянии описания замкнутой двусторонней поверхности. Рассматриваются существующие методы удаления самопересечений поверхностной неструктурированной сетки. Предлагается метод удаления самопересечений поверхностной неструктурированной сетки с помощью дробления ячеек по всем точкам и отрезкам пересечения с другими ячейками, выполняемого в процессе обхода внешней поверхности, после чего все необойденные ячейки классифицируются как лишние и удаляются.

Другой затрагиваемый в главе вопрос связан с проведением расчетов в областях, для которых поверхностная сетка является границей. Так как геометрия поверхностной сетки может быть сложной и эволюционирующей, то поддержка согласованной объемной сетки является слишком ресурсозатратной. Рассматривается выполнение расчетов на сетке с фиксированной геометрией и сопряжение ее с рассматриваемой поверхностной сеткой с помощью метода погруженной границы. Для реализации этого метода рассматривается задача определения пересечения поверхностной неструктурированной сетки и объемной декартовой сетки.

## 2.1 Основные используемые геометрические примитивы

Приведем определения и соглашения касательно геометрических примитивов, которые будут использоваться в этой и последующих главах.

### 2.1.1 Линейные примитивы

При рассмотрении геометрических объектов наряду с понятием точки $P = (P_x, P_y) = (x_P, y_P)$ в двумерном случае и $P = (P_x, P_y, P_z) = (x_P, y_P, z_P)$ в трехмерном случае будем использовать понятие радиус-вектора $\overline{P}$, компонентами которого являются координаты этой точки. В дальнейшем будем считать эти термины взаимозаменяемыми.

Прямую, проходящую через точку $A$ и направленную вдоль вектора $\overline{D}$, которую будем обозначать $Line(A, \overline{D})$, будем определять как геометрическое место точек (ГМТ) $\overline{P}(\alpha)$ на плоскости или в пространстве, заданное следующим образом:

$$\overline{P}(\alpha) = \overline{A} + \alpha\overline{D},\ \alpha \in \mathbb{R}. \tag{2.1}$$

Если даны две точки $A$ и $B$, то через $\overline{AB}$ будем обозначать вектор, направленный из точки $A$ в точку $B$ ($\overline{AB} = \overline{B} - \overline{A}$).

Отрезок с концами $A$ и $B$, обозначаемый $Segm(A, B)$, определяется как ГМТ $\overline{P}(\alpha)$ на плоскости или в пространстве, заданное следующим образом:

$$\begin{cases} \overline{P}(\alpha) = \overline{A} + \alpha\overline{AB},\ \alpha \in \mathbb{R}, \\ 0 \le \alpha \le 1. \end{cases} \tag{2.2}$$

Треугольник с вершинами $A$, $B$, $C$, обозначаемый $Tri(A, B, C)$, будем определять как ГМТ $\overline{P}(\beta, \gamma)$ в пространстве, заданное следующим образом:

$$\begin{cases} \overline{P}(\beta, \gamma) = \overline{A} + \beta\overline{AB} + \gamma\overline{AC},\ \beta \in \mathbb{R},\ \gamma \in \mathbb{R}, \\ \beta \ge 0,\ \gamma \ge 0,\ \beta + \gamma \le 1. \end{cases} \tag{2.3}$$

### 2.1.2 Окрестности геометрических объектов

Шар с центром в точке $C$ и радиуса $R$, обозначаемый как $Ball(C, R)$, будем определять как ГМТ $\overline{P}$, удовлетворяющих условию

$$|\overline{P} - \overline{C}| \leq R. \tag{2.4}$$

Сферу с центром в точке $C$ и радиуса $R$, представляющую собой поверхность шара $Ball(C, R)$, будем обозначать $Sphere(C, R)$.

**Определение 2.1.** *Пусть дано некоторое ГМТ $G$, на точках $P \in G$ которого определена неотрицательная функция $R : G \to \mathbb{R}_{\geq 0}$. Тогда окрестностью множества точек $G$, заданной указанной функцией $R(P)$, будем называть ГМТ, определяемое следующим образом:*

$$O_R(G) = \{P : \exists C \in G \implies P \in Ball(C, R(C))\}. \tag{2.5}$$

Таким образом, окрестность множества $G$ это объединение множества шаров $Ball(C, R(C))$, с центрами $C \in G$ и определенными в этих точках радиусами.

### 2.1.3 Поверхностная неструктурированная сетка

Двумерной поверхностной неструктурированной сеткой будем называть ломаную на плоскости, не содержащую самопересечений и состоящую из $n$ ячеек-отрезков $F_i$ длиной $l_i$ $(0 \leq i < n)$. Ячейка $F_i$ имеет инцидентные узлы $N_i$, $N_{i+1}$ $(F_i = Segm(N_i, N_{i+1}))$, для нее определена внешняя единичная нормаль $\overline{n}_i^F$. Если $N_0 = N_n$, то сетка является замкнутой. Для узла $N_i$ определена единичная нормаль $\overline{n}_i^N = \frac{\overline{n}_{i-1}^F + \overline{n}_i^F}{|\overline{n}_{i-1}^F + \overline{n}_i^F|}$, лежащая на биссектрисе телесного угла $2\phi_i$ между ячейками $F_{i-1}$ и $F_i$ (рис. 2.1 слева).

Теперь рассмотрим структуру и требования, предъявляемые к поверхностной неструктурированной сетке в трехмерном случае. Элементами сетки являются узлы $N \in \mathscr{N}$, ребра $E \in \mathscr{E}$ и ячейки $F \in \mathscr{F}$, имеющие форму

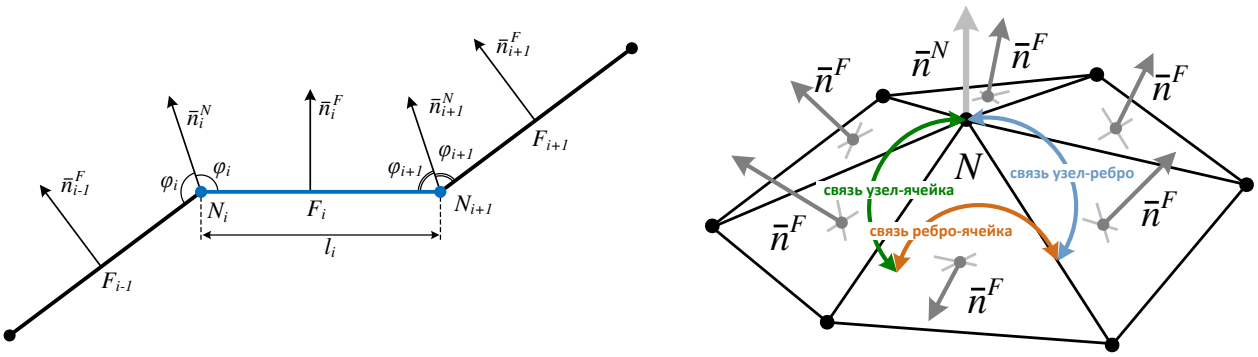


Рис. 2.1. Двумерная (слева) и трехмерная (справа) неструктурированная поверхностная сетка

треугольников. Каждый элемент сетки связан со всеми своими инцидентными элементами: так связаны между собой инцидентные узлы и ребра, узлы и ячейки, ребра и ячейки (рис. 2.1 справа). Множества инцидентных узлов ребра $E$ или ячейки $F$ будем обозначать $\mathscr{N}(E)$ и $\mathscr{N}(F)$ соответственно, множества инцидентных ребер узла $N$ или ячейки $F$ будем обозначать $\mathscr{E}(N)$ и $\mathscr{E}(F)$ соответственно, множества инцидентных ячеек узла $N$ или ребра $E$ будем обозначать $\mathscr{F}(N)$ и $\mathscr{F}(E)$ соответственно.

К сетке предъявляются следующие требования. Во-первых, сетка должна быть целостной, то есть каждое ребро имеет ровно два инцидентных узла, отсутствуют изолированные и висячие узлы, а также отсутствуют ребра без инцидентных ячеек. Во-вторых, как уже упоминалось, все ячейки должны представлять собой треугольники (это гарантирует, что ячейка является плоской, так как четыре и более произвольных узла могут не лежать в одной плоскости). В третьих, каждое ребро имеет ровно две инцидентные ячейки, в этом случае сетка является замкнутой. В отдельных случаях могут рассматриваться незамкнутые сетки, содержащие ребра, инцидентные только одной ячейке (такие ребра являются границей сетки).

$$\begin{cases} \forall N \in \mathscr{N} \implies |\mathscr{E}(N)| > 2, |\mathscr{F}(N)| > 2, \\ \forall E \in \mathscr{E} \implies |\mathscr{N}(E)| = 2, |\mathscr{F}(E)| = 2, \\ \forall F \in \mathscr{F} \implies |\mathscr{N}(F)| = 3, |\mathscr{E}(F)| = 3. \end{cases} \tag{2.6}$$

В качестве дополнения также будем требовать, чтобы сетка представляла собой двустороннюю поверхность, то есть для каждой ячейки $F$ однозначно определена нормаль к поверхности $\overline{n}_F^F$. Также никакие два узла сетки не совпадают и отсутствуют ячейки с нулевой площадью (так как это сделает невозможным вычислений нормалей). Для узла сетки $N$ будем рассматривать понятие единичной нормали к поверхности и определять эту нормаль как

$$\overline{n}_N^N = \frac{\sum_{F \in \mathscr{F}(N)} \overline{n}_F^F}{\left| \sum_{F \in \mathscr{F}(N)} \overline{n}_F^F \right|}. \tag{2.7}$$

### 2.1.4 Заметаемые площади и объемы при движении ячеек

В разделе приводятся формулы заметаемых площадей и объемов при движении отдельных ячеек поверхностной неструктурированной сетки.

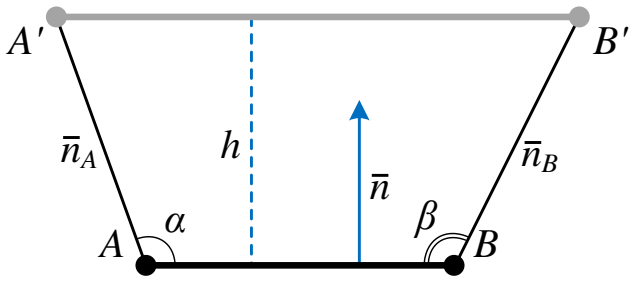


Рис. 2.2. Заметаемая площадь при параллельном смещении отрезка

Рассмотрим задачу о заметаемой площади при параллельном смещении отрезка. Пусть дан отрезок $Segm(A, B)$ с направлением единичной нормали $\overline{n}$, а также с направлениями движения его концов $\overline{n}_A$, $\overline{n}_B$. На расстоянии $h$ от прямой $AB$ проходит параллельная прямая, пересекающая направления $\overline{n}_A$, $\overline{n}_B$ в точках $A'$ и $B'$ соответственно. Требуется найти зависимость площади трапеции $S_{AA'B'B}$ от расстояния между прямыми $h$ (рис. 2.2).

Рассмотрим движение точки $A$. Выразим новое положение этой точки $\overline{A}' = \overline{A} + t\overline{n}_A$. Так как расстояние между прямыми $AB$ и $A'B'$ равно $h$, то $(t\overline{n}_A, \overline{n}) = h$, откуда можно выразить $t = \frac{h}{(\overline{n}_A, \overline{n})}$.

Вычисляя положения точек $A'$ и $B'$, можно получить выражение для площади трапеции $AA'B'B$:

$$\begin{aligned}
&\overline{A}' = \overline{A} + h\overline{u}_A,\ \overline{u}_A = \frac{\overline{n}_A}{(\overline{n}_A, \overline{n})},\\
&\overline{B}' = \overline{B} + h\overline{u}_B,\ \overline{u}_B = \frac{\overline{n}_B}{(\overline{n}_B, \overline{n})},\\
&S_{AA'B'B}(h) = \frac{1}{2}\left(|\overline{A} - \overline{B}| + |\overline{A}' - \overline{B}'|\right) h.
\end{aligned} \tag{2.8}$$

Отметим один частный случай, в котором трапеция $AA'B'B$ является равнобокой с углом $\alpha$ при основании $AB$. Тогда при $AB = l$ имеем $A'B' = l - 2h\operatorname{ctg}\alpha$, и

$$S_{AA'B'B}(h) = (l - h\operatorname{ctg}\alpha)h, \tag{2.9}$$

откуда $h$ может быть выражена с помощью решения квадратного уравнения

$$h(S, l, \alpha) = \frac{l - \sqrt{l^2 - 4S\operatorname{ctg}\alpha}}{2\operatorname{ctg}\alpha}. \tag{2.10}$$

Рассмотрим задачу о заметаемой площади при произвольном смещении отрезка Рассматривается ячейка, представленная на плоскости отрезком $AB$ длины $l$. При перемещении точек $A$ и $B$ в новые точки $A'$ и $B'$ соответственно образуется четырехугольник $AA'B'B$. Требуется найти его площадь, выраженную явно через параметры $a = AA'$ и $b = BB'$ и углы при вершинах $A$ и $B$ (рис. 2.3).

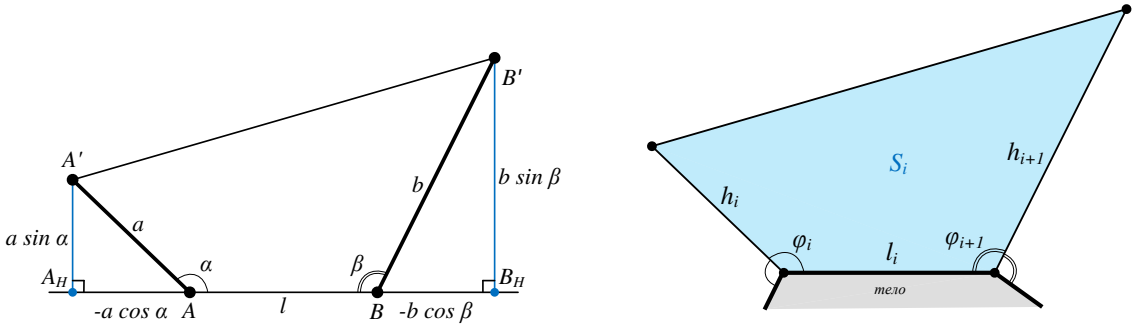


Рис. 2.3. Вычисление заметаемой площади через смещения узлов ячейки

**Лемма 2.1.** *Пусть дан четырехугольник $AA'B'B$, в котором $AA' = a$, $BB' = b$, $\angle A'AB = \alpha$, $\angle B'BA = \beta$. Тогда площадь этого четырехугольника выражается формулой*

$$S_{AA'B'B} = \frac{1}{2}\left(l(a\sin\alpha + b\sin\beta) - ab\sin(\alpha+\beta)\right). \tag{2.11}$$

Опустим перпендикуляры из точек $A'$ и $B'$ на прямую $AB$. Их основаниями будут точки $A_H$ и $B_H$ соответственно. Если $\alpha \geq \frac{\pi}{2}$, то перпендикуляр $A'A_H$ проходит снаружи четырехугольника $AA'B'B$, в противном случае – внутри. Аналогично, если $\beta \geq \frac{\pi}{2}$, то перпендикуляр $B'B_H$ проходит снаружи четырехугольника $AA'B'B$, в противном случае – внутри. В любом случае искомая площадь может быть представлена в следующем виде:

$$S_{AA'B'B} = S_{A_HA'B'B_H} - \operatorname{sign}\left(\alpha - \frac{\pi}{2}\right) S_{AA'A_H} - \operatorname{sign}\left(\beta - \frac{\pi}{2}\right) S_{BB'B_H}. \tag{2.12}$$

Эту площадь можно выразить явно через значения углов $\alpha$ и $\beta$:

$$S_{AA'B'B} = \frac{1}{2}(l - a\cos\alpha - b\cos\beta)(a\sin\alpha + b\sin\beta) + \frac{1}{2}a^2\sin\alpha\cos\alpha + \frac{1}{2}b^2\sin\beta\cos\beta. \tag{2.13}$$

После раскрытия скобок и преобразований получим (2.11). ■

Отметим, что если $\alpha+\beta \geq \pi$ то $S_{AA'B'B}$ всегда положительна и возрастает с ростом $a$ или $b$. Это означает, что лучи $AA'$ и $BB'$ не пересекаются (этот случай изображен на рис. 2.3).

Рассмотрим случай $\alpha + \beta < \pi$. Запишем отдельно условия неубывания $S_{AA'B'B}$ при росте $a$ и $b$:

$$\begin{aligned} \frac{\partial S_{AA'B'B}}{\partial a} &= \frac{1}{2}(l\sin\alpha - b\sin(\alpha+\beta)) \geq 0, \\ \frac{\partial S_{AA'B'B}}{\partial b} &= \frac{1}{2}(l\sin\beta - a\sin(\alpha+\beta)) \geq 0, \end{aligned} \tag{2.14}$$

или

$$a \le \frac{l \sin \beta}{\sin(\alpha + \beta)},$$
$$b \le \frac{l \sin \alpha}{\sin(\alpha + \beta)}. \tag{2.15}$$

Если выполняются оба условия из (2.15), то четырехугольник $AA'B'B$ продолжает оставаться выпуклым. Если выполняется только одно условие из (2.15), то четырехугольник является невыпуклым. Если не выполняется ни одно из условий (2.15), то произошло самопересечение сетки, то есть оба отрезка $AA'$ и $BB'$ вышли за пределы треугольника $ABO$, где $O$ – точка пересечения лучей $AA'$ и $BB'$. Случаи самопересечения должны быть обработаны отдельно для возможности продолжения работы с расчетной сеткой. В этом разделе вопросы самопересечения расчетной сетки не рассматриваются, они будут рассмотрены далее в трехмерной постановке.

Рассмотрим задачу о заметаемом объеме при параллельном смещении треугольника. Эта задача аналогична задаче о параллельном смещении отрезка. Рассмотрим треугольник $Tri(A, B, C)$. Пусть известно направление единичной нормали этого треугольника $\overline{n}$ и направления движения его вершин $\overline{n}_A$, $\overline{n}_B$, $\overline{n}_C$. Пусть на расстоянии $h$ от плоскости $ABC$ проходит параллельная плоскость, пересекающая направления $\overline{n}_A$, $\overline{n}_B$, $\overline{n}_C$ в точках $A'$, $B'$, $C'$. Требуется найти зависимость объема полученной фигуры $V_{ABCA'B'C'}$ от расстояния между плоскостями $h$ (рис. 2.4).

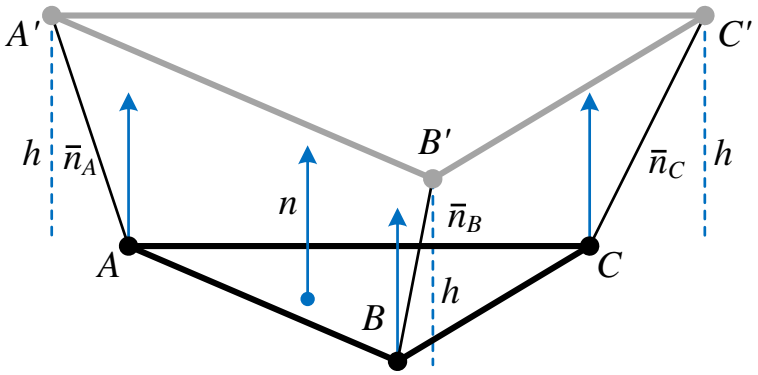


Рис. 2.4. Заметаемый объем при параллельном смещении треугольника

Вычисляя положения точек $A'$, $B'$, $C'$ таким же образом, как это было сделано для трапеции в двумерном случае, можно получить выражение для объема фигуры $ABCA'B'C'$:

$$\begin{aligned}
&\overline{A}' = \overline{A} + h\overline{u}_A,\ \overline{u}_A = \frac{\overline{n}_A}{(\overline{n}_A, \overline{n})},\\
&\overline{B}' = \overline{B} + h\overline{u}_B,\ \overline{u}_B = \frac{\overline{n}_B}{(\overline{n}_B, \overline{n})},\\
&\overline{C}' = \overline{C} + h\overline{u}_C,\ \overline{u}_C = \frac{\overline{n}_C}{(\overline{n}_C, \overline{n})},\\
&S_{ABC} = \frac{1}{2}\|(\overline{B} - \overline{A}) \times (\overline{C} - \overline{A})\|,\\
&S_{A'B'C'} = \frac{1}{2}\|(\overline{B'} - \overline{A'}) \times (\overline{C'} - \overline{A'})\|,\\
&V_{ABCA'B'C'}(h) = \frac{1}{3}\left(S_{ABC} + \sqrt{S_{ABC}S_{A'B'C'}} + S_{A'B'C'}\right)h.
\end{aligned} \tag{2.16}$$

Полученное в (2.16) выражение $V_{ABCA'B'C'}(h)$ является кубической функцией от $h$, поэтому $h$ может быть выражено через $V_{ABCA'B'C'}$ путем решения кубического уравнения.

## 2.2 Постановка задачи перестроения сетки

В качестве условия, накладываемого на изменение положения ячеек поверхностной неструктурированной сетки, будем использовать соответствие заметаемой площади или объема при движении ячеек целевым величинам. В двумерном случае целевую площадь для ячейки $F_i$ будем обозначать $T_i$, в трехмерном – целевой объем для ячейки $F$ будем обозначать $T_F$.

**Определение 2.2.** *Целевой площадью для ячейки $F_i$ двумерной поверхностной неструктурированной сетки будем называть площадь, заметаемую ячейкой при смещении на величину $H_i$, то есть $T_i = l_i H_i$. Величина $H_i$ называется величиной смещения ячейки.*

**Определение 2.3.** *Целевым объемом для ячейки $F$ трехмерной поверхностной неструктурированной сетки будем называть объем, заметаемый ячейкой при смещении на величину $H_F$, то есть $T_F = S_F H_F$. Величина $H_F$ называется величиной смещения ячейки.*

Если просто сдвинуть каждую ячейку в направлении своей нормали, то сетка потеряет целостность, поэтому мы можем осуществлять движение только узлов. При перестроении поверхностной неструктурированной сетки требуется определить новые положения узлов.

В двумерном случае требуется определить такие новые положения узлов $N'_i$, при которых площадь $S_i = S_{N_i N'_i N'_{i+1} N_{i+1}}$, заметаемая ячейкой $F_i$, как можно меньше отличается от целевой площади $T_i$. Задача рассматривается при фиксированных направлениях смещения узлов вдоль их нормалей $\overline{n}_i^N$ [267], то есть требуется определить только величины смещения узлов $h_i$. Для оценки отклонения заметаемой площади от целевой используются обозначения: $\Delta_i = S_i - T_i$ – абсолютное отклонение от целевой площади, $\delta_i = \frac{\Delta_i}{T_i}$ – относительное отклонение от целевой площади.

В трехмерном случае требуется определить такие новые положения узлов $N'$, при которых объем $V_F = V_{ABCA'B'C'}$, заметаемая ячейкой $F = ABC$, как можно меньше отличается от целевого объема $T_F$. Задача также рассматривается при фиксированных направлениях смещения узлов вдоль их нормалей $\overline{n}_N^N$, то есть требуется определить только величины смещения узлов $h_N$. Для оценки отклонения заметаемого объема от целевого используются обозначения: $\Delta_F = V_F - T_F$ – абсолютное отклонение от целевого объема, $\delta_F = \frac{\Delta_F}{T_F}$ – относительное отклонение от целевого объема.

## 2.3 Смещение узлов в двумерной постановке

Рассмотрим геометрическую задачу о перестроении поверхности в двумерном случае в общем виде [268]. Для решения поставленной задачи сначала требуется вычислить заметаемую площадь для каждой отдельной ячейки. Затем, просуммировав заметаемые площади для всех ячеек, можно получить площадь, заметаемую при движении сетки целиком.

Метод градиентного спуска является одним из наиболее простых методов оптимизации для нахождения локального минимума функции ($f(x)$). При условии, что в любой точке функции можно вычислить ее градиент, то начи-

ная с некоторого начального приближения $x_0$, находящегося в окрестности локального минимума, строится итерационная последовательность [269]:

$$x_{k+1} = x_k - \gamma_k \nabla f(x_k), \tag{2.17}$$

где $\gamma_k \geq 0$ задает длину шага и, соответственно, скорость градиентного спуска.

Метод градиентного спуска находит свое основное применение в задаче поиска минимума или максимума функции. Направление антиградиента является направлением наискорейшего убывания функции. Основная проблема метода заключается в выборе шага $\gamma$. При слишком больших значениях шага существует вероятность миновать искомый минимум функции. К тому же, метод не гарантирует нахождение глобальных экстремумов.

Рассмотрим решение задачи определения величин смещения узлов поверхностной расчетной сетки при перестроении методом градиентного спуска. Неизвестными параметрами являются величины сдвигов узлов сетки $h_i$ для $0 \leq i \leq n$. Опираясь на решение локальной задачи об определении заметаемой площади из леммы 2.1, можно записать заметаемую площадь при движении отдельной ячейки (рис. 2.3 справа):

$$S_i = \frac{1}{2}\left(l_i(h_i \sin\phi_i + h_{i+1}\sin\phi_{i+1}) - h_i h_{i+1}\sin(\phi_i + \phi_{i+1})\right). \tag{2.18}$$

Для нахождения решения задачи перестроения поверхностной сетки в соответствии с методом наименьших квадратов будем искать минимум функции

$$D(\overline{h}) = D(h_0, \ldots, h_n) = \sum_{i=0}^{n-1} \Delta_i^2. \tag{2.19}$$

Для нахождения градиента требуется вычислить частные производные $\frac{\partial D}{\partial h_i}$. Эти частные производные можно записать в явном виде:

$$\frac{\partial D}{\partial h_i} = \frac{\partial(\Delta_{i-1})^2}{\partial h_i} + \frac{\partial(\Delta_i)^2}{\partial h_i}, \tag{2.20}$$

где

$$\begin{cases} \frac{\partial(\Delta_{i-1})^2}{\partial h_i} = \Delta_{i-1}(l_{i-1}\sin\phi_i - h_{i-1}\sin(\phi_{i-1}+\phi_i)), \\ \frac{\partial(\Delta_i)^2}{\partial h_i} = \Delta_i(l_i\sin\phi_i - h_{i+1}\sin(\phi_i+\phi_{i+1})). \end{cases} \tag{2.21}$$

Также при осуществлении метода градиентного спуска требуется следить за соблюдением дополнительных условий, которые накладываются на неизвестные $h_i$. Например, очевидным условием является выполнение соотношения $h_i \geq 0$, что запрещает движение узлов сетки в отрицательном направлении. Также можно использовать более строгие условия $\min(H_{i-1}, H_i) \leq h_i \leq \max(H_{i-1}, H_i)$, которые не позволят величинам смещения узлов сетки выходить за пределы смещений инцидентных им ячеек сетки.

Решение задачи о перестроении сетки методом градиентного спуска требовательно к вычислительным ресурсам при увеличении размера сетки. К тому же качество решения зачастую оказывается неудовлетворительным при попадании в локальные минимумы. Метод может быть расширен на трехмерный случай, однако в трехмерной постановке он еще более требователен к вычислительным ресурсам. Поэтому требуется рассмотрение приближенных методов перестроения поверхностной неструктурированной сетки.

## 2.4 Приближенные методы перестроения

### 2.4.1 Методы перестроения в двумерном случае

Сначала рассмотрим приближенные методы перестроения поверхностной неструктурированной сетки в двумерном случае. Для этого используется представление целевой площади в виде примитивных геометрических фигур.

#### 2.4.1.1 Метод прямоугольников

В качестве первого метода перестроения рассмотрим приближение, при котором целевая площадь для $i$-ой ячейки представлена прямоугольником со сторонами $l_i$ и $H_i$. В качестве величины смещения узла берется среднее арифметическое двух высот инцидентных ячеек $h_i = \frac{H_{i-1}+H_i}{2}$ (рис. 2.5 слева). Этот

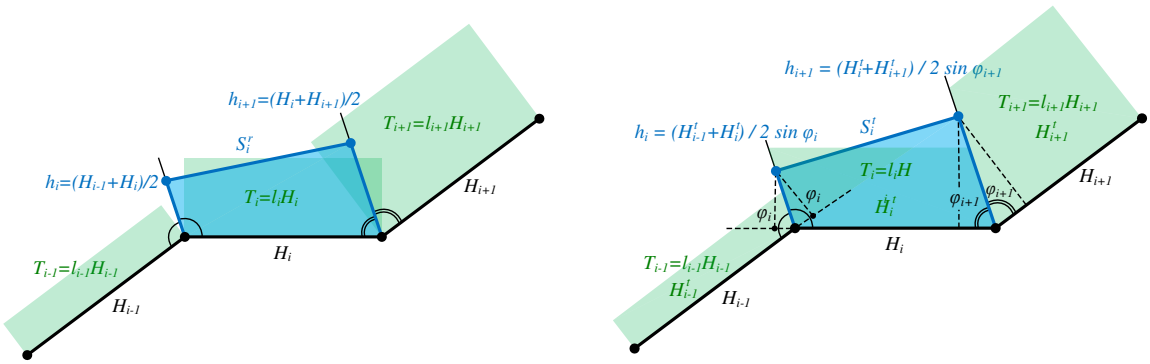


Рис. 2.5. Перестроение двумерной поверхностной сетки методом прямоугольников (слева) и трапеций (справа)

метод перестроения будем называть методом прямоугольников (в литературе он также называется алгебраическим методом [110]). Заметаемую $i$-ой ячейкой площадь при использовании метода прямоугольников будем обозначать $S_i^r$, а также обозначим абсолютное и относительное отклонения от целевой площади (погрешность перестроения) через $\Delta_i^r = S_i^r - T_i$ и $\delta_i^r = \frac{\Delta_i^r}{T_i}$ соответственно.

#### 2.4.1.2 Метод трапеций

Если в методе прямоугольников применить итерационное уточнение [110] (при стремящемся к бесконечности количестве итераций) без изменения направления нормалей узлов, то получим другой метод, который будем называть методом трапеций. В методе трапеций целевая площадь $i$-ой ячейки представляется трапецией с площадью $T_i$. Боковые стороны этой трапеции лежат на направлениях движения двух узлов, инцидентных рассматриваемой ячейке. Высота трапеции $H_i^t$ находится из (2.8) с помощью решения квадратного уравнения. После построения трапеций для всех ячеек сетки у каждого узла появляется две новые потенциальные позиции для сдвига (образованные ячейкой слева и ячейкой справа). В качестве финальной новой позиции выбирается их среднее значение (рис. 2.5 справа). Таким образом, величина смещения узла определяется как $h_i = \frac{H_{i-1}^t+H_i^t}{2\sin\phi_i}$. Заметаемую $i$-ой ячейкой площадь при использовании метода трапеций будем обозначать $S_i^t$, а абсолютное и относительное отклонения от целевой площади – $\Delta_i^t = S_i^t - T_i$ и $\delta_i^t = \frac{\Delta_i^t}{T_i}$ соответственно.

#### 2.4.1.3 Метод окрестностей в двумерном случае

Предлагается новый метод перестроения поверхностной неструктурированной сетки, который будем называть методом окрестностей (рис. 2.6).

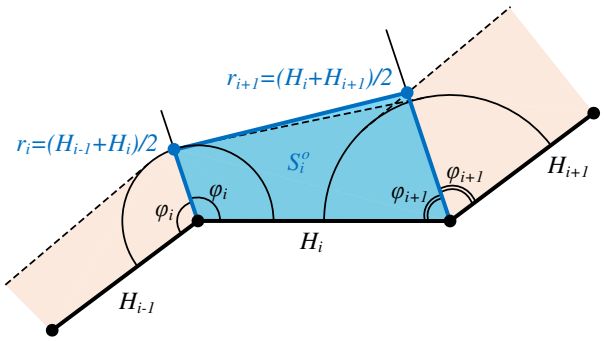


Рис. 2.6. Перестроение поверхности методом окрестностей

Изначально входными данными перестроения сетки является величина смещения ячейки $F_i$ в направлении ее внешней нормали. В методе окрестностей перестроения поверхностной сетки в двумерном случае в качестве нового положения узла будем брать пересечение направления смещения узла и границы окрестности двух инцидентных этому узлу ячеек. Для построения окрестности ячейки необходимо определить функцию радиуса для всех точек ячейки. Для каждого узла $N_i$ сетки определим радиус, равный $R_i = \frac{H_{i-1}+H_i}{2}$. Также будем считать, что внутри ячейки радиус шара, построенного на ее внутренних точках меняется линейно. Заметаемую $i$-ой ячейкой площадь при использовании метода окрестностей будем обозначать $S_i^o$, а абсолютное и относительное отклонения от целевой площади – $\Delta_i^o = S_i^o - T_i$ и $\delta_i^o = \frac{\Delta_i^o}{T_i}$ соответственно. Аналитические формулы для определения новых положений узлов для перестроения сетки с помощью метода окрестностей будут приведены далее в общем виде в трехмерной постановке.

### 2.4.2 Методы перестроения в трехмерном случае

Приведенные методы прямоугольников и трапеций перестроения сетки в двумерном случае могут быть обобщены для трехмерного случая. Для этого в трехмерном случае у нас имеются заданные значения целевого объема $T_F$

для каждой ячейки. Он вычислен в результате физических расчетов, в случае задачи моделирования обледенения $T_F$ соответствует объему накопленного в ячейке $F$ льда [270]. Для узла сетки $N$ требуется найти его новое положение в пространстве $N'$, чтобы для ячеек $F = ABC$ объем пространства, ограниченный фигурой $ABCA'B'C'$, соответствовал целевому объему.

Следует отметить, что поставленная задача может не иметь точного решения, и в этом случае следует стремиться к минимизации по модулю ошибки по объему $\Delta_F = V_{ABCA'B'C'} - T_F$.

Задачу определения новых положений узлов расчетной сетки можно разделить на две задачи: определение направлений смещения узлов и определение величин смещения. Классические методы перестроения выполняются в предположении, что направление смещения узла $N$ совпадает с нормалью, проведенной из этого узла $\overline{n}_N^N$. Таким образом, необходимо лишь определить величину смещения.

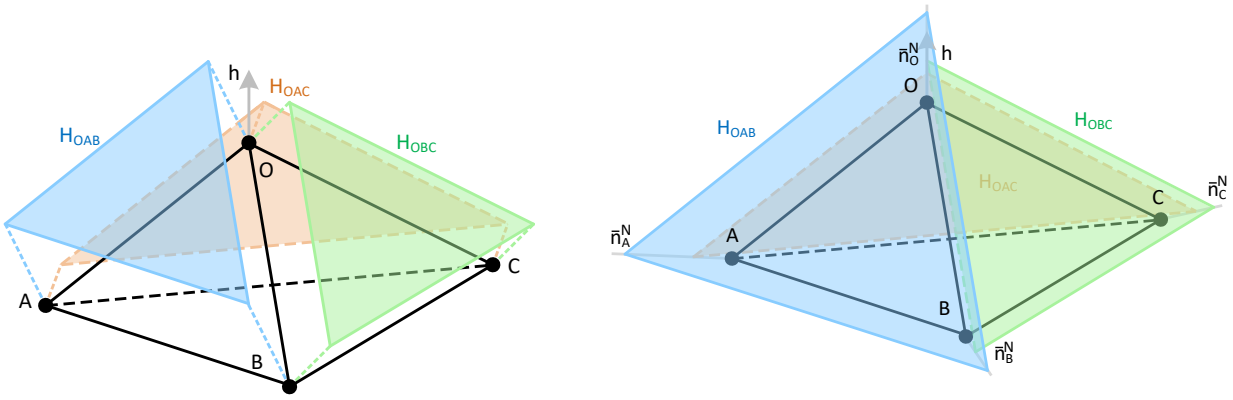


Рис. 2.7. Перестроение поверхностной неструктурированной сетки в трехмерном случае с помощью метода призм (слева) и пирамид (справа)

### 2.4.2.1 Метод призм

В качестве первого метода перестроения поверхностной неструктурированной сетки в трехмерном пространстве рассмотрим метод призм, иллюстрация которого приведена на рис. 2.7 слева (в двумерной постановке аналогом этого метода является метод прямоугольников). В этом методе входными данными является целевой объем в каждой ячейке сетки ($T_F$). На первым шаге

в каждой ячейке ищется величина смещения ячейки в предположении, что заметаемый объем имеет форму призмы, и ячейка является основанием этой призмы. Тогда величина смещения ячейки равняется $H_F = \frac{V_F}{S_F}$, где $S_F$ – площадь ячейки. После чего величина смещения каждого узла вычисляется как среднее арифметическое величин смещения всех инцидентных ячеек:

$$h_N = \frac{1}{|\mathscr{F}(N)|} \sum_{F \in \mathscr{F}(N)} H_F. \tag{2.22}$$

#### 2.4.2.2 Метод пирамид

Второй метод перестроения будем называть методом пирамид, он проиллюстрирован на рис. 2.7 справа (в двумерной постановке аналогом является метод трапеций). Входными данными также является целевой объем в каждой ячейке сетки ($T_F$). В отличие от метода призм, целевой объем представляется не призмой, а призматоидом, основанием которого является ячейка, а боковые ребра направлены вдоль нормалей узлов (будем условно называть эту фигуру усеченной пирамидой, чтобы не путать с методом призм, хотя в общем случае она усеченной пирамидой не является – прямые, содержащие нормали трех инцидентных этой ячейке узлов, не обязаны пересекаться в одной точке). Высота этой фигуры ищется из соотношений (2.16) с помощью решения кубического уравнения (решением является наименьший неотрицательный корень этого уравнения). Тогда как узлы ячейки являются точками первого основания построенной пирамиды, точки второго основания представляют собой новые положения узлов, вычисленные относительно рассматриваемой ячейки. Таким образом, у каждого узла сетки вычисляется несколько новых положений (каждое из которых вычислено относительно своей инцидентной ячейки). Для двумерного случая получается ровно два таких новых положения (так как в двумерном случае каждый узел имеет ровно две инцидентные ячейки), для трехмерного случае таких точек более двух ($|\mathscr{F}(N)|$ штук). Для выбора единственного нового положения узла сетки берется среднее значение из всех положений, вычисленных относительно инцидентных ячеек.

### 2.4.3 Дополнительные аспекты перестроения сетки

Перед тем, как перейти к формулировке метода окрестностей перестроения поверхностной неструктурированной сетки в трехмерном случае рассмотрим дополнительные методы, применяемые при перестроении сетки для повышения точности (уменьшения погрешности) и качества получаемого решения.

#### 2.4.3.1 Многослойное перестроение

Вне зависимости от используемого метода перестроения значительно повысить точность перестроения можно с помощью многослойного подхода [110].

В этом случае вместо однократного перестроения сетки по целевому объему в каждой ячейке ($T_F$), выбирается фиксированное количество шагов перестроения $k$, и процедура выполняется $k$ раз подряд с использованием целевого объема $\frac{T_F}{k}$. Точность повышается из-за того, что после каждого шага перестроения нормали в узлах сетки меняют свое направление, и общий заметаемый ячейкой объем становится более криволинейным, лучше учитывает геометрию сетки и точнее соответствует исходному значению $T_F$ (рис. 2.8).

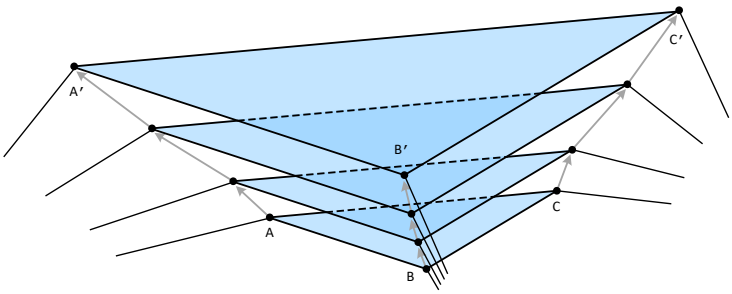


Рис. 2.8. Многослойное перестроение сетки

Для многослойного метода перестроения может быть применена коррекция с учетом фактически заметенного объема движущейся ячейкой при перестроении поверхности. Пусть по-прежнему используется многослойный подход с $k$ шагами перестроения (с номерами шагов $i \in [0, k-1]$). Кроме целевого объема в ячейке $T_F$ будем также сохранять фактически построенный объем на предыдущих шагах перестроения в переменной $T'_F$, которая перед перестроением равна нулю. Тогда на $i$-ом шаге перестроения используется целевой объем

$\frac{T_F - T'_F}{k-i}$, а переменная $T'_F$ увеличивается на объем фактически заметенного объема на этом шаге ($V_{ABCA'B'C}$). При использовании коррекции в многослойном методе перестроения общая точность совпадает с точностью перестроения на последнем шаге с номером $k-1$. Заметим, что при использовании коррекции перестроение может закончить работу досрочно при условии $T_F \leq T'_F$ после некоторого шага перестроения.

#### 2.4.3.2 Перестроение со сглаживаниями

В работах [113,114] приведено описание итерационного алгоритма эволюции поверхностной сетки, в котором на каждой итерации используются различные виды сглаживания. Будем называть его методом перестроения Tong. В нем используется ряд улучшений по сравнению с классическими методами.

Многослойный подход, реализованный в этом методе, не использует константное количество шагов – величина наращиваемого объема на каждом шаге алгоритма рассчитывается исходя из максимально допустимой доли наращивания объема, после превышения которого возможно развитие численной нестабильности в эволюции поверхности. То есть при перестроении расчетной сетки на текущем шаге вместо полного целевого объема в ячейках $T_F$ используется только его часть $\alpha T_F$ $(0 < \alpha \leq 1)$, не приводящая к аварийному завершению перестроения (после этого перестроение продолжается с оставшимся неиспользованным объемом). Наиболее очевидный случай численной нестабильности возникает, когда проекции нормалей узлов на общую инцидентную ячейку пересекаются, в этом случае слишком большой шаг по времени приведет к самопересечению сетки. Чтобы идентифицировать ячейки, которые будут демонстрировать подобное поведение на текущем временном шаге, предполагается, что объем, образованный путем вытягивания треугольной ячейки с использованием параллельной плоскости смещения, образует призматоид, объем которого определяется кубической функцией величины смещения ячейки $H$:

$$V_F(H) = aH + bH^2 + cH^3, \tag{2.23}$$

где константы $a$, $b$, $c$ определяются позициями узлов, их нормалей и нормалью конкретной ячейки $F$ (вычисление этих коэффициентов приведено в (2.16)).

Рассмотрим корни квадратного уравнения, которое получается в результате дифференцирования уравнения (2.23). Если корни являются положительными вещественными значениями, то наименьший положительный корень определяет высоту, на которой достигается максимальный объем для рассматриваемой ячейки $F$, не приводящий к возникновению численной нестабильности, который обозначается как $V_F^{max}$, иначе функция $V_F(H)$ монотонно возрастает и ограничение на шаг по времени в ячейке $F$ не требуется. Исходя из этого, можно вычислить максимальную долю $\alpha$, которая требуется для обеспечения устойчивого перестроения. В дополнение к этому пределу размера шага вводится предел стабильности $\alpha_{jiao}$, который основан на том, как изменяются направления нормалей по мере эволюции поверхности [271]. Тогда, допустимая доля для $i$-й ячейки определяется как

$$\alpha_F = \begin{cases} \min\left(s\frac{V_F^{max}}{T_F}, \alpha_{jiao}, 1\right), & \text{если } V_F^{max} \neq \infty, \\ \alpha_{jiao}, & \text{если } V_F^{max} = \infty, \end{cases} \tag{2.24}$$

где $s$ $(0 < s < 1)$ – эмпирически определяемый коэффициент, $T_F$ – целевой объем приращения объема для ячейки $F$. Тогда объем, наращиваемый для текущего шага, равен $\alpha T_F$, где $\alpha$ представляет собой глобальное минимальное значение для всех ячеек.

Другой важной особенностью алгоритма является введение первичного и нулевого пространств [272]. Если эволюционное движение узлов сетки происходит в первичном пространстве, то их перемещение в нулевом пространстве будет сохранять заметаемый объем, благодаря чему мы можем проводить сглаживание поверхности сетки с сохранением объема.

В алгоритме используется несколько видов сглаживаний. Первое сглаживание – сглаживание нормалей в узлах и ячейках сетки. Перемещение узлов при наращивании льда происходит только по их нормалям. По мере эволюции, на поверхности может усиливаться шум – если его не контролировать,

может возникнуть ситуация, когда двугранный угол между гранями станет слишком малым и ограничит максимальную долю шага перестроения $\alpha$. Для уменьшения поверхностного шума, перед наращиванием целевого объема применяется локальное сглаживание, регулирующее направление смещения узлов в проблемных областях, чтобы оно более точно совпадало с направлениями его соседей. Этот метод может улучшить гладкость поверхности.

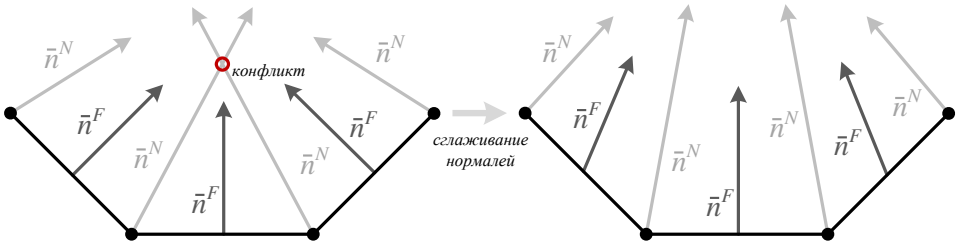


Рис. 2.9. Сглаживание нормалей

Основная цель сглаживания нормалей – вытолкнуть точки из вогнутых областей, где нормали могут локально сходиться. Сглаживание нормалей достигается с помощью серий взвешенных средних, которые предназначены для придания веса нормалям, генерируемым проблемными областями. То есть применяется ряд итераций, во время которых направления смещения ячеек определяются через направления смешения узлов и наоборот. На рис. 2.9 проиллюстрировано, как сглаживание нормалей помогает предотвратить возникновение конфликтов (в рассматриваемом случае это сворачивание ячейки) и потенциально увеличивает величину $\alpha$.

Второе сглаживание – сглаживание высот. После вычисления доли временного шага и объема, наращиваемого для текущего шага перестроения, для эволюции поверхности необходимо определить поле высот, которое будет соответствовать этому объему, чтобы по нему определить смещения узлов сетки. Решение уравнения $V_F(H) = \alpha T_F$ обеспечивает поле начальных высот, которое используется для движения сетки. Цель дополнительного шага сглаживания высоты состоит в том, чтобы отфильтровать высокочастотный шум в поле высот за счет уменьшения разницы высот между соседними ячейками.

Как правило, высоты двух треугольных ячеек, имеющих общее ребро, не будут равными. На этом шаге используется сглаживание высот с сохранением объема путем его перераспределения между соседними ячейками.

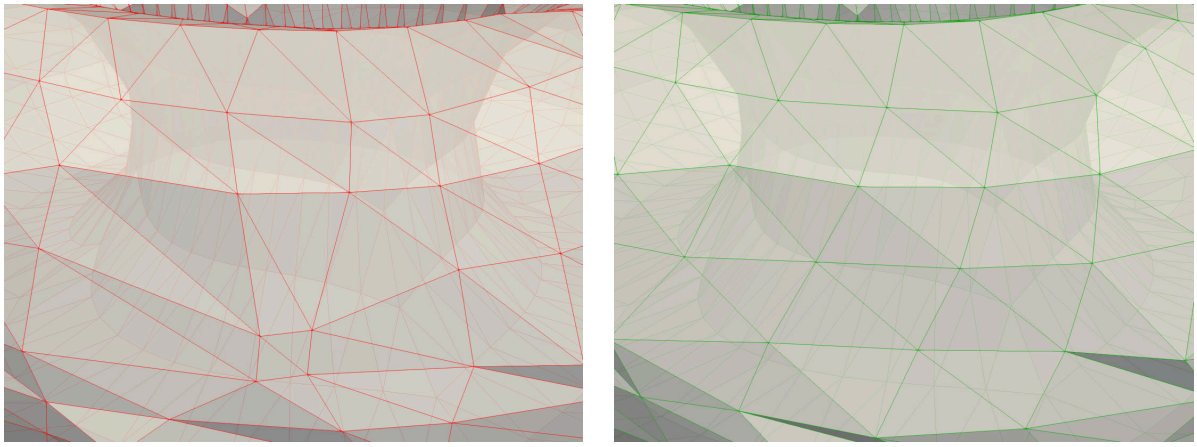

Рис. 2.10. Вид поверхностной сетки до сглаживания в нулевом пространстве (слева) и после него (справа)

Последним типом сглаживания является сглаживание в нулевом пространстве. Эволюция поверхности будет стремиться упаковать узлы в вогнутые области, где сходятся нормали к поверхности, тогда как расширение сетки происходит в выпуклых областях, где нормали к поверхности расходятся. Если узлы не будут перераспределяться, может стать невозможным продолжать стабильное перестроение сетки (слишком сильное сгущение сетки в вогнутых областям может приводить к конфликтам, а расширение сетки приводит к огрублению сетки). Для улучшения качества поверхностной сетки узлы перераспределяются на поверхности с помощью сглаживания в нулевом пространстве. Этот метод способен перераспределять точки, сохраняя при этом целостность базовой геометрии. Нулевое пространство определяется касательной плоскостью (для гладких областей), касательной линией (для складок поверхностной сетки) или пустым пространством (для углов), движущиеся в нем узлы остаются на поверхности, так что объем и форма поверхности могут быть сохранены (2.10). Алгоритм сглаживания может быть распространен на незамкнутые сетки путем закрепления граничных узлов сетки, описание этого механизма приведено в [273].

Слабым местом метода Tong является выбор доли $\alpha$ используемого объема при перестроении. При возникновении дефектов расчетной сетки (острые пики и впадины) параметр $\alpha$ оказывается близким к нулю даже при использовании всех видов сглаживания, и время перестроения может неограниченно возрасти, что приводит к невозможности продолжать расчеты.

## 2.5 Метод окрестностей перестроения поверхностной сетки в трехмерном случае

Предлагается метод перестроения поверхностной неструктурированной сетки, в котором каждый узел $N$ в процессе перестроения смещается на границу окрестности множества инцидентных ячеек $\mathscr{F}(N)$.

Рассмотрим геометрическую задачу определения новых положений узлов расчетной сетки, если для каждого узла $N$ известна скорость распространения фронта $v_N$ [274]. Будем считать, что движение сетки в любой точке роста выполняется одновременно во всех направлениях аналогично принципу Гюйгенса-Френеля распространения волн. Таким образом, фронт произвольной точки $P$ через промежуток времени $\Delta t$ будет иметь форму шара $Ball(P, v_P\Delta t)$. Далее будем считать, что мы выполняем расчет новых положений узлов через некоторый фиксированный момент времени $\Delta t$, то есть для каждого узла $N$ известен радиус $R_N = v_N\Delta t$. Так как мы рассматриваем поверхностную неструктурированную сетку, ячейки которой имеют форму треугольников, а радиусы заданы только в узлах сетки, необходимо определить радиусы для каждой точки внутри ячейки.

### 2.5.1 Постановка задачи для отдельной ячейки

Рассмотрим некоторую ячейку сетки, представляющую собой треугольник $F = Tri(A, B, C)$. Определим для каждой точки $P(\beta, \gamma)$ внутри треугольника радиус следующим образом:

$$R(P(\beta,\gamma)) = R(\beta,\gamma) = R(A) + \beta(R(B) - R(A)) + \gamma(R(C) - R(A)), \quad (2.25)$$

и на каждой точке $P(\beta,\gamma)$ построим шар $Ball(P(\beta,\gamma),R(\beta,\gamma))$, объединяя которые, получим окрестность всей ячейки $O_R(ABC)$ (см. рис. 2.11).

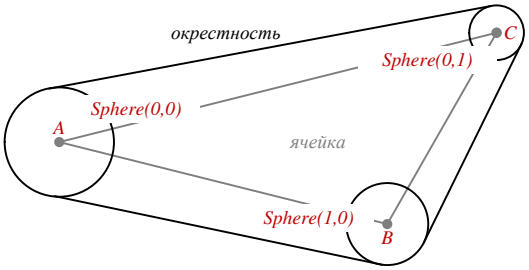


Рис. 2.11. Окрестность ячейки расчетной сетки

При изменении положения узлов ячейки (точки $A$, $B$, $C$) будем исходить из предположения, что новые положения узлов (точки $A'$, $B'$, $C'$) будут находиться на границе окрестности ячейки $O_R(ABC)$ (пока в расчете не учитываем влияние соседних расчетных ячеек). Без ограничения общности можно рассмотреть только одну вершину расчетной ячейки (точка $A$). Пусть траектория движения точки $A$ описывается уравнением полупрямой $\overline{P}(\alpha) = \overline{A} + \alpha\overline{D}$ при $\alpha \geq 0$ ($\overline{D}$ – вектор направления движения точки, при этом можно считать $|\overline{D}| = 1$).

Для поиска точек пересечения траектории движения точки $\overline{P}(\alpha) = \overline{A} + \alpha\overline{D}$ с произвольной сферой $Sphere(\beta,\gamma)$ необходимо подставить координаты точки $\overline{P}(\alpha)$ в уравнение сферы $|\overline{P} - \overline{C}(\beta,\gamma)| = R(\beta,\gamma)$, где $\overline{C}(\beta,\gamma)$ – ее центр. В результате получим уравнение

$$|(\overline{A} + \alpha\overline{D}) - \overline{C}(\beta,\gamma)| = R(\beta,\gamma). \tag{2.26}$$

Уравнение (2.26) решается относительно неизвестной $\alpha$ при фиксированных параметрах $\beta$, $\gamma$. Это уравнение является квадратным, оно имеет не более двух корней, которые будем рассматривать как функцию с двумя параметрами $\alpha_{1,2} = \alpha_{1,2}(\beta,\gamma)$. Для определения нового положения точки $A$ необходимо найти максимальное значение вещественного корня такого уравнения для всех допустимых значений параметров $\beta$ и $\gamma$: $\alpha = \max_{\substack{\beta\geq 0,\ \gamma\geq 0 \\ \beta+\gamma\leq 1}} \alpha_{1,2}(\beta,\gamma)$.

При этом точка пересечения траектории движения точки $A$ с границей окрестности ячейки $O_R(ABC)$ может находиться на разных участках этой границы, что продемонстрировано на рис. 2.12 и связано с условиями, которым удовлетворяют параметры $\beta$, $\gamma$ (а, б, в – пересечение со сферой с центром в вершинах треугольника; г, д, е – пересечение со сферой с центром на ребрах треугольника, g – пересечение со сферой с центром внутри треугольника).

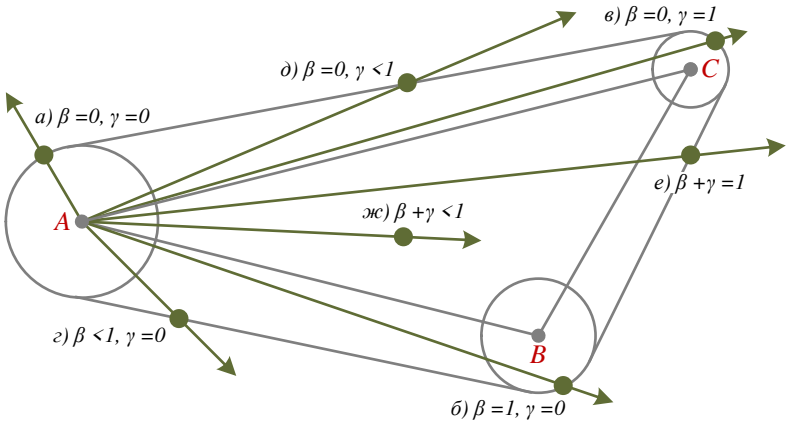


Рис. 2.12. Различные варианты пересечения траектории смещения узла и окрестности ячейки

Введем следующие обозначения $R_A = R(A)$, $R_{AB} = R(B) - R(A)$, $R_{AC} = R(C) - R(A)$, тогда (2.26) можно записать в виде

$$|\alpha\overline{D} - (\beta\overline{AB} + \gamma\overline{AC})|^2 = (R_A + \beta R_{AB} + \gamma R_{AC})^2, \tag{2.27}$$

или явно как квадратное уравнение относительно переменной $\alpha$:

$$|\overline{D}|^2\alpha^2 - 2(\beta(\overline{D}, \overline{AB}) + \gamma(\overline{D}, \overline{AC}))\alpha + |\beta\overline{AB} + \gamma\overline{AC}|^2 - (R_A + \beta R_{AB} + \gamma R_{AC})^2 = 0. \tag{2.28}$$

Наибольший корень этого квадратного уравнения вычисляется следующим образом (с учетом условия $|\overline{D}| = 1$):

$$\alpha(\beta,\gamma) = \beta(\overline{D},\overline{AB}) + \gamma(\overline{D},\overline{AC}) + \\ \sqrt{(\beta(\overline{D},\overline{AB}) + \gamma(\overline{D},\overline{AC}))^2 - |\beta\overline{AB} + \gamma\overline{AC}|^2 + (R_A + \beta R_{AB} + \gamma R_{AC})^2}, \quad (2.29)$$

или

$$\begin{aligned}
&\alpha(\beta,\gamma) = k_\beta\beta + k_\gamma\gamma + \sqrt{T},\\
&T = q_{\beta^2}\beta^2 + q_{\gamma^2}\gamma^2 + q_{\beta\gamma}\beta\gamma + q_\beta\beta + q_\gamma\gamma + q,\\
&k_\beta = (\overline{D},\overline{AB}),\ k_\gamma = (\overline{D},\overline{AC}),\\
&q_{\beta^2} = (\overline{D},\overline{AB})^2 - |\overline{AB}|^2 + R_{AB}^2,\ q_{\gamma^2} = (\overline{D},\overline{AC})^2 - |\overline{AC}|^2 + R_{AC}^2,\\
&q_{\beta\gamma} = 2\left((\overline{D},\overline{AB})(\overline{D},\overline{AC}) - (\overline{AB},\overline{AC}) + R_{AB}R_{AC}\right),\\
&q_\beta = 2R_AR_{AB},\ q_\gamma = 2R_AR_{AC},\ q = R_A^2.
\end{aligned} \quad (2.30)$$

### 2.5.2 Пересечения прямой с окрестностью отрезка

Для поиска решений уравнения (2.30) для всех вариантов допустимых параметров $\beta$ и $\gamma$, представленных на рис. 2.12, рассмотрим задачу поиска точек пересечения прямой с окрестностью отрезка [275].

Пусть в пространстве задан отрезок $Segm(A,B)$. Рассмотрим его окрестность $O_R(Segm(A,B))$, где функция $R$ задана на точках этого отрезка $\overline{C}(\alpha) = \overline{A} + \alpha\overline{AB},\ 0 \le \alpha \le 1$ следующим образом:

$$\begin{cases}
R(A) = R_A,\\
R(B) = R_B,\\
R(C(\alpha)) = R(\alpha) = R_A + \alpha(R_B - R_A) = R_A + \alpha\Delta R.
\end{cases} \quad (2.31)$$

Также пусть в пространстве задана прямая. Без ограничения общности можно считать, что эта прямая проходит через начало координат, пусть она имеет вид $Line(\overline{0},\overline{V}) = \{\overline{P}(\tau) = \tau\overline{V} : \tau \in \mathbb{R}\}$.

Требуется найти точки пересечения прямой $Line(\overline{0},\overline{V})$ и окрестности отрезка $O_R(Segm(A,B))$.

Точки пересечения прямой $Line(\overline{0}, \overline{V})$ и сферы $Sphere(C, R)$ находятся из решения системы уравнений

$$\begin{cases} \overline{P} = \tau \overline{V}, \\ |\overline{P} - \overline{C}| = R^2. \end{cases} \tag{2.32}$$

Система уравнений (2.32) преобразуется в квадратное уравнение относительно $\tau$, дискриминант которого равен $4\left((\overline{C}, \overline{V})^2 - |\overline{V}|^2\left(|\overline{C}|^2 - R^2\right)\right)$. Общие точки прямой и сферы существуют если этот дискриминант неотрицателен.

Аналогично, точки пересечения прямой $Line(\overline{0}, \overline{V})$ и сферы с центром в точке $C(\alpha)$ и радиусом $R(\alpha)$ существуют, если выполняется условие

$$(\overline{C}(\alpha), \overline{V})^2 - |\overline{V}|^2\left(|\overline{C}(\alpha)|^2 - R(\alpha)^2\right) \geq 0. \tag{2.33}$$

Подставив конкретные выражения для центра и радиуса сферы, получим квадратное неравенство относительно параметра $\alpha$:

$$k_2\alpha^2 + 2k_1\alpha + k_0 \geq 0, \tag{2.34}$$

где

$$\begin{aligned} k_2 &= (\Delta\overline{C}, \overline{V})^2 + |\overline{V}|^2\left(\Delta R^2 - |\Delta\overline{C}|^2\right), \\ k_1 &= (\overline{C}_A, \overline{V})(\Delta\overline{C}, \overline{V}) + |\overline{V}|^2\left(R_A\Delta R - (\overline{C}_A, \Delta\overline{C})\right), \\ k_0 &= (\overline{C}_A, \overline{V})^2 + |\overline{V}|^2\left(R_A^2 - |\overline{C}_A|^2\right). \end{aligned} \tag{2.35}$$

В системе (2.35) для удобства введены обозначения $C_A = A$, $\Delta\overline{C} = \overline{AB}$.

Неравенство (2.34) может быть решено относительно $\alpha$, так как все коэффициенты из (2.35) могут быть вычислены непосредственно. Нас интересует решение неравенства (2.34) только на отрезке $[0, 1]$, оно представляет собой либо пустое множество, либо отрезок. Если на отрезке $[0, 1]$ неравенство (2.34) не имеет решения, то прямая $Line(\overline{0}, \overline{V})$ не пересекает $O_R(Segm(A, B))$. Допустим на отрезке $[0, 1]$ неравенство (2.34) имеет решение $[\alpha_1, \alpha_2] \subseteq [0, 1]$.

Для произвольного $\alpha \in [\alpha_1, \alpha_2]$ корни уравнения

$$|\overline{V}|^2\tau^2 - 2(\overline{C}(\alpha), \overline{V})\tau + \left(|\overline{C}(\alpha)| - R(\alpha)^2\right) = 0 \tag{2.36}$$

существуют и выражаются следующим образом:

$$\tau_{2,1}(\alpha) = \frac{(\overline{C}(\alpha), \overline{V}) \pm \sqrt{k_2\alpha^2 + 2k_1\alpha + k_0}}{|\overline{V}|^2}. \tag{2.37}$$

Искомые точки пересечения прямой $Line(\overline{0}, \overline{V})$ с границей окрестности отрезка $O_R(Segm(A, B))$ соответствуют минимальному и максимальному значениям параметра $\tau_{1,2}(\alpha)$, достигаемым на отрезке $[\alpha_1, \alpha_2]$. Минимальное и максимальное значение $\tau_{1,2}(\alpha)$ могут достигаться либо на концах отрезка, либо в точках локального экстремума. Для точки локального экстремума должно быть выполнено условие $\tau'_{1,2}(\alpha) = 0$, или в явном виде

$$(\Delta\overline{C}, \overline{V}) \pm \frac{k_2\alpha + k_1}{\sqrt{k_2\alpha^2 + 2k_1\alpha + k_0}} = 0. \tag{2.38}$$

Введя обозначение $q = (\Delta\overline{C}, \overline{V})^2$, получим квадратное уравнение относительно $\alpha$:

$$k_2(k_2 - q)\alpha^2 + 2k_1(k_2 - q)\alpha + (k_1^2 - qk_0) = 0, \tag{2.39}$$

решая которое, получим потенциальные точки локальных экстремумов, которые обозначим $\tilde{\alpha}_1$, $\tilde{\alpha}_2$.

Точки $\tilde{\alpha}_1$, $\tilde{\alpha}_2$ нужно учитывать только если они попадают в отрезок $[\alpha_1, \alpha_2]$. В общем случае значения $\tau_{1,2}$ находятся по четырем значениям $\alpha$

$$\begin{cases} \tau_1 = \min(\tau_1(\alpha_1), \tau_1(\alpha_2), \tau_1(\tilde{\alpha}_1), \tau_1(\tilde{\alpha}_2)), \\ \tau_2 = \max(\tau_2(\alpha_1), \tau_2(\alpha_2), \tau_2(\tilde{\alpha}_1), \tau_2(\tilde{\alpha}_2)). \end{cases} \tag{2.40}$$

Множеством общих точек $Line(\overline{0}, \overline{V})$ и $O_R(Segm(A, B))$ является $\{\overline{P}(\tau) = \tau\overline{V} : \tau_1 \le \tau \le \tau_2\}$ (рис. 2.13).

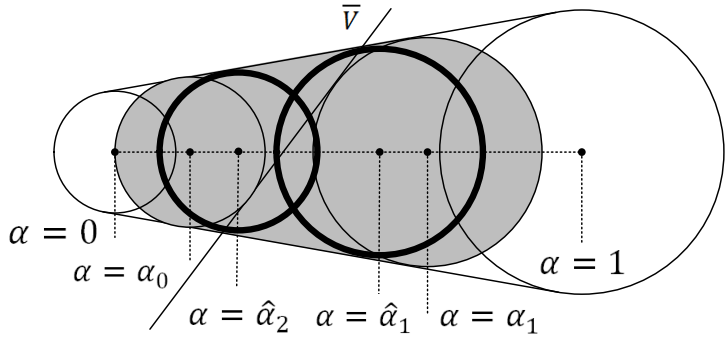


Рис. 2.13. Пересечение прямой с окрестностью отрезка

### 2.5.3 Поиск нового положения узла сетки

Для поиска нового положения узла $A$ требуется найти максимум выражения $\alpha(\beta, \gamma)$ из (2.30) при условии соблюдения ограничений $\beta \geq 0$, $\gamma \geq 0$, $\beta + \gamma \leq 1$. Максимум выражения $\alpha(\beta, \gamma)$ достигается либо при условии нахождения центра сферы внутри треугольника $ABC$ , либо на одной из его сторон. В случае нахождения центра сферы на стороне $AB$ треугольника $ABC$ выполняется условие $\gamma = 0$, и выражение для величины $\alpha$ имеет вид

$$\alpha_{\gamma=0} = k_\beta \beta + \sqrt{q_{\beta^2}\beta^2 + q_\beta \beta + q}. \tag{2.41}$$

В случае нахождения центра сферы на стороне $AC$ треугольника $ABC$ выполняется условие $\beta = 0$, и выражение для величины $\alpha$ имеет вид

$$\alpha_{\beta=0} = k_\gamma \gamma + \sqrt{q_{\gamma^2}\gamma^2 + q_\gamma \gamma + q}. \tag{2.42}$$

В случае нахождения центра сферы на стороне $BC$ треугольника $ABC$ выполняется условие $\beta + \gamma = 1$, и выражение для величины $\alpha$ имеет вид

$$\alpha_{\beta+\gamma=1} = (k_\gamma - k_\beta)\gamma + k_\beta + \sqrt{(q_{\beta^2} + q_{\gamma^2} + q_{\beta\gamma})\gamma^2 + (-2q_{\beta^2} + q_{\beta\gamma} - q_\beta + q_\gamma)\gamma + (q_{\beta^2} + q_\beta + q)}. \tag{2.43}$$

Во всех случаях нахождения центра сферы на одной из сторон треуголь-

ника задача нахождения максимального значения $\alpha(\beta, \gamma)$ при заданных ограничениях сводится к задаче поиска максимального значения функции вида $\alpha(x) = k_x x + k + \sqrt{q_{x^2} x^2 + q_x x + q}$ (с учетом этих ограничений), что является аналогом задачи поиска пересечения прямой и окрестностью отрезка, описание которой приведено в разделе 2.5.2 (точкой максимума является либо точка локального экстремума, либо точка, находящаяся на границе области определения функции).

Отдельно рассмотрим вариант, когда центр искомой сферы находится внутри треугольника. В этом случае точка пересечения траектории $\overline{P}(\alpha) = \overline{A} + \alpha\overline{D}$ находится на общей касательной плоскости к сферам $Sphere(0,0)$, $Sphere(1,0)$, $Sphere(0,1)$ (плоскость $A'B'C'$, рис. 2.14).

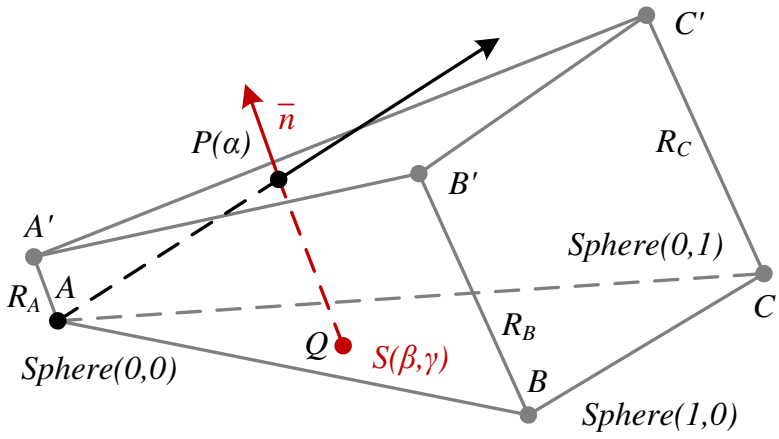


Рис. 2.14. Нахождение пересечения траектории движения точки с границей окрестности ячейки, если центр сферы лежит внутри треугольника

Центр искомой сферы находится в точке пересечения плоскости $ABC$ и прямой, проходящей через точку $\overline{P}(\alpha)$ и направленной вдоль нормали к плоскости $A'B'C'$ . Если вектор единичной нормали плоскости $A'B'C'$ обозначить через $\overline{n}$, то взаимосвязь точек плоскостей $ABC$ и $A'B'C'$ можно выразить следующим образом:

$$\begin{aligned} \overline{A}' &= \overline{A} + \overline{n}R_A, \\ \overline{B}' &= \overline{B} + \overline{n}R_B, \\ \overline{C}' &= \overline{C} + \overline{n}R_C. \end{aligned} \tag{2.44}$$

Для нахождения вектора $\overline{n}$ используем тот факт, что результат векторного произведения $\overline{A'B'} \times \overline{A'C'}$ коллинеарен вектору $\overline{n}$. Запишем это в явном виде:

$$(\overline{AB} + \overline{n}R_{AB}) \times (\overline{AC} + \overline{n}R_{AC}) = t\overline{n}, \tag{2.45}$$

$$\overline{AB} \times \overline{AC} + R_{AB}(\overline{n} \times \overline{AC}) - R_{AC}(\overline{n} \times \overline{AB}) = t\overline{n}, \tag{2.46}$$

$$t \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} + R_{AC} \begin{pmatrix} n_y AB_z - n_z AB_y \\ n_z AB_x - n_x AB_z \\ n_x AB_y - n_y AB_x \end{pmatrix} - R_{AB} \begin{pmatrix} n_y AC_z - n_z AC_y \\ n_z AC_x - n_x AC_z \\ n_x AC_y - n_y AC_x \end{pmatrix} = \overline{AB} \times \overline{AC}. \tag{2.47}$$

Соотношение (2.47) может выполняться как при $t > 0$, так и при $t < 0$, что соответствует существованию двух общих касательных плоскостей к трем сферам. Перепишем приведенное соотношение в виде системы линейных уравнений относительно составляющих нормали при произвольном значении параметра $t$.

$$\begin{pmatrix} t & R_{AC}AB_z - R_{AB}AC_z & R_{AB}AC_y - R_{AC}AB_y \\ R_{AB}AC_z - R_{AC}AB_z & t & R_{AC}AB_x - R_{AB}AC_x \\ R_{AC}AB_y - R_{AB}AC_y & R_{AB}AC_x - R_{AC}AB_x & t \end{pmatrix} \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \overline{AB} \times \overline{AC}. \tag{2.48}$$

Из системы уравнений (2.48) находятся два возможных направления нормали к общей касательной плоскости для сфер $Sphere(0,0)$, $Sphere(1,0)$, $Sphere(0,1)$. Для каждого направления нормали находится плоскость $A'B'C'$ и точка $\overline{P}(\alpha) = \overline{A} + \alpha\overline{D}$ на ней (и сам искомый параметр $\alpha$). Эту точку можно учитывать в общем наборе решений только в том случае, если ей соответствует сфера $Sphere(\beta, \gamma)$, параметры которой удовлетворяют условиям $\beta \geq 0$, $\gamma \geq 0$, $\beta + \gamma \leq 1$. Параметры $\beta$, и $\gamma$ можно определить путем поиска точки пересечения прямой $Line(P(\alpha), \overline{n})$ и плоскости $ABC$.

Найдя решения всех потенциально возможных частных случаев, представленных на рис. 2.12, находим множество решений $\alpha$, из которых для определения нового положения узла $A'$ необходимо выбрать максимальное. Изложенный выше метод касается определения смещения узла с учетом только одной инцидентной ячейки. При рассмотрении отдельного узла расчетной сетки требуется вычислить смещение этого узла относительно каждой инцидентной ячейки и выбрать среди этих смещений максимальное.

### 2.5.4 Алгоритм перестроения поверхностной сетки по методу окрестностей в трехмерной постановке

**Алгоритм 2.1.** *Алгоритм перестроения поверхностной неструктурированной сетки по методу окрестностей в трехмерной постановке (`Remesh`)*

**Вход:** $M(\mathscr{N}, \mathscr{E}, \mathscr{F})$ – поверхностная неструктурированная сетка.
**Выход:** Поверхностная неструктурированная сетка с измененными положениями узлов.

`// определение новых положений узлов`
**for** $N \in \mathscr{N}$ **do**
  $\alpha \leftarrow 0$
  **for** $F \in \mathscr{F}(N)$ **do**
    $A \leftarrow N$
    $\{B, C\} \leftarrow (\mathscr{N}(F) \setminus A)$
    $\overline{D} \leftarrow \overline{n}_N^N$
    $\alpha_F \leftarrow \max_{\substack{\beta\geq 0,\ \gamma\geq 0 \\ \beta+\gamma\leq 1}} \alpha(\beta, \gamma)$ из формулы (2.30)
    $\alpha \leftarrow \max(\alpha, \alpha_F)$
  **end**
  $NewPosition(N) \leftarrow (\overline{N} + \alpha \overline{n}_N^N)$
**end**
`// обновление положений узлов`
**for** $N \in \mathscr{N}$ **do**
  $N \leftarrow NewPosition(N)$
**end**

Алгоритм перестроения поверхностной неструктурированной сетки по методу окрестностей в трехмерной постановке представляет собой определение

новых положений узлов сетки. Новое положение узла сетки определяется как точка пересечения направления нормали этого узла с границей окрестности всех инцидентных узлу ячеек. Вычисление новых положений узлов выполняется явно по формуле (2.30).

Алгоритм `Remesh` имеет линейную сложность по количеству узлов сетки $O(|\mathscr{N}|)$. ■

Алгоритм перестроения поверхностной неструктурированной сетки по методу окрестностей в трехмерной постановке реализован в программном комплексе «Кристалл» компьютерного моделирования процесса обледенения элементов авиационных силовых установок [126, 127] и используется при моделировании обледенения поверхности, при этом эволюционирующая поверхностная неструктурированная сетки представляет собой внешнюю поверхность нарастаемого ледяного покрова.

### 2.5.5 Анализ метода окрестностей

Проведенные вычислительные эксперименты показали, что разработанный метод окрестностей перестроения поверхностной неструктурированной сетки, обладает способностью сглаживать дефекты сетки – острые пики и впадины. Для проверки сглаживания впадин выполнялась проверка метода на тестовых трехмерных моделях, в частности на модели Stanford Bunny [276].

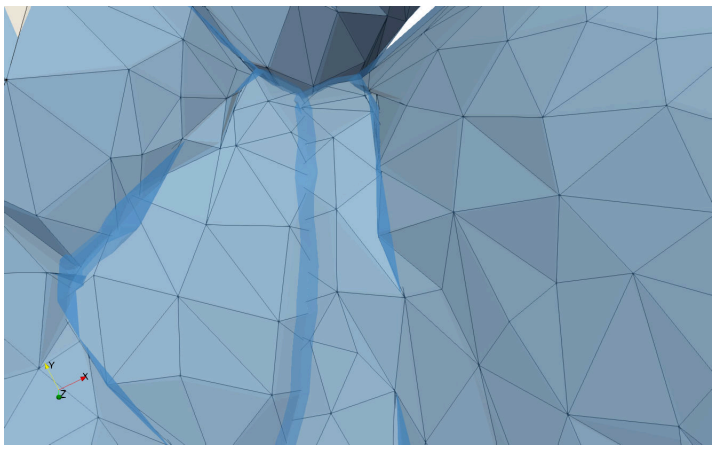

Рис. 2.15. Результат работы алгоритма перестроения сетки на тестовой модели Stanford Bunny

На рис. 2.15 представлен результат сравнения методов призм и окрестностей с точки зрения способности сглаживания впадин. Светло-синим цветом на рисунке отображена расчетная сетка, перестроенная с помощью метода призм. Темно-синим цветом показаны участки расчетной сетки, на которых было выполнено сглаживание впадин при перестроении с помощью метода окрестностей с использованием тех же величин смещения ячеек $H_F$.

Разработанный метод окрестностей перестроения поверхностной сетки используется в программном комплексе «Кристалл» для моделирования формы ледяных наростов в процессе обледенения летательных аппаратов [260–263]. При моделировании формы образовывающегося льда важно обеспечить увеличение моделируемого физического времени, в рамках которого исследуемый объект находится в корректном состоянии, в данном случае это означает описание поверхностной неструктурированной сеткой поверхности льда. Для этого сетка не должна содержать самопересечений и разделять расчетную область на две части: внутреннюю часть – содержащую лед, внешнюю часть – область окружающего пространства. Предложенный метод окрестностей позволяет увеличить моделируемое физическое время при моделировании обледенения.


Рис. 2.16. Псевдотрехмерные профили крыла с ледяным наростом при различных условиях обледенения

На рис. 2.16 представлены профили крыла летательного аппарата с образовавшимися ледяными наростами при различных условиях обледенения, на которых было произведено сравнение влияния методов перестроения сетки на моделируемое физическое время при эволюции поверхности льда. На рис. 2.17 сверху показан участок сетки, на котором накапливается и развивается само-

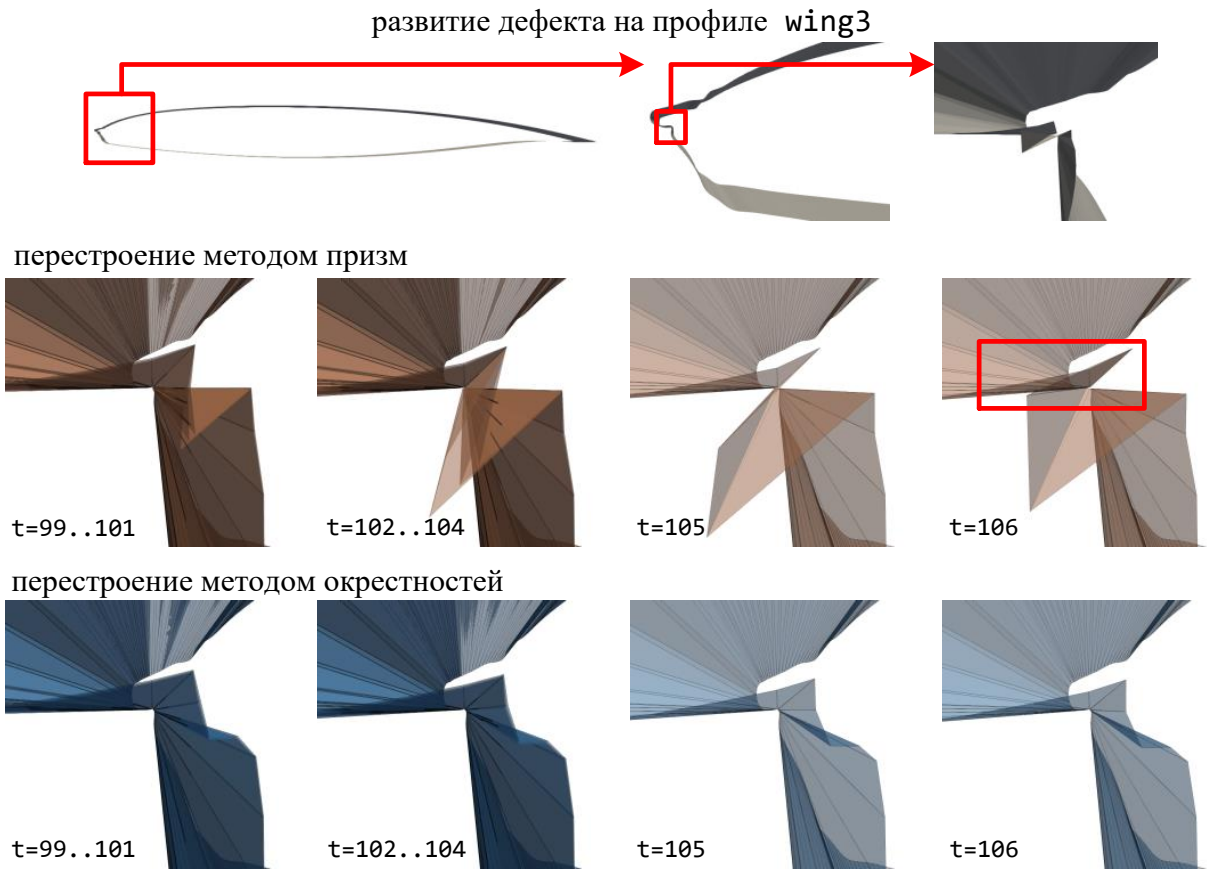


Рис. 2.17. Иллюстрация развития дефекта для профиля `wing3` на интервале времени $t \in [99, 106]$ секунд при использовании методов призм и окрестностей перестроения сетки

пересечение, после возникновение которого необходимо выполнять коррекцию. На рис. 2.17 снизу показаны профили в моменты времени $t \in [99, 106]$, в которые при использовании метода призм перестроения сетки (многослойное перестроение в 10 шагов) локальный дефект сетки развивается в самопересечение, а при использовании метода окрестностей перестроения сетки (также многослойное перестроение в 10 шагов) локальный дефект не развивается из-за свойства стягивания впадин.

Таблица 2.1. Статистика увеличения моделируемого физического времени $t$ на псевдотрехмерных профилях при моделировании обледенения

| тест | метод призм | метод окрестностей | увеличение $t$ |
|---|---|---|---|
| `wing1` | 50 с. | 73 с. | 46% |
| `wing2` | 59 с. | 77 с. | 31% |
| `wing3` | 106 с. | 133. с. | 25% |

В таблице 2.1 указаны моменты физического времени, в которые локальные дефекты развиваются в самопересечения сетки для разных профилей при использовании методов прямоугольников и окрестностей перестроения сетки. Из проведенного анализа можно сделать вывод, что метод окрестностей перестроения сетки повышает надежность перестроения и увеличивает моделируемое физическое время при моделировании объектов со сложной геометрией на примере моделирования поверхности ледяного нароста.

## 2.6 Оценки точности в двумерном случае

В предыдущем разделе для трехмерного метода перестроения поверхностной неструктурированной сетки свойство сглаживания локальных дефектов сетки продемонстрировано качественно. Приведем некоторые оценки точности и эффективности сглаживания дефектов, полученные в аналитическом виде на модельных сетках в двумерном случае.

Оценки будем проводить для модельной сетки, которая удовлетворяет следующим требованиям. Все ячейки сетки одинаковые и имеют длину $l$. Для любой ячейки $AB$ и ее соседей $A_1A$ и $BB_1$ углы $\angle(\overline{BA}, \overline{AA_1})$ и $\angle(\overline{AB}, \overline{BB_1})$ являются постоянной величиной и равны $\alpha$ (рис. 2.18). Пусть величины смещений ячеек $A_1A$, $AB$ и $BB_1$ равны $H_{i-1}$, $H_i$ и $H_{i+1}$ соответственно.

### 2.6.1 Оценки точности для выпуклой сетки

Сначала рассмотрим случай, когда сетка является выпуклой.

**Определение 2.4.** *Поверхностная расчетная сетка называется выпуклой в некоторой ячейке $AB$, если из двух полуплоскостей, на которые прямая $AB$ разбивает плоскость, все смежные $AB$ ячейки лежат в одной полуплоскости, а внешняя нормаль $\overline{n}^F_{AB}$ направлена в другую полуплоскость.*

Для вычисления величин $\Delta$, $\delta$ необходимо вычислить площадь $S_i = S_{AA'B'B}$. Эта площадь различается в зависимости от используемого метода перестроения неструктурированной сетки.

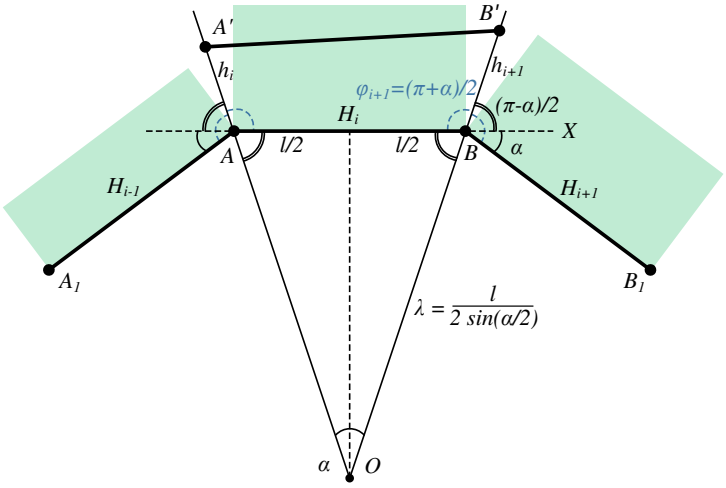


Рис. 2.18. Оценка приближенных методов перестроения выпуклой в ячейке $AB$ поверхностной расчетной сетки в двумерном случае

**Лемма 2.2.** *Для поверхностной расчетной сетки, выпуклой в ячейке* $F_i = AB$*, при выполнении условия* $\angle(\overline{BA}, \overline{AA_1}) = \angle(\overline{AB}, \overline{BB_1}) = \alpha$ *(рис. 2.18) верно соотношение*

$$S_i(h_i, h_{i+1}) = \frac{1}{2} \sin\alpha \left(\lambda(h_i + h_{i+1}) + h_i h_{i+1}\right), \tag{2.49}$$

*где* $\lambda = \frac{l}{2\sin\frac{\alpha}{2}}$, $l = AB$, $h_i = AA'$, $h_{i+1} = BB'$.

Так как рассматриваемая сетка выпуклая в ячейке $AB$, то лучи $A'A$ и $B'B$ пересекаются в некоторой точке $O$. При этом треугольник $AOB$ равнобедренный, $\angle BAO = \angle ABO = \angle B'BX = \angle B'BB_1 - \alpha = \frac{\pi-\alpha}{2}$, $\phi_i = \phi_{i+1} = \frac{\pi+\alpha}{2}$, $\angle AOB = \alpha$, откуда $OA = OB = \lambda = \frac{l}{2\sin\frac{\alpha}{2}}$. Далее выражаем площадь $S_i = S_{A'OB'} - S_{AOB}$, что после преобразования и приводит к (2.49). ■

Далее будем рассматривать линейное изменение величины смещения ячеек сетки, то есть $H_i = H$, $H_{i-1} = H - \Delta H$, $H_{i+1} = H + \Delta H$.

Для перестроения методом прямоугольников имеем $h_i = H - \frac{1}{2}\Delta H$, $h_{i+1} = H + \frac{1}{2}\Delta H$, откуда

$$S_i^r = \cos\frac{\alpha}{2}\left(lH + (H^2 - \frac{1}{4}\Delta H^2)\sin\frac{\alpha}{2}\right). \tag{2.50}$$

Теперь перейдем к вычислению $S_i^t$ для метода трапеций. В методе трапеций мы должны представить целевые площади в ячейках $A_1A$, $AB$, $BB_1$ с помощью равнобоких трапеций, для которых известно одно из оснований (одинаково для всех ячеек и равно $l$), угол при этом основании (одинаковый для всех ячеек и равен $\phi = \frac{\pi+\alpha}{2}$, а также площади этих трапеций, равные $T_{i-1} = l(H - \Delta H)$, $T_i = lH$ и $T_{i+1} = l(H + \Delta H)$ соответственно. На основании этих данных нам необходимо вычислить высоты трапеций, что можно сделать согласно (2.10). Обозначим высоты этих трапеций:

$$\begin{aligned} H_{i-1}^t &= H^t\left(l(H-\Delta H), l, \frac{\pi+\alpha}{2}\right), \\ H_i^t &= H^t\left(lH, l, \frac{\pi+\alpha}{2}\right), \\ H_{i+1}^t &= H^t\left(l(H+\Delta H), l, \frac{\pi+\alpha}{2}\right). \end{aligned} \tag{2.51}$$

Тогда $h_i = \frac{H_{i-1}^t + H_i^t}{2\sin\phi_i} = \frac{H_{i-1}^t + H_i^t}{2\cos\frac{\alpha}{2}}$, $h_{i+1} = \frac{H_i^t + H_{i+1}^t}{2\sin\phi_{i+1}} = \frac{H_i^t + H_{i+1}^t}{2\cos\frac{\alpha}{2}}$ и выражение (2.49) для $S_i^t$ принимает следующий вид:

$$S_i^t = \frac{1}{4}\left((H_{i-1}^t + 2H_i^t + H_{i+1}^t)l + (H_{i-1}^t + H_i^t)(H_i^t + H_{i+1}^t)\operatorname{tg}\frac{\alpha}{2}\right). \tag{2.52}$$

Для получения аналитической оценки точности для метода окрестностей в случае выпуклой сетки рассмотрим следующее вспомогательное утверждение.

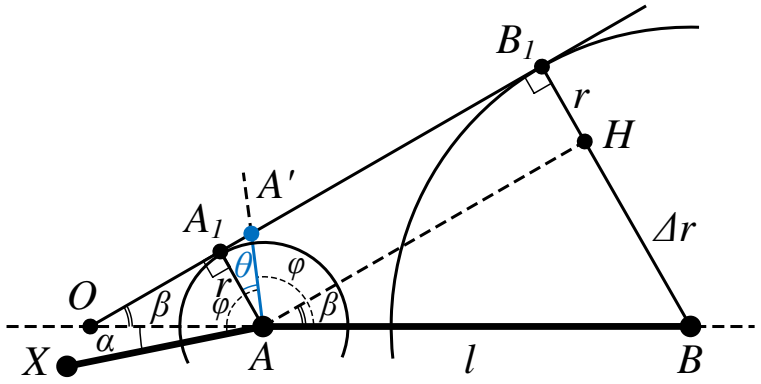


Рис. 2.19. Вычисление смещения узла в методе окрестностей для выпуклой расчетной сетки

**Лемма 2.3.** *Пусть $AB = l$, ниже прямой $AB$ расположена точка $X$ такая, что $\angle(\overline{AX}, \overline{BA}) = \alpha$. Пусть построены две окружности с центрами в точках $A$ и $B$ и с радиусами $r$ и $r + \Delta r$ соответственно. Для построенных окружностей построена выпуклая оболочка, состоящая из дуг этих окружностей, а также отрезков общих касательных (точки касания общей касательной построенных окружностей, лежащие выше прямой $AB$, обозначим $A_1$ и $B_1$ соответственно). Пусть на этой выпуклой оболочке – на окружности с центром в точке $A$, либо на отрезке $A_1B_1$ – лежит точка $A'$ такая, что $\angle XAA' = \angle BAA' = \phi$, $AA' = h$, тогда*

$$\begin{cases} h = r, \ \sin\frac{\alpha}{2} > \frac{\Delta r}{l}, \\ h(\alpha, l, r, \Delta r) = \frac{r}{\cos\left(\arcsin\frac{\Delta r}{l} - \frac{\alpha}{2}\right)}, \ \sin\frac{\alpha}{2} \leq \frac{\Delta r}{l}. \end{cases} \tag{2.53}$$

Если $\Delta r \leq 0$, то точка $A'$ находится на окружности с центром в точке $A$, то есть $h = r$.

Рассмотрим случай, когда $A'$ находится на отрезке $A_1B_1$, в этом случае $\Delta r > 0$. Через точку $A$ проведем прямую, параллельную $A_1B_1$, до пересечения с $BB_1$ в точке $H$. Тогда $\angle B_1OB = \angle HAB = \beta = \arcsin\frac{\Delta r}{l}$. Так как $\angle A'AB = \phi = \frac{\pi+\alpha}{2}$, а $\angle OAA_1 + \angle A_1AA' + \phi = \pi$, то $\theta = \angle A_1AA' = \beta - \frac{\alpha}{2}$.

Таким образом, условие нахождения точки $A'$ на отрезке $A_1B_1$ равносильно $\theta \geq 0$. То есть при $\theta < 0$, получаем $h = r$, а при $\theta \geq 0$ значение $h$ можно выразить из прямоугольного треугольника $AA'A_1$. ■

Используя лемму 2.3, для метода окрестностей можно выразить $S_i^o$ из (2.49) подставив туда выражения для $h_i$ и $h_{i+1}$ из (2.53):

$$\begin{aligned} h_i &= h\left(\alpha, l, H - \frac{1}{2}\Delta H, \Delta H\right), \\ h_{i+1} &= h\left(\alpha, l, H + \frac{1}{2}\Delta H, \Delta H\right). \end{aligned} \tag{2.54}$$

### 2.6.2 Оценки точности для вогнутой сетки

Теперь рассмотрим случай вогнутой сетки, представленный на рис. 2.20.

**Определение 2.5.** *Поверхностная расчетная сетка называется вогнутой в некоторой ячейке $AB$, если из двух полуплоскостей, на которые прямая $AB$ разбивает плоскость, все смежные $AB$ ячейки лежат в одной полуплоскости, и внешняя нормаль $\overline{n}^F_{AB}$ направлена в ту же полуплоскость.*

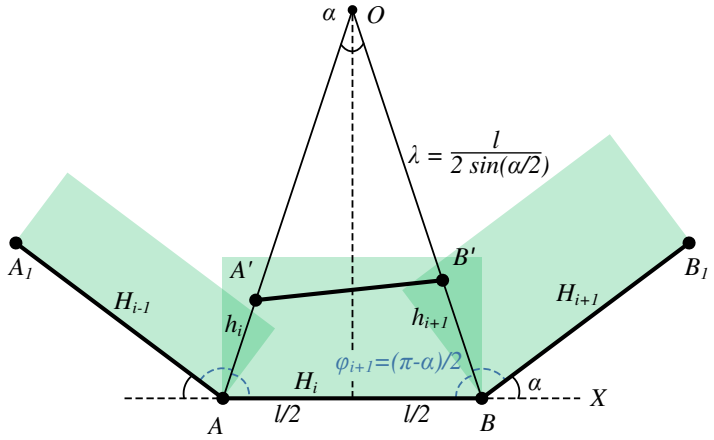


Рис. 2.20. Оценка приближенных методов перестроения вогнутой в ячейке $AB$ поверхностной расчетной сетки в двумерном случае

**Лемма 2.4.** *Для поверхностной расчетной сетки, вогнутой в ячейке $F_i = AB$, при выполнении условия $\angle(\overline{BA}, \overline{AA_1}) = \angle(\overline{AB}, \overline{BB_1}) = \alpha$ (рис. 2.20) верно соотношение*

$$S_i = \frac{1}{2}\sin\alpha\left(\lambda(h_i + h_{i+1}) - h_i h_{i+1}\right), \tag{2.55}$$

*где $\lambda = \frac{1}{2\sin\frac{\alpha}{2}}$, $l = AB$, $h_i = AA'$, $h_{i+1} = BB'$, при этом формула (2.55) применима при выполнении хотя бы одного из условий $h_i \le \lambda$, $h_{i+1} \le \lambda$.*

Вычисления производятся аналогично лемме 2.2, с той лишь разницей, что $S_i = S_{AOB} - S_{A'OB'}$, откуда имеем выражение (2.55).

Выполнение хотя бы одного из условий $h_i \le \lambda$, $h_{i+1} \le \lambda$ необходимо для предотвращения самопересечения сетки. ■

Для перестроения методом прямоугольников и методом трапеций в случае вогнутой сетки получим формулы, аналогичные (2.50), (2.51) и (2.52):

$$S_i^r = \cos\frac{\alpha}{2}\left(lH - (H^2 - \frac{1}{4}\Delta H^2)\sin\frac{\alpha}{2}\right), \tag{2.56}$$

$$\begin{aligned} H_{i-1}^t &= H^t\left(l(H-\Delta H), l, \frac{\pi-\alpha}{2}\right), \\ H_i^t &= H^t\left(lH, l, \frac{\pi-\alpha}{2}\right), \\ H_{i+1}^t &= H^t\left(l(H+\Delta H), l, \frac{\pi-\alpha}{2}\right). \end{aligned} \tag{2.57}$$

$$S_i^t = \frac{1}{4}\left((H_{i-1}^t + 2H_i^t + H_{i+1}^t)l - (H_{i-1}^t + H_i^t)(H_i^t + H_{i+1}^t)\operatorname{tg}\frac{\alpha}{2}\right). \tag{2.58}$$

Заметим, что формулы (2.56) – (2.58) идентичны формулам (2.50) – (2.52) с учетом знака $\alpha$, поэтому формулы (2.50) – (2.52) применимы как для выпуклой расчетной сетки (в этом случае используется положительное значение угла $\alpha$), так и для вогнутой сетки (в этом случае используем формулы (2.50) – (2.52) с отрицательным значением угла $\alpha$).

Для получения аналитической оценки точности для метода окрестностей в случае вогнутой сетки рассмотрим следующее вспомогательное утверждение, аналогичное лемме 2.3.

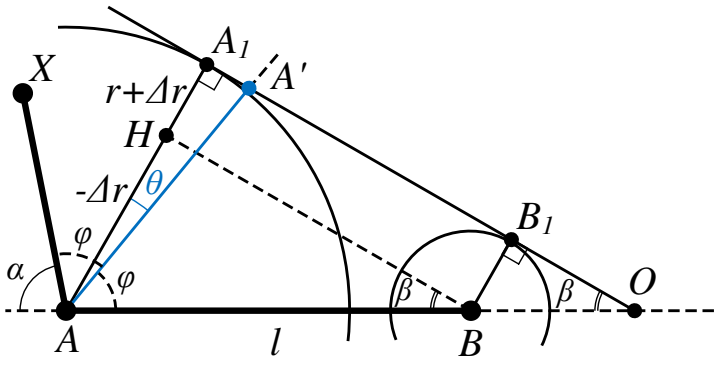


Рис. 2.21. Вычисление смещения узла в методе окрестностей для вогнутой сетки

**Лемма 2.5.** *Пусть $AB = l$, выше прямой $AB$ расположена точка $X$ такая, что $\angle(\overline{AX}, \overline{BA}) = \alpha$. Пусть построены две окружности с центрами*

*в точках $A$ и $B$ и с радиусами $r$ и $r+\Delta r$ соответственно. Для построенных окружностей построена выпуклая оболочка, состоящая из дуг этих окружностей, а также отрезков общих касательных (точки касания общей касательной построенных окружностей, лежащие выше прямой $AB$, обозначим $A_1$ и $B_1$ соответственно). Пусть на этой выпуклой оболочке – на окружности с центром в точке $A$ либо на отрезке $A_1B_1$ – лежит точка $A'$ такая, что $\angle XAA' = \angle BAA' = \phi$, $AA' = h$, тогда*

$$\begin{cases} h = r, \ \sin\frac{\alpha}{2} < \frac{-\Delta r}{l}, \\ h(\alpha, l, r, \Delta r) = \frac{r}{\cos\left(\frac{\alpha}{2} - \arcsin\frac{-\Delta r}{l}\right)}, \ \sin\frac{\alpha}{2} \geq \frac{-\Delta r}{l}. \end{cases} \tag{2.59}$$

Если $\Delta r \geq 0$, то точка $A'$ находится на отрезке $A_1B_1$.

Рассмотрим случай, когда $A'$ находится на отрезке $A_1B_1$, и $\Delta r < 0$. Через точку $B$ проведем прямую, параллельную $A_1B_1$, до пересечения с $AA_1$ в точке $H$. Тогда $\angle A_1OA = \angle HBA = \beta = \arcsin\frac{-\Delta r}{l}$. Так как $\angle A'AB = \phi = \frac{\pi-\alpha}{2}$, то $\theta = \angle A_1AA' = \angle A_1AO - \phi = \frac{\alpha}{2} - \beta$.

Таким образом, условие нахождения точки $A'$ на отрезке $A_1B_1$ равносильно $\theta \geq 0$. То есть при $\theta < 0$, получаем $h = r$, а при $\theta \geq 0$ значение $h$ можно выразить из прямоугольного треугольника $AA'A_1$. ■

Используя лемму 2.5, для метода окрестностей можно выразить $S_i^o$ из (2.55) подставив туда выражения для $h_i$ и $h_{i+1}$ из (2.59):

$$\begin{aligned} h_i &= \max\left(h\left(\alpha, l, H - \frac{3}{2}\Delta H, \Delta H\right), h\left(\alpha, l, H - \frac{1}{2}\Delta H, \Delta H\right)\right), \\ h_{i+1} &= \max\left(h\left(\alpha, l, H - \frac{1}{2}\Delta H, \Delta H\right), h\left(\alpha, l, H + \frac{1}{2}\Delta H, \Delta H\right)\right). \end{aligned} \tag{2.60}$$

### 2.6.3 Анализ оценок точности

Используя полученные в предыдущем разделе соотношения для $S_i^r$, $S_i^t$, $S_i^o$, проанализируем зависимости величин $\delta_i^r$, $\delta_i^t$, $\delta_i^o$ от $\alpha$ и $\frac{\Delta H}{l}$, при этом будем использовать фиксированную величину $H = \frac{l}{2}$ (рис. 2.22).

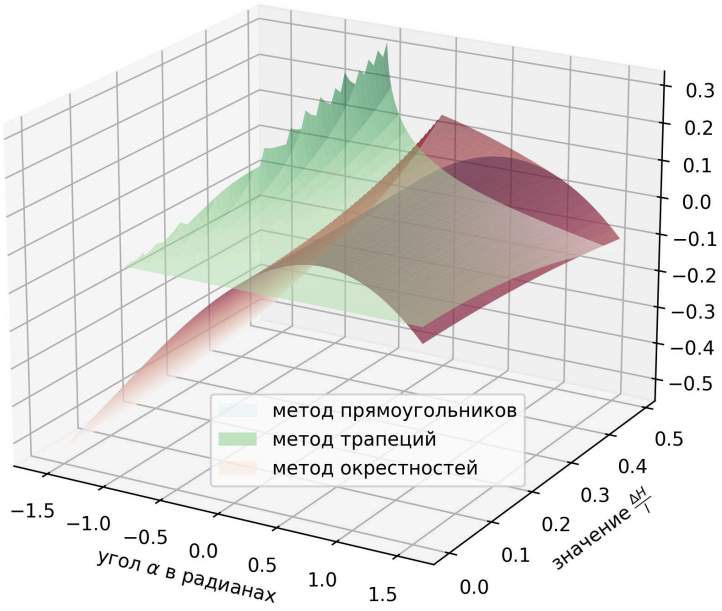


Рис. 2.22. Графики поверхностей $\delta_i^r(\alpha, \frac{\Delta H}{l})$, $\delta_i^t(\alpha, \frac{\Delta H}{l})$, $\delta_i^o(\alpha, \frac{\Delta H}{l})$ при фиксированном значении $H = \frac{l}{2}$

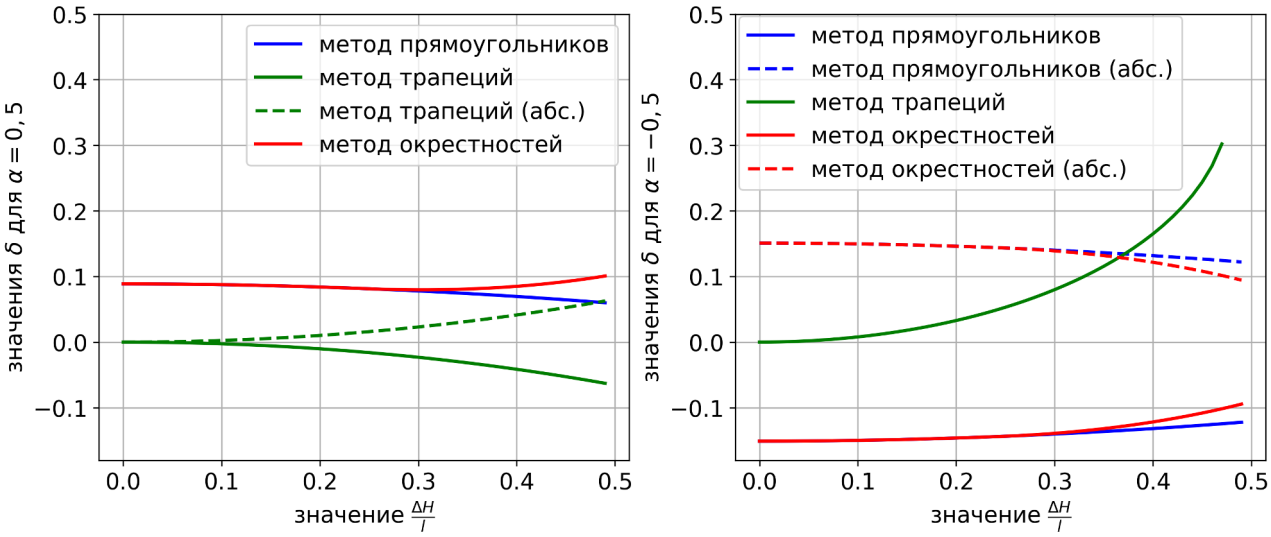


Рис. 2.23. Графики зависимостей $\delta_i^r(\frac{\Delta H}{l})$, $\delta_i^t(\frac{\Delta H}{l})$, $\delta_i^o(\frac{\Delta H}{l})$ при фиксированных значениях $\alpha = 0,5$ (слева) и $\alpha = -0,5$ (справа)

Анализируя полученные зависимости, графики которых приведены на рис. 2.22, можно отметить, что при $\Delta H = 0$ метод трапеций абсолютно точен. Также отметим, что при малых значениях $\Delta H$ отклонения $\delta_i^r$ и $\delta_i^o$ практически совпадают.

Построим дополнительно два графика зависимостей $\delta_i^r$, $\delta_i^t$, $\delta_i^o$ от $\frac{\Delta H}{l}$ при фиксированных значениях $\alpha = 0,5$ (для выпуклой сетки) и $\alpha = -0,5$ (для

вогнутой сетки) (рис. 2.23). Из рисунка видно, что метод трапеций перестроения поверхности при небольших значениях $\frac{\Delta H}{l}$ наиболее точен, а точность методов прямоугольников и окрестностей достаточно близки.

## 2.7 Оценки сглаживания пиков и впадин

В процессе перестроения поверхностной расчетной сетки могут возникать мелкие дефекты, такие как острые пики (если значение $\phi_i$ близко к $\pi$) и впадины (если значение $\phi_i$ близко к нулю).

Приведем оценки сглаживания острых пиков и впадин для методов прямоугольников, трапеций и окрестностей. Для этого будем рассматривать расчетную сетку с одинаковыми ячейками-отрезками длины $l$.

Для оценки сглаживания острых пиков будем считать, что сетка является абсолютно плоской за исключением двух соседних ячеек, которые образуют острый пик с углом $2\alpha$ (рис. 2.24 слева). Аналогично для оценки сглаживания впадин будем считать, что сетка является абсолютно плоской за исключением двух соседних ячеек, которые образуют впадину с углом $2\alpha$ (рис. 2.24 справа).

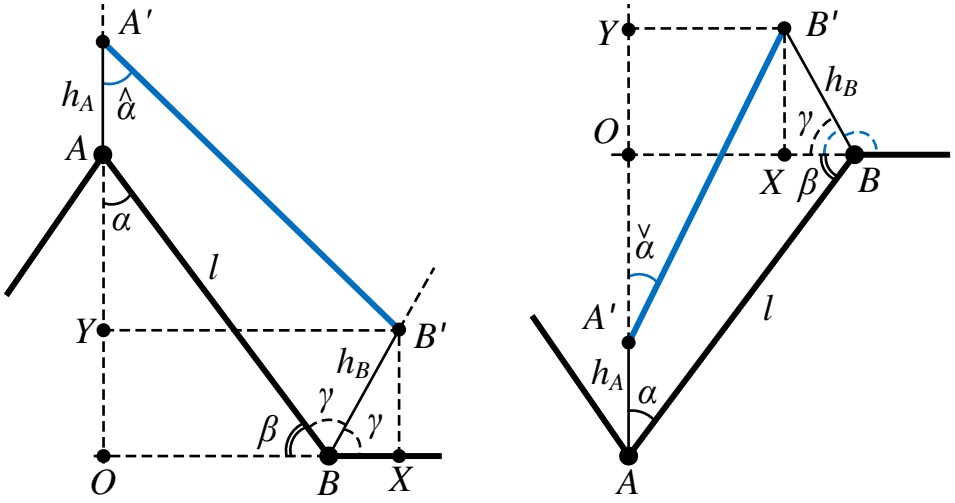


Рис. 2.24. Оценка сглаживания угла при остром пике (слева) и при впадине (справа)

### 2.7.1 Получение оценок сглаживания

Во всех трех методах (прямоугольников, трапеций и окрестностей) направления смещения узлов совпадают (и лежат на биссектрисах углов, образованных соседними ячейками), поэтому будем рассматривать задачу сглаживания угла при остром пике и при впадине для произвольных смещений узлов $A$ и $B$. Для этого докажем следующие леммы.

**Лемма 2.6.** *При перемещении узлов $A$ и $B$ – узлов одной из сторон острого пика – вдоль биссектрис углов, образованных инцидентными ячейками, на расстояния $h_A$ и $h_B$ соответственно (рис. 2.24 слева) значение сглаженного угла $\hat{\alpha} = \angle OA'B'$ выражается через $\alpha = \angle OAB$ и $\gamma = \frac{\pi}{4} + \frac{\alpha}{2}$ следующим образом:*

$$\hat{\alpha}(h_A, h_B) = \operatorname{arctg} \frac{l \sin\alpha + h_B \cos\gamma}{l \cos\alpha + h_A - h_B \sin\gamma}. \tag{2.61}$$

Выразим через $\alpha$ остальные углы: $\beta = \angle OBA = \frac{\pi}{2} - \alpha$, $\gamma = \angle ABB' = \angle XBB' = \frac{\pi}{4} + \frac{\alpha}{2}$. Через углы $\alpha$ и $\gamma$ находим $OA = l\cos\alpha$, $OB = l\sin\alpha$, $BX = h_B\cos\gamma$, $B'X = h_B\sin\gamma = OY$, откуда получаем $YB' = OB + BX = l\sin\alpha + h_B\cos\gamma$, $YA' = OA + AA' - OY = l\cos\alpha + h_A - h_B\sin\gamma$, откуда получаем $\hat{\alpha}(h_A, h_B) = \operatorname{arctg}\frac{YB'}{YA'}$, что приводит к (2.61). ■

**Лемма 2.7.** *При перемещении узлов $A$ и $B$ – узлов одной из сторон впадины – вдоль биссектрис углов, образованных инцидентными ячейками, на расстояния $h_A \le AO$ и $h_B$ соответственно (рис. 2.24 справа) значение сглаженного угла $\check{\alpha} = \angle OA'B'$ выражается через $\alpha = \angle OAB$ и $\gamma = \frac{\pi}{4} + \frac{\alpha}{2}$ следующим образом:*

$$\check{\alpha}(h_A, h_B) = \operatorname{arctg} \frac{l \sin\alpha - h_B \cos\gamma}{l \cos\alpha - h_A + h_B \sin\gamma} \tag{2.62}$$

*при ограничениях на угол $\alpha$*

$$\arcsin \frac{h_B\left(\sqrt{8l^2 + h_B^2} - h_B\right)}{4l^2} \le \alpha \le \arccos \frac{h_A}{l}. \tag{2.63}$$

Выразим через $\alpha$ остальные углы: $\beta = \angle OBA = \frac{\pi}{2} - \alpha$, $\gamma = \angle OBB' = \frac{\pi}{4} + \frac{\alpha}{2}$. Через углы $\alpha$ и $\gamma$ находим $OA = l\cos\alpha$, $OB = l\sin\alpha$, $BX = h_B\cos\gamma$,

$B'X = h_B \sin\gamma = OY$, откуда получаем $YB' = OB - BX = l\sin\alpha - h_B\cos\gamma$, $YA' = OA - AA' + OY = l\cos\alpha - h_A + h_B\sin\gamma$, откуда получаем $\check{\alpha}(h_A, h_B) = \operatorname{arctg}\frac{YB'}{YA'}$, что приводит к (2.62).

Рассмотрим условия применимости (2.62). Условие $h_A \le AO$ равносильно $\alpha \le \arccos\frac{h_A}{l}$. Требованием на отсутствие самопересечения сетки является условие $BX \le OB$, то есть $h_B\cos\gamma \le l\sin\alpha$. Подставим выражение для $\cos\gamma$:

$$h_B\frac{1}{\sqrt{2}}\left(\cos\frac{\alpha}{2} - \sin\frac{\alpha}{2}\right) \le l\sin\alpha. \tag{2.64}$$

Так как $\alpha < \frac{\pi}{2}$, то обе части неравенства положительные, поэтому возведем их в квадрат, и после преобразований получим

$$1 - \sin\alpha \le 2\left(\frac{l}{h_B}\right)^2 \sin^2\alpha. \tag{2.65}$$

Введя обозначение $p = 2\left(\frac{l}{h_B}\right)^2$ и решая квадратное неравенство (2.65) относительно $\sin\alpha$, получим условие $\alpha \ge \arcsin\frac{\sqrt{4p+1}-1}{2p}$, что в совокупности с условием $\alpha \le \arccos\frac{h_A}{l}$ приводит к (2.63). ■

### 2.7.2 Анализ оценок сглаживания пиков

На основе Леммы 2.6 получим оценки сглаживания угла при остром пике при условии постоянного значения $H$ во всех ячейках сетки.

Для метода прямоугольников имеем $h_A = h_B = H$ (рис. 2.25 слева).

Для метода трапеций необходимо сначала найти высоты двух трапеций $AA'B^-B$ и $CC'B^+B$ с рис. 2.25 в центре из условия равенства площадей этих трапеций значению $lH$. Высоты этих трапеций $H^-$ и $H^+$ получаются из (2.8) с помощью решения квадратного уравнения. Тогда $h_A = \frac{H^-}{\sin\alpha}$, $h_B = \frac{H^- + H^+}{2\sin\gamma}$. Из выражения для $h_A$ видно, что при малых значениях $\alpha$ смещение $h_A$ неконтролируемо возрастает.

В случае метода окрестностей $h_A = H$, $h_B = \frac{H}{\sin\gamma}$ (рис. 2.25 справа).

На рис. 2.26 приведены графики сравнения методов прямоугольников, трапеций и окрестностей в применении к сглаживанию острых пиков. При

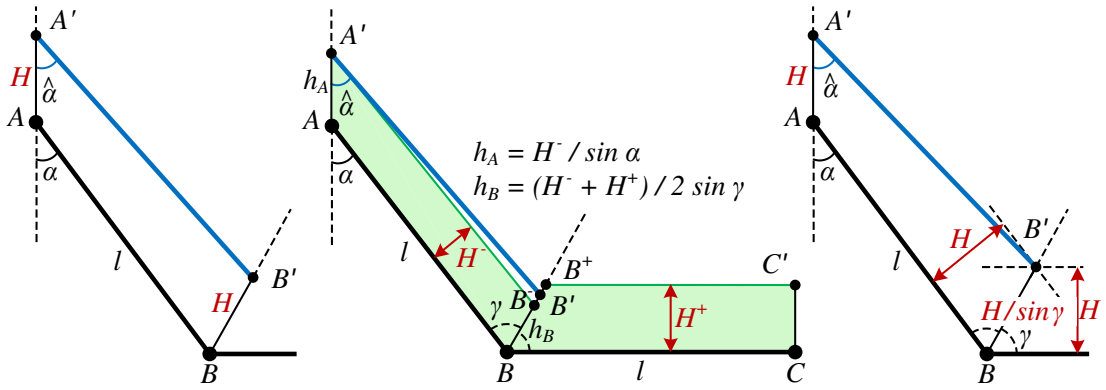


Рис. 2.25. Сглаживание пика при методах прямоугольников (слева), трапеций (в центре), окрестностей (справа)

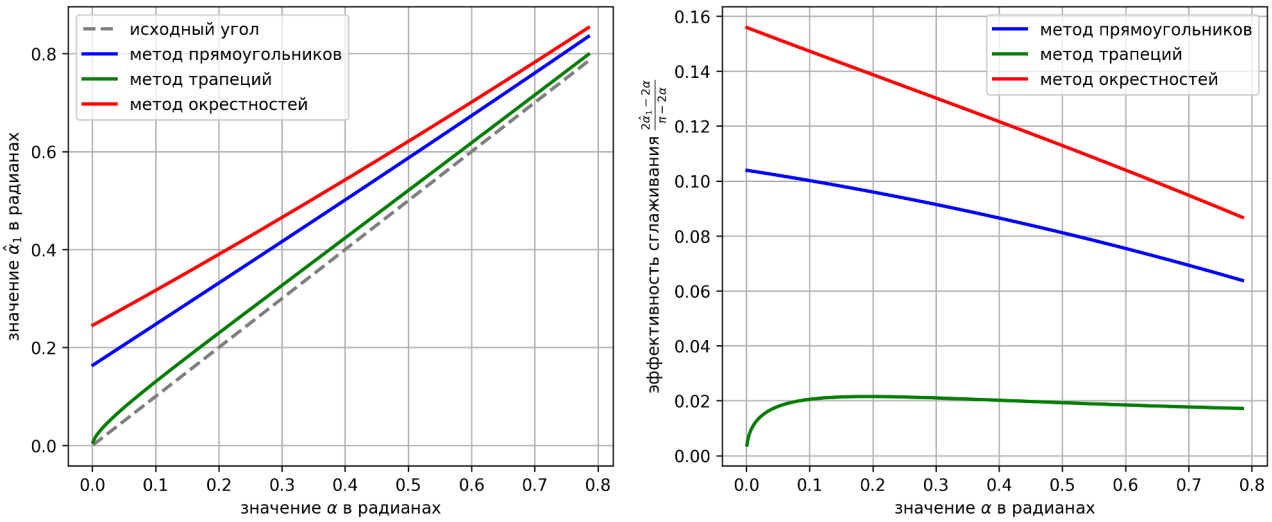


Рис. 2.26. Сравнение сглаживания острого пика для методов прямоугольников, трапеций и окрестностей

этом используется фиксированная величина смещения ячейки $H = \frac{l}{4}$. На графике слева показана зависимость изменения сглаженного угла $\hat{\alpha}$ от $\alpha$. На графике справа показана эффективность сглаживания, выраженная формулой $\frac{\check{\alpha}-\alpha}{\frac{\pi}{2}-\alpha} = \frac{2\check{\alpha}-2\alpha}{\pi-2\alpha}$ (значение 1 означает полное сглаживание пика до угла $\hat{\alpha} = \frac{\pi}{2}$).

Можно отметить, что наиболее эффективное сглаживание угла $\alpha$ обеспечивает метод окрестностей, а наименее эффективное – метод трапеций. Дополнительно заметим, что использование метода трапеций для малых углов $\alpha$ приводит к неконтролируемому росту $h_A$, что делает применение этого метода неприемлемым.

### 2.7.3 Анализ оценок сглаживания впадин

На основе Леммы 2.7 получим оценки сглаживания угла при впадине при условии постоянного значения $H$ во всех ячейках сетки.

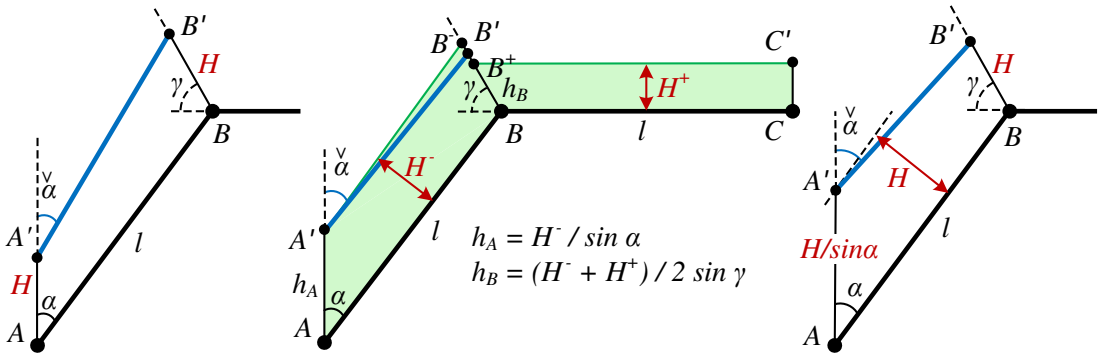


Рис. 2.27. Сглаживание впадины при методах прямоугольников (слева), трапеций (в центре), окрестностей (справа)

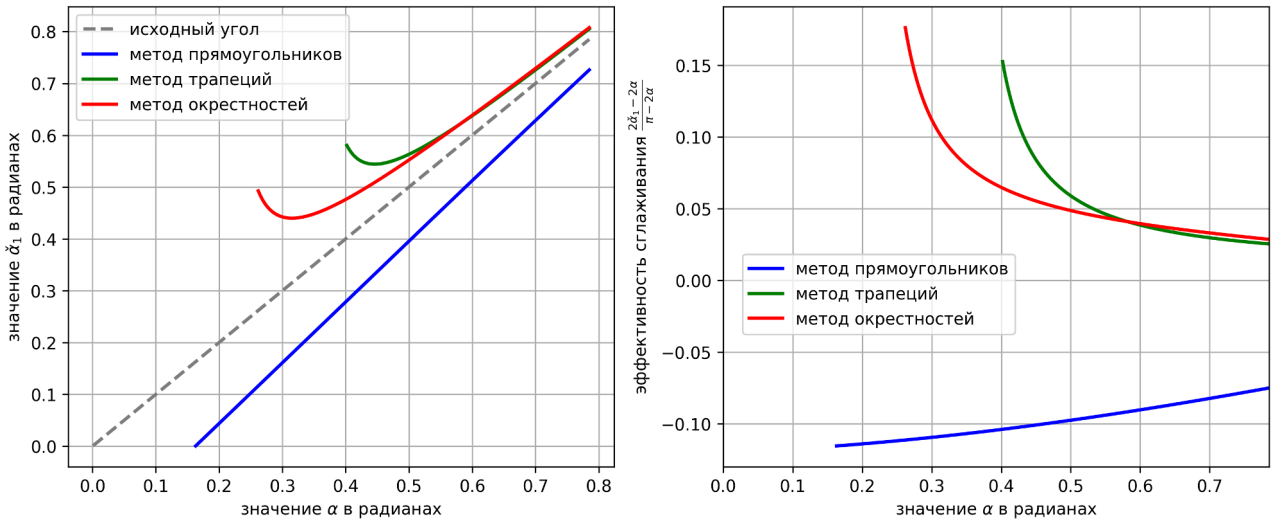


Рис. 2.28. Сравнение сглаживания впадины для методов прямоугольников, трапеций и окрестностей

Для метода прямоугольников имеем $h_A = h_B = H$ (рис 2.27 слева).

Для метода трапеций необходимо сначала найти высоты двух трапеций $AA'B^-B$ и $CC'B^+B$ с рис. 2.27 в центре из условия равенства площадей этих трапеций значению $lH$. Высоты этих трапеций $H^-$ и $H^+$ получаются из (2.8) с помощью решения квадратного уравнения. Тогда $h_A = \frac{H^-}{\sin\alpha}$, $h_B = \frac{H^- + H^+}{2\sin\gamma}$.

В случае метода окрестностей $h_A = \frac{H}{\sin\alpha}$, $h_B = H$ (рис. 2.27 справа).

На рис. 2.28 приведены графики сравнения методов перестроения при сглаживании впадин. Используется фиксированная величина смещения ячейки $H = \frac{l}{4}$. На графике слева показана зависимость изменения сглаженного угла $\check{\alpha}$ от $\alpha$. На графике справа показана эффективность сглаживания, выраженная формулой $\frac{2\check{\alpha}-2\alpha}{\pi-2\alpha}$ (значение 1 – полное сглаживание впадины до угла $\check{\alpha} = \frac{\pi}{2}$).

Можно отметить, что метод прямоугольников не сглаживает угол, а наоборот, делает его еще более острым (значение эффективности сглаживания меньше нуля). Метод трапеций показывает лучшую эффективность сглаживания, но с меньшей областью применимости.

Из приведенных оценок на рис. 2.26 и рис. 2.28 можно сделать следующие выводы. С точки зрения сглаживания дефектов сетки метод трапеций неприменим, так как приводит к неконтролируемому росту острых пиков. Метод прямоугольников неэффективен при сглаживании острых пиков и впадин. Метод окрестностей позволяет сглаживать как острые пики, так и впадины.

## 2.8 Удаление самопересечений поверхностной сетки

Во время эволюции поверхностной сетки применение сглаживаний может отложить возникновение самопересечений, но полностью их исключить нельзя (рис. 2.29), поэтому возникает необходимость обработки этого глобального дефекта для поддержания сетки в корректном состоянии описания замкнутой двусторонней поверхности.

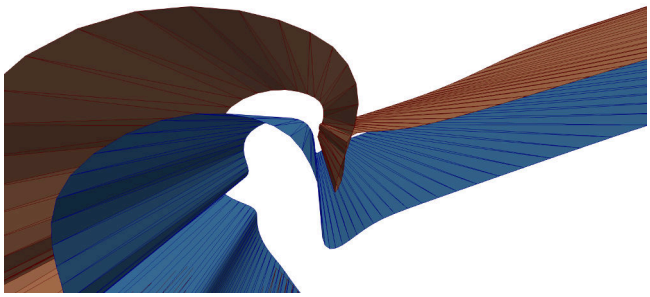

Рис. 2.29. Возникновение самопересечения при перестроении сетки

Методы сглаживания поверхностной сетки, рассмотренные в разделе 2.4.3.2, пригодны для устранения локальных дефектов, тогда как для

удаления самопересечений необходимы принципиально другие подходы [277]. Удаление самопересечений будем рассматривать для трехмерной замкнутой поверхностной неструктурированной сетки с треугольными ячейками, определенной в разделе 2.1.3, для такой сетки заданы направления внешних нормалей ячеек, эта сетка разделяет пространство на внутреннюю и внешнюю области.

Одним из известных подходов к реконструкции поверхностной сетки является использование построенной тетраэдральной объемной сетки в расчетной области. Этот метод впервые был описан в [170], после чего были разработаны его усовершенствованные версии, направленные на компенсацию ошибок вычислений с плавающей точкой [171], а также на ускорение обработки [118]. Узким местом такого подхода является необходимость построения тераэдральной объемной сетки в расчетной области, что связано с дополнительными вычислительными затратами, поэтому рассматриваются методы, работающие непосредственно на поверхностной сетке.

Среди ранних работ по поиску и удалению самопересечений (или пересечений различных сеток), работающих непосредственно на поверхностной сетке, следует отметить следующие. Рассмотрение пересечений в виде замкнутых ломаных и идентификация пересечений с помощью обхода ломаных на базе алгоритма TNOIT (Tracing of the Neighbours Of Intersecting Triangles) [172], при этом обычно рассматриваются непересекающиеся ломаные, которые могут быть обработаны независимо. Для поиска стартовых точек обхода ломаных пересечений эффективным оказывается использование BVH-деревьев (Bounding Volume Hierarchy) [278], позволяющее быстро находить пары потенциально пересекающихся ячеек. Обход ломаной пересечения порождает задачу дробления ячеек на более мелкие, чтобы звенья ломаной пересечения являлись ребрами ячеек (триангуляция конфликтных ячеек по ломаной).

Центральной проблемой обработки пересечений сеток является определение пересечения пары ячеек – двух треугольников в пространстве. Прямое решение этой задачи может выполняться в ошибками из-за вычислений с плавающей точкой (при использовании вещественных координат узлов). Из-за погрешности вычислений пересечение пары ячеек может быть определено

некорректно. Для уточнения ситуаций, в которых проблема погрешности вычислений стоит особенно остро, введем понятие простой сетки.

**Определение 2.6.** *Трехмерную замкнутую поверхностную неструктурированную сетку с треугольными ячейками будем называть простой, если в ней отсутствуют конфликты между элементами, которые могут быть некорректно определены из-за погрешности вычислений. А именно: никакие два узла не совпадают, никакой узел не лежит на неинцидентном ему ребре, никакой узел не лежит в неинцедентной ему ячейке, никакие два несмежные ребра не пересекаются (ни по точке, ни по отрезку), отсутствует наложение ячеек.*

По аналогии с определением 2.6 можно определить простое взаимное расположение двух сеток. При использовании простых сеток с некоторым допустимым значением погрешности вычислений $\varepsilon$ задача обработки пересечений и самопересечений решается гораздо проще. В общем случае, применяются различные подходы для компенсации погрешности вычислений. Среди них локальная коррекция сетки для приведения к простому виду [120], компенсация ошибок с помощью введения дополнительных контрольных кодов классификации пересечения пар ячеек и восстановления отдельных потерянных звеньев ломаной пересечения [121], использование рациональной арифметики для обеспечения вычислений с абсолютной точностью [124].

Среди современных методов удаления самопересечений отметим два наиболее развитых метода, основанных на поиске пересечений всех пар ячеек в сетке и дроблении ячеек по точкам и отрезкам пересечения: Exact and Efficient Intersection Resolution (EEIR) [123] и Fast and Robust Mesh Arrangements (FRMA) [122]. В этих методах вычисления выполняются с использованием чисел с плавающей точкой, а для компенсации ошибок вычислений используются неявные представления точек пересечения через другие точки, чьи координаты заданы точно: представление пересечения двух прямых Line-Line Intersection (LLI), представление пересечения прямой и плоскости Line-Plane Intersection (LPI), представление пересечения трех плоскостей Triplet-Plane Intersection

(TPI). В работе [124] отмечено, что такой подход не гарантирует полностью избавление от ошибок точности, поэтому в ней рассмотрено применение рациональных координат, а для сокращения количества пар конфликтующих ячеек применяется предобработка сетки.

При рассмотрении удаления самопересечений сетки отметим основные особенности решения этой задачи. Самопересечения могут состоять из произвольных ломаных (в том числе пересекающихся) и плоских объектов (не исключено наложение ячеек). Сетка не может рассматриваться как простая, локальная коррекция не обеспечивает гарантированного успеха приведения сетки к простому виду. Так как ячейки сетки имеют определенные направления внешней нормали, то может быть определена внешняя поверхность сетки, разделяющая пространство на внутреннюю и внешнюю области, при этом во внешней области не будет содержаться ни одной ячейки, а все лишние ячейки будут находиться во внутренней области и являться претендентами на удаление. Наличие определенных внешних нормалей ячеек позволяет использовать это для развития нового метода поиска и удаления самопересечений, так как требуется поиск только самопересечений, находящихся на внешней поверхности сетки, что может быть использовано для повышения производительности вычислений [279].

### 2.8.1 Адаптация сетки

Все рассмотренные до этого в работе методы перестроения поверхностной сетки объединяет одно сходство – они сохраняют количество элементов сетки (узлы, ребра, ячейки) и связи между ними. Несмотря на некоторые специальные методы по предотвращению конфликтов между ячейками сетки и методы сглаживания, после формирования новой поверхности возможно возникновение самопересечений, появление ячеек неправильной формы, а также неравномерное распределение ячеек в сетке по размеру. Пока опустим самопересечения сетки и будем рассматривать только вопросы, касающиеся формы и размера ячеек.

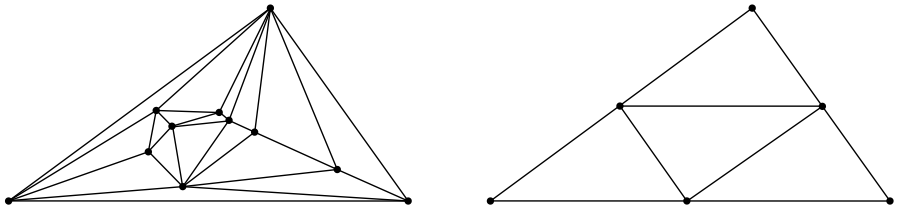

Рис. 2.30. Разбиение ячейки на более мелкие треугольники

Рассмотрим операцию разбиения ячейки на более мелкие. В качестве примера такого разбиения можно рассмотреть триангуляцию Делоне по заданному набору точек разбиения внутри ячейки (рис. 2.30 слева) [280]. Триангуляция Делоне обладает свойством максимизировать минимальный угол из всех углов треугольников, образованных в результате триангуляции. Таким образом, триангуляция Делоне позволяет избежать треугольников с очень острыми углами, возникновение которых может привести к нестабильности вычислений на поверхностной расчетной сетке. Стоит отметить, что разбиение ячейки можно производить с сохранением локальной кривизны сетки, как это показано в [281], но мы этот вариант рассматривать не будем, а будем производить разбиение исключительно в плоскости ячейки. Если точка разбиения находится не внутри ячейки, а на ее ребре, то разбить придется также вторую инцидентную ячейку этого ребра (если такая есть). Иногда для уменьшения размера требуется просто разбить ячейку на более мелкие без задания точек разбиения. В этом случае необходимо следить за качеством результирующих ячеек. Для определения качества ячейки треугольной формы $f = ABC$ можно пользоваться простым показателем качества $Q(f) = \frac{4\sqrt{3}S_f}{|\overline{AB}|^2+|\overline{BC}|^2+|\overline{AC}|^2}$, где $Q(f) = 1$ соответствует идеальному случаю равносторонней ячейки, а $Q(f) = 0$ – худший случай для ячеек с нулевой площадью [282]. Для выполнения разбиения ячейки на более мелкие с сохранением их качества можно просто выполнить разбиение по серединам всех ее сторон (рис. 2.30 справа) либо на большее количество частей аналогично [283].

В задаче удаления самопересечений поверхностной неструктурированной сетки важна специфическая операция дробления ячейки – дробление по фик-

сированному множеству отрезков (для простой сетки) и по фиксированному множеству отрезков и точек (в общем случае). На рис. 2.31 слева показана ячейка, которая пересекается с расчетной сеткой по четырем отмеченным траекториям – ломаным (некоторые из этих траекторий пересекаются друг с другом). Естественно, при выполнении триангуляции этой ячейки не должно появляться новых отрезков, которые пересекают один из отрезков какой-либо траектории пересечения с сеткой. Такая операция может быть выполнена с помощью модификации триангуляции Делоне – триангуляции Делоне с ограничениями [284, 285]. Если мы предполагаем, что при удалении самопересечений сетки требуется выполнять дробление ячейки по небольшому количеству точек и отрезков, то триангуляцию можно выполнять по наиболее простому жадному алгоритму с помощью последовательного добавления новых отрезков в триангуляцию. На рис. 2.31 приведен пример триангуляции по множеству отрезков, а на рис. 2.32 – по множеству отрезков и точек.

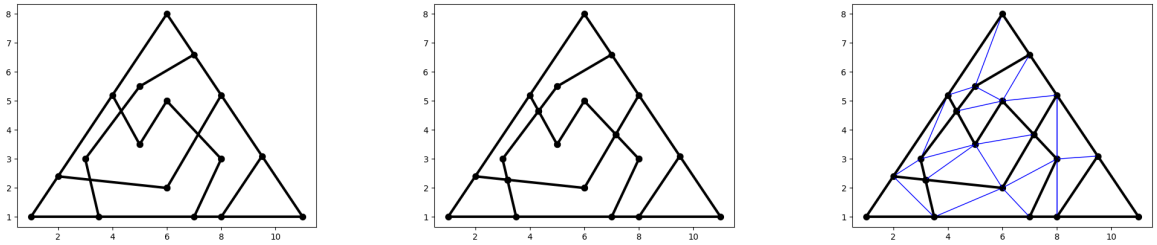

Рис. 2.31. Разбиение ячейки на более мелкие
по множеству фиксированных отрезков

Опишем триангуляция ячейки. Сначала ищутся точки пересечения траекторий друг с другом и выполняется дробление траекторий по этим точкам (рис. 2.31 в центре). После этого все отрезки траекторий считаются фиксированными и входят в финальную триангуляцию. Для завершения триангуляции к ним может понадобится добавить другие отрезки, соединяющие между собой некоторые точки траекторий (на рис. 2.31 справа отмечены синим). Такую операцию дробления ячейки будем использовать в дальнейшем. В общем случае триангуляцию требуется выполнять по множеству фиксированных отрезков и точек, примеры триангуляций показаны на рис. 2.32.

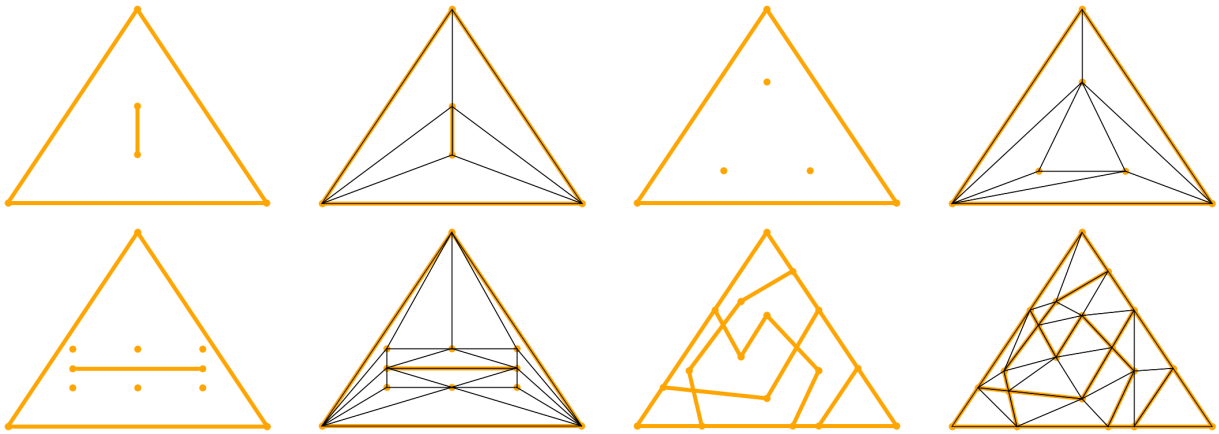

Рис. 2.32. Разбиение ячейки на более мелкие по множеству фиксированных отрезков и точек

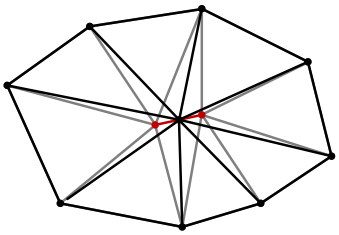

Рис. 2.33. Удаление короткого ребра

Вторая операция, которая необходима для адаптации сетки, связана с огрублением. Достаточно часто во время выполнения триангуляции по множеству заданных точек могут появляться ячейки с низким показателем качества (у них могут присутствовать либо слишком острые углы, либо близкий к развернутому угол). Если в ячейке присутствует угол, близкий к развернутому, то избавиться от него поможет разбиение по наибольшей стороне (при этом в качестве точки разбиения следует выбрать основание высоты, опущенной из противолежащего узла). Если же в треугольнике присутствуют лишь близкие к острым углы, то значит в нем есть слишком короткая сторона, которая может быть удалена. При удалении ребра $AB$ удаляются обе инцидентные ей грани, узлы $A$ и $B$ соединяются в единый узел $A'$, а все ребра из множества $\mathscr{E}(A) \cup \mathscr{E}(B)$ перенаправляются на узел $A'$ с учетом удаления кратных ребер (рис. 2.33). Эту операцию будем называть стягиванием ребра [286]. Путем применения стягивания наиболее коротких ребер в расчетной сетке можно добиться произвольной степени огрубления [287].

### 2.8.2 Поиск пересекающихся ячеек

Для поиска самопересечения необходимо определить, какое отношение двух ячеек сетки можно трактовать как самопересечение. Конечно, простое наличие общих точек у двух ячеек не может служить критерием самопересечения, так как у каждой ячейки есть смежные ячейки (то есть две ячейки могут иметь как общую вершину, так и общее ребро). Так как смежными могут быть только ячейки, имеющие общие инцидентные объекты (общую инцидентную вершину или общее инцидентное ребро), а все отношения инцидентности зафиксированы в расчетной сетке, то не представляет труда отделить пересечение ячеек по общему инцидентному объекту от самопересечения. Отметим, что две смежные ячейки, имеющие одну общую вершину, могут быть пересекающимися. Аналогично две смежные ячейки, имеющиеся общее ребро, также могут пересекаться (в случае их наложения).

В общем случае для идентификации всех фактов самопересечения сетки требуется проанализировать все пары ячеек и проверить пересечение ячеек в паре (то есть проверить пересечение каждой ячейки с каждой). Так как прямой перебор всех пар ячеек имеет квадратичную сложность по количеству ячеек, то его использование на крупных сетках невозможно. В этом случае целесообразно выполнять поиск пар пересекающихся ячеек с помощью BVH-деревьев [288], в которых множество ячеек представлено в виде древовидной структуры, связанной с ограничением геометрических объектов прямоугольными параллелепипедами в пространстве.

Прямоугольный параллелепипед будем определять как ГМТ $P = (x, y, z)$ в пространстве, удовлетворяющих следующей системе неравенств:

$$\begin{cases} x_l \leq x \leq x_h, \\ y_l \leq y \leq y_h, \\ z_l \leq z \leq z_h. \end{cases} \tag{2.66}$$

Для такого прямоугольного параллелепипеда будем использовать обозначение $Box([x_l, x_h], [y_l, y_h], [z_l, z_h])$.

**Определение 2.7.** *Контейнером произвольного множества точек $M$ в пространстве, который будем обозначать $[M]$, будем называть наименьший прямоугольный параллелепипед, содержащий все эти точки:*

$$[M] = Box\left(\left[\min_{P\in M} P_x, \max_{P\in M} P_x\right], \left[\min_{P\in M} P_y, \max_{P\in M} P_y\right], \left[\min_{P\in M} P_z, \max_{P\in M} P_z\right]\right) \quad (2.67)$$

Контейнер можно рассматривать для произвольного множества точек. Будем использовать контейнеры для построения BVH-дерева сетки. Мы будем его использовать для ячейки расчетной сетки, а также для множества ячеек. Так как любой треугольник является выпуклой фигурой, то $[ABC] = [\{A, B, C\}]$, то есть для построения контейнера для треугольника достаточно рассмотреть только его вершины. При поиске пар пересекающихся ячеек будем использовать следующий факт: если два треугольника пересекаются, то пересекаются и их контейнеры: $ABC \cap A'B'C' \neq \emptyset \implies [ABC] \cap [A'B'C'] \neq \emptyset$. То есть если не пересекаются контейнеры двух треугольников, то не пересекаются и сами треугольники: $[ABC] \cap [A'B'C'] = \emptyset \implies ABC \cap A'B'C' = \emptyset$ (рис. 2.34).

**Определение 2.8.** *Два треугольника $ABC$ и $A'B'C'$ назовем потенциально пересекающимися, если пересекаются их контейнеры $[ABC] \cap [A'B'C'] \neq \emptyset$.*

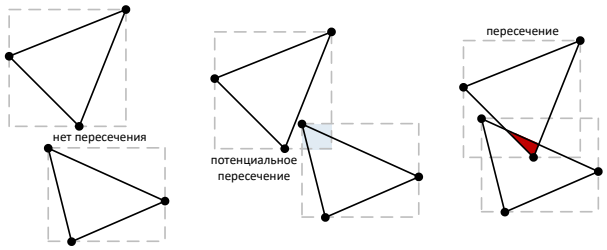


Рис. 2.34. Потенциальное пересечение треугольников в двумерном варианте

В отличие от анализа пары треугольников на пересечение, проверка на пересечение двух прямоугольных параллелепипедов со сторонами, параллельными координатным осям, является простой операцией. То есть на первом этапе будем искать пары потенциально пересекающихся ячеек сетки.

Для поиска потенциально пересекающихся ячеек построим бинарное BVH-дерево. Корнем это дерева является контейнер всех ячеек сетки. Рассмотрим

процедуру построения дерева, то есть разделения контейнера на два более мелких на примере двумерного случая, проиллюстрированного на рис. 2.35. Путь мы хотим разделить контейнер некоторого множества треугольников ($M$) по горизонтальной прямой на два множества: верхнее ($U$) и нижнее ($D$). Тогда в верхнее множество попадут все треугольники, лежащие выше прямой, либо пересекающие ее. Аналогично в нижнее множество попадут все треугольники, лежащие ниже прямой, либо пересекающие ее. После выделения верхнего и нижнего множества треугольников, для каждого из них строятся свои контейнеры, которые становятся дочерними контейнерами исходного. После этого получившиеся контейнеры можно делить дальше, используя произвольное направление разбиения ($X$, $Y$, $Z$).

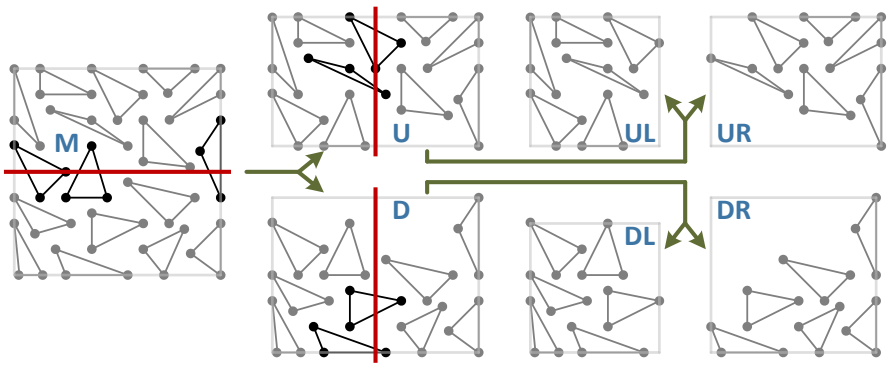


Рис. 2.35. Схема построения BVH-дерева

В частности, на рис. 2.35 продемонстрировано разбиение исходного контейнера по схеме $[M] \to \{[U], [D]\} \to \{\{[UL], [UR]\}, \{[DL], [DR]\}\}$. За одну операцию контейнер можно разбивать на произвольное количество дочерних контейнеров аналогичным образом. В качестве же направления разбиения контейнера целесообразно выбирать наиболее протяженное (вдоль которого длина контейнера имеет наибольшее значение). Проводить разбиение контейнеров можно произвольное количество раз, листовые контейнеры в построенном дереве могут содержать произвольное количество треугольников, вплоть до одного треугольника на контейнер. Если при разбиении контейнера по любому направлению в один из дочерних контейнеров попадают все ячейки этого контейнера, то дальнейшее разбиение невозможно.

Введем следующие обозначения. Пусть $M$ и $M'$ – два множества треугольников, $K = [M]$ и $K' = [M']$ – их контейнеры. Обозначим через $\mathscr{P}(K, K')$ множество пар потенциально пересекающихся треугольников $(T, T')$ таких, что $T \in M$, $T' \in M'$. Через $\mathscr{C}(K)$ обозначим множество дочерних контейнеров для контейнера $K$. Тогда множество пар потенциально пересекающихся треугольников может быть вычислено рекурсивно следующим образом:

$$\mathscr{P}(K, K') = \begin{cases} \bigcup\limits_{\substack{L \in \mathscr{C}(K) \\ L' \in \mathscr{C}(K')}} \mathscr{P}(L, L'), & \text{если } \mathscr{C}(K) \neq \emptyset,\ \mathscr{C}(K') \neq \emptyset \\ \bigcup_{L \in \mathscr{C}(K)} \mathscr{P}(L, K'), & \text{если } \mathscr{C}(K) \neq \emptyset,\ \mathscr{C}(K') = \emptyset \\ \mathscr{P}(K', K), & \text{если } \mathscr{C}(K) = \emptyset,\ \mathscr{C}(K') \neq \emptyset \\ \{(T, T') : T \in M,\ T' \in M', \\ \qquad T \neq T',\ [T] \cap [T'] \neq \emptyset\}, & \text{если } \mathscr{C}(K) = \emptyset,\ \mathscr{C}(K') = \emptyset \end{cases} \tag{2.68}$$

Построенное BVH-дерево позволяет сократить количество проверяемых потенциальных пересечений [289], так как если $K \cap K' = \emptyset$, $L$ – дочерний контейнер для $K$, а $L'$ – для $K'$, то $L \cap L' = \emptyset$. Для поиска пар потенциально пересекающихся треугольников из множества всех треугольников расчетной сетки $M$ нужно найти пересечение этого контейнера с самим собой $\mathscr{P}([M], [M])$. Сложность нахождения всех пар потенциально пересекающихся треугольников составляет $n \log n$, где $n$ – количество треугольников в сетке.

Чтобы определить, действительно пересекаются ли два потенциально пересекающихся треугольника, рассмотрим задачу пересечения треугольника и отрезка в пространстве. Пусть в пространстве задан треугольник $Tri(A, B, C)$ и отрезок $Segm(P, Q)$. Для поиска точек пересечения этого треугольника и отрезка требуется найти решение следующей системы уравнений

$$\begin{cases} x_A + (x_B - x_A)\beta + (x_C - x_A)\gamma = x_P + (x_Q - x_P)\alpha, \\ y_A + (y_B - y_A)\beta + (y_C - y_A)\gamma = y_P + (y_Q - y_P)\alpha, \\ z_A + (z_B - z_A)\beta + (z_C - z_A)\gamma = z_P + (z_Q - z_P)\alpha \end{cases} \tag{2.69}$$

при ограничениях $0 \leq \alpha \leq 1$, $\beta \geq 0$, $\gamma \geq 0$, $\beta + \gamma \leq 1$.

Система уравнений (2.69) может не иметь решений, может иметь ровно одно решение (если прямая $Line(P, \overline{PQ})$ пересекает плоскость треугольника) либо имеет бесконечное количество решений (если прямая $Line(P, \overline{PQ})$ лежит в плоскости треугольника).

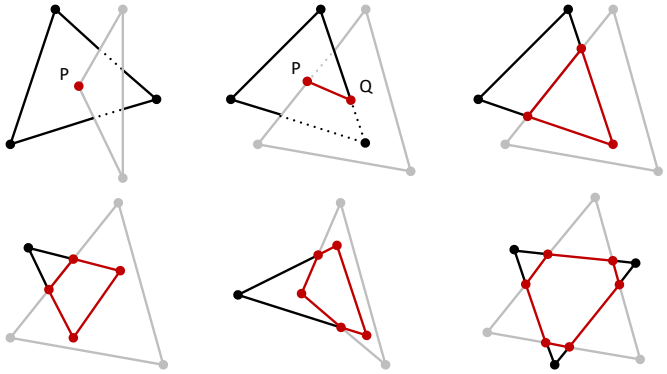


Рис. 2.36. Пересечение двух треугольников в пространстве

Пусть теперь в пространстве заданы два треугольника. Так как треугольник является выпуклой фигурой, то пересечение двух треугольников либо пусто, либо также является выпуклой фигурой (это может быть выпуклая фигура с количеством вершин от 1 до 6, рис. 2.36).

Вершины области пересечения двух треугольников это точки пересечения трех сторон первого треугольника со вторым треугольником и наоборот. То есть для поиска вершин области пересечения двух треугольников в общем случае нужно решить 6 задач пересечения треугольника с отрезком в трехмерном пространстве.

При нахождении пересечений двух треугольников возможны ошибки определения пересечений из-за точности представления координат вершин с помощью вещественных чисел. Так возможно возникновение ошибок определения пересечений в случае наложения отдельных объектов (вершин, сторон, самих треугольников), нахождения вершины треугольника на стороне или в плоскости другого треугольника и других ситуаций. В работе [121] описан подход, в котором некоторые ошибки, связанные с вычислениями с использованием вещественных чисел, могут быть устранены с помощью обнаружения некорректных пересечений прямой со сторонами треугольника.

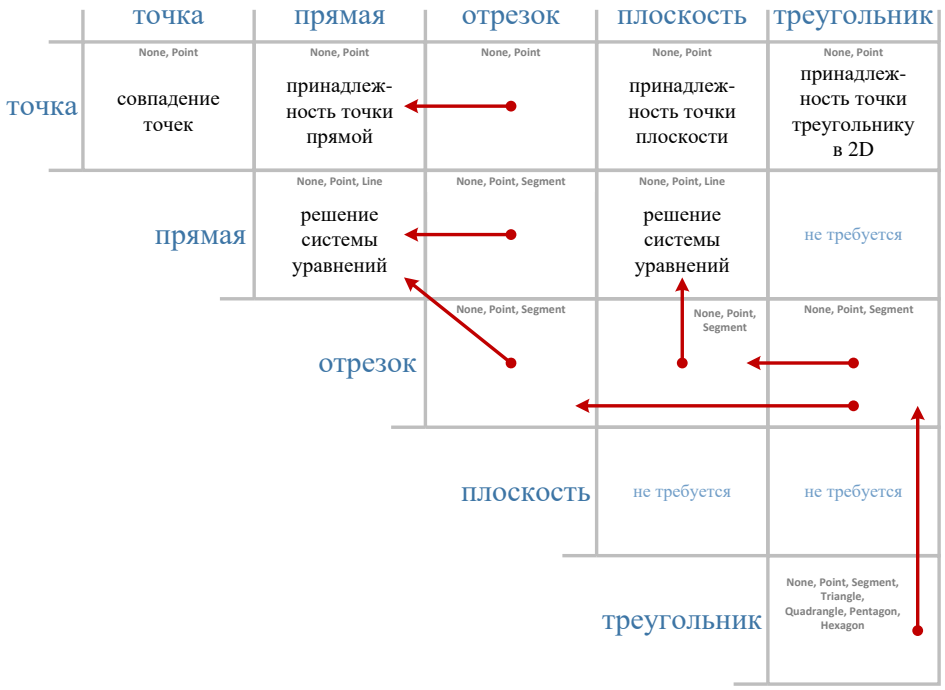


Рис. 2.37. Пересечение геометрических объектов

Другим подходом, с помощью которого можно полностью избежать проблем точности, связанных с использованием вещественных координат, является представление координат узлов сетки с помощью рациональных чисел. Для решения задачи поиска самопересечений поверхностной неструктурированной сетки используются задачи определения пересечений следующих объектов: точка, прямая, отрезок, плоскость, треугольник (рис. 2.37). Поиск пересечений всех этих объектов на плоскости и в пространстве являются стандартными задачами вычислительной геометрии и решаются с использованием только действий сложения, вычитания, умножения и деления. Таким образом, поиск пересечения всех перечисленных объектов решается в рациональных числах без потери точности. Для использования этого подхода координаты узлов поверхностной сетки необходимо представить в виде рациональных чисел. Основным недостатком использования вычислений в рациональных числах является более медленная обработка по сравнению с использованием вещественных чисел, однако полное избавление от ошибок точности вычислений компенсирует это замедление.

### 2.8.3 Обход внешней поверхности

Для удаления самопересечений используется стратегия обхода внешней поверхности результирующей сетки с поиском точек и отрезков пересечения обходимых ячеек и их дробления по мере необходимости. На рис. 2.38 слева показана сетка до возникновения самопересечения (в двумерном случае). На том же рисунке справа показана сетка после возникновения самопересечения и обозначена ее внешняя поверхность. Все ячейки расчетной сетки можно разделить на три категории (рис. 2.38).

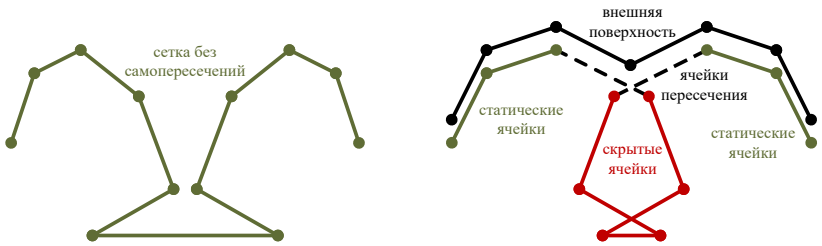


Рис. 2.38. Сетка до самопересечения (слева), категории ячеек при самопересечении сетки (справа)

**Определение 2.9.** *Статическими будем называть ячейки, которые не пересекаются ни с какими другими ячейками, и которые должны стать частью итоговой сетки (будем считать, что в любой момент времени у нас есть возможность указать хотя бы одну статическую ячейку).*

**Определение 2.10.** *Скрытыми будем называть ячейки, которые не пересекаются ни с какими другими ячейками, но которые не должны стать частью итоговой сетки (ячейки внутренних петель самопересечения поверхности, которые должны быть удалены).*

**Определение 2.11.** *Ячейками пересечения (или конфликтными ячейками) будем называть ячейки, которые пересекаются с какими-то другими.*

Как видно из рис. 2.38 справа, при возникновении самопересечения область скрытых ячеек сама содержит ячейки, пересекающиеся с другими ячейками.

Однако, так как вся скрытая область должна быть удалена после обхода внешней поверхности сетки, то такие пересечения вовсе не требуется обрабатывать, что позволяет избежать лишних вычислений.

Рассмотрим подход к удалению самопересечений, при котором ячейки пересечения дробятся на более мелкие по всем точкам и отрезкам пересечения [121]. В качестве примера на рис. 2.39 слева показаны два треугольника, которые пересекаются по отрезку (две точки пересечения). После выполнения дробления получаем конструкцию, показанную на рис. 2.39 справа. В общем случае точек, по которым необходимо разбивать ячейку, может быть сколь угодно много, и дробление должно выполняться с помощью выполнения триангуляции по этим точкам, причем некоторые ребра триангуляции должны быть фиксированы, как было отмечено ранее.

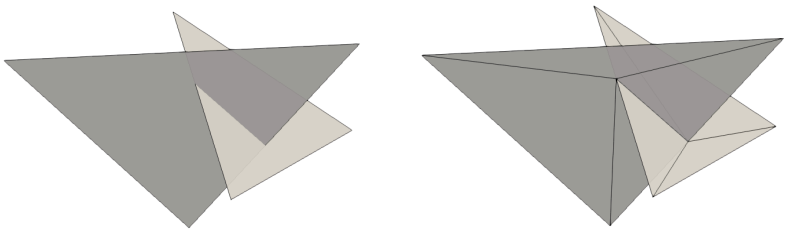

Рис. 2.39. Разбиение треугольника по точкам пересечения с одним фиксированным отрезком

Обход внешней поверхности сетки начинается с произвольной статической ячейки (рис. 2.40 сверху). Далее выполняется обход ячеек сетки, в котором переход между ячейками выполняется через общее ребро. Обход продолжается, пока не будет встречена ячейка пересечения. При попадании в ячейку пересечения выполняется поиск всех ее отрезов и точек пересечения, осуществляется ее дробление, а также дробление всех пересекающихся с ней ячеек (рис. 2.40 в центре). После дробления обход продолжается (рис. 2.40 внизу).

После дробления ячеек по отрезкам и точкам пересечения между ячейками сетки могут остаться только следующие виды отношений: две ячейки не имеют общих точек, две ячейки имеют одну общую вершину, две ячейки имеют одно общее ребро. При этом в сетке появляются ребра, имеющие более двух инцидентных ячеек. Для простой сетки верно то, что если ребро имеет более

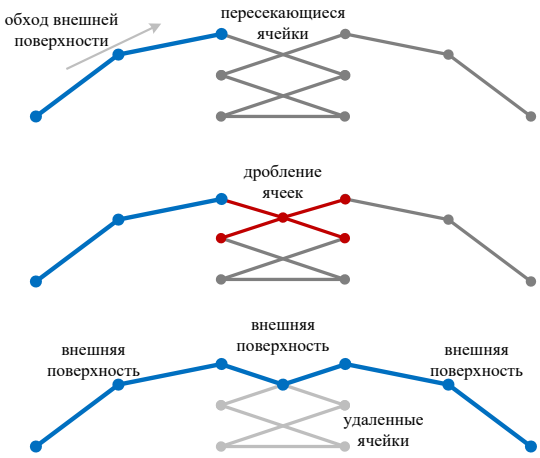


Рис. 2.40. Обход внешней поверхности сетки с дроблением: начало обхода со статической ячейки (сверху), дробление ячеек пересечения (в центре), удаление скрытых ячеек (внизу)

двух инцидентных ячеек, то это количество в точности равно 4, причем эти 4 инцидентные ячейки попарно лежат в одной плоскости (первая и третья в порядке обхода вокруг общего ребра лежат в одной плоскости, вторая и четвертая также лежат в одной плоскости).

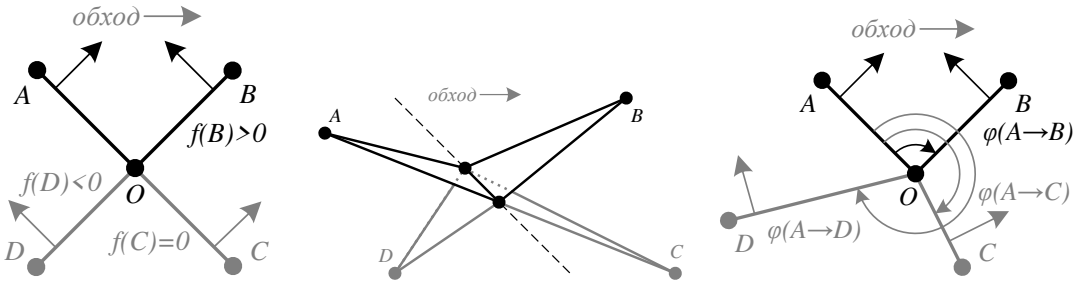


Рис. 2.41. Обход внешней поверхности при устранении самопересечений: простая сетка (слева), схема обхода (в центре), общий случай (справа)

Рассмотрим процедуру выбора следующей ячейки обхода сетки для ребер, имеющих количество инцидентных ячеек, больше двух (рис. 2.41 в центре). На этом рисунке обозначено направление обхода слева направо, вслед за ячейкой, инцидентной вершине $A$, в результирующую сетку должна войти ячейка, инцидентная вершине $B$. Рассмотрим эту процедуру для простых сеток, а также для сеток в общем виде. Для простоты будем рассматривать задачу в проекции на плоскость, перпендикулярную ребру.

Рассмотрим случай простой сетки (рис. 2.41 слева). Пусть уже известно, что ячейка, инцидентная вершине $A$, помечена, а ее внешняя нормаль равна $\overline{n}_A$. Из оставшихся трех ячеек (инцидентных вершинам $B$, $C$, $D$ соответственно) необходимо выбрать только одну для продолжения обхода сетки, а остальные удалить. Так как сетка простая, то $\angle AOC = \angle BOD = \pi$, а значит $(\overline{n}_A, \overline{OB}) > 0$, $(\overline{n}_A, \overline{OC}) = 0$, $(\overline{n}_A, \overline{OD}) < 0$. Таким образом, из всех ячеек, необходимо выбрать ту, для которой значение $(\overline{n}_A, \overline{OP})$ максимально.

Теперь рассмотрим сетку в общем виде. В этом случае количество ячеек, инцидентных рассматриваемому ребру, может быть произвольным больше двух, а про значение функции $(\overline{n}_A, \overline{OP})$ сказать ничего нельзя. Для выбора следующей ячейки для обхода сетки будем поворачивать текущую ячейку вокруг рассматриваемого ребра в направлении $\overline{n}_A$ до совпадения с первой ячейкой (рис. 2.41 справа). Эта первая ячейка и должна быть выбрана в качестве следующей для продолжения обхода. Если через $\phi(A \to P)$ обозначить угол поворота исходной ячейки в направлении $\overline{n}_A$ до ячейки, инцидентной вершине $P$, то в качестве следующей ячейки для обхода необходимо выбрать ту, для которой $\phi(A \to P)$ будет минимально. При этом

$$\phi(A \to P) = \begin{cases} \arccos \frac{(\overline{OA}, \overline{OP})}{|\overline{OA}| \cdot |\overline{OP}|}, \text{если } (\overline{n}_A, \overline{OP}) \geq 0, \\ 2\pi - \arccos \frac{(\overline{OA}, \overline{OP})}{|\overline{OA}| \cdot |\overline{OP}|}, \text{если } (\overline{n}_A, \overline{OP}) < 0. \end{cases} \tag{2.70}$$

Объединяя вместе оба рассмотренных случая, получаем критерий выбора следующей для обхода ячейки: должна быть выбрана ячейка, для которой значение показателя $f(P)$ максимально, где $f(P) = (\overline{n}_A, \overline{OP})$ для простых сеток и $f(P) = -\phi(A \to P)$ в общем случае.

После завершения обхода сетки все помеченные ячейки считаются ячейками результирующей поверхности, остальные ячейки удаляются.

Дробление ячейки выполняется только в случае ее обхода. Ячейки, которые находятся под внешней поверхностью и пересекаются с другими ячейками, не обрабатываются, что повышает производительность нахождения самопересечений в случае необходимости удаления обширных внутренних областей.

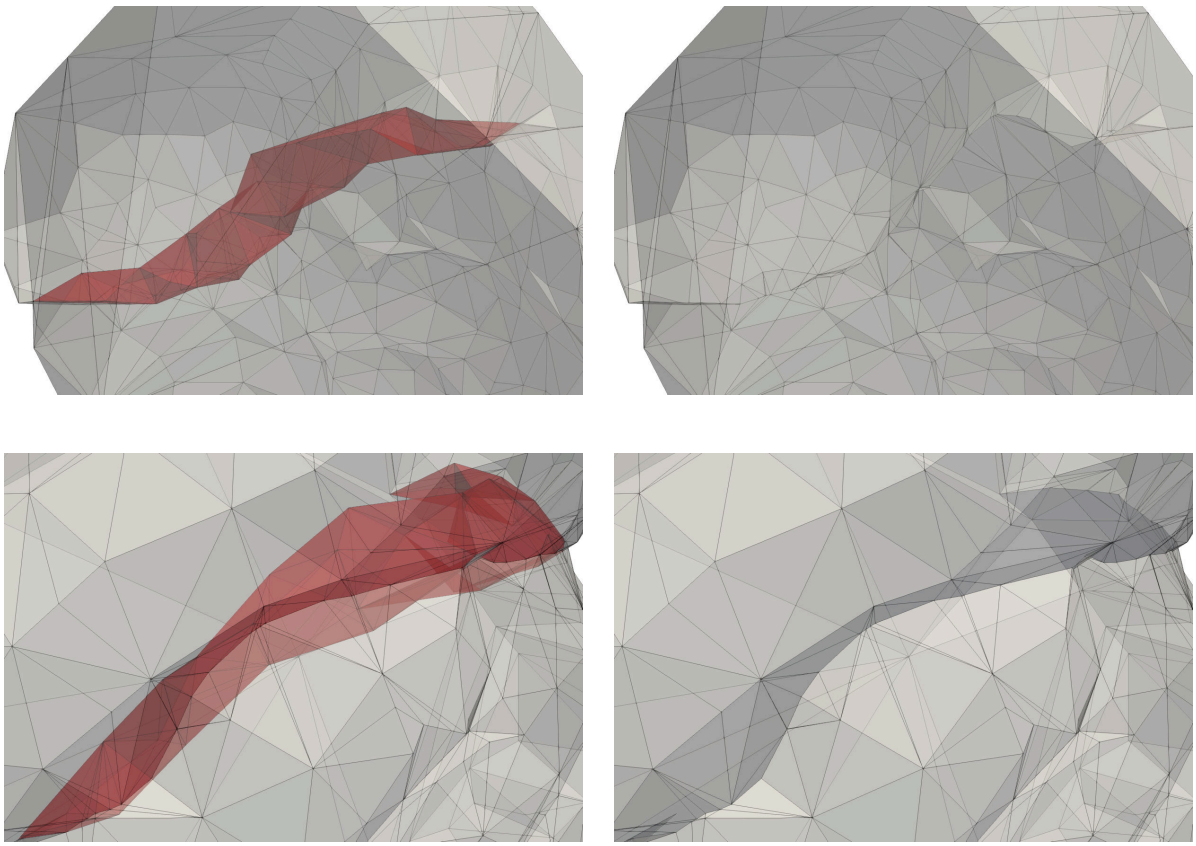

Рис. 2.42. Демонстрация удаления самопересечений

На рисунке рис. 2.42 проиллюстрированы примеры устранения самопересечений сетки в виде скрытых петель. После удаления скрытых ячеек результирующая расчетная сетка снова становится корректной, удовлетворяет соотношениям (2.6) и может быть использована для дальнейших вычислений.

Отдельным видом поверхностных неструктурированных сеток, используемых в расчетах, является псевдотрехмерный профиль – незамкнутая расчетная сетка, полученная из двумерного профиля, представляющая собой ленту шириной в одну расчетную ячейку.

Поиск точек пересечения ячеек псевдотрехмерного профиля не может быть выполнен точно даже с использованием рациональных координат. Однако, с псевдотрехмерным профилем удобно работать, если принудительно выровнять его положение по одной из координат, например выровнять его параллельно плоскости $OXY$, повернув таким образом, чтобы вершины сетки имели координаты $(x, y, 0)$ или $(x, y, d)$, где $d$ – ширина профиля. К такому профилю может быть применен предложенный метод удаления самопересечений.

### 2.8.4 Алгоритмы метода удаления самопересечений поверхностной неструктурированной сетки, основанного на обходе ее внешней поверхности

Для поиска пар потенциально пересекающихся ячеек используется бинарное BVH-дерево с рекурсивным разделением пополам по одному из направлений, параллельному осям координат $OX$, $OY$, $OZ$ [290]. При этом для ограничивающих объемов используется $\varepsilon$-превышение (к максимальному значению координаты прибавляется $\varepsilon$, от минимального значения координаты отнимается $\varepsilon$) для избежания ошибок, связанных с точностью вычислений. Разделение объема выполняется по тому направлению, которое обеспечивает наиболее равномерное распределение ячеек по дочерним объемам. Деление заканчивается, когда охватывающий объем не может быть разделен ни по одному направлению. Поиск пар потенциально пересекающихся ячеек ищется глобально для всей сетки перед началом обхода внешней поверхности.

Для определения пересечений геометрических примитивов используются рациональные координаты [291]. Поиск пересечений примитивов точка-отрезок, отрезок-отрезок, точка-треугольник, отрезок-треугольник, треугольник-треугольник выполняется без потери точности, и результатом таких пересечений является точка, отрезок, либо набор отрезков, составляющих выпуклую плоскую фигуру. Геометрические объекты считаются неизменяемыми, для каждого типа примитивов используется кэширование во избежание создания дублирующих объектов. Для линейных элементов сетки создаются несущие геометрические примитивы (для отрезка – содержащая его прямая, для треугольника – плоскость) для быстрого определения лежащих на одной прямой отрезков и лежащих в одной плоскости треугольников.

Для триангуляции ячейки по множеству точек и отрезков используется жадный алгоритм триангуляции с ограничениями [284]. На первом шаге алгоритма в триангуляцию сначала добавляются все отрезки, являющиеся ограничениями (фиксированные отрезки), в случае пересечения они дробятся по всем точкам пересечения. На втором шаге алгоритма создается список

всех возможных отсортированных по длине отрезков с концами в точках триангуляции, после чего производится попытка последовательного добавления отрезков в итоговую триангуляцию, начиная с кратчайшего.

Алгоритм обхода внешней поверхности состоит из следующих шагов:

1. Строится BVH-дерево ячеек для ускорения поиска потенциальных пересечений. Сложность построения $O(n \log n)$, где $n$ – количество ячеек. Переход к шагу 2.
2. Выполняется глобальный поиск всех пар потенциально пересекающихся ячеек согласно (2.68). Для каждой ячейки $F$ инициализируется список потенциально пересекающихся с ней ячеек $P(F)$. Сложность $O(n \log n)$. Переход к шагу 3.
3. Начинается обход внешней поверхности в ширину по дуальному графу сетки, начиная с любой статической ячейки. Переход к шагу 4.
4. Если для текущей ячейки $F$ выполняется $P(F) = \emptyset$, то переход к шагу 5, иначе переход к шагу 6.
5. Текущая ячейка помечается. Если на следующую непомеченную ячейку обхода переход выполняется через ребро $E$ такое, что $|\mathscr{F}(E)| = 2$, то переходим на следующую ячейку. Если же переход выполняется через ребро $E$, для которого $|\mathscr{F}(E)| > 2$ (значит это ребро, входящее в пересечение), то выбираем следующую ячейку перехода в соответствии с (2.70) и переходим на нее. Переход к шагу 4. Если следующей ячейки для обхода нет, переход на шаг 7.
6. Поиск пересечений для ячейки $F$ с каждой ячейкой из $P(F)$ и дробление по точкам и отрезкам пересечения с помощью жадного алгоритма. Операция рекурсивно продолжается для всех ячеек из $P(F)$, для которых пересечение найдено. Текущей остается ячейка $F'$, имеющая с ячейкой $F$ общее ребро, через которое был осуществлен переход на ячейку $F$. Переход к шагу 5.
7. Конец обхода. Сложность обхода в среднем $O(n)$. Все помеченные ячейки формируют внешнюю поверхность. Ячейки, которые не были помечены в процессе обхода, классифицируются как лишние и удаляются из сетки.

Метод удаления самопересечений, основанный на обходе внешней поверхности, в применении к псевдотрехмерным профилям реализован в программном комплексе «Кристалл» [126, 127] и апробирован при решении практической задачи по моделированию процесса обледенения.

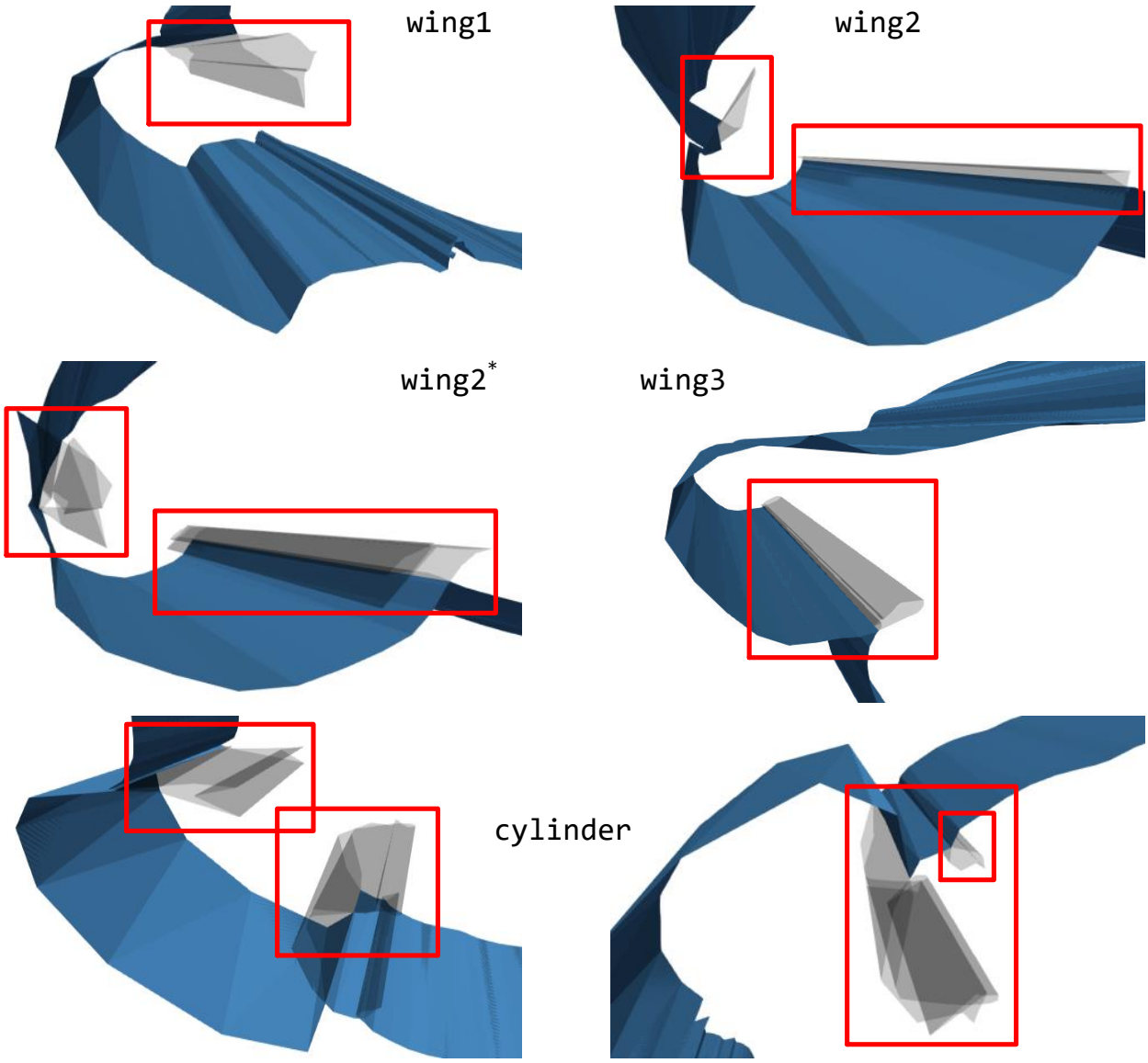


Рис. 2.43. Демонстрация удаления самопересечений тестовых сеток `wing1`, `wing2`, `wing3`, `cylinder`

В таблице 2.2 приведена статистика применения алгоритма удаления самопересечений сетки с помощью обхода внешней поверхности на тестовых примерах (демонстрация результатов удаления самопересечений приведена на рис. 2.43). Таблица разделена на две части, в первой части приведены данные работы алгоритма на псевдотрехмерных профилях, во второй части – на

Таблица 2.2. Статистика применения алгоритма удаления самопересечений сетки на тестовых примерах

| тест | элементы | $r_t$ (%) | $d$ | $r_b$ (%) | $r_c$ (%) | новые эл-ты (%) |
|---|---|---|---|---|---|---|
| `wing1` | 720/1440/720 | 2,92 | 4,14 | 96 | 100 | 102/105/101 |
| `wing2` | 720/1440/720 | 2,08 | 3,93 | 99 | 54 | 101/104/99 |
| `wing2`* | 720/1440/720 | 4,44 | 4,94 | 96 | 40 | 105/112/96 |
| `wing3` | 720/1440/720 | 0,56 | 4,00 | 96 | 100 | 100/101/97 |
| `cylinder` | 720/1440/720 | 9,72 | 5,60 | 86 | 46 | 113/129/90 |
| `dbl_small` | 84/240/160 | 23,75 | 4,79 | 83 | 100 | 120/175/122 |
| `dbl_mid` | 3844/11520/7680 | 3,83 | 4,24 | 81 | 100 | 106/110/86 |
| `dbl_big` | 4996/14976/9982 | 9,47 | 5,15 | 67 | 100 | 119/132/85 |
| `dbl_large` | 100K/300K/200K | 3,00 | 5,05 | 75 | 98 | 106/110/80 |

произвольных замкнутых поверхностных сетках. В столбце «тест» приведено название тестовой сетки (для сетки `wing2` было рассмотрено два варианта развития самопересечений), в столбце «элементы» – количество узлов, ребер и ячеек сетки. Приведенные в таблице показатели являются характеристиками тестов и иллюстрируют долю ячеек, для которых можно не выполнять их обработку ввиду нахождения этих ячеек ниже внешней поверхности сетки.

Значение $r_t$ представляет собой процентную долю количества ячеек, которые конфликтуют с другими ячейками сетки, то есть которые на самом деле имеют пересечения с другими ячейками.

В столбце $d$ приведены данные по средней степени дробления конфликтующих треугольников, это значение соответствуют среднему значению количества новых ячеек, получающихся при триангуляции одной ячейки исходной сетки. Значение $d$ является важной характеристикой, отражающей сложность возникающих пересечений, значение 4 соответствует наиболее простому пересечению ячеек, характерному для случая простой сетки (рис. 2.39). Низкое значение показателя $d$ оправдывает использование жадного алгоритма триангуляции ячеек, имеющего квадратичную сложность от количества точек разбиения, так как в большинстве случаев триангуляция представляет собой простое разбиение ячейки по отрезку.

Значение $r_b$ представляет собой процентную долю ячеек, не конфликтующих с другими ячейками (ячейки, которые не нужно дробить, или большие ячейки, big), которые расположены на внешней поверхности сетки и должны быть обработаны в процессе обхода.

Значение $r_c$ представляет собой процентную долю ячеек, конфликтующих с другими ячейками (ячейки, которые нужно дробить, conflict), которые расположены на внешней поверхностнсти и должны быть раздроблены в процессе обхода (чем ниже значение показателя $r_c$, тем для меньшего количества конфликтующих ячеек требуется выполнять триангуляцию, экономя таким образом вычислительные ресурсы).

В столбце «новые эл-ты» приведено количество вершин, ребер и ячеек результирующей сетки в процентном выражении относительно количества соответствующих элементов исходной сетки. Для большинства тестов можно наблюдать существенное увеличение количества узлов в сетке, что говорить о необхдимости ее огрубления после удаления самопересечений.

## 2.9 Сопряжение с объемными решателями

При проведении расчетов в областях со сложной геометрией необходимо выполнять сопряжение работающих в этих областях решателей с геометрией поверхностной неструктурированной сетки. В качестве примеров можно привести работу газодинамических решателей при расчете обтекания тел со сложной поверхностью (CFD) [292]. При этом в целях повышения быстродействия приложений рассматриваются решатели, работающие с трехмерными блочно-структурированными сетками. Существует большое разнообразие газодинамических решателей, работающих на объемных блочно-структурированных сетках. Например, для системы уравнений Эйлера [294] можно привести такие встречающиеся в работе решатели, как противопоточная схема Стегера-Уорминга [293], метод Годунова [294] на базе точного римановского решателя [295] и другие. Для решения системы уравнений Навье-Стокса можно отметить метод высокого разрешения RANS/ILES [296], работающий на криво-

линейных блочно-структурированных расчетных сетках и применяемый для расчета турбулентных течений. Он является комбинацией расчета осредненных по Рейнольдсу уравнений Навье-Стокса (Reynolds averaged Navier-Stokes, RANS) вблизи поверхности, а вдали от нее ILES (implicit LES) – метода моделирования крупных вихрей (large eddy simulation, LES) с неявной подсеточной моделью турбулентности (subgrid scale, SGS).

### 2.9.1 Пересечение поверхностной сетки с декартовой

Рассмотрим локальную геометрическую задачу, связанную с поиском пересечения поверхностной неструктурированной сетки с декартовой сеткой, состоящей из прямоугольных ячеек, стороны которых направлены вдоль осей координат. Такая задача возникает в различных приложениях, например, при использовании декартовой сетки в качестве вспомогательной подложки, при реализации метода погруженной границы или при использовании накладывающихся сеток.

Для нахождения пересечения поверхностной сетки с декартовой, необходимо найти попарное пересечение их ячеек, то есть пересечение треугольника и прямоугольного параллелепипеда в пространстве. Пусть в пространстве определены треугольник $Tri(A, B, C)$ и прямоугольный параллелепипед $Box([x_l, x_h], [y_l, y_h], [z_l, z_h])$. Требуется определить, имеют ли они общие точки (см. рис. 2.44).

Для решения этой задачи подставим $\overline{P}(\beta, \gamma)$ из (2.3) в систему неравенств (2.66), получив следующую систему неравенств с двумя переменными $\beta$ и $\gamma$:

$$\begin{cases} x_l \le x_A + \beta(x_B - x_A) + \gamma(x_C - x_A) \le x_h, \\ y_l \le y_A + \beta(y_B - y_A) + \gamma(y_C - y_A) \le y_h, \\ z_l \le z_A + \beta(z_B - z_A) + \gamma(z_C - z_A) \le z_h, \\ \beta \ge 0, \\ \gamma \ge 0, \\ \beta + \gamma \le 1. \end{cases} \tag{2.71}$$

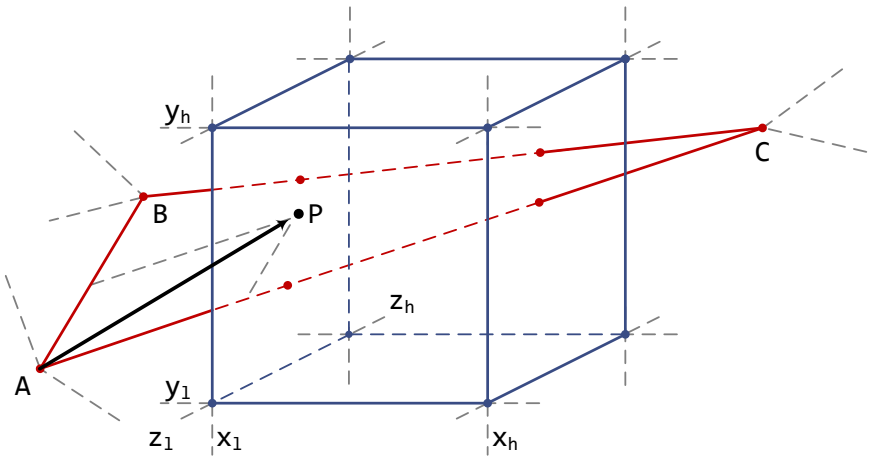


Рис. 2.44. Пересечение прямоугольного параллелепипеда и треугольника

Система (2.71) может быть решена методом свертывания конечных систем линейных неравенств [297]. Для применения метода свертывания систем линейных неравенств неравенства системы (2.71) необходимо привести к виду $k_\beta\beta + k_\gamma\gamma + k \le 0$, после чего получим следующую систему неравенств:

$$\begin{cases} (x_B - x_A)\beta + (x_C - x_A)\gamma + (x_A - x_h) \le 0, \\ (x_A - x_B)\beta + (x_A - x_C)\gamma + (x_l - x_A) \le 0, \\ (y_B - y_A)\beta + (y_C - y_A)\gamma + (y_A - y_h) \le 0, \\ (y_A - y_B)\beta + (y_A - y_C)\gamma + (y_l - y_A) \le 0, \\ (z_B - z_A)\beta + (z_C - z_A)\gamma + (z_A - z_h) \le 0, \\ (z_A - z_B)\beta + (z_A - z_C)\gamma + (z_l - z_A) \le 0, \\ -1 \cdot \beta + 0 \cdot \gamma \le 0, \\ 0 \cdot \beta + (-1) \cdot \gamma \le 0, \\ \beta + \gamma + (-1) \le 0. \end{cases} \tag{2.72}$$

Система неравенств (2.72) является системой с двумя переменными ($\beta$ и $\gamma$), поэтому после выполнения одного шага свертывания (деформации) она превратится в систему неравенств с одной переменной. Деформация системы для исключения из нее переменной $\beta$ выполняется следующим образом. Составляется новая система неравенств, в которую войдут все неравенства

системы (2.72) вида $k_\gamma\gamma + k \leq 0$, а каждая пара неравенств

$$\begin{aligned} k^1_\beta\beta + k^1_\gamma\gamma + k^1 \leq 0, \\ k^2_\beta\beta + k^2_\gamma\gamma + k^2 \leq 0. \end{aligned} \tag{2.73}$$

в которой коэффициенты при $\beta$ удовлетворяют неравенствам $k^1_\beta < 0$ и $k^2_\beta > 0$, войдет в деформированную систему неравенств в виде

$$(k^1_\gamma k^2_\beta - k^2_\gamma k^1_\beta)\gamma + (k^1 k^2_\beta - k^2 k^1_\beta) \leq 0. \tag{2.74}$$

Так как система (2.72) содержит 9 неравенств, из которых хотя бы в одном коэффициент при $\beta$ нулевой, не более чем в четырех — положительный, и не более чем в четырех – отрицательный, то в результате выполнения деформации получим систему, состоящую не более чем из 17 неравенств. Если деформированная система неравенств с одной переменной имеет решение, то исходные треугольник и прямоугольный параллелепипед имеют общие точки.

Для того, чтобы найти все ячейки объемной сетки, которые пересекаются с поверхностной сеткой, в общем случае необходимо проверить факт пересечения каждой объемной ячейки с каждой поверхностной ячейкой. В общем случае эта процедура является очень затратной, так как сетки могут содержать большое количество ячеек. Вместо этого для каждой ячейки поверхностной расчетной сетки сначала будем определять диапазон ячеек объемной сетки, с которыми в принципе возможно пересечение.

Пусть изначально объемная сетка, описывающая область $[X_l, X_h]\times[Y_l, Y_h]\times[Z_l, Z_h]$, размера $S_x \times S_y \times S_z$ разделена на одинаковые ячейки размера $s_x \times s_y \times s_z$, где $n_x = \frac{S_x}{s_x}$, $n_y = \frac{S_y}{s_y}$, $n_z = \frac{S_z}{s_z}$ ($n_x$, $n_y$, $n_z$ – количество ячеек по направлению $X$, $Y$, $Z$ соответственно).

Таким образом, объемная сетка представлена трехмерным массивом ячеек, координаты которых $(x_l, x_h, y_l, y_h, z_l, z_h)$ могут быть вычислены по индексам

$(i, j, k)$ в этом трехмерном массиве:

$$\begin{aligned} x_l(i) &= X_l + is_x, \quad x_h(i) = X_l + (i+1)s_x, \\ y_l(j) &= Y_l + js_y, \quad y_h(j) = Y_l + (j+1)s_y, \\ z_l(k) &= Z_l + ks_z, \quad z_h(k) = Z_l + (k+1)s_z. \end{aligned} \tag{2.75}$$

Для треугольника $ABC$, являющегося ячейкой поверхностной сетки, если он пересекает некоторую объемную ячейку, то его контейнер $[\{A, B, C\}] = Box([\tilde{x}_l, \tilde{x}_h], [\tilde{y}_l, \tilde{y}_h], [\tilde{z}_l, \tilde{z}_h])$ также пересекает эту ячейку. Для определения пересечения поверхностной ячейки со всеми ячейками объемной сетки достаточно проверить только те ячейки, с которыми пересекается контейнер рассматриваемого треугольника. Так как координаты анализируемых параллелепипедов могут быть записаны явно, то можно вычислить диапазоны индексов ячеек, которые требуется проверить на предмет пересечения с треугольником. Факт пересечения по координате $x$ двух отрезков $[x_l(i), x_h(i)]$ и $[\tilde{x}_l, \tilde{x}_h]$ можно записать в виде системы из двух неравенств

$$\begin{cases} x_l(i) \le \tilde{x}_h, \\ x_h(i) \ge \tilde{x}_l. \end{cases} \tag{2.76}$$

Преобразовав систему (2.76), и выполнив аналогичные действия для координат по $y$ и $z$, получим диапазоны индексов ячеек объемной сетки

$$\begin{cases} \dfrac{\tilde{x}_l - X_l}{s_x} - 1 \le i \le \dfrac{\tilde{x}_h - X_l}{s_x}, \\ \dfrac{\tilde{y}_l - Y_l}{s_y} - 1 \le j \le \dfrac{\tilde{y}_h - Y_l}{s_y}, \\ \dfrac{\tilde{z}_l - Z_l}{s_z} - 1 \le k \le \dfrac{\tilde{z}_h - Z_l}{s_z}. \end{cases} \tag{2.77}$$

Диапазон индексов из (2.77) содержит малую долю всех ячеек объемной сетки и эта доля тем меньше, чем меньше размер ячейки декартовой сетки, который обычно является параметром, который можно менять.

### 2.9.2 Метод погруженной границы

Для задач с областями со сложной геометрией [298], построение согласованной расчетной сетки может быть очень требовательной по ресурсам задачей. Альтернативой является использование метода погруженной границы (immersed boundary method, IBM) [80]. Этот метод позволяет использовать для расчетов несогласованную сетку и даже простую декартову сетку [299], что сильно упрощает и ускоряет проведение вычислений. Применение метода погруженной границы позволяет проводить расчеты на простых структурированных сетках [300], что также упрощает реализацию параллельных вычислений на большом количестве вычислительных узлов суперкомпьютера [301]. Тонким моментом метода является выполнение граничных условий на сложной границе, которое достигается путем модификации решаемой системы уравнений [302]. Можно выделить два основных подхода к разрешению граничных условий в методах погруженной границы, различающихся по способу выполнения расчетов на границе: задание граничных условий посредством внешних (источниковых) членов [82] и методы, использующие фиктивные (вспомогательные в расчетах) ячейки [86].

Рассмотрим подробнее метод погруженной границы с использованием фиктивных ячеек [87] на примере задачи обтекания тела со сложной геометрией [303]. Пусть в некоторой области пространства расположено тело, ограниченное сложной границей, представленной неструктурированной поверхностной сеткой. В охватывающей тело области пространства строится объемная сетка, ячейки которой разделяются на внешние (большей частью лежащие вне тела), внутренние (лежащие внутри тела и внутри которых не рассчитываются газодинамические параметры) и слой фиктивных ячеек, находящихся между внешними и внутренними. На каждой итерации расчетов для фиктивных ячеек требуется выполнить аппроксимацию газодинамических величин, чтобы эти фиктивные ячейки могли быть использованы для определения потоков между ними и соседними с внешними ячейками [84].

Аппроксимация газодинамических параметров фиктивных ячеек выполня-

ется на каждой итерации проведения расчетов, после для всех ячеек расчетной сетки кроме внутренних выполняется пересчет потоков между ячейками с использованием любого конечно-объемного метода. Обработка фиктивных ячеек является основной особенностью рассматриваемого метода, для вычисления газодинамических параметров фиктивных ячеек требуется использование аппроксимации скалярных и векторных величин по данным, взятым из близлежащих точек пространства и поверхности обтекаемого тела.

Сначала рассмотрим формулу линейной аппроксимации скалярной величины по заданным четырем точкам в пространстве. Пусть в пространстве определена скалярная величина $\phi = \phi(x, y, z)$ как функция от трех координат. Пусть известно значение этой величины в четырех точках пространства $\phi(x_0, y_0, z_0) = \phi_0$, $\phi(x_1, y_1, z_1) = \phi_1$, $\phi(x_2, y_2, z_2) = \phi_2$, $\phi(x_3, y_3, z_3) = \phi_3$. Требуется выполнить линейную аппроксимацию этой величины, то есть найти представление вида $\phi(x, y, z) = a_0 + a_1 x + a_2 y + a_3 z$, где коэффициенты $a_0$, $a_1$, $a_2$, $a_3$ находятся по известным значениям в четырех точках. Для двумерного случая описание этой задачи можно найти в [86], и в трехмерном варианте она выглядит аналогично. Для нахождения коэффициентов $a_0$, $a_1$, $a_2$, $a_3$ решается следующая система линейных уравнений $B_{<0123>} a = \phi_{<0123>}$, где $\phi_{<0123>}$ – это вектор-столбец $[\phi_0, \phi_1, \phi_2, \phi_3]^T$, $a$ – вектор-столбец искомых коэффициентов $[a_0, a_1, a_2, a_3]^T$, а матрица $B_{<0123>}$ имеет вид

$$B_{<0123>} = \begin{bmatrix} 1 & x_0 & y_0 & z_0 \\ 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \end{bmatrix}, \tag{2.78}$$

откуда можно найти коэффициенты $a$ по формуле $a = B^{-1}_{<0123>} \phi_{<0123>}$.

Следующим шагом рассмотрим аппроксимацию все той же скалярной величины $\phi = \phi(x, y, z)$ с использованием граничного условия Неймана. Только на этот раз пусть известно ее значение только в трех точках $\phi(x_1, y_1, z_1) = \phi_1$, $\phi(x_2, y_2, z_2) = \phi_2$, $\phi(x_3, y_3, z_3) = \phi_3$. Дополнительно в точке $(x_0, y_0, z_0)$ задано

условие, соответствующее граничному условию Неймана $\frac{\partial\phi}{\partial\overline{e}_0}(x_0, y_0, z_0) = \phi'_0$, где $\overline{e}_0$ – некоторое направление (граничное условие Неймана задается как производная по нормали к поверхности обтекаемого тела). При этом будем полагать, что $|\overline{e}_0| = 1$. Для заданной скалярной величины также требуется выполнить линейную аппроксимацию, то есть найти представление вида $\phi(x, y, z) = a_0 + a_1x + a_2y + a_3z$. Для двумерного случая описание этой задача может быть найдено в [86], рассмотрим ее для трехмерного случая.

Пусть компоненты вектора направления $\overline{e}_0$ равны $e_{0,x}$, $e_{0,y}$, $e_{0,z}$ соответственно. Граничное условие Неймана записывается в виде $\frac{\partial\phi}{\partial\overline{e}_0}(x_0, y_0, z_0) = \frac{\partial\phi}{\partial x}e_{0,x} + \frac{\partial\phi}{\partial y}e_{0,y} + \frac{\partial\phi}{\partial z}e_{0,z} = \phi'_0$.

Так как известен общий вид аппроксимации функции $\phi(x, y, z) = a_0 + a_1x + a_2y + a_3z$, то и ее частные производные можно выписать в явном виде $\frac{\partial\phi}{\partial x} = a_1$, $\frac{\partial\phi}{\partial y} = a_2$, $\frac{\partial\phi}{\partial z} = a_3$. Получаем систему линейных уравнений $B_{<0'123>}a = \phi_{<0'123>}$, где $\phi_{<0'123>}$ – вектор-столбец $[\phi'_0, \phi_1, \phi_2, \phi_3]^T$, $a$ – вектор-столбец коэффициентов $[a_0, a_1, a_2, a_3]^T$, матрица $B_{<0'123>}$ имеет вид

$$B_{<0'123>} = \begin{bmatrix} 0 & e_{0,x} & e_{0,y} & e_{0,z} \\ 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \end{bmatrix}, \tag{2.79}$$

откуда получим выражение для коэффициентов $a = B^{-1}_{<0'123>}\phi_{<0'123>}$.

Перейдем к аппроксимации векторных величин. Для двумерного случая решение приведено в [84], рассмотрим его в трехмерном случае. Рассмотрим векторную величину $\overline{v} = [v_x, v_y, v_z] = [v_x(x, y, z), v_y(x, y, z), v_z(x, y, z)]$, которая задана в трехмерном пространстве как функция от трех координат. Пусть известны ее значения в трех точках:

$$\begin{aligned} \overline{v}(x_1, y_1, z_1) &= \overline{v}_1 = [v_{1,x}, v_{1,y}, v_{1,z}], \\ \overline{v}(x_2, y_2, z_2) &= \overline{v}_2 = [v_{2,x}, v_{2,y}, v_{2,z}], \\ \overline{v}(x_3, y_3, z_3) &= \overline{v}_3 = [v_{3,x}, v_{3,y}, v_{3,z}]. \end{aligned} \tag{2.80}$$

Дополнительно в точке $(x_0, y_0, z_0)$ задано следующее условие: проекция вектора $\overline{v}_0 = \overline{v}(x_0, y_0, z_0)$ на направление $\overline{e}_0$ равна нулю, а производная составляющей, перпендикулярной направлению $\overline{e}_0$, по этому направлению также равна нулю. То есть $\overline{v}_0 = \overline{v}_0^n + \overline{v}_0^\tau$, $\overline{v}_0^n \parallel \overline{e}_0$, $\overline{v}_0^\tau \perp \overline{e}_0$, $|\overline{e}_0| = 1$, $|\overline{v}_0^n| = 0$, $\frac{\partial |\overline{v}^\tau|}{\partial \overline{e}_0}(x_0, y_0, z_0) = 0$.

Требуется определить значение векторной величины в некоторой точке $(x_G, y_G, z_G)$, то есть $\overline{v}_G = \overline{v}(x_G, y_G, z_G)$. При этом под направлением $\overline{e}_0$ будем подразумевать некоторую нормаль к поверхности обтекаемого тела, для которого выполняется расчет. Тогда логично называть составляющие вектора $\overline{v}_0^n$ и $\overline{v}_0^\tau$ нормальной и тангенциальной составляющими вектора $\overline{v}_0$ соответственно. Сначала запишем условие равенства нулю нормальной составляющей вектора $\overline{v}_0$, выражающееся в равенстве нулю скалярного произведения $(\overline{v}_0, \overline{e}_0)$.

Распишем скалярное произведение $(\overline{v}_0, \overline{e}_0)$ покомпонентно:

$$v_{0,x} e_{0,x} + v_{0,y} e_{0,y} + v_{0,z} e_{0,z} = 0. \tag{2.81}$$

Значения $v_{0,x}$, $v_{0,y}$, $v_{0,z}$ неизвестны, выразим их с помощью линейной интерполяции скалярной величины через точки $P_1 = (x_1, y_1, z_1)$, $P_2 = (x_2, y_2, z_2)$, $P_3 = (x_3, y_3, z_3)$, $P_G = (x_G, y_G, z_G)$:

$$\begin{cases} v_{0,x} = [1, x_0, y_0, z_0] B^{-1}_{<G123>} v_{<G123>,x}, \\ v_{0,y} = [1, x_0, y_0, z_0] B^{-1}_{<G123>} v_{<G123>,y}, \\ v_{0,z} = [1, x_0, y_0, z_0] B^{-1}_{<G123>} v_{<G123>,z}. \end{cases} \tag{2.82}$$

Подставляя полученные соотношения для $v_{0,x}$, $v_{0,y}$, $v_{0,z}$ в выражение скалярного произведения $(\overline{v}_0, \overline{e}_0) = 0$, получим:

$$[1, x_0, y_0, z_0] B^{-1}_{<G123>} \left( \begin{bmatrix} v_{G,x} \\ v_{1,x} \\ v_{2,x} \\ v_{3,x} \end{bmatrix} e_{0,x} + \begin{bmatrix} v_{G,y} \\ v_{1,y} \\ v_{2,y} \\ v_{3,y} \end{bmatrix} e_{0,y} + \begin{bmatrix} v_{G,z} \\ v_{1,z} \\ v_{2,z} \\ v_{3,z} \end{bmatrix} e_{0,z} \right) = 0. \tag{2.83}$$

Введем обозначения $[d_G, d_1, d_2, d_3] = [1, x_0, y_0, z_0]B^{-1}_{<G123>}$, тогда уравнение может быть переписано в виде

$$d_G(v_{G,x}e_{0,x} + v_{G,y}e_{0,y} + v_{G,z}e_{0,z}) + d_1(\overline{v}_1, \overline{e}_0) + d_2(\overline{v}_2, \overline{e}_0) + d_3(\overline{v}_3, \overline{e}_0) = 0, \quad (2.84)$$

или $v_{G,x}e_{0,x} + v_{G,y}e_{0,y} + v_{G,z}e_{0,z} = Q$, где $Q = -\frac{d_1(\overline{v}_1,\overline{e}_0)+d_2(\overline{v}_2,\overline{e}_0)+d_3(\overline{v}_3,\overline{e}_0)}{d_G}$.

Условие на тангенциальную составляющую $\overline{v}_0^{\tau}$ не будем записывать в явном виде. Вместо этого будем работать с проекциями вектора $\overline{v}_0$ на векторы, лежащие в плоскости, перпендикулярной вектору $\overline{e}_0$. Так как компоненты вектора $\overline{e}_0$ известны, то можно без труда найти векторы, перпендикулярные ему. Это будут, например, следующие векторы: $\overline{f}_1 = [-e_{0,y}, e_{0,x}, 0]$, $\overline{f}_2 = [-e_{0,z}, 0, e_{0,x}]$, $\overline{f}_3 = [0, -e_{0,z}, e_{0,y}]$.

Далее запишем выражения для линейной аппроксимации величин $(\overline{f}_1, \overline{v}_G)$, $(\overline{f}_2, \overline{v}_G)$, $(\overline{f}_3, \overline{v}_G)$ через точки $P_0$, $P_1$, $P_2$, $P_3$ с учетом граничного условия Неймана:

$$\begin{cases} -v_{G,x}e_{0,y} + v_{G,y}e_{0,x} = [1, x_G, y_G, z_G]B^{-1}_{<0'123>} \begin{bmatrix} 0 \\ -v_{1,x}e_{0,y} + v_{1,y}e_{0,x} \\ -v_{2,x}e_{0,y} + v_{2,y}e_{0,x} \\ -v_{3,x}e_{0,y} + v_{3,y}e_{0,x} \end{bmatrix}, \\ -v_{G,x}e_{0,z} + v_{G,z}e_{0,x} = [1, x_G, y_G, z_G]B^{-1}_{<0'123>} \begin{bmatrix} 0 \\ -v_{1,x}e_{0,z} + v_{1,z}e_{0,x} \\ -v_{2,x}e_{0,z} + v_{2,z}e_{0,x} \\ -v_{3,x}e_{0,z} + v_{3,z}e_{0,x} \end{bmatrix}, \\ -v_{G,y}e_{0,z} + v_{G,z}e_{0,y} = [1, x_G, y_G, z_G]B^{-1}_{<0'123>} \begin{bmatrix} 0 \\ -v_{1,y}e_{0,z} + v_{1,z}e_{0,y} \\ -v_{2,y}e_{0,z} + v_{2,z}e_{0,y} \\ -v_{3,y}e_{0,z} + v_{3,z}e_{0,y} \end{bmatrix}. \end{cases} \quad (2.85)$$

В системе уравнений (2.85) правые части могут быть непосредственно вычислены, обозначим их $T_{xy}$ , $T_{xz}$ и $T_{yz}$ соответственно и добавим в систему

уравнение для выражения $(\overline{v}_G, \overline{e}_0)$. Получим следующую систему:

$$\begin{bmatrix} e_{0,x} & e_{0,y} & e_{0,z} \\ -e_{0,y} & e_{0,x} & 0 \\ -e_{0,z} & 0 & e_{0,x} \\ 0 & -e_{0,z} & e_{0,y} \end{bmatrix} \begin{bmatrix} v_{G,x} \\ v_{G,y} \\ v_{G,z} \end{bmatrix} = \begin{bmatrix} Q \\ T_{xy} \\ T_{xz} \\ T_{yz} \end{bmatrix}. \tag{2.86}$$

Система (2.86) является переопределенной, и в качестве решения для $\overline{v}_G$ может быть взято его псевдорешение.

Первым шагом реализации МПГ с фиктивными ячейками является построение структурированной сетки и разметка ячеек, участвующих в расчетах. Целью разметки (классификации) является выделение из множества ячеек двух подмножеств: внешние (OUTER) ячейки – обычные ячейки для проведения вычислений, фиктивные (GHOST) ячейки – фиктивные ячейки для которых на каждом шаге расчетов требуется выполнять аппроксимацию газодинамических величин (рис. 2.45). Будем использовать наиболее простой вид сеток – декартову сетку с кубическими ячейками (рис. 2.46).

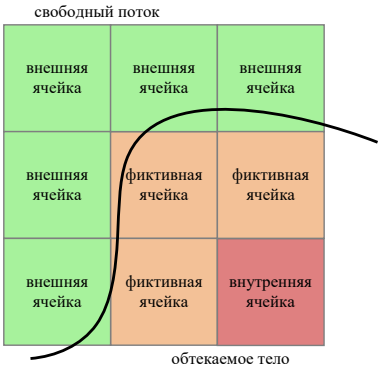


Рис. 2.45. Классификация ячеек

На каждой итерации для фиктивных ячеек должны быть пересчитаны газодинамические величины на основании данных близлежащих внешних ячеек, а также точки поверхности для аппроксимации граничного условия (для каждой фиктивной ячейки используется ближайшая к ней точка поверхности).

Рассмотрим последовательность действий, выполняемых на одной итерации расчетов с использованием МПГ для схемы Стерега-Уорминга из [293].

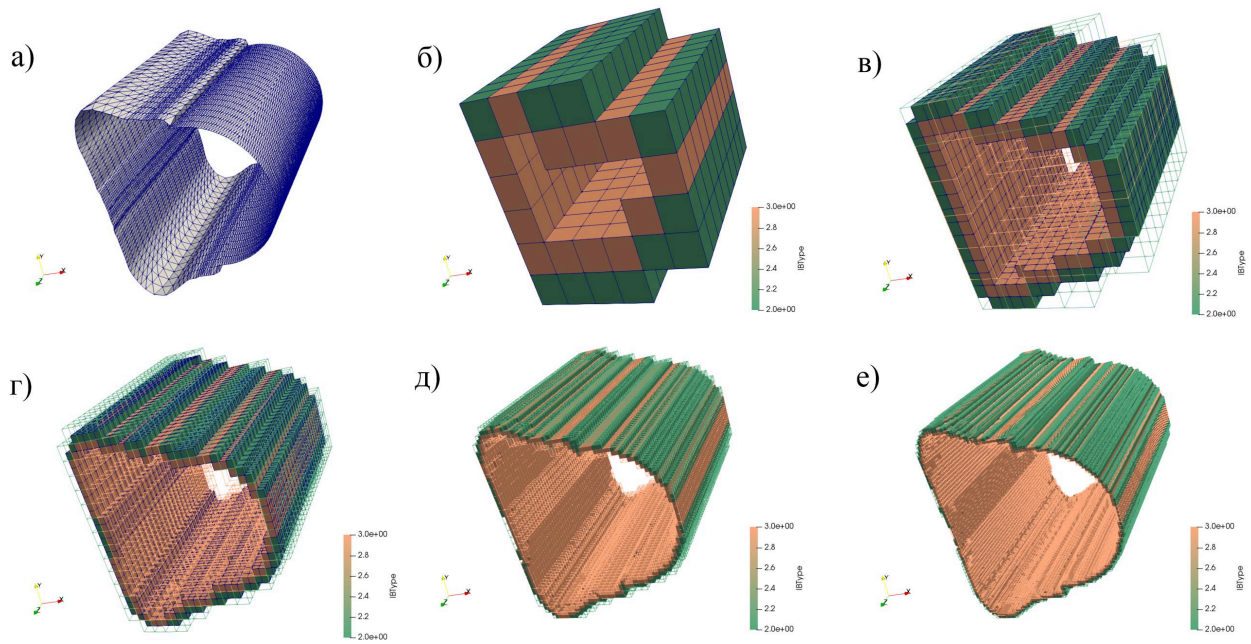


Рис. 2.46. Обстраивание поверхностной сетки декартовыми сетками разного размера, зеленый цвет – граничные ячейки, бежевый – фиктивные: а) исходная геометрия поверхностной сетки, б) размер $10 \times 10 \times 10$, в) $25 \times 25 \times 25$, г) $50 \times 50 \times 50$, д) $100 \times 100 \times 100$, е) $150 \times 150 \times 150$

Рассматривается система уравнений Эйлера [294]

$$\frac{\partial}{\partial t}U + \frac{\partial}{\partial x}F + \frac{\partial}{\partial y}G + \frac{\partial}{\partial z}H = 0, \tag{2.87}$$

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{pmatrix}, \; F = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(E+p) \end{pmatrix}, \; G = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ v(E+p) \end{pmatrix}, \; H = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ w(E+p) \end{pmatrix}, \tag{2.88}$$

$$E = \rho\left(\frac{V^2}{2} + e\right), \; V^2 + u^2 + v^2 + w^2, \; e(p,\rho) = \frac{p}{\rho(\gamma - 1)}. \tag{2.89}$$

В системе уравнений (2.87) $U$ – вектор консервативных переменных, $\rho$ – давление, $u$, $v$, $w$ – составляющие вектора скорости вдоль декартовых коорди-

нат, $p$ – давление, $e$ – внутренняя энергия, $E$ – полная энергия, $\gamma$ – показатель адиабаты.

Система уравнений (2.87) решается на декартовой расчетной сетке с кубическими ячейками с помощью метода конечных объемов путем вычисления потоков консервативных величин $F$ через грани ячейки, перпендикулярные направлению сетки $X$, вектора $G$ через грани ячейки, перпендикулярные направлению сетки $Y$, вектора $H$ через грани ячейки, перпендикулярные направлению сетки $Z$:

$$U_i^{n+1} = U_i^n - \frac{\Delta t}{\Delta h}\left(F_{i+\frac{1}{2}} - F_{i-\frac{1}{2}} + G_{j+\frac{1}{2}} - G_{j-\frac{1}{2}} + H_{k+\frac{1}{2}} - H_{k-\frac{1}{2}}\right), \qquad (2.90)$$

Потоки получаются следующим образом (на примере потока $F$):

$$F_{i+\frac{1}{2}} = F_i^+(U_i^n) + F_i^-(U_{i+1}^n), \qquad (2.91)$$

$$F^{\pm} = \frac{\rho}{2\gamma}\begin{pmatrix} \lambda_1^{\pm} + 2(\gamma-1)\lambda_2^{\pm} + \lambda_5^{\pm} \\ (u-a)\lambda_1^{\pm} + 2(\gamma-1)u\lambda_2^{\pm} + (u+a)\lambda_5^{\pm} \\ v(\lambda_1^{\pm} + 2(\gamma-1)\lambda_2^{\pm} + \lambda_5^{\pm}) \\ w(\lambda_1^{\pm} + 2(\gamma-1)\lambda_2^{\pm} + \lambda_5^{\pm}) \\ (H-ua)\lambda_1^{\pm} + (\gamma-1)V^2\lambda_2^{\pm} + (H+ua)\lambda_5^{\pm} \end{pmatrix}, \qquad (2.92)$$

$$\lambda_i^{\pm} = \frac{\lambda_i \pm |\lambda_i|}{2},\ \lambda_1 = u-a,\ \lambda_2 = \lambda_3 = \lambda_4 = u,\ \lambda_5 = u+a,\ H = \frac{E+p}{\rho}. \qquad (2.93)$$

1. Для всех GHOST ячеек сетки выполнение аппроксимации величин $D = [\rho, u, v, w, p]$ на основе единожды вычисленных шаблонов (подготовку шаблонов и предварительное вычисление матриц мы не рассматриваем, так как это однократное действие, вклад которого в общем процессе расчета стремится к нулю при увеличении времени расчета). Обозначим это действие функцией `approximate_values`.
2. Для всех GHOST и OUTER ячеек сетки получение из вектора величин

$D = [\rho, u, v, w, p]$ вектора $U = [\rho, \rho u, \rho v, \rho w, E]$. Обозначим это действие функцией `d_to_u`.

3. Для всех GHOST и OUTER ячеек сетки вычисление векторов потоков $F^{\pm}$, $G^{\pm}$ и $H^{\pm}$ из (2.91), (2.92). Обозначим это действие функцией `calc_fgh`.
4. Для всех OUTER ячеек корректировка вектора $U = [\rho, \rho u, \rho v, \rho w, E]$ с помощью потоков. Обозначим это действие функцией `calc_flows`.
5. Для всех OUTER ячеек обратный пересчет обновленного вектора $U = [\rho, \rho u, \rho v, \rho w, E]$ в вектор $D = [\rho, u, v, w, p]$. Обозначим это действие функцией `u_to_d`.

Схема выполнения расчетов продемонстрирована на рис. 2.47.

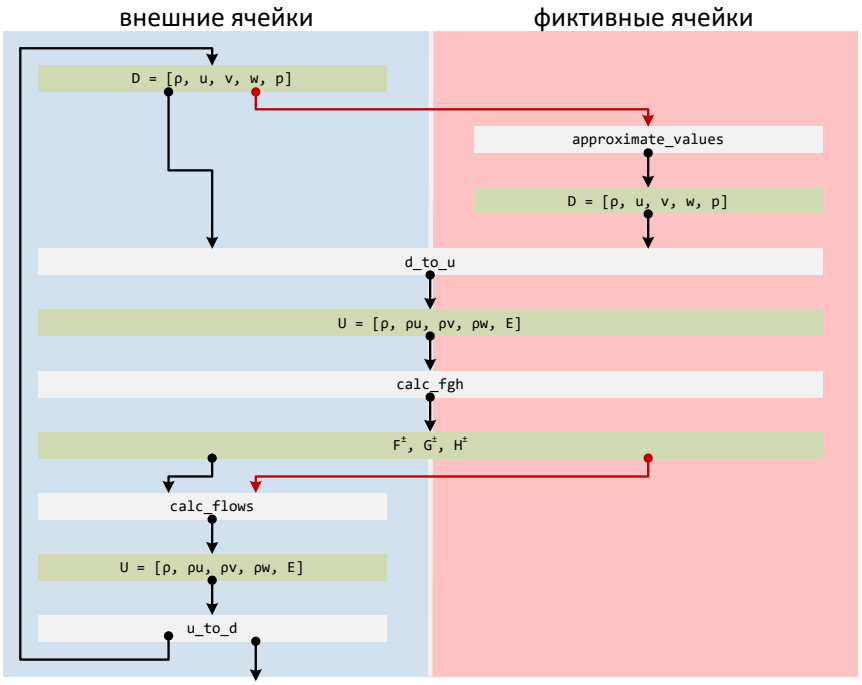


Рис. 2.47. Схема выполнения итерации расчетов методом погруженных границ с использованием схемы Стегера-Уорминга.

Использование метода погруженных границ с фиктивными ячейками позволяет проводить газодинамические расчеты вблизи сложных границ без необходимости построения согласованных сеток.

## 2.10 Выводы

Разработан метод окрестностей перестроения поверхностной неструктурированной сетки, основанный на поиске пересечения траектории смещения узла с границей окрестности инцидентных узлу ячеек и обладающий способностью сглаживания локальных дефектов для предотвращения их накопления. Основу метода составляет модель границы расчетной подобласти, применяемой при моделировании объектов со сложной геометрией, в виде замкнутой двусторонней поверхности с определенным направлением внешней нормали в каждой точке поверхности, которая при эволюции в соответствии с геометрическим принципом Гюйгенса обладает свойством сглаживания локальных дефектов (пиков и впадин). Метод является локальным, то есть для определения нового положения узла сетки использует только информацию об окрестности всех инцидентных ячеек этого узла. Получены аналитические оценки точности перестроения и эффективности сглаживания локальных дефектов для разработанного метода перестроения и других методов, с помощью этих аналитических оценок произведено сравнение разработанного метода с другими локальными методами перестроения. Алгоритм перестроения сетки по предложенному методу окрестностей обладает линейной сложностью по количеству узлов сетки. Метод реализован в программном комплексе «Кристалл» (с включением в функционал возможностей многослойного перестроения, сглаживания полей нормалей и высот) и используется при решении практической задачи по моделированию процесса ледообразования, в которой поверхностная сетка описывает внешнюю границу ледяного покрова обледеневающего тела, в процессе расчетов подтверждено свойство метода сглаживать локальные дефекты при перестроении сетки. Кроме этого метод реализован в программе «crys-gsu» подготовки неструктурированной поверхностной расчетной сетки для проведения расчетов на суперкомпьютере. Метод окрестностей перестроения поверхностной неструктурированной сетки позволяет повысить надежность перестроения для увеличения моделируемого физического времени при моделировании объектов со сложной геометрией.

Разработан метод удаления самопересечений поверхностной неструктурированной сетки с треугольными ячейками, основанный на обходе ее внешней поверхности, которая определяется направлениями внешних нормалей ячеек. При разработке метода учтены основные особенности возникновения самопересечений. Обход внешней поверхности избавляет от необходимости обрабатывать пересечения ячеек, входящих в скрытую область. Во время обхода внешней поверхности все ячейки, которые пересекаются с другими ячейками сетки, дробятся на более мелкие по всем объектам пересечения. Метод использует рациональные координаты, что избавляет от проблемы точности вычислений. Для поиска пар потенциально пересекающихся ячеек используется классический подход с применением BVH-дерева, что позволяет быстро находить пары потенциально пересекающихся ячеек. Основой метода удаления самопересечений являются алгоритмы определения пересечений геометрических примитивов, алгоритм триангуляции треугольника по множеству точек и отрезков, алгоритм обхода внешней поверхности. Метод удаления самопересечений поверхностной неструктурированной сетки путем обхода ее внешней поверхности обеспечивает корректное состояние сетки, описывающей границу расчетной подобласти.

Разработан трехмерный метод погруженной границы с фиктивными ячейками, являющийся обобщением двумерного случая и основанный на аппроксимации физических величин в фиктивной ячейке по трем точкам пространства и направлению нормали в точке на границе. Метод погруженной границы с фиктивными ячейками в трехмерной постановке позволяет проводить вычисления в расчетной подобласти со сложной границей без необходимости построения объемной сетки, согласованной с поверхностной сеткой, описывающей границу подобласти.

## Глава 3. Вычисления на блочно-структурированной сетке

Глава посвящена задаче разработки методов повышения производительности параллельных вычислений на объемных блочно-структурированных сетках. Для решения крупных расчетных задач используются современные высокопроизводительные вычислительные системы. Такие системы состоят из множества вычислительных узлов, соединенных между собой каналами связи. Для решения задачи на высокопроизводительной вычислительной системе необходимо разделить ее данные на отдельные подобласти, распределить эти данные по узлам вычислительной системы и организовать обмен сообщениями для учета взаимодействия подобластей. Подобласти данных решаемой задачи будем называть доменами, или партициями. Использование вычислительных систем с большим количеством узлов и запущенных на них процессов продиктовано стремлением сократить время выполнения, для чего необходимо решать актуальную проблему отображения вычислитеьной задачи на топологию вычисительной системы [304].

В главе предложена архитектура блочно-структурированной сетки с поддержкой дробления блоков. Рассматриваются алгоритмы распределения блоков между вычислительными процессами без дробления блоков и делается вывод о необходимости дробления блоков для обеспечения высокой производительности вычислений. Предложены алгоритмы распределения блоков сетки по вычислительным процессам с дроблением блоков, выполняется их анализ.

### 3.1 Показатели распараллеливания вычислений на распределенной памяти

Высокопроизводительные вычисления, выполняющиеся на расчетных сетках, как правило, состоят из большого количества отдельных итераций по времени, на каждой из которых обрабатываются все ячейки сетки, а между итерациями подобласти расчетной задачи обмениваются данными расчетных ячеек, находящихся вблизи общих границ. Сначала рассмотрим одну итерацию расчетов без учета информационных обменов. Рассмотрим расчет-

ную сетку, содержащую $n$ ячеек, и которую требуется декомпозировать на $k$ отдельных подобластей для обработки в $k$ вычислительных процессах. Все ячейки расчетной сетки могут обрабатываться одновременно и параллельно. Будем считать, что все ячейки являются одинаковыми с точки зрения времени их обработки. Если разбить расчетную сетку на $k$ доменов с одинаковым количеством ячеек и обрабатывать каждый домен в отдельном процессе, то обработка каждого домена на одной итерации будет занимать одно и то же время. Так как обработка всех доменов будет производиться параллельно, то это время и будет временем, затрачиваемым на одну итерацию обработки всех ячеек сетки. Если распределение ячеек сетки по доменам будет неравномерным, то время выполнения одной итерации расчетов будет определяться самым крупным доменом (так как время его обработки будет максимальным, и все остальные процессы будут вынуждены ждать окончания его обработки). Таким образом, в качестве показателя качества декомпозиции сетки можно принять неравномерность распределения ячеек по доменам – максимальное абсолютное отклонение размера домена от теоретически возможного среднего значения

$$D = \max_{0 \le i < k} \left( n_i - \frac{n}{k} \right), \tag{3.1}$$

где $n_i$ – количество ячеек в $i$-ом домене. В идеальном случае равномерного распределения ячеек по доменам показатель $D$ становится равным нулю. В худшем случае все ячейки распределяются в один домен, и $D = n\left(1 - \frac{1}{k}\right)$.

Так как абсолютный показатель не слишком удобен для сравнения разных алгоритмов декомпозиции на разных сетках, то дополнительно будем использовать показатель, который характеризует долю накладных расходов неравномерного распределения ячеек по доменам по отношении к идеальному распределению $D^* = \frac{D}{n/k}$. Значение $D^*$ изменяется в диапазоне $[0, k-1]$, где $D^* = 0$ означает идеальное распределение ячеек по доменам, а $D^* = k - 1$ соответствует наихудшему случаю, когда все ячейки отнесены к одному домену.

**Определение 3.1.** *Показатели $D$, $D^*$ будем называть показателями неравномерности декомпозиции расчетной сетки.*

После завершения итерации расчетов требуется произвести информационные обмены на границах доменов. Считаем, что каждый домен обрабатывается в своем процессе, информационный обмен происходит с использованием механизмов межпроцессного взаимодействия. Таким образом, для каждой пары доменов, имеющих общую границу необходимо организовывать межпроцессный обмен. Все обмены могут происходить одновременно, и общее затрачиваемое на них время определяется длиной максимальной границы между доменами. Для учета информационных обменов рассмотрим дуальный граф расчетной сетки $G = (V, E)$, вершинами которого являются ячейки сетки. Две вершины в дуальном графе соединены ребром, если две соответствующие ячейки являются соседними (имеют общую грань). Для каждого ребра $e \in E$ дуального графа под $e_a$ и $e_b$ будем понимать инцидентные ему вершины. Тогда в качестве второй характеристики качества декомпозиции будем использовать величину наиболее протяженной границы между парой доменов, или

$$L_{ij} = \left|\{e \in E : \{d(e_a), d(e_b)\} = \{i, j\}\}\right|, \; L = \max_{0 \le i < j < k} L_{ij}, \tag{3.2}$$

где $d(v)$ – домен, к которому относится вершина $v$. В идеальном случае значение характеристики $L$ может быть сколь угодно малым, даже нулевым (в случае, если сетка представляет собой $k$ одинаковых по количеству ячеек несвязных областей). В наихудшем случае $L = |E|$, если ячейки распределены в два домена в шахматном порядке, и каждое ребро относится к границе между этими двумя доменами.

**Определение 3.2.** *Показатель $L$ будем называть показателями максимальной длины границы между доменами при декомпозиции сетки.*

В качестве дополнительной характеристики качества декомпозиции можно использовать суммарную длину границ между доменами

$$I = \sum_{0 \le i < j < k} L_{ij} = |\{e \in E : d(e_a) \neq d(e_b)\}|, \tag{3.3}$$

что соответствует общему количеству данных межпроцессного обмена.

**Определение 3.3.** *Междоменным, или кросссдоменным ребром, или кросс-ребром $e \in E$ дуального графа будем называть ребро, инцидентные вершины которого относятся к разным доменам, то есть $d(e_a) \neq d(e_b)$.*

Так как домены граничат между собой по ребрам дуального графа, то суммарная длина всех границ между доменами представляет собой просто количество всех междоменных ребер. Наряду с показателем $I$ будем использовать долю междоменных ребер среди общего количества ребер сетки $I^* = \frac{I}{|E|}$. Показатель $I^*$ также является нормированным, он изменяется в диапазоне $[0, 1]$ и принимает значение 0 в идеальном случае и 1 – в наихудшем.

**Определение 3.4.** *Показатели $I$, $I^*$ будем называть показателями суммарной длины границ между доменами при декомпозиции расчетной сетки.*

## 3.2 Архитектура блочно-структурированной сетки с поддержкой функции дробления блоков

В разделе предложена архитектура объемной блочно-структурированной сетки с поддержкой блоков, интерфейсов касания блоков, областей задания начальных и граничных условий, механизма дробления блоков.

Одним из распространенных типов расчетных сеток является блочно-структурированная сетка, состоящая из нескольких блоков, некоторые из которых имеют общие границы. Такая структура позволяет выполнять вычисления для различных блоков независимо друг от друга, обмениваясь общими данными на границах. Блочно-структурированная объемная сетка используется в ПО «Лазурит» [259], используемом в качестве внешнего решателя для программного комплекса «Кристалл» для расчета газодинамических величин вокруг обледеневающего тела и для расчета движения капель. Для эффективного проведения вычислений на блочно-структурированных сетках важную роль имеет реализация сетки в виде внутреннего представления и организация механизма межпроцессного обмена данными между соседними блоками сетки [305].

Будем рассматривать объемные блочно-структурированные сетки, состоящие из отдельных блоков, каждый из которых представляет собой упорядоченный трехмерный массив ячеек. Упорядоченность размещения ячеек обеспечивает быстрый доступ к ячейками по индексам и снижает требования к объему памяти, однако строить блочно-структурированные сетки гораздо сложнее, чем неструктурированные.

Блоки блочно-структурированной сетки могут иметь сложную форму. Для доступа к отдельным ячейкам блока вводятся три индекса, связанные с криволинейной системой координат блока. Система координат задается тремя линиями координат $(I, J, K)$, каждая из которых связывает пары противоположных граней блока. Для наглядности будем изображать блоки сетки в виде прямоугольных параллелепипедов, как показано на рис. 3.1 слева.

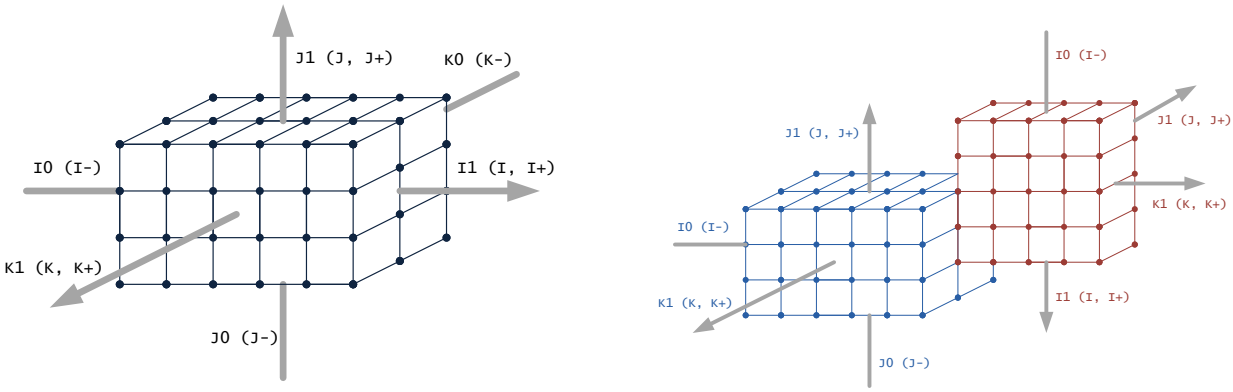


Рис. 3.1. Блок сетки (слева) и касание блоков (справа)

Блоки могут граничить между собой по области, называемой интерфейсом. Один интерфейс описывает соприкосновение двух блоков прямоугольными подобластями своих граней. При этом системы координат двух соседних блоков не обязаны согласовываться между собой (рис. 3.1 справа).

Выполнение расчетов на сетке носит итерационный по времени характер. Для каждой итерации по времени выполняется пересчет расчетных данных ячеек согласно той или иной вычислительной схеме. Во время обработки одной ячейки возникает потребность обращаться за данными к другим ячейкам, находящимся рядом с рассматриваемой.

**Определение 3.5.** *Совокупность ячеек расчетной сетки, данные которых используются при обработке конкретной ячейки, называют вычислительным шаблоном, или вычислительной молекулой, или вычислительной окрестностью этой ячейки.*

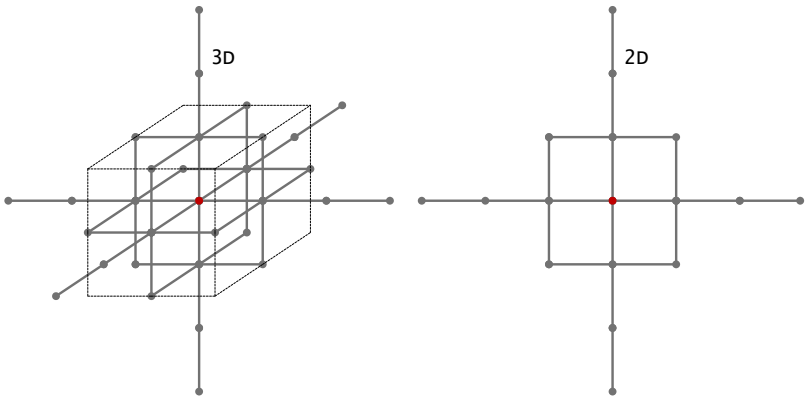


Рис. 3.2. Примеры вычислительных окрестностей ячеек сетки

На рис. 3.2 показаны примеры вычислительных окрестностей ячейки. В дальнейшем для большей наглядности будем изображать блоки в двумерном виде (с системой координат $IJ$). Во время произведения расчетов различные блоки сетки обрабатываются независимо друг от друга. При этом для некоторых ячеек обрабатываемого блока их вычислительные окрестности выходят за границу блока и затрагивают соседние блоки. Для эффективного счета необходимо обеспечить функционал обмена данными между блоками. Если два соседних блока обрабатываются в разных вычислительных процессах, то для обмена данными нужно использовать передачу сообщений. Приведем классификацию ячеек внутри блока сетки (рис. 3.3 слева).

Внутренними ячейками будем называть те ячейки блока, вся вычислительная окрестность которых лежит внутри того же блока. Ячейки, не являющиеся внутренними ячейками блока, будем называть граничными ячейками. Если окрестность ячейки частично накрывает соседний блок, то такую ячейку будем относить к интерфейсным ячейкам, так как ее окрестность пересекает интерфейс блока. Если окрестность ячейки частично накрывает блок, который обрабатывается в другом вычислительном процессе, то будем относить

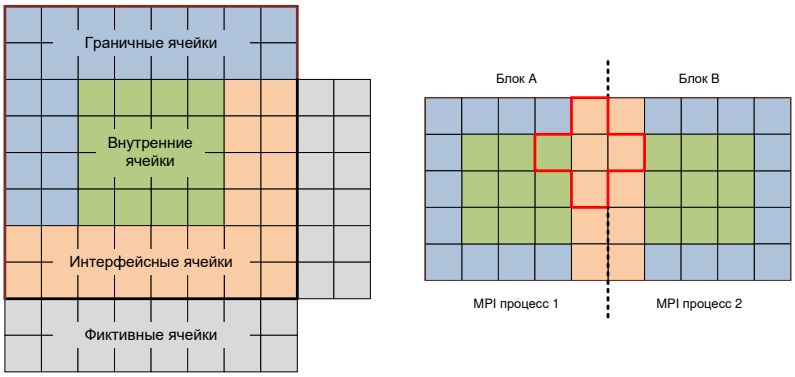


Рис. 3.3. Ячейки разных типов

ее к ячейкам MPI-обмена, для получения данных этой окрестности нужно использовать средства межпроцессного обмена (рис. 3.3 справа).

При обработке ячеек MPI-обмена мы должны обращаться за данными к другому процессу. Если делать это только по мере необходимости, то придется совершать большое количество обменов, что негативно скажется на производительности. Поэтому для повышения быстродействия вместо этого для блока заводится специальный слой внешних ячеек, куда копируются все необходимые данные из соседних процессов, которые могут понадобиться при обработке блока. Такие ячейки будем называть фиктивными.

**Определение 3.6.** *Фиктивными ячейками блока будем называть дополнительно созданные для этого блока ячейки, которые содержат скопированные из соседних блоков данные, необходимые для выполнения вычислений в рассматриваемом блоке. Множество фиктивных ячеек блока также будем называть его теневым слоем.*

Ширина теневого слоя блока определяется размером вычислительного шаблона обработки ячейки. Объединение всех ячеек блока с множеством фиктивных ячеек является замкнутой структурой, которая более не требует для своей обработки обращений к другим вычислительным процессам на одной итерации выполнения расчетов.

Опишем основные объекты, используемые для организации работы с блочно-структурированной сеткой и проведения вычисления на ней [129].

Основным объектом блочно-структурированной расчетной сетки является блок (Block) (рис. 3.4 слева). Блок состоит из трехмерного массива ячеек размера $a \times b \times c$ ($a$, $b$, $c$ – размеры вдоль измерений $I$, $J$ и $K$ соответственно). Каждая ячейка содержит набор физических величин, ассоциированных с центром ячейки, и другие вспомогательные данные. Также блок содержит данные о всех вершинах своих ячеек, эти данные хранятся в трехмерном массиве размера $(a+1) \times (b+1) \times (c+1)$. Для определения геометрии блока необходимо задать его линейные размеры и координаты всех его узлов.

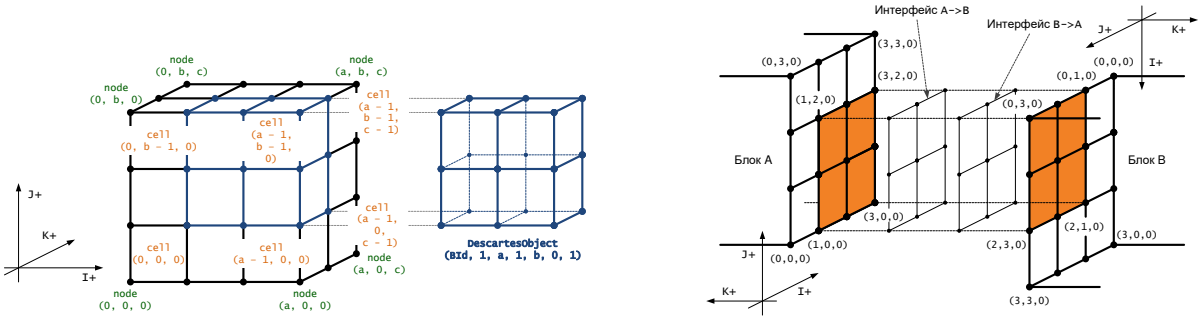


Рис. 3.4. Координаты узлов блока (слева) и структура интерфейса (справа)

Объектом, описывающим соприкосновение двух соседних блоков, является интерфейс (Iface) (рис. 3.4 справа). Интерфейс является однонаправленным, он сообщает, что у рассматриваемого блока конкретная прямоугольная часть границы соприкасается с другим блоком. Чтобы определить с какой частью другого блока граничит рассматриваемый блок, нужно рассмотреть смежный ему интерфейс. Таким образом, полная информация о касании двух соседних блоков описывается парой смежных интерфейсов. Интерфейс описывается в системе координат того блока, к которому он относится. Координаты задают размер области соприкосновения по всем трем направлениям. В приведенном на рис. 3.4 справа примере касания двух блоков два смежных интерфейса, описывающих касание блоков $A$ и $B$, задаются следующим образом: $A(1,3,0,2,0,0) \to B$, $B(0,2,1,3,0,0) \to A$.

На границе блока кроме другого блока может находиться граница расчетной области с граничными условиями (BCond). Граничные условия задаются аналогично интерфейсу (с помощью системы координат блока), однако вместо

соседнего блока подается ссылка на описатель граничных условий. Таким образом, и интерфейс и граничное условие являются частным случаем границы блока (Border). Также в архитектуре поддержан специальный вид граничных условий – связанные граничные условия, которые применяются для моделирования в областях, обладающих симметрией. В работе [301] описано применение предложенной архитектуры, с помощью которой проведение вычислений в области, состоящей из симметрично расположенных секторов, заменяется на вычисления, выполняемые в рамках одного сектора.

В процессе вычислений необходимо иметь возможность быстро узнать для конкретной граничной ячейки, какого типа граница ей соответствует и быстро найти соответствующий интерфейс или граничное условие. Для этой цели служат грани блока (Facet). Каждый блок имеет 6 граней (для каждого из направлений $I^-$, $I^+$, $J^-$, $J^+$, $K^-$, $K^+$). Каждая грань является двумерным массивом, содержащим для каждой ячейки, граничащей с гранью, ссылку на интерфейс или на граничное условие (рис. 3.5).

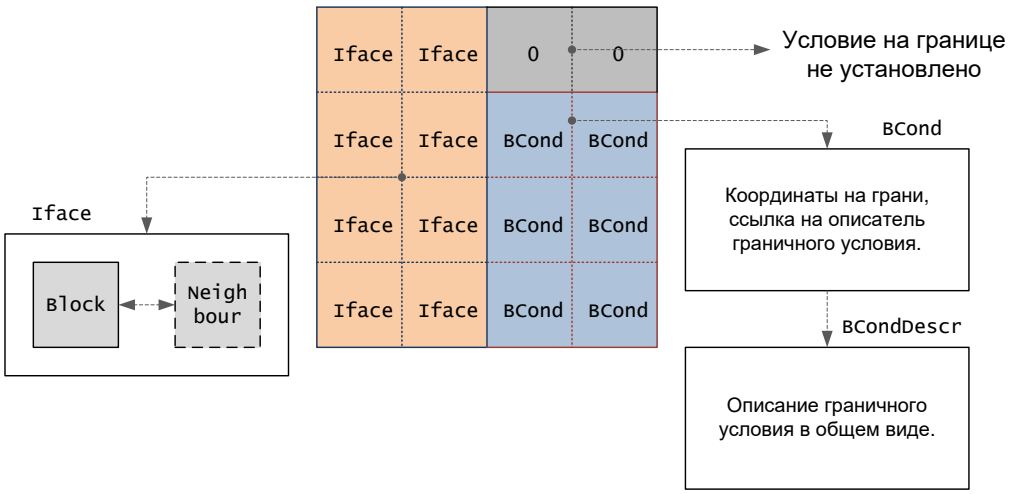


Рис. 3.5. Структура грани блока

Для описания начальных условий используется трехмерная область блока (Scope), которая реализована аналогично граничным условиям.

Таким образом, все основные объекты расчетной сетки являются двумерными или трехмерными декартовыми объектами, то есть такими объектами, чья геометрия задается двумерными или трехмерными массивами узлов

блоков сетки. Также заметим, что граничные условия и области являются именованными объектами, так как им присваиваются имена для определения механизмов задания граничных и начальных условий расчета. Получаем следующую иерархию классов, реализующих расчетную сетку (рис. 3.6).

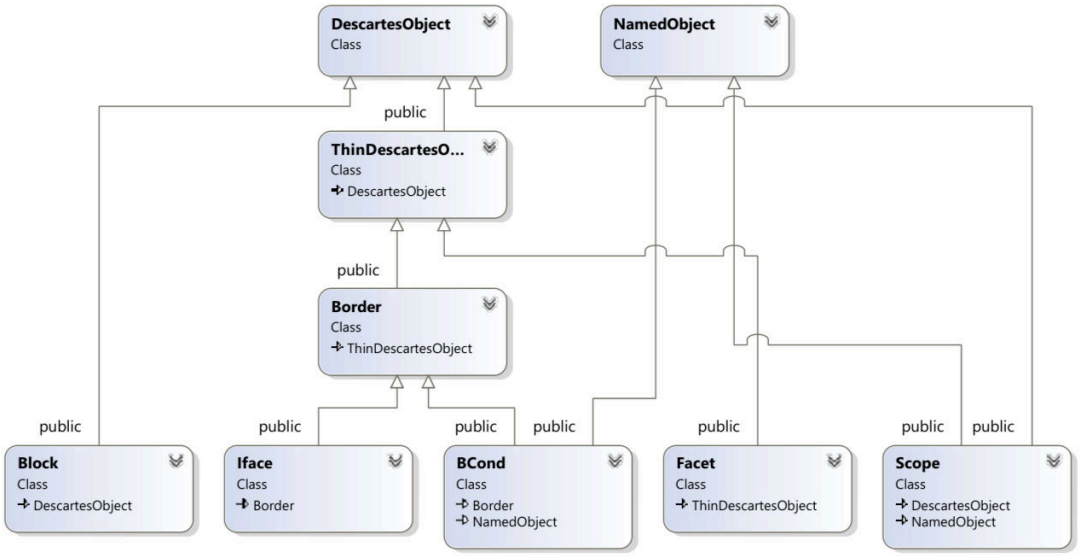


Рис. 3.6. Иерархия объектов внутреннего представления сетки

Рассмотрим механизм обмена данными между блоками сетки. Если два соседние блока, между которыми должен быть осуществлен обмен данными, обрабатываются в разных процессах при выполнении расчетов на суперкомпьютере, то обмен данными может быть осуществлен с помощью обмена сообщениями с использованием MPI. Каждый блок должен отправить своим соседям данные, находящиеся в его интерфейсных ячейках, и получить данные, которые соответствуют ячейкам его теневого слоя.

Логика обменов устроена следующим образом. Так как пара смежных интерфейсов определяет факт касания двух блоков, между которыми должен произойти обмен, то на каждом из этих интерфейсов заводится буфер для данных обмена, после чего происходит обмен данными буферами (рис. 3.7). Обмены данными происходят одновременно по всем интерфейсам сетки с помощью асинхронных функций `MPI_Isend`, `MPI_Irecv`. Рассмотрим механизм обмена данными между соседними блоками, находящимися в разных процессах, более подробно.

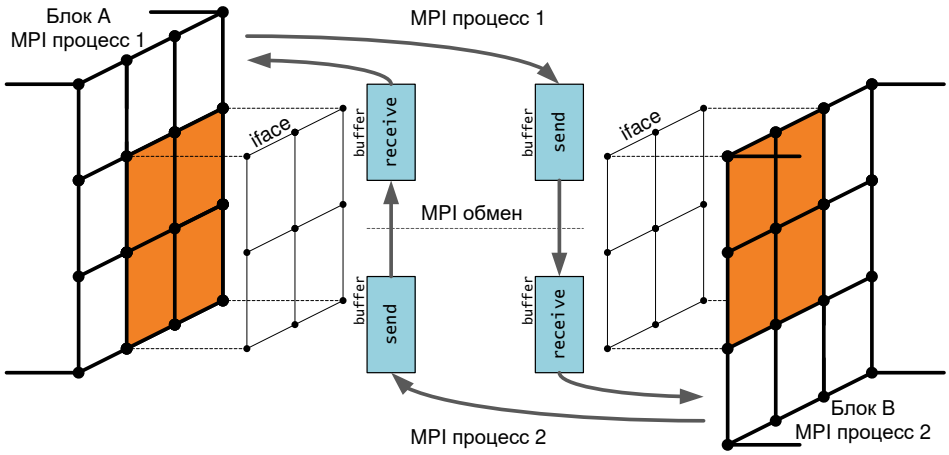


Рис. 3.7. Обмен данными между блоками сетки

Пусть Блок $A$ обрабатывается в процессе 1, а блок $B$ – в процессе 2, как показано на рис. 3.7. Каждый из двух смежных интерфейсов, описывающих касание этих двух блоков имеет свой буфер обмена данными. Так как в процессе 1 активным является блок $A$, то буфер его интерфейса работает на прием данных, а буфер интерфейса со стороны блока $B$ – на отправку (данных блока $A$). В процессе 2 все происходит ровно наоборот: активным является блок $B$, буфер интерфейса блока $A$ работает на отправку данных блока $B$, буфер интерфейса со стороны блока $B$ работает на прием данных. Для всех интерфейсов происходит асинхронный запуск всех вызовов функций обмена, после чего происходит ожидание завершения всех операций. После этого каждый блок может извлечь данные из буфера интерфейса, в котором он является активным, и поместить эти данные в свою теневую зону, после чего начинается следующая итерация расчетов.

Предложенная в разделе архитектура блочно-структурированной сетки реализована в рамках программы «GridMaster» подготовки блочно-структурированной расчетной сетки для проведения расчетов на суперкомпьютере [129] и используется при проведении газодинамических расчетов с помощью ПО «Лазурит» [301, 306, 307].

## 3.3 Распределение вычислительной нагрузки при проведении параллельных вычислений на блочно-структурированной сетке

Рассмотрим декомпозированную расчетную задачу (вычисления на блочно-структурированной сетке, либо на неструктурированной сетке, разбитой на домены). При распределении частей расчетной задачи между вычислителями суперкомпьютерного кластера для уменьшения времени выполнения задачи необходимо учитывать характеристики самого кластера, к которым относится производительность вычислителей и скорость обмена данными между ними. В этом разделе приводится постановка задачи распределения вычислительной нагрузки на вычислительном кластере с учетом топологии расчетной задачи и характеристик кластера [308, 309].

**Определение 3.7.** *Графом вычислительного кластера назовем граф* $H = (V_H, E_H)$. *Узлы этого графа соответствуют отдельным вычислителям кластера (одному процессору или сопроцессору), а ребра – логическим соединениям между вычислителями (то есть возможностью обмена данными).*

Так как между любыми двумя вычислителями возможен обмен данными, то граф $H$ является полным (и даже является псевдографом, так как содержит петли). Введем на графе $H$ две весовые функции. Первая функция $f : V_H \to \mathbb{R}_{>0}$ каждому узлу графа ставит в соответствие скорость работы соответствующего вычислителя, то есть количество расчетных данных, которые могут быть обработаны на нем за одну условную единицу времени. Вторая функция $l : E_H \to \mathbb{R}_{>0}$ несет смысл скорости обмена данными между вычислителями, она показывает количество данных, которые могут быть переданы между соответствующими вычислителями за одну условную единицу времени.

**Определение 3.8.** *Графом расчетной задачи назовем граф* $G = (V_G, E_G)$, *который отражает особенности декомпозиции расчетной области (это может быть граф смежности блоков блочно-структурированной сетки или граф смежности доменов неструктурированной сетки). Узлы этого*

*графа соответствуют подобластям данных задачи, каждая из которых обрабатывается одним вычислителем. Ребра графа соответствуют потокам данных между разными смежными частями этой задачи.*

На графе $G$ также вводятся две весовые функции. Первая функция $w : V_G \to \mathbb{R}_{>0}$ имеет значение объема вычислительных данных, обрабатываемых в соответствующей части задачи. Вторая функция $i : E_G \to \mathbb{R}_{>0}$ отражает объем данных, пересылаемых между частями задачи при межпроцессном обмене. Значения всех четырех функций $f$, $l$, $w$, $i$ измеряются в одних и тех же единицах (это могут быть ячейки расчетной сетки, количество байтов или вещественных значений). Однако $w$ и $i$ имеют смысл объема вычислений и пересылок, тогда как функции $f$ и $l$ относятся к фиксированной единице времени, то есть имеют смысл скорости обработки данных либо их пересылки при межпроцессном обмене.

Расчет задачи состоит из двух чередующихся фаз: проведение итерации расчета и выполнение межпроцессных обменов данными. Требуется таким образом распределить задачу между вычислителями кластера, чтобы суммарное время одной итерации расчета и одной итерации межпроцессных обменов было минимально. Распределение задачи между вычислителями кластера определяется функцией $\gamma : V_G \to V_H$.

Время выполнения одной итерации вычислений определяется по самому загруженному вычислителю:

$$t_1^{max} = \max_{v_H \in V_H} \left\{ \frac{1}{f(v_H)} \sum_{\substack{v_G \in V_G \\ \gamma(v_G) = v_H}} w(v_G) \right\}. \tag{3.4}$$

Аналогично время выполнения одной итерации межпроцессных обменов определяется по самому загруженному каналу обмена:

$$t_2^{max} = \max_{e_H \in E_H} \left\{ \frac{1}{l(e_H)} \sum_{\substack{e_G = (u_G, v_G) \in E_G \\ (\gamma(u_G), \gamma(v_G)) = e_H}} i(e_G) \right\}. \tag{3.5}$$

Таким образом, задача поиска оптимального распределения задачи по вычислителям кластера сводится к поиску функции $\gamma$, минимизирующей значение функционала $t^{max}(\gamma) = t_1^{max}(\gamma) + t_2^{max}(\gamma)$.

Для вычислительных задач с малой долей мепроцессных обменов можно ограничиться минимизацией функционала $t^{max}(\gamma) = t_1^{max}(\gamma)$. Также можно рассматривать задачу распределения для гомогенного кластера, для которого функции $f$ и $l$ являются константами. Минимизируемый функционал в этом случает принимает следующий вид:

$$t^{max}(\gamma) = \frac{1}{f} \max_{v_H \in V_H} \left\{ \sum_{\substack{v_G \in V_G \\ \gamma(v_G) = v_H}} w(v_G) \right\} + \frac{1}{l} \max_{e_H \in E_H} \left\{ \sum_{\substack{e_G = (u_G, v_G) \in E_G \\ (\gamma(u_G), \gamma(v_G)) = e_H}} i(e_G) \right\}. \quad (3.6)$$

Далее будем рассматривать распределение вычислительной нагрузки между процессами гомогенного вычислительного кластера без учета характеристик каналов связи между вычислителями, то есть $t^{max}(\gamma) = t_1^{max}(\gamma)$ в применении к блочно-структурированным расчетным сеткам.

### 3.3.1 Распределение блоков без дробления

Для распределения блоков расчетной сетки по вычислительным процессам рассмотрим следующую задачу разделения множества $m$ весов на $k$ подмножеств. Пусть дано множество $X$ вещественных чисел $x_i \geq 0$ для $i \in M$, где $M = [0, m-1]$. Рассмотрим также множество индексов $j \in K$, где $K = [0, k-1]$. Будем говорить, что определено разбиение множества $X$ на $k$ подмножеств, если введена функция $\gamma : M \to K$. Множество всех функций разбиения будем обозначать $\Gamma(M, K)$. Веса результирующих подмножеств будем определять естественным образом для $j \in K$: $X_j = \sum_{\substack{i \in M \\ \gamma(i) = j}} x_i$. Требуется найти такую функцию разбиения $\gamma \in \Gamma(M, K)$, чтобы минимизировать наиболее тяжелое из результирующих подмножеств $X^{max} = \max_{j \in K} X_j$.

Задача может быть расширена на случай распределения вычислительной нагрузки между вычислителями суперкомпьютера с разной производительно-

стью. При этом формулировка задачи меняется только в части приведения всех процессов к одному показателю с помощью весовых коэффициентов. Коэффициентом приведения $\kappa(j)$ для $j \in K$ назовем такую положительную функцию $\kappa : K \to \mathbb{R}_{>0}$, что время выполнения нагрузки $\kappa(j)$ на процессе $j$ не зависит от $j$. Тогда в общем случае задача о разбиении множества $m$ весов на $k$ подмножеств с коэффициентами приведения $\kappa(j)$ для $j \in K$ формулируется следующим образом. Требуется найти такую функцию разбиения $\gamma \in \Gamma(M, K)$, чтобы минимизировать наиболее тяжелое из результирующих подмножеств с учетом коэффициентов приведения $\max_{j \in K} \frac{X_j}{\kappa(j)}$. Эта задача имеет практическое применение при распределении вычислительной нагрузки между вычислителями гетерогенного суперкомпьютера. Далее будем рассматривать задачу без учета весовых коэффициентов $\kappa$.

Поставленная задача о распределении множества $m$ весов на $k$ подмножеств при условии минимизации $X^{max}$ связана с большими вычислительными затратами. В [310] можно найти описание алгоритма решения этой задачи (называемой задачей о куче камней), основанного на решении другой, так называемой $z$-задачи, которая формулируется следующим образом: определить, существует ли такое разбиение множества $m$ весов на $k$ подмножеств, для которого выполнено условие $X^{max} \leq z$. Используя $z$-задачу, исходную задачу о разбиении с минимизацией $X^{max}$ можно решить с помощью бинарного поиска. Решение же $z$-задачи выполняется методом ветвей и границ с односторонним обходом дерева вариантов разбиения. При решении $z$-выполняется попытка последовательно заполнить каждое из $k$ подмножеств весами, начиная с наибольшего, не превышая ограничения на вес подмножества $z$. При этом в памяти постоянно хранится значения общего свободного места в результирующих подмножествах. Если значение общего свободного места становится отрицательным, то последнее действие по назначению веса в подмножество отменяется и продолжается перебор по дереву вариантов. Такой подход хоть существенно сокращает время работы алгоритма, однако не снижает его сложность и при больших значениях $m$ и $k$ неприменим.

С другой стороны, поставленная задача может быть решена приближен-

но с помощью жадного алгоритма. Опишем последовательность действий жадного алгоритма. Множество распределяемых весов будем хранить в виде массива, который перед началом работы алгоритма отсортируем по убыванию. Далее в алгоритме будем последовательно обрабатывать все веса, начиная с наибольшего. Каждый необработанный вес будем относить к наиболее легкому на текущий момент подмножеству весов. Эта последовательность действий формализована как алгоритм 3.2.

**Алгоритм 3.2.** *Жадный алгоритм распределения множества весов по подмножествам (`DistributeGreedy0`)*

**Вход:** $m$ – количество распределяемых весов, $X = [x_0, x_1, \ldots, x_{m-1}]$ – массив распределяемых весов, $k$ – количество результирующих подмножеств.

**Выход:** $[\gamma_0, \gamma_1, \ldots, \gamma_{m-1}]$ – массив, в котором элемент $\gamma_i$ содержит номер подмножества, к которому отнесен вес $x_i$.

1 отсортировать массив $X$ по убыванию
2 $P \leftarrow \{p : p.w = 0, p.i = i, \forall i \in [0, k-1]\}$ – множество, описывающее текущее состояние результирующих подмножеств, где $p.w$ – вес подмножества, $p.i$ – его номер, а сравнение подмножеств выполняется по весу
3 **for** $i \leftarrow 0$ **to** $(m-1)$ **do**
4 $\quad p \leftarrow P.min$
5 $\quad p.w \leftarrow p.w + x_i$
6 $\quad \gamma_i \leftarrow p.i$
7 **end**

Сложность алгоритма `DistributeGreedy0` зависит от используемой реализации множества $P$. Целесообразно реализовать множество $P$ с помощью упорядоченного по возрастанию множества пар (вес, индекс), тогда создание этого множества будет иметь сложность $O(k \log k)$, поиск минимального элемента $P.min$ будет иметь сложность $O(1)$, а обновление веса – $O(\log k)$. Тогда общая сложность алгоритма составляет $O(m \log m + k \log k + m \log k)$, где $O(m \log m)$ – сложность подготовки массива весов (сотрировка массива весов), $O(k \log k)$ – сложность подготовки множества $P$ (создание упорядоченного

множества). Можно считать $m > k$, в этом предположении сложность алгоритма равна $O(m \log m)$, так как остальные члены оценки имеют порядок, не больший $O(m \log m)$. ■

Проведем анализ эффективности работы алгоритма 3.2. Для удобства без ограничения общности будем считать, что изначальное множество весов упорядочено по убыванию.

Определим остаточный член $r_i$ для $i \in M$ по следующей формуле:

$$r_i = \max\left(x_i - \frac{1}{k}\sum_{p=i}^{m} x_p, 0\right). \tag{3.7}$$

В приведенных обозначениях и предположениях верна следующая лемма, касающаяся оценки неравномерности распределения весов:

**Лемма 3.1.** *При использовании жадного алгоритма разбиения множества $m$ весов на $k$ подмножеств отклонение наиболее тяжелого подмножества $X^{max}$ от среднего значения веса подмножеств $X^{avg} = \frac{1}{k}\sum_{j\in K} X_j$ не превышает максимальный остаточный член $r_i$, то есть*

$$X^{max} - X^{avg} \le \max_{i\in M} r_i. \tag{3.8}$$

Пусть $X_q = X^{max}$ – наиболее тяжелое получившееся в результате работы жадного алгоритма подмножество, а $x_t$ – последний добавленный в него элемент. Так как алгоритм является жадным, то до обработки веса $x_t$ подмножество $X_q$ не превосходит по весу никакое другое подмножество, то есть для любого $j \in K$ выполняется соотношение

$$X_q - x_t \le X_j - \chi_j, \tag{3.9}$$

где $\chi_j$ – сумма весов, добавленных в $j$-е подмножество начиная с момента обработки веса $x_t$. В частности можно заметить, что при $j = q$ неравенство (3.9) превращается в равенство, так как $\chi_q = x_t$.

Почленно суммируя (3.9) по $j \in K$, получим соотношение

$$kX_q - kx_t \leq \sum_{j \in K} X_j - \sum_{p=t}^{m} x_p, \tag{3.10}$$

после деления которого на $k$ и переноса членов в нужные части неравенства, получим $X^{max} - X^{avg} \leq r_t \leq \max_{i \in M} r_i$. ■

Для оценки работы жадного алгоритма распределения $m$ весов по $k$ подмножествам достаточно проанализировать ряд остаточных членов, полученный из отсортированного массива распределяемых весов. Также из приведенной леммы можно сделать вывод, что неравномерное распределение весов по множествам означает существование остаточного члена с высоким значением (индикатор некоторого большого значения $x_i$).

Задачу о разбиении множества $m$ весов по $k$ подмножествам можно применить для распределения блоков расчетной сетки по вычислительным процессам гомогенного вычислительного кластера. При этом весом блока является количество его ячеек. Заметим, что применительно к задаче распределения блоков расчетной сетки по вычислительным процессам оценка, полученная в лемме 3.1, является оценкой показателя $D$ неравномерности распределения вычислительной нагрузки. Обозначим множество блоков через $B$ ($|B| = m$), для каждого блока $b \in B$ определен его вес $b.w$, также определен вес всего множества блоков $B.w = \sum_{b \in B} b.w$. Блоки сравниваются между собой по их весу. Обозначим множество результирующих подмножеств, по которым распределяются блоки (эти подмножества будем называть партициями), через $P$, для каждой партиции $p \in P$ определено множество входящих в нее блоков $p.B$ и ее вес $p.w = p.B.w$. Партиции сравниваются между собой по их весу. Для множества партиций $P$ определен показатель неравномерности распределения блоков $P.D^*$. Для всех множеств определены операции добавления и удаления элементов. В приведенных обозначениях рассмотрим описание жадного алгоритма распределения блоков расчетной сетки по партициям.

**Алгоритм 3.3.** *Жадный алгоритм распределения множества блоков по партициям (`DistributeGreedy`)*

**Вход:** $B$ – множество блоков ($|B| = m$), $k$ – количество результирующих партиций.
**Выход:** $P$ – множество результирующих партиций.

```
1 Функция DistributeGreedy(B, k):
2     P ← {p : p.B = ∅, ∀i ∈ [0, k − 1]}
3     while B ≠ ∅ do
4         b ← B.max
5         p ← P.min
6         B ← B \ {b}
7         p.B ← p.B ∪ {b}
8     end
9     return P
```

Алгоритм `DistributeGreedy`, по аналогии с алгоритмом 3.2, может быть реализован со сложностью $O(m \log m + k \log k + m \log k)$ при реализации множества $B$ в виде отсортированного массива, а множества $P$ в виде упорядоченного множества пар (вес партиции, номер партиции), и $O(m \log m)$ в предположении $m > k$. ■

Высокие значения остаточных членов $r_i$, которые используются при оценке качества разбиения множества $m$ весов на $k$ подмножеств в лемме 3.1, являются индикаторами наличия крупных блоков в сетке, что ухудшает показатель качества распределения $D$ ухудшается. Для повышения эффективности распределения вычислительной нагрузки для блочно-структурированной расчетной сетки необходимо использовать дробление крупных блоков.

### 3.3.2 Механизм дробления блоков

Одним из важнейших для распределения вычислительной нагрузки действий по управлению блочно-структурированной расчетной сеткой при выполнении вычислений на суперкомпьютере является дробление ее блоков [313]. Так как при запуске задач на суперкомпьютере постоянно возрастает степень

параллельности (используется все больше параллельных процессов обработки блоков сетки), то для сохранения равномерности распределения блоков по вычислительным процессам требуется уметь измельчать блоки. Блок сетки может быть разделен на два блока по любому из трех измерений: $I$, $J$, $K$.

Кроме блоков сетка содержит другие объекты, которые требуют корректировки после разделения блока. К таким объектам относятся интерфейсы, описывающие касание блоков друг друга. На границе расчетной области граничные условия задаются с помощью специальных объектов, которые также должны быть разделены в случае пересечения их линией разреза блока. Также должны быть по необходимости разделены области, описывающие начальные условия. Каждый из этих объектов имеет жесткую привязку к блоку, а значит после дробления может возникнуть необходимость разделения этого объекта. Наибольшую сложность представляет дробление блоков в случае самокасания. В этом случае дробление блока может спровоцировать дробление этого же блока по другому направлению.

Граничные условия и области начальных условий обрабатываются наиболее просто и похожим образом. Рассмотрим, например, граничные условия в двумерном случае (в координатах $IJ$).

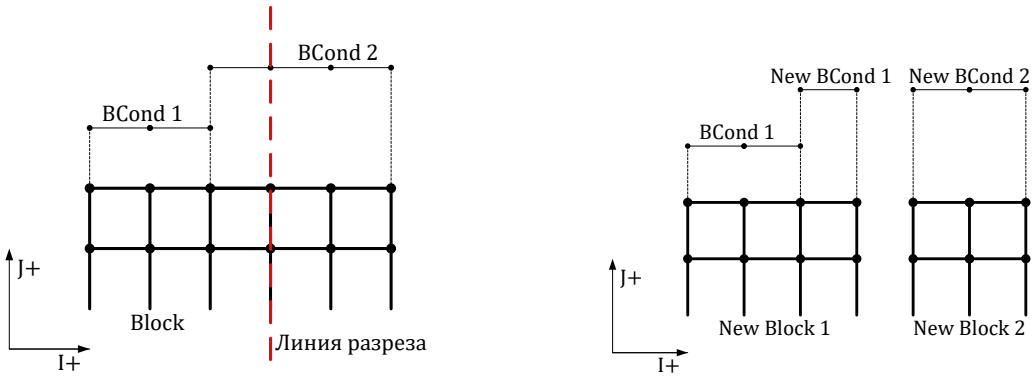


Рис. 3.8. Дробление блока может спровоцировать дробление других объектов сетки

Пусть блок Block должен быть разрезан по направлению $I^+$ как показано на рис. 3.8 слева. Пусть этот блок имеет граничные условия по направлению $J^+$. Тогда возможны два варианта. Либо линия разреза не пересекает

граничное условие, и тогда граничное условие целиком отходит одному из результирующих блоков (BCond 1). Если же линия разреза проходит через граничное условие, то это граничное условие также должно быть разделено, и его части отойдут двум результирующим блокам (рис. 3.8 справа). Такая же ситуация может сложиться в отношении областей начальных условий. В случае дробления интерфейсов может возникнуть необходимость дробления соседних блоков или повторного деления того же блока (если имеет место самокасание блока).

### 3.3.3 Распределение блоков с дроблением

Сначала рассмотрим следующую задачу по выделению из блока расчетной сетки части заданного размера при условии минимизации количества дроблений блока. Пусть есть трехмерный блок размера $a \times b \times c$ ($a > 0$, $b > 0$, $c > 0$). Требуется определить минимальное количество дроблений, которые необходимо выполнить для того, чтобы выделить из этого блока часть, размера $t \geq 0$ (возможно эта часть будет состоять из нескольких блоков). Обозначим это минимальное количество дроблений через $P^t_{a \times b \times c}$.

Сначала рассмотрим задачу для одномерного блока размера $a$. Минимальное количество дроблений для выделения из него части размера $t$ обозначим $P^t_a$. Для одномерного случая, очевидно, что при $t > a$ решение отсутствует, для $t = 0$ и $t = a$ дробления выполнять не нужно, а для остальных значений $t$ достаточно одного разреза блока:

$$P^t_a = \begin{cases} +\infty, & \text{если } t > a, \\ 0, & \text{если } (t = 0) \vee (t = a), \\ 1, & \text{если } 0 < t < a. \end{cases} \tag{3.11}$$

Далее рассмотрим задачу для двумерного случая. В этом случае рассматривается двумерный блок размера $a \times b$, а минимальное количество дроблений для выделения из него части размера $t$ обозначим через $P^t_{a \times b}$. Основная идея выделения из блока части размера $t$ заключается в дроблении блока на два новых блока по одному из измерений и выделения из них частей размера $t_1$ и

$t - t_1$ соответственно (рис. 3.9). Для получения ответа необходимо рассмотреть все возможные варианты первоначального разделения блока на два новых и выбрать тот вариант, который приводит к наименьшему количеству дроблений. При этом отдельно следует рассмотреть краевые случаи: при $t > ab$ решение отсутствует, при $t = 0$ или $t = ab$ никаких дроблений выполнять не нужно, а для случаев $a = 1$ или $b = 1$ задача сводится к одномерному случаю и вычисление выполняется по формуле (3.11).

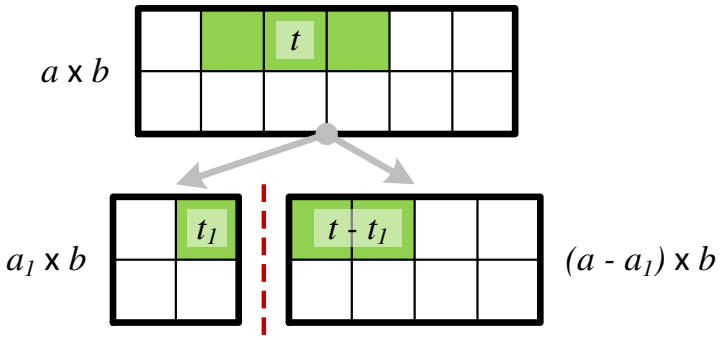


Рис. 3.9. Рекурсивное выделение из блока $a \times b$ части размера $t$

Исходя из этого, значение $P^t_{a\times b}$ определяется следующим образом:

$$P^t_{a\times b} = \begin{cases} +\infty, & \text{если } t > ab, \\ 0, & \text{если } (t = 0) \vee (t = ab), \\ P^t_b, & \text{если } a = 1, \\ P^t_a, & \text{если } b = 1, \\ \min\Big[\min\limits_{\substack{1\le a_1<a\\ 0\le t_1<t}}\{P^{t_1}_{a_1\times b} + P^{t-t_1}_{(a-a_1)\times b}\}, & \\ \quad \min\limits_{\substack{1\le b_1<b\\ 0\le t_1<t}}\{P^{t_1}_{a\times b_1} + P^{t-t_1}_{a\times(b-b_1)}\}\Big] + 1', & \text{если } (\min(a,b) > 1) \\ & \quad \wedge (0 < t < ab). \end{cases} \tag{3.12}$$

Для трехмерного случая значение $P^t_{a\times b\times c}$ вычисляется аналогично с помощью дробления блока по одному из трех измерений:

$$P^t_{a\times b\times c} = \begin{cases} +\infty, & \text{если } t > abc, \\ 0, & \text{если } (t=0) \\ & \quad \vee (t=abc), \\ P^t_{b\times c}, & \text{если } a = 1, \\ P^t_{a\times c}, & \text{если } b = 1, \\ P^t_{a\times b}, & \text{если } c = 1, \\ \min\Big[ \min\limits_{\substack{1\le a_1<a \\ 0\le t_1<t}} \{P^{t_1}_{a_1\times b\times c} + P^{t-t_1}_{(a-a_1)\times b\times c}\}, \\ \quad \min\limits_{\substack{1\le b_1<b \\ 0\le t_1<t}} \{P^{t_1}_{a\times b_1\times c} + P^{t-t_1}_{a\times (b-b_1)\times c}\}, & \text{если } (\min(a,b,c)>1) \\ \quad \min\limits_{\substack{1\le c_1<c \\ 0\le t_1<t}} \{P^{t_1}_{a\times b\times c_1} + P^{t-t_1}_{a\times b\times (c-c_1)}\}\Big] + 1 & \quad \wedge (0<t<abc). \end{cases} \tag{3.13}$$

Используя формулы (3.11) – (3.13), можно вычислить значения $P^t_{a\times b\times c}$. Прямое вычисление этих значений сопряжено с экспоненциально возрастающими вычислительными затратами. Для ускорения вычислений можно использовать некоторые простые свойства $P^t_{a\times b\times c}$, сформулированные в следующе лемме:

**Лемма 3.2.** *Значения $P^t_{a\times b\times c}$ минимального количества разбиений блока расчетной сетки $a\times b\times c$ для выделения части размера $t$ обладают следующими свойствами:*

1. *Симметричность относительно аргументов $a$, $b$, $c$. Верны соотношения $P^t_{a\times b\times c} = P^t_{a\times c\times b} = P^t_{b\times a\times c} = P^t_{b\times c\times a} = P^t_{c\times a\times b} = P^t_{c\times b\times a}$.*
2. *Симметричность относительно аргумента $t$. Для всех допустимых значений параметров и при условии $0 \le t \le abc$ имеет место соотношение $P^t_{a\times b\times c} = P^{abc-t}_{a\times b\times c}$.*
3. *Ограниченность. Для всех допустимых значениях параметров и при условии $0 \le t \le abc$ верно ограничение $P^t_{a\times b\times c} \le 5$.*

Первое свойство следует непосредственно из того, что все блоки $a\times b\times c$, $a\times c\times b$, $b\times a\times c$, $b\times c\times a$, $c\times a\times b$, $c\times b\times a$ являются одним и тем же блоком с точностью до порядка перечисления размеров сторон.

Второе свойство следует из того, что отделение от блока $a \times b \times c$ (имеющего размер $abc$) части размера $t$, означает разделение его на две части – размерами $t$ и $abc-t$, поэтому выделение части размера $t$ это то же самое, что и выделение части размера $abc - t$.

Ограниченность $P_a^t$ и $P_{a\times b}^t$ следует непосредственно из формул (3.11) – (3.12), из которых нетрудно видеть, что $P_a^t \le 1$ при $t \le a$, $P_{a\times b}^t \le 3$ при $t \le ab$. Для получения оценки $P_{a\times b\times c}^t \le 5$ построим разбиение, на которое гарантированно не потребуется более пяти разрезов. Если $t = 0$ или $t = abc$, то ничего не делаем, иначе продолжаем. Первым разрезом отрежем от блока $a\times b\times c$ часть $\lfloor \frac{t}{bc} \rfloor \times b \times c$. Нам осталось выделить из блока $\left(a - \lfloor \frac{t}{bc} \rfloor\right) \times b \times c$ часть размера $t_1 = t \bmod bc$. Если $t_1 = 0$, то останавливаемся, иначе продолжаем. Вторым разрезом от блока $\left(a - \lfloor \frac{t}{bc} \rfloor\right) \times b \times c$ отрезаем $1 \times b \times c$ и далее работаем с ним. Третьим разрезом от блока $1 \times b \times c$ отрезаем часть $1 \times \lfloor \frac{t_1}{c} \rfloor \times c$. Теперь осталось от блока $1 \times \left(b - \lfloor \frac{t_1}{c} \rfloor\right) \times c$ отрезать часть размера $t_2 = t_1 \bmod c$. Если $t_2 = 0$, то останавливаемся, иначе продолжаем. Четвертым разрезом от блока $1 \times \left(b - \lfloor \frac{t_1}{c} \rfloor\right) \times c$ отрезаем $1 \times 1 \times c$ и уже от него пятым разрезом отрезаем недостающую часть $t_2$. В результате выполненных дроблений выделена часть

$$\left\lfloor \frac{t}{bc} \right\rfloor bc + \left\lfloor \frac{t_1}{c} \right\rfloor c + t_2 = (t - (t \bmod bc)) + ((t \bmod bc) - ((t \bmod bc) \bmod c)) + ((t \bmod bc) \bmod c) = t. \quad (3.14)$$

Схема дроблений, представленная на (3.14), соответствует применению 5 разрезов, откуда следует требуемое утверждение. ■

На рис. 3.10 приведен характерный вид графика функции $f(t) = P_{n\times n\times n}^t$ на примере $n = 6$. На приведенном рисунке наиболее распространенными значениями $P_{6\times 6\times 6}^t$ являются значения 3 и 4.

Из свойств 1 и 2 леммы 3.2 следует, что достаточно вычислить значения $P_{a\times b\times c}^t$ при ограничениях $a \ge b \ge c$, $t \le abc$. Для возможности эффективного вычисления $P_{a\times b\times c}^t$ можно использовать принцип динамического программирования, то есть сохранение ранее вычисленных значений для последующего

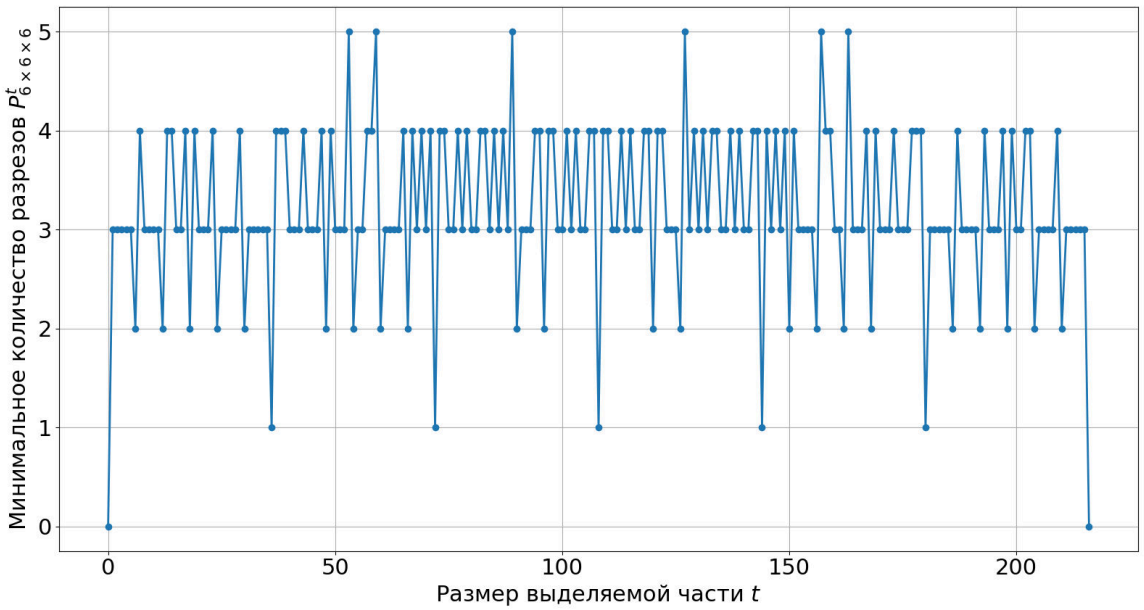


Рис. 3.10. График зависимости $P^t_{6\times6\times6}$ от $0 \le t \le 6^3$

повторного использования в рекурсивных вызовах. Размер памяти, требуемой для сохранения промежуточных значений $P^t_{a\times b\times c}$ при ограничениях $a \ge b \ge c$, $t \le abc$ может быть вычислен как

$$\sum_{a_i=1}^{a}\sum_{b_i=1}^{a_i}\sum_{c_i=1}^{b_i}\frac{a_i b_i c_i}{2} = \frac{1}{96}a^6 + o(a^6), \tag{3.15}$$

что говорит о слишком больших накладных расходах по памяти. Таким образом, точное определение количества разрезов и их координат для выделения из блока части заданного размера, нецелесообразно с точки зрения производительности вычислений.

Если предположить, что мы можем вычислить все требуемые значения $P^t_{a\times b\times c}$, а также все координаты требуемых разрезов блоков для выделения из блока $a \times b \times c$ части размера $t$, то равномерное распределение блоков расчетной сетки между произвольным количеством вычислительных процессов, очевидно, может быть получено с показателем $D = 0$ (если $k \mid n$). При этом, в общем случае, для каждого процесса кроме последнего необходимо будет выполнять дробление последнего добавляемого в него блока для выделения из этого блока части требуемого размера. Общее количество разрезов, необ-

ходимое для достижения равномерного распределения можно оценить, что сформулировано в следующей лемме:

**Лемма 3.3.** *При распределении произвольного количества блоков расчетной сетки с суммарным количеством ячеек $n$ по $k$ партициям и выполнении условия $k \mid n$ равномерное распределение с показателем $D = 0$ может быть достигнуто с использованием не более $5(k-1)$ разрезов блоков.*

Справедливость утверждения следует непосредственно из того, что в худшем случае для каждой партиции кроме последней потребуется выделение из некоторого блока части требуемого размера (для достижения наполнения этой партиции ровно $\frac{n}{k}$ ячейками) и на каждый такой блок потребуется не более 5 разрезов, согласно лемме 3.2. ■

Рассмотренный вариант построения распределения блоков расчетной сетки по партициям с использованием дробления блоков хоть и приводит к равномерному распределению с показателем $D = 0$, однако обладает рядом недостатков, что делает его непригодным при использовании в практических приложениях. Поиск значений $P^t_{a\times b\times c}$ требует слишком высоких затрат по времени вычисления и по памяти для хранения уже вычисленных значений. С учетом выполненных оценок и предположений количество разрезов блоков, требуемых для построения распределения блоков по партициям, кратно превосходит количество партиций. В процессе дробления блоков могут появляться блоки со слишком маленькими размерами по одному или нескольким измерениям, что приводит к возрастанию теневых зон и увеличению объема межпроцессных обменов.

Ввиду сделанных выводов требуется рассмотрение приближенных методов распределения блоков между партициями, которые могут быть использованы на практике. Будем вести поиск таких методов распределения, при которых показатель $D$ остается на достаточно низком уровне при существенном сокращении количества дроблений блоков.

### 3.3.4 Метод распределения блоков сетки по вычислительным процессам с использованием дробления блоков, основанный на алгоритме распределения множества блоков по партициям с помощью дробления наибольших блоков

Рассмотрим стратегию распределения блоков по партициям с использованием дробления. Для распределения будем использовать жадный алгоритм. Если после распределения с использованием жадного алгоритма показатель $D^*$ окажется выше требуемого значения, то выполним разрез наиболее крупного блока пополам по наиболее протяженному направлению и повторим распределение. Без ограничения общности будет считать, что размеры блока отсортированы по убыванию (то есть наибольшим размером блока является размер $a$). Для дробления блока будем использовать следующее обозначение $\{b_1, b_2\} = b.cut(i)$, означающее дробление блока $b$ на два блока путем разреза по наиболее протяженному измерению в позиции $i$ (при котором сторона длины $a$ делится на две части размерами $i$ и $a - i$).

**Алгоритм 3.4.** *Алгоритм распределения множества блоков по партициям с помощью дробления наибольших блоков (`DistributeHalfMaxBlock`)*

**Вход:** $B$ – множество блоков ($|B| = m$), $k$ – количество результирующих партиций, $\varepsilon$ – допустимое значение показателя $D^*$.
**Выход:** $P$ – множество результирующих партиций.

```
1  Функция DistributeHalfMaxBlock(B, k, ε):
2      while true do
3          P ← DistributeGreedy(B, k)
4          if P.D* ≤ ε then
5              return P
6          end
7          else
8              b ← B.max
9              B ← (B \ {b}) ∪ b.cut(⌊b.a/2⌋)
10         end
11     end
```

Так как блоки можно раздробить хоть до размера $1\times1\times1$, то можно добиться распределения с оптимальным $D^*$. Для определения сложности алгоритма примем количество разрезов за $\sigma$. Алгоритм состоит из последовательного применения `DistributeGreedy` (алгоритм 3.3) с увеличением количества блоков от $m$ до $m+\sigma$ и вычислением значения $P.D^*$ (выполняется со сложностью $O(k)$, чем можно пренебречь). При этом подготовка множества блоков перед первым применением `DistributeGreedy` имеет сложность $O(m\log m)$, как и раньше. Перед последующими применениями `DistributeGreedy` подготовка множества блоков заключается в изъятии из исходного множества максимального блока, его дроблении и помещении двух блоков обратно, что имеет логарифмическую сложность от текущего количества блоков. Аналогично, подготовка множества партиций перед первым применением `DistributeGreedy` имеет сложность $O(k\log k)$, а перед последующими – $O(k)$ (копирование всей структуры множества партиций целиком). Сложность отдельных составляющих работы алгоритма приведена в таблице 3.1.

Таблица 3.1. Сложность подготовки множества $B$, множества $P$ и работы алгоритма `DistributeGreedy` на разных итерациях цикла в алгоритме `DistributeHalfMaxBlock`

| количество разрезов | подготовка $B$ | подготовка $P$ | алгоритм |
|---|---|---|---|
| 0 | $O(m\log m)$ | $O(k\log k)$ | $O(m\log k)$ |
| 1 | $O(\log m)$ | $O(k)$ | $O((m+1)\log k)$ |
| ... | ... | ... | ... |
| $i$ | $O(\log(m+i-1))$ | $O(k)$ | $O((m+i)\log k)$ |
| ... | ... | ... | ... |
| $\sigma$ | $O(\log(m+\sigma-1))$ | $O(k)$ | $O((m+\sigma)\log k)$ |

Просуммировав все выражения для сложности отдельных составляющих частей алгоритма при разном количестве дроблений, получим

$$m\log m+\sum_{i=1}^{\sigma}\log(m+i-1)+k\log k+\sigma k+\sum_{i=0}^{\sigma}(m+i)\log k\le$$
$$m\log m+\sigma\log(m+\sigma)+k\log k+\sigma k+(\sigma+1)(m+\sigma)\log k, \quad (3.16)$$

откуда с учетом допущения $m > k$ выражение для сложности принимает вид $O(m \log m + \sigma(m + \sigma) \log k)$. ■

Предложенный алгоритм `DistributeHalfMaxBlock` реализован в программе «GridMaster» подготовки блочно-структурированной расчетной сетки для проведения расчетов на суперкомпьютере [129] и используется при проведении газодинамических расчетов [301,306,307] с помощью ПО «Лазурит». Алгоритм был протестирован на различных расчетных сетках.

Ниже представлен результат тестирования на двух сетках. Первая рассматриваемая сетка содержит 13 блоков, 80 интерфейсов, 148 граничных условий, 13 областей начальных условий и 5,75 млн ячеек, размер вычислительной окрестности равен 3. Вторая рассматриваемая сетка содержит 300 блоков, 1796 интерфейсов, 1643 граничных условия, 300 областей начальных данных и 94,34 млн ячеек. Размер вычислительной окрестности также равен 3.

Для этих сеток были проведено распределение блоков для 128 партиций. Статистика распределения блоков по вычислительным процессам для двух различных вариантов: без использования дробления блоков и с использованием дроблений до достижения показателя неравномерности распределения не более $D^* = 0{,}1$ (рис. 3.11).

Предложенный алгоритм распределения блоков расчетной сетки между вычислительными процессами приводит к более равномерной загрузке вычислительных ресурсов суперкомпьютерного кластера (снижение значения показателя $D$), что повышает эффективность его использования в расчетных задачах. Часто применение такого подхода на большом количестве процессов приводит к кратному ускорению вычислений. Особенно это актуально для сеток содержащих небольшое количество блоков или для сеток, имеющих ярко выраженные крупные блоки.

Негативным фактором применения дробления блоков является увеличение количества блоков и количества граничных ячеек, что приводит к увеличению объема данных межпроцессных обменов. Проанализируем, как увеличение количества блоков и граничных ячеек влияет на производительность вычис-

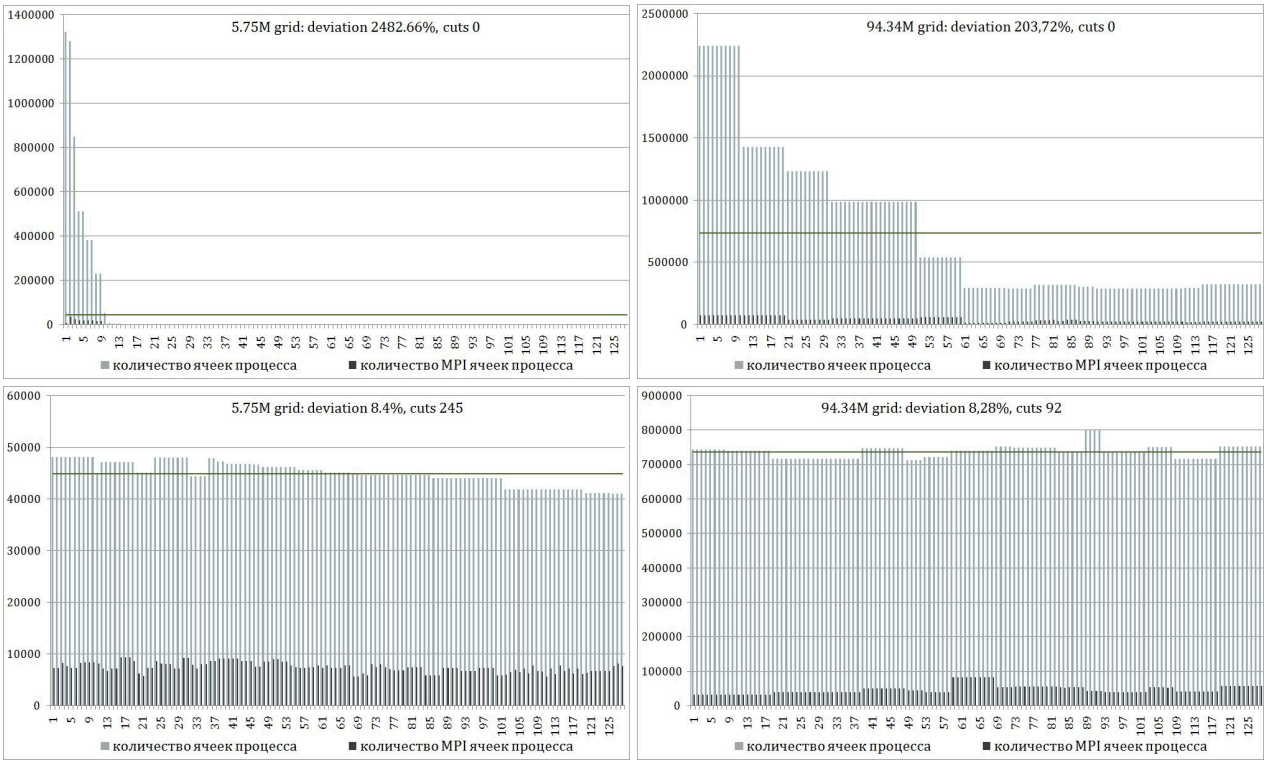


Рис. 3.11. Распределение блоков тестовых сеток (5,75 млн ячеек – слева, 94,34 млн ячеек – справа) без дробления (наверху) и с дроблением с допустимым отклонением $D^* = 0{,}1$ (снизу)

лений. При этом последовательность дробления блоков сетки для достижения требуемого целевого значения показателя $D^*$ при распределении блоков между $k$ процессами будем называть подготовкой сетки.

Для анализа эффективности распределения блоков по вычислительным процессам с использованием дробления блоков и влияниях этого распределения на масштабирование вычислений был поставлен эксперимент по выполнению газодинамических расчетов методом RANS/ILES (ПО «Лазурит») на блочно-структурированной расчетной сетке, содержащей 9,4 млн ячеек [301]. Эксперимент проводился сегменте суперкомпьютера МВС-10П МП2, состоящем из узлов, каждый из которых содержит по одному микропроцессору Intel Xeon Phi 7290 KNL [311, 312] (характеристики вычислительных узлов системы МВС-10П МП2 приведены в таблице 3.2).

Перед вычислениями выполнялась подготовка расчетной сетки для $k = 16$, $k = 32$ и $k = 64$ путем распределения вычислительной нагрузки с дроблением наиболее крупных блоков сетки. При подготовке расчетной сетки исполь-

Таблица 3.2. Характеристики вычислительных узлов МВС-10П МП2 на базе микропроцессора Intel Xeon Phi 7290 KNL

| Процессор | Intel Xeon Phi 7290 KNL |
|---|---|
| Частота процессора | 1,5 ГГц |
| Количество ядер процессора | 72 |
| Количество виртуальных процессоров | 288 |
| Пиковая производительность процессора | 3,456 Тфлопс |
| Кэш 1-го уровня | 32 КБ на ядро |
| Кэш 2-го уровня | 512 КБ на ядро |
| Количество процессоров в узле | 1 |
| Количество ядер в узле | 72 |
| Количество виртуальных процессоров в узле | 288 |
| Пиковая производительность в узле | 3,456 Тфлопс |
| Объем оперативной памяти в узле | 96 ГБ |
| Коммуникационная и транспортная сеть | Onmi-Path |
| Сеть мониторинга и управления | Gigabit Ethernet |
| Сеть управления заданиями | Gigabit Ethernet |
| Системы управления прохождением заданий | SLURM/СУППЗ |

зовалось допустимое значение неравномерности распределения блоков по процессам $D^* = 0,1$, для подготовки вычислений для $k = 64$ дополнительно использовалось допустимое отклонение $D^* = 0,01$ (характеристики расчетных сеток приведены в таблице 3.3).

Таблица 3.3. Характеристики используемых расчетных сеток, после подготовки для разных значений $k$ и $D^*$

| Расчетная сетка | Блоки | Интерфейсы | Граничные условия |
|---|---|---|---|
| исходная сетка (9,4 млн ячеек) | 30 | 152 | 204 |
| подготовленная сетка $k = 16$, $D^* = 0,1$ | 39 | 224 | 218 |
| подготовленная сетка $k = 32$, $D^* = 0,1$ | 54 | 340 | 248 |
| подготовленная сетка $k = 64$, $D^* = 0,1$ | 170 | 982 | 429 |
| подготовленная сетка $k = 64$, $D^* = 0,01$ | 383 | 2356 | 682 |

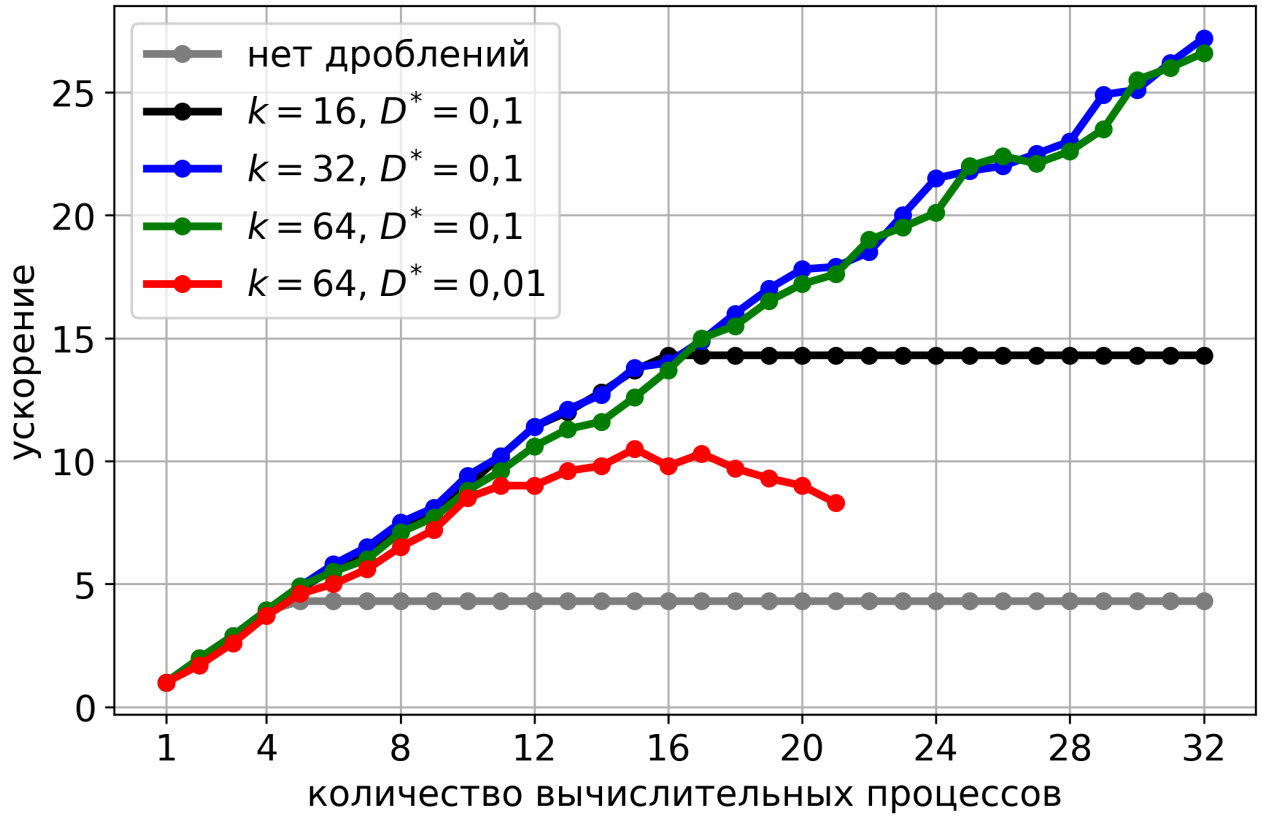


Рис. 3.12. Ускорение вычислений при различных параметрах подготовки сетки на 1-32 микропроцессорах Intel Xeon Phi KNL

На рис. 3.12 представлены результаты численных экспериментов на сегменте суперкомпьютера МВС-10П МП2, состоящем из узлов, каждый из которых содержит по одному микропроцессору Intel Xeon Phi 7290 KNL. При проведении расчетов на каждом узле запускался один MPI-процесс. Количество узлов менялось от 1 до 32. Проанализируем графики ускорения вычислений, представленные на рис. 3.12.

Для вычислений на неподготовленной сетке (на рис. 3.12 соответствует графику "нет дроблений") наблюдается ускорение для количества процессов до 5. Далее ускорение остается на одной и той же отметке (около 4,3) и не меняется при дальнейшем увеличении количества узлов. Это связано с наличием крупного блока, который мешает равномерному распределению вычислительной нагрузки, так как время расчета ограничено снизу временем обработки этого крупного блока одним вычислительным процессом.

При подготовке сетки для $k = 16$ (черный график на рис. 3.12) ускорение

также останавливается, но на более высокой отметке (в районе 14,0). Это также связано с наличием крупного блока, но его размер меньше, чем в случае отказа от дроблений (наиболее крупный блок был раздроблен, что привело к эффективному распределению вычислительной нагрузки для 16 процессов, однако для большего количества процессов размер этого блока препятствует равномерному распределению блоков по процессам).

При подготовке сетки для $k = 32$ и $k = 64$ с допустимым отклонением $D^* = 0,1$ (синий и зеленый графики на рис. 3.12) получаем примерно одинаковое возрастание ускорения вычислений с ростом количества узлов. Это говорит о том, что при подготовка сетки для большего количества процессов, чем реально будет использоваться для запусков, избыточно.

При подготовке сетки для $k = 64$ с допустимым отклонением $D^* = 0,01$ (красный график на рис. 3.12) наблюдается сильная деградация ускорения и эффективности распараллеливания при использовании количества узлов выше 10. Это связано с сильным возрастанием количества блоков, граничных ячеек, что привело к увеличению объема данных при межпроцессных обменах.

Из графиков, приведенных на рис. 3.12, можно сделать вывод, что при использовании распределения вычислительной нагрузки с дроблением блоков целесообразно придерживаться двух эвристических правил. Во-первых, подготовку расчетной сетки не следует производить для количества процессов, больше того, которое реально будет использоваться при запусках. Во-вторых, не стоит выбирать слишком низкий показатель допустимого отклонения $D^*$, так как в случае слишком низкого значения этого показателя равномерность распределения вычислительной нагрузки будет нивелирована возрастанием объема межпроцессных обменов.

Так как подготовка расчетной сетки с помощью дробления наиболее крупных блоков при снижении показателя $D^*$ приводит к сильному увеличению количества блоков и объема данных межпроцессных обменов, то требуется использование других подходов к дроблению блоков сетки с целью уменьшения количества производимых разрезов блоков.

### 3.3.5 Метод распределения блоков сетки по вычислительным процессам с использованием дробления блоков, основанный на параметрическом алгоритме распределения блоков по партициям с уменьшением количества дроблений

В разделе предложен метод распределения блоков сетки по вычислительным процессам с использованием дробления блоков, основанный на параметрическом алгоритме распределения блоков по партициям с уменьшением количества дроблений. Основной идеей алгоритма является выполнения дробления блока непосредственно перед добавлением в партицию.

Для устранения крупных блоков без лишних дроблений предлагается алгоритм распределения блоков расчетной сетки по партициям с использованием дробления блоков и уменьшающий количество дроблений [306, 307]. Прежде всего определим, на какие части может быть разрезан конкретный блок. Без ограничения общности будем считать, что размеры блока $b$ отсортированы по убыванию, и сторона по первому измерению $b.a$ является наибольшей.

На возможные разрезы блока будем накладывать следующие ограничения. Во первых, будем выполнять разрезы только по наиболее протяженному измерению (то есть разбивать наибольшую сторону размера $b.a$ на две части размера $a_1$ и $b.a - a_1$), так как это приводит к уменьшению количества интерфейсных ячеек. Для запрета появления слишком тонких блоков будем запрещать выполнять разрезы блока слишком близко к границе. Для этого вводится параметр $\delta$, по которому разрешается деление блока только в сегменте $[\delta, b.a - \delta]$. Вводится еще одно ограничение, не позволяющее производить слишком мелкие блоки, – наименьшее допустимое количество ячеек в результирующем блоке, при котором разрешено дробление, это ограничение будем обозначать $\Delta$.

**Определение 3.9.** *Позицию разреза $i$ блока $b$ вдоль наиболее протяженного направления $a$ будем называть допустимой, если $i \in [\delta, b.a - \delta]$ и $\min(i, b.a - i)\frac{b.w}{b.a} \geq \Delta$ (то есть разрез выполнен вдали от границы блока и в результате разрезания блока не порождаются слишком мелкие блоки).*

Для реализации алгоритма с уменьшением количества дроблений блоков рассмотрим четыре основные операции, используемые в алгоритме.

Первая операция: для заданного множества блоков $B$ и множества партиций $P$ найти такой блок $b \in B$ и партицию $p \in P$, что величина $p.w + b.w$ является максимально возможной, но не превышающей заданное значение $\tau$. Эту операцию будем называть заполнением партиции целым блоком снизу (`FillFullUnder`).

Вторая операция: для заданного множества блоков $B$ и множества партиций $P$ найти такой блок $b \in B$ и партицию $p \in P$, а также допустимую позицию разреза $i$, что величина $p.w + b.w\frac{i}{b.a}$ является максимально возможной, но не превышающей заданное значение $\tau$. Эту операцию будем называть заполнением партиции частью блока снизу (`FillPartUnder`).

Третья операция: для заданного множества блоков $B$ и множества партиций $P$ найти такой блок $b \in B$ и партицию $p \in P$, что величина $p.w + b.w$ является минимально возможной превышающей заданное значение $\tau$. Эту операцию будем называть заполнением партиции целым блоков сверху (`FillFullAbove`).

Четвертая операция: для заданного множества блоков $B$ и множества партиций $P$ найти такой блок $b \in B$ и партицию $p \in P$, а также допустимую позицию разреза $i$, что величина $p.w + b.w\frac{i}{b.a}$ является минимально возможной превышающей заданное значение $\tau$. Эту операцию будем называть заполнением партиции частью блока сверху (`FillPartAbove`).

Тогда алгоритм распределения множества блоков по партициям будем определять как последовательность попыток произвести одну из введенных четырех операций, пока множество распределяемых блоков не станет пустым $B$. При этом параметр $\tau$ несет смысл максимально допустимого размера партиции на текущий момент. Понятно, что при выполнении операции заполнения партиции целым блоком множество $B$ уменьшается, тогда как при заполнении партиции частью блока размер множества $B$ не меняется. Приведем формальное описание алгоритма с использованием операций заполнения партиций целыми блоками и частями блоков снизу и сверху.

**Алгоритм 3.5.** *Параметрический алгоритм распределения блоков по партициям с уменьшением количества дроблений (`DistributeMinCuts`), который будем обозначать* $\mathscr{A}_{U/u}^{A/a}$

**Вход:** $B$ – множество блоков, $k$ – количество партиций, $\delta$ – ограничение на позицию разреза, $\Delta$ – минимальный размер блока, $U$ – признак использования `FillFullUnder`, $u$ – признак использования `FillPartUnder`, $A$ – признак использования `FillFullAbove`, $a$ – признак использование `FillPartAbove`.
**Выход:** $P$ – множество результирующих партиций.

**Функция** `DistributeMinCuts`<$U$, $u$, $A$, $a$>($B$, $k$, $\delta$, $\Delta$)**:**
  $P \leftarrow \{p : p.B = \emptyset, \forall i \in [0, k-1]\}$
  $\tau \leftarrow \frac{B.w}{k}$
  **while** $B \neq \emptyset$ **do**
    **if** $U \wedge (((b,p) \leftarrow FillFullUnder(B,P,\tau)) \neq null)$ **then**
      $(B, p.B) \leftarrow (B \setminus \{b\}, p.B \cup \{b\})$
      **continue**
    **end**
    **if** $u \wedge (((b,p,i) \leftarrow FillPartUnder(B,P,\tau,\delta,\Delta)) \neq null)$ **then**
      $\{b_1, b_2\} \leftarrow b.cut(i)$
      $(B, p.B) \leftarrow ((B \setminus \{b\}) \cup \{b_2\}, p.B \cup \{b_1\})$
      **continue**
    **end**
    **if** $a \wedge (((b,p,i) \leftarrow FillPartAbove(B,P,\tau,\delta,\Delta)) \neq null)$ **then**
      $\{b_1, b_2\} \leftarrow b.cut(i)$
      $(B, p.B) \leftarrow ((B \setminus \{b\}) \cup \{b_2\}, p.B \cup \{b_1\})$
      $\tau \leftarrow p.w$
      **continue**
    **end**
    **if** $A \wedge (((b,p) \leftarrow FillFullAbove(B,P,\tau)) \neq null)$ **then**
      $(B, p.B) \leftarrow (B \setminus \{b\}, p.B \cup \{b\})$
      $\tau \leftarrow p.w$
      **continue**
    **end**
  **end**
  **return** $P$

Операция заполнения снизу без дробления `FillFullUnder` аналогична известной задаче поиска в двух упорядоченных массивах $A$ и $B$ пары элементов $a \in A, b \in B$ с максимальной суммой, но не превосходящей заданного значения.

Для поиска такой пары достаточно рассмотреть каждый из элементов одного массива (например, $A$) и найти для него наилучшего кандидата из второго массива для составления пары. Поиск кандидата из второго массива выполняется с помощью бинарного поиска с логарифмической сложностью. Таким образом, сложность операции `FillFullUnder` (и аналогичной ей операции `FillFullAbove`) составляет $O(k \log m)$.

С операциями `FillPartUnder` и `FillPartAbove` дело обстоит несколько сложнее, так как следует учитывать выполнение разреза выбираемого блока. В общем случае для выполнения этих операций нужно рассмотреть каждую пару (блок, партиция) и для этой пары найти наилучший разрез блока со сложностью $O(1)$. Таким образом, сложность операций `FillPartUnder` и `FillPartAbove` составляет $O(mk)$.

Если количество разрезов блоков равно $\sigma$, то алгоритм выполняется $m + \sigma$ итераций, на каждой из которых обрабатывается не более $m$ блоков и ровно $k$ партиций, откуда получим сложность выполнения всех итераций алгоритма $O((m + \sigma)mk)$. Так как удаление и добавление элементов в упорядоченные множества имеет логарифмическую сложность, то этими действиями можно пренебречь, то есть общая сложность алгоритма составляет $O((m + \sigma)mk)$.

Порядок применения операций на каждой итерации алгоритма не случаен. Применение сначала операций заполнения снизу продиктовано сдерживанием значения $\tau$, которое влияет на равномерность распределения весов по партициям. По этой же причине операция `FillPartAbove` применяется раньше `FillFullAbove`. Для уменьшения количества разрезов блоков выполнение операции `FillFullUnder` применяется раньше `FillPartUnder`.

Параметрический алгоритм $\mathscr{A}_{U/u}^{A/a}$ представляет собой группу из 16 алгоритмов, однако не все из них являются жизнеспособными. Для того, чтобы алгоритм гарантированно завершил работу, необходимо применение заполнения целыми блоками как снизу, так и сверху, поэтому $U = A = true$. Для того, чтобы алгоритм являлся алгоритмом распределения с дроблением блоков необходимо выполнения условия $u \vee a = true$. Таким образом, получим группу из трех алгоритмов, $\mathscr{A}_{1/0}^{1/1}$, $\mathscr{A}_{1/1}^{1/0}$, $\mathscr{A}_{1/1}^{1/1}$. ■


Рис. 3.13. Графики сравнения алгоритмов $\mathscr{A}_{1/0}^{1/1}$, $\mathscr{A}_{1/1}^{1/0}$, $\mathscr{A}_{1/1}^{1/1}$ по неравномерности распределения $D^*$ и количеству дроблений при распределении 10 случайных блоков по 16 партициям

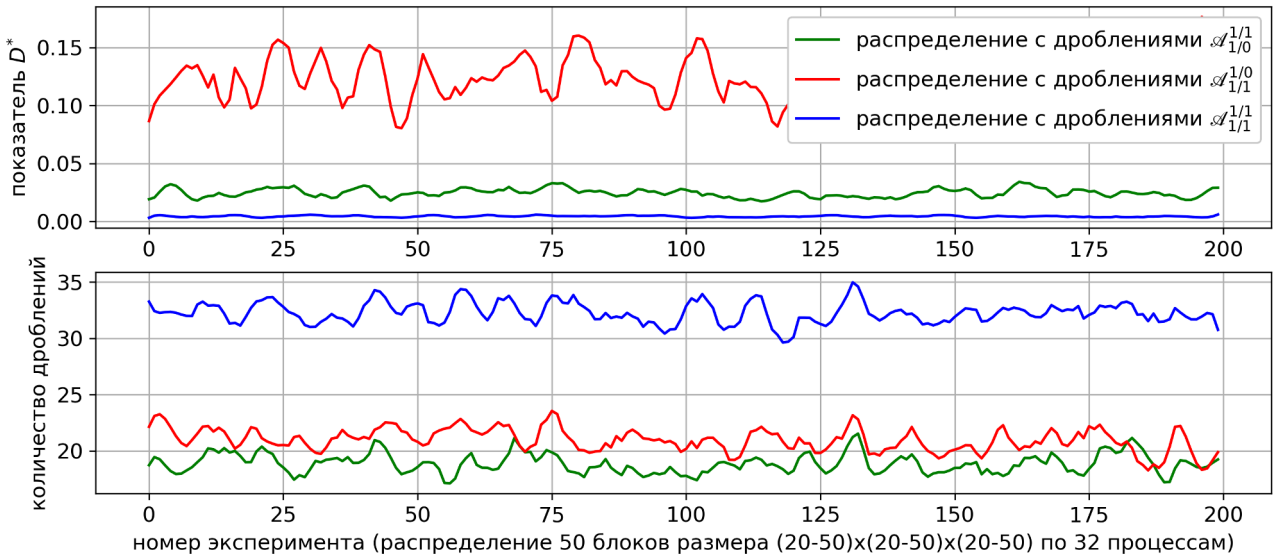


Рис. 3.14. Графики сравнения алгоритмов $\mathscr{A}_{1/0}^{1/1}$, $\mathscr{A}_{1/1}^{1/0}$, $\mathscr{A}_{1/1}^{1/1}$ по неравномерности распределения $D^*$ и количеству дроблений при распределении 50 случайных блоков по 32 партициям

Результаты работы алгоритмов $\mathscr{A}_{1/0}^{1/1}$, $\mathscr{A}_{1/1}^{1/0}$, $\mathscr{A}_{1/1}^{1/1}$ сравнивались между собой по двум характеристикам: показателю неравномерности распределения блоков по партициям $D^*$ и количеству разрезов $\sigma$. Сравнение выполнялось

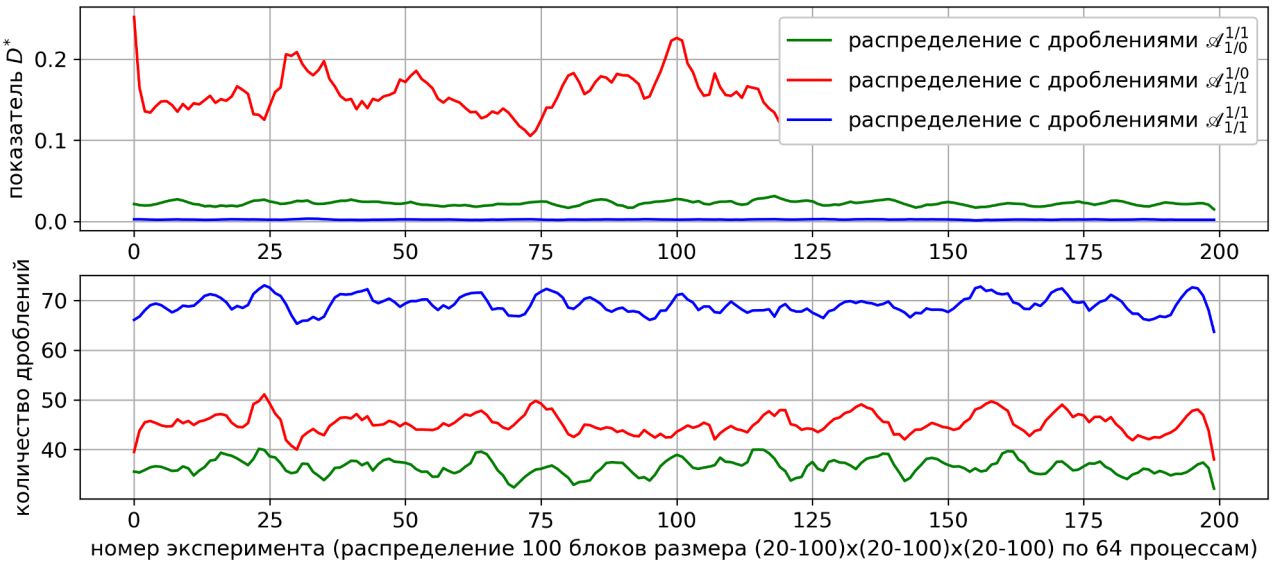


Рис. 3.15. Графики сравнения алгоритмов $\mathscr{A}_{1/0}^{1/1}$, $\mathscr{A}_{1/1}^{1/0}$, $\mathscr{A}_{1/1}^{1/1}$ по неравномерности распределения $D^*$ и количеству дроблений при распределении 100 случайных блоков по 64 партициям

на наборах случайно сгенерированных блоков со сторонами, размер которых изменяется в заданных пределах. Рассматривались следующие примеры тестовых множеств блоков: распределение 10 блоков по сторонами размера в диапазоне 20-30 по 16 партициям (рис. 3.13), распределение 50 блоков со сторонами размера в диапазоне 20-50 по 32 партициям (рис. 3.14), распределение 100 блоков со сторонами размера в диапазоне 20-100 по 100 партициям (рис. 3.15). Во всех случаях были приняты одни и те же значения других параметров $\delta = 3$, $\Delta = 64$.

Из рисунков 3.13 – 3.15 можно сделать следующие выводы. Алгоритм $\mathscr{A}_{1/1}^{1/1}$ порождает наиболее равномерное распределение и приводит к наибольшему количеству дроблений, так как он использует обе операции заполнения частями блоков (`FillPartUnder` и `FillPartAbove`). Алгоритм $\mathscr{A}_{1/1}^{1/0}$ порождает наименее равномерное распределение, так как для заполнения сверху он использует только целые блоки (`FillFullAbove`), что приводит к более быстрому росту ограничения $\tau$ в процессе работы алгоритма, чем если бы использовалась операция `FillPartAbove`. Алгоритм $\mathscr{A}_{1/0}^{1/1}$ является наиболее предпочтительным, так как по равномерности распределения он не сильно уступает алгоритму $\mathscr{A}_{1/1}^{1/1}$, а по количеству разрезов является наилучшим.

Предложенный алгоритм `DistributeMinCuts` реализован в программе «GridMaster» подготовки блочно-структурированной расчетной сетки для проведения расчетов на суперкомпьютере [129] и используется при проведении газодинамических расчетов [301, 306, 307].

## 3.4 Выводы

Разработаны архитектура объемной блочно-структурированной сетки и методы распределения блоков сетки по вычислительным процессам с использованием дробления блоков для распараллеливания вычислений на распределенной памяти. Разработанная архитектура поддерживает основные объекты сетки: блоки, интерфейсы касания соседних блоков, области задания начальных и граничных условий, связанные граничные условия. Архитектура также поддерживает механизм дробления блоков и связанных с разрезаемым блоком объектов сетки. Разработанные методы распределения блоков по вычислительным процессам основаны на механизме дробления блоков сетки и алгоритмах распределения множества блоков по партициям: на алгоритме распределения множества блоков по партициям с помощью дробления наибольших блоков и на параметрическом алгоритме распределения блоков по партициям с уменьшением количества дроблений. Параметрический алгоритм представляет собой семейство алгоритмов, тестирование которых на наборах случайных блоков позволило выбрать экземпляр, сочетающий в себе низкий показатель неравномерности распределения блоков между партициями с количеством дроблений, сравнимым с количеством блоков сетки. Разработанные методы реализованы в программе «GridMaster» подготовки блочно-структурированной расчетной сетки для проведения расчетов на суперкомпьютере и используются при проведении газодинамических численных расчетов. Методы распределения блоков сетки по вычислительным процессам позволяют преодолеть проблему низкой производительности вычислений из-за наличия крупных блоков и снизить показатель неравномерности распределения вычислительной нагрузки между процессами.

# Глава 4. Вычисления на поверхностной сетке

Глава посвящена задаче разработки методов повышения производительности параллельных вычислений на поверхностной неструктурированной сетке, описывающей границу расчетной подобласти при моделировании объектов со сложной геометрией.

## 4.1 Декомпозиция поверхностной неструктурированной сетки

В предыдущей главе рассматривалась проблема распараллеливания вычислений на блочно-структурированной сетке на распределенной памяти. Для поверхностной неструктурированной сетки стоит аналогичная задача распараллеливания вычислений на распределенной памяти, и для этого сетка должна быть разделена (декомпозирована) на отдельные домены (множества ячеек), которые обрабатываются в своих процессах и обмениваются друг с другом данными на общих границах. В разделе будем рассматривать подходы к декомпозиции неструктурированной поверхностной сетки с треугольными ячейками [314]. Для ускорения расчетов ячейки сетки разбиваются на отдельные домены, при этом нужно стремиться к уменьшению значений характеристик качества декомпозиции $D$, $L$, $I$, которые были введены в разделе 3.1. В отличие от блочно-структурированных сеток у рассматриваемой неструктурированной сетки ячейки не объединяются в блоки, а являются независимыми объектами проведения расчетов. Для неструктурированных поверхностных сеток будем считать, что вычислительная окрестность каждой ячейки будет включать кроме нее самом только три смежные ячейки (граничащие с рассматриваемой по каждому из трех ребер).

### 4.1.1 Обзор методов декомпозиции

Кратко рассмотрим методы декомпозиции на примере тестовой трехмерной поверхностной сетки (wing), состоящей из ячеек-треугольников и представляющей собой профиль крыла летательного аппарата, полученный из двумерного профиля NACA 0012 [315]. Сетка содержит порядка $n = 2 \cdot 10^4$ ячеек.

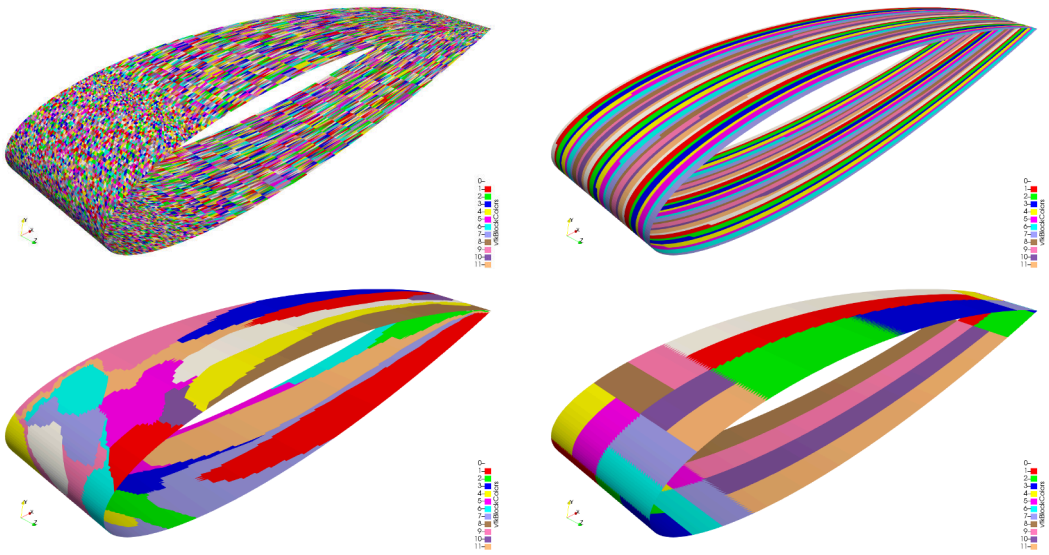

Рис. 4.1. Результат декомпозиции расчетной сетки на 32 домена: сверху вниз слева направо – случайное распределение, линейное распределение, наращивание доменов и иерархическое дробление

При случайном распределении ячеек сетки по доменам (рис. 4.1 сверху слева) получим распределение, которое имеет низкое значение показателя $D$, однако в этом случае практически все ребра с высокой долей вероятности являются междоменными, что порождает большой объем данных, которыми нужно обмениваться при синхронизации вычислений, поэтому такое распределение неприменимо при организации параллельных вычислений.

Рассмотрим семейство методов декомпозиции, в которых учитываются только индексы распределяемых ячеек и не учитываются другие данные. Такие алгоритмы не могут претендовать на высокое качество, так как их результат зависит от расположения данных сетки в памяти. Самым простым из алгоритмов этого класса является алгоритм линейного распределения ячеек по доменам. Так как в каждый домен в среднем должно войти по $\frac{n}{k}$ ячеек, где $n$ – общее количество ячеек сетки, $k$ – количество доменов (для простоты не будем обращать внимание на то, что это число может быть нецелым), то можно ячейки с номерами из диапазона $[0, \frac{n}{k} - 1]$ отнести к первому домену, ячейки с номерами $[\frac{n}{k}, \frac{2n}{k} - 1]$ ко второму домену и так далее. При такой декомпозиции можно добиться равномерного распределения ячеек по доменам, однако значения характеристик $L$ и $I$ в общем случае предугадать

невозможно. Например, на рассматриваемой тестовой сетке видно, что деление на домены вдоль профиля крыла (как это показано на рис. 4.1 сверху справа) порождает очень длинные границы между соседними доменами. В общем случае ячейки с непрерывным диапазоном индексов не обязаны составлять не то что компактные домены с границами небольшой протяженности, а вообще могут порождать несвязные домены, домены с изолированными ячейками, выколотыми ячейками и другими дефектами.

Большой класс алгоритмов декомпозиции формируется из так называемых алгоритмов наращивания доменов. В основе этих алгоритмов лежит следующий принцип: вначале выбирается ячейка (или несколько ячеек), от которой (или которых) далее производится наращивание одного или нескольких доменов путем последовательного добавления соседних ячеек. В рамках этого класса алгоритмов при последовательном создании доменов размера $\frac{n}{k}$ получим алгоритм Фархата, результатом которого являются домены с очень протяженными границами [153]. При использовании одновременного роста сразу нескольких доменов от случайных ячеек расчетной сетки получим алгоритм пузырькового роста [316], который требуется запускать на расчетной сетке итерационно несколько раз с корректировкой инициирующих ячеек для получения приемлемых характеристик качества декомпозиции. При этом алгоритм пузырькового роста не гарантирует сбалансированного разбиения ячеек сетки по доменам (для достижения этого необходимо производить дополнительную коррекцию). Отметим инкрементальный алгоритм декомпозиции [317], особенностью которого является возможность освобождения части ячеек, уже распределенных по доменам, с повторением роста доменов.

Рассмотрим простой алгоритм наращивания доменов, в котором все домены наращиваются одновременно, начиная с некоторых случайно выбранных ячеек сетки. При этом поддерживается связность доменов – если в какой-то момент домену больше некуда расти, то он прекращает свой рост и больше не расширяется. Таким образом, возможна генерация крайне неравномерных по количеству ячеек доменов. Пример результата применения этого алгоритма приведен на рис. 4.1 снизу слева.

Среди распространенных алгоритмов существует множество подходов к декомпозиции расчетных сеток, включая алгоритмы, основанные на методе спектральной бисекции [318], диффузионные и генетические алгоритмы [319], иерархические алгоритмы [320] и многие другие. Наиболее полный обзор алгоритмов декомпозиции расчетных сеток можно найти в [321] и [104].

Для практического использования нас будут интересовать алгоритмы декомпозиции поверхностной неструктурированной сетки, с помощью которых можно добиться достаточно низких значений характеристик $D$ и $L$.

В работе [321] приведено описание параллельного алгоритма геометрической декомпозиции сеточных данных. Во время работы этого алгоритма происходит последовательное деление текущего домена пополам с помощью сечения плоскостью. Этот алгоритм можно расширить, введя в него произвольные критерии разбиения текущего домена на пару более мелких доменов. Если задан массив ячеек домена $C$ и произвольная функция извлечения признака из ячейки $fun$, то можно выполнить разделение домена на две части, сформировав массив признаков всех ячеек домена, тогда половина массива с меньшими значениями признака формирует один дочерний домен, а вторая половина с большими значениями признаков формирует второй дочерний домен (рис. 4.2).

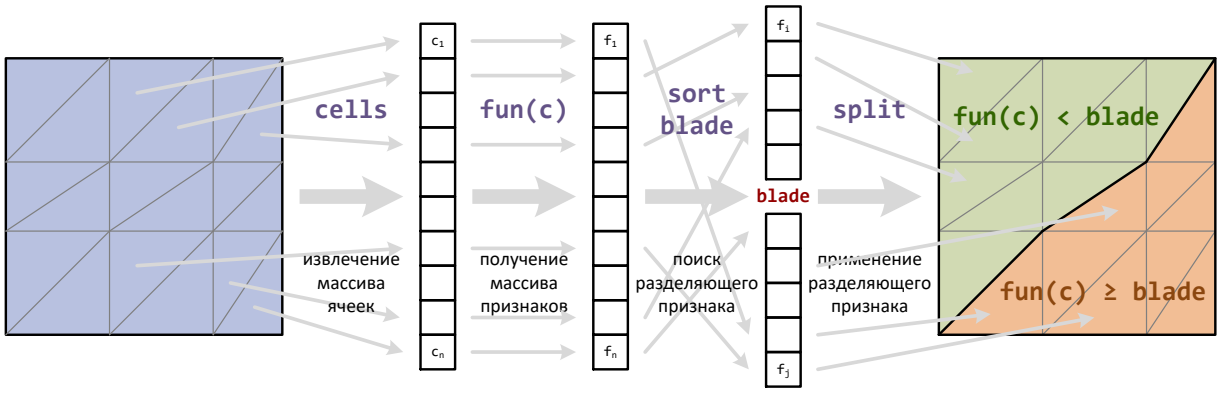


Рис. 4.2. Схема выполнения разделения домена пополам по заданному признаку fun

После разделения домена на два более мелких домена можно вычислить параметр, отражающий эффективность разбиения. В качестве такого парамет-

ра предлагается использовать длину границы между двумя образованными новыми доменами. Таким образом, критерий разбиения зависит от функции вычисления признака $fun$. В свою очередь это означает, что при выполнении разбиения не обязательно ограничиваться одной функцией вычисления признака, а можно подать список функций, для каждой функции вычислитель показатель качества разбиения и в результате остановиться на той функции вычисления признака, которая приводит к наиболее эффективному разбиению. Если в качестве функций вычисления признака ячейки использовать просто извлечение трех координат центров ячеек, то получим алгоритм геометрической бисекции с выбором для дробления наиболее протяженного размера по одной из координат. Результат применения этого алгоритма показан на рис. 4.1 снизу справа.

На рис. 4.3 представлены графики характеристик $D^*$, $L$, $I^*$, вычисленные во время применения различных алгоритмов декомпозиции поверхностной расчетной сетки на разное количество доменов от 2 до 32.

Для эксперимента использовались две сетки. Первая сетка – wing, вторая сетка – ref – более приближена к реальной геометрии летательного аппарата и содержит порядка $n = 4 \cdot 10^5$ ячеек. На графиках введены следующие краткие названия методов декомпозиции: linear – линейное разделение ячеек между доменами по индексу, rgrow – наращивание доменов от случайных ячеек, hierarchical – иерархическое деление доменов пополам по одной из трех координат (геометрическая бисекция).

При использовании метода rgrow использовалась только одна итерации наращивания для демонстрации неравномерности размеров доменов при случайном выборе инициирующих ячеек (из рисунков видно, что значение параметра $D^*$ достигает значений 0,6 и 0,8 для сеток wing и ref соответственно).

Из приведенных графиков можно сделать вывод, что из достаточно простых методов декомпозиции могут быть выбраны методы, при использовании которых генерируется достаточно равномерное распределение ячеек по доменам при низкой общей доле междоменных ребер и малой протяженности границ между доменами.


Рис. 4.3. Графики показателей $D^*$, $L$, $I^*$ для сеток wing и ref при использовании методов декомпозиции linear, rgrow, hierarchical

Анализ графиков с рис. 4.3 позволяет сделать вывод, что алгоритм наращивания доменов является пригодным для практического применения с точки зрения характеристик $L$, $I$ и для повышения его эффективности необходимо добиться уменьшения показателя $D$. Это можно сделать с помощью генетического подхода к улучшению расположения инициирующих вершин [322].

Генетические алгоритмы представляют собой природоподобные эвристические алгоритмы, с помощью которых можно искать решение оптимизационных задач с большим количеством параметров [323]. В процессе применения гене-

тического алгоритма моделируется процесс создания, выживания и размножения популяции потенциально возможных решений поставленной проблемы в условиях сложно устроенной окружающей среды. Каждое решение является отдельной особью, которая создается из набора характеризующих ее генов (генотипа). Для особи определена функция пригодности, либо наоборот штрафная функция. На каждой эпохе генетического алгоритма на основе текущей популяции создается следующая популяция, которая по ожиданиям в среднем должна быть более приспособлена к условиям окружающей среды. Генетические алгоритмы широко применяются во многих областях, для поиска приближенного решения задач с большим количеством параметров и ограничений. В частности генетические алгоритмы применимы в задаче декомпозиция графа [324, 325].

При использовании генетических алгоритмов важно различать понятие генотипа и особи. Генотип – это набор генов, некий код, который кодирует механизм создание особи. Особь – сформированный на основе генотипа индивид, который обладает определенными свойствами и характеристиками, и для которого может быть вычислена функция пригодности или штрафная функция. Зачастую при использовании генетических алгоритмов различие между генотипом и особью стирается, и вместо генотипа используется просто явное представление особи. Так в работе [324] при декомпозиции графа в генотипе кодируется каждое ребро рассматриваемого графа, а в работе [325] генотип представляет собой точное соотнесение каждой вершины графа и результирующего домена. Использование генотипа в роли особи в генетических алгоритмах приводит к существенному замедлению сходимости, что снижает ценность использования этих алгоритмов. К тому же применение механизмов скрещивания и мутаций к особям вместо генотипов также вызывает сомнение с точки зрения корректности таких операций. Рассмотрим подход, в котором в качестве генотипа используется максимально короткий код, по которому может быть быстро построена особь.

Рассмотрим алгоритм декомпозиции графа порядка $n$ на $k$ доменов. В качестве генотипа будем использовать вектор длины $k$, задающий $k$ номеров

вершин (инициирующие, или опорные вершины). При инициализации каждую $i$-ую опорную вершину будем относить к $i$-му домену. При построении особи на основе генотипа будем распределять вершины между доменами с помощью простого алгоритма пузырькового роста доменов, одновременно обходя граф в ширину начиная с опорных вершин. В качестве штрафной функции может использоваться комбинация характеристик декомпозиции $Q = \delta D + \lambda L$.

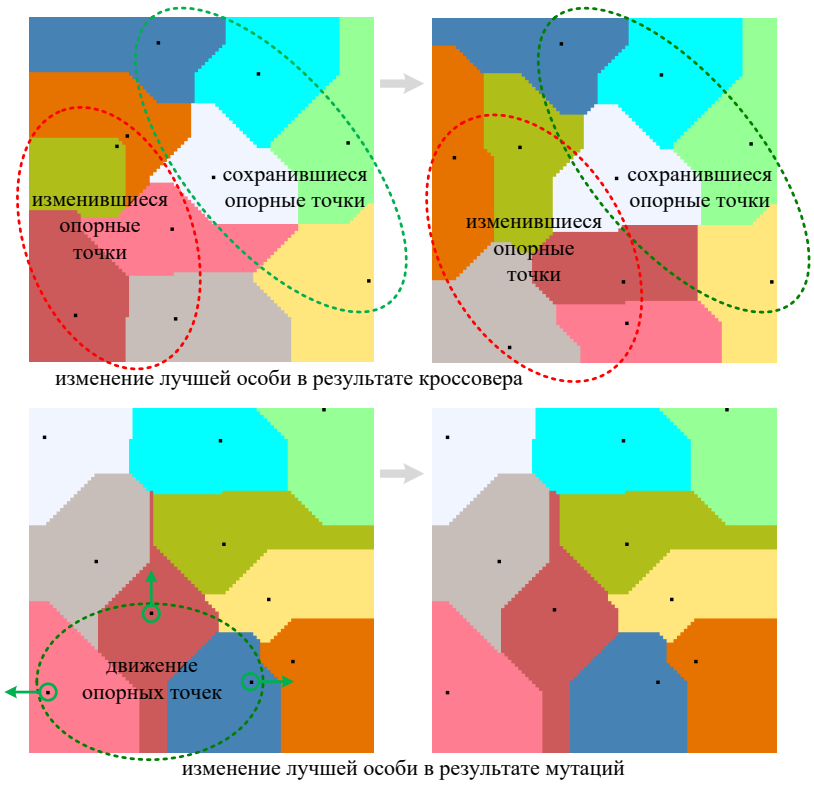


Рис. 4.4. Замещение части генотипа в процессе скрещивания (сверху), и мутация опорных вершин (снизу)

На основе такого способа построения особи будем моделировать процесс эволюции. Пусть мы имеем текущую популяцию, в которой для каждой особи известна характеристика ее качества. Далее наихудшая часть популяции удаляется, а из оставшейся части популяции случайные особи формируют потомков с помощью операций скрещивания (кроссовера) и мутаций. Операция скрещивания устроена предельно просто: для каждого номера $i$ от 1 до $k$ в качестве $i$-го гена случайным образом выбирается $i$-й ген одного из родителей. После выполнения скрещивания в новой особи с заданной вероятностью

применяется мутация. В качестве мутации случайная опорной вершина за меняется на случайного соседа.

На рис. 4.4 приведены характерные варианты изменения лучшей особи при переходе от текущей популяции к популяции следующей эпохи. Сверху на рисунке проиллюстрирована ситуация, когда часть опорных вершин лучшей особи сохранилась, а другая часть была заменена на опорные вершины второго родителя, что в целом привело к снижению значения штрафной функции. Снизу на рисунке проиллюстрировано влияние последовательности мутаций на лучшую особь. Так, даже незаметное на рисунке смещение опорных вершин привело к визуально различимому изменению геометрии образованных этими опорными вершинами доменов.

Генетический алгоритм применим к расчетным сеткам произвольного вида, с его помощью можно быстро получать качественное решение декомпозиции, произвольным образом задавая требуемую штрафную функцию. Алгоритм допускает параллельное исполнение, хорошо масштабируется для работы с расчетными сетками большего размера. Генетический алгоритм требует для своего применения эффективной реализации процедуры обхода графа в ширину от инициирующих вершин.

### 4.1.2 Метод сглаживания границ между доменами поверхностной неструктурированной сетки

При использовании декомпозиции расчетной сетки основным показателем качества декомпозиции является параметр $D$, так как он отражает равномерность распределения ячеек по доменам. Но если игнорировать остальные показатели, то в процессе декомпозиции могут появляться протяженные «пилообразные» границы между доменами, которые приводят к возрастанию показателей $L$ и $I$, что негативно сказывается на производительности. Пример возникновения таких пилообразных границ можно увидеть на рис. 4.1 снизу справа, на рис. 4.5 фрагмент сетки с пилообразными границами между доменами приведен более крупно.

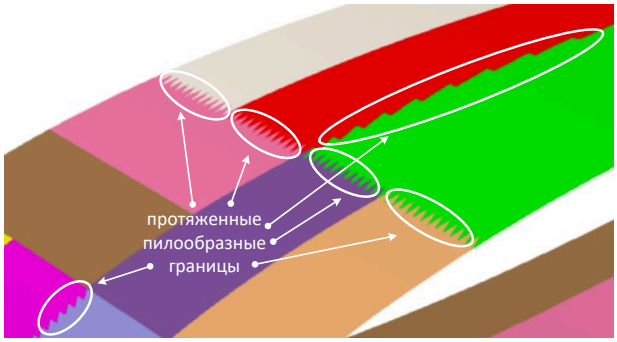


Рис. 4.5. Возникновение протяженных пилообразных границ между доменами

Такие границы необходимо отдельно обрабатывать с целью сокращения их длины. При использовании алгоритмов декомпозиции, основанных на наращивании доменов путем обхода графа в ширину, образуются более сглаженные границы, однако из-за этого деградирует параметр качества декомпозиции $D$. Возникновение протяженных пилообразных границ между доменами особенно характерно для неструктурированных поверхностных сеток, на которых зачастую встречаются ячейки с очень острыми углами. Будем рассматривать алгоритм сглаживания границ между доменами, на который накладывается требование сохранения значения показателя $D$ [326].

Алгоритм сглаживания границ между доменами носит локальный характер, он применяется последовательно к каждой паре доменов и направлен на уменьшение длины границы между ними с сохранением баланса количества ячеек. Граница между двумя доменами может быть представлена в виде набора простых циклов и простых цепей. Простой цикл может быть обработан таким же образом, как и простая цепь, с учетом совпадения первого и последнего узла этой цепи (для такого виртуального размыкания простого цикла может быть выбран произвольный узел этого цикла). При выполнении сглаживания может быть выполнено виртуальное размыкание всех простых циклов, после чего все образовавшиеся цепи рассматриваются последовательно. Без ограничения общности можно считать, что работа ведется с одной простой цепью (полученной с помощью записи отдельных простых цепей подряд), представляющей границу между парой доменов.

Вначале одним проходом по цепи с линейной сложностью выполняется поиск всех пригодных для сглаживания границы шаблонов, представленных на рис. 4.6.

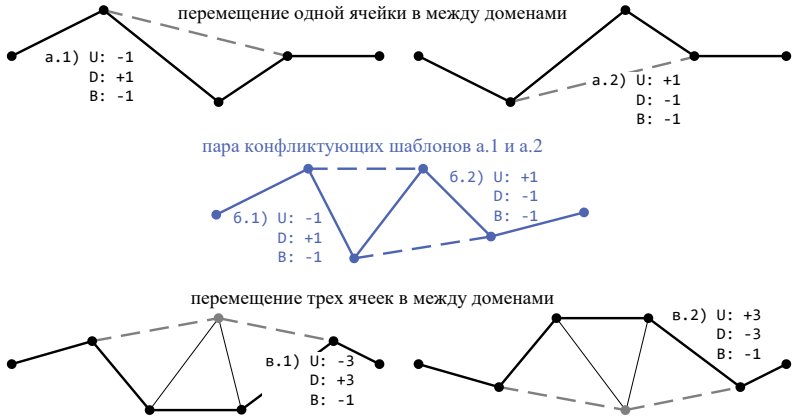


Рис. 4.6. Шаблоны элементарных действий сглаживания границ между доменами

На рис. 4.6 представлены шаблоны, которые мы будем использовать для уменьшения длины границы между двумя доменами. Рассмотрим шаблон а.1. На нем обозначена часть границы между двумя доменами (домен сверху от ломаной обозначен буквой $U$, домен снизу от ломаной обозначен буквой $D$. Черной сплошной линией прочерчена текущая граница между доменами. Пунктирной линией показано возможное улучшение, которое в рассматриваемом случае приведет к следующим последствиям: уменьшение количества ячеек верхнего домена на 1 ($U : -1$), увеличение количества ячеек нижнего домена на 1 ($D : +1$), уменьшение длины границы между двумя доменами на 1 ($B : -1$). Таким образом, шаблон а.1 направлен на втягивание одной ячейки из верхнего домена в нижний домен с сокращением длины границы на единицу. Шаблон а.2 выполняет симметричное действие по втягиванию одной ячейки из нижнего домена в верхний домен, также сокращая при этом длину границы на единицу.

В центре рис. 4.6 синим цветов проиллюстрирован шаблон, который на самом деле представляет собой объединение пары шаблонов а.1 и а.2, конфликтующих друг с другом. Из этой пары шаблонов одновременно может быть

применен только один, что обозначено действиями б.1 и б.2 соответственно.

Шаблоны в.1 и в.2 выполняют действия, аналогичны шаблонам а.1 и а.2, однако вместо одной ячейки происходит втягивание сразу трех соседних ячеек с одновременным уменьшением длины границы между доменами на единицу.

Можно рассматривать и другие, более сложные шаблоны, однако эксперименты показали, что они уже не сильно влияют на результат работы алгоритма.

После того, как внутри цепи найдены все шаблоны, потенциально пригодные для оптимизации границы, выполняется разметка того, как найденные шаблоны могут влиять друг на друга (являются конфликтующими или нет). С учетом того, что любой шаблон может повлиять только на своих непосредственных соседей, такое действие также выполняется с линейной сложностью относительно длины границы.

Центральным местом работы алгоритма является выбор такого множества шаблонов, которые не конфликтуют друг с другом (то есть могут быть применены все одновременно) и не нарушают суммарного баланса ячеек (так как важнейшим показателем эффективности декомпозиции расчетной сетки является равномерность распределения ячеек по доменам). После выбора наибольшего возможного набора шаблонов они применяются, и на этом обработка цепи считается завершенной.

Рассмотрим более подробно алгоритм выбора максимального подмножества неконфликтующих шаблонов, не нарушающих общий баланс ячеек между парой доменов. Для этого рассмотрим пример, представленный на рис. 4.7

На рис. 4.7 представлена граница между двумя доменами (они условно обозначены $U$ – верхний и $D$ – нижний) в виде простой цепи $0 - 16$. Шаблон, который может быть применен для сглаживания границы будем обозначать $t_{a-b}$, где $a$ – номер стартовой вершины, а $b$ – номер конечной вершины ($b > a$). На рис. 4.7 можно выделить следующее множество шаблонов

$$T = \{t_{0-2}, t_{1-4}, t_{3-5}, t_{5-7}, t_{6-8}, t_{7-10}, t_{9-11}, t_{10-12}, t_{11-14}, t_{13-15}, t_{14-16}\}. \quad (4.1)$$

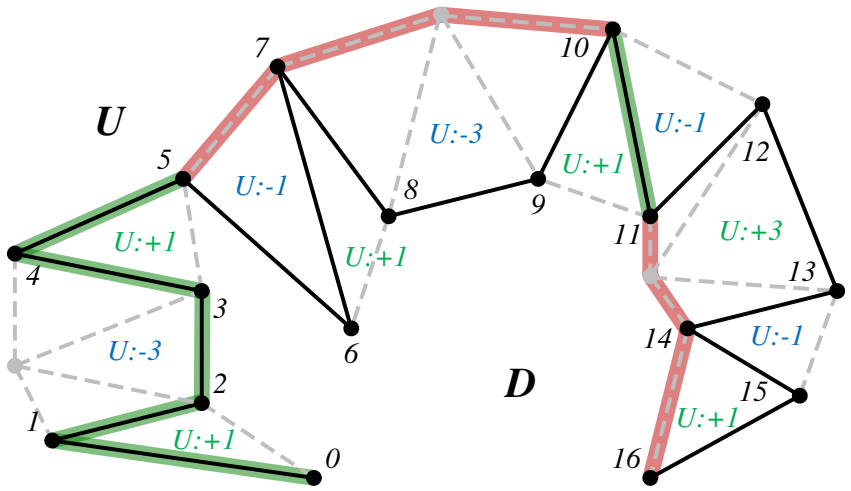


Рис. 4.7. Тест для алгоритм сглаживания границы между доменами

Использование каждого из приведенных шаблонов уменьшает длину границы, но нарушает баланс ячеек между доменами. Будем использовать функцию $U(t)$, которая обозначает изменение количества ячеек в домене $U$ из-за применения шаблона $t$. Так $U(t_{7-10}) = -3$, что означает, что при применении шаблона $t_{7-10}$ домен $U$ потеряет 3 ячейки.

**Определение 4.1.** *Под сравнением двух шаблонов $t_{a-b}$ и $t_{c-d}$ будем понимать сравнение пар, составленных из номеров их стартовых и конечных вершин. То есть $t_{a-b} = t_{c-d} \iff (a, b) = (c, d)$, также для шаблонов определено отношение порядка $t_{a-b} < t_{c-d} \iff (a, b) < (c, d)$.*

**Определение 4.2.** *Два шаблона $t_{a-b} < t_{c-d}$ назовем конфликтующими (или пересекающимися по ребрам), если результатом пересечения сегментов $[a, b]$ и $[c, d]$ является сегмент положительной длины (обозначение $t_{a-b} \cap t_{c-d} \neq \emptyset$).*

На рис. 4.7 каждая пара соседних шаблонов кроме $\{t_{3-5}, t_{5-7}\}$ является конфликтующей. Два конфликтующих шаблона применить одновременно нельзя, так как это либо невозможно технически, либо не приведет к уменьшению длины границы.

Ввиду того, что применение каждого отдельного шаблона уменьшает длину границы на 1, задача поиска оптимального решения может быть сформулирована в виде поиска в множестве шаблонов подмножества максимального

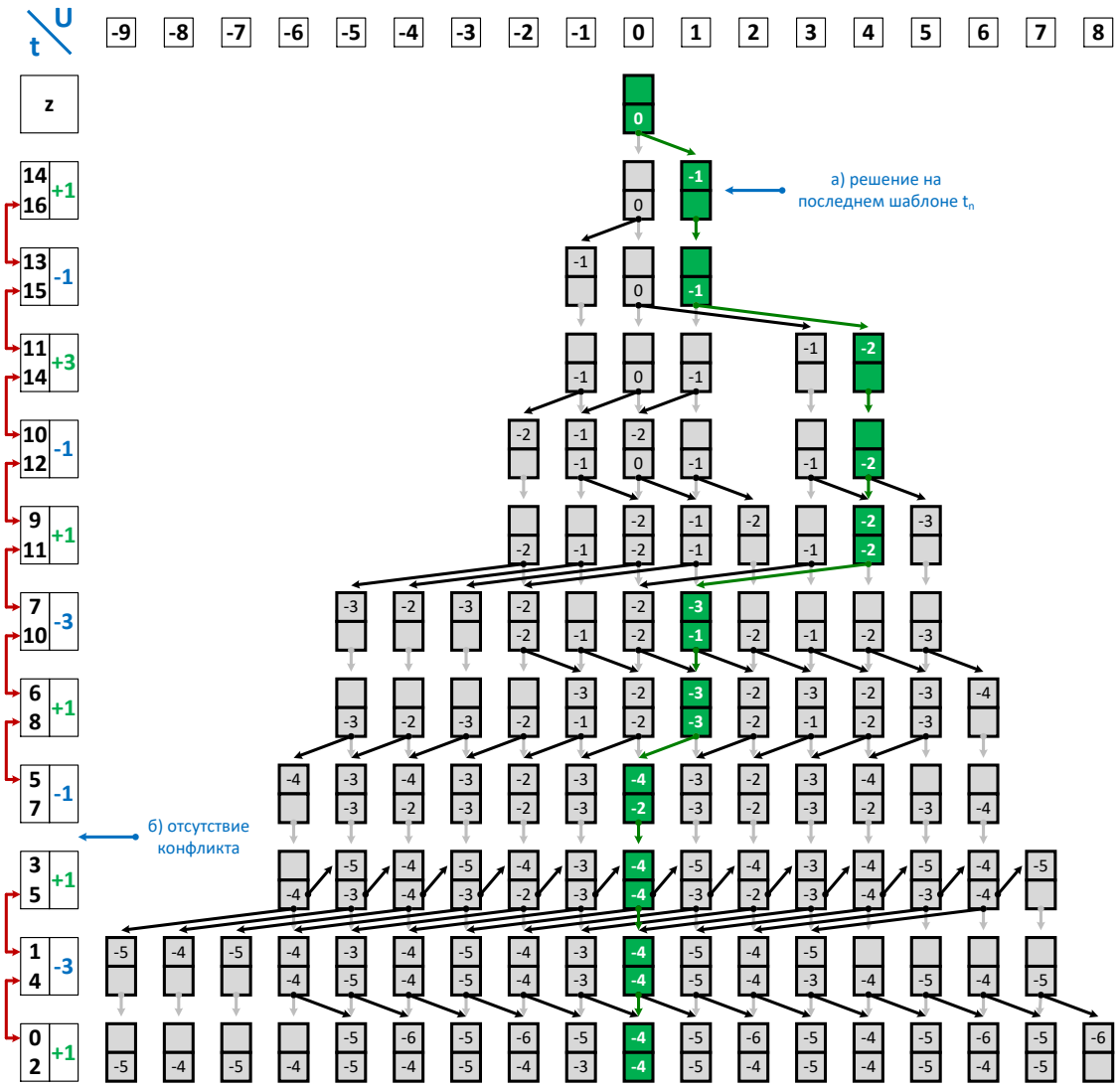


Рис. 4.8. Схема поиска решения задачи сглаживания границы между доменами

размера, не содержащего конфликтующих шаблонов. Поставленную задачу будем решать методом динамического программирования (схема работы алгоритма для примера с рис. 4.7 представлена на рис. 4.8). Для этого рассмотрим функцию $B(t, u, x)$, отражающую максимальное сокращение длины границы при решении задачи на множестве шаблонов $\{t' \in T : t' \geq t\}$ при условии изменения количества ячеек в домене $U$ на $u$ единиц. Параметр $x$ является булевым и принимает значение 1, если шаблон $t$ вошел в решение, и 0 – в противном случае. Решением поставленной задачи является значение $\min(B(t_0, 0, 1), B(t_0, 0, 0))$, где $t_0$ – первый шаблон на рассматриваемой цепи, в нашем случае это $t_{0-2}$.

Решение задачи начинается с последнего шаблона $t_{n-1}$ (в нашем случае это $t_{14-16}$) (рис. 4.8, а). Для него все множество используемых шаблонов состоит только из него самого, и значение функции $B$ определяется для случаев применения или неприменения этого шаблона:

$$\begin{cases} B(t_{n-1}, 0, 0) = 0, \\ B(t_{n-1}, U(t_{n-1}), 1) = -1. \end{cases} \tag{4.2}$$

Для других значений $u$ значение функции $B(t_{n-1}, u, x)$ не определено, примем его за $+\infty$ (это означает, что на последнем шаблоне кроме значений 0 и $U(t_{n-1})$ недостижимы другие значения изменения количества ячеек в домене $U$).

Пусть задача решена для некоторого шаблона $t_{i+1}$. Рассмотрим переход к решению для шаблона $t_i$. Сначала все значения $B(t_i, u, x)$ принимаются равными $+\infty$.

Во-первых, имея любое допустимое решение для шаблона $t_{i+1}$, мы можем получить решение для шаблона $t_i$, просто проигнорировав этот шаблон, то есть для всех $u$, для которых найдено решение для шаблона $t_{i+1}$, выполним следующую операцию (на рис. 4.8 представлена направленными вниз серыми стрелками):

$$B(t_i, u, 0) = \min\left(B(t_{i+1}, u, 0), B(t_{i+1}, u, 1)\right). \tag{4.3}$$

Игнорирование шаблона $t_i$ не меняет баланс ячеек между доменами, из рис. 4.8 это видно, так как все серые стрелки направлены строго вниз.

Рассмотрим возможные варианты использования шаблона $t_i$ в зависимости от наличия конфликта с шаблоном $t_{i+1}$. Если $t_i \cap t_{i+1} = \emptyset$, то есть конфликта между шаблонами нет (см. рис. 4.8, б), то шаблон $t_i$ можно использовать вне зависимости от того, был ли использован шаблон $t_{i+1}$, так как они могут быть использованы одновременно. То есть для всех $u$, для которых было найдено решение для шаблона $t_{i+1}$, выполним следующее действие:

$$B(t_i, u + U(t_i), 1) = B(t_i, u, 0) - 1. \tag{4.4}$$

И наконец, в случае конфликтующих $t_i$ и $t_{i+1}$ мы можем использовать шаблон $t_i$ только в том случае, если не был использован шаблон $t_{i+1}$, то есть для всех $u$ для которых $B(t_{i+1}, u, 0) \neq +\infty$, нужно выполнить операцию

$$B(t_i, u + U(t_i), 1) = B(t_{i+1}, u, 0) - 1. \tag{4.5}$$

Действие для шаблона $t_{n-1}$ согласно (4.2) и действия для шаблонов $t_i$ при изменении $i$ от $n-2$ до 0 согласно (4.3) – (4.5) составляют определение функции $B(t, u, x)$ (прямой ход алгоритма выбора подмножества шаблонов). Функция $B(t, u, x)$ может быть реализована в виде трехмерной матрицы размера $[0, n-1] \times [U_{min}, U_{max}] \times [0, 1]$ (без ограничения общности будем считать, что функция $B$ задана элементами матрицы), все значения которой изначально инициализированы $+\infty$. Значения $U_{min}$ и $U_{max}$ представляют собой теоретически минимально и максимально возможное изменение количества ячеек в домене $U$ и определяются как

$$U_{min} = \frac{1}{2} \sum_{t \in T} (U(t) - |U(t)|),\ U_{max} = \frac{1}{2} \sum_{t \in T} (U(t) + |U(t)|). \tag{4.6}$$

По построенной функции $B(t, u, x)$ выполняется выбор подмножества шаблонов для сглаживания границы между доменами с сохранением баланса ячеек (обратный ход алгоритма). Для этого требуется построить путь в матрице $B$, начиная с ячейки $B(t_i = t_0, j = 0)$, соответствующий максимальному сокращению длины границы (на рис. 4.8 обозначен зеленым цветом), для ячейки $B(t_i = t_0, j = 0)$ максимальное сокращение длины границы равно $c = \min(B(t_0, 0, 0), B(t_0, 0, 1))$.

Если для текущей ячейки $B(t_i, j)$ верно $B(t_i, j, 0) = c$, то шаблон $t_i$ можно не использовать, и следует перейти к ячейке $B(t_{i+1}, j)$ с тем же значением $c$. Если же $B(t_i, j, 0) \neq c$, то $B(t_i, j, 1) = c$, то есть шаблон следует использовать, и нужно перейти к ячейке $B(t_{i+1}, j - U(t_i))$, скорректировав максимальное сокращение длины границы для новой ячейки $c = c + 1$.

**Алгоритм 4.6.** *Алгоритм выбора подмножества шаблонов для сглаживания границы между доменами (`BorderSmoothing`)*

**Вход:** $[t_0, t_1, \ldots, t_{n-1}]$ – упорядоченный по возрастанию список шаблонов.
**Выход:** $r$ – оптимальное изменение длины границы между доменами, $T_{out}$ – список шаблонов для достижения $r$.

$B \leftarrow (+\infty)_{[0,n-1]\times[U_{min},U_{max}]\times[0,1]}$
$(B(t_{n-1},0,0), B(t_{n-1},U(t_{n-1}),1)) \leftarrow (0,-1)$
// прямой ход - построение значений $B$
**for** $i \in [n-2,\ldots,0]$ **do**
  **for** $j \in [U_{min}, U_{max}]$ **do**
    $B(t_i,j,0) \leftarrow \min(B(t_{i+1},j,0), B(t_{i+1},j,1))$
  **end**
  **if** $t_i \cap t_{i+1} = \emptyset$ **then**
    **for** $j : B(t_i,j,0) \neq +\infty$ **do**
      $B(t_i, j+U(t_i), 1) \leftarrow B(t_i,j,0) - 1$
    **end**
  **end**
  **else**
    **for** $j : B(t_{i+1},j,0) \neq +\infty$ **do**
      $B(t_i, j+U(t_i), 1) \leftarrow B(t_{i+1},j,0) - 1$
    **end**
  **end**
**end**
$r \leftarrow \min(B(t_0,0,0), B(t_0,0,1))$
$(T_{out}, j, c) \leftarrow ([], 0, r)$
// обратный ход - выбор подмножества шаблонов по матрице $B$
**for** $i \in [0, n-1]$ **do**
  **if** $B(t_i,j,0) \neq c$ **then**
    $(T_{out}, j, c) \leftarrow (T_{out} \cup \{t_i\}, j - U(t_i), c+1)$
  **end**
**end**
**return** $r, T_{out}$

Алгоритм 4.6 вычисляет максимальное сокращение длины границы между доменами с сохранением баланса ячеек и возвращает подмножество шаблонов, при использовании которых достигается это сокращение. Алгоритм может быть легко скорректирован для выбора подмножества шаблонов для произвольного возможного изменения баланса количества ячеек между доменами,

так как построенная в процессе работе алгоритма матрица $B$ содержит всю необходимую для этого информацию. На рис. 4.8 зеленым цветом выделен вариант для достижения наилучшего сокращения пути при сохранении баланса ячеек между доменами. В нашем случае одним (но не единственным) из оптимальных решений является использование шаблонов $t_{5-7}, t_{7-10}, t_{11-14}, t_{14-16}$, что сокращает границу между доменами на 4, что отражено на рис. 4.7.

В результате работы алгоритма рассчитываются оптимальные решения для всех значений $u$ из сегмента $[U_{min}, U_{max}]$. Тогда длина диапазона равна $U_{max} - U_{min} + 1 = \sum_{t \in T} |U(t)| + 1$, а сложность алгоритма равна $O\left(|T| \cdot \sum_{t \in T} |U(t)|\right)$. Учитывая то, что рассматриваемые шаблоны являются короткими ($|U(t)| \leq 3$), то можно считать что сложность алгоритма равна $O(|T|^2) = O(n^2)$.

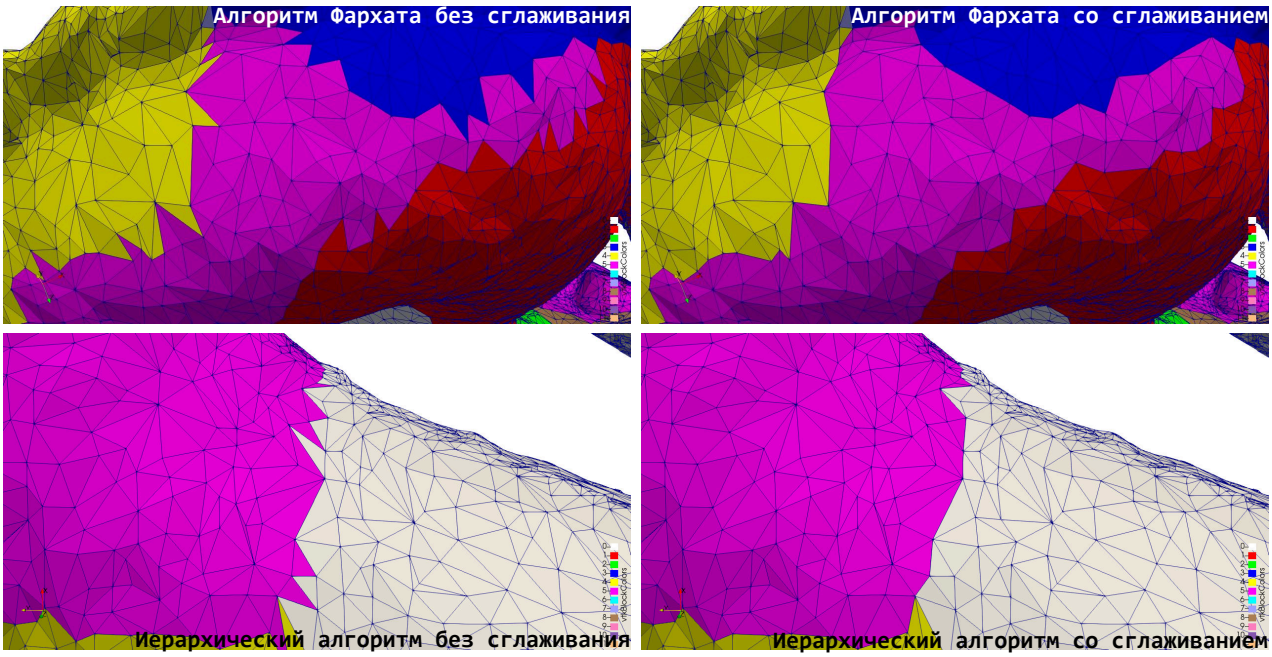


Рис. 4.9. Результат применения сглаживания границ между доменами после алгоритма Фархата (сверху) и иерархического деления доменов (снизу)

На рис. 4.9 представлена визуализация декомпозиции тестовой расчетной сетки с помощью алгоритма Фархата и иерархического деления доменов. В обоих случаях декомпозиция выполнятся на 32 домена. На рисунке крупным планом продемонстрированы отдельные части сетки. На этом рисунке виден эффект от применения алгоритма сглаживания границ меду доменами, прежде всего от заключается в устранении одиноких ячеек, которые вторгаются в

соседний домен одной своей вершиной. После применения алгоритма границы между доменами визуально выглядят более гладко, их длина уменьшается.

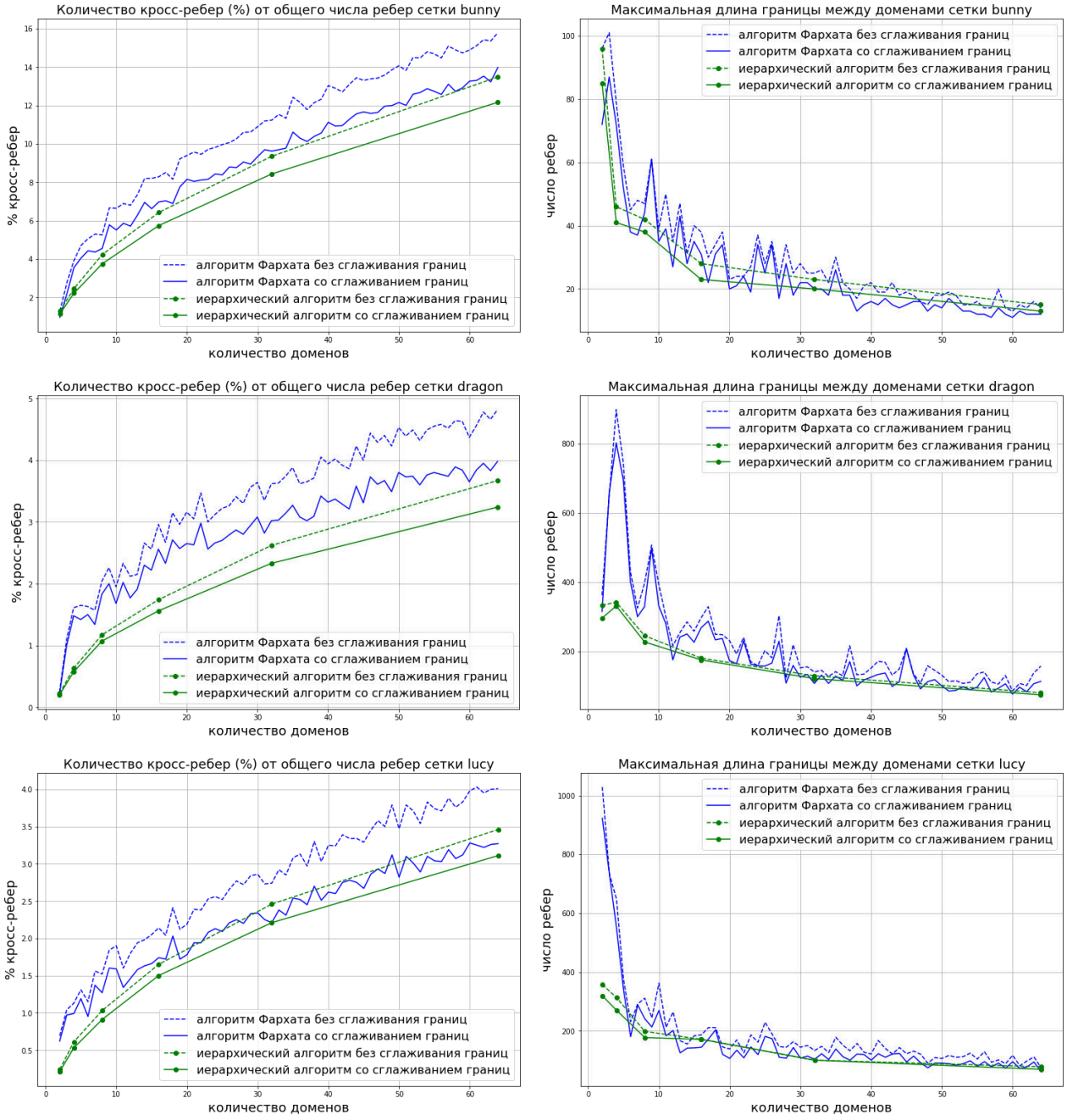


Рис. 4.10. Графики доли кросс-ребер от общего числа ребер сетки (показатель $I^* \cdot 100\%$) и максимальной длины границы между доменами (показатель $L$) при декомпозиции тестовых сеток bunny, dragon и lucy на количество доменов от 2 до 64

Для тестирования эффективности алгоритма сглаживания границ между доменами использовались тестовые неструктурированные поверхностные сетки bunny (количество ячеек $5 \cdot 10^3$), dragon (количество ячеек $10^5$), lucy

(количество ячеек $10^5$) [276], к которым были применены алгоритмы декомпозиции с показателем $D = 0$ и сравнены показатели качества декомпозиции до и после сглаживания границ между доменами. Результаты проведения экспериментов представлены на рис. 4.10.

Из приведенных на рис. 4.10 графиков видно, что в целом проиллюстрированные показатели эффективности декомпозиции расчетных сеток (процентная доля ребер сетки, являющихся граничными ребрами между доменами, и максимальная длина границы между доменами) лучше для алгоритма иерархического деления доменов с выбором наилучшего направления в пространстве ($X$, $Y$, $Z$). Применение алгоритма сглаживания границ приводит к сокращению как общего количества граничных ребер (кросс-ребер), так и длины максимальной границы для тестовых сеток (снижение составляет 10-12%). Этот эффект приводит к снижению времени межпроцессных обменов для задач, выполняющих расчеты на поверхностных расчетных сетках.

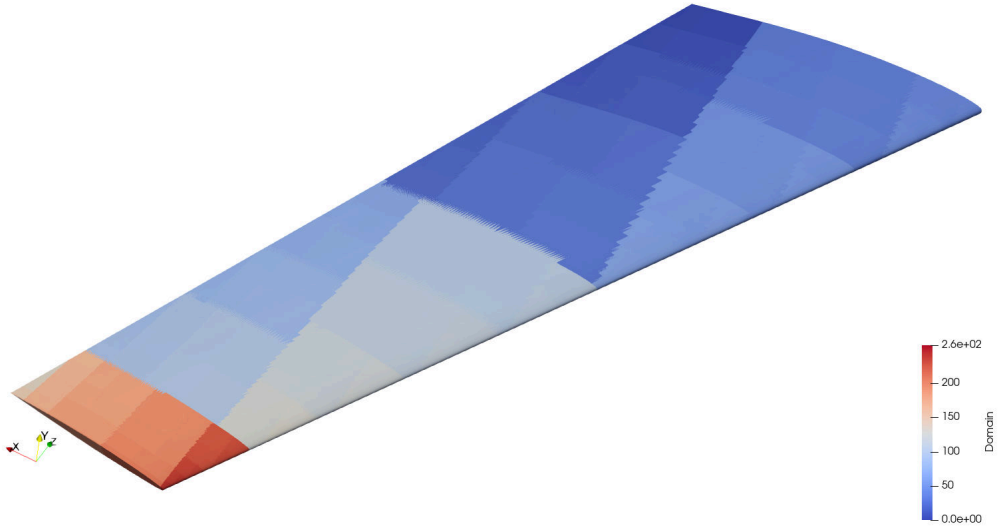


Рис. 4.11. Тестовая сетка GLC-305, декомпозированная с помощью иерархического деления доменов пополам с помощью геометрической бисекции

Метод сглаживания границ между доменами наиболее эффективен при работе с протяженными пилообразными границами (рис. 4.5). На таких границах возможно сокращение длины границ вплоть до 50%. Для анализа влияния сглаживания границ на масштабирование вычислений (ускорение вычислений и эффективность распараллеливания) был поставлен эксперимент по

моделированию обледенения поверхности крыла летательного аппарата. В эксперименте была использована тестовая расчетная сетка профиля GLC-305, содержащая $4 \cdot 10^5$ ячеек. В качестве метода декомпозиции было использовано иерархическое деление доменов пополам с помощью геометрической бисекции, для которой характерно наиболее низкое значение коэффициента неравномерности размера доменов $D$, а также появление протяженных пилообразных границ между доменами (рис. 4.11). Вычисления проводились в условиях малого (3 параметра на ребро границы) и большого (300 параметров на ребро границы) объема пересылаемых данных во время межпроцессных обменов при масштабировании вычислений до 512 MPI-процессов.

Таблица 4.1. Характеристики вычислительных узлов МВС-10П ОП2 на базе микропроцессоров Intel Xeon Gold 6248R Cascade Lake

| Процессор | Intel Xeon Gold 6248R Cascade Lake |
|---|---|
| Частота процессора | 3,0 ГГц |
| Количество ядер процессора | 24 |
| Количество виртуальных процессоров | 48 |
| Пиковая производительность процессора | 2,304 Тфлопс |
| Кэш 1-го уровня | 64 КБ на ядро |
| Кэш 2-го уровня | 24 МБ |
| Кэш 3-го уровня | 37,75 МБ |
| Количество процессоров в узле | 2 |
| Количество ядер в узле | 48 |
| Количество виртуальных процессоров в узле | 96 |
| Пиковая производительность в узле | 4,608 Тфлопс |
| Объем оперативной памяти в узле | 192 ГБ |
| Коммуникационная и транспортная сеть | Onmi-Path |
| Сеть мониторинга и управления | Gigabit Ethernet |
| Сеть управления заданиями | Gigabit Ethernet |
| Системы управления прохождением заданий | SLURM/СУППЗ |

Для эксперимента были использованы вычислительные узлы сегмента суперкомпьютера МВС-10П ОП2 на базе базе двух микропроцессоров Intel Xeon Gold 6248R Cascade Lake, содержащих по 24 ядра каждый (при запуске одного MPI-процесса на ядро). Вычислительные узлы соединены между

собой высокоскоростной коммуникационной сетью Omni-Path с пропускной способностью 100 Гбит/с. Характеристики вычислительных узлов системы МВС-10П ОП2 приведены в таблице 4.1)

Сглаживание границ для рассматриваемой расчетной сетки приводит к сокращению длин границ между доменами в диапазоне 32-38%. Результаты эксперимента демонстрируют более выраженное влияние сокращения длин границ между доменами на масштабирование вычислений в условиях большого размера сообщений при межпроцессных обменах (рис. 4.12).

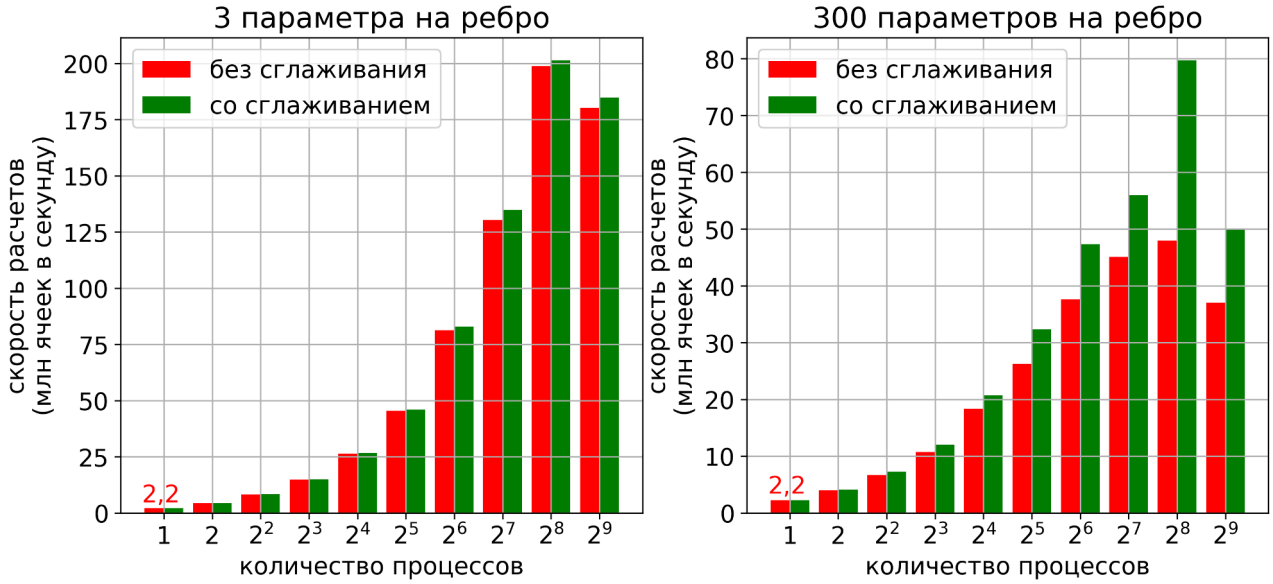


Рис. 4.12. Диаграммы масштабирования вычислений без сглаживания и со сглаживанием границ при малом (слева) и большом (справа) размере сообщений

Метод сглаживания границ между доменами декомпозированной поверхностной неструктурированной сетки, позволяющий из множества локальных шаблонов сглаживания границы между парой доменов выбрать подмножество шаблонов, при одновременном применении которых обеспечивается максимальное сокращение длины границы при заданном изменении баланса ячеек между этими доменами, реализован в программе «crys-gsu» подготовки неструктурированной поверхностной расчетной сетки для проведения расчетов на суперкомпьютере [128].

### 4.1.3 Масштабирование вычислений

При проведении расчетов на декомпозированной сетке на каждой итерации счета необходимо выполнять обмен данными на каждой границе между доменами. В нашем случае при выполнении декомпозиции расчетной сетки граница между двумя доменами представлена произвольным набором междоменных ребер. При этом граница может быть разрывной, она и вовсе может состоять из отдельных ребер, поэтому последовательность междоменных ребер при описании границы не имеет значения.

На рис. 4.13 представлена схема организации межпроцессных обменов. На этой иллюстрации два домена (будем условно их называть левым и правым), обрабатывающиеся в MPI-процессах с номерами 0 и 1, разделены границей, состоящей из трех ребер. При этом в левом домене присутствует три ячейки, которые примыкают к рассматриваемой границе, в правом домене таких ячеек только две (так как ячейка 23R примыкает сразу к двум ребрам границы). Для организации межпроцессных обменов в каждом домене для каждого ребра рассматриваемой границы создаются фиктивные ячейки, участвующие при расчете потоков через ребра границы. Пересчет физических величин в фиктивных ячейках выполнять не нужно, данные для фиктивных ячеек получаются с помощью MPI-обменов из ячеек соседнего домена.

Для пересылки данных из настоящих граничных ячеек домена в фиктивные ячейки соседнего домена в каждом из двух соседних доменов организуются буферы отправки и приема данных. Последовательность обмена данными выглядит следующим образом (схема показана на рис. 4.13). Сначала данные из настоящих граничных ячеек записываются в соответствующие буферы отправки данных (send buff), далее для всех границ расчетной сетки выполняются асинхронные команды `MPI_Irecv` приема сообщений в буферах получения данных (recv buff). После чего также одновременно для всех границ расчетной сетки выполняются команды асинхронной отправки данных `MPI_Isend` из буферов отправки. Далее выполняется ожидание завершения всех асинхронных обменов данными с помощью функции `MPI_Waitall`. Последним шагом,

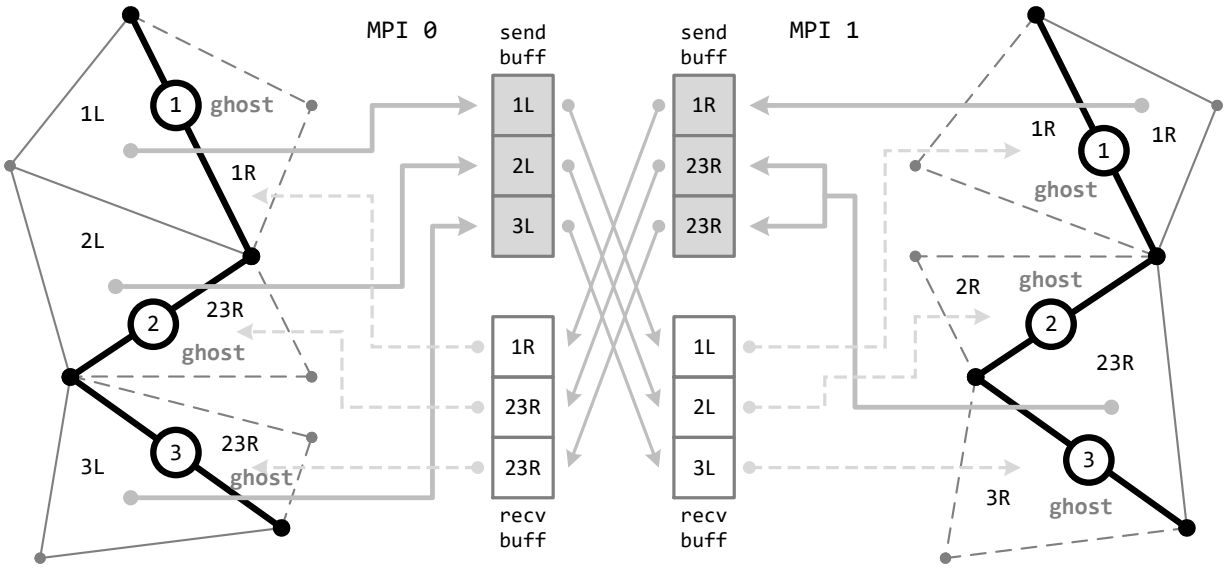


Рис. 4.13. Схема выполнения MPI-обменов через границу двух доменов

завершающим обмен данными между соседними доменами, является перенос полученных физических значений из буферов получения данных (recv buff) в соответствующие фиктивные ячейки.

При использовании фиктивных ячеек возможно некоторое дублирование данных. На представленной схеме ячейке 23R из правого домена соответствуют сразу две фиктивные ячейки в левом домене. Эти ячейки содержат одинаковые данные. Такое дублирование информации является приемлемым, так как данные фиктивных ячеек используются только на чтение для выполнения вычисления потоков через границу доменов, поэтому выполнять какую-либо синхронизацию одинаковых фиктивных ячеек не нужно.

Для замеров показателей масштабируемости вычислений на неструктурированной поверхностной расчетной сетке была использована тестовая поверхность GLC-305 обтекаемого трехмерного тела, содержащая порядка $2 \cdot 10^5$ узлов и $4 \cdot 10^5$ ячеек. На этой сетке производились вычисления по моделированию обледенения поверхности с помощью программного комплекса «Кристалл» [126, 127]. В ячейках выполнялись расчеты, связанные с моделированием течения жидкой пленки, решением уравнений теплового баланса на поверхности, а также перестроение и сглаживание поверхности. Для декомпо-

зиции поверхностной сетки использовался простой иерархический алгоритм деления доменов пополам из раздела 4.1.1, в котором в качестве признаков ячеек брались три координаты центра, а критерием выбора конкретной координаты для деления домена являлась минимизация длины границы между двумя доменами. К результату декомпозиции был применен алгоритм сглаживания границ между доменами из раздела 4.1.2.

Для измерения показателей масштабируемости вычислений использовались гомогенные сегменты вычислительной системы МВС-10П: МВС-10П МП2 (таблица 3.2) и МВС-10П ОП2 (таблица 4.1) [327], содержащие до 8 узлов. Во время запусков внутри каждого вычислительного узла использовалось распараллеливание на общей памяти на разное количество доступных потоков: внутри узлов МВС-10П МП2 использовались варианты с 1, 72, 144, 216 и 288 потоками, внутри узлов МВС-10П ОП2 использовались варианты с 1, 24, 48, 72 и 96 потоками.


Рис. 4.14. Производительность вычислений по моделированию обледенения на вычислительных узлах МВС-10П МП2

Основной целью запусков было проведение замеров показателей сильной масштабируемости вычислений при частичном и полном распараллеливании внутри вычислительных узлов с использованием OpenMP. То есть для всех


Рис. 4.15. Производительность вычислений по моделированию обледенения на вычислительных узлах МВС-10П ОП2

запусков использовалась одна и та же поверхностная расчетная сетка (которая дробилась на нужное количество вычислительных процессов).

При проведении расчетов замеры выполнялись независимо для каждой вычислительной системы. На рис. 4.14 и рис. 4.17 приведены диаграммы производительности вычислений при увеличении количества вычислительных узлов для разных вычислительных систем. Скорость вычислений исчисляется в количестве обрабатываемых ячеек расчетной сетки в секунду.

По результатам масштабирования вычислений для численного решения задачи обледенения были составлены рекомендации по эффективному использованию вычислительных ресурсов. Для вычислительных узлов МВС-10П МП2 рекомендован запуск на 8 вычислительных узлах с использованием 144-216 потоков в узле. Для вычислительных узлов МВС-10П ОП2 рекомендован запуск на 8 вычислительных узлах с использованием 48-72 потоков в узле. Использование двухуровневого распараллеливания MPI+OpenMP, а также разработанного метода сглаживания границ между доменами позволило сократить время расчетов более чем на два порядка при моделировании обледенения на вычислительных узлах МВС-10П ОП2.

На сегменте МВС-10П ОП2 был поставлен эксперимент по масштабированию вычислений по моделированию обледенения фюзеляжа вертолета (рис. 4.16), описываемого сетками, содержащими $3{,}5 \cdot 10^5$ и $1{,}4 \cdot 10^6$ ячеек, на 1-16 узлах при использовании одного потока на ядро.

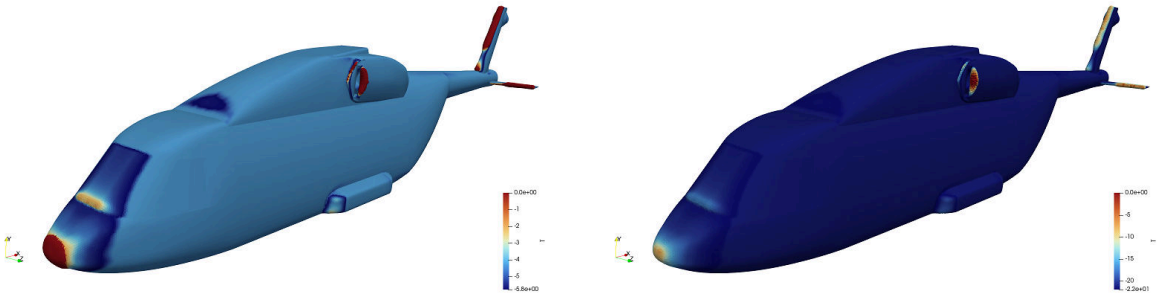

Рис. 4.16. Картина обледенения фюзеляжа вертолета при температурах $T = -8°C$ (слева) и $T = -25°C$ (справа)

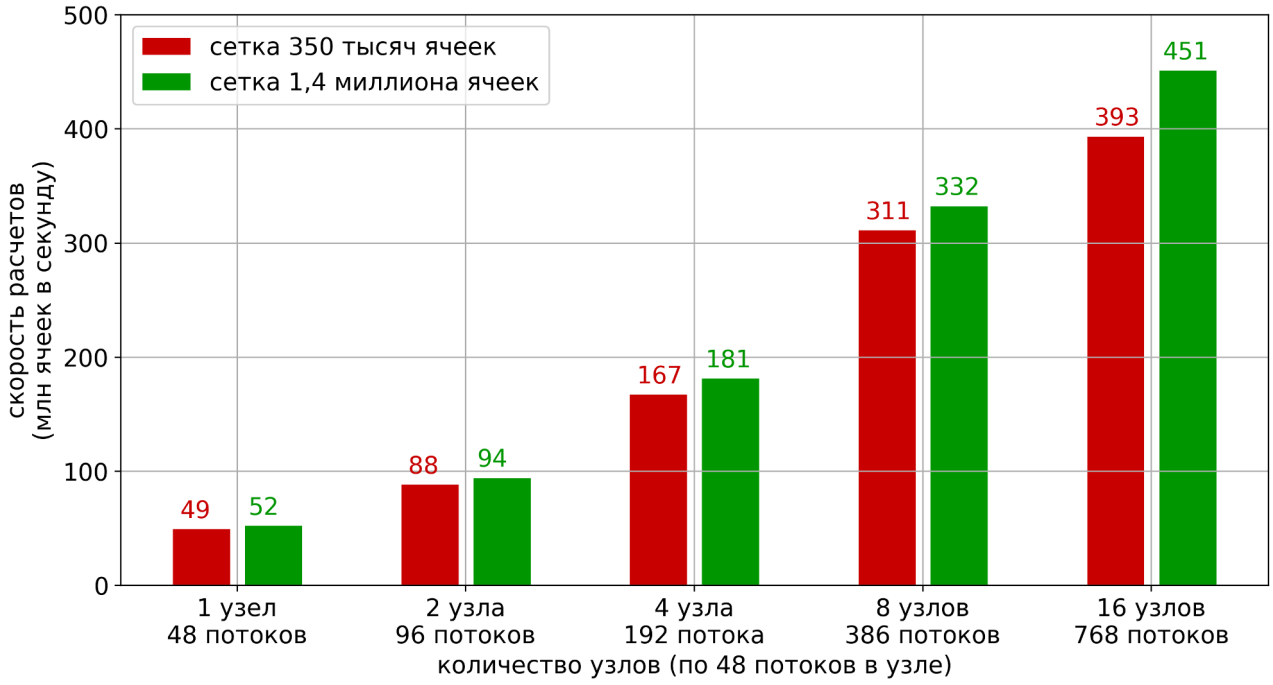


Рис. 4.17. Производительность вычислений по моделированию обледенения фюзеляжа вертолета на вычислительных узлах МВС-10П ОП2

Отмечена масштабируемость вычислений при использовании вплоть до 16 вычислительных узлов при общем ускорении вычислений более 200 раз относительно нераспараллеленной версии, а также более высокая производительность вычислений на сетке с большим числом ячеек.

## 4.2 Распараллеливание на общей памяти

В предыдущих разделах рассмотрены вопросы распараллеливания вычислений между узлами суперкомпьютера и запущенными на них процессами. Современные микропроцессоры, входящие в состав вычислительных узлов, содержат десятки вычислительных ядер [328–330], что с учетом многопоточности приводит необходимости организации вычислений в рамках одного узла с использованием сотен потоков, работающих на общей памяти.

Так же как и в случае распределения вычислений между узлами суперкомпьютера, распараллеливание на общей памяти преследует цель сокращения времени выполнения.

Основной проблемой снижения эффективности распараллеливания вычислений на общей памяти являются конфликты при доступе разных потоков к одной и той же области данных. В разделе рассмотрены вопросы организации вычислений на поверхностной неструктурированной расчетной сетке на общей памяти с целью устранения конфликтов по данным между разными потоками.

Во время выполнения расчетов на поверхностной неструктурированной сетке при обработке ячейки необходимо обращаться за данными к ее вычислительной окрестности, которая в простейшем случае представляет собой все соседние по ребрам ячейки. Аналогично, все ячейки из окрестности обращаются к рассматриваемой ячейке, и при одновременном обращении к каким-либо данным на запись возникает конфликт между потоками, обрабатывающими разные ячейки окрестности. Добиться устранения таких конфликтов можно путем запрета параллельной обработки ячеек, принадлежащих к вычислительно окрестности одной и той же ячейки. Эта задача напрямую связана с задачей реберной раскраски дуального графа расчетной сетки, в результате которой множество ребер разбиваются на подмножества без конфликтов.

### 4.2.1 Распараллеливание для задачи обледенения

Рассмотрим проблему распараллеливания вычислений на общей памяти применительно к задаче расчета обледенения поверхности с помощью конечно-

объемного метода [331]. На каждой итерации расчетов в ячейках производится решение системы уравнений массового и теплового балансов, из которой получаются основные данные состояния ячейки. Между расчетными итерациями выполняется расчет протекания массы и тепла между соседними ячейками согласно логике конечно-объемного численного метода.

Листинг 4.1. Расчет протекания массы и тепла через ребро сетки

```
void Edge::calc_flows()
{
    // cells pair
    Cell* fst = ..., sec = ...;

    // flows calculation
    double flow_w = ..., flow_q = ...;

    // flows correction
    fst->flow_out_w += flow_w;
    fst->flow_out_q += flow_q;
    sec->flow_out_w -= flow_w;
    sec->flow_out_q -= flow_q;
}
```

Пересчет состояния ячейки выполняется независимо от других ячеек, вычисления могут выполняться параллельно. Пересчет массы и тепла выполняются через каждое ребро сетки, перетекание выполняется из одной инцидентной ребру ячейки в другую. Общая схема пересчета перетекания массы и тепла для одного ребра может выглядеть так, как представлено на листинге 4.1.

Для выполнения пересчета перетекания для всех ребер домена расчетной сетки необходимо обработать все ребра в цикле. При обработке ребер домена расчетной сетки в цикле возникает желание распараллелить этот цикл. При распараллеливании вычислений с помощью OpenMP следует учитывать конфликты по данным, которые могут возникнуть при выполнении корректировки данных в ячейках, инцидентных обрабатываемому ребру (в случае если одновременно начнется изменение значения в одной области памяти). Для устранения этих конфликтов достаточно выполнять операции корректировки данных в атомарном режиме, как показано на листинге 4.2.

Листинг 4.2. Атомарные операции корректировки данных

```
#pragma omp atomic
fst->flow_out_w += flow_w;
#pragma omp atomic
fst->flow_out_q += flow_q;
#pragma omp atomic
sec->flow_out_w -= flow_w;
#pragma omp atomic
sec->flow_out_q -= flow_q;
```

Использование `#pragma omp atomic` гарантирует, что указанная команда одновременно будет обрабатываться только одним вычислительным потоком (то есть, что между чтением старого значения переменной и записью нового значения не попадут операции другого потока). При большом количестве используемых потоков это может приводить к потерям производительности.

### 4.2.2 Разделение ребер поверхностной неструктурированной сетки на неконфликтующие подмножества, основанное на реберной раскраске дуального графа

Другой подход к разрешению конфликтов по доступу к данным при распараллеливании вычислений на общей памяти связан с задачей реберной раскраски дуального графа сетки [332].

Рассмотрим ситуацию, при которой возможно возникновение конфликта при пересчете потоков во время параллельной обработки двух ребер сетки. Конфликт возможен, если оба обрабатываемых ребра являются инцидентными одной и той же ячейке. Без ограничения общности будем считать, что рассматриваемая расчетная сетка замкнута, то есть каждая ячейка имеет ровно трех соседей, и ее дуальный граф является кубическим. Задача разбиения ребер расчетной сетки на неконфликтующие множества сводится к построению реберной раскраски дуального графа. Будем рассматривать только односвязные поверхности, то есть анализируемый дуальный граф является планарным кубическим графом. Так как ни одна ячейка сетки не может быть соседней себе самой, а значит в построенном дуальном графе нет мостов. То есть рассматривается планарный кубический граф без мостов.

Так как рассматриваемый граф является кубическим, то он очевидно допускает реберную раскраску в 5 цветов, и эта раскраска может быть построена с линейной сложностью по количеству ребер просто за один обход всех ребер графа. При такой раскраске планарность графа не учитывается.

Согласно теореме Визинга [333, 334] кубический граф допускает реберную раскраску в 4 цвета. Однако, теорема Визинга также не использует планарность графа, и алгоритм, построенный исходя из доказательства, будет иметь сложность, выше линейной по количеству ребер [335]. Приведем альтернативное доказательство с использованием планарности графа и более низкой сложностью алгоритма построения реберной раскраски в 4 цвета.

**Лемма 4.1.** *Кубический планарный граф допускает реберную раскраску в* 4 *цвета, и сложность ее построения линейна по количеству ребер.*

Доказательство существования раскраски проведем по индукции по количеству вершин графа. База индукции: кубический граф с минимальным количеством вершин это $K_4$, для него утверждение очевидно. Пусть утверждение верно для всех кубических планарных графов, порядок которых изменяется от 4 до $n-2$ (порядок кубического графа четный).

Рассмотрим кубический планарный граф, с множеством вершин $V$ ($n = |V| = \nu$), множеством ребер $E$ ($|E| = \varepsilon$) и множеством граней $F$ ($|F| = \zeta$). Внешнюю область вокруг графа также считаем гранью (рассматриваем граф, расположенный на сфере). Через $\zeta_i$ обозначим количество граней, содержащих ровно $i \geq 3$ вершин (грань имеет размер $i$), тогда $\zeta = \sum_{i=3}^{\infty} \zeta_i$.

Рассмотрим соотношения, выполняемые для этого графа. Так как степень каждой вершины равна 3, то лемма о рукопожатиях для этого графа приобретает вид $3\nu = 2\varepsilon$. Так как каждое ребро входит ровно в две грани, то верно соотношение $2\varepsilon = \sum_{i=3}^{\infty} i\zeta_i$. Наконец, для рассматриваемого графа выполняется соотношение Эйлера $\nu + \zeta - \varepsilon = 2$. Подставив в соотношение Эйлера выражения для $\nu$, $\zeta$ и $\varepsilon$ получим соотношение $\sum_{i=3}^{\infty} (6-i)\zeta_i = 12$, откуда следует, что в рассматриваемом графе найдется грань размера 3, 4 или 5. Рассмотрим каждый этих случаев отдельно.

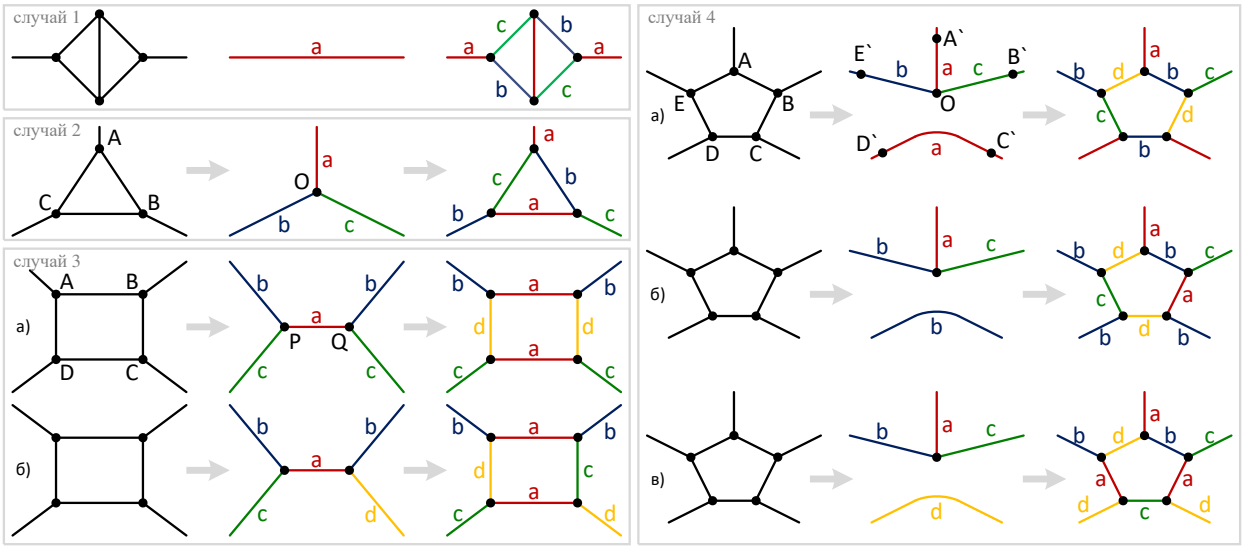


Рис. 4.18. Разные случаи перекраски ребер графа

Случай 1. Сначала рассмотрим случай наличия в графе гамака, состоящего из двух треугольных граней (рис. 4.18, случай 1). Стянув гамак в одно ребро, получим граф с $n-4$ вершинами, ребра которого могут быть раскрашены в 4 цвета по предположению индукции. Тогда по этой раскраске не представляется сложным раскрасить исходный граф.

Случай 2. В графе нет гамаков, состоящих из двух треугольных граней, но найдется треугольная грань (рис. 4.18, случай 2). Стянем треугольную грань $ABC$ в точку $O$. Так как в графе не было гамаков, состоящих из двух треугольных граней, то у грани $ABC$ не было соседних треугольных граней по ребру, а значит при стягивании $ABC \to O$ ни одна грань кроме $ABC$ не выродилась. Это значит, что мы получили плоский кубический граф с $n-2$ вершинами, ребра которого могут быть раскрашены в 4 цвета по предположению индукции. Тогда по этой раскраске можно построить раскраску исходного графа, как показано на рис. 4.18, случай 2.

Случай 3. В графе нет треугольных граней, но найдется четырехугольная грань (рис. 4.18, случай 3). Рассмотрим четырехугольную грань $ABCD$. Выполним стягивание ребер $AD \to P$, $BC \to Q$. Так как в графе не было треугольных граней, то при стягивании выродилась только грань $ABCD$. Мы получили плоский кубический граф с $n-2$ вершинами, ребра которого

могут быть раскрашены в 4 цвета по предположению индукции. По этой раскраске построим раскраску исходного графа. Для этого перенесем цвета всех ребер (кроме $PQ$) на исходный граф, непокрашенными останутся только ребра цикла $A-B-C-D$. Пусть $\gamma(PQ)=a$. Если для покраски остальных 4 ребер, инцидентных вершинам $P$ и $Q$, были использованы только 2 цвета ($b$ и $c$), то исходный граф можно покрасить в 4 цвета, покрасив ребра цикла $A-B-C-D$ в цвета $a$ и $d$ (рис. 4.18, случай 3, а). Если для покраски остальных 4 ребер, инцидентных вершинам $P$ и $Q$, были использованы все оставшиеся три цвета ($b$, $c$ и $d$), то одним из этих цветов было покрашено 2 ребра (пусть это будет цвет $b$), а двумя другими цветами – по одному ребру (рис. 4.18, случай 3, б). В этом случае раскрасим ребра цикла $A-B-C-D$ в цвета $a$, $c$, $d$ следующим образом. К циклу $A-B-C-D$ примыкает одно ребро цвета $c$, оно смежно двум ребрам этого цикла (которые являются смежными между собой), которые не могут быть покрашены с цвет $c$. То есть в цикле $A-B-C-D$ найдется пара смежных ребер, одно из которых может быть покрашено в цвет $c$ (обозначим эту пару $p_c$). Аналогично, в цикле $A-B-C-D$ найдется пара смежных ребер, одно из которых может быть покрашено в цвет $d$ (обозначим эту пару $p_d$). Вне зависимости от того пересекаются ли $p_c$ и $p_d$, в цикле $A-B-C-D$ можно найти такие несмежные ребра $e_c$ и $e_d$, что $e_c \in p_c$, $e_d \in p_d$. Тогда покрасим ребро $e_c$ в цвет $c$, ребро $e_d$ – в цвет $d$, а остальные два несмежных ребра – в цвет $a$ (рис. 4.18, случай 3, б).

Случай 4. В графе нет треугольных и четырехугольных граней, но найдется пятиугольная грань (рис. 4.18, случай 4). Рассмотрим пятиугольную грань $ABCDE$. Удалим ребра $ED$ и $BC$. Стянем вершины $E$, $A$, $B$ в одну вершину $O$. Стянем вершины $D$, $C$ в одну вершину степени 2, которую сразу удалим, соединим инцидентные ей ребра в одно $D'C'$. Так как в графе отсутствовали треугольные грани, то ни одна грань кроме $ABCDE$ не выродилась. Таким образом, мы получили граф с $n-4$ вершинами, ребра которого могут быть раскрашены в 4 цвета. По этой раскраске построим раскраску исходного графа. Для этого перенесем цвета всех ребер на исходный граф, непокрашенными останутся только ребра цикла $A-B-C-D-E$. Если для покраски ребер

$OA'$, $OE'$, $OB'$, $D'C'$ были использованы только три цвета ($a$, $b$, $c$), то без ограничения общности можно рассмотреть только два варианта раскраски: $\gamma(D'C') = \gamma(OA')$ (рис. 4.18, случай 4, а) и $\gamma(D'C') \neq \gamma(OA')$ (рис. 4.18, случай 4, б). Случай же, когда для покраски ребер $OA'$, $OE'$, $OB'$, $D'C'$ были использованы все 4 цвета, представлен на рис. 4.18, случай 4, в. Во всех трех случаях ребра цикла $A-B-C-D-E$ могут быть раскрашены с сохранением правильной реберной раскраски в 4 цвета.

Рассмотрев все возможные случаи, удалось раскрасить текущий граф с $n$ вершинами в 4 цвета, опираясь на реберные раскраски графов меньших порядков. Таким образом, предположение индукции верно, и плоский кубический граф может быть раскрашен в 4 цвета.

Доказательство существования раскраски конструктивно, алгоритм раскраски может быть построен путем стягивания гамаков и граней графа вплоть до графов минимального размера, с последующим восстановлением и построением раскраски. Каждое действие по стягиванию и построению раскраски текущего графа на основе раскраски графа меньшего порядка, осуществляется за количество действий $O(1)$. Каждое стягивание уменьшает количество граней графа хотя бы на 1, таким образом общая сложность алгоритма $O(\zeta)$, что для плоского кубического графа равносильно $O(\nu)$ или $O(\varepsilon)$. ■

Построение реберной раскраски рассматриваемого графа в 4 цвета, хоть и имеет также линейную сложность по количеству ребер, является значительно более сложной, чем построение раскраски в 5 цветов.

Отметим кратко еще один факт, касающийся построения реберной раскраски рассматриваемого дуального графа. Согласно [335, 336], утверждение о возможности раскраски ребер плоского кубического графа без мостов в три цвета равносильно задаче о четырех красках. Если кубический граф допускает правильную реберную раскраску в 3 цвета, то такая раскраска называется раскраской Тейта. В цикле работ [337–339] авторы предлагают способ построения раскраски Тейта для плоских кубических графов без мостов, а в [331] предлагается вариант реализации с квадратичной сложностью по количеству

ребер. Такая сложность не позволяет использовать раскраску в 3 цвета для эффективного практического применения.

Для оценки эффективности реберной раскраски для устранения конфликтов по данным при расчете перетекания массы и тепла в задаче моделирования процесса обледенения поверхности были выполнены запуски модельной задачи на одном вычислительном узле МВС-10П МП2 (таблица 3.2).

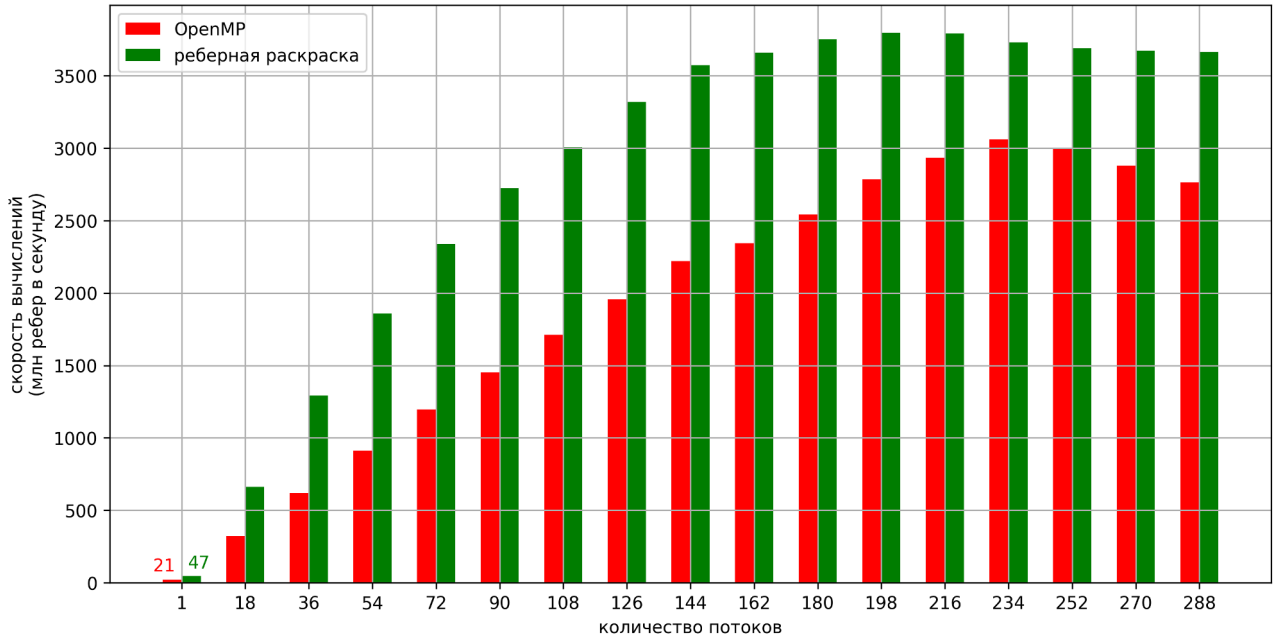


Рис. 4.19. Производительность вычислений при распараллеливании пересчета перетекания массы и тепла для разных способов устранения конфликтов по данным

Для сравнения эффективности методов устранения конфликтов при параллельном расчете перетекания массы и тепла через ребра ячеек замерялось количество обрабатываемых ребер в секунду при разном количестве используемых вычислительных потоков. Сравнивались два метода устранения конфликтов по доступу к данным: использование `omp atomic` операций и разделение множества ребер на неконфликтующие подмножества с помощью реберной раскраски.

Из рис. 4.19 можно видель, что метод реберной раскраски более эффективен, чем использование директивы `omp atomic`. Эксперименты показали, что количество цветов реберной раскраски не влияет на производительность, поэтому целесообразно использовать раскраску в 5 цветов.

Механизм разделения ребер поверхностной неструктурированной сетки на неконфликтующие подмножества, основанный на алгоритмах реберной раскраски, реализован в программном комплексе «Кристалл» компьютерного моделирования процесса обледенения элементов авиационных силовых установок [126, 127] и используется при моделировании обледенения поверхности.

## 4.3 Выводы

Разработан метод сглаживания границ между доменами декомпозированной поверхностной неструктурированной сетки, позволяющий из множества локальных шаблонов сглаживания границы между парой доменов выбрать подмножество шаблонов, при одновременном применении которых обеспечивается максимальное сокращение длины границы при заданном изменении баланса ячеек между этими доменами. В основе метода лежит алгоритм сглаживания границы при помощи динамического программирования, обладающий квадратичной сложностью по количеству шаблонов. Метод сглаживания границ между доменами поверхностной неструктурированной сетки позволяет уменьшить длину границ и повысить тем самым производительность вычислений за счет сокращения объема межпроцессных обменов.

Реализован механизм разделения ребер поверхностной сетки на неконфликтующие подмножества, основанный на алгоритмах реберной раскраски дуального графа расчетной сетки. Механизм позволяет повысить производительность вычислений на общей памяти за счет разрешения конфликтов по доступу к данным.

Предложенные решения по повышению производительности параллельных вычислений на поверхностной сетке при распараллеливании на распределенной и общей памяти реализованы в программном комплексе «Кристалл», что позволило добиться ускорения расчетов по моделированию обледенения более чем на два порядка по сравнению с нераспараллеленной версией.

## Глава 5. Векторизация вычислений

В главе рассматриваются оптимизации программного кода, направленные на повышение его производительности с помощью векторизации вычислений, а также предложена методика векторизации, применимая к широкому спектру расчетных приложений. Для оценки качества векторизации программного кода в этой главе будем использовать следующие понятия.

**Определение 5.1.** *Шириной векторизации назовем величину* $w = \frac{v}{t}$*, где* $v$ *– размер векторного регистра,* $t$ *– размер типа расчетных данных.*

Например, при расчетах на данных FP32 ширина векторизации при использовании AVX-512 (размер векторного регистра 512 битов) равна 16.

**Определение 5.2.** *Ускорением от векторизации будем называть величину* $s = \frac{T}{T_v}$*, где* $T$ *– время выполнения скалярной версии кода,* $T_v$ *– время выполнения векторной версии кода.*

**Определение 5.3.** *Эффективностью векторизации будем называть величину* $e = \frac{s}{w}$*.*

Эффективность векторизации является удобным показателем для оценки применения оптимизаций. Значение $e$ может превышать единицу при использовании, например, большого количества векторных операций FMA. Однако, в большинстве случаев этот показатель меньше единицы.

**Определение 5.4.** *Логическим ускорением от векторизации будем называть величину* $s^* = \frac{L}{L_v}$*, где* $L$ *– количество выполненных инструкций в скалярном коде,* $L_v$ *– количество выполненных аналогичных им инструкций в векторном коде (будем считать, что в идеальном случае* $w$ *одинаковых скалярных инструкций объединяются в одну аналогичную векторную инструкцию, и* $s^* = w$*). То есть* $s^*$ *имеет смысл ускорения в терминах количества инструкций.*

**Определение 5.5.** *Логической эффективностью векторизации будем называть величину* $e^* = \frac{s^*}{w}$*.*

В главе рассматриваются оптимизации по векторизации с использованием набора инструкций AVX-512 как наиболее перспективного и обладающего особенностями, позволяющими строить высокопроизводительный код.

## 5.1 Особенности векторных инструкций

Множество инструкций AVX-512 состоит из подмножеств, их перечень варьируется для разных поколений микропроцессоров [174]. AVX-512 F (Foundation) – основной набор, который содержит базовые операции для работы с векторными данными, включая арифметику, операции конвертации, сравнения, операции перестановки элементов векторов и другие. Набор AVX-512 VL (Vector Length) позволяет расширить многие векторные инструкции для работы со 128-битными регистрами XMM и 256-битными регистрами YMM на длину вектора 512. Набор AVX-512 BW (Byte and Word) расширяет инструкции для работы с элементами векторов размера 8 и 16 бит, а AVX-512 DQ (Doubleword and Quadword) добавляет новые инструкции, для 32 и 64-битных элементов. Набор AVX-512 CD (Conflict Detection) содержит операцию для нахождения пар совпадающих целых значений в двух векторах (VPCONFLICTD/Q), операции записи маски в элементы вектора и подсчета количества ведущих нулей в элементах вектора. Операции подсчета количества единиц в элементах векторов определены в наборах AVX-512 BITALG (Bit Algorithms) и AVX-512 VPOPCNTDQ (Vector population count). AVX-512 ER (Exponential and Reciprocal) включает в себя инструкции для вычисления значений $2^x$, $1/x$, $1/x^2$. AVX-512 PF (PreFetch) содержит операции предварительной подкачки данных для VGATHERDPS/VSCATTERDPD, что позволяет уменьшить вероятность промаха в кэш при дальнейших обращениях в память. Наборы AVX-512 VBMI (Vector Byte Manipulation Instructions) и AVX-512 VBMI2 добавляют новые операции, работающие с 8-битными данными, такие как перестановка элементов, чтение и запись, конкатенация со сдвигом. AVX-512 IFMA (Integer Fused Multiply Add) содержит комбинированные операции над целыми числами, которые используются для реализации

работы с большими числами или для шифрования [197]. Набор AVX-512 FP16 включает арифметические операции и операции конвертации для работы с 16-битными вещественными числами, а набор AVX-512 BF16 добавляет инструкции для работы с форматом brain float 16 [344]. AVX-512 VNNI (Vector Neural Network Instructions) и AVX-512 4VNNIW содержат операции поэлементного перемножения пар целых чисел с последующим их сложением. Операции выполняются над 8-битными и 16-битными данными и используются в задачах линейной алгебры, в частности для реализации искусственных нейронных сетей [345]. Набор AVX-512 4FMAPS (Fused Multiply Accumulation Packed Single) содержит инструкции V4FMADDPS и V4FNMADDPS, реализующие комбинированные операции над четырьмя 512-битными операндами и значениями из памяти, объединяя в себе 64 операции перемножения и 64 операции сложения 16-битных вещественных чисел. AVX-512 VPCLMULQDQ содержит операцию VPCLMULQDQ, предназначенную для перемножения 128-битных целых чисел. AVX-512 VP2INTERSECT (Vector pair intersection) содержит операции VP2INTERSECTD и VP2INTERSECTQ, используемые для определения пересечений двух векторов 32-битных или 64-битных целых чисел [346]. AVX-512 VAES (Vector Advanced Encryption Standard) содержит векторные операции для поддержки алгоритма блочного шифрования [347]. AVX-512 GFNI (Galois Field New Instructions) включает в себя операции для работы в конечном поле Галуа $GF(2^8)$ [348].

Для поддержки выборочного применения операций над упакованными данными к конкретным элементам большинство инструкций AVX-512 использует регистры-маски (k0 – k7) в качестве аргументов. Маски используются в командах для осуществления условной операции над элементами или для слияния векторов. Маски могут использоваться для выборочного чтения из памяти и запись в память элементов векторов, для аккумулирования результатов логических операций над элементами. Эта уникальная особенность набора инструкций AVX-512 обеспечивает реализацию предикатного режима исполнения [349], который поддержан в таких архитектурах, как ARM или «Эльбрус» [350]. Наличие предикатного режима исполнения позволяет

применять оптимизацию слияния ветвей исполнения и, таким образом, избавляться от лишних операций передачи управления, что помогает создавать высокоэффективный параллельный код.

Из других важных особенностей набора AVX-512 можно отметить операции множественного чтения элементов векторов с произвольными смещениями от базового адреса, аналогичные операции записи в память (операции gather/scatter). Хотя эти операции крайне медленные, они в некоторых случаях помогают существенно упростить логику векторизованного кода. Следует отметить большое разнообразие операций перестановки, перемешивания, дублирования, пересылки элементов векторов, что позволяет произвольным образом менять порядок обработки данных. Также существенное ускорение способны принести комбинированные операции, объединяющие операцию умножения и сложения в одну операцию.

Для упрощения применения векторных инструкций при оптимизации программного кода, написанного на языке C/C++, разработаны функции-интринсики [183]. Эти функции покрывают не все множество инструкций AVX-512, но избавляют от необходимости писать ассемблерный код. Вместо этого предоставляется возможность оперировать встроенными типами данных для 512-битных векторов и использовать их при работе с функциями-инстринсиками как обычные базовые типы (при построении компилятором исполняемого кода для этих типов данных будут использованы регистры ZMM). Некоторые функции-инстринсики соответствуют не одной отдельной команде, а целой последовательности, как, например, группа функций reduce, другие же просто раскрываются в вызов библиотечной функции (например, тригонометрические функции).

В главе рассмотрены методы векторизации программного кода с помощью оптимизирующего компилятора и с помощью явного использования функций-интринсиков. В некоторых листингах по тексту главы некоторые имена интринсиков сокращены, где это не мешает пониманию: например, при рассмотрении операций с вещественными числами FP32 вместо имени `_mm512_add_ps` используется просто обозначение `ADD`.

## 5.2 Выделение однотипных операций для векторизации

Основной смысл векторной команды состоит в поэлементной обработке векторных данных с помощью одной и той же операции. Для векторизации необходимо выделить в коде однотипные операции, которые далее могут быть объединены в векторные команды. От способа выделения однотипных операций существенным образом зависит производительность кода. В качестве примера программного контекста для анализа способа выделения однотипных операций рассмотрим перемножение матриц малой размерности [351].


Рис. 5.1. Схема вычисления результата в операции `matmat8`

Рассмотрим перемножение двух матриц $8 \times 8$ (рис. 5.1). Результатом перемножения является матрица, элементами которой будут попарные скалярные произведения строк первой матрицы на столбцы второй. Использование 512-битных команд загрузки данных из памяти позволяет за одну операцию загрузить две соседние строки первой матрицы или два соседних столбца второй матрицы (с помощью операции gather). Далее нужно вычислить скалярные произведения каждой половины загруженного из первой матрицы вектора с каждой половиной загруженного из второй матрицы вектора. Для этого выполняются две операции поэлементного перемножения двух загруженных векторов, в одном из которых старшая и младшая половины второго

вектора переставлены местами. Далее возникает потребность вычисления суммы 8 младших элементов вектора и 8 старших элементов этого же вектора. Каждый элемент результирующей матрицы должен быть получен с помощью горизонтальной операции суммирования 8 младших или старших элементов некоторого регистра ZMM.

Перемножение матриц $16 \times 16$ выполняется аналогично, там возникает задача суммирования всех 16 элементов. Возникает потребность объединения горизонтальных операций. Рассмотрим задачу в общем виде: даны 16 регистров ZMM ($a$ – $p$), каждый из которых содержит по 16 элементов FP32. Требуется посчитать суммы их элементов и записать в один регистр ZMM. Набор команд AVX-512 содержит возможности для реализации этого функционала. Схема вычислений состоит из четырех фаз, они реализуется операциями перестановок с последующим сложением и слиянием векторов.

На рис. 5.2 представлена схема суммирования элементов векторов. Будем реализовывать сложение двух элементов одного и того же вектора с помощью покомпонентного сложения этого вектора со своей копией, в которой переставлены элементы. Это действие выполняется на каждой обозначенной на рис. 5.2 фазе. Так как вектор содержит 16 элементов, то количество фаз для суммирования его элементов равно 4. После первой фазы (после выполнения операции слияния blend) можно видеть вектор, содержащий суммы пар соседних элементов векторов, после второй фазы вектор состоит уже из сумм четверок элементов, после третьей фазы вектор содержит суммы 8 элементов, после четвертой фазы вектор содержит суммы всех элементов.

На рис 5.3 представлен поток данных суммирования всех элементов 16 векторов. На нем черные квадраты – операции перестановки элементов векторов, черные круги – операции сложения вектора со своей пермутированной копией, белые круги – операции слияния двух векторов по маске. Прямоугольниками с пунктирной границей очерчены фазы из рис. 5.2.

Граф потока данных на рис. 5.3 состоит из схожих блоков операций: 2 swizzle/permute + 2 add + 1 blend. Причем на каждой фазе свои маски перестановок и слияний, но они постоянны для всех входных векторов. Для

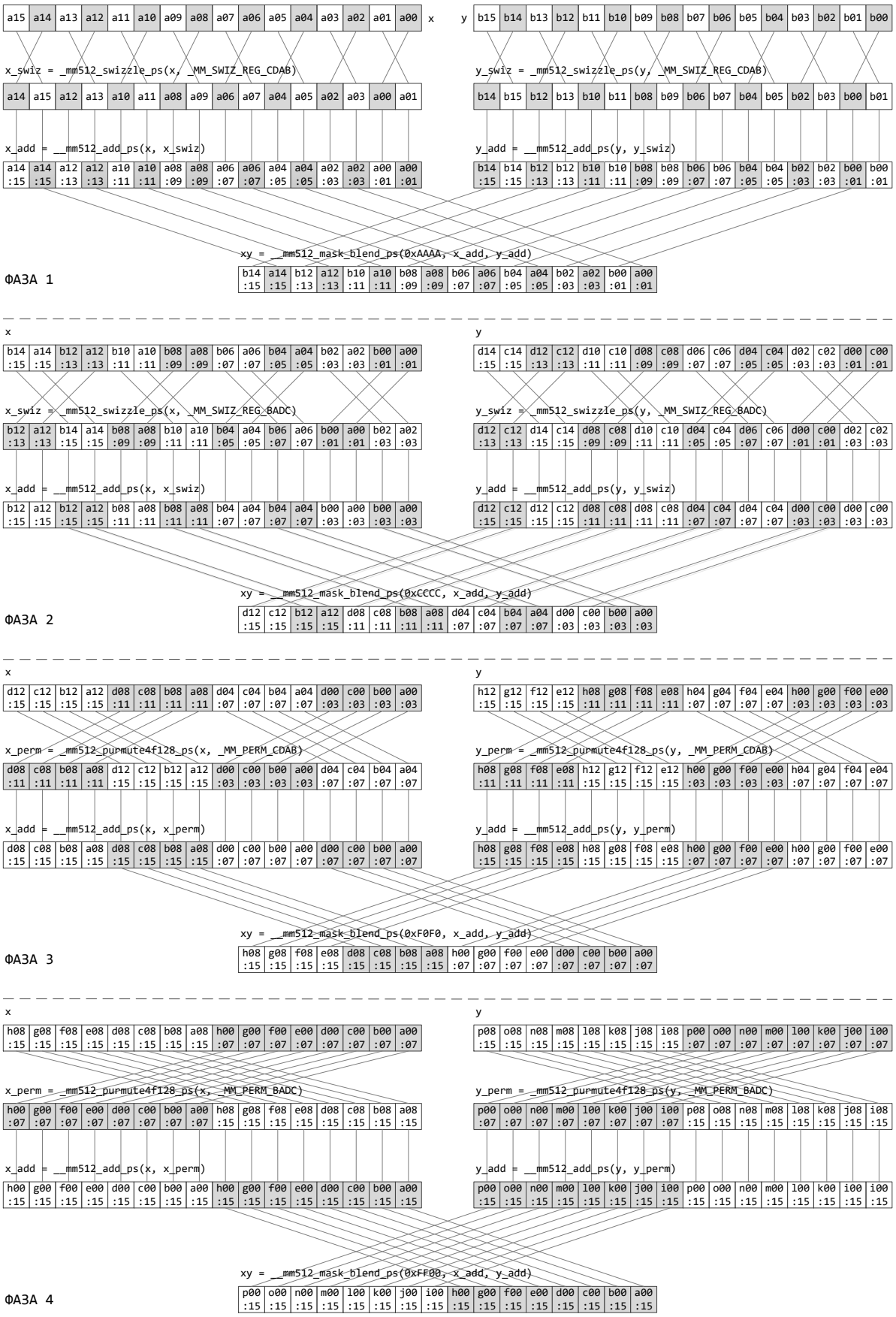


Рис. 5.2. Схема суммирования элементов вектора

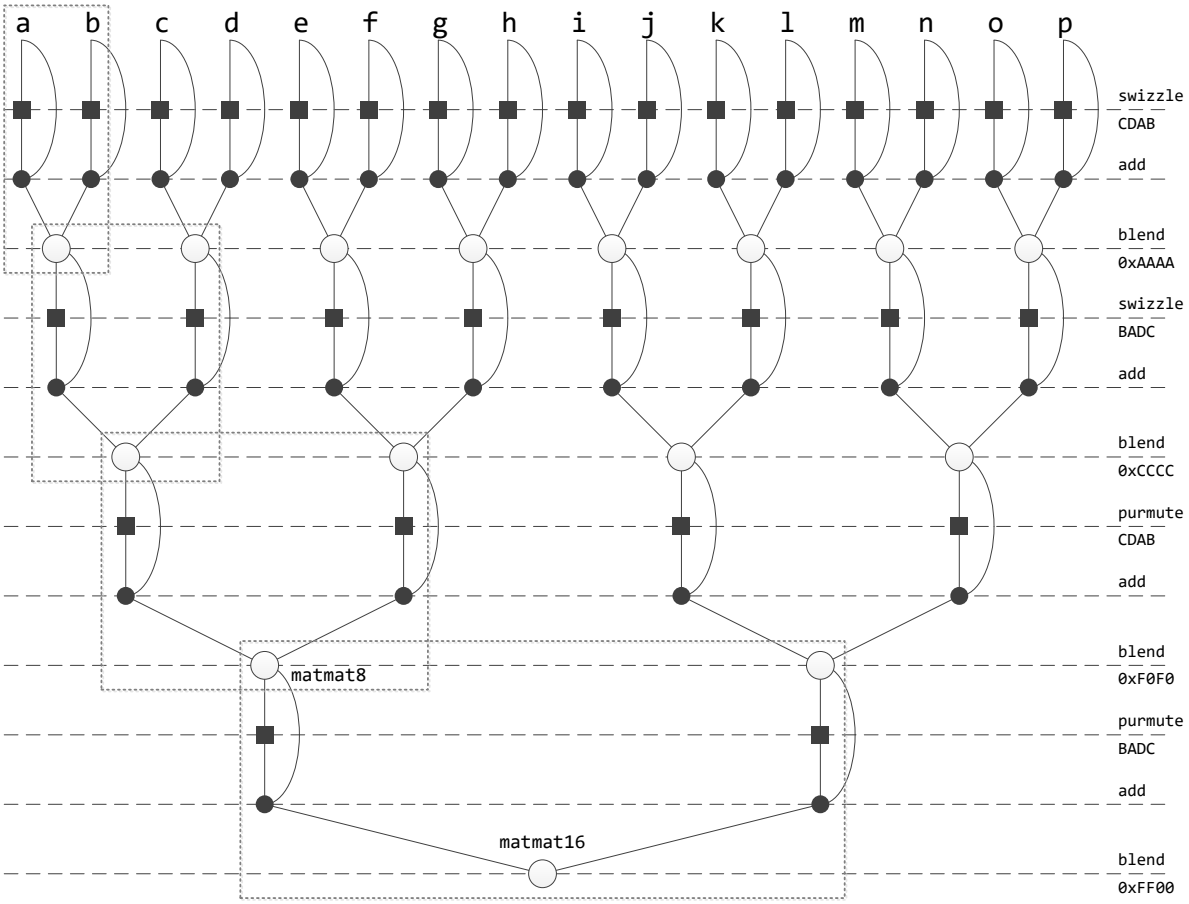


Рис. 5.3. Схема суммирования элементов 16 векторов

удобства можно определить макросы, реализующие обозначенные фазы. На вход макрос получает пару обрабатываемых векторов, а также параметры перестановки элементов и слияния результирующей пары. Интринсик swizzle предназначен для перестановки элементов внутри 128-битных четвертей, а permute4f128 переставляет местами эти четверти (листинг 5.1).

Листинг 5.1. Определение макросов для реализации фаз суммирования элементов векторов

```
#define SWIZ_2_ADD_2_BLEND_1(X, Y, SWIZ_TYPE, BLEND_MASK)\
    _mm512_mask_blend_ps(BLEND_MASK,\
        _mm512_add_ps(X, _mm512_swizzle_ps(X, SWIZ_TYPE)),\
        _mm512_add_ps(Y, _mm512_swizzle_ps(Y, SWIZ_TYPE)))

#define PERM_2_ADD_2_BLEND_1(X, Y, PERM_TYPE, BLEND_MASK)\
    _mm512_mask_blend_ps(BLEND_MASK,\
        _mm512_add_ps(X, _mm512_permute4f128_ps(X, PERM_TYPE)),\
        _mm512_add_ps(Y, _mm512_permute4f128_ps(Y, PERM_TYPE)))
```

В реализации `matmat8_opt` (листинг 5.2) суммирование половинок четырех регистров ZMM заканчивается на операции blend третьей фазы из рис. 5.2.

Листинг 5.2. Векторизованная версия перемножения матриц $8 \times 8$

```
void matmat8_opt(float* __restrict a, float* __restrict b,
                 float* __restrict r)
{
    ...
    ind = ... // indices for read b matrix data
    for (int j = 0; j < V8; j += 2)
    {
        ...
        // two columns of matrix b
        bj = _mm512_i32gather_ps(ind, &b[j], _MM_SCALE_4);
        bj2 = _mm512_permute4f128_ps(bj, _MM_PERM_BADC);
        ...
        // prepare data for horizontal add
        a0 = _mm512_load_ps(&a[ii0]);
        m0 = _mm512_mul_ps(a0, bj);
        m1 = _mm512_mul_ps(a0, bj2);
        ...
        // horizontal add
        x0 = SWIZ_2_ADD_2_BLEND_1(m0, m1, _MM_SWIZ_REG_CDAB, 0xAAAA);
        x1 = SWIZ_2_ADD_2_BLEND_1(m2, m3, _MM_SWIZ_REG_CDAB, 0xAAAA);
        x2 = SWIZ_2_ADD_2_BLEND_1(m4, m5, _MM_SWIZ_REG_CDAB, 0xAAAA);
        x3 = SWIZ_2_ADD_2_BLEND_1(m6, m7, _MM_SWIZ_REG_CDAB, 0xAAAA);
        m0 = SWIZ_2_ADD_2_BLEND_1(x0, x1, _MM_SWIZ_REG_BADC, 0xCCCC);
        m1 = SWIZ_2_ADD_2_BLEND_1(x2, x3, _MM_SWIZ_REG_BADC, 0xCCCC);
        x0 = PERM_2_ADD_2_BLEND_1(m0, m1, _MM_PERM_CDAB, 0xF0F0);
        ...
        // save result
        ind1 = ... // indices for save result
        _mm512_i32scatter_ps(r, ind1, x0, _MM_SCALE_4);
    }
}
```

Реализация перемножения матрицам $16 \times 16$ выполняется аналогично. Выполнение горизонтальных операций сложения для `matmat16` охватывает все четыре фазы из рис. 5.2.

Такой подход удобен для матриц $8 \times 8$ и $16 \times 16$, так как в один векторный регистр помещается ровно 8 значений формата FP64 или 16 значений формата FP32. При работе с другими размерностями возникает проблема неполного использования элементов векторных регистров, что приводит к снижению производительности. При проведении вычислений с помощью метода RANS/ILES [296] расчетные параметры компонуются внутри матриц $5 \times 5$, эти матрицы являются подматрицами матриц размера $8 \times 8$ (рис. 5.4). Рассмотрим векторизацию перемножения таких матриц [352].

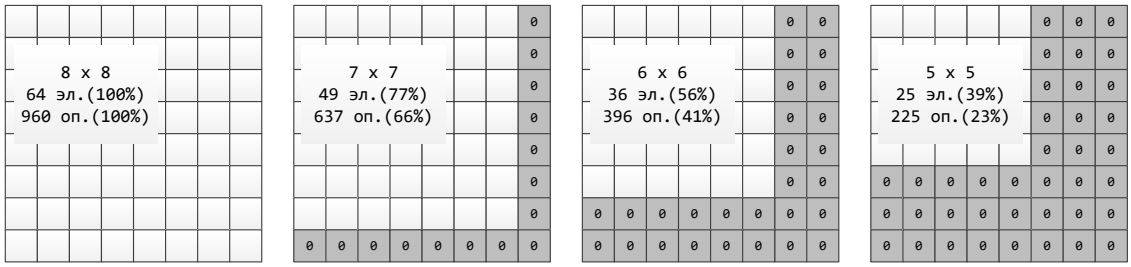


Рис. 5.4. Иллюстрация матриц размера $8 \times 8$, $7 \times 7$, $6 \times 6$, $5 \times 5$, представленных как подматрицы матрицы $8 \times 8$

Идея объединения скалярных произведений обладает существенными недостатками. Наличие медленных инструкций gather/scatter приводит к замедлению чтения данных из памяти. Также выполнение сначала поэлементного перемножения векторов, а затем параллельное нахождение сумм их элементов делают невозможным использование комбинированных FMA-инструкций.

Запишем значения элементов $i$-й строки результирующей матрицы:

$$\begin{cases} r_{i0} = a_{i0}b_{00} + a_{i1}b_{10} + \cdots + a_{i7}b_{70} \\ \cdots \\ r_{i7} = a_{i0}b_{07} + a_{i1}b_{17} + \cdots + a_{i7}b_{77} \end{cases} \tag{5.1}$$

или в векторном виде $\overline{r}_i = a_{i0}\overline{b}_0 + a_{i1}\overline{b}_1 + \cdots + a_{i7}\overline{b}_7$, что в объединении с выражением для $i+1$-ой строки $\overline{r}_{i+1} = a_{i+1,0}\overline{b}_0 + a_{i+1,1}\overline{b}_1 + \cdots + a_{i+1,7}\overline{b}_7$ приводит к выражению

$$\overline{r}|_{i+1}^{i} = \sum_{j=0}^{7} \left( \overline{a}|_{i+1,j}^{ij} \circ \overline{b}|_{j}^{j} \right), \tag{5.2}$$

где $\overline{r}|_{i+1}^{i}$ – комбинированный вектор состоящий из векторов $\overline{r}_i$ и $\overline{r}_{i+1}$, $\overline{b}|_{j}^{j}$ – комбинированный вектор состоящий из двух копий вектора $\overline{b}_j$, а $\overline{a}|_{i+1,j}^{ij}$ – вектор, первые 8 элементов которого равны $a_{ij}$, а остальные 8 элементов равны $a_{i+1,j}$. Получающийся по формуле (5.2) комбинированный вектор $\overline{r}|_{i+1}^{i}$ расположен в памяти последовательно, и записывать его в память можно с помощью интринсика `_mm512_store_ps` (для четного $i$ вектор $\overline{r}|_{i+1}^{i}$ выровнен в памяти должным образом). Другие комбинированные векторы в (5.2) получаются с

помощью инструкции perm (интринсик `_mm512_permutexvar_ps`), примененной к соответствующим загруженным соседним строкам матриц $a$ и $b$. При реализации (5.2) не требуется использование gather/scatter, так как работа ведется только со строками. После того, как векторы $\overline{a}|^{ij}_{i+1,j}$ и $\overline{b}|^{j}_{j}$ сформированы, нужно выполнить их попарное поэлементное перемножение, после чего сложить в один вектор (8 операций поэлементного умножения, 7 операций сложения). Эти действия можно выполнить, используя комбинированные операции fmadd. Для вычисления значения $\overline{r}|^{i}_{i+1}$ потребуется 8 векторных операций (1 mul и 7 fmadd) (рис. 5.5).

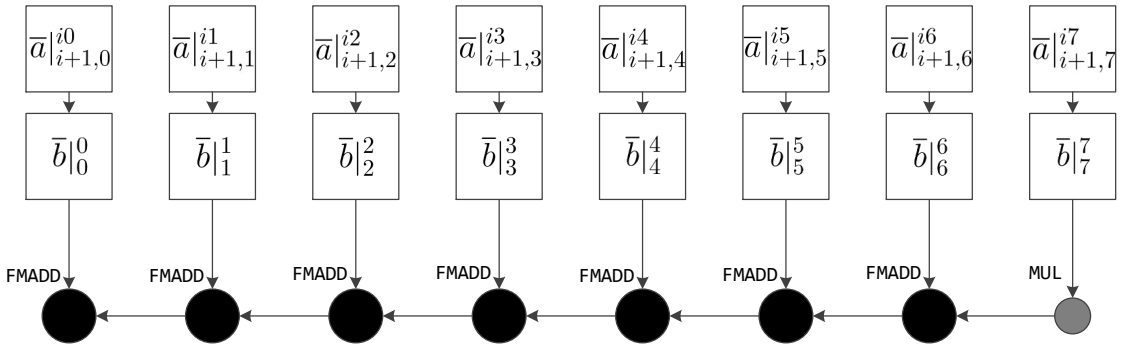


Рис. 5.5. Схема вычисления двух соседних строк результирующей матрицы путем последовательного сложения попарно перемноженных векторов

Рассмотрим перемножение двух матриц размера $8 \times 8$. Для реализации выполним полную загрузку обеих матриц $a$ и $b$. Далее требуется сформировать 8 векторов вида $\overline{b}|^{j}_{j}$, для чего потребуется еще 8 операций perm (для каждой пары загруженных строк матрицы $b$ нужно выполнить дублирование первой строки и дублирование второй строки). После подготовки всех необходимых данных выполняется вычисление значений результирующей матрицы. Схема с рис. 5.5 представляет собой блок операций, обеспечивающий вычисление двух соседних строк результирующей матрицы. Реализация блока состоит из 8 операций perm, 1 операции mul и 7 операций fmadd, кроме того выполняется одна операция записи в память. Всего выполняется четыре таких блока, что в сумме и с учетом операций подготовки данных приводит к следующему итогу: 8 простых операций чтения из памяти, 40 операций perm, 4 операции mul, 28 операций fmadd, 4 простые операции записи в память.

Таблица 5.1. Статистика по количеству операций в невекторизованном и векторизованном вариантах функций перемножения матриц размера $8 \times 8$, $7 \times 7$, $6 \times 6$, $5 \times 5$

| | Невекторизованный вариант | Векторизованный вариант |
|---|---|---|
| $8 \times 8$ | 512 mul, 448 add<br>960 арифметических операций | 4 mul, 28 fmadd, 40 perm<br>960 арифметических операций |
| $7 \times 7$ | 343 mul, 294 add<br>637 арифметических операций | 4 mul, 24 fmadd, 35 perm<br>832 арифметические операции |
| $6 \times 6$ | 216 mul, 180 add<br>396 арифметических операций | 3 mul, 15 fmadd, 24 perm<br>528 арифметических операций |
| $5 \times 5$ | 125 mul, 100 add<br>225 арифметических операций | 3 mul, 12 fmadd, 20 perm<br>432 арифметические операции |

Заметим, что 28 векторных операций fmadd и 4 векторные операции mul соответствуют $(28 \times 2 + 4) \times 16 = 960$ скалярным операциям, что совпадает с количеством скалярных операций, требуемых для перемножения двух матриц $8 \times 8$, то есть в реализации нет лишних арифметических операций. При реализации перемножения матриц размера $7 \times 7$, $6 \times 6$, $5 \times 5$ удаляются заведомо лишние векторные операции, однако все равно остаются элементы векторов, обработка которых избыточна, что приводит к снижению эффективности векторизации. Приведем таблицу с точным подсчетом количества скалярных операций для невекторизованных функций и количества векторных операций для их векторизованных аналогов (таблица 5.1).

Из таблицы 5.1 видно, что с уменьшением размера перемножаемых матриц возрастает избыточность вычислений (для размера $5 \times 5$ эта избыточность почти достигает двукратного размера), что сказывается отрицательно на эффективности векторизации. На рис. 5.6 приведена статистика по количеству операций mul и add в скалярных версиях перемножения матриц, а также mul, fmadd и perm в векторизованных версиях.

Приведенный способ векторизации перемножения матриц размера был опробован на процессоре Intel Xeon Phi 7290 KNL (таблица 3.2). Были рассмотрены 4 функции: `mul_8x8`, `mul_7x7`, `mul_6x6`, `mul_5x5`, выполняющие перемножение матриц соответствующих размеров. Для каждой функции бы-

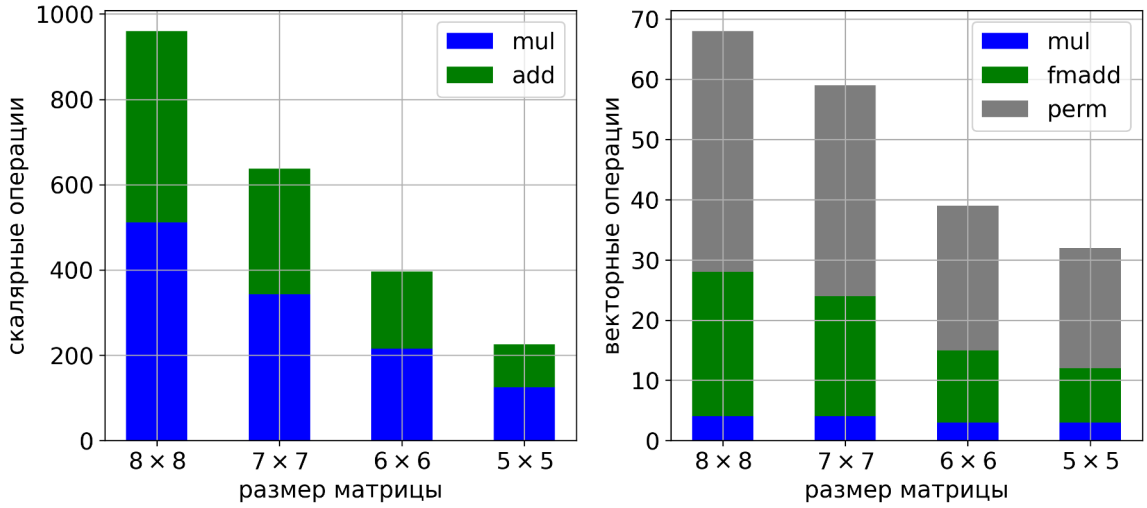


Рис. 5.6. Количество операций в скалярных и векторных вариантах перемножения матриц разных размеров

ли рассмотрены 3 варианта реализации. `VECT_OLD` – векторизация с помощью объедиенения скалярных произведений, `FULL_UNROLL` – ручное вычисление каждого элемента с полной раскруткой всех циклов, `VECT_NEW` – рассмотренный подход, основанный на обращениях только к строкам матриц и использовании комбинированных операций fmadd.

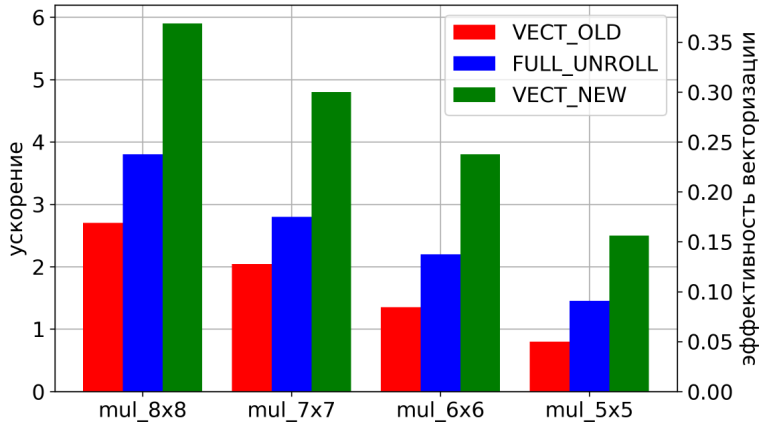


Рис. 5.7. Ускорение перемножения матриц с помощью подходов `OLD_VECT`, `FULL_UNROLL`, `NEW_VECT`

На рис. 5.7 показаны результаты тестирования рассмотренных подходов на машине. Из рисунка следует, что `VECT_OLD` оказался наименее эффективным подходом, его применение даже менее выгодно, чем прямое написание скалярного кода. Для матриц размера $5 \times 5$ этот метод оптимизации вовсе приводит к замедлению оригинальной неоптимизированной версии функции. Метод

VECT_NEW демонстрирует наилучший результат, на матрицах размера $8 \times 8$ продемонстрировано ускорение почти в 6 раз по сравнению с оригинальным кодом. При понижении размерности матриц эффективность векторизации несколько снижается, однако даже для матриц размера $5 \times 5$ наблюдается ускорение примерно в 2,5 раза.

## 5.3 Плоский цикл

В разделе 5.2 были рассмотрены различные варианты векторизации функций, реализующих работу с малоразмерными матрицами. Эти варианты базировались на идее выделения однотипных операций замены их на векторные аналоги. Из результатов, полученных в разделе 5.2, можно заключить, что эффективность векторизации сильно зависит от способа выделения таких однотипных операций, от порядка обращения в память за данными, от избыточности вычислений и других факторов.

Неоднозначность и искусственность векторизации программного кода путем поиска однотипных операций порождает потребность выработки некоторого универсального подхода к векторизации, который мог бы применяться к широкому спектру расчетных приложений. Выполним поиск такого программного контекста, векторизация которого может быть выполнена достаточно прозрачно и универсально, а результирующий код будет обладать высокой эффективностью.

### 5.3.1 Понятие плоского цикла

Векторизация программного контекста не может быть применена автоматически к коду произвольного вида. В этом разделе определим подходящий для векторизации программный контекст специального вида – плоский цикл – и опишем его свойства [353, 354].

Идея плоского цикла состоит в объединении $w$ экземпляров скалярного блока в единый векторный блок, состоящий из аналогичных векторных инструкций. На рис. 5.8 слева представлена схема скалярного блока `block`,


Рис. 5.8. Скалярный блок block и аналогичный емувекторный блок BLOCK

со скалярными входными данными $x$ и $y$ и скалярным результатом $r$. Будем считать, что этот блок соответствует концепции чистых вычислений, то есть результат его выполнения зависит только от значений $x$ и $y$ (например, отсутствуют побочные эффекты через глобальную память или операции ввода-вывода). Идеология чистых вычислений берет свое начало из парадигмы функционального программирования [355, 356], использование такого подхода открывает возможности для оптимизации программного кода, в частности для компиляторов. Если рассмотреть вместо одной копии скалярного блока `block` несколько копий ($w$ штук) с разными наборами входных данных, то их можно трактовать как выполнение векторного блока `BLOCK`, входными данными которого являются векторы $X$ и $Y$ длины $w$, а выходным значением является вектор $R$ также длины $w$. В этом случае можно говорить, что векторный блок `BLOCK` представляет собой векторизованную версию скалярного блока `block` при ширине векторизации $w$ (рис. 5.8 справа). В этом случае логику векторного блока `BLOCK` можно записать в виде, представленном на листинге 5.3.

Листинг 5.3. Логика векторного блока `BLOCK`

```
// BLOCK
for (int i = 0; i < w; ++i)
{
    r[i] = block(x[i], y[i])
}
```

Цикл вида, представленного на в листинге 5.3, будем называть плоским циклом. Определим свойства, присущие плоскому циклу.

В качестве плоского цикла будем рассматривать цикл for с индуктивной переменной $i \in [0, w-1]$ (`for (int i = 0; i < w; ++i`). Такое условие не является строгим ограничением, так как произвольный цикл for с количеством

итераций $n$ может быть разбит (с помощью расщепления по индуктивной переменной) на $\left\lfloor \frac{n}{w} \right\rfloor$ циклов с количеством итераций $w$ каждый и еще один цикл с количеством итераций $n - \left\lfloor \frac{n}{w} \right\rfloor w$ (если $w \nmid n$), называемый эпилогом цикла. Будем считать, что мы рассматриваем циклы с большим числом итераций $n \gg 1$. В таком случае эпилогом цикла можно пренебречь, поэтому будем считать, что $w \mid n$, и эпилог отсутствует.

Во-вторых, будем считать, что внутри плоского цикла на $i$-ой итерации все обращения в память на запись имеют вид `a[i]`. Обращения к глобальным данным на чтение могут иметь вид `a[i]` либо могут быть обращениями к глобальным скалярным данным. Такое ограничение гарантирует отсутствие конфликтов между итерациями по обращениям в память. Дополнительно будем считать, что при обращении в память все данные выровнены правильным образом, то есть адрес элемента `a[0]` выровнен в памяти на размер вектора.

Последним требованием будем считать отсутствие других межитерационных зависимостей внутри цикла. Пример цикла с межитерационными зависимостями можно увидеть на листинге 5.4.

Листинг 5.4. Пример цикла с межитерационной зависимостью

```
1 for (int i = 0; i < w; ++i)
2 {
3     s += x[i]; // global variable modification
4 }
```

**Определение 5.6.** *Плоским циклом будем называть цикл for, в котором индуктивная переменная $i$ последовательно принимает значения от 0, до $w - 1$, где $w$ – ширина векторизации (`for (int i = 0; i < w; ++i)`), и удовлетворяющий требованиям по обращению к глобальным данным (на $i$-ой итерации все обращения к данным на запись имеют вид `a[i]`, а обращения к данным на чтение имеют вид либо `a[i]`, либо являются чтением скалярных значений), выравниванию глобальных данных в памяти (все массивы данных, обращение к которым на $i$-ой итерации имеет вид `a[i]`, выровнены в памяти на размер вектора), а также отсутствию других межитерационных зависимостей.*

**Определение 5.7.** *Цикл* `for (int i = 0; i < w; ++i)`*, в котором нарушаются некоторые требования, предъявляемые к плоским циклам, будем называть квазиплоским.*

На рис. 5.9 приведена схема векторизации плоского цикла, тело которого содержит только простые арифметические инструкции. На этом рисунке сверху показана векторизация плоского цикла, путем замены всех арифметических операций векторными аналогами. На рисунке внизу приведено обозначение векторизации плоского цикла.

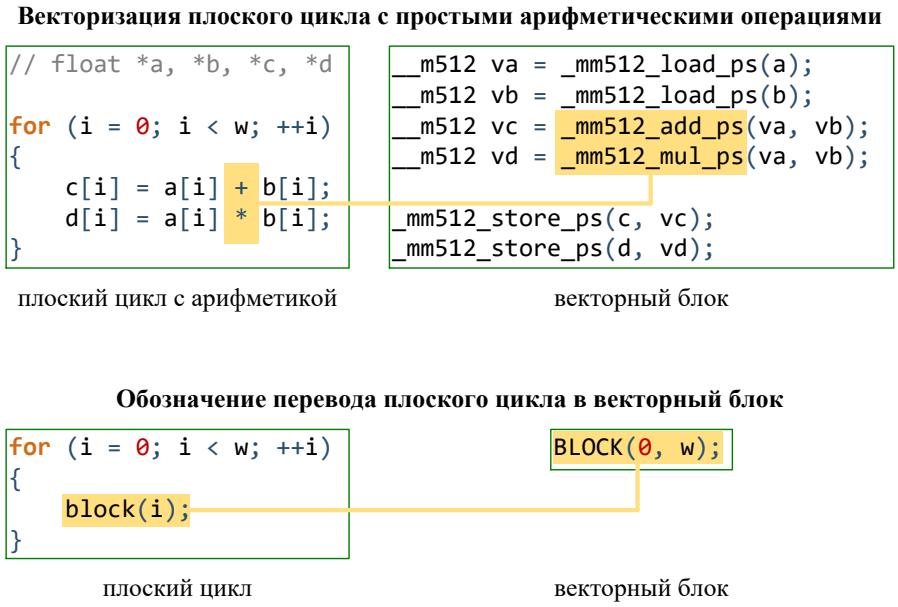


Рис. 5.9. Пример векторизации плоского цикла с простой арифметикой

Учитывая все требования, предъявляемые к плоскому циклу, можно заключить, что итерации плоского цикла являются независимыми, а значит могут выполняться в произвольном порядке, в том числе и одновременно.

Плоские циклы, обладающие этими свойствами, являются удобным контекстом для векторизации, в большинстве случаев они могут быть векторизованы с помощью векторных инструкций AVX-512 с помощью перевода тела цикла в предикатное представление и замены скалярных инструкций векторными аналогами, реализованными с помощью интринсиков [183, 357].

На рис. 5.10 приведена схема векторизации плоского цикла, тело которого содержит операции, выполняемые под условием. Для этого используется пред-

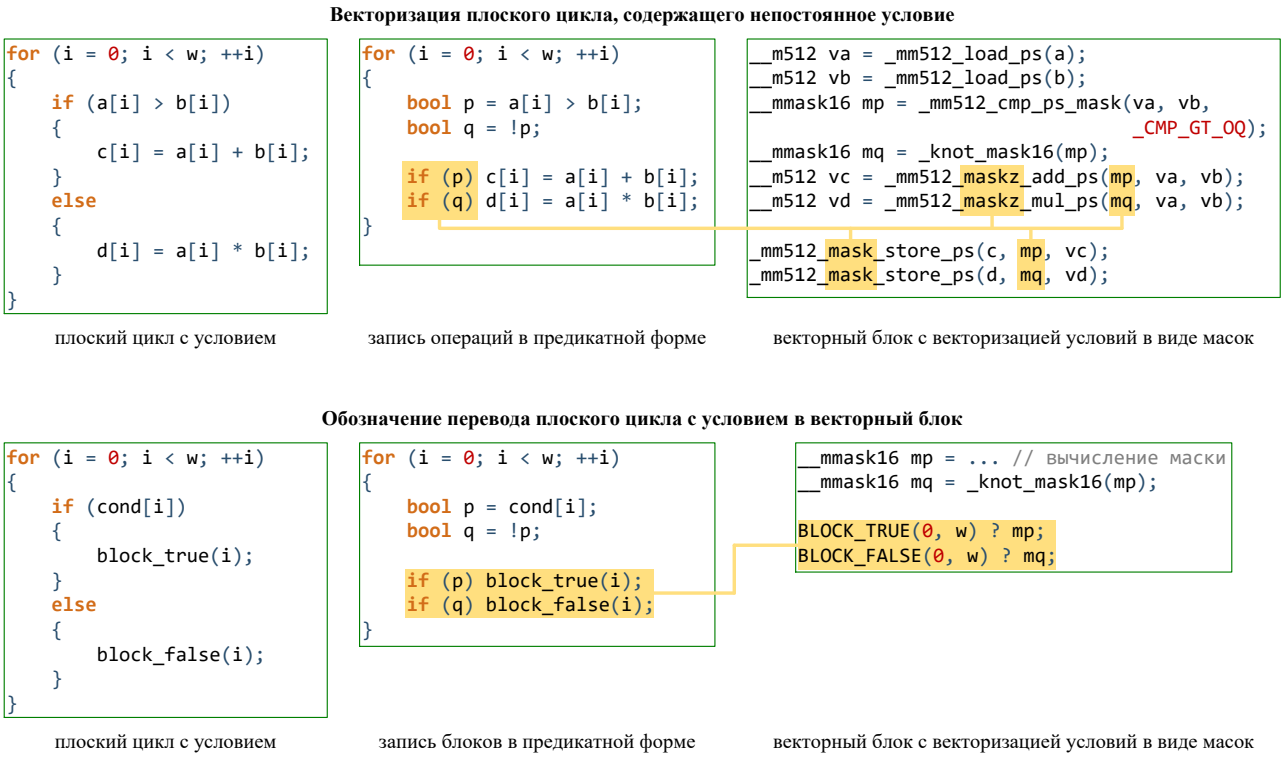


Рис. 5.10. Пример векторизации плоского цикла с использованием предикатов

ставление тела цикла в предикатной форме, то есть для каждого выполняемого оператора указано условие выполнения. При векторизации предикаты трансформируются в векторные маски, что позволяет использовать это уникальное свойство для векторизации программного кода с условиями [184].

Цикл, отвечающий всем требованиям плоских циклов, кроме количества итераций (если он содержит $n > w$ итераций) может быть расщеплен по индуктивной переменной и представленный в виде последовательности плоских циклов, как показано на рис. 5.11. Эпилог цикла векторизуется с использованием дополнительного условия, формируемого по количеству итераций.

Многие практические вычислительные задачи состоят из выполнения однотипных вычислений, применяемых к разным наборам данных, которые можно сгруппировать, трансформировав в плоский цикл, как это продемонстрировано на рис. 5.8 на примере скалярного блока `block`. Отметим, что практика организации вычислений в виде плоских циклов не только расширяет возможности по ускорению кода (с помощью оптимизирующего компилятора или вручную), но само следование условиям по упрощению структуры про-

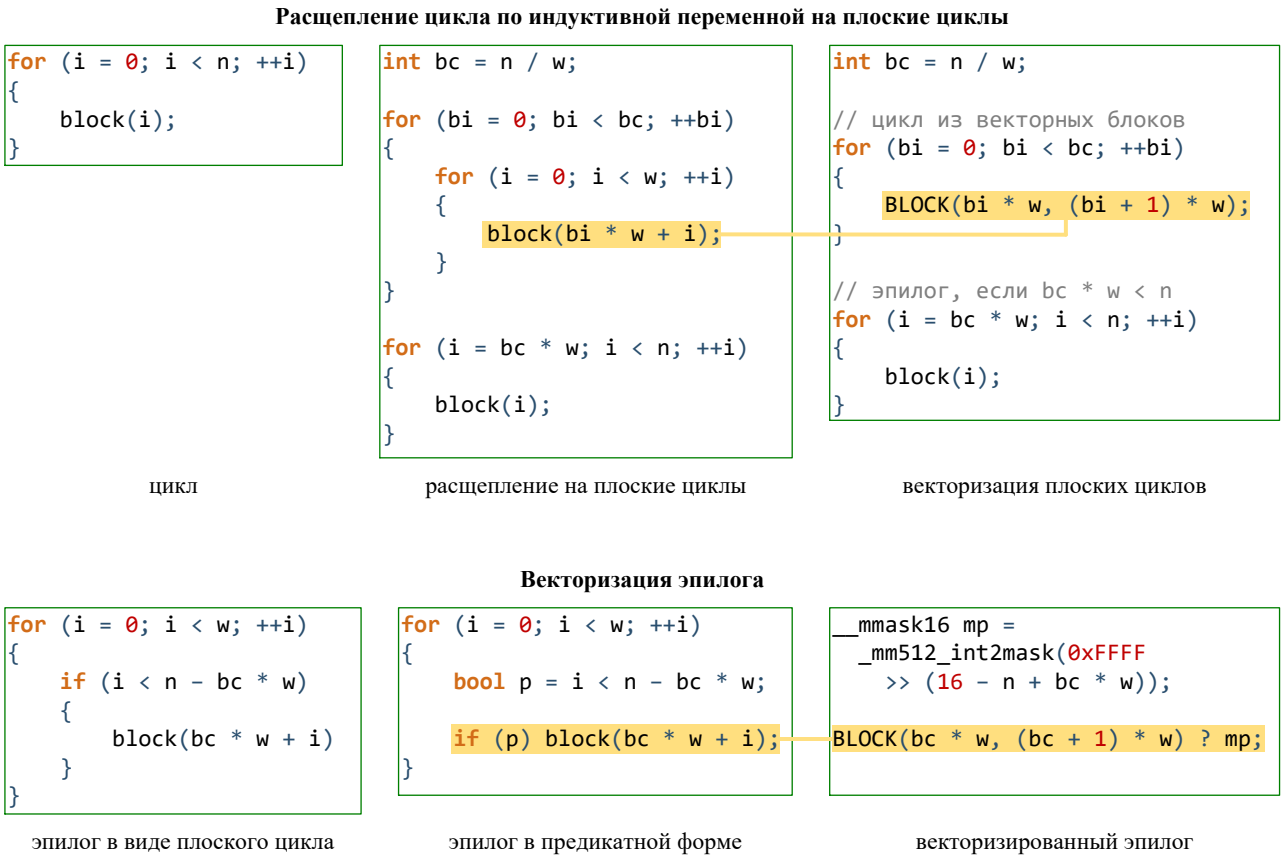


Рис. 5.11. Пример расщепления цикла по индуктивной переменной на плоские циклы

граммного кода приводит к повышению безопасности и надежности программ, что можно наблюдать на примере специализированных стандартов, например MISRA, широко применяемого в авиационной промышленности [358].

### 5.3.2 Граф потока управления тела плоского цикла

Сложность векторизации плоского цикла зависит от особенностей его тела `block`. Исследовать структуру тела можно с помощью графа потока управления (control flow graph, CFG) [359]. Узлами этого графа являются линейные участки, состоящие из последовательностей инструкций, а ребрами – передача управления между этими линейными участками, выполняемая с помощью команд перехода либо перехода по провалу [360]. Граф потока управления является логической структурой, отражающей параллелизм программы на уровне линейных участков. Он используется компилятором для применения различных глобальных оптимизаций [361].

Кроме самой структуры графа потока управления для оптимизации программного кода важна статистическая информация об исполнении программы: количество выполнений различных линейных участков и данные о частоте переходов между ними. Такая статистическая информация называется профилем исполнения, и для корректного проведения оптимизаций требуется правильным образом собирать и корректировать этот профиль [362]. Для векторизации программного кода профиль исполнения программы приобретает особенную важность, так как на эффективность векторизации сильно влияет плотность и даже структура векторных масок, которая сильно разнится при сравнении результатов, полученных при генерации случайных входных данных, и при использовании данных реальных расчетов.

Плотностью векторной маски будем называть отношение количества единичных битов в ней к ее длине (либо просто количество единичных битов). В качестве профиля исполнения приложения будем использовать данные, содержащие только счетчики узлов и ребер, а также вероятности ребер. Обозначим некоторый узел CFG через $v$. Счетчиком ребра $e$ называют количество переходов по этому ребру в процессе выполнения программы. Счетчиком узла $v$ называют суммарное количество счетчиков всех входных ребер. Вероятностью ребра $e$ называют отношение счетчика этого ребра с счетчику узла, из которого это ребро выходит. Построенный по телу плоского цикла граф потока управления с собранным профилем исполнения используется для принятия решения о выборе методов векторизации программного контекста.

### 5.3.3 Семантика векторных инструкций

Векторные инструкции AVX-512 позволяют не просто объединять $w$ однотипных скалярных операций, но также обеспечивают выборочное их исполнение с помощью масок (предикатов), что делает инструкции AVX-512 мощным инструментом для векторизации плоских циклов. Опишем семантику некоторых векторных инструкций AVX-512, пригодных для векторизации плоских циклов. Основное внимание будем уделять операциям, работающим с

вещественными числами, так как обычно они являются основой суперкомпьютерных приложений. Все приводимые ниже векторные инструкции работают с упакованными данными формата FP32, для FP64 существуют аналоги.

Маленькими латинскими буквами будем обозначать элементы данных, с которыми производятся операции. Арифметические операции будем записывать в естественном виде, например $r = a + b$. Векторы, составленные из отдельных элементов будем записывать с помощью заглавных латинских букв. То есть будем считать, что $A$ это вектор, состоящий из $w$ отдельных элементов $A[i]$. Под записью $R = A + B$ будем понимать поэлементную сумму векторов $A$ и $B$ и копирование результата в вектор $R$. Заменяя операцию сложения произвольной операцией $op$ (не обязательно операцией двух аргументов), получим запись семантики векторной поэлементной операции в виде $R = op\ A, B$.

Для рассмотрения семантики векторных операций, работающих с масками, понадобится представление векторных предикатов. Предикаты будем обозначать латинскими буквами в галочкой наверху. Использование предиката при выборе одного из двух аргументов будем записывать с помощью тернарного оператора: $r = \check{p}\ ?\ a : b$. В векторном аналоге этой записи $R = \check{P}\ ?\ A : B$ эта операция выполняется поэлементно для всех элементов векторов.

После рассмотрения записи семантики векторных инструкций для векторизации плоских циклов рассмотрим основные классы пригодных для этого инструкций (примеры операций и их семантика приведены в таблице 5.2).

Первым типом операций являются векторные операции с одним аргументом. В этом случае к каждому элементу вектора применяется одна и та же операция, после чего результаты записываются в результирующий вектор. С помощью предикатного аргумента можно выбрать множество обрабатываемых элементов. Если к элементу вектора не должна быть применена рассматриваемая операция, то соответствующий элемент результирующего вектора может быть либо оставлен без изменения либо обнулен.

Аналогично записывается семантика векторных инструкций с двумя и тремя аргументами. Арифметические операции с тремя аргументами комбинированные операции FMA для вычисления значений $\pm a \cdot b \pm c$.

Таблица 5.2. Инструкции AVX-512 для работы с вещественными числами и их семантика

| Имя инструкции | Семантика инструкции |
|---|---|
| VMOVAPS, VMOVUPS, VSQRTPS, VGETEXPPS, VGETMANTPS, VRCP14PS, VREDUCEPS, VRNDSCALEPS, VRSQRT14PS, VSCALEFPS | $R = op\ A$<br>$R = \check{P}\ ?\ (op\ A) : R$<br>$R = \check{P}\ ?\ (op\ A) : 0$ |
| VADDPS, VANDPS, VANDNPS, VDIVPS, VMAXPS, VMINPS, VMULPS, VORPS, VSUBPS, VRANGEPS | $R = op\ A, B$<br>$R = \check{P}\ ?\ (op\ A, B) : R$<br>$R = \check{P}\ ?\ (op\ A, B) : 0$ |
| VFMADD*PS, VFMSUB*PS, VFNMADD*PS, VFNMSUB*PS | $R = op\ R, A, B$<br>$R = \check{P}\ ?\ (op\ R, A, B) : R$<br>$R = \check{P}\ ?\ (op\ R, A, B) : 0$ |
| VCMPPS | $\check{P} = op\ A, B$<br>$\check{P} = \check{Q}\ ?\ (op\ A, B) : 0$ |
| VBLENDPS | $R = \check{P}\ ?\ A : B$ |

Следующий класс операций это операции сравнения. В таблице 5.2 этот класс представлен единственной операцией VCMPPS, однако данная операция скрывает в себе все множество различных операций сравнения (VCMPEQPS, VCMPLEPS, VCMPNEPS и остальные), что регулируется отдельным параметром. Эти операции выполняют поэлементное сравнение двух векторов и записывают результаты в векторный предикат.

Последний рассматриваемый класс векторных инструкций представлен одной инструкцией VBLENDPS, которая является реализацией векторного тернарного оператора $R = \check{P}\ ?\ A : B$ (эта операция широко используется при оптимизации кода, будем называть ее слиянием векторов по маске).

Глядя на таблицу 5.2, можно заметить, что опсания семантики приведенных в ней векторных операций являются плоскими циклами. Верно также и обратное – если некий плоский цикл можно записать в виде семантики одной или нескольких векторных инструкций, то он может быть реализован с помощью этих инструкций. В следующих разделах рассматриваются оптимизации, с помощью которых плоские циклы различного вида могут быть записаны с помощью семантики некоторых векторных операций.

### 5.3.4 Представление расчетов в виде плоских циклов

Если вычисления организованы в виде набора плоских циклов, а тела этих плоских циклов являются достаточно простыми с точки зрения управления, оптимизирующий компилятор может успешно создавать векторный код. В этом случае нет необходимости явно использовать ассемблерные вставки или функции. Рассмотрим основные аспекты приведения программного кода в форму, пригодную для автоматической векторизации [363], для которой достигается высокая производительность результирующего кода. Подходы, применяемые к векторизации плоских циклов, могут частично применяться также к квазиплоским циклам с некоторой потерей производительности. В этом разделе проведен анализ потерь производительности при векторизации квазиплоских циклов на примере газодинамических расчетов с использованием схемы Стегера-Уорминга и метода погруженной границы из раздела 2.9.2. Для анализа эффективности векторизации проводился эксперимент по организации программного кода этого решателя и сборки его под микропроцессор Intel Xeon Phi 7290 KNL (таблица 3.2).

Рассмотрим на примере функции `d_to_u` из схемы вычислений, представленной на рис. 2.47, способы организации данных для эффективного проведения вычислений. Функция `d_to_u` переводит вектор величин $D = [\rho, u, v, w, p]$ в вектор величин $U = [\rho, \rho u, \rho v, \rho w, E]$ для каждой ячейки. Все ячейки обрабатываются в цикле независимо друг от друга. Интуитивным способом вычисления могут быть организованы в виде следующей схемы: для каждой ячейки создается структура, которая содержит все необходимые величины, и действия по переводу вектора $D$ в вектор $U$ выполняется над полями этой структуры (листинг 5.5). Такая организация данных называется организацией в виде «массива структур» (Array Of Structures, AOS).

При попытке векторизации цикла из листинга 5.5 во время объединения $w$ итераций вместо чтения из памяти скалярного значения (например, плотности) должно происходить чтение значений плотности из несколько последовательно расположенных структур данных. То есть чтение вектора

плотностей должно осуществляться не из последовательной области памяти. Для таких целей в наборе векторных инструкций AVX-512 предусмотрены инструкции gather/scatter, однако их эффективность гораздо ниже чтения последовательной области памяти размера 512 бит даже при использовании предварительной подкачки данных. И хоть организация данных виде «массива структур» наиболее предпочтительная с точки зрения идеологии объектно-ориентированного программирования, она оказывается непригодной для успешного применения векторизации.

Листинг 5.5. Организация данных в виде «массив структур»

```
struct Cell
{
    double rho;
    double u; double v; double w;
    double p;
    double rho_u; double rho_v; double rho_w;
    double E;
};
Cell cells[N];

void d_to_u()
{
    for (int i = 0; i < N; ++i)
    {
        double rho = cells[i].rho;
        double u = cells[i].u;
        double v = cells[i].v;
        double w = cells[i].w;
        double p = cells[i].p;

        cells[i].rho_u = rho * u;
        cells[i].rho_v = rho * v;
        cells[i].rho_w = rho * w;
        cells[i].E = 0.5 * rho * (u * u + v * v + w * w)
                     + p / (GAMMA - 1.0);
    }
}
```

Естественным решением оптимизации вычислений является организация расположения данных в памяти в виде набора массивов (Strusture Of Arrays, SOA), как показано на листинге 5.6. Теперь после объединения нескольких итераций цикла команды скалярного доступа в память трансформируются в векторные аналоги доступа к последовательной области памяти размера 512 бит. Вся же арифметика, которая присутствует в коде функции, имеет свои векторные аналоги в наборе инструкций AVX-512.

Листинг 5.6. Организация данных в виде «набор массивов»

```
double rhos[N];
double us[N]; double vs[N]; double ws[N];
double ps[N];
double rho_us[N]; double rho_vs[N]; double rho_ws[N];
double Es[N];

void d_to_u()
{
    for (int i = 0; i < N; ++i)
    {
        double rho = rhos[i];
        double u = us[i];
        double v = vs[i];
        double w = ws[i];
        double p = ps[i];

        rho_us[i] = rho * u;
        rho_vs[i] = rho * v;
        rho_ws[i] = rho * w;
        Es[i] = 0.5 * rho * (u * u + v * v + w * w)
                + p / (GAMMA - 1.0);
     }
}
```

Изменение расположения данных в памяти для облегчения компилятору задачи по векторизации может происходить как в ручном режиме, так и с использованием специальных инструментов. Для прозрачной транформации может быть использована библиотека шаблонов SIMD Data Layout Template (SDLT), реализованная для языка программирования C++ [364].

Другой критической проблемой, влияющей на эффективность векторизации, является наличие избыточных условных операций внутри цикла. Набор инструкций AVX-512 содержит специальные масочные аргументы, с помощью которых можно векторизовать программный код с управлением практически любой сложности, но при увеличении количества условий эффективность векторизации существенно снижается. В качестве примера такого негативного эффекта рассмотрим гнездо циклов из функции `calc_flows`, в которой корректируются газодинамические величины $U = [\rho, \rho u, \rho v, \rho w, E]$ с помощью потоков через все грани ячейки. Для обработки граничных условий в код добавляются дополнительные проверки (например, проверка $i = 0$ для обработки левого граничного условия), что увеличивает количество условий внутри цикла и снижает эффективность векторизации (листинг 5.7).

Листинг 5.7. Гнездо циклов с условием во внутреннем цикле

```
for (int k = 0; k < NZ; ++k)
{
    for (int j = 0; j < NY; ++j)
    {
        for (int i = 0; i < NX; ++i)
        {
            if (i == 0)
            {
                ... // left boundary condition
            }

            ... // rest code
        }
    }
}
```

Все три цикла гнезда (по индуктивным переменным $i$, $j$, $k$) не содержат межитерационных зависимостей, они могут быть переставлены в произвольном порядке, как и итерации внутри любого из них. Условия обработки границ расчетной области являются константными для некоторого среза ячеек сетки. Это означает, что для гнезда циклов такое условие является частично константным, и гнездо может быть разбито по этому условию. Общая схема расщепления цикла по константному условию приведена на рис. 5.12.

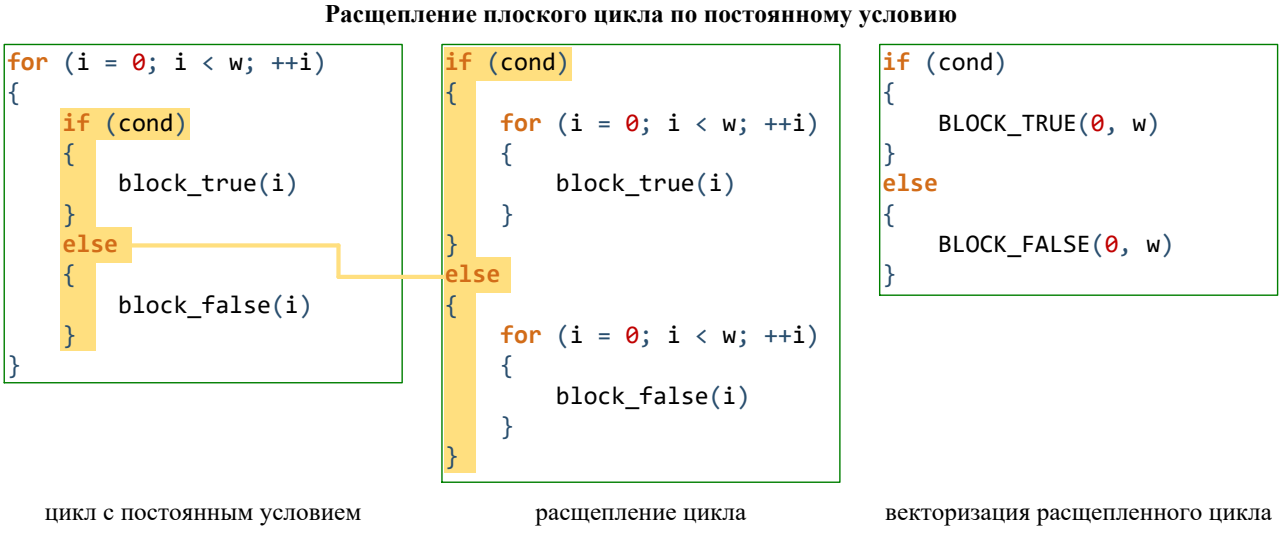


Рис. 5.12. Схема расщепления цикла по константному условию

На листинге 5.8 продемонстрировано разбиение гнезда по условию $i = 0$. Аналогично можно выполнить разбиение по остальным условиям, освободив от них основное гнездо циклов, которое после этого успешно векторизуется.

Листинг 5.8. Расщепление гнезда циклов по условию

```
for (int k = 0; k < NZ; ++k)
{
    for (int j = 0; j < NY; j++)
    {
        ... // left boundary condition with i = 0
    }
}

for (int k = 0; k < NZ; ++k)
{
    for (int j = 0; j < NY; ++j)
    {
        for (int i = 1; i < NX; ++i)
        {
            ... // rest code
        }
    }
}
```

При таком расщеплении циклов по условию нарушается условие выровненности данных внутри цикла, то есть цикл становится квазиплоским.

На рис 2.47 представлена общая схема выполнения одной итерации расчетов. Основными функциями в этих расчетах являются `approximate_values`, `d_to_u`, `calc_fgh`, `calc_flows`, `u_to_d`. Все эти функции содержат внутри себя обработку ячеек расчетной сетки в гнезде циклов либо в одном цикле. Функция `approximate_values` выполняет аппроксимацию газодинамических величин в фиктивных ячейках, содержит векторные и матричные операции и сложное управление. Функции `d_to_u`, `calc_fgh`, `u_to_d` не содержат операторов передачи управления и при условии реорганизации данных в виде «набор массивов» могут быть представлены в виде композиции плоских циклов. Функция `calc_flows` с помощью расщепления гнезда циклов по 6 условиям может быть представлена в виде композиции квазиплоских циклов.

Для автоматически векторизованного с помощью оптимизирующего компилятора icc программного кода газодинамического решателя был поставлен эксперимент по замеру ускорения и эффективности векторизации на микропроцессоре Intel Xeon Phi 7290 KNL (таблица 3.2). На рис. 5.13 представлены результаты ускорения и эффективности векторизации. Ускорение различных функций в результате векторизации сильно отличается друг от друга ввиду особенностей этих функций.

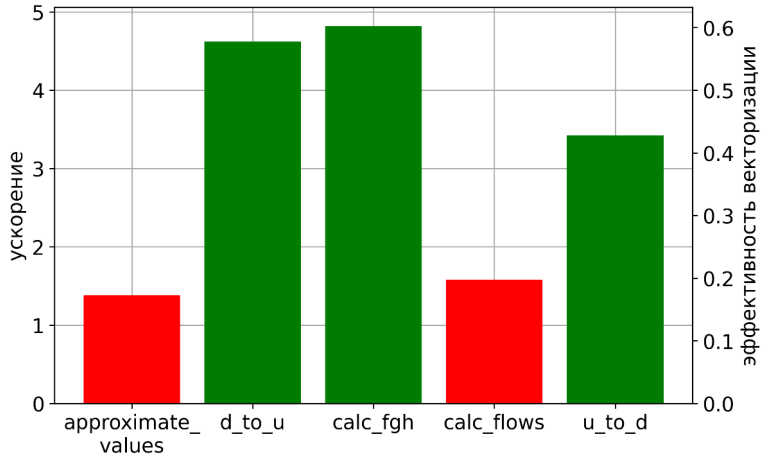


Рис. 5.13. Ускорение кода и эффективность векторизации отдельных функций газодинамического решателя

Наибольшее ускорение продемонстрировала функция `calc_fgh` вычисления потоков $F^{\pm}$, $G^{\pm}$, $H^{\pm}$. Функция содержит плоский цикл с простой арифметикой, полностью удовлетворяющий всем условиям, что делает возможным замену всех скалярных операций на векторные аналоги. В скалярной версии используются библиотечные вызовы `abs` и `sqrt`, имеющие векторные аналоги в наборе AVX-512. Обилие операций умножения и сложения делает возможным применение векторных комбинированных операций вида $\pm a \cdot b \pm c$.

Также ускорение выше среднего продемонстрировали функции `d_to_u` и `u_to_d`. Эти функции также состоят из плоских циклов, поэтому скалярные операции могут быть заменены на векторные аналоги. Более низкое ускорение функции `u_to_d` объясняется наличием операций деления, которые выполняются медленнее операций сложения и умножения.

Ускорение функции `calc_flows` составило менее двух раз даже после избавления от условий, связанных с обработкой граничных условий. Такая низкая эффективность векторизации объясняется тем, что внутри этой функции присутствуют квазиплоские циклы. Для корректировки консервативных величин внутри ячейки $(i, j, k)$ требуется обращаться за данными к смежным по граням ячейкам: $(i \pm 1, j, k)$, $(i, j \pm 1, k)$, $(i, j, k \pm 1)$. Это нарушает требование по унификации обращения к массивам данных на одной итерации плоского цикла и приводит к появлению операций gather/scatter.

Для функции `approximate_values` ситуация выглядит аналогичной, так как для выполнения аппроксимации данных в ячейке $(i, j, k)$ требуется обращаться за данными в ячейки, относящиеся к шаблону аппроксимации (на рис 2.47 нарушение требования унификации обращения за данными продемонстрировано красными стрелками, означающими, что при обработке одной ячейки сетки мы вынуждены обращаться за данными к другой ячейке).

## 5.4 Вынос маловероятных регионов

В разделе рассматривается оптимизация выноса маловероятного региона из плоского цикла. Инструкции передачи управления не имеют векторных аналогов. Наличие команд передачи управления по условию в теле плоского цикла является основной причиной потери производительности при векторизации. При наличии большого количества операторов управления оптимизирующий компилятор либо создает крайне неэффективный векторный код, либо вообще отказываться от векторизации из-за слишком низкого ожидания ускорения [365]. В процессе исполнения не весь код в теле векторизуемого цикла одинаково вероятен. Если в теле присутствуют явно маловероятные регионы (если вероятности всех входящих в регион ребер близка к нулю), то можно выполнить их вынос из цикла, а оставшийся код в теле цикла может быть векторизован либо автоматически, либо с минимальными усилиями.

Рассмотрим оптимизацию выноса маловероятного региона из цикла с помощью временного сохранения условия. Пусть есть плоский цикл, в теле которого находится маловероятный регион, все выходы которого являются выходами с итерации цикла (рис. 5.14 слева).

На рис. 5.14 слева представлен плоский цикл с тремя блоками внутри тела. Блок `block(i)` выполнятся в начале тела цикла. Далее в зависимости от условия `cond(i)` выполняется либо блок `block_true(i)` с вероятностью около единицы, либо блок `block_false(i)` с вероятностью около нуля. Пусть блоки `block(i)` и `block_true(i)` являются простым по структуре и пригодным для векторизации, а блок `block_false(i)` – маловероятный и непригодный для

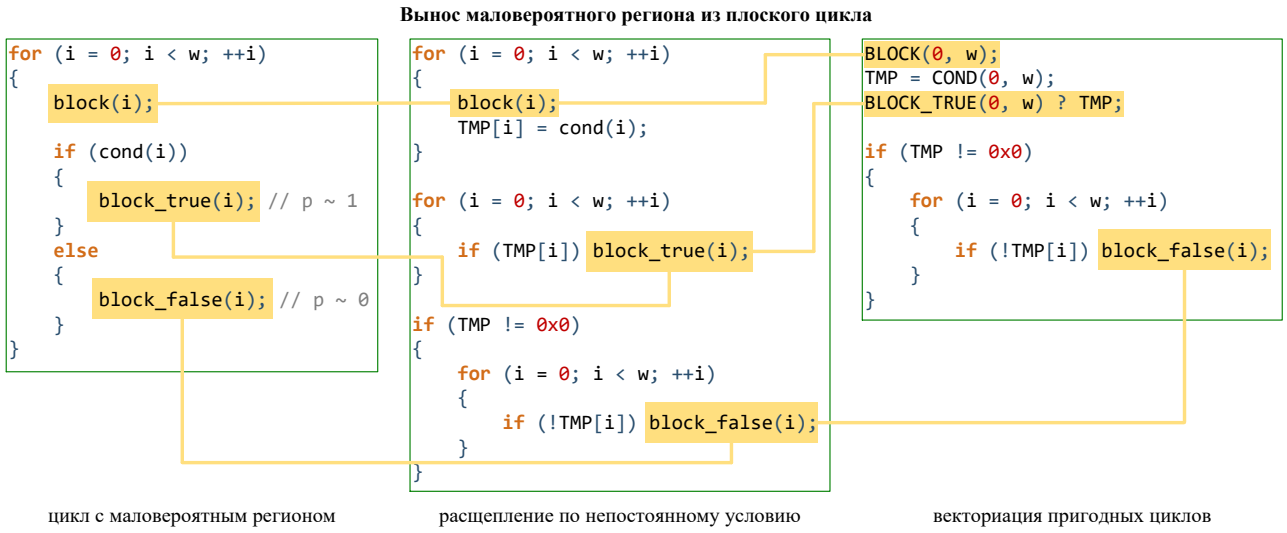


Рис. 5.14. Схема выноса маловероятного региона из плоского цикла

векторизации. В исходном виде компилятор, как правило, не справляется с векторизацией такого цикла, поэтому для его векторизации можно использовать преобразование, связанное с расщеплением этого цикла по условию. Создадим вместо одного цикла три новых. В первом цикле будет выполняться только блок `block(i)`, а все условия `cond(i)` записываться в локальный массив `TMP`. Во втором цикле на каждой итерации при истинном значении `TMP[i]` выполняется `block_true(i)`. Третий цикл инкапсулирует выполнение маловероятного кода при условии ложности `TMP[i]` (рис. 5.14 в центре).

В результате выполненных преобразований первый цикл на рис. 5.14 в центре содержит только векторизуемый безусловный код, второй цикл также содержит векторизуемый код, но с использованием предиката, который при векторизации преобразуется в векторную маску, накладываемую на все операции этого цикла. Третий код является маловероятным и его вообще не требуется оптимизировать (дополнительно перед его выполнением следует проверить маску на наличие в ней ложных элементов, что также в большинстве случаев избавит от необходимости выполнять все итерации цикла). В результате схематично векторизованная версия всех трех циклов будет выглядеть так, как это представлено на рис. 5.14 справа.

Это преобразование следует использовать при уверенности, что условие `cond(i)` является вероятным. При значении вероятности этого условия, близ-

кой к единице, логическое ускорение от применения векторизации будет близко к ширине векторизации (при условии, что `block(i)` и `block_true(i)` векторизуются идеально). Однако, если условие `cond(i)` не является вероятным, то выполнение такого преобразования приведет к деградации производительности. Также это преобразование применимо только если все выходы из выносимого региона являются выходами с итерации плоского цикла.

В разделе 2.5.2 приведено описание задачи нахождения пересечения прямой с окрестностью отрезка в пространстве. Эта задача имеет практическое значение в авиации применительно к обеспечению безопасности полетов воздушных судов. Во время полета летательный аппарат (ЛА) генерирует вихревой спутный след [366], который со временем эволюционирует и конечном итоге разрушается. Этот след может представлять опасность для других участников воздушного движения [367]. Вихревой след может быть описан как совокупность окрестностей отрезков траектории движения. Для определения конфликта требуется решить задачу наличия пересечения траектории движения собственного ЛА в виде полупрямой с множеством окрестностей отрезков. При этом наличие хотя бы одного конфликта является редкой исключительной ситуацией. Если рассматривать поставленную задачу в виде плоского цикла, то вероятной ветвью исполнения в нем будет анализ на наличие пересечения, а маловероятной – нахождения самих точек пересечения и принятие решения об избегании конфликта [275, 368].

Вернемся к задаче пересечения траектории собственного ЛА с окрестностью отрезка движения другого ЛА из раздела 2.5.2. Рассмотрим неравенство (2.34) и случай $k_2 < 0$, или более подробно $k_2 = (\Delta\overline{C}, \overline{V})^2 + |\overline{V}|^2 \left(\Delta R^2 - |\Delta\overline{C}|^2\right) < 0$. Раскрыв скалярное произведение векторов и выполнив необходимые преобразования, получим условие на угол между траекторией движения собственного ЛА и отрезком $AB$: $\left|\sin\angle(\Delta\overline{C}, \overline{V})\right| > \frac{|\Delta R|}{|\Delta\overline{C}|}$, где $\frac{|\Delta R|}{|\Delta\overline{C}|}$ представляет собой синус угла раствора окрестности отрезка $AB$. Заметим, что угол раствора всегда очень мал, так как характеристики ЛА меняются медленно во время движения, и $\Delta R$ близко к нулю. Таким образом, случай $k_2 < 0$ выполняется в подавляющем большинстве случаев.

Но даже в случае $k_2 < 0$ множество решений неравенства (2.34) на отрезке $[0, 1]$ в подавляющем числе случаев оказывается пустым. Переписав условие отсутствия решения (2.34) на отрезке $[0, 1]$ при условии $k_2 < 0$ получим практически всегда выполняющееся условие $(k_2 < 0) \wedge (m > 0) \wedge (k_1^2 - k_2 k_0 < m^2)$, где $m = \max(k_1 + k_2, -k_1)$. С использованием этого условия как вероятного для выноса маловероятной ветви исполнения из плоского цикла, был поставлен эксперимент по векторизации программного кода определения конфликтов со спутными следами ЛА. В результате к основному циклу была применена автоматическая векторизация и было достигнуто ускорение в районе 5,1 раз на микропроцессоре Intel Xeon Phi 7290 KNL (таблица 3.2) при использовании вещественных чисел формата FP32 [275].

## 5.5 Векторизация с помощью слияния путей исполнения

Универсальным способом векторизации программного кода с условиями является слияние всех путей исполнения под соответствующими предикатами [369]. Рассмотрим это действие на примере простого условия `cond`, по результату которого выполняется переход на один из блоков `block A` и `block B`. Пусть известны вероятности переходов на эти блоки – они равны $p$ и $1 - p$ соответственно. Длины рассматриваемых блоков нормируем так, чтобы в сумме они давали единицу, а отношение их длин задавалось параметром $\alpha$, то есть они равны $\frac{\alpha}{\alpha+1}$ и $\frac{1}{\alpha+1}$ соответственно (см. рис. 5.15 слева). Условимся считать, что длина блока и время его исполнения это по сути одно и то же.

Согласно схеме на рис. 5.15 слева математическое ожидание времени исполнения рассматриваемых блоков в зависимости от условия, будет равно

$$L = p\,\frac{\alpha}{\alpha+1} + (1-p)\,\frac{1}{\alpha+1} = p\left(\frac{\alpha-1}{\alpha+1}\right) + \left(\frac{1}{\alpha+1}\right). \qquad (5.3)$$

Время выполнения $w$ таких участков кода в невекторизованном виде будет равно $wL$. При векторизации кода необходимо избавиться от операций перехода, вместо этого все операции блоков `block A` и `block B` должны быть поставлены под предикаты `cond` и `~cond` соответственно. Далее выполняется

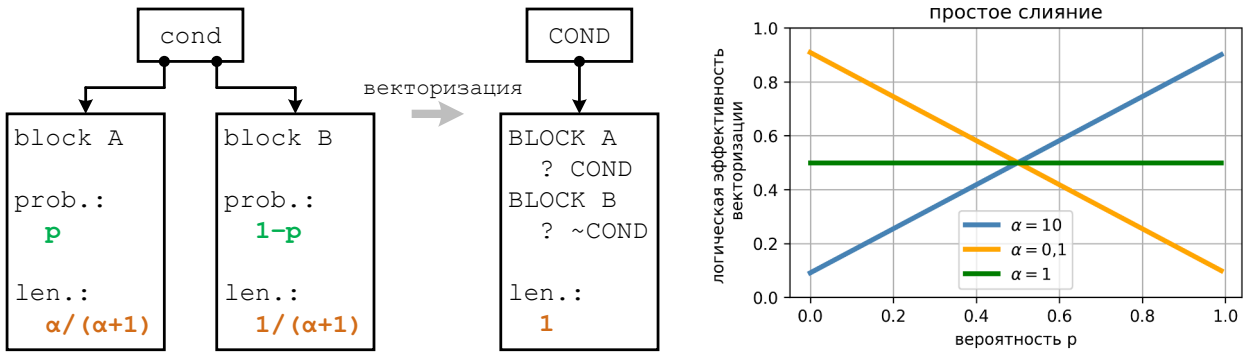


Рис. 5.15. Схема векторизации при простом слиянии блоков (слева) и графики зависимостей логической эффективности векторизации при фиксированных значениях $\alpha$ (справа)

объединение $w$ блоков, при котором скалярные операции под предикатами `cond`, `~cond` заменяются на векторные аналоги, выполняющиеся с использованием векторных масок `COND`, `~COND`. Так как длины блоков выбирались таким образом, чтобы в сумме они давали единицу, то время исполнения векторной версии кода в точности равно $L_v = 1$. То есть логическая эффективность векторизации рассмотренного фрагмента кода совпадает со значением (5.3) и равно

$$e^* = \frac{L}{L_v} = p\left(\frac{\alpha - 1}{\alpha + 1}\right) + \left(\frac{1}{\alpha + 1}\right). \tag{5.4}$$

На рис. 5.15 справа представлены графики зависимостей логической эффективности векторизации из (5.4) для разных значений параметра $\alpha$. Из графиков видно, что при $\alpha = 1$ (то есть при одинаковых длинах блоков `block A` и `block B`) логическая эффективность векторизации постоянна и равно $0,5$. В тех же случаях, когда длины блоков отличаются, логическая эффективность векторизации возрастает, если вероятность перехода на более длинный блок выше, чем на более короткий блок. В любом случае, такой подход прямого слияния ветвей исполнения под соответствующими предикатами в единый линейный участок является крайне неэффективным. При возрастании количества вложенных условий логическая эффективность векторизации с помощью слияния путей исполнения падает экспоненциально. Это связано с появлением в программном коде большого количества векторных инструкций с практически пустыми масками. Для повышения эффективности

векторизации контекста с условиями требуется рассмотрение других подходов, позволяющих повысить плотность масок.

Рассмотрим модификацию слияния путей исполнения, добавив проверку на пустоту векторных масок `COND` и `~COND`, под которыми исполняются векторизованные блоки `BLOCK A` и `BLOCK B` [369] (см. рис. 5.16 слева). Будем считать, что временем проверки маски на пустоту можно пренебречь по сравнению с временем выполнения блока.

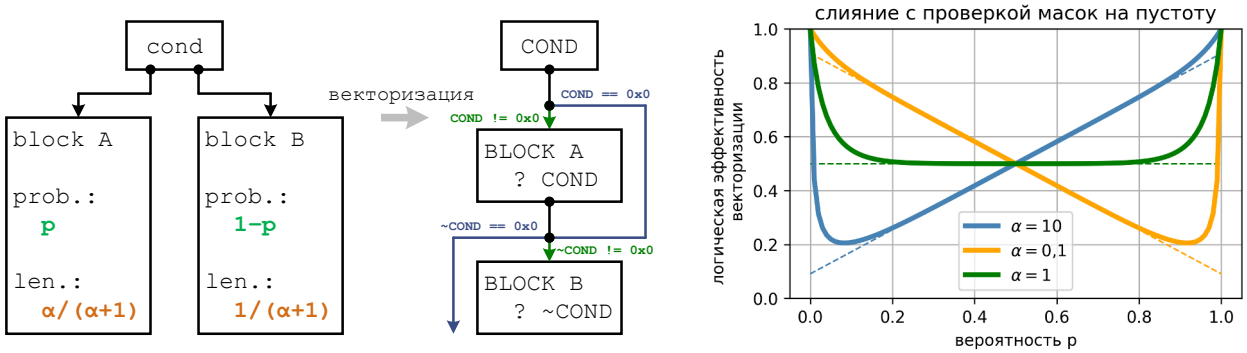


Рис. 5.16. Схема векторизации при слиянии блоков с проверкой масок на пустоту (слева) и графики зависимостей логической эффективности векторизации при фиксированных значениях $\alpha$ (справа)

Если векторная маска, под которой должен быть исполнен блок, пуста, то выполнять команды этого блока нет необходимости, поэтому проверка векторных масок на пустоту может повысить производительность векторного кода. Использование проверок векторных масок на пустоту оправдано, если вероятность появления пустых масок достаточно высока, а также исполняемый блок не является коротким (в этом случае накладные расходы на лишнюю операцию сравнения и возможный переход нивелируют потенциальную пользу от применяемой оптимизации).

Скорректируем выражение для логической эффективности векторизации из (5.4) с учетом проверок векторных масок на пустоту. Величина $L$ остается той же, что и в (5.3), а вот $L_v$ несколько изменится. Если считать, что в каждом наборе обрабатываемых данных переход на тот или иной блок является случайной величиной (хотя в реальных приложениях это не так), то вероятность пустой маски `COND` будет равна $(1-p)^w$, тогда как вероятность

пустой маски `~COND` равна $p^w$. Таким образом, длина векторизованного кода может быть выражена как

$$L_v = (1 - (1-p)^w) \left( \frac{\alpha}{\alpha + 1} \right) + (1 - p^w) \left( \frac{1}{\alpha + 1} \right), \tag{5.5}$$

а логическая эффективность векторизации примет следующий вид:

$$e^* = \frac{p(\alpha - 1) + 1}{(1 - (1-p)^w)\,\alpha + (1 - p^w)}. \tag{5.6}$$

На рис. 5.16 справа представлены зависимости логической эффективности векторизации (5.6) при разных значениях параметра $\alpha$ с учетом проверок масок на пустоту для ширины векторизации $w = 16$, что соответствует использованию данных FP32 в 512-битных регистрах.

Из рис. 5.16 справа видно, что логическая эффективность векторизации возрастает, если значение вероятности перехода на один из блоков близко к единице, однако в среднем ускорение остается невысоким.

Отметим, что условия перехода `cond` с соседних итераций плоского цикла в расчетных приложениях не являются независимыми друг от друга, что зачастую приводит к появлению полных и пустых векторных масок в векторном коде, что делает оправданным применение проверок масок на пустоту. Этот аспект более детально рассматривается в следующих разделах.

## 5.6 Объединение и комбинирование векторных масок

При векторизации плоского цикла $w$ соседних скалярных итераций объединяются в один векторный блок, то есть $n$ итераций плоского цикла трансформируются в $\lfloor \frac{n}{w} \rfloor$ векторных блоков без учета эпилога. Слияние путей исполнения по условию порождает векторные блоки под векторными масками, то есть образуются последовательности одинаковых векторных блоков под разными векторными масками. В разделе рассматривается повышение плотности масок векторного кода с помощью объединения и комбинирования масок соседних векторных блоков для повышение эффективности векторизации.

В результате слияния ветвей исполнения внутри тела плоского цикла, расщепления плоского цикла по условию и векторизации мы получаем совокупность векторных блоков, обрабатывающихся сходим образом: загрузка входных данных `in_data` под маской векторного блока, выполнение вычислений `block` под маской блока, сохранение результатов `out_data` под маской блока (рис. 5.17). На этой схеме `in_data` и `out_data` могут являться как одиночными векторами, так и наборами векторов.

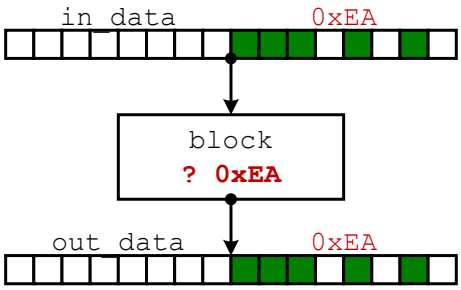


Рис. 5.17. Схема вычислений векторного блока с входными данными `in_data`, выходными данными `out_data` и маской исполнения `0xEA`

Проверка маски блока на пустоту может ускорить код, если маски часто оказываются пустыми. Однако это не поможет в том случае, если в маске выставлено несколько битов. В некоторых случаях достичь ускорения можно путем одновременного выполнения двух соседних векторных блоков [369].

Если в результате векторизации появились два соседних векторных блока `block(in_data_1)` $\rightarrow$ `out_data_1` и `block(in_data_2)` $\rightarrow$ `out_data_2`, которые должны выполняться под разными векторными масками `mask_1` и `mask_2`, и в дополнение к этому для этих масок выполнено условие `(mask_1 & mask_2) == 0x0` (то есть маски не пересекаются), то вычисление этих двух соседних блоков можно объединить. Вместо последовательного выполнения двух векторных блоков можно объединить их входные данные `in_data_1` и `in_data_2` с помощью слияния `in_data = _mm512_mask_blend_ps(mask_1, in_data_2, in_data_1)`, после чего выполнить тот же блок вычислений под объединенной маской `mask_1 | mask_2`. Ввиду отсутствия пересечения векторных масок в результирующих выходных данных `out_data` будут содержаться как необхо-

димые элементы данных `out_data_1`, так и необходимые элементы данных `out_data_2`. В конце остается извлечь из объединенного результата `out_data` данные `out_data_1` и `out_data_2` (рис. 5.18).

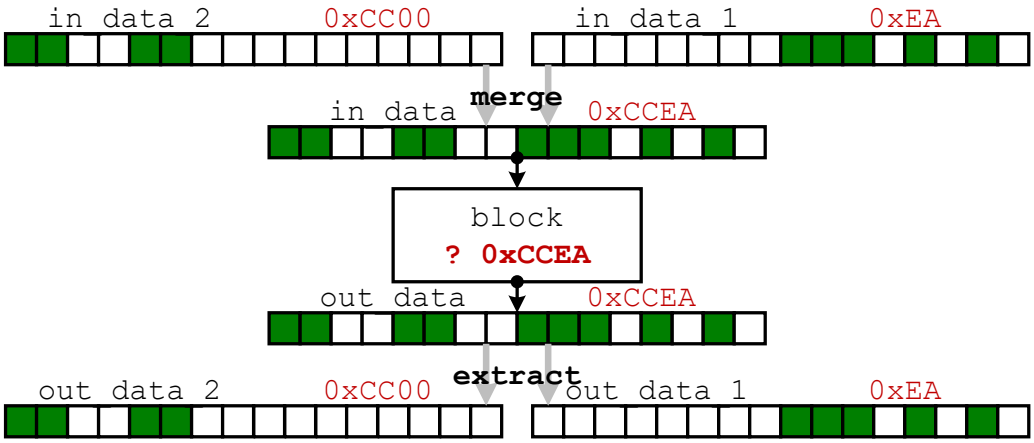


Рис. 5.18. Схема одновременного выполнения двух одинаковых векторных блоков при условии непересечения их масок

В результате такого преобразования в случае отсутствия пересечения векторных масок количество вычислений рассматриваемого блока `block` сокращается вдвое, а плотность векторных масок внутри блока повышается. Однако вместе с этим появляются накладные расходы, связанные с проверками масок, а также операции слияния данных до вычислений блока и выделения нужных данных после вычислений. Заметим, что эту технику можно применять для объединения трех и более соседних блоков, однако это связано с еще большим возрастанием накладных расходов.

Еще один подход, о котором стоит упомянуть, анализ эффективности которого проверятся только с теоретической точки зрения, связан с объединением соседних блоков с пересекающимися масками, то есть для которых `(mask_1 & mask_2) != 0x0`.

Если мы имеем дело с двумя масками низкой плотности, которые пересекаются, но для которых выполнено условие непревышения суммарной плотности ширины векторизации `popcnt(mask_1) + popcnt(mask_2) <= w`, то такие блоки также можно объединить. Для этого перед объединением необходимо применять преобразование одной или обеих масок.

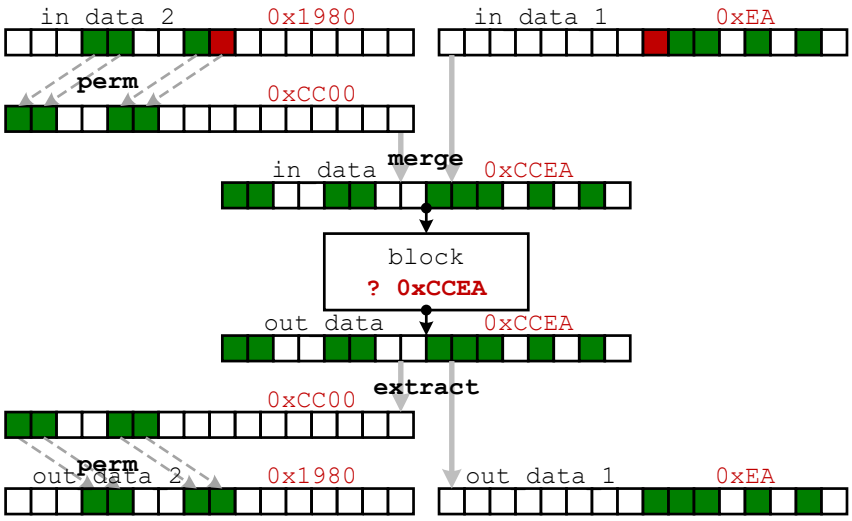


Рис. 5.19. Схема одновременного выполнения двух одинаковых векторных блоков при условии пересечения их масок

**Определение 5.8.** *Преобразование* `perm_to` *векторной маски* `mask` *называется обратимым, если существует преобразование* `perm_from` *такое, что* `perm_from(perm_to(mask)) = mask`.

Для выполнения комбинирования векторных масок двух соседних блоков необходимо найти обратимое преобразование одной из масок (например, `mask_1`) `perm_to` такое, что `(perm_to(mask_1) & mask_2) == 0x0`. В этом случае элементы входных данных переставляются местами в соответствии с преобразованием `perm_to`, применяется объединение блоков, а для выходных данных выполняется перестановка в соответствии с преобразованием `perm_from` (рис. 5.19). Объединять можно не только два соседние блока, но и более, но это увеличивает накладные расходы. Также преобразование исходных масок можно применять сразу к обеим маскам.

Опишем эксперимент по применению объединения векторных масок на примере функции `prefun` из точного римановского газодинамического решателя (листинг 5.9). Функция содержит одно условие и две ветви исполнения с достаточно тяжелыми вычислениями, что делает оправданным применение проверки масок на пустоту. Операции в реализации функции имеют векторные аналоги, она может быть векторизована путем замены скалярных операций векторными аналогами и слияния ветвей исполнения.

Листинг 5.9. Скалярная версия функции `prefun`

```
void scase_prefun_1(float& f, float& fd,
                    float p, float dk, float pk, float ck)
{
    if (p <= pk)
    {
        float prat = p / pk;
        f = riemann::sg4 * ck * (pow(prat, riemann::sg1) - 1.0f);
        fd = (1.0f / (dk * ck)) * pow(prat, -riemann::sg2);
    }
    else
    {
        float ak = riemann::sg5 / dk;
        float bk = riemann::sg6 * pk;
        float qrt = sqrt(ak / (bk + p));
        f = (p - pk) * qrt;
        fd = (1.0f - 0.5f * (p - pk) / (bk + p)) * qrt;
    }
}
```

В разделе 5.5 вычислялась вероятность пустой маски, и использовалось предположение, что выполнение условий для разных наборов скалярных данных являются независимыми. На самом деле это не так, и зависит от локальности размещения данных, участвующих в расчетах [370]. Рассмотрим условие `p <= pk`. Элементы данных `p` и `pk` свои для каждой расчетной ячейки. Если речь идет о физических расчетах, то значение элемента данных изменяется медленно при переходе от одной ячейки к соседней (рис. 5.20).

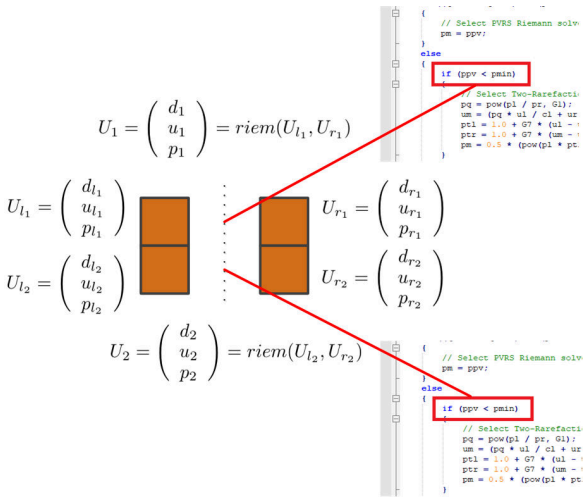


Рис. 5.20. Иллюстрация тенденции сохранения значения условия при переходе к соседним ячейками в расчетных задачах

Значение условия `p <= pk` при переходе от одной ячейки к соседней будет изменяться также медленно. Но условие это дискретная величина, а это зна-

чит, что часто значение условия будет сохраняться при переходе к соседней ячейке. Для функции `prefun` были собраны расчетные данные распределения плотности маски условия `p <= pk`, чтобы оценить вероятность появления пустых масок `cond` и `ncond`. На рис. 5.21 слева представлено распределение плотности масок в случае независимости условий для разных наборов скалярных данных. Результаты распределения плотности маски `cond`, собранные по настоящему профилю исполнения векторного кода на реальных данных, представлены на рис. 5.21 справа.

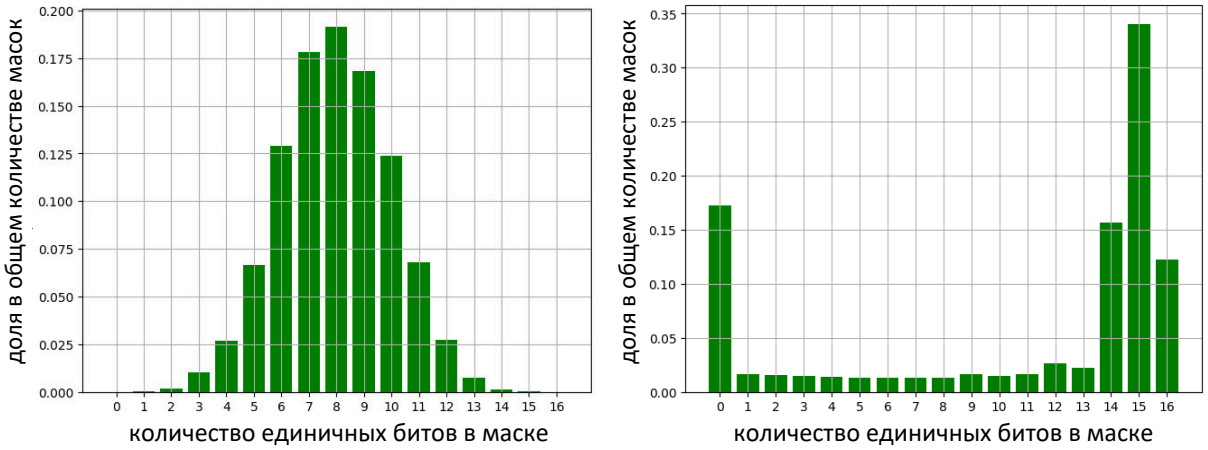


Рис. 5.21. Гистограмма распределения количества единичных битов маски `cond` при условии, что все условия `p <= pk` для наборов скалярных данных независимы (слева) и на реальном профиле исполнения (справа)

Из рис. 5.21 видно, что распределение плотностей масок на реальных данных радикально отличается от распределения, вычисленного в предположении о независимости условий переходов. Можно заметить, что в реальном коде более четверти всех масок `cond` являются либо пустыми, либо полными (в этом случае пустой является маска `ncond`), а значит использование проверок масок на пустоту обосновано.

Для анализа полученных результатов были рассмотрены следующие три подхода к векторизации плоского цикла с условием. В качестве базового метода векторизация принималось простое слияние путей исполнения под соответствующими предикатами с последующим объединением $w$ последовательных скалярных итераций в одну векторную (*простое слияние*). Этот

базовый метод сравнивался с двумя рассмотренными выше улучшениями: проверка векторных масок на пустоту (*проверка масок*) и слияние двух соседних блоков при условии отсутствия пересечения их масок (*объединение масок*). Анализ эффективности применения преобразований рассматривался для функции `prefun` из реализации газодинамического римановского решателя. Профиль исполнения функции собирался на задачах моделирования распада разрыва при различных начальных условиях [371]. Эффективность векторизации при выбранных подходах измерялась на микропроцессоре Intel Xeon Phi 7290 KNL. Результаты сравнения представлены на рис. 5.22.

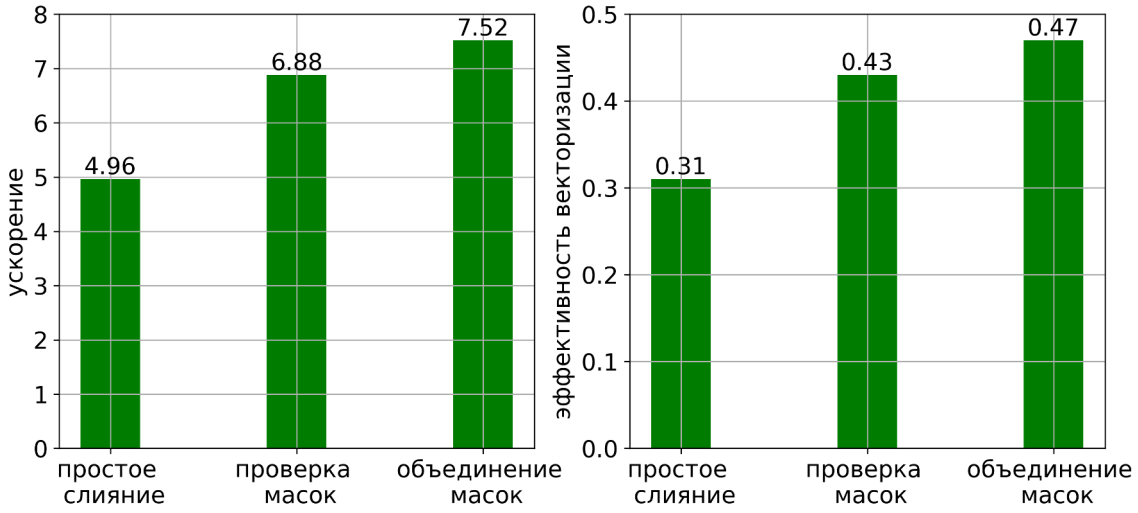


Рис. 5.22. Ускорение и эффективность векторизации при простом слиянии, с проверкой масок и с объединением масок на микропроцессоре Intel Xeon Phi 7290 KNL

Эксперимент показал, что слияние путей исполнения позволило достичь эффективности векторизации 0,31, а использование проверок масок и объединения масок позволило повысить ее до 0,43 и 0,47 соответственно.

## 5.7 Векторизация гнезд циклов

В расчетных приложениях часто встречается код, в котором тело плоского цикла содержит другие циклы или гнезда циклов. В разделе проводится анализ программного контекста, в котором тело плоского цикла содержит другой цикл, такую конструкцию будем называть «плоский цикл / внутренний цикл» (рис. 5.23).

При выполнении векторизации тело внутреннего цикла *block*, выполняемое по условию *cond*, переводится в векторный блок $BLOCK$, инструкции которого выполняются под маской $COND$, которая постепенно истощается от итерации к итерации пока не станет пустой. При этом выполняется соотношение $I_v = \max_{i=0}^{w-1} I(i)$, где $I(i)$ – количество итераций внутреннего цикла на $i$-ой итерации плоского цикла в скалярной версии, $I_v$ – количество итераций внутреннего цикла в векторной версии.


Рис. 5.23. Схема векторизации структуры «плоский цикл / внутренний цикл»

Ускорение от векторизации рассматриваемого программного кода зависит от характера изменения условия *cond* (и соответственно количества итераций внутреннего цикла $I(i)$) при переходе между итерациями плоского цикла. Слишком резкое изменения условия *cond* между итерациями плоского цикла приводит к сильному изменению значений $I(i)$, что приводит возрастанию количества итераций векторизованного цикла, выполняемых под неполной маской $COND$. Это приводит к деградации производительности при векторизации структуры «плоский цикл / внутренний цикл».

Обозначим ускорение и эффективность векторного кода рассматриваемого гнезда в терминах количества итераций внутреннего цикла естественным образом как $s^I = \frac{\sum_{i=0}^{w-1} I(i)}{I_v}$, $e^I = \frac{s^I}{w}$. Обозначим логическое ускорение и логическую эффективность векторизации итерации внутреннего цикла через $s^*_{iter}$ и $e^*_{iter}$ соответственно. В приведенных обозначениях справедлива следующая лемма:

**Лемма 5.1.** *При векторизации структуры «плоский цикл / внутренний цикл» верно соотношение*

$$e^* = e^I e^*_{iter}. \tag{5.7}$$

Пусть $\lambda$ – количество операций на одной итерации внутреннего цикла в скалярной версии, а $\lambda_v$ – количество операций на одной итерации внутреннего цикла в векторной версии. Очевидно, что $s^*_{iter} = \frac{w\lambda}{\lambda_v}$ и $e^*_{iter} = \frac{\lambda}{\lambda_v}$. Тогда

$$e^* = \frac{L}{wL_v} = \frac{\sum_{i=0}^{w-1}\sum_{j=0}^{I(i)-1}\lambda}{w\sum_{j=0}^{I_v}\lambda_v} = \frac{\sum_{i=0}^{w-1} I(i)}{wI_v}\frac{\lambda}{\lambda_v} = e^I e^*_{iter}. \blacksquare \tag{5.8}$$

Введем характеристику $\varepsilon = \max_{i=0}^{w-1} I(i) - \min_{i=0}^{w-1} I(i)$, которую будем называть коэффициентом неравномерности количества итераций внутреннего цикла. В приведенных обозначениях справедлива следующая лемма:

**Лемма 5.2.** *При векторизации структуры «плоский цикл / внутренний цикл» верна оценка*

$$e^I \geq 1 - \frac{\varepsilon}{I_v}. \tag{5.9}$$

Так как $\forall i \in [0, w-1] \implies I(i) \geq I_v - \varepsilon$, то

$$e^I = \frac{\sum_{i=0}^{w-1} I(i)}{wI_v} \geq \frac{w(I_v - \varepsilon)}{wI_v} = 1 - \frac{\varepsilon}{I_v}. \blacksquare \tag{5.10}$$

Значение характеристики $\varepsilon$ существенным образом влияет на эффективность векторизации. Если $\varepsilon = 0$, то $I(i)$ является константной, в этом случае будем говорить о внутреннем цикле с постоянным количеством итераций. Если $\varepsilon$ мало, то количество итераций внутреннего цикла меняется не сильно, будем говорить о внутреннем цикле с непостоянным количеством итераций. В случае же $\varepsilon$ сравнимым с $I_v$ количество итераций внутреннего цикла меняется сильно, будем говорить о внутреннем цикле с нерегулярным количеством итераций. Далее в разделе приводятся примеры векторизации плоских циклов, внутренние циклы в телах которых представляют собой циклы различного вида по характеристике $\varepsilon$.

### 5.7.1 Внутренний цикл с постоянным количеством итераций

Рассмотрим программный контекст, в котором внутренний цикл, находящийся в теле плоского цикла, выполняется с постоянным количеством итераций, то есть $\varepsilon = 0$ и $e^I = 1$.

Листинг 5.10. Определение пересечения треугольника и прямоугольного параллелепипеда с помощью свертывания системы линейных неравенств

```
for (i = 0; i < bec; ++i)
{
    gi0 = g[i][0];

    if (gi0 == 0.0)
    {
        if (!upgrade(g[i][1], g[i][2], &lo, &hi)) return 0;
    }
    else
    {
        for (j = i + 1; j < bec; ++j)
        {
            if (gi0 * g[j][0] < 0.0)
            {
                f0 = gi0 * g[j][1] - g[j][0] * g[i][1];
                f1 = gi0 * g[j][2] - g[j][0] * g[i][2];

                if (gi0 < 0.0)
                {
                    f0 = -f0;
                    f1 = -f1;
                }

                if (!upgrade(f0, f1, &lo, &hi)) return 0;
            }
        }
    }
}

return 1;
```

В разделе 2.9.2 приведен метод погруженной границы для выполнения газодинамических расчетов вокруг тела со сложной геометрией. На первом этапе его реализации выполняется определение пересечения ячеек неструктурированной поверхностной расчетной сетки с ячейками объемной декартовой сетки, для чего многократно решается задача обнаружения пересечения треугольника и прямоугольного параллелепипеда в пространстве из раздела 2.9.1 (листинг 5.10) с помощью свертывания системы линейных неравенств [297].

Код с листинга 5.10 возвращает 1 при наличии пересечения. Коэффициенты системы неравенств (2.72) заданы в массиве `g` размера $9 \times 3$. Допустимые значения переменной $\gamma$ представлены отрезком `[lo, hi]`, который изменяется внутри функции `upgrade` в процессе выполнения одного шага свертывания. Возникновение условия `lo > hi` означает отсутствие пересечения.

Получившийся программный код можно охарактеризовать как имеющий сложное управление с уровнем вложенности 5, несколькими выходами и вызовом функции. Для векторизации $w$ экземпляров этого кода можно объединить в плоский цикл, рассматривая коэффициенты $w$ систем линейных неравенств, заданные в массиве размера $9 \times 3 \times w$.

Для эффективной векторизации необходимо избавляться от избыточных операций передачи управления. Так как набор инструкций AVX-512 включает в себя векторизованные операции, реализующиеся через условия (операции abs, min, max, blend), то использование математических тождеств в некоторых случаях позволяет существенно упростить код. Например, используя тождество $\forall a, b, c, d \in \mathbb{R} : ab < 0 \implies (a \geq 0)\ ?\ (ad - bc) : (bc - ad) = |a|d + |b|c$, вычисление значений `f0` и `f1` из строк 13-22 листинга 5.10 можно заменить на представленное на листинге 5.11.

Листинг 5.11. Использование тождества для векторизации условия

```
1 f0 = fabs(gi0) * g[j][1] + fabs(g[j][0]) * g[i][1]
2 f1 = fabs(gi0) * g[j][2] + fabs(g[j][0]) * g[i][2]
```

Также цикл с листинга 5.10 можно расщепить по условию `gi0 == 0.0` со строки 5, получив отдельные цикл и гнездо, которые могут быть векторизованы независимо друг от друга. Все внутренние циклы содержат фиксированное количество итераций (условие выхода – достижение индуктивной переменной значения `bec`), таким образом, условия выхода из внутренних циклов не требуется векторизовать, они переносятся в векторный код в неизменном виде (общая схема векторизации представлена на рис. 5.24).

Из рис. 5.24 видно, что рассматриваемый программный контекст достаточно просто переводится из скалярного вида в векторный. Это происходит

скалярный код

```
float g[bec][3][w];
<инициализация g>

for (wi = 0; wi < w; ++wi) r[wi] = 1;

for (wi = 0; wi < w; ++wi)
{
    lo = 0.0;
    hi = 1.0;

    for (i = 0; i < bec; ++i)
    {
        upgrade(g[i][0][wi] == 0.0,
                g[i][1][wi], g[i][2][wi], &lo, &hi);
        if (lo > hi) break;
    }

    for (i = 0; i < bec; ++i)
    {
        gi0 = g[i][0][wi];
        agi0 = fabs(gi0);

        for (j = i + 1; j < bec; ++j)
        {
            gj0 = g[j][0][wi];
            agj0 = fabs(gj0);

            upgrade(gi0 * gj0 < 0.0,
                    agi0 * g[j][1][wi] + agj0 * g[i][1][wi],
                    agi0 * g[j][2][wi] + agj0 * g[i][2][wi],
                    &lo, &hi);
            if (lo > hi) break;
        }

        if (lo > hi) break;
    }

    if (lo > hi) r[wi] = 0;
}
```

векторный код

```
__m512 g[bec][3];
<инициализация g>

_mm512_store_epi32(r, _mm512_set1_epi32(1));


__m512 lo = z0;
__m512 hi = z1;

for (i = 0; i < bec; i++)
{
    upgrade(_mm512_cmpeq_ps_mask(g[i][0], z0),
            g[i][1], g[i][2], &lo, &hi);
    if (!_mm512_cmplt_ps_mask(lo, hi)) break;
}

for (i = 0; i < bec; i++)
{
    gi0 = g[i][0];
    agi0 = ABS(gi0);

    for (j = i + 1; j < bec; j++)
    {
        gj0 = g[j][0];
        agj0 = ABS(gj0);

        upgrade(_mm512_cmplt_ps_mask(MUL(gi0, gj0), z0),
                FMADD(agi0, g[j][1], MUL(agj0, g[i][1])),
                FMADD(agi0, g[j][2], MUL(agj0, g[i][2])),
                &lo, &hi);
        if (!_mm512_cmplt_ps_mask(lo, hi)) break;
    }

    if (!_mm512_cmplt_ps_mask(lo, hi)) break;
}

_mm512_mask_store_epi32(r,
                        _mm512_cmplt_ps_mask(hi, lo),
                        _mm512_set1_epi32(0));
```

Рис. 5.24. Схема векторизации $w$ экземпляров кода свертывания системы линейных неравенств

по причине следующих факторов: отсутствие избыточных условий, а также независимость условий выхода из внутренних циклов от номера итерации векторизуемого плоского цикла. Также стоит обратить внимание на реализацию условного вызова функции `upgrade`, который заменяется на безусловный вызов с передачей условия внутрь нее для последующей векторизации. В результате выполненных преобразований был получен эффективный векторный код для поиска пересечений поверхностной и объемной расчетных сеток. Функция определения пересечения пар ячеек продемонстрировала ускорение в 6,7 раза по сравнению с невекторизованной версией на микропроцессоре Intel Xeon Phi 7290 KNL (таблица 3.2) [372].

На рассмотренном примере векторизации программного контекста, содержащего гнездо циклов, в котором внутренний цикл имеет постоянное количество итераций, применялось два важных преобразования, которые имеют критическое значение, так как направлены на преодоление проблемы векторизации кода, содержащего команды передачи управления.

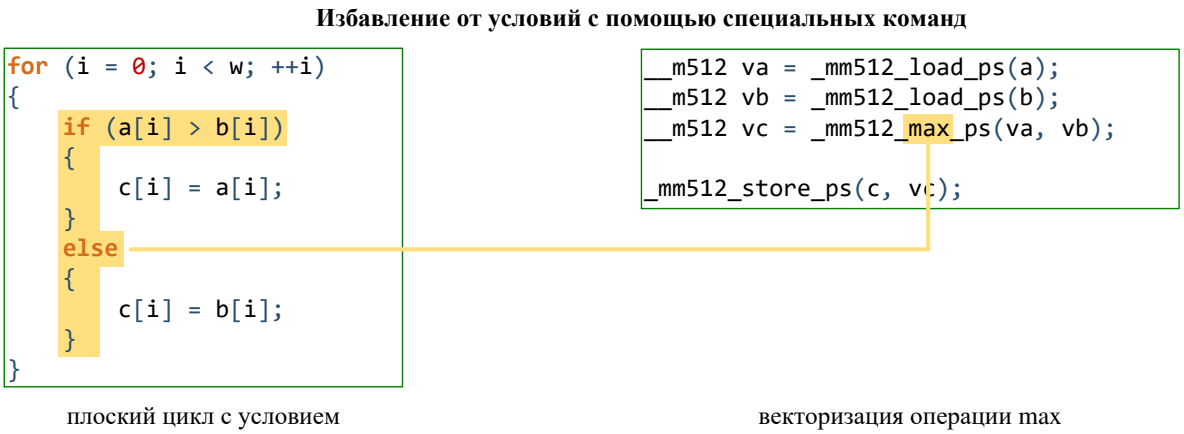


Рис. 5.25. Избавление от условий с помощью специальных операций

Первое из этих преобразований – применение математических тождеств с использованием специализированных векторных инструкций, содержащих внутри себя логику [373]. Основные математические действия, имеющие векторные аналоги в наборе инструкций AVX-512 это получение модуля числа ($|x|$), минимального и максимального значения, выбор значения по условию ($p\ ?\ x : y$). Запись выражений с помощью тождеств с использованием указанных операций приводит к удалению лишних операций передачи управления в теле цикла и повышает эффективность векторизации (рис. 5.25).

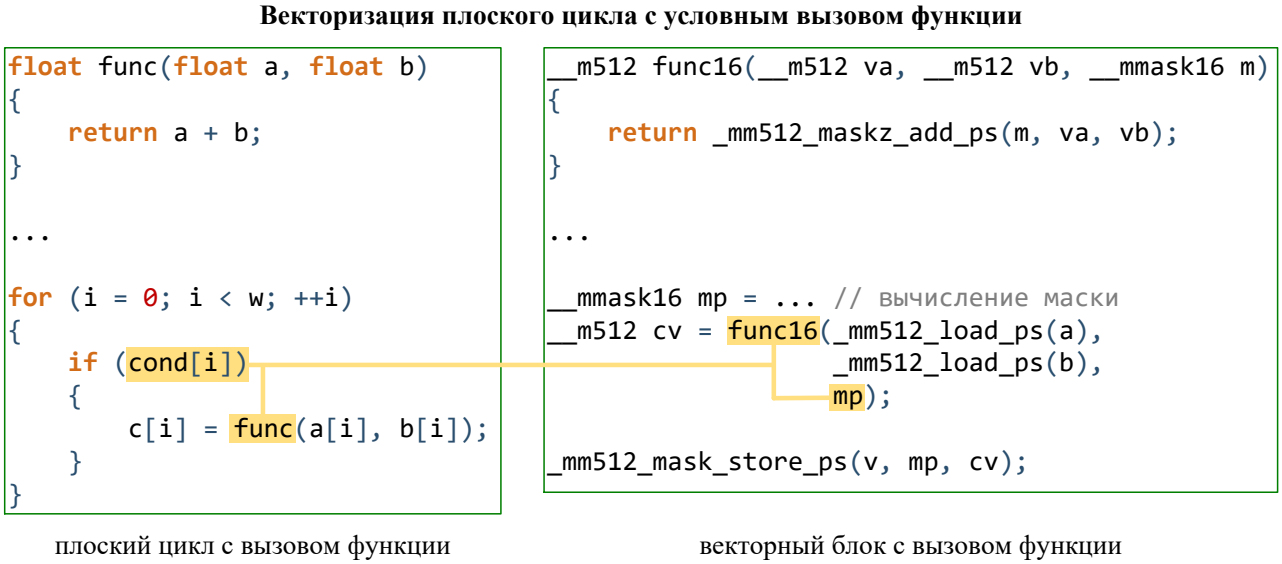


Рис. 5.26. Схема векторизации с вызовом функции под условием

Второе важное преобразование – векторизация плоского цикла, в теле которого находится вызов функции под условием. Для векторизации условие, под

котором выполняется вызов функции, передается в эту функцию в качестве дополнительного параметра и используется внутри нее для выполнения тела функции. При этом сам вызов функции становится безусловным, что является деградацией, однако она будет компенсирована за счет векторизации. При векторизации это условие трансформируется в векторную маску, под которой выполняются все инструкции векторизованного тела функции (рис. 5.26).

### 5.7.2 Внутренний цикл с непостоянным количеством итераций

Если количество итераций внутреннего цикла не является постоянным, и условия выхода из внутреннего цикла зависят от номера итерации плоского цикла, то такие условия необходимо переводить в векторную форму с использованием векторных предикатов. Рассмотрим пример программного контекста, в котором количество итераций внутреннего цикла является непостоянным, то есть $\varepsilon \neq 0$. Будем говорить о непостоянном количестве итераций внутреннего цикла, если значение характеристики $\varepsilon$ достаточно мало, то есть значение $e^I$ близко к единице. Такое поведение характерно для расчетных кодов компьютерного моделирования, в частности газодинамических решателей. В [374, 375] рассмотрена векторизация точного римановского решателя, где $w$ экземпляров вызовов скалярного решателя $U = [d, u, p] = riem(U_l, U_r)$ ($U_l = [d_l, u_l, p_l]$, $U_r = [d_r, u_r, p_r]$ – параметры с двух сторон от разрыва) заменяется одним вызовом векторного решателя $\overline{U} = [\overline{d}, \overline{u}, \overline{p}] = riem(\overline{U}_l, \overline{U}_r)$, представленного в виде плоского цикла.

На рис. 5.27 приведена схема потока данных в римановском газодинамическом решателе. Этот решатель включает в себя функции с достаточно простым управлением – `guessp`, `prefun`, а также функции с циклами и с обилием вложенных условий – `starpu`, `sample`. Наибольший интерес представляет функция `starpu`, содержащая цикл с неизвестным количеством итераций (листинг 5.12). Цикл, расположенный в строках 13-24 листинга 5.12, кроме неизвестного количества итераций содержит также операторы передачи управления и вызовы функций, что также усложняет его векторизацию.

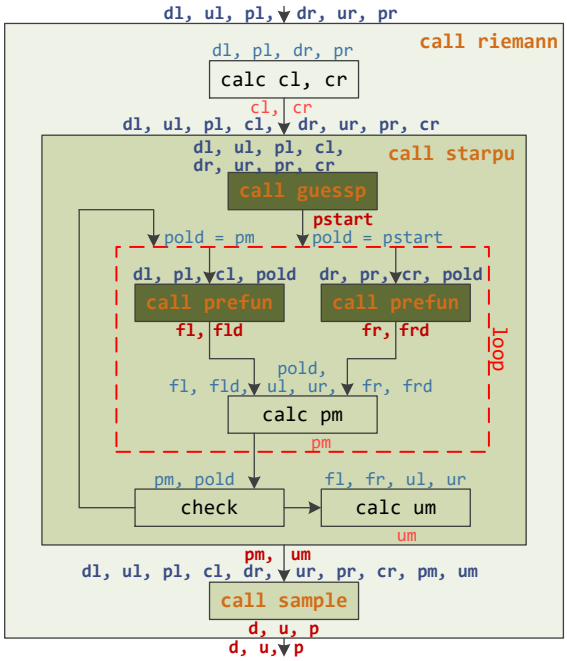


Рис. 5.27. Схема потока данных в римановском решателе

Листинг 5.12. Скалярная версия функции `starpu`

```
void starpu(float dl, float ul, float pl, float cl,
            float dr, float ur, float pr, float cr, float &p, float &u)
{
    const int nriter = 20;
    const float tolpre = 1.0e-6;
    float change, fl, fld, fr, frd, pold, pstart, udiff;

    guessp(dl, ul, pl, cl, dr, ur, pr, cr, pstart);
    pold = pstart;
    udiff = ur - ul;

    int i = 1;
    for ( ; i <= nriter; i++)
    {
        prefun(fl, fld, pold, dl, pl, cl);
        prefun(fr, frd, pold, dr, pr, cr);
        p = pold - (fl + fr + udiff) / (fld + frd);
        change = 2.0 * abs((p - pold) / (p + pold));

        if (change <= tolpre) break;
        if (p < 0.0) p = tolpre;

        pold = p;
    }

    if (i > nriter)
    {
        cout << "divergence in Newton-Raphson iteration" << endl;
        exit(1);
    }

    u = 0.5 * (ul + ur + fr - fl);
}
```

Перед выполнением векторизации этот цикл необходимо преобразовать в предикатную форму, в которой тело не должно содержать операций передачи управления. Заводится предикат выполнения итерации цикла, под который ставятся все инструкции тела цикла и который передается в вызовы функций `prefun`. Тогда при векторизации предикат, под которым выполняется тело цикла, преобразуется в векторную маску, как это показано на листинге 5.13, и векторизованный цикл прекращает работу либо в случае истощения этой маски, либо в случае превышения допустимого количества итераций (аварийная остановка). Похожий подход к векторизации циклов с неизвестным количеством итераций был применен в [376] на примере расчета фрактала Мандельброта.

Листинг 5.13. Векторизованная версия функции `starpu`

```
void starpu_16(__m512 dl, __m512 ul, __m512 pl, __m512 cl,
               __m512 dr, __m512 ur, __m512 pr, __m512 cr,
               __m512 *p, __m512 *u)
{
    ...
    two = SET1(2.0); tolpre = SET1(1.0e-6); tolpre2 = SET1(5.0e-7);
    udiff = SUB(ur, ul);

    guessp_16(dl, ul, pl, cl, dr, ur, pr, cr, &pold);

    // Start with full mask.
    m = 0xFFFF;
    for (; (iter <= nriter) && (m != 0x0); iter++)
    {
        prefun_16(&fl, &fld, pold, dl, pl, cl, m);
        prefun_16(&fr, &frd, pold, dr, pr, cr, m);
        *p = _mm512_mask_sub_ps(*p, m, pold,
                 _mm512_mask_div_ps(z, m,
                     ADD(ADD(fl, fr), udiff), ADD(fld, frd)));
        change = ABS(_mm512_mask_div_ps(z, m,
                         SUB(*p, pold), ADD(*p, pold)));
        cond_break = _mm512_mask_cmp_ps_mask(m, change,
                         tolpre2, _MM_CMPINT_LE);
        m &= ~cond_break;
        cond_neg = _mm512_mask_cmp_ps_mask(m, *p, z, _MM_CMPINT_LT);
        *p = _mm512_mask_mov_ps(*p, cond_neg, tolpre);
        pold = _mm512_mask_mov_ps(pold, m, *p);
    }
    if (iter > nriter)
    {
        cout << "divergence in Newton-Raphson iteration" << endl;
        exit(1);
    }

    *u = MUL(SET1(0.5), ADD(ADD(ul, ur), SUB(fr, fl)));
}
```

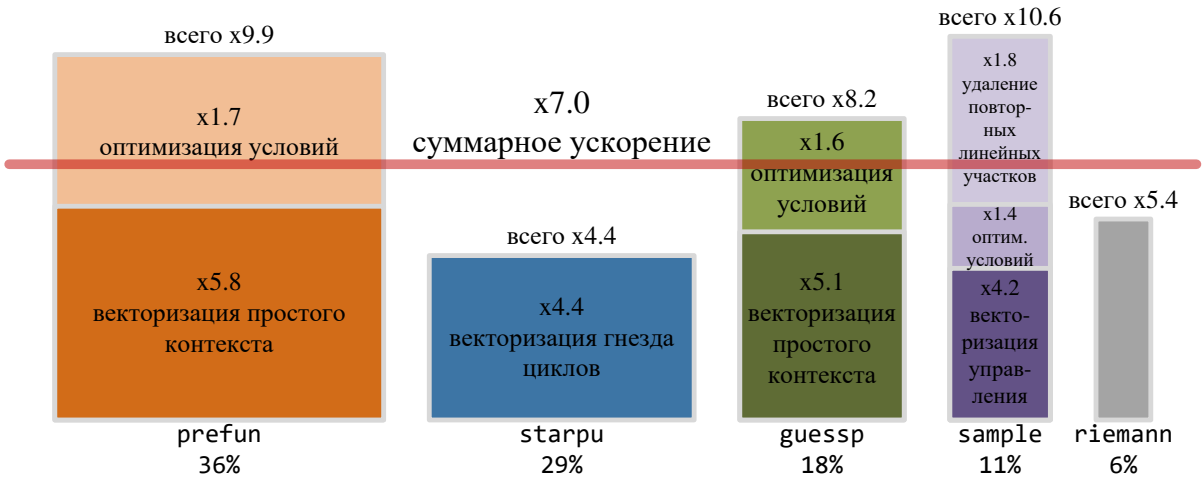


Рис. 5.28. Диаграмма ускорения отдельных функций и суммарного ускорения римановского решателя

На листинге 5.13 в строке 12 видна изначальная инициализация полной маски выполнения векторизованных итераций цикла. По мере работы цикла маска истощается (строка 25), и при полном ее обнулении цикл завершает работу. Эта же маска передается в векторизованную функцию `prefun_16`, в которой все операции должны быть выполнены под этой маской. Эффективность векторизации напрямую связана со скоростью изменения плотности векторной маски `m`. Нетрудно видеть, что величина $\frac{\varepsilon}{I_v}$ соответствует доли количества итераций векторизованного цикла, на которых векторная маска `m` является неполной.

Для замеров эффективности векторизации функций точного римановского решателя на микропроцессоре Intel Xeon Phi 7290 KNL (таблица 3.2) был поставлен численный эксперимент, в рамках которого запускалась скалярная и векторная версии кода на одном и том же наборе входных данных, полученных из прогона стандартных тестов для задачи распада разрыва [377]. На диаграмме рис. 5.28 показан эффект от применения различных оптимизаций к каждой из рассматриваемых функций, а также суммарное ускорение, полученное вследствие векторизации.

Ускорение от векторизации функции `starpu` составило примерно 4,4 раза при использовании чисел FP32, что скромнее результатов, продемонстриро-

ванных при векторизации гнезда циклов с постоянным количеством итераций из раздела 5.7.1. Ускорение от векторизации функции `starpu` оказалось ниже ускорения остальных функций решателя, которые содержат только арифметические вычисления и команды передачи управления. Результат ускорения функции `sample` не является показательным, так как для нее применялись специальные преобразования, связанные с заменой переменных и удалением повторяющихся участков программного кода [365].

Поставлен эксперимент по определению максимальной производительности векторной версии точного римановского решателя на сегментах суперкомпьютера МВС-10П МП2 KNL (таблица 3.2) и МВС-10П ОП2 (таблица 4.1). При этом использовалось распараллеливание с помощью OpenMP на различном количестве потоков. В качестве анализируемого кода выбрано численное решение задачи газовой динамики методом Годунова [340, 341] с использованием точного римановского решателя. Производительность измерялась в миллионах образатываемых экземпляров задачи Римана о распаде разрыва.

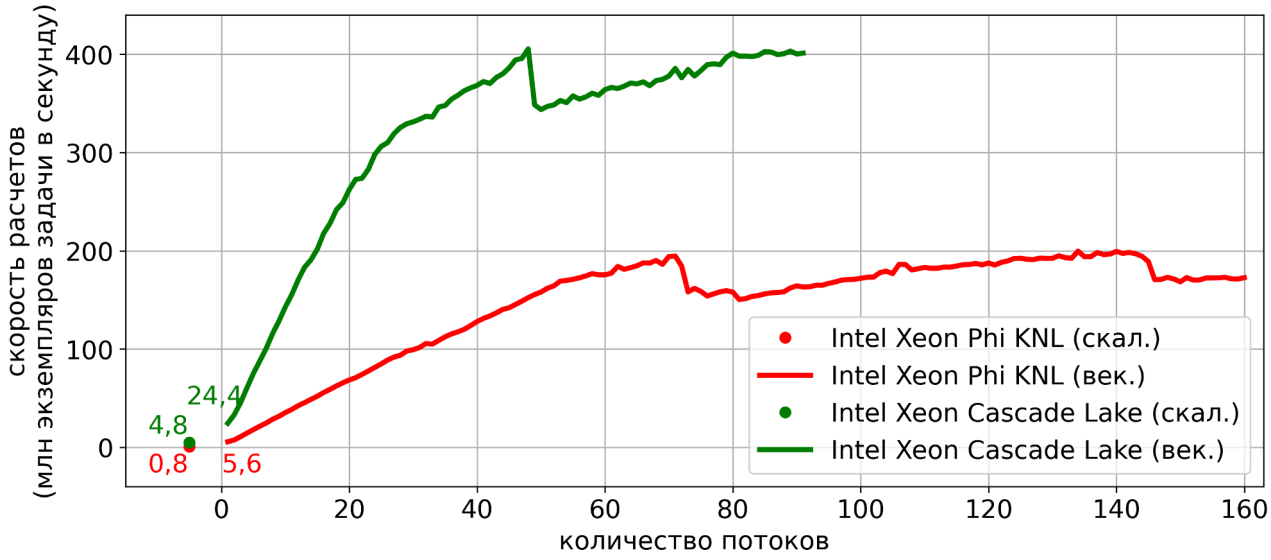


Рис. 5.29. Распараллеливание выполнения нескольких экземпляров решения задачи о распаде разрыва на вычислительных узлах сегментов МВС-10П МП2 KNL и МВС-10П ОП2

На рис. 5.29 представлены графики скорости расчетов для количества потоков до 160. При большем количестве потоков для микропроцессора Intel Xeon Phi 7290 KNL наблюдается деградация, поэтому результаты не приводятся.

### 5.7.3 Внутренний цикл с нерегулярным количеством итераций

В разделе рассматривается программный контекст, в котором на итерациях плоского цикла невозможно предсказать условие выхода из внутреннего цикла, что приводит к возникновению гнезд циклов с нерегулярным количеством итераций [378,379], что характеризуется значением $\varepsilon$, сравнимым с $I_v$, и низким значением $e^I$.

В качестве примера такого программного контекста рассмотрим реализацию сортировки Шелла, представляющую собой расширение сортировки вставками, которое работает быстрее, так как позволяет на ранних этапах упорядочить далеко расположенные друг от друга элементы массива, и это приводит к тому, что массив становится частично упорядоченным. Во время сортировки Шелла выполняется последовательная сортировка подмассивов основного массива, являющихся срезами, при этом шаг среза постоянно уменьшается и на завершающем этапе выполняется обычная сортировка вставками (это соответствует срезу массива с шагом 1). Выполнение сортировки срезов массива с большими шагами облегчает сортировку срезов с меньшими значениями шага, эффективность сортировки существенно зависит от выбранной последовательности шагов. Существует большое количество различных последовательностей шагов, некоторые из них являются эвристическими, примеры последовательностей шагов приведены в таблице 5.3.

Таблица 5.3. Различные последовательности шагов, используемые в сортировке Шелла ($n$ – размер массива)

| Последовательность шагов | Формула |
|---|---|
| Последовательность Шелла | $k_1 = \lfloor \frac{n}{2} \rfloor,\ k_i = \lfloor \frac{k_{i-1}}{2} \rfloor,\ k_t = 1$ |
| Последовательность Хиббарда | $2^i - 1 \leq n,\ i \in \mathbb{N}$ |
| Последовательность Пратта | $2^i \cdot 3^j \leq \frac{n}{2},\ i \in \mathbb{N},\ j \in \mathbb{N}$ |
| Последовательность Седжвика | $k_i = \begin{cases} 9 \cdot 2^i - 9 \cdot 2^{\frac{i}{2}} + 1, & i = 2m,\ m \in \mathbb{Z}_{\geq 0} \\ 8 \cdot 2^i - 6 \cdot 2^{\frac{i+1}{2}}, & i = 2m+1,\ m \in \mathbb{Z}_{\geq 0} \end{cases}$ |

Реализация сортировки Шелла состоит из гнезда циклов, содержащего три цикла. Внешний цикл выполняется по всем шагам из используемой последовательности шагов, начиная с максимального и заканчивая шагом 1. Два внутренних цикла осуществляют сортировку всех подмассивов, являющихся срезами исходного массива с текущим шагом $k$ (листинг 5.14).

Листинг 5.14. Реализация сортировки Шелла

```
void shell_sort(float *m, int n, int *ks, int k_ind)
{
    for (int k = ks[k_ind]; k > 0; k = ks[--k_ind])
    {
        for (int i = k; i < n; ++i)
        {
            float t = m[i];

            for (int j = i; j >= k; j -= k)
            {
                if (t < m[j - k])
                    m[j] = m[j - k];
                else break;
            }

            m[j] = t;
        }
    }
}
```

Рассмотрим возможности по векторизации сортировки Шелла для массива вещественных значений типа FP32 (вектор AVX-512 содержит 16 таких значений). Самый вложенный цикл (цикл с счетчиком $j$, будем называть его просто внутренним) выполняет сортировку одного среза, состоящего из элементов массива, с расстоянием $k$ между соседними элементами. Внутренний цикл не может быть векторизован без выполнения дополнительных модификаций кода, так как между записью элемента `m[j]` и чтением элемента `m[j - k]` существует межитерационная зависимость. Две итерации среднего по вложенности цикла (цикла с индуктивной переменной $i$, будем называть его промежуточным) с номерами $i_1$ и $i_2$ не пересекаются по данным и могут быть выполнены параллельно при выполнении условия $|i_1 - i_2| < k$. Выполним декомпозицию сортировки Шелла для того, чтобы можно было явно выделить ядро, поддающееся векторизации.

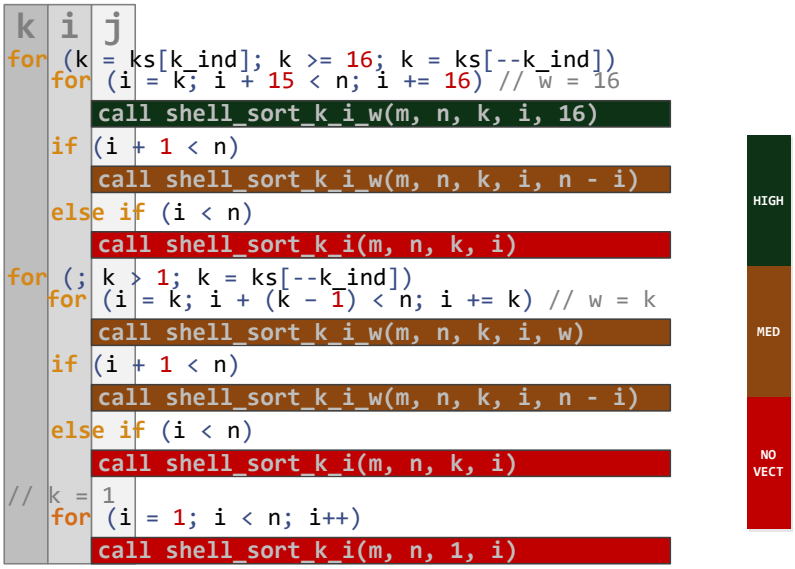


Рис. 5.30. Декомпозиция сортировки Шелла для выделения векторизуемых участков кода

На рис. 5.30 представлена схема декомпозиции сортировки Шелла, в которой выделены участки с разной максимально допустимой шириной векторизации, под которой мы будем понимать количество соседних итераций плоского цикла, независимых между собой. Если максимально допустимая ширина векторизации оказывается ниже фактического значения (в нашем случае это $w = 16$), то вычисления необходимо производить с неполными масками, что приведет к деградации производительности). При $k \geq 16$ можно параллельно выполнять 16 соседних итераций промежуточного цикла, при этом достигается максимальная плотность векторизации (показано зеленым цветом на схеме). Все итерации промежуточного цикла разбиваются на группы по 16 соседних итераций и остаток, который векторизуется с шириной меньше 16 (показано желтым цветом, а в том случае, когда остаток состоит всего из одной итерации, то векторизация не применяется, что показано красным цветом на схеме). При $1 < k < 16$ максимально допустимая ширина векторизации всегда меньше 16, к тому же, как и в предыдущем блоке, возможно появление невекторизуемого остатка. При $k = 1$ имеет место невекторизуемая сортировка вставками. Наличие участков кода с шириной векторизации менее 16 приводит к неоптимальному результирующему коду.

Функция `shell_sort_k_i_w`, появившаяся после декомпозиции алгоритма сортировки Шелла, содержит реализацию сортировки `w` соседних срезов массива, взятых с шагом $k$ (допустимое значение параметра `w` изменяется от 1 до 16). Количество итераций внутреннего цикла при сортировке одного среза никак не связано с количеством итераций внутреннего цикла при сортировке соседнего среза. Это является проявлением нерегулярности количества итераций внутреннего цикла. Для объединения соседних итераций промежуточного цикла перепишем код сортировки среза в предикатной форме, после чего заменим все инструкции векторными аналогами, а предикаты — векторными масками (рис. 5.31).

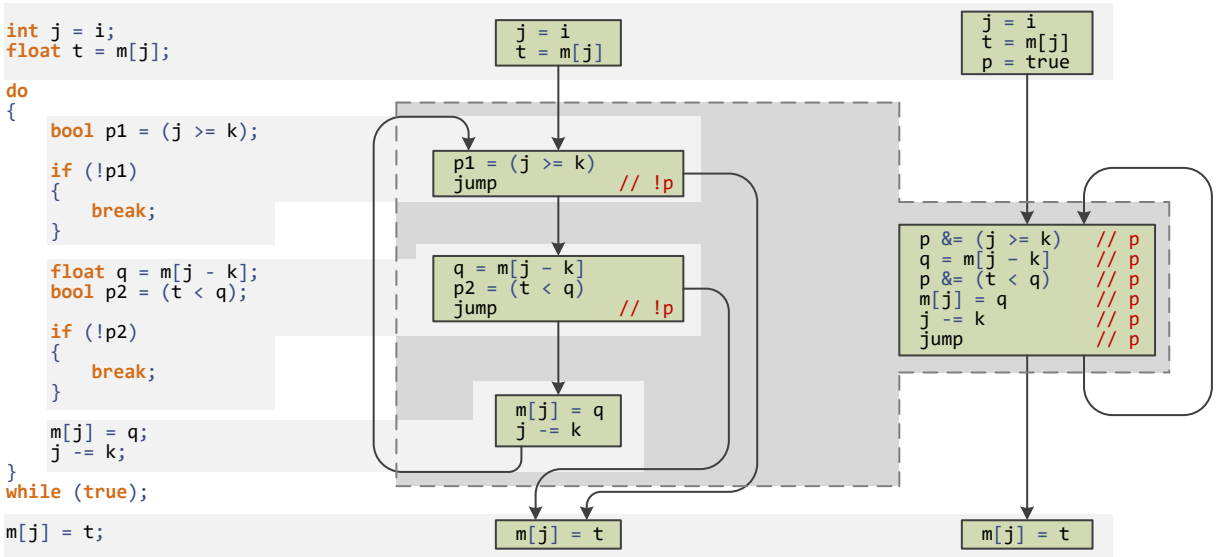


Рис. 5.31. Схема перевода тела внутреннего цикла сортировки Шелла в предикатную форму

Если значения количества итераций соседних объединяемых циклов различаются сильно (а для сортировки Шелла это утверждение верно), то для внутреннего цикла имеем большое значение $\varepsilon$, что приводит к потере эффективности векторизации из за низкой плотности масок векторных инструкций, то есть малый процент элементов векторов на самом деле обрабатывается при выполнении векторной операции. Векторизованная версия ядра сортировки Шелла представлена на листинге 5.15.

Листинг 5.15. Векторизованный вариант ядра сортировки Шелла

```
1  void shell_sort_k_i_w(float *m, int n, int k, int i, int w)
2  {
3      int j = i;
4      __mmask16 ini_mask = ((unsigned int)0xFFFF) >> (16 - w);
5      __mmask16 mask = ini_mask;
6      __m512i ind_j = _mm512_add_epi32(_mm512_set1_epi32(i),
7                                       ind_straight);
8      __m512 t = _mm512_mask_load_ps(t, mask, &m[j]), q;
9
10     do
11     {
12         mask = mask & _mm512_mask_cmp_epi32_mask(mask, ind_j, ind_k,
13                                                  _MM_CMPINT_GE);
14         q = _mm512_mask_load_ps(q, mask, &m[j - k]);
15         mask = mask & _mm512_mask_cmp_ps_mask(mask, t, q,
16                                               _MM_CMPINT_LT);
17         _mm512_mask_store_ps(&m[j], mask, q);
18         ind_j = _mm512_mask_sub_epi32(ind_j, mask, ind_j, ind_k);
19         j -= k;
20     }
21     while (mask != 0x0);
22
23     _mm512_mask_i32scatter_ps(m, ini_mask, ind_j, t, _MM_SCALE_4);
24 }
```

Также обратим внимание на медленную операцию scatter (листинг 5.15, строка 23) – векторный аналог операции записи в память из скалярного кода (листинг 5.14, строка 16) которая появилась также из-за нерегулярности количества итераций внутреннего цикла. Кроме нее в векторизованном коде присутствуют команды обращения в память в общем случае по невыровненным адресам (так как в микропроцессорах Intel скорость обращения в память vmovaps не отличается от скорости обращения vmovups при условии выровненного обращения, то компилятор icc вовсе не генерирует инструкции vmovaps [381]).

Для экспериментов на микропроцессоре Intel Xeon Phi KNL были использованы две версии исходного кода: скалярная функция сортировки Шелла и векторизованная с помощью функций-инстринсиков. Обе версии сортировки были собраны компилятором icc с уровнем оптимизации -O3. На рис. 5.32 представлены результаты экспериментальных запусков для замеров ускорения векторизованного кода для последовательностей шагов Сэджвика, Пратта и Хиббарда. Результаты показали, что ускорение редко превышает отметку 2, что говорит о низкой эффективности векторизации.

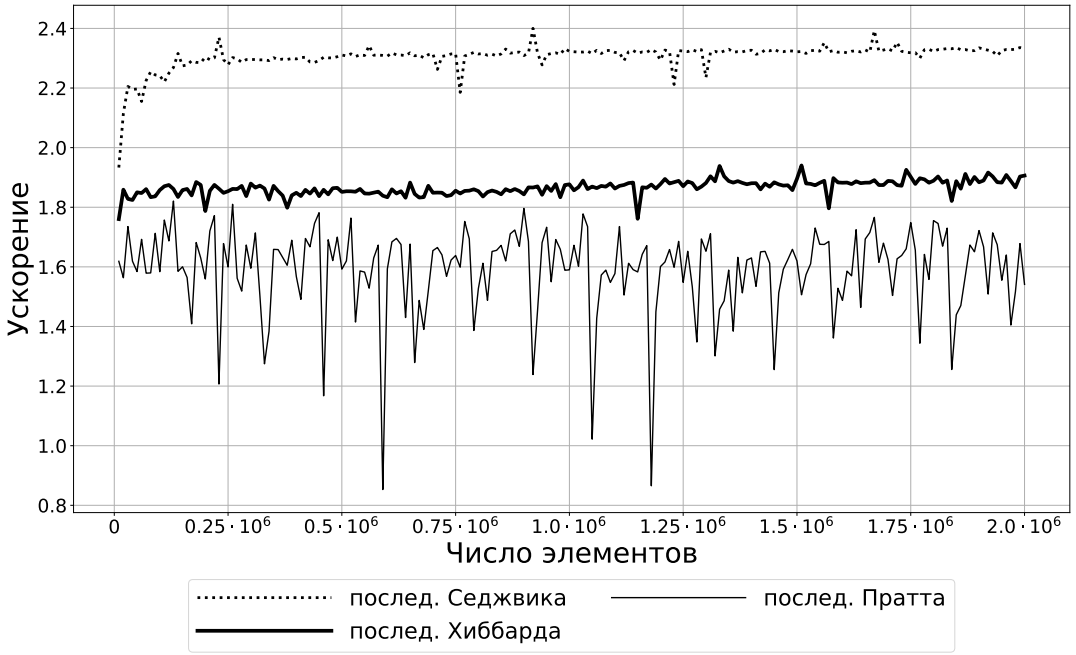


Рис. 5.32. Ускорение векторизованной версии сортировки Шелла для различных последовательностей шагов

В работе [382] рассматривается другой пример векторизации программного контекста, в котором присутствует внутренний цикл с нерегулярным количеством итераций. Рассматривается задача декомпозиции графа, программных контекст которой характеризуется целочисленными вычислениями. Результаты векторизации демонстрируют ускорение векторной версии в 1,7-2,2 раза в зависимости от размера графа, что хорошо согласуется с результатами, полученными на вещественном программном контексте.

## 5.8 Методика векторизации вычислений

На основе результатов, полученных в разделах 5.3-5.7, может быть сформулирована методика векторизации программного кода с использованием плоских циклов [383]:

- Представление вычислений в виде наборов однотипных независимых скалярных блоков, объединенных в цикл, которые впоследствии могут быть объединены в векторный блок, согласно схеме, представленной на листинге 5.3. В результате этого действия получается цикл с независимыми итерациями.

- Организация входных и выходных данных в виде наборов массивов для избавления от медленных операций множественного обращения в память gather/scatter согласно схеме, представленной на листинге 5.6. В результате этого действия достигается требование на вид обращений в память (на $i$-ой итерации все обращения к данным на запись имеют вид `a[i]`, а обращения к данным на чтение имеют вид либо `a[i]`, либо являются чтением скалярных значений).
- Обеспечение требуемого выравнивания массивов данных, обрабатываемых в цикле. В результате этого действия достигается требование на выравнивание данных (все массивы данных, обращение к которым на $i$-ой итерации имеет вид `a[i]`, выровнены в памяти на размер вектора).
- Выполнение расщепления цикла по индуктивной переменной для обеспечения количества итераций цикла, равного ширине векторизации $w$ согласно схеме, представленной на рис. 5.11. После этого действия цикл преобразуется в набор плоских циклов с одинаковым телом.
- Избавление от лишних операций передачи управления в теле плоского цикла с помощью математических тождеств, использующих семантику действий abs, min, max, blend, которые могут быть впоследствии переведены в векторные аналоги, согласно схеме на рис. 5.25.
- Избавление от лишних константных условий внутри тела плоского цикла путем расщепления цикла по константному условию согласно схеме, представленной на рис. 5.12.
- Использование профиля исполнения тела плоского цикла для определения маловероятных регионов со сложным управлением, которые препятствуют векторизации и которые могут быть вынесены из цикла путем сохранения условий входа в эти регионы, согласно схеме на рис. 5.14. В результате этого действия достигается упрощение структуры тела плоского цикла с точки зрения управления.
- Использование профиля исполнения тела плоского цикла для определения целесообразности применения слияния путей исполнения согласно схеме, представленной на рис. 5.15. При оценке целесообразности выпол-

нения слияния путей исполнения следует учитывать степень вложенности условий в теле цикла, которая не должна превышать двух для недопущения деградации логической эффективности векторизации.

- Использование профиля исполнения тела плоского цикла для определения блоков, получившихся после слияния путей исполнения, для которых целесообразно применение проверки маски на пустоту перед выполнением блока, согласно схеме представленной на рис. 5.16.
- Обработка вызовов функций, выполняющихся в теле плоского цикла под условием, с целью помещения условия вызова внутрь функции в качестве ее аргумента согласно схеме, представленной на рис. 5.26. После этого вызов функции становится безусловным и эта функция может быть векторизована в соответствии с рекомендациями настоящей методики.
- Использование профиля исполнения тела плоского цикла для определения свойств циклов и гнезд циклов внутри тела плоского цикла. Принятие решения о целесообразности векторизации на основе свойств распределения количества итераций внутреннего цикла согласно исследованиям из раздела 5.7. Внутренний цикл с постоянным количеством итераций – ожидаемая эффективность высокая, внутренний цикл с непостоянным количеством итераций – ожидаемая эффективность допустимая, внутренний цикл с нерегулярным количеством итераций – ожидаемая эффективность низкая.
- Представление операций внутри тела плоского цикла в предикатной форме, то есть когда для каждой операции прописывается условие ее выполнение в виде логической переменной (предиката) согласно схеме, представленной на рис. 5.10.
- Перевод тела плоского цикла в векторную форму путем замены скалярных операций на векторные аналоги (при этом предикаты трансформируются в векторные маски) согласно схеме на рис. 5.9 и 5.10.
- Использование профиля исполнения для анализа масок последовательности векторных блоков для применения объединения масок с целью повышения плотности масок внутри векторного блока.

Использование предложенной методики векторизации программного кода путем его организации в виде плоских циклов и использования свойств плоских циклов при выполнении оптимизаций для повышения эффективности векторизации показали свою применимость и позволили достичь кратного ускорения программного кода, что продемонстрировано на широком спектре приложений в разделах 5.3-5.7.

## 5.9 Эффективность векторизации программного контекста

В разделе рассматривается общий анализ эффективности векторизации программного контекста различного вида с помощью методов векторизации, предложенных и исследованных в настоящей главе.

Рассмотрены и предложены новые методы векторизации программного кода. На рис. 5.33 приведена карта сравнения программного контекста различного вида по эффективности векторизации $e$. На рисунке схожие по свойствам тестовые примеры объединены в одной цветовой гамме. Многие из приведенных фрагментов векторизованного программного кода были использованы при численном решении задач обледенения и газовой динамики.

В разделе 5.2 продемонстрированы подходы к векторизации перемножения матриц малой размерности. Рассмотрены разные способы выделения однотипных операций и продемонстрирована потеря производительности из-за низкой плотности масок (рис. 5.33, малоразмерные матрицы).

В разделе 5.3 введено понятие плоского цикла, с помощью которого унифицируется объединение однотипных операций путем записи тела плоского цикла в предикатной форме с последующей заменой скалярных операций векторными аналогами. Плоский цикл может быть векторизован при произвольном виде его тела (тело может содержать сложное управления, гнезда циклов, вызовы функций). Приведено описание подхода, при котором вычисления могут быть представлены в виде композиции плоских циклов с относительно простым телом, после чего успешно векторизованы оптимизирующим компилятором в автоматическом режиме (рис. 5.33, `calc_fgh`, `calc_d_to_u`, `calc_u_to_d`).

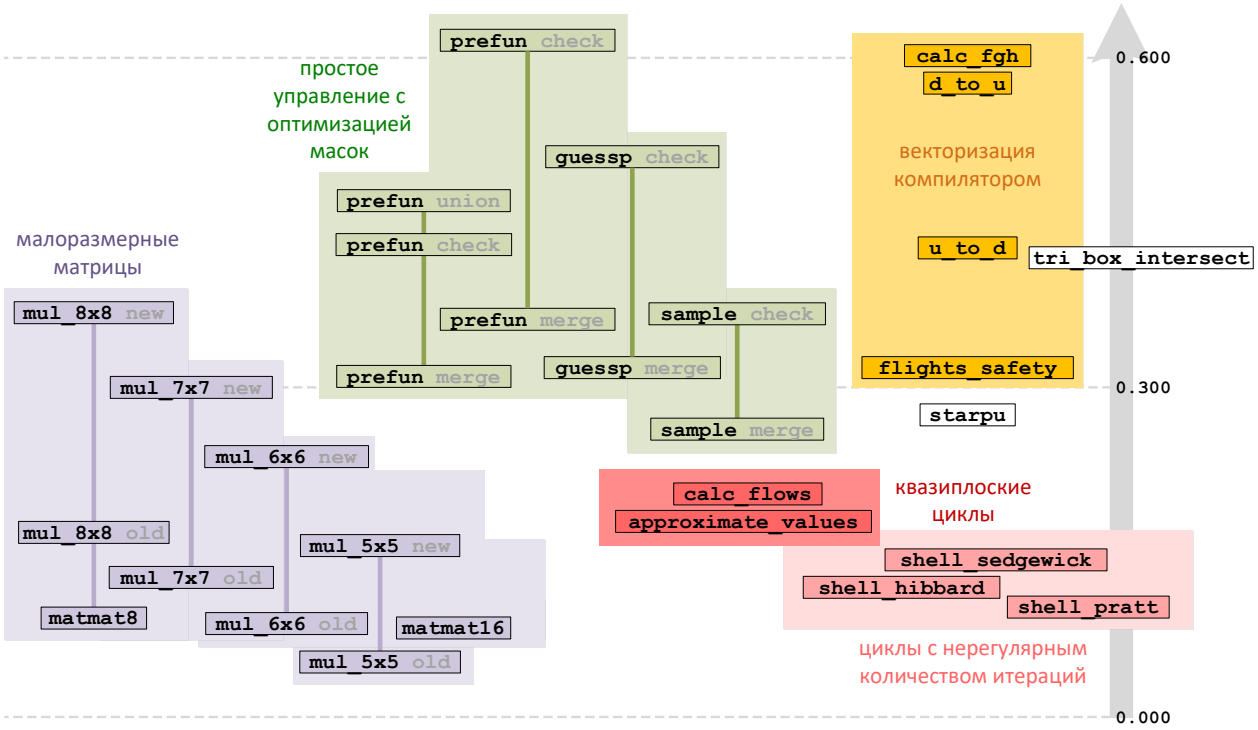


Рис. 5.33. Карта эффективности векторизации

Отмечено, что цикл, не являющийся в полной мере плоским (квазиплоский цикл), также может быть успешно векторизован, но с некоторой потерей производительности (рис. 5.33, `calc_flows`).

Так как основной преградой к эффективной векторизации является наличие операций передачи управления внутри тела плоского цикла, то в разделе 5.4 рассмотрена оптимизация, направленная на избавление от условий внутри цикла – вынос маловероятного региона из цикла. Поставлен эксперимент по определению эффективности автоматической векторизации при использовании этой оптимизации (рис. 5.33, `flights_safety`).

В разделе 5.5 приведено описание и теоретическая оценка общего подхода к избавлению от операций передачи управления и слиянию путей исполнения в теле плоского цикла с помощью векторных инструкций blend. Приведена операция проверки векторных масок на пустоту, которая оказывает наибольший прирост производительности при векторизации простого программного контекста (рис. 5.33, функции с пометкой `check`).

В разделе 5.6 предложен метод объединения векторных масок, позволяющий одновременно исполнять блоки векторных инструкций с непересекающи-

мися масками (рис. 5.33, `prefun union`). Предложен метод комбинирования векторных масок, позволяющий одновременно исполнять блоки векторных инструкций с пересекающимися масками.

В разделе 5.7 приведен анализ эффективности гнезд циклов с разными свойствами внутреннего цикла. Рассмотрены случаи внутренних циклов с постоянным (рис 5.33, `tri_box_intersect`), непостоянным (рис. 5.33, `starpu`) и нерегулярным (рис. 5.33, циклы с нерегулярным количеством итераций) количеством итераций. Наиболее сложным для векторизации случаем является векторизация гнезд циклов с нерегулярным количеством итераций. Такой программный контекст характерен для дискретных задач и он демонстрирует наиболее низкую эффективность векторизации.

## 5.10 Выводы

Введено понятие плоского цикла с рядом ограничений на его структуру как удобного программного контекста для векторизации вычислений, разработаны методы векторизации плоских циклов с телом произвольного вида. Методы направлены на сокращение количества операций передачи управления и повышение плотности векторных масок внутри тела плоского цикла и позволяют ускорить выполнение плоских циклов.

Разработана методика повышения производительности программ путем представления вычислений в виде плоских циклов, которая обеспечивает возможность объединения нескольких экземпляров подпрограммы для решения на одном процессорном ядре.

Разработанные методы векторизации и методика векторизации вычислений с помощью плоских циклов были применены для повышения производительности широкого класса практических приложений, в частности для газодинамических решателей и отдельных функций программного комплекса «Кристалл».

## Заключение

Как итог многолетних исследований и разработок, в работе автором представлен комплекс математических методов и алгоритмов для повышения производительности вычислений при моделировании объектов со сложной геометрией на современных параллельных вычислительных системах. Обобщение теоретического и практического опыта автора в области математического и программного обеспечения вычислительных систем и комплексов, проведенное в диссертации, позволило получить следующие основные результаты.

1. Разработана архитектура и состав программного комплекса, в котором реализованы математические методы и алгоритмы, применяемые для моделирования объектов со сложной геометрией с использованием высокопроизводительных вычислительных систем. Предложенные в работе методы и алгоритмы реализованы в программном комплексе расчета процесса обледенения элементов авиационных силовых установок «Кристалл». В составе комплекса решаются ресурсоемкие вычислительные задачи, возникающие при моделировании объектов со сложной геометрией с использованием высокопроизводительных вычислительных систем. В эти задачи входят перестроение и коррекция поверхностной неструктурированной сетки, описывающей границы расчетных подобластей, а также выполнение параллельных вычислений на объемных блочно-структурированных и поверхностных неструктурированных сетках.
2. Разработаны методы перестроения и коррекции поверхностной неструктурированной сетки.
    - Метод окрестностей перестроения поверхностной неструктурированной сетки, основанный на поиске пересечения траектории смещения узла с границей окрестности инцидентных узлу ячеек, обладающий способностью сглаживания локальных дефектов для предотвращения их накопления и позволяющий повысить надежность перестроения для увеличения моделируемого физического времени при моделировании объектов со сложной геометрией.

- Метод удаления самопересечений поверхностной неструктурированной сетки, основанный на обходе ее внешней поверхности, которая определяется направлениями внешних нормалей ячеек, обеспечивающий корректное состояние сетки, описывающей границу расчетной подобласти.

3. Разработаны решения по повышению производительности параллельных вычислений на объемных блочно-структурированных и поверхностных неструктурированных сетках.
   - Методы распределения блоков сетки по вычислительным процессам с использованием дробления блоков для распараллеливания вычислений на распределенной памяти, позволяющие преодолеть проблему низкой производительности вычислений из-за наличия крупных блоков и снизить показатель неравномерности распределения вычислительной нагрузки между процессами.
   - Метод сглаживания границ между доменами декомпозированной поверхностной неструктурированной сетки, позволяющий из множества локальных шаблонов сглаживания границы между парой доменов выбрать подмножество шаблонов, при одновременном применении которых обеспечивается максимальное сокращение длины границы при заданном изменении баланса ячеек между этими доменами, позволяющий уменьшить длину границ и повысить тем самым производительность вычислений за счет сокращения объема межпроцессных обменов.
   - Трехмерный метод погруженной границы с фиктивными ячейками, являющийся обобщением двумерного случая и основанный на аппроксимации физических величин в фиктивной ячейке по трем точкам пространства и направлению нормали в точке на границе, позволяющий проводить вычисления в расчетной подобласти со сложной границей без необходимости построения согласованной сетки.

4. На основе введенного понятия плоского цикла разработаны методы

векторизации таких циклов с телом произвольного вида и методика повышения производительности программ путем представления вычислений в виде плоских циклов, которая обеспечивает возможность объединения нескольких экземпляров подпрограммы для решения на одном процессорном ядре и позволяет повысить производительность вычислений на широком классе приложений.

Предложенные в диссертации архитектурные и технические решения составили комплекс математических методов и алгоритмов для повышения производительности вычислений при моделировании объектов со сложной геометрией на современных параллельных вычислительных системах. Разработанные методы и алгоритмы реализованы в составе программного комплекса «Кристалл» компьютерного моделирования процесса обледенения элементов авиационных силовых установок [126, 127], программы «crys-gsu» подготовки неструктурированной поверхностной расчетной сетки для проведения расчетов на суперкомпьютере [128], программы «GridMaster» подготовки блочно-структурированной расчетной сетки для проведения расчетов на суперкомпьютере [129] и других программных средствах [130–132]. Перечисленные программные средства внедрены в практику Национального исследовательского центра «Курчатовский институт», Федерального автономного учреждения «Центральный институт авиационного моторостроения имени П. И. Баранова», Акционерного общества «Национальный центр вертолетостроения имени М. Л. Миля и Н. И. Камова» и применяются для производства расчетов при проектировании новых промышленных изделий.

Изложенные в диссертации результаты могут быть использованы широким кругом исследователей для разработки и повышения эффективности программ для современных суперкомпьютерных систем.

**Список сокращений**

**ГМТ** – Геометрическое Место Точек

**ДУЧП** – Дифференциальное Уравнение в Частных Производных

**ИИ** – Искусственный Интеллект

**ИМ** – Имитационное Моделирование

**ЛА** – Летательный Аппарат

**МГД** – МагнитоГидроДинамический

**МДВ** – Метод Дискретных Вихрей

**МПГ** – Метод Погруженной Границы

**НЛГ БАС-СТ** – Нормы Летной Годности Беспилотных Авиационных Систем Самолетного Типа

**НСКФ** – Национальный СуперКомпьютерный Форум

**ПОС** – ПротивоОбледенительная Система

**РМГ** – Разрывный Метод Галёркина

**СЛАУ** – Система Линейных Алгебраических Уравнений

**СУППЗ** – Система Управления Прохождением Параллельных Заданий

**ЭВМ** – Электронно-Вычислительная Машина

**AC** – Avdisory Circular

**AMD** – Advanced Micro Devices

**AMR** – Adaptive Mesh Refinement

**AOS** – Array Of Structures

**ALE** – Arbitrary Lagrangian-Eulerian

**ARM** – Advanced RISC Machine

**ARP** – Aerospace Recommended Practices

**AVX** – Advanced Vector eXtensions

**BF** – Brain Float

**BITALG** – BIT ALGorithm

**BVH** – Boundede Volume Hierarchy

**BW** – Byte and Word

**CD** – Conflict Detection

**CFD** – Computational Fluid Dynamics

**CFG** – Control Flow Graph

**CFR** – Code of Federal Regulations

**CNN** – Convolutional Neural Network

**CPU** – Central Processing Unit

**CVT** – Centroidal Voronoi Tesselation

**DGM** – Discontinuous Galerkin Method

**DQ** – Doubleword and Quadword

**DT** – Delaunay Triangulation

**EEIR** – Exact and Efficient Intersection Resolution

**ER** – Exponential and Reciprocal

**FAA** – Federal Aviation Administration

**FMA** – Fused Multiply-Add

**FMAPS** – Fused Multiply Accumulation Packed Single

**FOAM** – Field Operation And Manipulation

**FP** – Floating Point

**FRMA** – Fast and Robust Mesh Arrangements

**GF** – Galois Field

**GFNI** – GF New Instructions

**GNN** – Graph Neural Network

**GPU** – Graphics Processing Unit

**ICAM** – International Conference on Aviation Motors

**IBM** – Immersed Boundary Method

**IFMA** – Integer FMA

**ILES** – Implicit Large Eddy Simulation

**ISO** – International Organization for Standardization

**IXPUG** – Intel eXtreme Performance Users Group

**KNC** – KNights Corner

**KNL** – KNights Landing

**LES** – Large Eddy Simulation

**LLI** – Line-Line Intersection

**LLM** – Large Language Model

**LPI** – Line-Plane Intersection

**LS** – Laplacian Smoothing

**MG** – MultiGrid

**MISRA** – Motor Industry Software Reliability Association

**MMX** – MultiMedia eXtensions

**MPI** – Message Passing Interface

**PF** – PreFetch

**PINN** – Physics-Informed Neural Network

**RANS** – Reynolds-Averaged Navier-Stokes

**RISC** – Recuced Instruction Set Computer

**RSCI** – Russian Science Citation Index

**SDLT** – SIMD Data Layout Template

**SGS** – SubGrid Scale

**SIMD** – Single Instruction, Muptiple Data

**SLURM** – Simple Linux Utility for Resource Management

**SOA** – Structure Of Arrays

**SPH** – Smoothed Particle Hydrodynamics

**SSE** – Streaming SIMD Extensions

**SVE** – Scalable Vector Extensions

**SWIM** – Shallow Water Icing Model

**TNOIT** – Tracing of the Neighbours Of Intersecting Triangles

**TPI** – Triplet-Plane Intersection

**VAES** – Vector Advanced Encryption Standard

**VBMI** – Vector Byte Manipulation Instructions

**VL** – Vector Length

**VLIW** – Very Long Instruction Word

**VMX** – Vector and Media eXtensions

**VNNI** – Vector Neural Network Instructions

**VNNIW** – VNNI Word

**VOF** – Volume Of Fluid

**Список литературы**


[1] **Шабанов Б. М.** Методы и способы построения, выбора и применения высокопроизводительных вычислительных систем для выполнения научных и технических задач. // Диссертация на соискание ученой степени доктора технических наук, Москва, 2019.

[2] **Соколов И. А., Степченков Ю. А., Бобков С. Г., Захаров В. Н., Дьяченко Ю. Г., Рождественский Ю. В., Сурков А. В.** Базис реализации супер-ЭВМ экзафлопсного класса. // Информатика и ее применения, 2014, Т. 8, № 1, С. 45-70. DOI: 10.14357/19922264140106.

[3] **Абрамов Н. С., Абрамов С. М.** Ноябрь 2022: состояние и перспективы развития суперкомпьютерной отрасли в мире и в России. // Программные системы: Теория и приложения, 2023, Т. 14, № 2(57), С. 49-93. DOI: 10.25209/2079-3316-2023-14-2-49-93.

[4] **Stegailov V., Agarkov A., Biryukov S., Ismagilov T. et al.** Early performance evaluation of the hybrid cluster with torus interconnect aimed at molecular-dymanics simulations. // In: Wyrzykowski R., Dongarra J., Deelman E., Karczewski K. (eds) Parallel Processing and Applied Mathematics, PPAM 2017, Lecture Notes in Computer Science, Vol. 10777, Springer, Cham. DOI: 10.1007/978-3-319-78024-5_29.

[5] **Корнев А. В., Козелков А. С.** Применение отечественных суперкомпьютерных технологий для создания перспективных образцов авиационной техники. // Современные информационные технологии и ИТ-образование, 2021, Т. 17, № 2, С. 250-264. DOI: 10.25559/SITITO.17.202102.250-264.

[6] **Wang Y., Han P., Hou J.** Design and implement of HPC simulation cloud platform for automobile industry. // In book: International Conference on Applications and Techniques in Cyber Intelligence ATCI 2019, 2020. DOI: 10.1007/978-3-030-25128-4_38.

[7] **Агеева А. Ф.** Роль суперкомпьютеров в вопросах националь-

ной безопасности. // Развитие экономики в условиях цифровой трансформации: Проблемы, достижения и инновации, 2023, № 1. DOI: 10.51409/v.a.2023.03.01.005.

[8] **Wang X., Meng X., Guo Z. et al.** 29-billion atoms molecular dynamics simulation with Ab Initio accuracy on 35 million cores of new Sunway supercomputer. // IEEE Transactions on Computers, 2025, 99, P. 1-14. DOI: 10.1109/TC.2025.3540646.

[9] **Cancemi S., Frano R., Angelucci M.** HPC modelling for nuclear safety assessment. // Journal of Physics: Conference Series, 2025, 2940:012025. DOI: 10.1088/1742-6596/2940/1/012025.

[10] **Quint D., Lundquist J., Rosencrans D.** Simulations suggest offshore wind farms modify low-level jets. // Wind Energy Systems, 2025, 10(1), P. 117-142. DOI: 10.5194/wes-10-117-2025.

[11] **Усманов Д. И., Поташев К. А., Салимьянова Д. Р.** Идентификация граничных условий фильтрационной модели нефтяного пласта по замерам давления в скважинах. Часть 1: Однородный пласт. // Ученые записки Казанского университета. Серия Физико-математическе науки, 2024, Т. 166, Кн. 4, С. 603-623. DOI: 10.26907/2541-7746.2024.4.603-623.

[12] **Didenko N., Skripnuk D., Merkulov V., Kikkas K., Skripniuk K.** Methodology for the formation of a digital model of the life cycle of an offshore oil and gas platform. // Resources, 2023, 12, 86. DOI: 10.3390/resources12080086.

[13] **Теплухин А. В.** Моделирование больших молекулярных агрегатов с использованием параллельных вычислений методом Монте-Карло. // В книге: Методы компьютерного моделирования для исследовния полимеров и биополимеров, под. ред. Иванов В., Рабинович А., Хохлов А., М:. Либроком, 2009, Гл. 17, С. 553-570. DOI: 10.13140/RG.2.1.2050.6725.

[14] **Colonnelli I., Cantalupo B., Spampinato C. et al.** Bringing AI pipelines

onto cloud-HPC: setting a baseline for accuracy of COVID-19 AI diagnosis. // arXiv, 2021. DOI: 10.48550/arXiv.2108.01033.

[15] **Ridhi S., Robert D., Soren P. et al.** Comparing the output of an artificial intelligence algorithm in detecting radiological signs of pulmonary tuberculosis in digital chest X-Rays and their smartphone-captured photos of X-Ray films: Retrospective study. // JMIR Formative Research, 2024, 8:e55641. DOI: 10.2196/55641.

[16] **Козачок А. В., Спирин А. А., Самоваров О. И., Козачок Е. С.** Применение моделей машинного обучения для многоклассовой классификации дерматоскопических снимков новообразований кожи. // Труды института системного программирования РАН, 2024, Т. 36, № 5, С. 241-252. DOI:10.15514/ISPRAS-2024-36(5)-17.

[17] **Kishore B., Pinjala S.** Transformation of evidence-based medicine to precision medicine by AI: A roadmap. // Journal of the American Association of Physicians of Indian Origin, 2024, 4(1,2), P. 14-36.

[18] **Василевский Ю. В., Симаков С. С., Гамилов Т. М., Саламатова В. Ю. и др.** Персонализация математических моделей в кардиологии: трудности и перспективы. // Компьютерные исследования и моделирование, 2022, Т. 14, вып. 4, С. 911-930, DOI: 10.20537/2076-7633-2022-14-4-911-930.

[19] **Ахметшина А. О., Стрыгина К. В., Хлесткина Е. К., Пороховинова Е. А., Брач Н. Б.** Высокопроизводительное секвенирование в генетике и селекции льна. // Экологическая генетика, 2020, Т. 18, № 1, С. 103-124. DOI: 10.17816/ecogen16126.

[20] **Mourant J., Fenimore P., Manore C., McMahon B.** Decision support for mitigation of livestock disease: Rinderpest as a case study. // Veterinary Epidemiology and Economics, 2018, Vol. 5. DOI: 10.3389/fvets.2018.00182.

[21] **Irfan S., Razali R., Ku K., Mansor N.** Modelling and simulations of controlled release fertilizer. // 4th International Conference on

Fundamental and Applied Sciences Conference Proceedings, 2016, 1787, 080025. DOI: 10.1063/1.4968164.

[22] **Zhang Z., Yu Z., Zhang Y., Shi Y.** Optimized nitrogen fertilizer application strategies under supplementary irrigation improved winter wheat (Triticum aestivum L.) yield and grain protein yield. // PeerJ, 2021, 9:e11467. DOI: 10.7717/peerj.11467.

[23] **Juntana P., Delahaye D., Chaimatanan S., Alam S.** Hyperheuristic approach based on reinforcement learning for air traffic complexity mitigation. // Journal of Aerospace Computing, Information and Communication, 2022, 19(1), P. 1-16. DOI: 10.2514/1.i011048.

[24] **Yan Q., Song R., Kim K.-H., Wang Y., Feng X.** Optimizing container relocation operations by using deep reinforcement learning. // Maritime Policy & Management, 2024, P. 1-23. DOI: 10.1080/03088839.2024.2424865.

[25] **Abramov A., Gonchar A., Evseev A., Shabanov B.** Results of the development of National Research Computer Network of Russia within the framework of the National Project "Science and Universities" in 2021—2024. // Informational Technologies, 2025, Vol. 31, № 1. DOI: 10.17587/it.31.35-41.

[26] **Kulkarni P., Manoharan S., Gaddi A.** Advancing climate modeling through high-performance computing: Towards more accurate and efficient simulations. // EAI Endorsed Transactions on Energy Web, 2024. DOI: 10.4108/ew.7049.

[27] **Wei J., Han X., Yu J., Jiang J. et al.** A performance-portable kilometer-scale global ocean model on ORISE and new Sunway heterogeneous supercomputers. // International Conference for High Performance Computing, Networking, Storage and Analysis, Atlanta, US, 2024. DOI: 10.1109/SC41406.2024.00009.

[28] **Rahman A., Tikhonov G., Oksanen J., Rossi T., Ovaskainen O.** Accelerating joint species distribution modeling with Hmsc-HPC: A 1000x faster GPU deployment. // bioRxiv, 2024. DOI: 10.1101/2024.02.13.580046.

[29] **Ostromsky T.** Optimization, performance and scalability experiments of a large air pollution model by using the EuroHPC petascale supercomputer Discoverer. // Studies in Computational Intelligence, 2024. DOI: 10.1007/978-3-031-57320-0_8.

[30] **Четверушкин Б. Н., Борисов В. Е., Луцкий А. Е., Ханхасаева Я. В.** Численное моделирование трехмерного обтекания воздухозаборника. // Математическое моделирование, 2024, Т. 36, № 3, С. 51-66. DOI: 10.20948/mm-2024-03-04.

[31] **Горобец А. В.** Параллельные технологии математического моделирования турбулентных течений на современных суперкомпьютерах. // Автореферат на соискание ученой степени доктора физико-математических наук, Москва, 2015.

[32] **Лобанова Е. Е., Тарасов Н. И.** Применение цифровой платформы для моделирования задач газовой динамики. // Препринты ИПМ им. М. В. Келдыша, 2023, № 77, 20 С. DOI: 10.20948/prepr-2023-77.

[33] **Taboada J., Araújo M., Basteiro F., Rodríguez J., Landesa L.** MLFMA-FFT parallel algorithm for the solution of extremely large problems in electromagnetics. // Proceedings of the IEEE, 2013, Vol. 101, No. 2, P. 350-363. DOI: 10.1109/JPROC.2012.2194269.

[34] **MacKie-Mason B., Greenwood A., Peng Z.** Adaptive and parallel surface integral equation solvers for very large-scale electromagnetic modeling and simulation. // Progress in Electromagnetics Research, 2015, Vol. 154, P. 143-162.

[35] **Подрыга В. О., Поляков С. В., Пузырьков Д. В.** Суперкомпьютерное молекулярное моделирование термодинамического равновесия в микросистемах газ-металл. // Вычислительные методы и программирование, 2015, Т. 16, С. 123-138, DOI: 10.26089/NumMet.v16r113.

[36] **Liu Z.-K.** Computational thermodynamics and its applications. // Acta Materialia, 2020, Vol. 200, P. 745-792. DOI: 10.1016/j.actamat.2020.08.008.

[37] **Гасилов В. А., Болдарев А. С., Ольховская О. Г., Бойков Д. С., Шарова Ю. С., Савенко Н. О., Котельников А. М.** MAPLE: программное обеспечение для мультифизического моделирования в задачах сплошных сред. // Препринты ИПМ им. М. В. Келдыша, 2023, № 37, 41 С. DOI: 10.20948/prepr-2023-37.

[38] **Koren C., Vicquelin R., Gicquel O.** Multiphysics simulation combining large-eddy simulation, wall heat conduction and radiative energy transfer to predict wall temperature induced by a confined premixed swirling flame. // Flow Turbulence Combustion, 2018, Vol. 101, P. 77–102. DOI: 10.1007/s10494-018-9895-5.

[39] **Yousuf A., Khawaja H. A., Virk M. S.** Analyzing gridded heater arrangement in anti-/de-icing systems through multiphysics thermal simulation. // Multiscale and Multidisciplinary Modeling, Experiments and Design, 2026, Vol. 9, 100. DOI: 10.1007/s41939-026-01168-z.

[40] **Кудрявцев А. О., Кошелев В. К., Избышев А. О., Дудина И. А., Курмангалеев Ш. Ф., Аветисян А. И., Иванников В. П., Велихов В. Е., Рябинкин Е. А.** Разработка и реализация облачной системы для решения высокопроизводительных задач. // Труды ИСП РАН, 2013, Т. 24, С. 13-34. URL: https://ispranproceedings.elpub.ru/jour/article/view/948 (дата обращения 24.06.2026).

[41] **Черных А. Н., Бычков И. В., Феоктистов А. Г., Горский С. А., Сидоров И. А., Костромин Р.О., Еделев А. В., Зоркальцев В. И., Аветисян А. И.** Смягчение неопределенности при разработке научных приложений в интегрированной среде. // Труды института системного программирования РАН, 2021, Т. 33, № 1, С. 151-172. DOI: 10.15514/ISPRAS-2021-33(1)-11.

[42] **Востокин С. В.** Обзор проекта автоматизации параллельных и распределенных вычислений Templet. // Перспективные информационные технологии (ПИТ 2022), труды Международной научно-технической конфе-

ренции, С. 235-238. URL: http://repo.ssau.ru/jspui/handle/123456789/3076 (дата обращения 24.06.2026).

[43] **Bacosi J., Constans M., Horváth Z., Kovács Á., Környei L., Charest M., Pandare A., Rutherford P., Waltz J.** Complex-geometry 3D computational fluid dynamics with automatic load balancing. // Fluids, 2023, 8, 147, 11 P. DOI: 10.3390/fluids8050147.

[44] **Василевский Ю. В., Терехов К. М.** Нелинейный конечно-объемный метод для задачи переноса-сжатия границы раздела сред на неструктурированных адаптивных сетках. // Журнал вычислительной математики и математической физики, 2022, Т. 62, № 7, С. 1067-1084. URL: 10.31857/S0044466922060151.

[45] **Меньшов И. С., Сережкин А. А.** Численная модель многофазных течений на основе подсеточного разрешения контактных границ. // Журнал вычислительной математики и математической физики, 2022, Т. 62, № 10, С. 1740-1760. DOI: 10.31857/S0044466922090101.

[46] **Балашов В. А.** Численное моделирование двумерных течений умеренно-разреженного газа в областях со сложной геометрией. // Препринты ИПМ им. М. В. Келдыша, 2016, № 104, 24 С. URL: https://library.keldysh.ru/preprint.asp?id=2016-104 (дата обращения 24.06.2026).

[47] **Способин А. В.** Метод скользящих адаптивных декартовых сеток расчета газодинамического взаимодействия частиц с ударным слоем в сверхзвуковом потоке. // Тепловые процессы в технике, 2022, Т. 14, № 4, С. 178-185. DOI: 10.34759/tpt-2022-14-3-178-185.

[48] **Елизарова Т. Г., Булатов О. В.** Численный алгоритм решения регуляризованных уравнений мелкой воды на неструктурированных сетках. // Препринты ИПМ им. М. В. Келдыша, 2014, № 21, 27 С. URL: https://library.keldysh.ru/preprint.asp?id=2014-21 (дата обращения 24.06.2026).

[49] **Sorokin D. L., Skuybin B. G., Galkin N. K., Litvinov V. N., Dushkin M. A., Ryabokon M. S.** Simulation of the electrostatic fields in devices with complex geometric shapes. // Journal of Physiscs: Conference Series, 2019, 1348(1):012048. DOI: 10.1088/1742-6596/1348/1/012048.

[50] **Жуковский М. Е., Воронин Ф. Н., Подоляко С. В.** Суперкомпьютерное моделирование переноса нейтронов в сложных технических объектах. // Препринты ИПМ им. М. В. Келдыша, 2015, № 101, 27 С. URL: https://library.keldysh.ru/preprint.asp?id=2015-101 (дата обращения 24.06.2026).

[51] **Вабищевич П. Н., Варламов С. П., Васильев В. И., Васильева М. В., Степанов С. П.** Математическое моделирование теплового режима железнодорожного полотна в условиях криолитозоны. // Вестник Северо-Восточного федерального университета им. М. К. Аммосова, 2013, Т. 10, № 5, С. 5-11. URL: https://www.elibrary.ru/item.asp?id=21271395&ysclid=mi4nnmge13161305810 (дата обращения 24.06.2026).

[52] **Chen N., Hu Y., Ji H., Cao G., Yuan Y.** A mathematical model based on unstructured mesh for ice accretion. // AIP Advances, 2019, Vol. 9, Issue 12, 125149. DOI: 10.1063/1.5127235.

[53] **Liseikin V. D.** Grid generation methods. // Springer, 2017, 541 P. DOI: 10.1007/978-3-319-57846-0.

[54] **Кудрявцева Л. Н.** Методы самоорганизации и оптимизации для построения трехмерных расчетных сеток. // Диссертация на соискание ученой степени кандидата физико-математических наук, 2014, Москва.

[55] **Boissonnat J.-D., Teillaud M.** Effective computational geometry for curves and surfaces. // Springer Berlin, Heidelberg, 2006, 352 P. DOI: 10.1007/978-3-540-33259-6.

[56] **Железнякова А. Л.** Унифицированный подход к созданию сложных виртуальных поверхностей и расчетных сеток для комплексного имита-

ционного 3D моделирования современных изделий аэрокосмической техники. // Физико-химическая кинетика в газовой динамике, 2016, Т. 17(2). URL: http://chemphys.edu.ru/issues/2016-17-2/articles/634 (дата обращения 24.06.2026).

[57] **Li Y., Jeong D., Kim J.** Adaptive mesh refinement for simulation of thin film flows. // Meccanica, 2014, Vol. 49, P. 239-252. DOI: 10.1007/s11012-013-9788-6.

[58] **Кошелев К. Б., Мельникова В. Г., Стрижак С. В.** Разработка решателя iceFoam для моделирования процесса обледенения. // Труды ИСП РАН, 2020, Т. 32, Вып. 4, С. 217-234. DOI: 10.15514/ISPRAS-2020-32(4)-16.

[59] **Гаранжа В. А., Кудрявцева Л. Н.** Движение и деформация квазиизометрической сетки с геометрически адаптированной метрикой. // Журнал вычислительной математики и математической физики, 2022, Т. 62, № 8, С. 1300-1322. DOI: 10.31857/S0044466922080063.

[60] **Самарский А. А., Вабищевич П. Н.** Разностные методы решения задач математической физики на нерегулярных сетках. // Математическое моделирование, 2001, Т. 13, № 2, С. 5-16.

[61] **Абалакин И. В., Козубская Т. К.** Схема на основе реберно-ориентированной квазиодномерной реконструкции переменных для решения задач аэродинамики и аэроакустики на неструктурированных сетках. // Математическое моделирование, 2013, Т. 25, № 8, С. 109-136.

[62] **Волков К. Н.** Многосеточный метод ускорения сходимости при решении задач газовой динамики на неструктурированных сетках. // Научно-технический вестник информационных технологий, механики и оптики, 2014, № 4(92). URL: https://cyberleninka.ru/article/n/mnogosetochnyy-metod-uskoreniya-shodimosti-pri-reshenii-zadach-gazovoy-dinamiki-na-nestrukturirovannyh-setkah (дата обращения 24.06.2026).

[63] **Мартыненко С. И.** Универсальная многосеточ-

ная технология. // ИПМ им. Келдыша, 2013, 244 С. URL: https://library.keldysh.ru/prep_vw.asp?pid=3715 (дата обращения 24.06.2026).

[64] **Бахвалов П. А.** О точности разрывного метода Галёркина для одномерного уравнения при длительном счете. // Препринты ИПМ им. М. В. Келдыша, 2017, № 134, 24 С. DOI: 10.20948/prepr-2017-134.

[65] **Zhang Y.-T., Shu C.-W.** Third order WENO scheme on three dimensional tetrahedral meshes. // Communications in Computational Physiscs, 2009, Vol. 5, No. 2-4, P. 836-848.

[66] **Liu M. B., Liu G. R.** Smoothed particle hydrodynamics (SPH): an overview and recent development. // Archives of Computational Methods in Engineering, 2010, Vol. 17, P. 25-76. DOI: 10.1007/s11831-010-9040-7.

[67] **Потапов И. И., Решетникова О. В.** К моделированию задачи осаждения твердой частицы в вязкой несжимаемой жидкости методом гидродинамики сглаженных частиц (SPH). // Труды ИСП РАН, 2024, Т. 36, Вып. 4, С. 191-202. DOI: 10.15514/ISPRAS–2024–36(4)–15.

[68] **Давыдов М. Н., Кедринский В. К.** Метод сглаженных частиц в задачах моделирования кавитационного разрушения жидкости при ударно-волновом нагружении. // Прикладная математика и техническая физика, 2013, Т. 54, № 6, С. 17-26.

[69] **Сумбатян М. А., Пискунов А. С.** Быстрый алгоритм мультиполя в бессеточном методе дискретных вихрей для течений идеальной жидкости. // Известия вузов. Северо-Кавказский регион. Серия: Естественные науки, 2022, № 3(215), С. 29-37. DOI: 10.18522/1026-2237-2022-3-29-37.

[70] **Зайцев Д. К., Смирнов П. Е., Якубов С. А., Балашов М. Е.** Комплекс программ для создания блочно-структурированных сеток. // Программные продукты и системы, 2012, № 2, С. 32-35. URL: https://swsys.ru/index.php?id=3107&page=article (дата обращения 24.06.2026).

[71] **Лыкосов В. Н., Глазунов А. В., Кулямин Д. В., Мортиков Е. В., Степаненко В. М.** Суперкомпьютерное моделирование в физике климатической системы. // Издательство Московского университета, 2012, 408 С. ISBN: 978-5-211-06341-9.

[72] **Титов П. А.** Моделирование 3D волновых полей в неоднородной области со сложной топографией при помощи схемы Лебедева. // Вестник НГУ, Серия: Информационные технологии, 2020, Т. 18, № 4, С. 66-85. DOI: 10.25205/1818-7900-2020-18-4-66-85.

[73] **Яцухно Д. С.** О некоторых практических аспектах построения расчетных сеток для задач вычислительной аэротермодинамики с использованием эллиптического сеточного генератора. // Физико-химическая кинетика в газовой динамике, 2023, Т. 24, № 6. DOI: 10.33257/PhChGD.24.6.1036.

[74] **Khairulin B., Rykovanov S., Zagidulin R.** Neural networks for structured grid generation. // Scientific Reports, 2025, 15:12526. DOI: 10.1038/s41598-025-97059-3.

[75] **Петросян А. С.** Дополнительные главы гидродинамики тяжелой жидкости со свободной границей. // Серия «Механика, управление, информатика», Москва, 2010, 128 С.

[76] **Винников В. В., Ревизников Д. Л.** Применение декартовых сеток для решения уравнений Навье-Стокса в областях с криволинейными границами. // Математическое моделирование, 2005, Т. 17, № 8, С. 15-30.

[77] **Луцкий А. Е., Меньшов И. С., Ханхасаева Я. В.** Использование метода свободной границы для решения задач обтекания движущихся тел. // Препринты ИПМ им. М. В. Келдыша, 2014, № 93, 16 С. DOI: https://library.keldysh.ru/preprint.asp?id=2014-93.

[78] **Mohan A., Tomar G.** Volume of fluid method: a brief review. // Journal of Indian Institute of Science, 2024, Vol. 104, № 1, P. 229-248. DOI: 10.1007/s41745-024-00424-w.

[79] **Мортиков Е. В.** Применение метода погруженной границы для решения

системы уравнений Навье-Стокса в областях сложной конфигурации. // Вычислительные методы и программирование, 2010, Т. 11, № 1, С. 32-42. URL: https://www.mathnet.ru/rus/vmp292 (дата обращения 24.06.2026).

[80] **Абалакин И. В., Жданова Н. С., Козубская Т. К.** Метод погруженных границ для численного моделирования невязких сжимаемых течений. // Журнал вычислительной математики и математической физики, 2018, Т. 58, № 9, С. 1462-1471. DOI: 10.31857/S004446690002525-8.

[81] **Афендиков А. Л., Меньшов И. С., Меркулов К. Д., Павлухин П. В.** Метод адаптивных декартовых сеток для решения задач газовой динамики. // Российсская академия наук, 2017, 64 С. ISBN: 978-5-906906-57-1.

[82] **Mittal R., Iaccarino G.** Immersed boundary methods. // Annual. Rev. Fluid Mech., 2005, Vol. 37, P. 239-261. DOI: 10.1146/annurev.fluid.37.061903.175743.

[83] **Yousefzadeh M., Battiato I.** High order ghost-cell immersed boundary method for generalized boundary conditions. // International Journal of Heat and Mass Transfer, 2019, 137, P. 585-598. DOI: 10.1016/j.ijheatmasstransfer.2019.03.061.

[84] **Винников В. В., Ревизников Д. Л.** Метод погруженной границы для расчета сверхзвукового обтекания затупленных тел на прямоугольных сетках. // Электронный журнал «Труды МАИ», 2007, № 27, 13 С.

[85] **Luo K., Zhuang Z., Fan J., Haugen N.** A ghost-cell immersed boundary method for simulations of heat transfer in compressible flows under different boundary conditions. // International Journal of Heat and Mass Transfer, 2016, 92, P. 708-717. DOI: 10.1016/j.ijheatmasstransfer.2015.09.024.

[86] **Tseng Y.-H., Ferziger J.** A ghost-cell immersed boundary method for flow in complex geometry. // Journal of Computational Physics, 2003, Vol. 192, P. 593-623. DOI: 10.1016/j.jcp.2003.07.024.

[87] **Peter S., De A.** A parallel implementation of the ghost-cell immersed

boundary method with application to stationary and moving boundary problems. // Sadhana, 2016, Vol. 41, № 4, P. 441-450. DOI: 10.1007/s12046-016-0484-9.

[88] **Волков-Богородский Д. Б., Сушко Г. Б., Харченко С. А.** Комбинированная MPI+Threads параллельная реализация метода блоков для моделирования тепловых процессов в структурно-неоднородных средах. // Вычислительные методы и программирование, 2010, Т. 11, № 1, С. 127-136.

[89] **Милюкова О. Ю.** MPI+OpenMP реализация метода сопряженных градиентов с факторизованным предобуславливателем на основе использования переупорядочения узлов сетки. // Препринты ИПМ им. М. В. Келдыша, 2023, № 18, 29 С. DOI: https://doi.org/10.20948/prepr-2023-18.

[90] **Saczek M., Wawrzak K., Tyliszczak A., Boguslawski A.** Hybrid MPI/Open-MP acceleration approach for high-order schemes for CFD. // Journal of Marketing Channels, 2018, Vol. 22, № 3, P. 179-193. DOI: 10.17466/tq2018/22.3/e.

[91] **Тюляева Е. О., Конюхов С. С., Московский А. А., Одинцов И. О.** Оценка потенциала использования платформы Эльбрус для высокопроизводительных вычислений. // Суперкомпьютерные дни в России, 2016, С. 373-385.

[92] **Волков К. Н., Емельянов В. Н., Карпенко А. Г., Тетерина И. В.** Моделирование течений вязкой несжимаемой жидкости на графических процессорах при помощи схемы расщепления и многосеточного метода. // Журнал вычислительной математики и математической физики, 2019, Т. 59, № 1, С. 143-157. DOI: 10.1134/S0044466919010162.

[93] **Брыков Н. А., Волков К. Н., Емельянов В. Н.** Использование векторизованных структур данных при реализации вычислительных алгоритмов решения задач механики сплошной среды. // Научно-технический

вестник информационных технологий, механики и оптики, 2022, Т. 22, № 1, С. 193-205. DOI: 10.17586/2226-1494-2022-22-1-193-205.

[94] **Жлуктов С. В., Аксенов А. А., Харченко С. А., Москалев И. В., Сушко Г. Б., Шишаева А. С.** Моделирование отрывных течений в программном комплексе FlowVision-HPC. // Вычислительные методы и программирование, 2010, Т. 11, С. 234-245.

[95] **Сорокин К. Э., Бывальцев П. М., Аксенов А. А., Жлуктов С. В., Савицкий Д. В., Бабулин А. А., Шевяков В. И.** Численное моделирование обледенения в программном комплексе FlowVision. // Компьютерные исследования и моделирование, 2020, Т. 2020, № 1, С. 83-96. DOI: 10.20537/2076-7633-2020-12-1-83-96.

[96] **Liu Y., Hu Y., Xie D., Zhang Y., Chen J.** A lightweight interactive approach for unstructured surface mesh generation. // Engineering Computations, 2025, Vol. 42, Issue 6, P. 1942-1961. DOI: 10.1108/EC-03-2023-0103.

[97] **Борисенко О. Н., Лукичев А. Н., Евстифеева Е. О., Панкратов Д. М., Цалко Т. В., Гиниятуллина А. Г.** Алгоритмы обработки особенностей геометрических моделей при построении поверхностных треугольных сеток в препроцессоре пакета программ "ЛОГОС". // ВАНТ, серия Математическое моделирование физических процессов, Вып. 3, С. 40-51.

[98] Ansys ICEM CFD User's Manual. // Release 2021 R2. URL: https://dl.cfdexperts.net/cfd_resources/Ansys_Documentation/ICEM%20CFD/Ansys_ICEM_CFD_Users_Manual.pdf (дата обращения 24.06.2026).

[99] **Sigrest P., Wu E., Inman D.** Energy considerations and flow fields over whiffling-inspired wings. // Bioinspiration & Biomimetics, 2023, № 18, 046007. DOI: 10.1088/1748-3190/acd28f.

[100] Introduction to COMSOL Multiphysics. // Version 6.2.

URL: https://cdn.comsol.com/doc/6.2.0.278/IntroductionToCOMSOLMultiphysics.pdf (дата обращения 24.06.2026).

[101] **Strijhak S. V., Koshelev K. B., Melnikova V. G.** Using a thermodynamic film model based on shallow water theory and a dynamic mesh model for the icing of 2D/3D bodies in the iceFoam solver simulation. // AIP Conference Proceedings, 2021, 2351, 030037. DOI: 10.1063/5.0052098.

[102] **Губайдуллин Д. А., Федяев В. Л., Моренко И. В.** Математическое моделирование процессов, протекающих при получении пористых композитных материалов, формировании покрытий. // Физико-химическая кинетика в газовой динамике, 2017, Т. 18, № 2. URL: https://chemphys.edu.ru/issues/2017-18-2/articles/694/ (дата обращения 24.06.2026).

[103] **Медведев С. Ю., Мартынов А. А., Пошехонов Ю. Ю.** MHD_NX: код идеальной МГД-устойчивости на двумерных неструктурированных сетках. // Препринты ИПМ им. М. В. Келдыша, 2022, № 39, 22 С. DOI: 10.20948/prepr-2022-39.

[104] **Железнякова А. Л.** Эффективные методы декомпозиции неструктурированных адаптивных сеток для высокопроизводительных расчетов при решении задач вычислительной аэродинамики. // Физико-химическая кинетика в газовой динамике, 2017, Т. 18, № 1, С. 1-20. URL: http://chemphys.edu.ru/issues/2017-18-1/articles/673/ (дата обращения 24.06.2026).

[105] **Якобовский М. В.** Введение в параллельные методы решения задач. // Издательство Московского университета, 2013, 328 С. ISBN: 978-5-211-06382-2.

[106] **Головченко Е. Н., Дорофеева Е. Ю.** Разбиение графов микродоменов. // Вестник ПНИПУ, Аэрокосмическая техника, 2014, № 39, С. 35-49.

[107] **Копысов С. П., Кадыров И. Р., Новиков А. К.** Ресурсно-эффективные конечно-элементные вычисления на многоядерных архитек-

турах. // Известия Института математики и информатики Удмуртского государственного университета, 2019, Т. 53, С. 83-97. DOI: 10.20537/2226-3594-2019-53-08.

[108] **Новиков А. К., Пиминова Н. К., Копысов С. П.** Послойное разделение конечно-элементных сеток для мультиядерных архитектур. // Суперкомпьютерные дни в России, 2016, С. 493-504.

[109] **Гуличева А. А.** Распараллеливание конечно-объемных численных методов для систем с общей памятью. // ИТ-Стандарт, 2022, № 3(32), С. 33-43.

[110] **Bourgault-Côté S., Hasanzadeh K., Lavoie P., Laurendeau E.** Multi-layer methodologies for conservative ice growth. // 7th European Conference for Aeronautics and Aerospace Sciences (EUCASS), 2017. DOI: 10.13009/EUCASS2017-258.

[111] **Son C.-K., Oh S.-J., Yee K.** Prediction of glaze ice accretion on 2D airfoil. // Journal of the Korean Society for Aeronautical & Space Sciences, 2010, 38(8), P. 747-757. DOI: 10.5139/JKSAS.2010.38.8.747.

[112] **Ye H., Liu Y., Chen B., Liu Z., Zheng J., Pang Y., Chen J.** Hybrid grid generation for viscous flow simulation in complex geometries. // Advances in Aerodynamics, 2020, 2:17, 18 P. DOI: 10.1186/s42774-020-00042-x.

[113] **Thompson D., Tong X., Arnoldus Q., Collins E., McLaurin D., Luke E.** Discrete surface evolution and mesh deformation for aircraft icing applications. // in Proceedings of the 5th AIAA Atmospheric and Space Environments Conference, 2013, P. 1–20. DOI: 10.2514/6.2013-2544.

[114] **Tong X., Thompson D., Arnoldus Q., Collins E., Luke E.** Three-dimensional surface evolution and mesh deformation for aircraft icing applications. // Journal of Aircraft, 2017, Vol. 54, P. 1047–1063. DOI: 10.2514/1.C033949.

[115] **Khan D., Plopski A., Fujimoto Y., Kanbara M., Jabeen G., Zhang Y., Zhang X., Kato H.** Surface remeshing: A systematic

literature review of methods and research directions. // IEEE Transactions on Visualization Computer Graphics, 2022, 28(3), P. 1680-1713. DOI: 10.1109/TVCG.2020.3016645.

[116] **Cignoni P., Rocchini C., Scopigno R.** Metro: measuring error on simplified surfaces. // Computer Graphics Forum, 1998, Vol. 17, No. 2, P. 167–174.

[117] **Hu K., Yan D. M., Bommes D., Alliez P., Benes B.** Error-bounded and feature preserving surface remeshing with minimal angle improvement. // IEEE Trans. on Vis. and Comp. Graphics, 2017, Vol. 23, No. 12, P. 2560–2573.

[118] **Hu Y., Schneider T., Wang B., Zorin D., Panozzo D.** Fast tetrahedral meshing in the wild. // ACM Trans. Graph., 2020, Vol. 39, № 4, 18 P. DOI: 10.1145/3386569.3392385.

[119] **McLaurin D., Marcum D. L., Remotigue M., Blades E.** Repairing unstructured triangular mesh intersections. // International Journal for Numerical Methods in Engineering, 2013, 93 (3), P. 266-275. DOI: 10.1002/nme.4385.

[120] **Magalhaes S., Franklin W., Andrade M.** Fast exact parallel 3D mesh intersection algorithm using only orientation predicates. // 25th ACM SIGSPATIAL International Conference, 2017. DOI: 10.1145/3139958.3140001.

[121] **Skorkovska V., Kolingerova I., Benes B.** A Simple and robust approach to computation of meshes intersection. // in Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, 2018, Vol. 1, P. 175–182. DOI: 10.5220/0006538401750182.

[122] **Cherchi G., Livesu M., Scateni R., Attene M.** Fast and robust mesh arrangements using floating-point arithmetic. // ACM Trans. Graph., 2020, Vol. 39, № 6, 16 P. DOI: 10.1145/3414685.3417818.

[123] **Guo J.-P., Fu X.-M.** Exact and efficient intersection resolution for

mesh arrangements. // ACM Trans. Graph., 2024, Vol. 43, № 6, 14 P. DOI: 10.1145/3687925.

[124] **Liu T., Ye H., Meng X., Liu Z., Chen J.** Robust and fast local repair for intersecting triangle meshes. // Computer-Aided Design, 2026, 190, 103963. DOI: 10.1016/j.cad.2025.103963.

[125] **Окольнишников В. В.** Представление времени в имитационном моделировании. // Вычислительные технологии, 2005, Т. 10, № 5, С. 57-80. URL: https://cyberleninka.ru/article/n/predstavlenie-vremeni-v-imitatsionnom-modelirovanii (дата обращения 24.06.2026).

[126] Свидетельство о государственной регистрации программы для ЭВМ № 2023666962 «Программный модуль компьютерного моделирования процесса обледенения элементов авиационных силовых установок («Кристалл 2023»)». Авторы: **Горячев А. В., Горячев П. А., Рыбаков А. А.** Дата регистрации: 08.08.2023.

[127] Свидетельство о государственной регистрации программы для ЭВМ № 2020615575 «Программный модуль расчета процесса обледенения элементов авиационных силовых установок (КРИСТАЛЛ)». Авторы: **Горячев А. В., Горячев П. А., Рыбаков А. А.** Дата регистрации: 26.05.2020.

[128] Свидетельство о государственной регистрации программы для ЭВМ № 2021660227 «Программа подготовки неструктурированной поверхностной расчетной сетки для проведения расчетов на суперкомпьютере (crysgsu)». Авторы: **Рыбаков А. А., Фрейлехман С. А., Шумилин С. С.** Дата регистрации: 23.06.2021.

[129] Свидетельство о государственной регистрации программы для ЭВМ № 2020618596 «Программа подготовки блочно-структурированной расчетной сетки для проведения расчетов на суперкомпьютере (GridMaster)». Авторы: **Рыбаков А. А., Чопорняк А. Д.** Дата регистрации: 30.07.2020.

[130] Свидетельство о государственной регистрации программы для ЭВМ

№ 2024689396 «Программа организации вычислений на поверхностной сетке». Авторы: **Рыбаков А. А.** Дата регистрации: 05.12.2024.

[131] Свидетельство о государственной регистрации программы для ЭВМ № 2023684174 «Программа эмуляции и мониторинга эффективности векторного кода плоских циклов». Авторы: **Рыбаков А. А.** Дата регистрации: 14.11.2023.

[132] Свидетельство о государственной регистрации программы для ЭВМ № 2019665254 «Векторизованная с помощью набора инструкций AVX-512 программа реализации решения задачи Римана о распаде произвольного разрыва». Авторы: **Рыбаков А. А., Шумилин С. С.** Дата регистрации: 21.11.2019.

[133] Свидетельство о государственной регистрации программы для ЭВМ № 2026663702 «Программа управления данными мультифизических решателей в суперкомпьютерной среде». Авторы: **Рыбаков А. А., Цынгалев П. С.** Дата регистрации: 15.05.2026.

[134] **Wei Z., Qian K., Li Y., Li S., Zhang J. et al.** GPU-accelerated optimization of discrete Ricci flow for high-resolution triangular meshes. // IET Image Processing, 2025, 19:e70237. DOI: 10.1049/ipr2.70237.

[135] **Wang Y.** Research and implementation of isosurface extraction algorithm based on Flexicubes. // Highlights in Science, Engineering and Technology, 2024, Vol. 83, P. 334-345. DOI: 10.54097/z6v6dd54.

[136] **Сухомлин В. А., Горькавый И. Н.** Технологическая система для построения программных комплексов автоматизации обработки трехмерных данных лазерного сканирования. // Информатика и ее применения, 2009, Т. 3, № 2, С. 53-64. URL: https://www.mathnet.ru/rus/ia61 (дата обращения 24.06.2026).

[137] **Garimella R. V., Kim J., Berndt M.** Polyhedral mesh generation and optimization for non-manifold domains. // In: Sarrate J., Staten M. (eds)

Proceedings of the 22nd International Meshing Roundtable, Springer, Cham. DOI: 10.1007/978-3-319-02335-9_18.

[138] **Гаранжа В. А., Кудрявцева Л. Н., Цветкова В. О.** Построение гибридных расчетных сеток Вороного. Алгоритмы и нерешенные проблемы. // Журнал вычислительной математики и математической физики, 2019, Т. 59, № 12, С. 2024-2044. DOI: 10.1134/S004446691912007X.

[139] **Василевский Ю. В., Липников К. Н.** Адаптивный алгоритм построения квазиоптимальных сеток. // Журнал вычислительной математики и математической физики, 1999, Т. 39, № 9, С. 1532-1551. URL: https://www.mathnet.ru/rus/zvmmf1617 (дата обращения 24.06.2026).

[140] **Peskin C.** The immersed boundary method. // Acta Numerica, 2002, № 11, P. 479-517. DOI: 10.1017/S0962492902000077.

[141] **Баранов А. В., Киселев Е. А., Телегин П. Н., Сорокин А. А.** Программное средство GraphHunter поиска отображения параллельной программы на структуру суперкомпьютерной системы. // Программные продукты и системы, 2022, Т. 35, № 4, С. 583–597. DOI: 10.15827/0236-235X.140.583-597.

[142] **Киселев Е. А., Аладышев О. С.** Алгоритм эффективного размещения программ на ресурсах многопроцессорных вычислительных систем. // Программные продукты и системы, 2012, № 4. С. 18-25.

[143] **Старостин Н. В., Панкратова М. А.** Генетический алгоритм решения задачи отображения графа. // Вестник ННГУ, 2013, №5 (1), С. 204-209. URL: https://cyberleninka.ru/article/n/geneticheskiy-algoritm-resheniya-zadachi-otobrazheniya-grafa (дата обращения: 24.06.2026).

[144] **Тарков М. С.** Отображение параллельных программ на многоядерные компьютеры рекуррентными нейронными сетями. // Прикладная дискретная математика, 2013, № 2 (20), С. 50-58. URL: https://cyberleninka.ru/article/n/otobrazhenie-parallelnyh-programm-

na-mnogoyadernye-kompyutery-rekurrentnymi-neyronnymi-setyami (дата обращения: 24.06.2026).

[145] **Воеводин Вл. В.** Методы анализа локальности данных в задаче эффективного отображения программ и алгоритмов на архитектуру вычислительных систем. // Диссертация на соискание ученой степени кандидата физико-математических наук, 2011, Москва.

[146] **Пленкин А. В., Чернышенко А. Ю., Чугунов В. Н., Капырин И. В.** Методы построения адаптивных неструктурированных сеток для решения гидрогеологических задач. // Вычислительные методы и программирование, 2015, Т. 16, С. 518-533.

[147] **Брагин М. Д., Рогов Б. В.** Бикомпактные схемы для многомерных уравнений гиперболического типа на декартовых сетках с адаптацией к решению. // Препринты ИПМ им. М. В. Келдыша, 2019, № 11, 27 С. DOI: 10.20948/prepr-2019-11.

[148] **Борисов В. Е., Кулешов А. А., Савенков Е. Б., Якуш С. Е.** Программный комплекс TCS 3D: вычислительная модель. // Препринты ИПМ им. М. В. Келдыша, 2015, № 110, 20 С. URL: http://library.keldysh.ru/preprint.asp?id=2015-110 (дата обращения 24.06.2026).

[149] **Ермаков М. К.** Генерация тетраэдальных сеток для суперкомпьютерного моделирования обтекания аэрокосмических объектов. // Вычислительные методы и программирование, 2020, Т. 21, С. 341-349. DOI: 10.26089/NumMet.v21r429.

[150] **Fan H., Xu Q., Bai F., Wei Z., Zhang Y., Lu X., Wei N., Zhang S., Yuan H., Liu S., Li X., Li X., Dai Y.** An unstructured mesh generation tool for efficient high-resolution representation of spatial heterogeneity in land surface models. // Geophysical Research Letters, 51, e2023GL107059. DOI: 10.1029/2023GL107059.

[151] **Евстифеева Е. О.** Алгоритм упрощения поверхностной треугольной

сетки при подготовке задач аэрогидродинамики в пакете программ Логос. // Молодежь в науке, Сборник докладов 19-й научно-технической конференции, Саров, 2022. DOI: 10.53403/9785951505200_63.

[152] **Арзуманян Р. В.** Применение кривой Гильберта для обхода точек расчетной области в задачах обработки изображений. // Инженерный вестник Дона, 2015, № 4.

[153] **Farhat C.** A simple and efficient automatic fem domain decomposer. // Computers & Structures, Vol. 28, No. 5, P. 579-602.

[154] **Якобовский М. В.** Инкрементный алгоритм декомпозиции графов. // Вестник Нижегородского университета им. Н. И. Лобачевского, серия Математическое моделирование и управление, Вып. 1(28), С. 243-250.

[155] **Kernighan B. W., Lin S.** An efficient heuristic procedure for partitioning graphs. // The Bell System Technical Journal, 1970, Vol. 49, No. 2, P. 291-307. DOI: 10.1002/j.1538-7305.1970.tb01770.x.

[156] **Fiduccia C. M., Mattheyses R. M.** A linear-time heuristic for improving network partitions. // 19th Design Automation Conference, Las Vegas, USA, 1982, P. 175-181. DOI: 10.1109/DAC.1982.1585498.

[157] **Берцун В. Н.** Математическое моделирование на графах. Часть II. // Издательство Томского университета, 2013, 88 С. ISBN: 978-5-7511-2211-9.

[158] **Якобовский М. В.** Вычислительный эксперимент на многопроцессорных системах: алгоритмы и инструменты. // Приборостроение, 2009, № 10. URL: https://cyberleninka.ru/article/n/vychislitelnyy-eksperiment-na-mnogoprotsessornyh-sistemah-algoritmy-i-instrumenty (дата обращения 24.06.2026).

[159] **Hendrickson B.** Chaco. In: Padua D. (eds) Encyclopedia of parallel computing. // Springer, Boston, 2011. DOI: 10.1007/978-0-387-09766-4_310.

[160] **Karypis G., Schloegel K.** ParMETIS. Parallel graph partitioning and sparse matrix ordering library. Version 4.0. // University of Minnesota, Department of Computer Science and Engineering, 2013.

[161] **McManus K., Walshaw C., Cross M., Leggett P., Johnson S.** Evaluation of the JOSTLE mesh partitioning code for prectical multiphysics applications. // Parallel Computational Fluid Dynamics, 1996. DOI: 10.1016/B978-044482322-9/50137-7.

[162] **Pellegrini F., Roman J.** Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. // In Liddell H., Colbrook A., Herzberger B., Sloot P. (eds) High-Performance Computing and Networking, HPCN-Europe, 1996, Lecture Notes in Computer Science, Vol. 1067, Springer. DOI: 10.1007/3-540-61142-8_588.

[163] **Воропинов А. А.** Декомпозиция данных для распараллеливания методики ТИМ-2D и критерии оценки ее качества. // Вестник ЮУрГУ, серия: Математическое моделирование и программирование, 2009, № 37(170). URL: https://cyberleninka.ru/article/n/dekompozitsiya-dannyh-dlya-rasparallelivaniya-metodiki-tim-2d-i-kriterii-otsenki-ee-kachestva (дата обращения 24.06.2026).

[164] **Головченко Е. Н., Якобовский М. В.** Пакет параллельной декомпозиции больших сеток GridSpiderPar. // Вычислительные методы и программирование, 2015, Т. 16, С. 507-517. DOI: 10.26089/NumMet.v16r448.

[165] **Акимова Е. Н., Мисилов В. Е.** Параллельные вычисления на суперкомпьютерах с помощью технологии OpenMP. // Учебное пособие, Издательство Уральского университета, 2023, 104 С. ISBN: 978-5-7996-3629-6.

[166] **Medeiros E., Siqueira M.** Good random multi-triangulation of surfaces. // IEEE Trans. on Vis. and Comp. Graphics, 2018, Vol. 24, No. 6, P. 1983–1996.

[167] **Huang X., Xu D.** Aspect-ratio based triangular mesh smoothing. // in ACM SIGGRAPH 2017 Posters, 2017, P. 68:1–68:2.

[168] **Chiang C.-H., Jong B.-S., Lin T.-W.** A robust feature-preserving semi-regular remeshing method for triangular meshes. // The Visual Computer, 2011, Vol. 27, No. 9, P. 811–825. DOI: 10.1007/s00371-011-0555-1.

[169] **Ma X., Zheng J., Shui Y., Zhou H., Shen L.** A novel method of mesh simplification using Hausdorff distance. // in 2012 Third World Congress on Software Engineering, 2012, P. 136–139.

[170] **Xiao Z., Chen J., Zheng Y., Zheng J., Wang D.** Booleans of triangulated solids by a boundary conforming tetrahedral mesh generation approach. // Computers & Graphics, 2016, 59, P. 13-27. DOI: 10.1016/j.cag.2016.04.004.

[171] **Hu Y., Zhou Q., Gao X., Jacobson A., Zorin D., Panozzo D.** Tetrahedral meshing in the wild. // ACM Trans. Graph., 2018, Vol. 37, № 4, 14 P. DOI: 10.1145/3197517.3201353.

[172] **Lo. S. H., Wang W. X.** A fast robust algorithm for the intersection of triangulated surfaces. // Engineering with Computers, 2004, 20, 11-21. DOI: 10.1007/s00366-004-0277-3.

[173] **Cebrian J., Natvig L., Lahre M.** Scalability analysis of AVX-512 extensions. // The Journal of Supercomputing, 2020, Vol. 76, P. 2082-2097. DOI: 10.1007/s11227-019-02840-7.

[174] Intel 64 and IA-32 architectures software developer's manual. Combined volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4. // Order number: 325462-087US, march 2025.

[175] **Жуйков Р. А., Плотников Д. В., Варданян М. А.** Автоматическая настройка оптимизационных преобразований компилятора GCC для платформы arm. // Труды ИСП РАН, 2012. URL: https://cyberleninka.ru/article/n/avtomaticheskaya-nastroyka-optimizatsionnyh-preobrazovaniy-kompilyatora-gcc-dlya-platformy-arm (дата обращения 24.06.2026).

[176] **Stephens N., Biles S., Boetthcher M., Eapen J.** The ARM Scalable Vector Extension. // IEEE Micro, 2017, Vol. 37, Issue 2, P. 26-39. DOI: 10.1109/MM.2017.35.

[177] **Okazaki R., Tabata T., Sakashita S., Kitamura K. et al.**

Supercomputer Fugaku CPU A64FX realizing high performance, high-density packaging, and low power consumption. // Fujitsu Technical Review, 2020. URL: https://www.fujitsu.com/global/about/resources/publications/technicalreview/2020-03/article03.html?ysclid=mbawuuc4vr289719768 (дата обращения 24.06.2026).

[178] **Eisen L., Ward III J., Tast H.-W., Mäding N. et al.** IBM Power6 accelerators: VMX and DFU. // IBM Journal of Research and Development, 2007, 51(6), P. 1-21. DOI: 10.1147/rd.516.0663.

[179] **Ишин П. А.** Оптимизация преобразования Фурье под архитектуру Эльбрус. // Современные информационные технологии и ИТ-образование, 2011 № 7. URL: https://cyberleninka.ru/article/n/optimizatsiya-preobrazovaniya-furie-pod-arhitekturu-elbrus (дата обращения: 24.06.2026).

[180] **Волконский В. Ю., Брегер А. В., Бучнев А. Ю., Грабежной А. В. и др.** Методы распараллеливания программ в оптимизирующем компиляторе. // Вопросы радиоэлектроники, 2012, Т. 4, № 3, С. 63-88.

[181] **Bai Z., Wu Q., Cao K., Sun Y. et al.** Application of regional meteorology and air quality models based on the microprocessor without interlocked piped stages (MIPS) and LoongArch CPU platforms. // Geoscientific Model Development, 2024, Vol. 17(10), P. 4383-4399. DOI: 10.5194/gmd-17-4383-2024.

[182] **Sun Z., Wang Z., Hua M., Xiong P. et al.** Accelerating ray tracing engine of BLENDER on the new Sunway architecture. // Engineering Reports, 2025, 7:e12789. DOI: 10.1002/eng2.12789.

[183] Intel Intrinsics Guide, https://alouettesu.github.io/Intrinsics/ (дата обращения 24.06.2026).

[184] **Jeffers J., Reinders J., Sodani A.** Intel Xeon Phi processor high performance programming: Knights Landing edition. // Morgan Kaufmann, 2016, 662 P.

[185] **Kulikov I., Chernykh I., Tutukov A.** A new hydrodynamic code with explicit vectorization instructions optimizations that is dedicated to the numerical simulation of astrophysical gas flow. I. Numerical method, tests, and model problem. // The Astrophysical Journal Supplement Series, 2019, Vol. 243, No. 4, 15 P. DOI: 10.3847/1538-4365/ab2237.

[186] **Glinting B., Mundani R.-P.** Comparison of shallow water solvers: applications for dam-break and tsunami cases with reordering strategy for efficient vectorization on modern hardware. // Water, 2019, Vol. 11(4), No. 639, 31 P. DOI: 10.3390/w11040639.

[187] **Yildirim A., Mader C., Martins J.** Accelerating parallel CFD codes on modern vector processors using blockettes. // PASC'21: Proceedings of the Platform for Advanced Scientific Computing Conference, 2021. DOI: 10.1145/3468267.3470615.

[188] **Rucci E., Moreno E., Pousa A., Chichizola F.** Optimization of the N-body simulation on Intel's architectures based on AVX-512 instruction set. // In book: Comminications in Computer and Information Science, 2020. DOI: 10.1007/978-3-030-48325-8_3.

[189] **Rucci E., Garcia C., Botella G., De Giusti A.** SWIMM 2.0: Enhanced Smith-Waterman on Intel's multicore and manycore architectures based on AVX-512 vector extensions. // International Journal of Parallel Programming, 2019, Vol. 47(17). DOI: 10.1007/s10766-018-0585-7.

[190] **Blacher M., Giesen J., Sanders P., Wassenberg J.** Vectorized and performance-portable Quicksort. // ArXiv, 2022, art. 2205.05982, P. 1–21. DOI: 10.48550/arXiv.2205.05982.

[191] **Long S., Fan X., Chao L., Yi L., Fan S., Guo X.-W., Yang C.** VecDualSPHysics: A vectorized implementation of Smoothed Particle Hydrodynamics method for simulating fluid flows on multi-core processors. // Journal of Computational Physics, 2022, Vol. 463, art. 111234. DOI: 10.1016/j.jcp.2022.111234.

[192] **Ponte-Fernández C., González-Domínguez J., Martín M. J.** A SIMD algorithm for the detection of epistatic interactions of any order. // Future Generation Comput. Sys., 2022, Vol. 132, P. 108–123. DOI: 10.1016/j.future.2022.02.009.

[193] **Quislant R., Fernandez I.** Time series analysis acceleration with advanced vectorization extensions. // The Journal of Supercomputing, 2023, Vol. 79, No. 9, P. 10178–10207. DOI: 10.1007/s11227-023-05060-2.

[194] **Buhrow B., Gilbert B., Haider C.** Parallel modular multiplication using 512-bit advanced vector instructions. // Journal of Cryptographic Engineering, 2022, Vol. 12, P. 95–105. DOI: 10.1007/s13389-021-00256-9.

[195] **Choi H., Seo S. C.** Efficient parallel implementations of PIPO Block Cipher on CPU and GPU. // IEEE Access, 2022, Vol. 10, P. 85995–86007. DOI: 10.1109/ACCESS.2022.3198707.

[196] **Sansone G., Cococcioni M.** Experiments on speeding up the recursive fast Fourier transform by using AVX-512 SIMD instructions. // Proc. ApplePies. LNEE, 2023, Vol. 1036, P. 255–263. DOI: 10.1007/978-3-031-30333-3_34.

[197] **Edamatsu T., Takahashi D.** Fast multiple-precision integer division using Intel AVX-512. // IEEE Transactions on Emerging Topics in Computing, 2023, Vol. 11, No. 1, P. 224–236. DOI: 10.1109/TETC.2022.3196147.

[198] **Медакин П. О., Никулин Р. Н., Авдеюк О. А., Королева И. Ю., Павлова Е. С., Лемешкина И. Г.** Векторизация и распараллеливание метода «частица-частица». // Инженерный вестник Дона, 2021, No. 1. URL: http://ivdon.ru/ru/magazine/archive/n1y2021/6800 (дата обращения: 24.06.2026).

[199] **Tayeb H., Paillat L., Bramas B.** Autovesk: Automatic vectorization of unstructured static kernels by graph transformations. // ArXiv, 2023, art. 2301.01018, P. 1–21. DOI: 10.48550/arXiv.2301.01018.

[200] **Laukemann J., Hammer J., Hager G., Wellein G.** Automatic

throughput and critical path analysis of x86 and ARM assembly kernels. // Proc. IEEE/ACM PMBS, 2019, P. 1–6. DOI: 10.1109/PMBS49563.2019.00006.

[201] **Kusswurm D.** Modern parallel programming with C++ and Assembly Language. X86 SIMD development using AVX, AVX2, and AVX-512. // CA, Apress Berkeley Publ., 2022, 633 P. DOI: 10.1007/978-1-4842-7918-2.

[202] **Lock C., Hassan O., Sevilla R., Jones J.** Meshing using neural networks for improving the efficiency of computer modelling. // Engineering with Computers, 2023, Vol. 39, P. 3791-3820. DOI: 10.1007/s00366-023-01812-z.

[203] **Chen X., Li T., Wan Q., He X., Gong C., Pang Y., Liu J.** MGNet: a novel differential mesh generation method based on unsupervised neural networks. // Engineering with Computers, 2022, Vol. 38. P. 4409-4421. DOI: 10.1007/s00366-022-01632-7.

[204] **Wang Z., Lorraine J., Wang Y., Su H., Zhu J., Fidler S., Zeng X.** LLaMA-Mesh: unifying 3D mesh generation with language models. // ArXiv, 2024, art. 2411.09595, P. 1-11. DOI: 10.48550/arXiv.2411.09595.

[205] **Fang S., Shen I-C., Wang Y., Tsai Y.-H. et al.** MeshLLM: empowering large language models to progressively understand and generate 3D mesh. // ArXiv, 2025, art. 2508.01242, P. 1-15. DOI: 10.48550/arXiv.2508.01242.

[206] **Ищенко А. В., Молоткова П. А.** Способы определения и устранения дефектов аддитивного строительства с применением искусственного интеллекта. // Вестник Евразийской Науки, 2024, Т. 16, № 6, 10 С. URL: https://esj.today/PDF/41SAVN624.pdf (дата обращения: 24.06.2026).

[207] **Князева Н. В., Назойкин Е. А., Орехов А. А.** Применение искусственного интеллекта для обнаружения дефектов в строительных конструкциях. // Строительство и архитектура, 2023, № 3. DOI: 10.29039/2308-0191-2023-11-3-18-18.

[208] **Sang W., Lin M., Yue Y., Zhai K.** Research on 3D reconstruction method of damaged object based on neural network. // In: Song H., Xu M.,

Yang L. (eds) Innovative Technologies for Printing, Packaging and Digital Media, CACPP 2023, 2024, Vol. 1144. DOI: 10.1007/978-981-99-9955-2_15.

[209] **Shumilin S.** Learning ice accretion with graph neural networks. // Lobachevskii Journal of Mathematics, 2022, Vol. 43, № 10, P. 2887-2892. DOI: 10.1134/S1995080222130418.

[210] **Ryabov A., Naumov V., Shumilin S., Yavich N. et al.** Learnable stability-aware computational grid coarsening for accelerating physics simulations. // Machine Learning: Science and Technology, 2026, 7, 015019. DOI: 10.1088/2632-2153/ae3429.

[211] **Rojek K., Wyrzykowski R., Gepner P.** AI-accelerated CFD simulation based on OpenFOAM and CPU/GPU computing. // In: Paszynski M., Kranzlmüller D., Krzhizhanovskaya V. V., Dongarra J. J., Sloot P. M. A. (eds) Computational Science – ICCS 2021, 2021, Vol. 12743. DOI: 10.1007/978-3-030-77964-1_29.

[212] **Конюхов И. В., Конюхов В. М., Черница А. А., Дюсенова А.** Особенности применения физически информированных нейронных сетей для решения обыкновенных дифференциальных уравнений. // Компьютерные исследования и моделирование, 2024, Т. 2024, № 7, С. 1621-1636. DOI: 10.20537/2076-7633-2024-16-7-1621-1636.

[213] **Кошелев К. Б., Стрижак С. В.** Применение физически-обоснованной нейронной сети на примере моделирования гидродинамических процессов, допускающих аналитическое решение. // Труды ИСП РАН, Т. 35, Вып. 5, С. 245-258. DOI: 10.15514/ISPRAS-2023-35(5)-16.

[214] **Somasekharan N., Yue L., Cao Y., Li W., Emami P., Bhargav P. S. et al.** CFDLLMBench: a benchmark suite for evaluating large language models in computational fluid dynamics. // ArXiv, 2025, art. 2509.20374, P. 1-28. DOI: 10.48550/arXiv.2509.20374.

[215] **Jiang P., Zhou Q., Shao X.** Surrogate model-based engineering design and optimization. // 2020, Springer, 246 P. DOI: 10.1007/978-981-15-0731-1.

[216] **Barcenas O., Pioquinto J., Kurkina E., Lukyanov O.** Surrogate aerodynamic wing modeling based on a multilayer perceptron. // Aerospace, 2023, Vol. 10, 149, 19 P. DOI: 10.3390/aerospace10020149.

[217] **Catalani G., Agarwal S., Bertrand X., Tost F., Bauerheim M., Morlier J.** Aero-Nef: neural fields for rapid aircraft aerodynamics simulations. // arXiv, 2024. DOI: 10.48550/arXiv.2407.19916.

[218] **Jimènes À. B.** An evaluation of LLM code generation capabilities through graded exercises. // ArXiv, 2024, art. 2410.16292, P. 1-22. DOI: 10.48550/arXiv.2410.16292.

[219] **Jiang J., Wang F., Shen J., Kim S., Kim S.** A survey on large language models for code generation. // ArXiv, 2024, art. 2406.00515, P. 1-70. DOI: 10.48550/arXiv.2406.00515.

[220] **Taneja J., Laird A., Yan C., Musuvathi M., Lahiri S. K.** LLM-Vectorizer: LMM-based verified loop vectorizer. // ArXiv, 2024, art. 2406.04693, P. 1-12. DOI: 10.48550/arXiv.240604693.

[221] **Raj L., Yee K., Myong R.** Sensivity of ice and aerodynamic performance degradation to critical physical and modeling parameters affecting airfoil icing. // Aerospace Science and Technology, 2020, 98, 105659, P. 1–46. DOI: 10.1016/j.ast.2019.105659.

[222] **Cui X., Habashi W.** A dendritic freezing model for in-flight supercooled large droplets impingement and solidification. // Computers & Fluids, 2023, Vol. 254(2), 105778. DOI: 10.1016/j.compfluid.2023.105778.

[223] **Cui X., Habashi W.** SPH simulation of supercooled large droplets impacting hydrophobic and superhydrophobic surfaces. // Computers & Fluids, 2021. DOI: 10.1016/j.compfluid.2021.105055.

[224] **Алексеенко С. В., Приходько А. А.** Численное моделирование обледенения цилиндра и профиля. Обзор моделей и результатов расчетов. // Ученые записки ЦАГИ, 2013, Т. XLIV, № 6, С.25-57.

[225] **Li X., Bai J., Hua J., Wang K., Zhang Y.** A spongy icing model for

aircraft icing. // Chineese Journal of Aeronautics, 2014, Vol. 27(1), P. 40-51. DOI: 10.1016/j.cja.2013.12.004.

[226] **Bartkus T., Struk P., Tsao J.-C.** Evaluation of a thermodynamic ice-crystal icing model using experimental ice accretion data. // in Proceedings of the Atmospheric and Space Environments Conference, 2018, P. 1-18. DOI: 10.2514/6.2018-4129.

[227] **Zhang X., Wu X., Min J.** Aircraft icing model considering both rime ice property variability and runback water effect. // International Journal of Heat and Mass Transfer, 2017, 104, P. 510–516. DOI: 10.1016/j.ijheatmasstransfer.2016.08.086.

[228] **Pena D., Hoarau Y., Laurendeau E.** A single step ice accretion model using level-set method. // Journal of Fluids and Structures, 2016, 65, P. 278–294. DOI: 10.1016/j.jfluidstructs.2016.06.001.

[229] **Wang Z., Zhong W., Liu C., Zhao H., Liu S.** Numerical simulation of ice crystal supercooled droplet mixed phase icing based on the improved Messinger model. // International Journal of Aerospace Engineering, 2023, 1776, P. 1–12. DOI: 10.1155/2023/6696084.

[230] **Liu Y., Wang T., Song Z., Chen M.** Spreading and freezing of supercooled water droplets impacting an ice surface. // Applied Surface Science, 2022, 583(1): 152374. DOI: 10.1016/j.apsusc.2021.152374.

[231] **Ruan Q.** Analysis of the types of aircraft icing and the solutions. // Applied and Computational Engineering, 2023, 9(1), P. 177–181. DOI: 10.54254/2755-2721/9/20230082.

[232] **Ignatowicz K., Morency F., Beaugendre H.** Numerical simulation of ice accretion using Messinger-based approach: effects of surface roughness. // CASI Aero 2019, Canadian Aeronautics and Space Institute's AERO 2019 Conference. URL: https://inria.hal.science/hal-02409011v1 (дата обращения 24.06.2026).

[233] **Тарасов А. А., Степанова М. М.** Моделирование донного таяния

антарктического ледяного щита на основе одномерной задачи Стефана. // Физика элементарных частиц и атомного ядра, 2024, Т. 55, Вып. 3, С. 638-643.

[234] **Martini F., Ibrahim H., Montoya L., Rizk P., Ilinca A.** Turbulence modeling of iced wind turbine airfoils. // Energies, 2022, 15, 8325, P. 1–21. DOI: 10.3390/en15228325.

[235] **Калюнин С. Л., Бабушкина А. В., Серегина М. А.** Численное моделирование работы электротепловой противообледенительной системы беспилотной авиационной системы. // Вестник Московского авиационного института, 2025, Т. 32, № 2, С. 77-85. URL: https://vestnikmai.ru/publications.php?ID=184993 (дата обращения 24.06.2026).

[236] **Алексанин Н. Д., Ефременков И. В.** Моделирование обледенения приемника воздушных давлений с помощью программного комплекса FENSAP-ICE. // Ученые записки УлГУ, Серия Математика и информационные технологии, 2020, № 1, С. 1-5. URL: https://ulsu.ru/ru/page/page_3946/ (дата обращения 24.06.2026).

[237] **Galanov N. G., Sarazov A. V., Zhuchkov R. N., Kozelkov A. S.** Application of various ice accretion simulation approaches in the LOGOS software package. // Journal of Physics: Conference Series, 2021, 2099, 012029, P. 1–8. DOI: 10.1088/1742-6596/2099/1/012029.

[238] **Галанов Н.Г., Козелков А. С., Жучков Р. Н., Саразов А. В.** Тестирование методики моделирования процесса обледенения в пакете программ ЛОГОС. // Информационные системы и технологии ИСТ-2021, сборник материалов XXVII Международной научно-технической конференции, 2021, С. 848-854.

[239] **Shannon T., McClain S.** As assessment of LEWICE roughness and convection enhancement models. // SAE International Journal of Advances and Current Practices in Mobility, 2(1). DOI: 10.4271/2019-01-1977.

[240] **Domingos R., Silva D.** 3D computational methodology for bleed air ice protection system parametric analysis. // SAE Technical Paper 2015-01-2109, 2015. DOI: 10.4271/2015-01-2109.

[241] **Villedieu P., Trontin P., Chauvin R.** Glaciated and mixed phase ice accretion modeling using ONERA 2D icing suite. // Transactions of Japanese Society for Medical and Biological Engineering, 51. DOI: 10.2514/6.2014-2199.

[242] Modeling airflow for aircraft icing using FENSAP (NACA 0012 tutorial). // SimuTech Group. URL: https://simutechgroup.com/modeling-airflow-for-aircraft-icing-using-fensap-naca-0012-tutorial/ (дата обращения 24.06.2026).

[243] **Al Tahhan A. B., Ramahi A., Dol S. S., Ahmad K. A.** Simulation study on ice accretion over NACA0012 airfoil under varying airflow conditions. // Journal of Advanced Research in Numerical Heat Transfer, 2025, Vol. 29, Issue 1, P. 62-85. DOI: 10.37934/arnht.29.1.6285.

[244] **Martini F., Montoya L. T. C., Ilinca A., Awada A.** Review of studies on the CFD-BEM approach for estimating power losses of iced-up wind turbines. // International Journal of Advanced Research (IJAR), 2021, 9(09), P. 633-652. DOI: 10.21474/IJAR01/13462.

[245] **Sahu A., Omar Y. A. D., Sauer R. A., Mandadapu K. K.** Arbitrary Lagrangian-Eulerian finite element method for curved and deforming surfaces. // ArXiv, 2020, art. 1812.05086, P. 1-59. DOI: 10.48550/arXiv.1812.05086.

[246] **Lepage C. Y., Habashi W. G.** Fluid-structure interactions using the ALE formulation. // 37th AIAA Aerospace Sciences Meeting and Exhibit, January 11-14, 1999, Reno, NV. DOI: 10.2514/6.1999-660.

[247] **Habashi W. G., Dompierre J., Bourgault Y.** Certifiable computational fluid dynamics through mesh optimization. // AIAA Journal, 1998, Vol. 36, No. 5. DOI: 10.2514/2.458.

[248] **Ozcer I., Moula G., Norman A., Pons D.** Ansys in-flight icing simulation methodology overview. // ANSYS Inc., 2023. URL: https://richahan.folk.ntnu.no/IPW/files/IPW2/Presentations/

IPW2_Presentations/0950_Ozcer_IPW2-Ansys-Vienna.pdf (дата обращения 24.06.2026).

[249] **Kobbelt L. P., Vorsatz J., Labsik U., Seidel H.-P.** A shrink wrapping approach to remeshing polygonal surfacs. // EUROGRAPHICS'99, 1999, Vol. 18, № 3, P. 119-130. DOI: 10.1111/1467-8659.00333.

[250] **Koshelev K. B., Osipov A. V., Strijhak S. V.** Using the iceFoam solver to simulate ice accretion process on a swept wing with GLC-305 airfoil. // AIP Conference Proceedings, 2023, 2504, 030028. DOI: 10.1063/5.0132710.

[251] **Koshelev K., Osipov A., Strijhak S., Tryaskin N.** Mathematical modeling of icing process of the outer surface of the hull for a marine vessel. // Journal of Hydrodynamics, 2023, 35(2), P. 232-239. DOI: 10.1007/s42241-023-0027-x.

[252] **Bourgault Y., Beaugendre H., Habashi W. G.** Development of a shallow-water icing model in FENSAP-ICE. // Journal of Aircraft, 2000, Vol. 37, No. 4, P. 640-646. DOI: 10.2514/2.2646.

[253] **Стрижак С.** Разработка эффективного алгоритма параллельных вычислений для моделирования процесса обледенения стреловидного крыла и использованием решателя iceFoam. // Сборник тезисов II Международной конференции «Математическое моделирование», 2021, С. 79-81.

[254] **Галанов Н. Г., Козелков А. С., Саразов А. В.** Улучшение методов численного моделирования процессов обледенения в ПП «Логос». // Материалы конференции "Новые горизонты прикладной математики: тезисы II Российской молодежной научной конференции", Москва, 2025, С. 50-50. URL: https://keldysh.ru/ngpm/2025/18.pdf (дата обращения 24.06.2026).

[255] **Саразов А. В., Жучков Р. Н.** Разработка методики моделирования процесса образования инея в пакете программ ЛОГОС. // Сборник трудов конференции «Супервычисления и математическое моделирование», Саров, 2019, С. 480-489. DOI: 10.53403/9785951504418_480.

[256] **Aksenov A. A., Byvaltsev P. M., Zhluktov S. V., Sorokin K. E.,**

**Babulin A. A., Shevyakov V. I.** Numerical simulation of ice accretion on airplane surface. // AIP Conference Proceedings 2125, 2019, 020001. DOI: 10.1063/1.5117361.

[257] **Сорокин К. Э., Аксенов А. А., Жлуктов С. В., Бабулин А. А., Шевяков В. И.** Методика расчета обледенения воздушных судов в широком диапазоне климатических и скоростных параметров. Применение в рамках норм летной годности НЛГ-25. // Компьютерные исследования и моделирование, 2023, Т. 15, № 4, С. 957-978. DOI: 10.20537/2076-7633-2023-15-4-957-978.

[258] **Карасев П. И., Шишаева А. С., Аксенов А. А.** Качественное построение расчетной сетки для решения задач аэродинамики в программном комплексе FlowVision. // Вестник ЮУрГУ, 2012, № 47(306), С. 46-58. DOI: 10.14529/cmse120205.

[259] **Бендерский Л. А., Иванов В. С., Любимов Д. А., Фролов С. М.** Моделирование стабилизации горения в турбулентном потоке с помощью программного модуля Лазурит. // Тезисы доклада 11-й Международного симпозиума по неравномерным процессам, плазме, горению и атмосферным явлениям “Неравновесные процессы”, 2024, г. Сочи.

[260] **Горячев А. В., Горячев П. А., Горячев Д. А., Любимов Д. А., Николаев А. А., Нуриев М. В., Скрипкин Р. В.** Моделирование процесса формирования льда на обогреваемых поверхностях летательных аппаратов. // Известия РАН. Механика жидкости и газа, 2025, № 4, С. 134-148. DOI: 10.7868/S3034534025040121.

[261] **Горячев А. В., Горячев П. А., Горячев Д. А., Нуриев М. В., Николаев А. А., Рыбаков А. А.** Численное моделирование процесса обледенения в условиях крупных переохлажденных капель. // Программные системы: Теория и приложения, 2025, Т. 16, № 4(67), С. 287-317, DOI: 10.25209/2079-3316-2025-16-4-287-317.

[262] **Горячев П. А., Горячев Д. А., Нуриев М. В., Николаев А. А.**

Модель расчета форм ледяных наростов, образующихся на элементах авиационной техники при полетах условиях обледенения. // Всероссийский научно-технический форум по двигателям и энергетическим установкам имени Н. Д. Кузнецова, сб. докд. всерос. науч.-техн. форума 10-11 окт. 2024, Самара: Изд-во Самар. ун-та, 2024, С. 61-63. URL: http://repo.ssau.ru/jspui/handle/123456789/35167 (дата обращения 24.06.2026).

[263] **Горячев П. А., Бурцев С. А.** Математическая модель формирования льда в атмосферных условиях, характеризующихся наличием ледяных кристаллов и смеси фаз. // Теплофизика высоких температур, 2025, Т. 63, № 1, С. 91-99. DOI: 10.31857/S0040364425010142.

[264] **Meshcheryakov A., Rybakov A.** Evolution of the surface computational mesh in the ice accretion process. // Lobachevskii Journal of Mathematics, 2023, Vol. 44, No. 11, P. 361-378. DOI: 10.1134/S1995080223110367.

[265] **Bagrov A. D., Rybakov A. A.** Selection of a method for solving nonlinear equations in shallow-water icing model implementation. // Lobachevskii Journal of Mathematics, 2021, Vol. 42, № 11, P. 2503-2509. DOI: 10.1134/S1995080221110068.

[266] **Anderson F.** Huygens' Principle geometric derivation and elimination of the wave and backward wave. // Scientific Reports, 2021, 11, 20257. DOI: 10.1038/s41598-021-99049-7.

[267] **Fortin G., Ilinca A., Laforte J.-L., Brandi V.** New roughness computation method and geometric accretion model for airfoil icing. // Journal of Aircraft, 20024, Vol. 41, P. 119–127. DOI: 10.2514/1.173.

[268] **Rybakov A. A, Shumilin S. S.** Approximate methods of the surface mesh deformation in two-dimensional cases. // Lobachevskii Journal of Mathematics, 2019, Vol. 40, No. 11, P. 1848-1852. DOI: 10.1134/S1995080219110258.

[269] **Канторович Л. В., Акилов Г. П.** Функциональный анализ. // Москва «Наука», 1984, 752 С.

[270] **Beaugendre H.**A PDE-based approach to in-flight ice accretion. // PhD Thesis (Dep. of Mech. Eng., McGill Univ., Montreal, Québec, 2003).

[271] **Jiao X.** Face offsetting: A unified approach for explicit moving interfaces. // Journal of Computational Physics, 2007, Vol. 220. P. 612–625. DOI: 10.1016/j.jcp.2006.05.021.

[272] **Jiao X.** Volume and feature preservation in surface mesh optimization. // in Proceedings of the 15th International Meshing Roundtable, 2006, P. 359–373. DOI: 10.1007/978-3-540-34958-7_21.

[273] **Шумилин С. С.** Методы закрепления граничных узлов при сглаживании треугольной поверхностной сетки. // Программные системы: Теория и приложения, 2021, Т. 12, № 2(49), С. 193-206. DOI: 10.25209/2079-3316-2021-12-2-193-206.

[274] **Рыбаков А. А.** Геометрическое перестроение расчетной сетки с помощью общей огибающей семейства сфер в задаче ледообразования. // Современные информационные технологии и ИТ-образование, 2023, Т. 19, № 2, С. 282-291. DOI: 10.25559/SITITO.019.202302.282-291.

[275] **Рыбаков А. А.** Оптимизация задачи об определении конфликтов с опасными зонами движения летательных аппаратов для выполнения на Intel Xeon Phi. // Программные продукты и системы, 2017, Т. 30, № 3, С. 524-528. DOI: 10.15827/0236-235X.119.3.524-528.

[276] The Stanford 3D Scanning Repository,
URL: https://graphics.stanford.edu/data/3Dscanrep/ (дата обращения 24.06.2026).

[277] **Freylekhman S., Rybakov A.** Self-intersections elimination for unstructured surface computational meshes. // Lobachevskii Journal of Mathematics, 2022, Vol. 43, No. 10, P. 2846-2852. DOI: 10.1134/S1995080222130133.

[278] **Горшков А. В., Коршунова А. Л.** Оптимальный алгоритм поиска пересечений в задаче Монте-Карло моделирования распространения

зондирующего излучения в головном мозге человека. // Вестник Нижегородского университета им. Н. И. Лобачевского, 2012, № 5(2), С. 73-80.

[279] **Рыбаков А. А.** Удаление самопересечений поверхностной неструктурированной сетки в задаче моделирования обледенения. // Современные информационные технологии и ИТ-образование, 2026, Т. 22, № 1, С. 63-75. DOI: 10.25559/SITITO.022.202601.63-75.

[280] **Rivara M.-C., Rodrigez-Moreno P.** Tuned terminal triangles centroid Delaunay algorithm for quality triangulation. // in Proceedings of the 27th International Meshing Roundtable, 2019, P. 211–228. DOI: 10.1007/978-3-030-13992-6_12.

[281] **Rakotoarivelo H., Ledoux F.** Accurate manycore-accelerated manifold surface remesh kernels. // in Proceedings of the 27th International Meshing Roundtable, 2019, P. 405–423. DOI: 10.1007/978-3-030-13992-6_22.

[282] **Borouchaki H., Laug P., George P.-L.** Parametric surface meshing using a combined advancing-front generalized Delaunay approach. // International Journal of Numeric Methods in Enineering., 2000, Vol. 49, P. 233–259. DOI: 10.1002/1097-0207(20000910/20)49:1/23.0.CO;2-G.

[283] **Сазонов В. В.** Построение интерактивной геометрической модели внешней поверхности космического аппарата. // Математическое моделирование, 2020, Т. 32, № 6, С. 37-52. DOI: 10.20948/mm-2020-06-03.

[284] **Скворцов А. В.** Триангуляция Делоне и ее применение. // Издательство Томского университета, 2002, 128 С.

[285] **Скворцов А. В.** Особенности реализации алгоритмов построения триангуляции Делоне с ограничениями. // Вестник Томского государственного университета, 2002, № 275. URL: https://cyberleninka.ru/article/n/osobennosti-realizatsii-algoritmov-postroeniya-triangulyatsii-delone-s-ogranicheniyami (дата обращения 24.06.2026).

[286] **Panchal D., Jayaswal D.** Feature sensitive geometrically faithful highly

regular direct triangular isotropic surface remeshing. // Sadhana, 2022, Vol. 47(94), P. 1–19. DOI: 10.1007/s12046-022-01866-7.

[287] **Шуткин В. Н., Морозкин Н. К., Семенов В. А., Тарлапан О. А.** Оптимизация генерации иерархических уровней детализации для масштабных полигональных сцен. // Труды института системного программирования РАН, 2025, Т. 37, № 3, С. 311-324. DOI: https://doi.org/10.15514/ISPRAS-2025-37(3)-22.

[288] **Budak A. S., Frolov V. A., Galaktionov V. A., Garifullin A. R.** Study of surface representation methods based on signed distance functions. // Programming and Computer Software, 2025, Vol. 3, P. 15-26. DOI: 10.7868/S3034584725030027.

[289] **Jung W., Shin H., Choi B.** Self-intersection removal in triangular mesh offsettings. // CAD Journal, 2004, No. 1, P. 477–484. DOI: 10.1080/16864360.2004.10738290.

[290] **Глумова Е. С.** Алгоритм адаптивного разбиения облака точек с использованием иерархии ограничивающих объемов. // Graphic information systems and immersive technologies in the product life cycle stages, 2025, P. 967-984. DOI: 10.25686/978-5-8158-2474-4-2025-976-984.

[291] **Lazard S., Valque L.** Removing self-intersections in 3D meshes while preserving floating-point coordinates. // 2025, URL: https://inria.hal.science/hal-04907149v1 (дата обращения 24.06.2026).

[292] **Blazek J.** Computational fluid dynamics. Principles and applications. // Elsevier, 2015, 451 P.

[293] **Смирнова Н. С.** Сравнение схем с расщеплением потока для численного решения уравнения Эйлера сжимаемого газа. // Труды МФТИ, 2018, Т. 10, № 1, С. 122-141.

[294] **Куликовский А. Г., Погорелов Н. В., Семенов А. Ю.** Математические вопросы численного решения гиперболических систем уравнений. // М.: Физматлит, 2001, 608 С.

[295] **Toro E.** Riemann solvers and numerical methods for fluid dynamics: A practical introduction. // Springer, Berlin–Heidelberg, 1999, 645 P.

[296] **Бендерский Л. А., Любимов Д. А.** Применение RANS/ILES метода высокого разрешения для исследования сложных турбулентных струй. // Ученые записки ЦАГИ, 2024, Т. XLV, № 2, С. 22-36.

[297] **Черников С. Н.** Свертывание конечных систем линейных неравенств. // Доклады АН СССР, 1963, Т. 152, № 5, С. 1075-1078.

[298] **Wright W., Struk P., Bartkus T., Addy G.** Recent advances in the LEWICE icing model. // SAE Technical Paper, 2015. DOI: 10.4271/2015-01-2094

[299] **Clarke D., Salas M., Hassan H.** Euler calculations for multielement airfoils using cartesian grids. // AIAA Journal, 1986, Vol. 24, № 3, P. 353-358. DOI: 10.2514/3.9273.

[300] **Farrashkhalvat M., Miles J.** Basic structured grid generation with an introduction to unstructured grid generation. // 2003, Butterworth Heinemann, 231 P.

[301] **Savin. G., Benderskiy L., Lyubimov D., Rybakov A.** RANS/ILES method optimization for effective calculations on supercomputer. // Lobachevskii Journal of Mathematics, 2019, Vol. 40, No. 5, P. 566-573. DOI: 10.1134/S1995080219050172.

[302] **Fadlun E., Verzicco R., Orlandi P., Mohd-Yusof J.** Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations. // Journal of Computational Physics, 2000, Vol. 161, P. 35-60. DOI: 10.1006/jcph.2000.6484.

[303] **Рыбаков А. А.** Метод погруженной границы с использованием фиктивных ячеек в трехмерной постановке. // Современные информационные технологии и ИТ-образование, 2020, Т. 16, № 2, С. 321-330. DOI: 10.25559/SITITO.16.202002.321-330.

[304] **Воеводин В. В.** Отображение проблем вычислительной математики на архитектуру вычислительных систем. // Вычислительные методы и программирование, 2000, Т. 1, № 2, С. 37-44. URL: https://www.mathnet.ru/rus/vmp796 (дата обращения 24.06.2026).

[305] **Рыбаков А. А.** Внутреннее представление и механизм межпроцессного обмена для блочно-структурированной сетки при выполнении расчетов на суперкомпьютере. // Программные системы: теория и приложения, 2017, Т. 8, Вып. 1, С. 121-134. DOI: 10.25209/2079-3316-2017-8-1-121-134.

[306] **Бендерский Л. А., Любимов Д. А., Рыбаков А. А.** Анализ эффективности масштабирования при расчетах высокоскоростных турбулентных течений на суперкомпьютере RANS/ILES методом высокого разрешения. // Труды НИИСИ РАН, 2017, Т. 7, № 4, С. 32-40. DOI: 10.25682/NIISI.2018.4.9975.

[307] **Бендерский Л. А., Любимов Д. А., Рыбаков А. А.** Инструментарий подготовки блочно-структурированной сетки для проведения расчетов методом RANS/ILES. // Труды НИИСИ РАН, 2018, Т. 8, № 4, С. 102-106. DOI: 10.25682/NIISI.2018.4.0012.

[308] **Рыбаков А. А.** Распределение вычислительной нагрузки между узлами гетерогенного вычислительного кластера. // Электронный научный журнал: Программные продукты, системы и алгоритмы, 2018, № 1, С. 26-32. DOI: 10.15827/2311-6749.26.300.

[309] **Рыбаков А. А.** Партицирование графа смежности блочно-структурированной сетки. // Современные информационные технологии и ИТ-образование, 2017, Т. 13, № 1, С. 43-48. DOI: 10.25559/SITITO.2017.1.422.

[310] **Романовский И. В.** Алгоритмы решения экстремальных задач. // Издательство «Наука», Москва, 1977.

[311] **Savin G. I., Shabanov B. M., Telegin P. N., Baranov A. V.** Joint Supercomputer Center of the Russian Academy of Sciences: present and future.

// Lobachevskii Journal of Mathematics, 2019, Vol. 40, № 11, P. 1853-1862. DOI: 10.1134/S1995080219110271.

[312] Инфраструктура и основные вычислительные ресурсы МСЦ РАН. URL: https://www.jscc.ru/resources/hpc/ (дата обращения 24.06.2026).

[313] **Рыбаков А. А.** Распределение вычислительной нагрузки между узлами суперкомпьютерного кластера при расчетах задач газовой динамики с дроблением расчетной сетки. // Современные информационные технологии и ИТ-образование, 2016, Т. 12, № 2, С. 101-107.

[314] **Рыбаков А. А., Чопорняк А. Д.** Декомпозиция поверхностной неструктурированной расчетной сетки для масштабирования вычислений на суперкомпьютере. // Современные информационные технологии и ИТ-образование, 2020, Т. 16, № 4, С. 851-861. DOI: 10.25559/SITITO.16.202004.851-86.

[315] NACA-0012. URL: https://orangeflowcfd.com/naca-0012 (дата обращения 24.06.2026).

[316] **Preis R., Diekmann R.** PARTY – a software library for graph partitioning. // In: Topping B.H.V. (ed.) Advances in Computational Mechanics for Parallel and Distributed Processing. Civil-Comp Press, Edinburgh, UK, 1997, P. 63-71. DOI: 10.4203/ccp.45.3.1.

[317] **Якобовский М. В.** Инкрементальный алгоритм декомпозиции графов. // Вестник Нижегородского университета им. Н. И. Лобачевского. Серия «Математическое моделирование и оптимальное управление», 2005, № 1(28), С. 243-250.

[318] **Urschel J., Zikatanov L.** Spectral bisection of graphs and connectedness. // Linear Algebra and its Applications, 2014, Vol. 449, P. 1–16. DOI: 10.1016/j.laa.2014.02.007.

[319] **Zhao L., Liu Y., Zhang C. et al.** Automatic optimal block decomposition for structured mesh generation using genetic algorithm. // Journal of

the Brazil Society of Mechanical Sciences and Engineering, 2019, Vol. 41. DOI: 10.1007/s40430-018-1510-0.

[320] **Kapyris G., Kumar V.** A fast and high quality multilevel scheme for partitioning irregular graphs. // SIAM Journal on Scientific Computing, 1998, Vol. 20, Issue 1, P. 359-392. DOI: 10.1137/S1064827595287997.

[321] **Головченко Е. Н.** Обзор алгоритмов декомпозиции графов. // Препринты ИПМ им. М. В. Келдыша, 2020, № 2, С. 1-38. DOI: 10.20948/prepr-2020-2.

[322] **Рыбаков А. А.** Декомпозиция расчетной сетки с помощью генетического алгоритма. // Программные продукты и системы, 2025, Т. 38, № 2, С. 337-344. DOI: 10.15827/0236-235X.150.337-344.

[323] **Chahar V., Katoch S., Chauhan S.** A review on genetic algorithm: past, present and future. // Multimedia Tools and Applications, 2021, Vol. 80, № 4. DOI: 10.1007/s11042-020-10139-6.

[324] **Chaouche A., Boulif M.** Edge-set reduction to efficiently solve the graph partitioning problem with the genetic algorithm. // arXiv, 2023. DOI: 10.48550/arXiv.2307.10410.

[325] **Li M., Cui H., Zhou C., Xu S.** GAP: genetic algorithm based large-scale graph partition in heterogeneous cluster. // IEEE Access, 2020. DOI: 10.1109/ACCESS.2020.3014351.

[326] **Багров А. Д., Рыбаков А. А.** Сглаживание границ между доменами поверхностной расчетной сетки. // Современные информационные технологии и ИТ-образование, 2021, Т. 17, № 2, С. 265-274. DOI: 10.25559/SITITO.17.202102.265-274.

[327] **Shabanov B., Rybakov A., Shumilin S., Vorobyov M.** Scaling of supercomputer calculations on unstructered surface computational meshes. // Lobachevskii Journal of Mathematics, 2021, Vol. 42, No. 11, P. 2571-2579. DOI: 10.1134/S1995080221110202.

[328] Рейтинг серверных процессоров Intel Xeon: сравнение производительности и выбор лучшего. // СерверМолл, 11.04.2025. URL: https://servermall.ru/blog/reyting-servernykh-protsessorov-intel-xeon-sravnenie-proizvoditelnosti-i-vybor-luchshego (дата обращения 24.06.2026).

[329] Intel такое и не снилось. AMD создала процессор для настольных ПК с поистине гигантским числом ядер. Конкурентов нет. // CNews, 21.05.2015. URL: https://www.cnews.ru/news/top/2025-05-21_intel_takoe_i_ne_snilosamd_sozdala (дата обращения 24.06.2026).

[330] **Кузьминский М. Б.** Современные серверные ARM-процессоры для суперЭВМ: A64FX и другие. Начальные данные тестов производительности. // Программные системы: теория и приложения, 2022, Т. 13, № 1(52), С. 63-129. DOI: 10.25209/2079-3316-2022-13-1-63-129.

[331] **Гуличева А. А., Рыбаков А. А.** Реберная раскраска кубического графа в задаче распараллеливания расчетов на неструктурированной поверхностной расчетной сетке. // Программные продукты и системы, 2024, Т. 37(3), С. 374-383. DOI: 10.15827/0236-235X.142.374-383.

[332] **Гильфанов Л. Л., Мигранов С. В., Бикбов А. А.** Распараллеливание решения задач с использованием раскраски графов. // Молодой ученый, 2021, № 5 (347), С. 4-6. URL: https://moluch.ru/archive/347/78208/ (дата обращения 24.06.2026).

[333] **Визинг В. Г.** Об оценке хроматического класса p-графа. // сб. Дискретный анализ. – Новосибирск: Институт математики СО АН СССР, 1964, Вып. 3, С. 25–30.

[334] **Визинг В. Г.** Критические графы с данным хроматическим классом. // сб. Дискретный анализ. – Новосибирск: Институт математики СО АН СССР, 1965, Вып. 5, С. 9–17.

[335] **Soifer A.** The mathematical coloring book. // 2009, Springer, 619 P.

[336] **Tait P.** Remarks on the colourings of maps. // Proceedings of

the Royal Society of Edinburgh, 1880, Vol. 10, № 4, P. 501-503. DOI: 10.1017/S0370164600044229.

[337] **Курапов С. В., Давидовский М. В., Толок А. В.** Визуальный алгоритм раскраски плоских графов. // Научная визуализация, 2018, Т. 10, № 3, С. 1-33. DOI: 10.26583/sv.10.3.01.

[338] **Курапов С. В., Давидовский М. В., Толок А. В.** Алгебраические методы раскраски кубических графов. // Научная визуализация, 2020, Т. 12, № 2, С. 21-36. DOI: 10.26583/sv.12.2.03.

[339] **Курапов С. В., Давидовский М. В.** Теорема о четырех красках. Теория, методы, алгоритмы. // Запорожье: Запорожский национальный университет, 2020, 94 С.

[340] **Годунов С. К.** Разностный метод расчета ударных волн. // Успехи математических наук, 1957, Т. 12, Вып. 1(73), С. 176-177. URL: https://www.mathnet.ru/rus/rm753 (дата обращения 24.06.2026).

[341] **Тишкин В. Ф., Жуков В. Т., Мышецкая Е. Е.** К обоснованию схемы Годунова в многомерном случае. // Математическое моделирование, 2016, Т. 28, № 2, С. 86-96. DOI: 10.1134/S2070048216050124.

[342] **Воробьев М. Ю., Рыбаков А. А., Чопорняк А. Д.** Сравнение стратегий распараллеливания векторизованного римановского решателя с помощью OpenMP для микропроцессора Intel Xeon Phi KNL. // Труды НИИСИ РАН, 2020, Т. 10, № 5-6, С. 113-119. DOI: 10.25682/NIISI.2020.5_6.0015.

[343] **Воробьев М. Ю., Рыбаков А. А., Никсон М. О.** Исследование масштабируемости плотных параллельных вычислений на микропроцессорах Intel. // Инжиниринг предприятий и управление знаниями (ИП & УЗ-2020). Сборник научных трудов XXIII Международной научной конференции, 2020, С. 45-52.

[344] **Kalamkar D., Mudigere D., Mellempudi N., Das D. et al.** A study of BFloat16 for deep learning training. // ArXiv, 2019, art. 1905.12322, P. 1-11. DOI: 10.48550/arXiv.1905.12322.

[345] **Zhou H., Han Q., Shi H., Zhang Y., Yao J.** Boost linear algebra computation performance via efficient VNNI utilization. // ASPLOS '24: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2024, Vol. 3, P. 149-163. DOI: 10.1145/3620666.3651333.

[346] **Díez-Cañas G.** Faster-than-native alternatives for x86 VP2INTERSECT instructions. // ArXiv, 2021, art. 2112.06342, P. 1-10. DOI: 10.48550/arXiv.2112.06342.

[347] **Kovats T., Rameshan N., Karunarathe K., Giannopoulos I., Sebastian A.** In-memory encryption using the advanced encryption standard. // Philosophical Transactions, 2025, Vol. 383, art. 20230396. DOI: 10.1098/rsta.2023.0396.

[348] **Yoo T.-H., Kivilinna J., Cho C.-H.** AVX-based acceleration of ARIA block cipher algorithm. // IEEE Access, 2023. DOI: 10.1109/ACCESS.2023.3298026.

[349] **Волконский В. Ю., Окунев С. К.** Предикатное представление как основа оптимизации программы для архитектур с явно выраженной параллельностью. // Информационные технологии, 2003, № 4, С. 36–45.

[350] **Ким А. К., Перекатов В. И., Ермаков С. Г.** Микропроцессоры и вычислительные комплексы семейства «Эльбрус», Питер, СПб., 2013, 273 С.

[351] **Бендерский Л. А., Лещев С. А., Рыбаков А. А.** Векторизация операций над матрицами малой размерности для процессора Intel Xeon Phi Knights Landing. // Современные информационные технологии и ИТ-образование, 2018, Т. 14, № 1, С. 73-90. DOI: 10.25559/SITITO.14.201801.073-090.

[352] **Бендерский Л. А., Рыбаков А. А., Шумилин С. С.** Векторизация перемножения малоразмерных матриц специального вида с использованием инструкций AVX-512. // Современные информаци-

онные технологии и ИТ-образование, 2018, Т. 14, № 3, С. 594-602. DOI: 10.25559/SITITO.14.201803.594-602.

[353] **Шабанов Б. М., Рыбаков А. А., Чопорняк А. Д.** Оптимизации, применяемые к графу потока управления программы для повышения эффективности векторизации плоских циклов. // Труды НИИСИ РАН, 2021, Т. 11, № 2, С. 11-19. DOI: 10.25682/NIISI.2021.2.0002.

[354] **Рыбаков А. А., Швиндт А. Н.** Создание инструментария для векторизации тела плоского цикла с помощью векторных инструкций AVX-512. // Программные продукты и системы, 2023, Т. 36, № 4, С. 561-572. DOI: 10.15827/0236-235X.142.561-572.

[355] **Armstrong J.** Programming Erlang. Software for a concurrent world. // The Pragmatic Programmers, 2013, 520 P.

[356] **Буздалов Д. В., Петренко А. К., Хорошилов А. В.** О представлении модельного времени при помощи механизмов функционального программирования. // Труды института системного программирования РАН, 2018, Т. 30, № 6, С. 341-366. DOI: 10.15514/ISPRAS-2018-30(6)-20.

[357] **Savin G., Shabanov B., Rybakov A., Shumilin S.** Vectorization of flat loops of arbitrary structure using instructions AVX-512. // Lobachevskii Journal of Mathematics, 2020, Vol. 41, No. 12, P. 2566-2574. DOI: 10.1134/S1995080220120331.

[358] **Бучацкий Р. А., Чуркин Я. А., Чибисов К. А., Пантилимонов М. В., Долгодворов Е. В., Вязовцев А. В., Волохов А. Г., Трунов В. В., Мирякян Г. О., Китаев К. Н., Белеванцев А. А.** Проверка программ на соответствие стандарту MISRA с использованием инфраструктуры Clang. // Труды института системного программирования РАН, 2023, Т. 35, № 5, С. 169-192. DOI: 10.15514/ISPRAS-2023-35(5)-12.

[359] **Muchnick S.** Advanced compiler design and implementation. // Morgan Kaufmann Publishers, 1997.

[360] **Рыбаков А. А.** Алгоритм создания случайных графов потока управле-

ния для анализа глобальных оптимизаций в компиляторе. // Parallel and Distributed Computing Systems PDSC 2013, Collection of scientific papers, P. 269-275.

[361] **Aho A., Lam M., Sethi R., Ulman J.** Compilers: principles, techniques, and tools. // Prentice Hall, 2nd ed, 2006.

[362] **Четверина О. А.** Методы коррекции профильной информации в процессе компиляции. // Труды ИСП РАН, 2015, Т. 27, Вып. 6, С. 49-66. DOI: 10.15514/ISPRAS-2015-27(6)-4.

[363] **Рыбаков А. А., Мещеряков А. О.** Векторизация трехмерного метода погруженных границ для повышения эффективности расчетов на микропроцессорах Intel. // Программные продукты и системы, 2023, Т. 36, № 1, С. 130-143. DOI: 10.15827/0236-235X.141.130-143.

[364] **Stpiczyński P.** Language-based vectorization and parallelization using intrinsics, OpenMP, TBB and Cilk Plus. // Journal of Supercomputing, 2018, 74:1461-1472. DOI: 10.1007/s11227-017-2231-3.

[365] **Рыбаков А. А., Шумилин С. С.** Векторизация сильно разветвленного управления с помощью инструкций AVX-512. // Труды НИИСИ РАН, 2018, Т. 8, № 4, С. 114-126. DOI: 10.25682/NIISI.2018.4.0014.

[366] **Аубакиров Т. О., Желанников А. И., Иванов П. Е., Ништ М. И.** Спутные следы и их воздействие на летательные аппараты. // Моделирование на ЭВМ, Алматы, 1999, 280 С.

[367] **Бабкин В. И., Белоцерковский А. С., Турчак Л. И., Баранов Н. А., Замятин А. И., Каневский М. И., Морозов В. В., Пасекунов И. В., Чижов Н. Ю.** Системы обеспечения вихревой безопасности полетов летательных аппаратов. // М.: Наука, 2008, 373 С.

[368] **Рыбаков А. А.** Векторизация программного кода, содержащего маловероятные регионы, в задачах вычислительной геометрии. // Современные информационные технологии и ИТ-образование, 2022, Т. 18, № 1, С. 28-38. DOI: 10.25559/SITITO.18.202201.28-38.

[369] **Рыбаков А. А.** Векторизация циклов с условными операциями с помощью комбинирования векторных масок. // Современные информационные технологии и ИТ-образование, 2024, Т. 20, № 3, С. 520-534. DOI: 10.25559/SITITO.020.202403.520-534.

[370] **Рыбаков А. А., Чопорняк А. Д.** Повышение производительности векторного кода с помощью мониторинга плотности масок в векторных инструкциях. // Труды НИИСИ РАН, 2020, Т. 10, № 4, С. 40-47. DOI: 10.25682/NIISI.2020.4.0006.

[371] **Toh Y.** Efficient non-iterative multi-point method for solving the Riemann problem. // Nonlinear Dynnamics, 2024, Vol. 112, P. 5439-5451. DOI: 10.1007/s11071-023-09229-5.

[372] **Рыбаков А. А.** Векторизация нахождения пересечения объемной и поверхностной сеток для микропроцессоров с поддержкой AVX-512. // Труды НИИСИ РАН, 2019, Т. 9, № 5, С. 5-14. DOI: 10.25682/NIISI.2019.5.0001.

[373] **Волконский В. Ю., Дроздов А. Ю., Ровинский Е. В.** Метод использования мелкоформатных векторных операций в оптимизирующем компиляторе. // Информационные технологии и вычислительные системы, 2004, выпуск 3, С. 63-77. URL: https://www.mathnet.ru/rus/itvs666 (дата обращения 24.06.2026).

[374] **Rybakov A., Shumilin S.** Vectorization of the Riemann solver using the AVX-512 instruction set. // Program Systems: Theory and Applications, 2019, Vol. 10, № 3(42), P. 41-58. DOI: 10.25209/2079-3316-2019-10-3-41-58.

[375] **Рыбаков А. А., Шумилин С. С.** Векторизация римановского решателя с использованием набора инструкций AVX-512. // Программные системы: Теория и приложения, 2019, Т. 10, № 3(42), С. 59-80. DOI: 10.25209/2079-3316-2019-10-3-59-80.

[376] **Krzikalla O., Wende F., Hohnerbach M.** Dynamic SIMD vector lane scheduling. // ISC High Performance 2016: High Performance

Computing, Lectuer Notes Computer Science, 2016, Vol. 9945, P. 354–365. DOI: 10.1007/978-3-319-46079-6_25.

[377] **Булат П. В., Волков К. Н.** Одномерные задачи газовой динамики и их решение при помощи разностных схем высокой разрешающей способности. // Научно-технический вестник информационных технологий, механики и оптики, 2015, Т. 15, № 4, С. 731–740. DOI: 10.17586/2226-1494-2015-15-4-731-740.

[378] **Рыбаков А. А., Шумилин С. С.** Исследование эффективности векторизации гнезд циклов с нерегулярным числом итераций. // Программные системы: Теория и приложения, 2019, Т. 10, № 4(43), С. 77-96. DOI: 10.25209/2079-3316-2019-10-4-77-96.

[379] **Shabanov B., Rybakov A., Shumilin S.** Vectorization of high-performance scientific calculations using AVX-512 instruction set. // Lobachevskii Journal of Mathematics, 2019, Vol. 40, No. 5, P. 580-598. DOI: 10.1134/S1995080219050196.

[380] **Рыбаков А. А., Телегин П. Н., Шабанов Б. М.** Проблемы векторизации гнезд циклов с использованием инструкций AVX-512. // Электронный научный журнал: Программные продукты, системы и алгоритмы, 2018, № 3, С. 1-11. DOI: 10.15827/2311-6749.28.314.

[381] Using Intel AVX without Writing AVX. // https://software.intel.com/enus/articles/using-intel-avx-without-writing-avx (дата обращения 24.06.2026).

[382] **Rybakov A. A.** Vectorization of the integer calculations in the graph decomposition problem. // Lobachevskii Journal of Mathematics, 2025, Vol. 46, № 11, P. 6012-6018. DOI: 10.1134/S1995080225610987.

[383] **Рыбаков А. А., Чопорняк А. Д., Шмелёв А. С.** Методика векторизации вычислений с помощью плоских циклов. // Современные информационные технологии и ИТ-образование, 2026, Т. 22, № 1, С. 50-62. DOI: 10.25559/SITITO.022.202601.50-62.