Московский государственный университет им М.В. Ломоносова
Механико-математический факультет

На правах рукописи

Павлухин Павел Викторович

Эффективное решение задач газовой динамики на кластерных системах с графическими ускорителями

Специальность 05.13.18 —
«Математическое моделирование, численные методы и комплексы программ»

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
д. ф.-м. н., профессор
Меньшов Игорь Станиславович

Москва — 2019

Оглавление

Стр.

Введение 4

Глава 1. Обзор 12

1.1 Этапы развития вычислительной техники 12

1.2 Особенности архитектуры графических ускорителей 14

1.3 Методы 18

1.4 Явные и неявные схемы 21

Глава 2. Базовый численный метод 24

2.1 Модель свободной границы 31

2.2 Решение систем дискретных уравнений 36

Глава 3. Параллельный алгоритм для метода LU-SGS 39

3.1 Схема безматричных вычислений в LU-SGS 39

3.2 Обзор подходов к распараллеливанию LU-SGS 43

3.3 Построение параллельного алгоритма для вычислительных систем с распределенной памятью 52

3.4 Модификация алгоритма для GPU 57

Глава 4. Детали программной реализации 59

Глава 5. Результаты вычислительных экспериментов 66

5.1 Дифракция ударной волны на клине 66

5.2 Обтекание профиля NACA0012 67

5.3 Сверхзвуковое обтекание клина 73

5.4 Течение вокруг системы цилиндров 74

5.5 Моделирование макета тягового устройства 75

Стр.

5.6 Моделирование течения вокруг профиля DLR F6 77

Глава 6. Исследование производительности и масштабируемости программной реализации 82

6.1 Производительность расчетов на одном GPU, сравнение с CPU . . 83

6.2 Исследование взаимодействия библиотек CUDA и MPI 85

6.3 Масштабируемость на большом числе GPU 91

Глава 7. Заключение 93

7.1 Перспективы дальнейшей работы 93

Список литературы 99

Введение

Актуальность темы исследования. Решение задач газовой динамики на современных вычислительных системах с новыми архитектурами сопряжено с рядом возникающих при этом трудностей [1]. Первая связана с дискретизацией расчетной области. В случае, если она является геометрически сложной, сеточное разбиение в подавляющем большинстве случаев не структурировано. Построение сетки при этом требует значительных вычислительных ресурсов и нередко «ручного» вмешательства для ее коррекции, что приводит к немалым временным затратам. Неструктурированные сеточные разбиения порождают нерегулярный доступ к памяти. Как результат, производительность программных реализаций методов для сеток данного типа оказывается ограничена не числом выполняемых в единицу времени арифметических операций (compute-bound), а пропускной способностью памяти (memory-bound). В большей степени это критично для массивно-параллельных вычислителей, поскольку эффективность их работы зависит в первую очередь от упорядоченности обращений в память. Поэтому наиболее предпочтительно использовать структурированные сеточные разбиения, для которых характерен регулярный шаблон обращений в память. Но построение таких сеток, согласованных с границей расчетной области, может оказаться в ряде случаев трудновыполнимой или вовсе неразрешимой задачей.

Другая сложность обусловлена взаимосвязанным развитием численных методов и архитектуры процессоров. Для решения задач на системах с ограниченными вычислительными ресурсами использовались сетки низкого разрешения. Чтобы получить на них более точное решение, приходилось строить численные методы высокого порядка со сложной алгоритмической структурой. С другой стороны, вычислительные ядра процессоров также становились все более “тяжелыми”: внеочередное исполнение команд, преднакачка данных, прогнозирование ветвлений, векторные инструкции позволяли эффективно ре-

ализовывать сложные методы. Но масштабируемость вычислительных систем на таких “тяжелых” ядрах ограничена, и их дальнейшее усложнение приводит к большему снижению эффективности. Это стало причиной появления систем на новых массивно-параллельных архитектурах с большим числом простых ядер. Вот здесь и проявилась проблема: накопленный за десятилетия “багаж” численных методов оказался плохо подходящим для реализации на новых вычислителях, поскольку высокая производительность в них достигается не за счет эффективного исполнения каждого из небольшого (порядка 10) числа “тяжелых” потоков, а за счет одновременной обработки значительно большего (порядка 1000) числа “легких”. Иными словами, простое устройство ядра массивно-параллельных вычислителей влечет за собой требование вычислительного примитивизма для применяемых численных методов.

Явные методы хорошо подходят для реализации на новых вычислительных архитектурах, но их применение сильно ограничено из-за условия устойчивости. В задачах со сложной геометрией, где неизбежно в сеточном разбиении будут присутствовать разномасштабные ячейки, глобальный шаг интегрирования по времени будет определяться размерами наименьшей из них, что приведет к неоправданно высокому росту вычислительной сложности. Явно-неявные, неявные методы позволяют обойти это ограничение, но они значительно сложнее с точки зрения указанных выше требований, особенно в части возможного их распараллеливания.

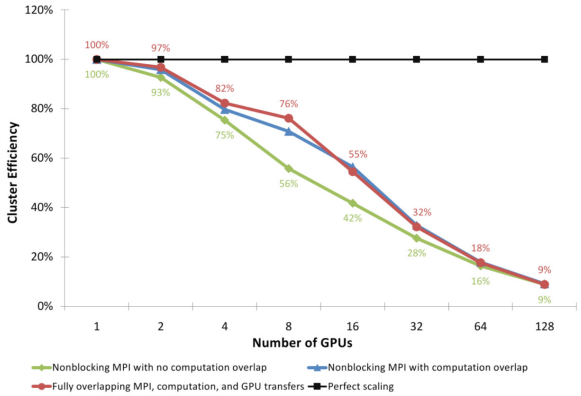
Реализация неявных схем для графических ускорителей (GPU) является непростой задачей, поскольку имеющаяся, как правило, в них зависимость по данным значительно затрудняет написание эффективного решателя, особенно в отсутствие средств глобальной синхронизации на GPU. В частности, в [2] метод LU-SGS [3], использующийся для решения СЛАУ, порожденной неявными схемами, рассматривается как один из наиболее подходящих методов для расчетов на графических ускорителях. Однако, из-за возникающей в нем зависимости по данным, сильно усложняющей распараллеливание, в указанной работе был выбран другой – DP-LUR [4], в котором отсутствует зависимость по дан-

ным, но при этом и выше вычислительная сложность: сравнение методов [5; 6] показывает, что для нахождения стационарного решения с помощью DP-LUR затрачивается процессорное время, почти двукратно превышающее время решения с помощью LU-SGS, поэтому с точки зрения производительности и эффективности использования вычислительных ресурсов предпочтительнее параллельная реализация последнего метода.

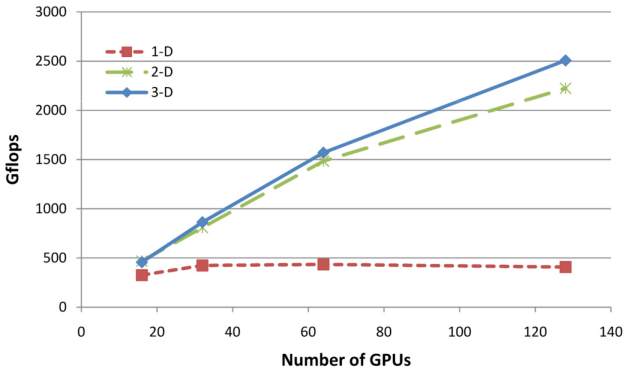
В большинстве работ (например, в [7]), параллельный алгоритм для LU-SGS соблюдает работу данного метода лишь частично – отдельно на каждой из подобластей, полученных после декомпозиции расчетной области на параллельные процессы, с упрощенной процедурой обмена граничными ячейками (т.е. с нарушением зависимости по данным в рассматриваемом методе), что в итоге может приводить к весьма значительному снижению скорости сходимости решения [8] . В тех же работах, где LU-SGS соблюдается на *всей* расчетной области, масштабируемость программных реализаций весьма ограничена [9].

Помимо собственно численного метода, важную роль, как уже было сказано, играет тип сеточного разбиения — GPU значительно эффективнее работают с регулярными структурами данных, в которых представляются структурированные сеточные разбиения, чем с нерегулярными, характерными для неструктурированных сеток. В свою очередь, применение структурированных сеток, адаптированных (конформных) к геометрии расчетной области, накладывает ограничения на сложность геометрии расчетной области. Поэтому возникает задача разработки и адаптации численных методов для решения задач газовой динамики со сложной геометрией на структурированных сетках [10].

Наконец, весьма остро стоит затронутый выше вопрос масштабируемости программных реализаций решателей на вычислительных системах. В частности, в работах [11; 12] масштабируемость метода Якоби с использованием декартовых сеток составила 18% на 64 GPU (1 млн ячеек/GPU) и 67.5% на 128 GPU (2 млн ячеек/GPU) соответственно, Рис. 0.1. Реализация на GPU неявного метода ADI [13] на декартовых сетках имеет эффективность в 75% уже на 4 GPU (2 млн ячеек/GPU).



а)



б)

Рисунок 0.1 — Масштабируемость решателей из [11] (а) и [12] (б) на вычислительных системах с множеством GPU

В части решателей масштабируемость изначально ограничена самим параллельным алгоритмом, лежащим в основе программной реализации, т.е. она лимитирована той частью вычислений, которую в соответствии с алгоритмом необходимо выполнять исключительно последовательно. Однако даже если алгоритм не содержит таких последовательных вычислений, эффективность его программной реализации тем не менее может значительно падать с увеличением вычислительных ресурсов для нее. Связано это, как правило, с тем, что отношение времени обмена данными между параллельными процессами ко времени счета начинает расти при выполнении задачи на всё бо́льшем числе вычислительных ядер; иными словами, время межпроцессных коммуникаций начинает вносить всё бо́льший вклад в общее время работы солвера. И здесь ключевой для повышения эффективности и масштабируемости является возможность *совмещения* по времени вычислений и обмена данных между процессами, т.е. пересылки сообщений между ними на фоне “полезного” счета. Данную возможность должен, во-первых, предоставлять параллельный алгоритм (т.е. обеспечивать корректность при совмещении счета и обменов данными), во-вторых, она должна быть соответствующим образом реализована в программном коде. Применительно к кластерным системам, оснащенным графическими ускорителями в качестве сопроцессоров, последнее требование означает использование нетривиальных схем координации доступа к совмест-

но используемым CUDA и MPI областям памяти через вызовы неблокирующих функций, предоставляемыми библиотеками указанных интерфейсов. Тем не менее, как будет показано в данной работе, даже при использовании сложных схем взаимодействия между CUDA и MPI совмещение счета и обмена данными между процессами может не работать, и для решения этой проблемы необходимо проводить достаточно глубокий анализ производительности программной реализации с привлечением знаний о деталях реализации библиотек MPI.

Таким образом, проблема адаптации методов, разработки эффективных параллельных алгоритмов для них и соответствующих программных реализаций для решения широкого класса задач газовой динамики на кластерных вычислительных системах с графическими ускорителями носит актуальный характер.

Целью диссертационной работы является построение вычислительных алгоритмов для получения максимальной эффективности при использовании современных и перспективных систем гибридной архитектуры для решения прикладных задач газовой динамики. Для достижения поставленной цели рассматриваются следующие задачи:

1. Разработка высокоэффективного масштабируемого параллельного алгоритма гибридной явно-неявной схемы и метода свободной границы на вычислительных системах с графическими ускорителями, который в точности реализует работу своего последовательного прототипа на всей расчетной области с доказательством его корректности.
2. Программная реализация разработанного алгоритма для высокопроизводительных вычислительных систем с графическими ускорителями с использованием технологии CUDA и стандарта MPI.
3. Проведение расчетов газодинамических течений с помощью разработанного программного комплекса и сравнение с результатами, полученными альтернативными методами, и экспериментальными данными.

4. Исследование эффективности и масштабируемости разработанной программной реализации при расчете больших задач с использованием нескольких сотен графических ускорителей.

Научная новизна. Представленные в работе результаты являются новыми. В частности, новыми являются разработанный параллельный алгоритм для методов свободной границы и LU-SGS и реализованный на их основе программный комплекс для решения нестационарных задач газовой динамики на вычислительных системах с большим числом графических ускорителей.

Теоретическая ценность и практическая значимость диссертационной работы состоят в разработанном масштабируемом параллельном алгоритме для методов свободной границы и LU-SGS, а также в разработанном программном комплексе на его основе с использованием технологий CUDA и MPI, который позволяет решать задачи газовой динамики, эффективно задействуя несколько сотен графических ускорителей.

На защиту выносятся следующие основные результаты и положения:

1. Предложено обобщение математической и численной модели свободной границы под специфику архитектуры графических ускорителей.
2. На основе модели свободной границы, гибридной явно-неявной схемы и итерационного метода LU-SGS разработан параллельный алгоритм для решения задач газовой динамики на вычислительных системах с графическими ускорителями.
3. Реализован программный комплекс на основе разработанного алгоритма с использованием технологий CUDA и MPI. Показана эффективность счета до 75% при использовании 768 GPU суперкомпьютера «Ломоносов».
4. С помощью разработанного программного комплекса проведены расчеты ряда газодинамических течений, включая моделирование обтекания вокруг профиля NACA0012 и DLR F6 на декартовой структурированной сетке, которые подтвердили высокую эффективность его работы.

Достоверность и обоснованность полученных результатов обеспечены строгостью используемого математического аппарата и подтверждаются сравнением результатов вычислительных экспериментов с известными в литературе экспериментальными и расчетными данными, а также данными, полученными с помощью других методов.

Апробация работы. Основные результаты работы докладывались на:

1. Ломоносовских чтениях, МГУ им. М.В. Ломоносова (Москва, 2011)
2. Международной научной конференции “Параллельные вычислительные технологии - 2012”. (Новосибирск, 2012) [14]
3. Международной научной конференции “Параллельные вычислительные технологии - 2013”. (Челябинск, 2013) [15]
4. International Conference MATHEMATICAL MODELING AND COMPUTATIONAL PHYSICS, 2013 (MMCP 2013), (Дубна, 2013) [16]
5. Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: многообразие суперкомпьютерных миров” (Абрау-Дюрсо, 2014) [17]
6. Пятой всероссийской конференции “Вычислительный эксперимент в аэроакустике” (Светлогорск, 2014) [18]
7. 13th International Conference on Parallel Computing Technologies (PACT-2015), (Петрозаводск, 2015) [19]
8. 14th International Conference on Parallel Computing Technologies (PACT-2017), (Нижний Новгород, 2017) [20]
9. 7th International Conference on Mathematical Modeling in Physical Sciences (Москва, 2018) [21]

Данная работа участвовала в следующих конкурсах с результатом:

1. Победитель первого этапа конкурса «Эффективное использование GPU-ускорителей при решении больших задач» (ОАО Т-Платформы, Москва, 2011).

2. 1-ое место в конкурсе научных статей Всероссийской конференции молодых ученых "Параллельные и распределенные вычисления"(Новосибирск, 2012).
3. 1-ое место в конкурсе «GPU: серьезные ускорители для больших задач» (ССОЕ МГУ, Nvidia, 2013)

Исследования, описанные в §5.5 пятой главы диссертации, были поддержаны грантом Министерства образования и науки РФ (договор № 14.G39.31.0001 от 13 февраля 2017 г.).

Публикации. Основные результаты по теме диссертации изложены в 6 печатных изданиях [1; 19; 22–25], 2 из которых изданы в журналах, рекомендованных ВАК ([22; 23]), 3 – в периодических научных журналах, индексируемых Web of Science и Scopus ([1; 19; 25]).

Личный вклад соискателя. Все исследования, изложенные в диссертационной работе, проведены лично соискателем в процессе научной деятельности. В публикациях в соавторстве с научным руководителем Меньшовым И.С. соискателю принадлежат описания параллельных алгоритмов и моделей, а также результаты вычислительных экспериментов; заимствованный материал обозначен в работе ссылками.

Структура и объем диссертации. Диссертация состоит из введения, шести глав, заключения и списка литературы. Работа представлена на 109 страницах, содержит 46 иллюстраций и 3 таблицы. Список литературы содержит 95 наименований.

Глава 1. Обзор

1.1 Этапы развития вычислительной техники

За последние десятилетия в области высокопроизводительных вычислений можно условно выделить несколько этапов. Самый ранний из них характеризуется использованием преимущественно последовательных программ для прикладных расчетов. Прирост производительности при этом обеспечивался совершенствованием архитектуры процессоров и повышением их частоты. Следующий этап связан с появлением

- многоядерных процессоров;
- систем кластерной архитектуры, когда несколько вычислительных узлов объединяются в единое решающее поле высокоскоростной сетью.

Первое связано с достижением предельной тактовой частоты в 3 – 4 Ггц, которую из-за физических ограничений уже не удается поднять выше даже с использованием более совершенного техпроцесса. Это привело к необходимости освоения технологий (таких, как OpenMP [26], Pthreads [27], Cilk [28]) и инструментария параллельного программирования в рамках модели общей памяти.

Однако увеличение количества ядер на процессорный чип также достигло своих пределов, в частности, по тепловыделению, и практически единственным путем для увеличения производительности вычислительных систем осталось наращивание узлов в кластерных установках. В части из них была аппаратная поддержка общей памяти, которая на других кластерах может быть реализована программными средствами, такими как Cluster OpenMP [29; 30]. Сравнительная простота использования техник параллельного программирования для многоядерного узла на таких кластерах нивелировалась плохой масштабируемостью получаемых программ, поэтому стандартом де-факто стала модель программирования с распределенной памятью, в которой параллельно выполняющиеся

процессы не имеют единого адресного пространства и обмениваются данными посредством пересылки сообщений с использованием библиотеки MPI. Данная модель еще более усложнила создание параллельных программ, поскольку явным образом требует решать от прикладного программиста задачи сетевого, достаточно низкоуровневого, взаимодействия процессов. Именно эффективность решения последней проблемы и определяет во многих случаях предел масштабируемости приложения, поскольку межпроцессное взаимодействие занимает все большую долю относительно общего времени работы программы с увеличением числа используемых вычислительных узлов.

Наконец, третий этап развития вычислительной техники уже связан с пределом роста числа узлов в кластерных системах, обусловленным уровнем энергопотребления. В наиболее производительных суперкомпьютерах из ТОП-500 на сегодняшний день он уже достигает значений более 10МВт [31]. Сама по себе организация инфраструктуры электроснабжения и, как следствие, охлаждения с таким уровнем мощности – нетривиальная и экономически затратная задача, не говоря уже об ее обслуживании в период «боевой» эксплуатации вычислительной системы. В связи с этим стала особо актуальной задача создания новой энергоэффективной архитектуры вычислителей, имеющей более высокое отношение числа выполняемых операций на единицу потребляемой мощности по сравнению с «традиционными» процессорными чипами. Чтобы приступить к решению этой задачи, необходимо определить наиболее ресурсоемкий компонент процессора.

На заре еще первого этапа развития время доступа к памяти было меньше времени выполнения операций над данными из нее, однако с совершенствованием технологических процессов это отношение сменилось на противоположное. Более того, в современных процессорах латентность оперативной памяти на 2 порядка превосходит время обработки одной вычислительной инструкции. Чтобы компенсировать эту разницу, в процессоры стали внедрять многоуровневую систему кэшей с механизмами вроде аппаратной преднакачки данных. В итоге, основную площадь процессорного кристалла стал составлять кэш, кото-

рый и потребляет значительную долю энергии [32]. Однако если значительно уменьшить объем кэша, то возникает необходимость реализовывать другой механизм компенсации задержек при обращении к памяти. Этим механизмом стала аппаратная поддержка массовой мультитредовости в графических ускорителях (GPU), получивших на сегодняшний день наиболее широкое распространение как альтернатива «классическим» процессорам. Каждое вычислительное ядро GPU способно держать активными десятки исполняемых потоков с быстрым переключением между ними. В случае выдачи обращения к памяти из текущего потока ядро переключается на другой, в котором данные уже на регистрах и готовы к операциям над ними, операции же обращения к памяти обслуживаются параллельно на отдельном конвейере. Таким образом, вместо ожидания получения данных из памяти выполняются операции в других потоках данного ядра, что и обеспечивает толерантность к задержкам при обращении в память. Основная часть кристалла GPU занята вычислительными ядрами, реализация же описанной поддержки мультитредовости занимает значительно меньшую часть в сравнении с размерами кэша на CPU. За счет большей «вычислительно полезной» доли кристалла графические ускорители имеют более высокие показатели производительности на ватт в сравнении с обычными процессорами, что и предопределило их массовое использование в суперкомпьютерах на третьем этапе развития.

1.2 Особенности архитектуры графических ускорителей

Таким образом, к основным компонентам кластерной системы – межузловой интерконнект, внутриузловая память, процессорные ядра – добавились новые – память и массив вычислительных ядер графического ускорителя. Усложнение архитектуры суперкомпьютеров привело к еще большим трудностям при разработке параллельных программ. Среди них можно выделить те, которые свя-

заны непосредственно с освоением новых элементов вычислительной системы - GPU. Модель программирования для них с использованием CUDA/OpenCL [33; 34] достаточно сильно отличается от OpenMP, Pthreads моделей для центральных процессоров (CPU) и имеет следующие основные особенности:

– иерархическая память. Помимо основной DRAM памяти ускорителя, разделяемой между всеми потоковыми мультипроцессорами и объем которой составляет порядка нескольких гигабайт, у прикладного программиста есть возможность использовать значительно более быструю т.н. shared memory, но ее объем ограничен несколькими десятками килобайт и доступностью лишь в пределах одного мультипроцессора. Несмотря на такие ограничения, грамотное ее использование позволяет в несколько раз повысить производительность для ряда задач.
– объединение транзакций обращения в DRAM (coalescing). В случае, если доступ к памяти из соседних нитей выполняется по адресам, лежащим в одном сегменте, то происходит объединение нескольких таких запросов в память в одну транзакцию, так что время доступа к памяти нескольких нитей становится равным времени обращения к ней лишь одной из них. Поскольку, несмотря на более высокую скорость, GDDR память графических ускорителей имеет большее значение латентности в сравнении с DDR RAM, для достижения эффективной работы программы на GPU необходимо, чтобы выполнялся coalescing большинства обращений в память, т.е. чтобы доступ к памяти имел регулярную структуру.
– нити (треды) объединяются в пучки (варпы), инструкции в которых выполняются синхронно, переключение между тредами на потоковом мультипроцессоре также выполняется целыми варпами. Именно внутри каждого варпа и выполняется coalescing. Таким образом, GPU можно представить как мультитредовый процессор, на котором активны несколько «тяжелых» тредов, каждый из которых состоит из пучка «легких» синхронных нитей.

– жесткое ограничение числа доступных регистров на тред. Операции над данными могут быть выполнены только после того, как они загружены в регистры. В случае, если переменные занимают объем больше размера регистрового файла, то они физически располагаются в медленной DRAM памяти, откуда по мере необходимости считываются в регистры и записываются обратно. Это так называемый registry spilling, порождающий дополнительные обращения в память и снижающий общую производительность. Поскольку в наиболее распространенном сегодня поколении ускорителей Nvidia Fermi возможно использовать максимум 63 32-битных регистра на тред, во многих приложениях registry spilling является основным ограничителем производительности (в более новом поколении - Kepler - число регистров на тред увеличено до 255).

Помимо проблем с освоением графического ускорителя как самостоятельного вычислителя для масштабируемых кластерных приложений необходимо решать задачу внутри- и межузлового взаимодействия GPU. Внутри одного узла пересылка данными между GPU возможна напрямую с использованием GPU Direct, если 2 ускорителя расположены в одном pci-дереве (имеют общий root complex), т.е., как правило, находятся в pci-express разъемах одного процессорного сокета. Однако большинство суперкомпьютеров состоят из многосокетных узлов, и ускорители в них обычно разносятся по разным сокетам. В этом случае копирование данных между GPU возможно только через дополнительный буфер в оперативной памяти. При межузловых взаимодействиях может дополнительно использоваться системный буфер сетевого адаптера. В CUDA/OpenCL графический ускоритель рассматривается как пассивное устройство, т.е. запуск кода на нем, копирование данных выполняется только командами CPU. Таким образом, в параллельных программах для обмена данными между GPU используется следующая цепочка вызовов:

– копирование данных из памяти ускорителя в оперативную память посредством API CUDA/OpenCL;

– межпроцессная пересылка с использованием send- и recv-функций из MPI [35];
– копирование данных из оперативной памяти в память ускорителя с помощью CUDA/OpenCL;

Как правило, при реализации программ под GPU-системы с использованием MPI используется простая схема: счет на GPU -> обмен данными с помощью указанной выше цепочки вызовов с использованием синхронных функций -> счет на GPU и т.д. При использовании все большего числа узлов под задачу время, затрачиваемое на достаточно сложную процедуру обмена данными, составляет все большую часть от общего времени работы, поэтому масштабируемость программы сильно ограничена. Улучшить ее можно с использованием полностью асинхронной схемы работы, т.е. когда возможно одновременное выполнение

– кода на GPU;
– копирования данных из оперативной памяти в память ускорителя;
– копирования данных из памяти ускорителя в оперативную память;
– межпроцессного обмена данными.

Для ее реализации доступны асинхронные версии send-,recv-функций в MPI, async-функции копирования в CUDA с использованием потоков (stream). В каждом из этих API есть функции синхронизации, с помощью которых и возможно построение корректной схемы работы программы. Введение асинхронного исполнения позволяет, как будет показано ниже, достичь масштабирования, близкого к линейному, на нескольких сотнях узлов с графическими ускорителями. Однако этот механизм, добавляющий, безусловно, сложности при написании, отладке и без того непростых приложений, эффективен только при поддержке его в параллельном алгоритме, а разработка последнего с учетом описанной специфики суперкомпьютерной архитектуры, доказательство его корректности – отдельная, не менее важная задача.

1.3 Методы

Методы решения задач газовой динамики можно разделить на несколько видов по типу используемого в них сеточного разбиения расчетной области. В коммерческих продуктах типа ANSYS, Abacus для расчета течений сжимаемого газа применяются в основном нерегулярные сетки, которые относительно просто строятся для произвольной геометрии двух- и трехмерных расчетных областей. Данный тип сеток порождает нерегулярный доступ в память, что является сильным негативным фактором для GPU, т.к. при этом возникает множество uncoalesced обращений к памяти. И хотя есть работы, как, например, в [36], в которых путем переупорядочивания ячеек сетки удается частично структурировать доступ к памяти, производительность программ на нерегулярных сетках ограничивается пропускной способностью подсистемы памяти графического ускорителя, а не его скоростью обработки арифметических операций. Однако даже без специальных техник оптимизации размещения неструктурированной сетки в памяти множество CFD-пакетов, включая [37], увеличивают свою производительность в десятки раз по сравнению с одним ядром CPU при задействовании графических ускорителей [38] (одна из первых реализаций решателей на GPU), что демонстрирует большой потенциал применения GPU для решения задач газовой динамики.

Использование адаптивных сеточных разбиений, когда в процессе решения выполняется перестроение узлов сетки, еще более усугубляет проблему создания эффективного решателя для GPU. Регулярное же разбиение области с использованием в том числе многоблочных сеток, напротив, очень хорошо подходит под архитектуру графических ускорителей, поскольку программная реализация в этом случае предполагает структурированный доступ к памяти. Тем не менее, применение регулярных сеточных разбиений ограничено небольшим классом задач с простой геометрией, решаемых, как правило, в двумерной постановке, поскольку построение структурированных сеток, конформных к

геометрии вычислительных областей представляет сложную, а в большинстве случаев (особенно трехмерных) – неразрешимую задачу.

Таким образом, неструктурированные сетки могут применяться для задач с геометрией разной сложности, но плохо подходят для реализаций на GPU, структурированные же, напротив, позволяют эффективно задействовать архитектуру графических ускорителей, но их использование возможно лишь для задач с геометрически простой расчетной областью. Следует отметить, что построение высококачественных сеток обоих типов часто не может быть выполнено полностью автоматически, и требуется «ручное» вмешательство оператора, в связи с чем возникающие трудозатраты по времени могут быть сравнимы с длительностью последующих расчетов на таких сетках. В качестве метода, реализующего преимущества сеток разных типов, в данной работе используется метод свободной границы [10], для которого была разработана эффективная реализация на графических ускорителях.

Рассмотрим метод декартовых сеток (используемый, например, в [39]). В нем расчетная область представляется кубом, на котором строится простейшая декартова сетка. Границы твердых включений пересекают определенные ячейки сеточного разбиения, и дискретизация методом конечных объемов ведется по ячейкам двух типов - шестигранным, полностью лежащим в области течения, и по упомянутым усеченным. Достоинство метода - полное отсутствие проблем с построением сеток, но при этом есть и существенный недостаток - необходимость определения пересечения поверхностей твердых включений с ячейками сетки. Для задач со сложной геометрией, особенно в трехмерном случае, это может привести к довольно непростым ситуациям, требующим многовариантного анализа, причем не только на стадии инициализации задачи, но и во время непосредственных расчетов. В программной реализации этого метода для графических ускорителей регулярная структура обращений в память нарушается из-за наличия усеченных ячеек, для корректной работы с которыми вычислительное ядро необходимо усложнять множественными условными переходами

(учет всех возможных вариантов усеченных ячеек и их геометрических соседей), что в итоге приводит к неэффективной работе на GPU.

В семействе методов погруженной границы (immersed boundary method) [40] также используются простые декартовы сетки, в расчете при этом не задействуются усеченные ячейки, вместо этого выполняются специальные процедуры поиска пересечений границ твердых включений и модификации значений в пересекаемых ячейках. Алгоритмическая неоднородность этих методов не позволяет эффективно реализовывать их на графических ускорителях. К тому же, большинство из них разработаны для модели несжимаемой жидкости, их применение в решении задач газовой динамики потребовало бы внесение нетривиальных изменений (в случае, если они вообще возможны).

Наконец, метод свободной границы [10], основанный на идеях методов погруженной границы, не имеет указанных недостатков последних. В нем используются несвязные с поверхностью твердых включений декартовы сетки и вычислительное ядро алгоритмически однородно – фактически, расчет на внутренних (полностью лежащих внутри твердых включений), внешних и пересекаемых ячейках ведется по одним и тем же формулам. Внутренние граничные условия на поверхностях твердых включений моделируются введением специальной поправки – компенсационного потока – в правую часть определяющей системы уравнений. В стандартной постановке, например, в рамках уравнений Эйлера решается задача в области вне твердых включений с граничным условием непроникания на их поверхности. В модифицированной же постановке для метода свободной границы ищется решение во всей области, содержащей твердые включения, а граничные условия заменяются на фиктивные источники массы, импульса и энергии в правой части определяющей системы, которые выбираются так, чтобы проекция получаемого при этом решения на область вне включений совпадала с решением в стандартной постановке с граничными условиями на поверхностях твердых тел. Помимо использования декартовых сеток для геометрии расчетной области любой сложности данный метод обладает еще одни важным свойством - алгоритмической однородностью, которая сохраняется

и в случае подвижных включений. С учетом вышеизложенного, метод свободной границы очень хорошо подходит для реализации параллельных вычислений, в особенности под архитектуры графических ускорителей.

1.4 Явные и неявные схемы

Еще один признак, по которому можно разделить методы решения задач газовой динамики – тип используемой схемы интегрирования по времени. Явные схемы идеально подходят для распараллеливания на графических ускорителях, поскольку вычисления на одном шаге по времени над всеми ячейками расчетной области можно вести одновременно и независимо. Тем не менее, их применимость сильно ограничена условием устойчивости: пропорциональный рост вычислительной сложности при увеличении сеточного разрешения расчетной области усугубляется за счет необходимости вести расчет с меньшим шагом по времени. То есть, решение задач с использованием более подробной сетки для явных схем во многих случаях возможно лишь с неоправданно возрастающим числом вычислительных операций. Поэтому больший интерес представляют неявные схемы, свободные от жесткого ограничения на выбор шага интегрирования по времени. Они, как правило, сводятся к решению на каждом временном шаге больших систем линейных уравнений с сильно разреженной матрицей. Вместо прямых методов ввиду большой вычислительной сложности применяются итеративные методы решения полученных систем, такие как GMRES [41], BI-CGSTAB [42] и др. Соответствующие реализации этих методов для GPU и multi-GPU систем описаны, например, в [43; 44], они обладают высокой вычислительной сложностью и требуют большого объема памяти для хранения элементов матрицы системы. Умножение разреженной матрицы на вектор (SpMV) – операция, которая занимает значительную часть времени в работе указанных методов. Эффективность ее распараллеливания во многом

определяет производительность и масштабируемость всего итеративного солвера. В большинстве случаев в реализациях SpMV используется нерегулярный доступ к памяти, поэтому, во-первых, GPU-версии ограничены не вычислительной производительностью ускорителя (она, как правило, составляет лишь несколько процентов от пиковой [45]), а пропускной способностью памяти (memory-bound); во-вторых, сильно усложняется задача распределения работы на несколько GPU и организации обменов между ними. В результате, хотя скорость работы на ускорителях оказывается выше в несколько раз по сравнению с процессорной версией, эффективность использования GPU остается очень низкой, и масштабируемость на multi-GPU системах сильно ограничивается из-за значительных по времени пересылок данных между ускорителями. Кроме того, масштабируемость ухудшается и из-за использования коллективных операций (редукция, широковещательная рассылка) в межпроцессных взаимодействиях [46]. Поэтому актуальна задача разработки параллельного итеративного метода для неявных схем, эффективно работающего на системах с большим числом графических ускорителей. Один из таких методов - DP-LUR - предложен в [4]. Эффективная его работа достигается за счет отсутствия зависимости по данным при блочной декомпозиции расчетной области, т.е. параллельный алгоритм счета на multi-GPU аналогичен используемому для явных схем.

Однако плата за простую реализацию на графических ускорителях – почти трехкратное увеличение числа арифметических операций по сравнению с другим похожим методом – LU-SGS [47], в котором для достижения заданной точности решения требуется меньшее число итераций. Но существенной особенностью LU-SGS, значительно осложняющей построение параллельного алгоритма для него, является зависимость по данным. Существуют несколько параллельных версий этого метода. В одном из них [9] наряду с одновременным расчетом на подобластях присутствует и последовательный этап сбора граничных (для всех подобластей) ячеек в одном процессе, их обсчет и обратная рассылка, что уже теоретически ограничивает масштабируемость. Более того, с увеличением числа процессов для задачи фиксированного размера время па-

раллельного счета уменьшается, а последовательной стадии – увеличивается, поскольку число граничных ячеек между смежными подобластями возрастает. В другом подходе [48] используется wavefront-метод: параллельный расчет на двумерной сетке ведется в каждый момент времени по подобластям, имеющим целочисленные координаты (i, j), для которых $i + j = const$.

Упомянутые параллельные реализации LU-SGS выполнены с задействованием лишь CPU. Адаптация [9] для GPU и трехмерных задач не представляет интереса, поскольку к последовательной фазе счета добавятся еще и накладные расходы на передачу данных между GPU, что дополнительно ухудшит масштабируемость. Для другой версии [48] такая адаптация более оправдана, однако она имеет сильно ограниченную масштабируемость, например, для сетки менее, чем с 60 млн ячеек уже с использованием 16 GPU простои ускорителей будут занимать более половины общего времени решения задачи.

Глава 2. Базовый численный метод

Описание моделей и методов, описанных в данной главе, представлено в [1].

Рассматривается численное решение системы уравнений Эйлера, описывающей движение идеальной сжимаемой жидкости. Система определяющих уравнений в декартовых координатах записывается в консервативной форме

$$\frac{\partial \vec{q}}{\partial t} + \frac{\partial \vec{f}_k}{\partial x_k} = 0, \tag{2.1}$$

где подразумевается суммирование по индексу координатного направления k,

$\vec{q} = (\rho, \rho u_m, \rho E)^T$ – вектор консервативных переменных,

$\vec{f}_k = (\rho u_k, \rho u_k u_m + \delta_{km} p, \rho u_k H)^T$ – векторы потоков,

$m = 1,2,3$, δ_{km} – символ Кронекера, ρ, u_k, p – плотность, компоненты вектора скорости и давление соответственно, $E = e + \frac{1}{2}u_k^2$, $H = e + p/\rho$ – удельные полная энергия и энтальпия. Система уравнений 2.1 замыкается уравнением состояния, связывающим термодинамические параметры среды, которое в настоящей работе берется в форме уравнения состояния идеального совершенного газа с показателем адиабаты γ:

$$p = (\gamma - 1)\rho e.$$

В расчетах, которые будут представлены ниже, показатель адиабаты берется для случая воздуха, $\gamma = 1.4$.

Дискретизация по пространству выполняется декартовой сеткой, ориентированной вдоль координатных направлений, с шагами h_k , которые, вообще говоря, могут быть переменными. Применяя метод конечного объема к уравнениям 2.1, приходим к следующей системе полудискретных уравнений:

$$\frac{d\vec{q}_i}{dt} = -\left(\frac{\Delta\vec{F}_k}{h_k}\right)_i, \tag{2.2}$$

где i является обобщенным индексом ячейки, зависящим от индекса координатного направления k и принимающим значение порядкового номера ячейки в этом направлении.

Разность потоков в правой части 2.2

$$\Delta\vec{F}_k = \vec{F}_{k,i+1/2} - \vec{F}_{k,i-1/2}, \tag{2.3}$$

где $\vec{F}_{k,i+1/2}$ – численный поток, аппроксимирующий дифференциальный поток $\vec{f}_k$ на грани между ячейками i и $i+1$. Этот численный поток берется в виде функции двух векторных аргументов

$$\vec{F}_{k,i+1/2} = \vec{F}_k(\vec{z}_i^{+}, \vec{z}_{i+1}^{-}), \tag{2.4}$$

удовлетворяющей условию аппроксимации $\vec{F}_k(\vec{z},\vec{z}) = \vec{f}_k(\vec{z})$. Здесь $\vec{z}$ обозначает вектор примитивных переменных, $\vec{z} = (\rho, u_k, p)$.

Верхние индексы “$+$” и “$^{-}$” в правой части уравнений 2.3 указывают, что соответствующие величины берутся в центре грани $i+1/2$. Выбор этих величин определяется точностью схемы. Например, схема первого порядка точности по пространству получается при простом выборе

$$\vec{z}_i^{+}, = \vec{z}_i, \vec{z}_{i+1}^{-}, = \vec{z}_{i+1} \tag{2.5}$$

Для увеличения порядка аппроксимации схемы необходимо применять подсеточную реконструкцию параметров более высокого порядка точности [49]. В настоящей работе используется кусочно-линейное подсеточное восполнение, обобщающее на неравномерные сетки интерполяцию схемы MUSCL (Monotone upstream-centered scheme for conservation laws), [50]:

$$\vec{z}^{\pm} = \vec{z} \pm \frac{1}{2}\delta^{\pm}\left[(1-k^{\pm})\Delta^{\mp} + (1+k^{\pm})\Delta^{\pm}\right] \tag{2.6}$$

с конечными разностями $\Delta^+ = \vec{z}_{i+1} - \vec{z}_i, \Delta^- = \vec{z}_i - \vec{z}_{i-1}$. В этом уравнении для сокращения записи опущен индекс ячейки i, $\delta^+ = h_i/(h_i + h_{i+1}), \delta^- = h_i/(h_i + h_{i-1})$ – параметры неравномерности сетки, $k^\pm = k(\delta^\pm)$ – функция, определяющая порядок аппроксимации интерполяционной схемы. При выборе $k(\delta) = -1$ получается стандартная MUSCL-схема второго порядка точности [51], $k(\delta) = 1$ дает неустойчивую центрально-разностную схему второго порядка, $k(\delta) = 0$ отвечает схеме Фромма ([52]), $k(\delta) = (12\delta^2 - 1)/(12\delta)$ приводит к MUSCL схеме третьего порядка аппроксимации по пространству. Ниже используется этот (последний) вариант схемы.

Схемы порядка точности по пространству второго и выше являются немонотонными и приводят к нефизичным осцилляциям в численных решениях вблизи поверхностей сильных разрывов ([53]). Для подавления этих осцилляций используют ограничители производных ([49; 51]). Действие ограничителя сводится к модификации разностей $\Delta^\pm$ таким образом, чтобы интерполяция 2.6 не приводила к образованию локальных экстремумов.

Наиболее распространенными являются minmod-ограничитель ([49; 50]):

$$\Delta^+ = minmod(\Delta^+, \varphi\Delta^-),\ \Delta^- = minmod(\Delta^-, \varphi\Delta^+),$$
$$\varphi = \frac{3-k}{1-k},\quad minmod(x,y) = \begin{cases} 0, & \text{если } xy < 0, \\ sign(x)min(|x|,|y|) & \text{если } xy \geqslant 0, \end{cases} \tag{2.7}$$

и ограничитель Ван Альбада ([54]):

$$\vec{z}^{\pm} = \vec{z} \pm \frac{1}{2} s\delta^\pm \left[(1 - sk^\pm)\Delta^\mp + (1 + sk^\pm)\Delta^\pm\right],$$
$$s = max\left(0, \frac{2\Delta^+\Delta^-}{\Delta^+\Delta^+ + \Delta^-\Delta^- + \varepsilon}\right), \tag{2.8}$$

где ε – малое число, служащее для предотвращения деления на ноль ($\varepsilon \sim 10^{-12}$ для операций с двойной арифметикой, $\varepsilon \sim 10^{-6}$ – для операций с одинарной). Первый ограничитель производных не является непрерывно-дифференцируемой функцией и может приводить к зацикливанию невязки в процессе

сходимости решения, поэтому используется второй ограничитель в виде гладкой функции, который во многих задачах решает проблему зацикливания невязки.

Метод С.К. Годунова из [55] применяется для аппроксимации функции численного потока. В этом подходе поток вычисляется на решении автомодельной задачи Римана для системы локально-одномерных уравнений

$$\frac{\partial \vec{q}}{\partial t} + \frac{\partial \vec{f}_k}{\partial x} = 0 \tag{2.9}$$

с начальными данными

$$\vec{z}(0,x) \;=\; \begin{cases} \vec{z}_i^{+}, & x < 0, \\ \vec{z}_{i+1}^{+} & x \geqslant 0. \end{cases}$$

Решение этой задачи подробно описывается в [56]. Оно фактически сводится к решению одного линейного уравнения относительно давления контактной зоны, что может быть реализовано ньютоновскими итерациями. Остальные параметры решения имеют точные аналитические выражения. Обозначим это решение как

$$\vec{z}(t,x) = \vec{Z}^{R,k}\big(\lambda, \vec{z}_i^{+}, \vec{z}_{i+1}^{-}\big),$$

где $\lambda = x/t$ – автомодельная переменная. Тогда стандартный годуновский численный поток будет иметь вид

$$\vec{F}_k\big(\vec{z}_i^{+}, \vec{z}_{i+1}^{-}\big) = \vec{f}_k\Big[\vec{Z}^{R,k}\big(0, \vec{z}_i^{+}, \vec{z}_{i+1}^{-}\big)\Big]. \tag{2.10}$$

Расчет численного потока выполняется на каждом ребре счетной ячейки и является по сути основной (в смысле временных затрат) операцией расчетного цикла. Поэтому в целях повышения эффективности схемы С.К. Годунова были разработаны различные безытерационные приближенные решения задачи Римана ([57]). Одно из таких приближений приводит к численному потоку, предложенному в [58], который используется ниже в линеаризации численного потока. Он имеет следующий вид:

$$\vec{F}_k\left(\vec{z}_i^{+}, \vec{z}_{i+1}^{-}\right) = \frac{1}{2}\left\{ \vec{f}_k\left(\vec{z}_i^{+}\right) + f_k\left(\vec{z}_{i+1}^{-}\right) - (|u_k| + c)_{i+1/2}\left[\vec{q}\left(\vec{z}_{i+1}^{-}\right) - \vec{q}\left(\vec{z}_i^{+}\right)\right] \right\}, \quad (2.11)$$

где индекс $i + 1/2$ означает осреднение, например $(*)_{i+1/2} = 0.5\left[(*)_i^{+} + (*)_{i+1}^{-}\right]$.

Интегрирование по времени системы полудискретных уравнений 2.2 – 2.4 с численным потоком в форме 2.10 проводится с использованием явно-неявной схемы из [59], которая обеспечивает минимальное вовлечение диссипативной неявной компоненты, гарантируя при этом свойство невозрастания max нормы (НВМН) для случая линейных уравнений. При надлежащем выборе шага по времени эта схема автоматически переходит в базовую явную схему второго порядка точности по времени и пространству, удовлетворяющую свойству НВМН.

Построение гибридной явно-неявной схемы начинается с выбора базовой явной схемы. В качестве такой схемы мы берем двухшаговую схему типа предиктор–корректор, хотя и другой выбор тоже возможен. В этой схеме на первом шаге вычисляются значения предиктора на полушаге по времени по явной эйлеровой схеме:

$$\widetilde{\vec{q}}_i = \vec{q}_i^{\,n} - \frac{\Delta t}{2}\left(\frac{\Delta F_k(\vec{z}^{\,n})}{h_k}\right)_i \quad (2.12)$$

с потоком $F_{k,i+1/2}$, определяемым по интерполированным на грани ячейки значениям,

$$\Delta\vec{F}_k(\vec{z}^{\,n}) = \vec{f}_k(\vec{z}_i^{+}) - \vec{f}_k(\vec{z}_i^{-}). \quad (2.13)$$

Здесь верхний индекс n обозначает дискретный уровень по временной переменной. Решение на новом временном слое $n + 1$ получается по явной схеме второго порядка точности на шаге Δt с потоками, вычисляемыми по значениям предиктора:

$$\begin{gathered} \vec{q}_i^{\,n+1} = \vec{q}_i^{\,n} - \Delta t\left(\frac{\Delta F_k(\widetilde{\vec{z}})}{h_k}\right)_i; \\ \vec{F}_{k,i+1/2} = \vec{F}_k\left(\widetilde{\vec{z}}_i^{+}, \widetilde{\vec{z}}_{i+1}^{-}\right). \end{gathered} \quad (2.14)$$

Описанная выше схема типа предиктор–корректор гарантирует (в линейном случае) свойство НВМН при выполнении условия Куранта–Фридрихса–Леви на шаг интегрирования по времени,

$$\Delta \leqslant \lambda_i(\vec{z}^n) \text{ для всех } i, \tag{2.15}$$

где функция в правой части определяется локальными скоростью течения и скоростью звука:

$$\lambda_i(\vec{z}^n) = K_s \left(\frac{|u_k| + C}{h_k} \right)^{-1}, \tag{2.16}$$

где K_s – коэффициент запаса ($0 < K_s \leqslant 1$), C – локальная скорость звука.

Базовую явную схему можно записать в операторном виде:

$$\vec{q}_i^{n+1} = \vec{q}_i^n + \Delta t L_2\big(\Delta t, \vec{q}^n\big), \tag{2.17}$$

где $L_2(\cdot)$ обозначает дискретный двухшаговой оператор перехода 2.12 – 2.14. Введем для каждой счетной ячейки параметр промежуточного слоя – скалярный параметр ω_i, $0 \leqslant \omega_i \leqslant 1$. Определим также вектор решения на промежуточном слое $\vec{q}^{\omega} = \omega \vec{q}^n + (1 - \omega)\vec{q}^{n+1}$. Тогда гибридная явно-неявная схема записывается в виде

$$\vec{q}_i^{n+1} = \vec{q}_i^n + \Delta t L_2\big(\omega_i \Delta t, \vec{q}^{\omega}\big). \tag{2.18}$$

Очевидно, что при выборе $\omega_i = 1$ во всех счетных ячейках гибридная схема переходит в точности в ее явный аналог. В другом предельном случае $\omega_i = 0$ она становится чисто неявной эйлеровской схемой с пространственной аппроксимацией второго порядка точности.

Неявная составляющая гибридной схемы привносит излишнюю диссипацию, которая стабилизирует численное решение, но при этом занижает его точность. Поэтому оптимальным решением в конструировании гибридной схемы будет выбор наибольших значений параметра промежуточного слоя ω_i в

счетных ячейках, обеспечивающий максимальное участие явной компоненты при условии выполнения свойства НВМН.

Оказывается, что такой выбор можно сделать. Необходимо заметить, что гибридная схема может быть переписана в эквивалентной форме как

$$\vec{q}_i^{n+1} = \vec{q}_i^{\omega} + \omega_i \Delta t L_2\left(\omega_i \Delta t, \vec{q}^{\omega}\right). \tag{2.19}$$

Нетрудно увидеть, что уравнение 2.19 в точности представляет явную двухшаговую схему перехода с переменного промежуточного слоя $t_\omega = \omega t^n + (1-\omega)t^{n+1}$ на верхний слой t^{n+1} . Поэтому невозрастание max нормы

$$\left\|\vec{q}^{n+1}\right\|_\infty \leqslant \left\|\vec{q}^{\omega}\right\|_\infty \tag{2.20}$$

будет выполняться при условии

$$\omega_i \Delta t \leqslant \lambda_i\left(\vec{z}^{\omega}\right). \tag{2.21}$$

Следствием неравенства 2.20 является неравенство $\left\|\vec{q}^{n+1}\right\|_\infty \leqslant \left\|\vec{q}^{n}\right\|_\infty$ ([59]). Поэтому неравенство 2.21 является также необходимым условием выполнения свойства НВМН численных решений гибридной схемы.

Неравенство 2.21 будет выполняться, если положить

$$\omega_i = min\left[1,\ \frac{\lambda_i\left(\vec{z}^{\omega}\right)}{\Delta t}\right] \tag{2.22}$$

или, с учетом уравнения 2.16,

$$\omega_i = min\left[1,\ \frac{K_s}{\Delta t}\left(\frac{|u_k^{\omega}| + C^{\omega}}{h_k}\right)_i^{-1}\right]. \tag{2.23}$$

Как видно, параметр промежуточного слоя в каждой счетной ячейке зависит от решения на верхнем временном слое. Таким образом, гибридная схема является фактически неявной, требующей решения нелинейной системы уравнений на каждом временном шаге. Необходимо также отметить, что выбором

величины шага Δt можно уменьшить число решаемых в системе уравнений, переведя в части счетных ячеек гибридную схему в чисто явную. Более подробно решение системы нелинейных уравнений обсуждается ниже.

2.1 Модель свободной границы

Для реализации внутренних граничных условий на декартовой сетке, не связанной с геометрией области решения, используется модификация метода свободной границы (МСГ) ([10]). Далее приводятся основные положения этого метода.

Пусть Ω является областью, занятой твердым телом, $\Gamma = \partial\Omega$ – его граница. Тогда уравнения Эйлера 2.1 описывают течение газа в области $\Re^3 \backslash \Omega$ и граничным условием для этих уравнений служит соотношение

$$u_k n_k = 0, \quad \vec{x} \in \Gamma, \tag{2.24}$$

где $\vec{n} = (n_k)$ – вектор единичной внешней к Ω нормали на поверхности Γ.

Суть метода свободной границы сводится к замене решения краевой задачи для однородной системы уравнений Эйлера в ограниченном пространстве решением неоднородной системы во всем пространстве. Исходная система уравнений модифицируется путем добавления в правую часть некоторого вектора $\vec{F}_w$, который далее называется *компенсационным потоком*:

$$\frac{\partial \vec{q}}{\partial t} + \frac{\partial \vec{f}_k}{\partial x_k} = -\vec{F}_w. \tag{2.25}$$

По своей структуре модификация уравнений 2.25 напоминают штрафные функции, которые используются в методе погруженной границы (Immersed boundary method) ([60]). По сути же это разные подходы. Штрафные функции являются объемными и никак не учитывают подсеточную структуру геометрии. Компенсационные потоки, напротив, определяются на поверхности, задающей

геометрию объекта, и вычисляются, как будет видно ниже, с учетом геометрических характеристик объекта на подсеточном уровне.

Система модифицированных уравнений 2.25 решается во всем пространстве $\Re^3$, а компенсационный поток $\vec{F}_w$ подбирается таким образом, чтобы сужение решения уравнения 2.25 в $\Re^3$ на область $\Re^3 \backslash \Omega$ в точности совпадало с решением исходной краевой задачи. В [10] предлагается выражение для компенсационного потока $\vec{F}_w$, которое обеспечивает выполнение этого условия, в виде

$$\vec{F}_w = \begin{pmatrix} \rho u_k n_k \\ \rho u_k u_m n_k + (p - p_w) n_m \\ \rho u_k n_k H \end{pmatrix} \delta(\vec{x}, \Gamma), \tag{2.26}$$

где $\delta(\vec{x}, \Gamma)$ обозначает обобщенную функцию Дирака поверхности Γ, определяемую соотношением

$$\int\limits_V \delta(\vec{x}, S)\varphi(\vec{x})dV = \int\limits_{V \cap \Gamma} \varphi(\vec{x})dS \quad \forall V \in \Re^3. \tag{2.27}$$

Величина p_w в уравнениях 2.26 представляет собой мгновенную реакцию жесткой стенки на воздействие со стороны потока жидкости. Для случая идеального газа с показателем адиабаты γ эта реакция определяется в зависимости от знака нормальной компоненты относительной скорости следующим образом.

Если $u_k n_k < 0$, то давление стенки p_w соответствует давлению за фронтом формирующейся ударной волны и определяется по формуле ([61])

$$p_w = p\left[1 + \frac{\gamma(\gamma+1)}{4} M^2 + \sqrt{\gamma^2 M^2 + \frac{\gamma^2(\gamma+1)^2}{16} M^4} \right]. \tag{2.28}$$

В противном случае $u_k n_k > 0$ формируется волна разрежения, и давление стенки имеет вид

$$p_w = p\left[1 - \frac{\gamma - 1}{2} M \right]^{\frac{2\gamma}{\gamma-1}}. \tag{2.29}$$

Здесь M обозначает относительное число Маха, $M = u_k n_k / a$, где a – скорость звука.

Необходимо отметить, что выбор компенсационного потока в форме соотношений 2.27 – 2.29 не является единственным с математической точки зрения. Наверное, можно указать другие формы правой части, при которых краевое условие 2.24 будет также выполняться. Текущий выбор основан на естественном физическом обосновании. Поток F_w моделируется двумя составляющими. Первая компенсирует протекающие через поверхность Γ массу, импульс и энергию, а вторая определяет реакцию стенки и соответствующую ей работу.

При численной реализации МСГ предполагается, что поверхность Γ, задающая геометрию тела, представляется набором дискретных элементов $\Gamma = \bigcup_j \Gamma_j$. В большинстве случаев это плоские многогранники или треугольные пространственные элементы. Ячейки декартовой сетки разбиваются границей тела на три типа: жидкостные – те, которые полностью находятся вне Γ; твердые, лежащие целиком внутри Γ, и, соответственно, пересекаемые. Пересекаемые ячейки наряду с расчетным вектором определяются также подсеточной структурой геометрии тела. В настоящей работе используется линейное восполнение, которое задается объемной долей, занимаемой жидкостью в пересекаемой ячейке, ω_f , и вектором внешней нормали (направленной в сторону жидкости) $\vec{n}_f$, $|\vec{n}_f| = s_f$, где s_f – площадь плоского элемента, аппроксимирующего пересечения счетной ячейки с поверхностью Γ.

Эти параметры вычисляются при анализе пересечения счетной ячейки с элементами поверхности Γ_j. Вектор нормали вычисляется осреднением единичных нормалей элементов с весовыми коэффициентами, равными площадям пересечения: $\vec{n}_f = \vec{n}_j s_j$, $s_j = area(\Gamma_j \cap C)$, где обозначает область ячейки сетки. Объемную долю ω_f в пересекаемой ячейке можно приближенно вычислить через средние расстояния от жидких вершин ячейки, d^+, и, соответственно, от твердых вершин, d^-, до плоскости, нормальной к $\vec{n}_f$ и проходящей через опорную точку $\vec{x}_0$:

$$\omega_f = \frac{d^+}{(d^+ + d^-)}. \tag{2.30}$$

Опорная точка определяется по центрам элементов поверхности $\vec{x}_j$:

$$\vec{x}^j = \frac{\left|\vec{n}^{j-1}\right|\vec{x}^{j-1} + x_j S_j}{\left|\vec{n}^{j-1}\right| + s_j}, \quad \vec{n}^j = \sum_{k=1}^{j} \vec{n}_k s_k, \quad \vec{x}_0 = \vec{x}^M, \tag{2.31}$$

где – число элементов поверхности Г.

Дискретизация системы уравнений 2.25 проводится в два этапа с применением метода разделения по физическим процессам. На первом этапе выполняется интегрирование однородной системы на множестве жидких и пересекаемых ячеек по численной схеме, рассмотренной в предыдущем разделе:

$$\begin{gathered} \vec{q}_i^{\,*} = \vec{q}_i^{\,n} - \Delta t \left(\frac{\Delta \vec{F}_k(\widetilde{\vec{z}})}{h_k} \right)_i; \\ \vec{F}_{k,i+1/2} = \vec{F}_k\left(\widetilde{\vec{z}}_i^{\,+}, \widetilde{\vec{z}}_{i+1}^{\,-}\right). \end{gathered} \tag{2.32}$$

Таким образом, решение на этом этапе, $\vec{q}_i^{\,*}$, описывает изменение газодинамических параметров на временном шаге в соответствии с моделью Эйлера, но без учета влияния стенки на параметры течения.

На втором этапе полученное решение корректируется компенсационным потоком правой части. Вывод соответствующих дискретных уравнений получается, исходя из физических соображений, упомянутых выше.

Беря решение $\vec{q}_i^{\,*}$ в качестве начальных данных, интегрируется однородная система уравнений Эйлера по жидкостной части счетной ячейки:

$$\omega_f V_i \frac{d\vec{q}_i}{dt} = -\sum_{\sigma \in f} \vec{F}_\sigma s_\sigma + \vec{F}_p s_f, \tag{2.33}$$

где $\vec{F}_p = (0, p_w n_m, 0)^T$ – численный поток на стенке, суммирование в правой части ведется по граням ячейки, полностью или частично находящимся в жидкости, s_σ – площадь жидкостной части грани. Поскольку на этом этапе учитывается только влияние стенки, потоки на гранях $\vec{F}_\sigma$ вычисляются по значениям $\vec{q}_i^{\,*}$ из ячейки.

Используя свойство консервативности потока, уравнение 2.33 можно переписать в виде

$$\omega_f V_i \frac{d\vec{q}_i}{dt} = -\vec{F}_c s_f + \vec{F}_p s_f, \tag{2.34}$$

с потоком $\vec{F}_c = (\rho u_k n_k, \rho u_k u_m n_k + p n_m, \rho u_k, n_k H)^T$ и $V_i = h_1 h_2 h_3$. Сумма потоков в правой части как раз дает введенный выше компенсационный поток, $\vec{F}_w = \vec{F}_c - \vec{F}_p$.

Для интегрирования по времени уравнения 2.34 используется неявная схема, чтобы не накладывать жесткие ограничения на шаг по времени из-за объемной доли ω_f, которая в некоторых пересекаемых ячейках может быть, вообще говоря, сколь угодно малой величиной. Это приводит к системе дискретных уравнений

$$\vec{q}_i^{\,n+1} = \vec{q}_i^{\,*} - \frac{\Delta t s_f}{\omega_f V_i} \vec{F}_w\left(\vec{q}_i^{\,n+1}\right). \tag{2.35}$$

Объединяя 2.32 и 2.35, получается следующая численная схема:

$$\begin{aligned} \vec{q}_i^{\,n+1} = \vec{q}_i^{\,n} - \Delta t \left(\frac{\Delta \vec{F}_k(\widetilde{\vec{z}})}{h_k} \right)_i - \frac{\Delta t s_f}{\omega_f V_i} \vec{F}_w\left(\vec{q}_i^{\,n+1}\right), \\ \vec{F}_{k,i+1/2} = \vec{F}_k\left(\widetilde{\vec{z}}_i^{\,+}, \widetilde{\vec{z}}_{i+1}^{\,-}\right), \end{aligned} \tag{2.36}$$

которая является абсолютно устойчивой при вычислении параметра промежуточного слоя по формуле 2.23 и обеспечивает третий порядок точности по пространству и второй порядок по времени.

Решение нелинейной системы уравнений 2.36 находится безматричным методом приближенной факторизации LU-SGS ([3; 62]). Этот метод экономичен и сводится к прямому и обратному циклам по счетным ячейкам, которые фактически реализуют явную схему. Детали метода решения описываются ниже.

2.2 Решение систем дискретных уравнений

Для решения уравнений 2.36 используется итерационный метод Ньютона. Он приводит к линейной системе относительно итерационных вариаций $\delta^s(\cdot) = (\cdot)^{n+1,s+1} - (\cdot)^{n+1,s}$, где s – итерационный индекс:

$$\left(I + \frac{\Delta t s_f}{\omega_f V_i} A_w^s\right)\delta^s \vec{q}_i = -\Delta^s \vec{q}_i - \Delta t \left(\frac{\Delta^s \vec{F}_k}{h_k}\right)_i - \\ -\frac{\Delta t s_f}{\omega_f V_i}\vec{F}_w\left(\vec{q}_i^{n+1,s}\right) - \Delta t \frac{\delta^s \vec{F}_{k,i+1/2} - \delta^s \vec{F}_{ki-1/2}}{h_k}. \tag{2.37}$$

В этом уравнении $\Delta^s = (\cdot)^{n+1,s} - (\cdot)^n$ обозначает итерационное приращение на временном шаге, $A_w = \partial \vec{F}_w / \partial \vec{q}$ – якобиан компенсационного потока по вектору консервативных переменных.

Линеаризация численного потока $\delta^s \vec{F}_{k,i+1/2}$ выполняется приближенно с использованием следующих упрощений. Во-первых, несмотря на то, что параметр ω зависит от итерационных значений, при линеаризации он предполагается замороженным. Во-вторых, при линеаризации не учитываются подсеточные интерполяции расчетного вектора и считается, что численный поток зависит от значений параметров в счетной ячейке, как в стандартной схеме первого порядка точности. И в-третьих, в качестве функции численного потока берется не точно годуновский поток, а его приближение в форме из [58], которое определяется соотношением 2.11. При этих допущениях линеаризация численного потока сводится к простому выражению

$$\delta^s \vec{F}_{k,i+1/2} = (1 - \omega_i)\frac{A_{k,i}^{\omega,s} + \left(|u_k| + c\right)_{i+1/2}^{\omega,s}}{2}\delta^s \vec{q}_i + \\ +(1 - \omega_{i+1})\frac{A_{k,i+1}^{\omega,s} + \left(|u_k| + c\right)_{i+1/2}^{\omega,s}}{2}\delta^s \vec{q}_{i+1}, \tag{2.38}$$

где $A_{k,i}^{\omega,s} = A_k\left(\vec{z}_i^{\omega,s}\right) = \partial \vec{f}_k / \partial \vec{q}$ – якобиан потока в k-м направлении по вектору консервативных переменных и верхний индекс ω указывает на то, что значение параметра в счетной ячейке берется с промежуточного слоя.

С учетом уравнения 2.38 система уравнений 2.37 сводится к линейной системе относительно итерационных невязок $\delta^s\vec{q}$:

$$D_i^s\delta^s\vec{q}_i = \vec{R}_i^s - \frac{\Delta t(1-\omega_{i+1})}{h_k}\left[\frac{A_{k,i+1}^{\omega,s} + \left(|u_k|+c\right)_{i+1/2}^{\omega,s}}{2}\right]\delta^s\vec{q}_{i+1} + \\ + \frac{\Delta t(1-\omega_{i-1})}{h_k}\left[\frac{A_{k,i-1}^{\omega,s} + \left(|u_k|+c\right)_{i-1/2}^{\omega,s}}{2}\right]\delta^s\vec{q}_{i-1}, \tag{2.39}$$

где

$$D_i^s = I + \frac{\Delta t s_f}{\omega_f V_i}A_w^s + \frac{\Delta t(1-\omega_i)}{2h_k}\left[\left(|u_k|+c\right)_{i-1/2}^{\omega,s} + \left(|u_k|+c\right)_{i+1/2}^{\omega,s}\right], \\ \vec{R}_i^s = -\Delta^s\vec{q}_i - \Delta t\left(\frac{\Delta^s\vec{F}_k}{h_k}\right)_i - \frac{\Delta t s_f}{\omega_f V_i}\vec{F}_w\left(\vec{q}_i^{\,n+1,s}\right). \tag{2.40}$$

Для дальнейшего рассмотрения будет удобно в правой части уравнения 2.39 сделать обратную линеаризацию, перейдя от произведения якобиана и итерационного приращения консервативного вектора к приращению соответствующего потокового вектора. Обозначая спектральный радиус якобиана через $r_k = |u_k| + c$, результирующую систему уравнений можно привести к следующему виду:

$$D_i^s\delta^s\vec{q}_i = \vec{R}_i^s - \frac{\Delta t(1-\omega_{i+1})}{2h_k}\left(\delta^s\vec{f}_{k,i+1}^{\omega} - r_{k,i+1/2}^{\omega,s}\delta^s\vec{q}_{i+1}\right) + \\ + \frac{\Delta t(1-\omega_{i-1})}{2h_k}\left(\delta^s\vec{f}_{k,i-1}^{\omega} - r_{k,i-1/2}^{\omega,s}\delta^s\vec{q}_{i-1}\right). \tag{2.41}$$

Решение этой системы ищется безматричным итерационным методом LU-SGS ([62]). Матрица системы является семидиагональной. Запишем ее в расщепленном операторном виде:

$$D\delta\vec{q} + L(\delta\vec{q}) + U(\delta\vec{q}) = \vec{R}, \tag{2.42}$$

введя верхнетреугольный матричный оператор

$$U(\delta\vec{q}) = \frac{\Delta t(1-\omega_{i+1})}{2h_k}\left(\delta^s \vec{f}^{\omega}_{k,i+1} - r^{\omega,s}_{k,i+1/2}\delta^s \vec{q}_{i+1}\right)$$

и нижнетреугольный матричный оператор

$$L(\delta\vec{q}) = \frac{\Delta t(1-\omega_{i-1})}{2h_k}\left(\delta^s \vec{f}^{\omega}_{k,i-1} - r^{\omega,s}_{k,i-1/2}\delta^s \vec{q}_{i-1}\right)$$

Переписывая систему уравнений 2.42 в виде

$$(D+L)D^{-1}(D+U)\delta\vec{q} = \vec{R} + LD^{-1}U, \tag{2.43}$$

для нахождения решения затем используется ее приближенная факторизация, которая состоит в отбрасывании последнего слагаемого в правой части уравнения 2.43. В результате получается упрощенная система

$$(D+L)D^{-1}(D+U)\delta\vec{q} = \vec{R}, \tag{2.44}$$

которая эффективно решается двумя расчетными циклами по ячейкам в прямом и обратном на- правлениях (по индексу ячейки) соответственно:

$$\begin{aligned} \delta\vec{q}^{*} &= D^{-1}[\vec{R} - L(\delta\vec{q}^{*})], \\ \delta\vec{q}^{s} &= \delta\vec{q}^{*} - D^{-1}U(\delta\vec{q}^{s}). \end{aligned} \tag{2.45}$$

Необходимо отметить, что отбрасываемые в описанной выше приближенной факторизации члены оказываются по порядку малости величинами $O(\Delta t^2)$. Поэтому, если при решении задачи используется малый шаг по времени, внесенная факторизацией ошибка будет небольшой.

Глава 3. Параллельный алгоритм для метода LU-SGS

3.1 Схема безматричных вычислений в LU-SGS

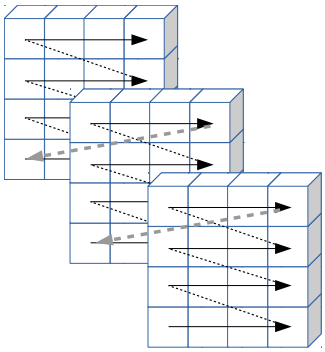
Как видно, в уравнениях линейной системы 2.41 – 2.45 неизвестные $\delta\vec{q}_i$ – итерационные вариации в ячейках сеточного разбиения расчетной области – входят с индексом i, т.е. изначально задается и фиксируется *глобальное линейное упорядочивание* всех неизвестных (ячеек сеточного разбиения). Данное упорядочивание определяет структуры $L(\delta\vec{q})$ и $U(\delta\vec{q})$ (2.42) – нижне- и верхнетреугольного матричных операторов.

Рассмотрим двумерный случай декартового сеточного разбиения расчетной области с часто используемым упорядочиванием ячеек “слева - направо”, “снизу - вверх”, тогда соответствующая матрица линейной системы 2.42 $A = L + U + D$ будет иметь 5-диагональную структуру (рис. 3.1). Для трехмерного декартового сеточного разбиения расчетной области с упорядочиванием ячеек “слева - направо”, “снизу - вверх”, “позади - вперед”, матрица A уже будет иметь 7-диагональную структуру (рис. 3.2).

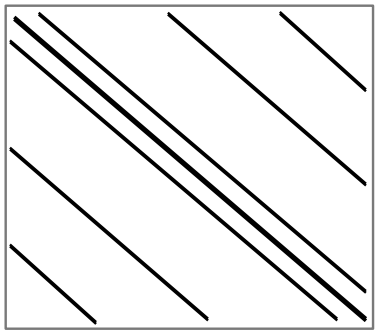
а) б)

Рисунок 3.1 — Обход двумерной области (а) и соответствующая структура матрицы СЛАУ (б)

Решение подсистем 2.45 при этом не требует обращения и хранения в явном виде матриц L, U и D, поскольку оно сводится к *обходу* в прямом (для



а)



б)

Рисунок 3.2 — Обход трехмерной области (а) и соответствующая структура матрицы СЛАУ (б)

подсистемы с матрицей L) и обратном (для подсистемы с матрицей U) направлениях всех ячеек сеточного разбиения расчетной области в последовательности, заданной *глобальным линейным упорядочиванием* при определении матричных операторов $L(\delta\vec{q})$, $U(\delta\vec{q})$ и $D(\delta\vec{q})$ (2.42). При таком итеративном обходе в прямом направлении на i-ом шаге для нахождения решения в i-ой ячейке необходимо определить ненулевые коэффициенты в i-х строках матриц L и D^{-1}. Для матрицы L таковые будут находиться в j-х столбцах, $j = j_1, j_2, ...$, причем номера этих столбцов будут равны порядковым номерам геометрически соседних (по граням) ячеек, которые находятся *перед* i-ой в выбранном обходе, т.е. $j < i$. Связь ненулевых коэффициентов в L с геометрическим соседством и ограничение их числа по этому признаку являются прямым следствием из исходной численной схемы 2.36, в которой для поиска решения в каждой ячейке на новом шаге необходимо вычислять потоки на ее гранях, а следовательно, использовать векторы состояния геометрически соседних (по граням) ячеек, данные из других ячеек при этом явно не используются. Для вычисления этих коэффициентов собственно и выполняется процедура нахождения численных потоков по всем граням для i-й ячейки. I-й диагональный элемент в D^{-1} требует лишь обращения D_i^s(2.40), для вычисления которого в свою очередь требуется лишь аппроксимация на гранях текущей i-й ячейки вектора состояния, т.е. опять-таки необходимо обращаться лишь к соседним ячейкам. Наконец, для вычисления $\vec{R}_i^s$ (2.40) используется вектор состояния текущей ячейки и численные потоки на всех ее гранях.

Из сказанного выше следует, что решение δq^* первой подсистемы из 2.45 можно найти гауссовой процедурой без явного хранения всей ее матрицы: достаточно лишь выполнить в прямом направлении итеративный обход ячеек в соответствии с исходной (служащей для определения матричных операторов $L(\delta\vec{q})$ и $U(\delta\vec{q})$) *глобальной линейной* нумерацией всех ячеек расчетной области. Для i-й ячейки в этом обходе вектор состояния аппроксимируется на каждую из ее граней с учетом значений из соответствующей соседней (по данной грани) j-й ячейки с последующим нахождением численных потоков на ней и аддитивным обновлением по ним D_i^s и R_i^s, затем, если $j < i$ (т.е. j-я ячейка расположена в обходе *до* i-й), выполняется вычисление матричного коэффициента L_{ij}. После завершения вычислений по всем граням i-й ячейки наконец определяется вектор решения $\delta\vec{q}^*$ по найденным D_i^s, R_i^s и всем ненулевым L_{ij}. Чтобы не дублировать вычисления потоков для двух соседних ячеек, они насчитываются для i-й ячейки, только если $i < j$, где j – номер соседней ячейки (т.е. она расположена в обходе *после* i-й), и добавляются как в i-ю, так и в j-ю (с противоположным знаком). Соответствующая схема вычислений приведена на рис. 3.3.

```
for each  i ∈ all_cells {
     for each j ∈ neighbors(i) {
          if (i > j) {
               L(i) += flux˜(i, j)
               L(j) -= flux˜(i, j)
          }
          else {
               U(i) += flux˜(i, j)
               U(j) -= flux˜(i, j)
          }
          D(i) += D˜(i, j)
          R(i) += R˜(i, j)
     }
     δq*(i) = R(i)
     for each j ∈ neighbors(i)
          if (i > j)
               δq*(i) -= L(j)
     δq*(i) *= D^-1(i)
}
```

Рисунок 3.3 — Схема вычислений для прямого обхода

Полностью аналогичным образом организуется и решение в безматричном виде для второй подсистемы 2.45 с матрицей $L + U$: выполняется обход всех ячеек, но уже в противоположном направлении (относительно решения первой подсистемы). Поскольку для i-й ячейки D_i^s уже вычислен, необходимо определить только ненулевые U_{ij}, т.е. вычисления по граням выполняются только в том случае, если $i > j$, где j – номер соответствующей соседней ячейки (т.е. она расположена в *обратном* обходе *до* i-й). Схема вычислений для решения данной подсистемы приведена на рис. 3.4.

```
for each  i ∈ all_cells {
      δq^s(i) = D(i) * δq*(i)
      for each j ∈ neighbors(i)
            if (i < j)
                  δq^s(i) -= U(j)
      δq^s(i) *= D^-1(i)
}
```

Рисунок 3.4 — Схема вычислений для обратного обхода

Подводя итог выше сказанному: все вычисления в методе LU-SGS можно проводить в безмаричной форме, т.е. без хранения в явном виде и обращения матриц СЛАУ. Изначально выбирается и фиксируется *глобальная линейная нумерация* всех ячеек сеточного разбиения, определяющая структуру матричных операторов L и U (2.42). В соответствии с таким упорядочиванием выполняется обход в прямом направлении по всем ячейкам; для каждой из них при этом производятся вычисления по всем ее граням с использованием значений вектора состояния из соответствующих соседних ячеек (т.е. выполняется отдельный локальный обход по всем геометрическим соседям текущей ячейки), причем эти вычисления зависят от того, *“до”* или *“после”* соседняя ячейка расположена в выбранном обходе относительно текущей. После завершения обхода в прямом направлении решение $\delta\vec{q}^*$ для 2.45 полностью вычислено, и запускается обход в обратном направлении. Схема вычислений в нем схожа с таковой в предыдущем (прямом): для каждой ячейки выполняется локальный обход по всем соседним

ячейкам, которые расположены *“до”* текущей (с учетом обратного направления обхода) для вычисления окончательного решения $\delta\vec{q}$.

Иными словами, все вычисления в LU-SGS определяются фиксированным линейным обходом всех ячеек в прямом и обратном направлениях с локальным обходом геометрических соседей для каждой из них.

3.2 Обзор подходов к распараллеливанию LU-SGS

Учитывая безматричную структуру вычислений при последовательном обходе ячеек в LU-SGS, естественным образом возникает задача построения параллельного алгоритма для него с сохранением данной структуры, т.е. без хранения в явном виде матрицы СЛАУ (2.42) и выполнением над ней соответствующих операций. Иными словами, необходимо организовать параллельные вычисления в LU-SGS на основе *линейного упорядоченного* обхода всех ячеек расчетной области. Сложность распараллеливания данного метода связана с возникающей зависимостью по данным при проведении вычислений в двух соседних ячейках: как было показано в предыдущем разделе, они зависят от того, как расположен сосед в выбранном линейном обходе относительно текущей ячейки (*“до”* или *“после”*).

Рассмотрим подход к распараллеливанию при “стандартном” (“слева - направо”, “снизу - вверх”) обходе дмумерной декартовой сетки (рис. 3.1). В случае ее декомпозиции на две подобласти горизонтальной линией, проходящей по центру области, расчет верхней части не начнется до тех пор, пока не будут обсчитаны ячейки в нижней до верхней границы. Это требование возникает из-за того, что при обращении из верхней подобласти к соседним ячейкам из нижней последние должны быть *уже* обсчитаны (в соответствии с выбранным исходным обходом *всей* области). Таком образом, из-за этой зависимости по данным в соседних ячейках на границе подобластей сначала почти полностью (за ис-

ключением верхнего ряда ячеек) обсчитывается нижняя подобласть, а затем уже начинается расчет в верхней, и общее время счета будет в итоге незначительно отличаться от полностью последовательного расчета исходной недекомпозированной области (рис. 3.1).

Как видно из примера выше, простая декомпозиция области для нескольких вычислительных устройств не позволяет получить эффективно реализуемый параллельный алгоритм. Данный подход хорошо работает для явных схем, но в явно-неявной, для которой в итоге используется LU-SGS, линейная упорядоченность ячеек и возникающая вследствие этого зависимость по данным служит значительным препятствием для построения *эффективного масштабируемого* алгоритма. Данное обстоятельство, как правило, является причиной отказа от использования LU-SGS на *всей* расчетной области и его применения лишь на *отдельных* подобластях ([63]). В таком подходе указанная ранее зависимость по данным на границах соседних подобластей не возникает; для каждой из подобластей данные соседних ячеек (формирующих т.н. “ghost” слой), ей не принадлежащие (т.е. расположенные в соседних подобластях) трактуются как граничные условия (изменяющиеся при переходе от одной итерации LU-SGS к другой). Очевидно, что в этом случае проводимые вычисления не являются эквивалентными вычислениям в LU-SGS на *всей* расчетной области, поскольку на границах подобластей данный метод не соблюдается. Платой за простоту подобного подхода является падение скорости сходимости решения, на некоторых задачах для достижения заданной точности “упрощенным” методом требуется до 2-х раз больше итераций по сравнению с “точным” LU-SGS [5; 8]. Тем не менее, существуют высокоптимизированные реализации, которые дают весьма небольшое снижение скорости сходимости при увеличении числа задействуемых ядер [64] и обладают хорошей масштабируемостью – 96% на 512 ядрах CPU (< 700 пространственных ячеек на одно ядро).

Еще одной альтернативой избавления от зависимости по данным при распараллеливании является модифицирование самого метода LU-SGS. Такой подход реализован в DP-LUR ([4]). В нем работа с неявными компонентами,

определяемыми операторами $L(\delta\vec{q})$ и $U(\delta\vec{q})$ (2.42) заменяется на релаксационную процедуру, фактически реализующую для решения 2.41 метод Якоби на каждой своей итерации, с начальными значениями

$$\delta^s\vec{q}_i^{(0)} = (D_i^s)^{-1}\vec{R}_i^s \tag{3.1}$$

На m-й итерации, $m = 1, ...m_{max}$, выполняется обновление векторов решения с использованием значений, которые *все* были получены на предыдущей итерации (т.е. зависимости по данным между соседними ячейками нет):

$$\begin{aligned}\delta^s\vec{q}_i^m = (D_i^s)^{-1}\Big(\vec{R}_i^s - \frac{\Delta t(1-\omega_{i+1})}{2h_k}\big(\delta^s\vec{f}^{\omega}_{k,i+1} - r^{\omega,s}_{k,i+1/2}\delta^s\vec{q}_{i+1}^{m-1}\big) + \\ + \frac{\Delta t(1-\omega_{i-1})}{2h_k}\big(\delta^s\vec{f}^{\omega}_{k,i-1} - r^{\omega,s}_{k,i-1/2}\delta^s\vec{q}_{i-1}^{m-1}\big)\Big),\end{aligned} \tag{3.2}$$

с итоговым решением, определяемым как

$$\delta^s\vec{q}_i = \delta^s\vec{q}_i^{(m_{max})}. \tag{3.3}$$

Данный метод позволяет полностью избавиться от зависимости по данным и провести простое распараллеливание как для явных схем. Однако платой за такое упрощение является бóльшая вычислительная сложность, т.е. увеличенное время счета, ведь теперь внутри каждой ньютоновской итерации необходимо выполнять m_{max} релаксационных итераций 3.2 (как правило, $m_{max} = 2...8$). Действительно, суммарное время счета для падения невязки до заданного значения оказывается почти вдвое больше при использовании DP-LUR по сравнению с LU-SGS [5; 6].

Таким образом, чтобы избежать проблем с падением скорости сходимости решения и/или увеличением времени счета необходимо *в точности* соблюдать LU-SGS на *всей* расчетной области. Однако, как показано выше, простой подход, применяемый при распараллеливании явных схем, практически не приводит к сокращению времени счета по сравнению с последовательным расчетом. Поэтому приходится разрабатывать более сложные алгоритмы для организации

параллельных вычислений, и для этого LU-SGS предоставляет определенную “свободу выбора”. В самом деле, как было указано в разделе 3.1, результирующая матрица для СЛАУ 2.41 определяется *линейным упорядочиванием* всех ячеек расчетной области. Очевидно, что такое упорядочивание можно выполнить не единственным способом, более того, не обязательно, чтобы в нем ячейки с порядковыми номерами i и $i+1$ были геометрически соседними; зависимость же по данным носит *исключительно локальный* характер, т.е. в каждой ячейке необходимо учитывать относительное (“до” или “после”) положение в исходной линейной последовательности ячеек *только* для ее соседей, относительный порядок для ячеек, не являющихся соседними, не имеет значения. Исходя из этих двух соображений и строятся параллельные алгоритмы для LU-SGS. В их основе – выбор *специального упорядочивания* ячеек, для которого затем определяется их *обход*, позволяющий параллельно (т.е. в один момент времени) выполнять вычисления в нескольких подмножествах ячеек. Принципиальную важность здесь имеет *корректность* параллельного алгоритма, т.е. *полная его эквивалентность* вычислениям в последовательном обходе ячеек, выполняемым в соответствии с выбранным линейным их упорядочиванием, как того требует безматричная схема вычислений LU-SGS (3.1). Иными словами, результат параллельного расчета должен быть идентичен некоторому последовательному прототипу на каждой ньютоновской итерации.

Исходя из архитектур современных вычислительных систем кластерного типа параллельные алгоритмы можно разделить на две группы: для моделей с *общей* и соответственно *распределенной* памяти. Каждая такая система состоит из множества вычислительных модулей (узлов), объединенных между собой, как правило, высокоскоростной низколатентной сетью. Внутри каждого модуля вычислительные ядра имеют прямой доступ к своей локальной оперативной памяти в едином адресном пространстве, но при этом адресные пространства всех узлов отделены друг от друга, и обмен данными между ними осуществляется посредством пересылок пакетов по связывающей их сети. Соответственно для алгоритмов и реализующих их программ, предназначенных для работы только

на одном вычислительном модуле и называемых часто многопоточными, используется модель общей памяти, когда каждое вычислительное ядро может напрямую обратиться к любому участку памяти, выделенной для всей программы. Характерными представителями таких моделей являются OpenMP [26], CUDA [33] и OpenCL [34]. В случае же алгоритмов и программ, предназначенных для работы одновременно на множестве вычислительных модулей, все обмены между ними уже необходимо явно описывать в одной из моделей распределенной памяти, стандартом де-факто среди которых на сегодня стал MPI [35].

Параллельные алгоритмы для модели распределенной памяти имеют более сложную структуру по сравнению с таковыми для модели общей памяти, но при этом позволяют использовать все доступные ресурсы кластерной вычислительной системы, не ограничиваясь лишь одним вычислительным модулем в ней. Существуют, однако, и runtime-системы, позволяющие выполнять программы, разработанные для одного модуля, сразу на всем кластере, например, ScaleMP [65]. Эта возможность достигается за счет объединения ресурсов кластера в один виртуальный модуль, в котором программе напрямую доступны все его вычислительные ядра и память в едином виртуальном адресном пространстве.

Несмотря на привлекательность идеи запуска многопоточных программ на множестве модулей (узлов) кластера, их производительность в упомянутых runtime-системах часто оказывается весьма низкой. Связано это с тем, что часть обращений в память в едином виртуальном адресном пространстве преобразуется в сетевой обмен данными между вычислительными модулями, в случае если соответствующий виртуальный адрес отображается на физическую память модуля, отличного от того, на котором в данный момент выполняется процесс(поток)-инициатор обращения. Такие обращения занимают на порядки бóльшее время по сравнению с обращением в локальную память модуля, из-за них и происходит общее снижение производительности. Иными словами, в алгоритмах и программах для моделей систем с общей памятью при запуске их в

средах типа ScaleMP никак не учитывается возможность межузловых сетевых обменов данными, которые в итоге могут приводить к значительной деградации производительности. Более того, во многих случаях для алгоритмов, разработанных под системы с общей памятью, просто невозможно создать программную реализацию для множества вычислителей с распределенной памятью, если не использовать упомянутые выше runtime-системы. Т.е. разработка параллельного алгоритма для систем с распределенной памятью является, как правило, отдельной задачей наряду с проблемой создания алгоритма в рамках модели общей памяти. Метод LU-SGS как раз является характерным примером, когда для эффективной программной реализации необходимо разрабатывать параллельный алгоритм для двух уровней – внутриузлового в рамках модели общей памяти и межузлового с распределенной памятью.

В [66] был предложен т.н. wavefront алгоритм для LU-SGS в рамках модели общей памяти. Данный алгоритм предназначен для структурированных двумерных и трехмерных сеточных разбиений. Рассмотрим его на примере двумерной расчетной области с декартовой сеткой: вычисления выполняются в несколько последовательных итераций, на каждой n-й итерации запускается параллельный обсчет в ячейках с целочисленными координатами (i, j) таких, что $i + j = n$, т.е. он ведется фронтами вдоль линий $y = n - x$. Для трехмерного случая расчет ведется аналогичным образом, только фронт ячеек на каждой итерации располагается не на прямой, а в плоскости $x + y + z = n$.

Легко видеть, что на начальных и конечных итерациях число одновременно считаемых ячеек мало (включая и случаи, когда их меньше числа доступных вычислительных ядер), и, соответственно, будут часто выполняться синхронизации после каждой итерации. Всё это приводит к снижению производительности при небольших сеточных разрешениях и/или большом количестве вычислительных ядер, выделенных задаче.

Wavefront алгоритм дает хорошие результаты по производительности при относительно небольшом числе используемых вычислительных ядер, однако он не подходит под архитектуры графических ускорителей, где сотням вычисли-

тельных ядер для эффективной работы необходимо обсчитывать параллельно тысячи (для GPU последних поколений - десятки тысяч) ячеек, а использование средств синхронизации после каждой итерации сопряжено с значительными накладными расходами. Иными словами, параллельный алгоритм для GPU, эффективно задействующий его ресурсы, должен обладать следующими свойствами:

– возможность вести параллельный расчет в тысячах (десятках тысяч) потоков с минимальным числом, а в лучшем случае – вообще без зависимостей по данным между ними;
– минимальное число или полное отсутствие глобальных синхронизаций в GPU для организации корректной работы алгоритма.

Предлагаемый в данной работе параллельный алгоритм для LU-SGS удовлетворяет указанным свойствам и позволяет получать решение на одном ускорителе для системы в 2.45, используя всего 4 глобальных синхронизации независимо от сеточного разрешения расчетной области.

В работе [48] wavefront алгоритм был адаптирован для вычислительных систем с распределенной памятью. Исходная сетка “разрезается” на тайлы, образующие декартову решетку, и каждому процессу с двумерным номером (i, j) назначается тайл с идентичными координатами, далее расчет идет аналогично версии для общей памяти, но фронт на каждой итерации формируется уже не отдельными ячейками, а тайлами. Очевидно, что при такой организации вычислений бóльшая часть процессов будет всегда простаивать (за исключением процессов (i, j), для которых $i + j = n$ на n-й итерации), поэтому в трехмерном случае тайлы преобразуются в столбцы ячеек, которые нарезаются плоскостями на сегменты перпендикулярно 3-му координатному (z-)направлению, и каждому процессу уже назначается по столбцу ячеек. Далее организуется конвейерная схема вычислений, когда процессы (i, j) с $i + j = n$ выполняют счет в $k - n$-м своем сегменте, где k - номер обсчитываемого сегмента в процессе $(0, 0)$. Простои процессов в таком алгоритме минимизируются при увеличении числа сегментов в каждом процессе, т.е. когда их число значительно превы-

шает размеры дмумерной сетки процессов по каждому направлению. Тем не менее, указанные простои приводят к весьма ограниченной масштабируемости при увеличении числа вычислителей для задачи, а предложенная программная реализация данного wavefront алгоритма дополнительно ее ограничивает из-за отсутствия совмещения по времени обмена данными и счета: на весьма подробной сетке размером $256 \times 256 \times 128$ масштабируемость составила всего лишь 73.6 % на 4 ядрах CPU (21 млн ячеек на ядро) [48].

В [67] предложен параллельный алгоритм для LU-SGS на системах с распределенной памятью. В нем после декомпозиции сеточной области ячейки делятся на несколько подмножеств, состоящих из граничных и внутренних ячеек подобластей. Расчет всех внутренних ячеек подобластей выполняется параллельно в разных процессах, однако оставшиеся граничные ячейки обсчитываются по сути в квази-последовательном режиме: В i-й подобласти обсчитываются граничные ячейки закрепленными за ней процессами, все остальные процессы (назначенные другим подобластям) в этот момент простаивают, данная процедура итеративно выполняется для всех подобластей. Указанные простои процессов приводят в итоге к ограниченной масштабируемости предлагаемого параллельного алгоритма – на 50 процессорах эффективность (отношение достигнутого ускорения к линейному ускорению времени счета) составила 64 – 72% в задаче с 600 тыс. ячеек [67]. Стоит, тем не менее, отметить, что параллельный алгоритм в [67] работает с неструктурированными сетками, в данной же работе он будет построен для структурированных сеточных разбиений с декартовой топологией ячеек и обобщен в заключительной главе для случая сеток с рекурсивным разбиением на основе квадратичных/восьмеричных деревьев.

В [68] также предлагается еще один оригинальный алгоритм для систем с распределенной памятью как дальнейшее развитие алгоритма для общей памяти [69]. В отличие от предыдущей работы используется более сложная рекурсивная декомпозиция области, позволяющая выделять подмножества параллельно обсчитываемых ячеек. В алгоритме присутствует последовательная часть (вычисление невязок) и отсутствует совмещение по времени обмена

данными между параллельными процессами и счета в них, что позволяет предположить достаточно сильное ограничение по масштабируемости соответствующей программной реализации, однако результаты ее работы еще не опубликованы.

Одной из причин падения эффективности счета с увеличением числа используемых для задачи вычислителей являются межпроцессные обмены данными. При этом время, затрачиваемое на них, составляет все бóльшую долю относительно общего времени работы программы и, начиная с некоторого числа используемых вычислителей, становится превалирующим над временем полезных вычислений, т.е. накладные расходы на передачу сообщений между процессами уже занимают основное время работы программы. Поэтому при разработке масштабируемых параллельных алгоритмов для систем с распределенной памятью ключевыми являются следующих два фактора:

– сведение к минимуму, а в лучшем случае – полное исключение последовательного исполнения части вычислений и/или простоев одного или нескольких параллельно исполняющихся процессов. Это требование рассматривается при условии реализации алгоритма на гипотетической вычислительной системе с бесконечно быстрой сетью, т.е. с нулевыми накладными расходами на обмен данными. Иными словами, алгоритм должен обладать близким к линейному масштабированием при условии мгновенной передачи сообщений между вычислительными ядрами, или, для закона Амдала, $\alpha \to 0$ в

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}},$$

где α – доля от общего объёма вычислений, которая может быть получена только последовательными вычислениями, S_p – ускорение, которое может быть достигнуто на вычислительной системе, состоящей из p процессоров, в сравнении с однопроцессорным решением;

– организация межпроцессного обмена данными на фоне “полезных” (содержательных с т.з. решаемой задачи) вычислений. Это требование уже рассматривается с точки зрения разработки реализаций для реальных вычислительных систем, обладающих сетями с ограниченной пропускной способностью и ненулевыми задержками при передаче данных. Параллельный алгоритм будет обладать бóльшей масштабируемостью, если он позволяет минимизировать возможные простои вычислителей, связанные с пересылками сообщений между процессами, за счет совмещения по времени межпроцессных коммуникаций и вычислений.

Рассмотренные выше параллельные алгоритмы для LU-SGS не удовлетворяют в той или иной степени этим двум свойствам, вследствие чего их программные реализации обладают весьма ограниченной масштабируемостью. Представляемый же в нижеследующих разделах параллельный алгоритм для LU-SGS (в *точности* реализующий вычисления своего последовательного прототипа на *всей* расчетной области), напротив, удовлетворяет указанным свойствам, что позволило в итоге соответствующей программной реализации с использованием CUDA и MPI достичь масштабируемости, близкой к линейной, на вычислительных системах с несколькими сотнями графических ускорителей (глава 6).

3.3 Построение параллельного алгоритма для вычислительных систем с распределенной памятью

Как уже было сказано, особенностью метода LU-SGS является обход расчетной области: все ячейки линейно упорядочиваются, то есть задается порядок, в соответствии с которым выполняется обход по всем сеточным ячейкам. Порядок этот можно задавать множеством способов, причем нет требования, чтобы ячейки с общей гранью располагались в нем рядом. Вы-

числения выполняются для каждой ячейки, начиная с первой в соответствии с заданным последовательным порядком и заканчивая последней, причем для операций в текущей ячейке используются данные из геометрически соседних ячеек.

В зависимости от того, была или не была соседняя ячейка еще посчитана, выполняется та или иная ветвь вычислений, соотвествующий результат может записываться не только в текущую, но и в соседнюю ячейку (при этом соседняя ячейка не становится после этого обсчитанной), Рис. 3.3.

Параллельный счет основан на геометрической декомпозиции сеточной расчетной области, когда каждому (MPI-)процессу назначается одна из подобластей. Работа параллельной программы будет корректной, если получаемое решение будет совпадать с результатом, полученным соответствующей последовательной версией. Для этого обходы ячеек во всех параллельно исполняющихся процессах должны однозначно соответствовать определенному обходу в последовательной программе. Иными словами, задавать порядок обхода элементов сеточного разбиения необходимо глобально для всей расчетной области, а не в отдельных подобластях (процессах), то есть нужно однозначно упорядочивать и граничные ячейки разных подобластей.

Далее построение параллельного алгоритма [22] выполняется для двумерных областей со структурированными сетками (для трехмерного случая он строится прямым обобщением). После декомпозиции области каждому процессу назначается сеточная подобласть, топологически эквивалентная четырехугольнику, и все они располагаются «стык в стык», т.е. подобласти также являются элементами структурированной (макро-)сетки.

Между внутренними ячейками всех подобластей отсутствует в явном виде зависимость по данным (т.к. такие ячейки из разных процессов не являются геометрически соседними), поэтому их, вообще говоря, можно обсчитывать параллельно, обеспечивая корректность LU-SGS. Необходимо только упорядочить обсчет граничных ячеек во всех блоках, так чтобы это не приводило к ограниченной масштабируемости всего параллельного алгоритма.

Чтобы вычисления были полностью корректными и проводились с высокой эффективностью, все подобласти делятся на два типа - “черные” и “белые” (black и white) - так, что они формируют «шахматную» структуру (все геометрические «соседи» каждой подобласти имею отличный от нее тип). Для каждого типа подобластей задается свой порядок операций.

В “черных” подобластях:

1. Запускаются пересылки граничных ячеек K соседним (“белым”) подобластям (Рис. 3.5); в “белых” подобластях эти ячейки считаются еще не обсчитанными. На фоне этих пересылок выполняется обход всех ячеек, кроме тех, которые образуют “кольцо” из двойного ряда граничных ячеек. Если бы запускался обход по *всем* внутренним ячейкам (т.е. без “кольца” из одного ряда граничных ячеек), граничные ячеки могли бы отсылаться соседям как с обновленными, так и с необновленными компонентами L и U, Рис. 3.3. При обходе же ячеек без двойного периметра из граничных для “белых” подобластей всегда будут отсылаться немодифицированные ячейки K.

2. Выполняется ожидание получения ghost ячеек от “белых” соседей (которые уже обсчитаны).

3. Выполняется обход по оставшимся ячейкам из двойного граничного кольца.

В “белых” подобластях:

1. Выполняется обход по первой половине внутренних ячеек.

2. Выполняется ожидание получения ghost ячеек K от “черных” соседей (которые уже обсчитаны)

3. Выполняется обход по граничным ячейкам.

4. Запускаются пересылки граничных ячеек K соседним (“черным”) подобластям; в “черных” подобластях эти ячейки считаются уже обсчитанными. На фоне этих пересылок выполняется обход по оставшейся второй половине внутренних ячеек.

Обратный обход в блоках строится в соответствии с прямым: в black он начинается с “двойного периметра” приграничных ячеек в обратном порядке и

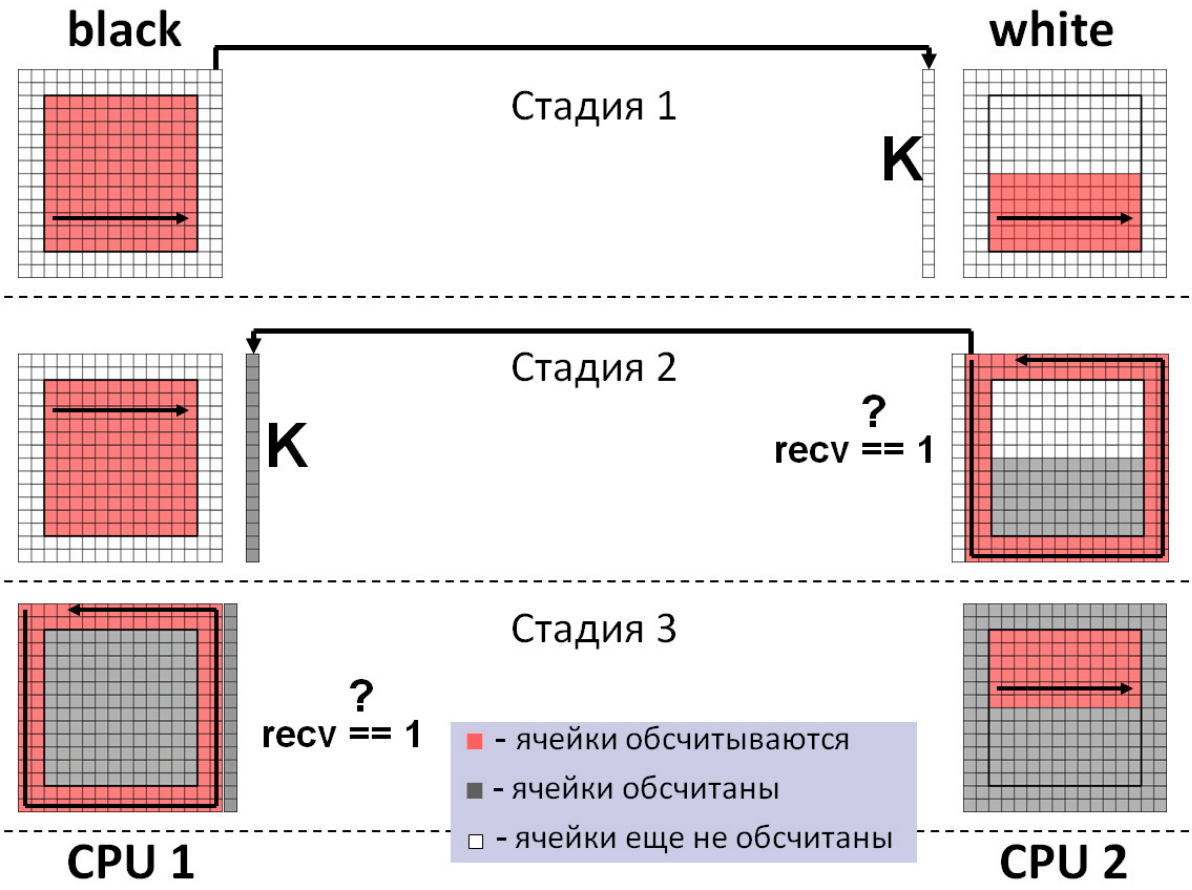


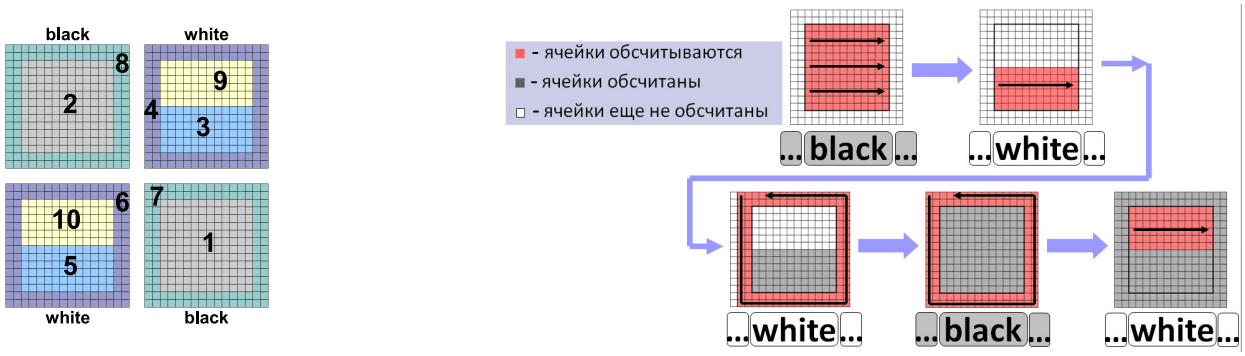
Рисунок 3.5 — Последовательность операций в “черных” и “белых” блоках

заканчивается на внутренней оставшейся их части (так же в обратном порядке); в white сначала выполняется обход второй половины внутренних ячеек, затем граничных и, наконец, оставшейся первой части (в обратном порядке). Все необходимые пересылки ячеек выполняются с последующей проверкой их завершения.

Если ячейки всей расчетной области равномерно распределены по подобластям, то описанный алгоритм будет работать эффективно, поскольку времена счета в разных процессах будут близкими друг к другу. Декомпозиция области с балансировкой на структурированной сетке на используемые в алгоритме подобласти делается тривиальным образом.

Построенный параллельный алгоритм обеспечивает точное соблюдение метода LU-SGS. Для доказательства этого утверждения необходимо глобально упорядочить ячейки *всей* сеточной области, последовательный обход по которым даст в точности такой же результат, как и параллельный алгоритм.

Упорядочивание ячеек для области, декомпозированной на 4 подобласти, показано на Рис. 3.6 а.



а) б)

Рисунок 3.6 — Порядок обхода ячеек для 4-х подобластей (а) и произвольного их числа (б)

Порядок обхода частей сеточной области определяется их номерами (от 1 до 10). Нетрудно видеть, что отношения между геометрически соседними ячейками («обсчитана/ не обсчитана») в данном обходе для разных частей в точности соответствуют таковым в параллельном алгоритме, и результат последовательных и параллельных вычислений будет идентичен.

В случае произвольной конфигурации $N \times M$ блоков, Рис. 3.6 б последовательный обход строится следующим образом: сначала выполняется обход всех ячеек без “двойного периметра” во всех блоках black в произвольном порядке, затем в каждом white первой половины внутренних ячеек и граничных ячеек (последовательность обхода самих блоков white произвольная), после обсчитываются все “двойные периметры” приграничных ячеек в black в произвольном порядке блоков, затем оставшиеся вторые половины внутренних ячеек в white (также произвольно). Нетрудно видеть, что, как и в случае с конфигурацией блоков 2×2 этот обход будет эквивалентным для параллельного алгоритма.

3.4 Модификация алгоритма для GPU

В предыдущей главе был изложен алгоритм распределения вычислений между MPI процессами, далее необходимо обеспечить параллельный счет внутри одного ускорителя. Архитектура GPU подразумевает одновременный счет многих ячеек, но здесь возникает проблема: как организовать порядок обсчета ячеек, чтобы положение в нем геометрически соседних ячеек оставалось неизменным относительно текущей, т.е. нужно гарантировать, что при обращении к каждой соседней ячейке она всегда была бы уже обсчитана (находится в очереди раньше текущей) или еще не обсчитана (в очереди – позже текущей). Эффективных средств глобальной синхронизации, которые помогли бы решить данную проблему, в современных GPU нет. Указанное условие приводит к задаче о раскраске графа: нужно “раскрасить” ячейки сетки минимальным количеством цветов так, чтобы любые геометрически соседние (по граням) всегда были разных цветов. В таком случае можно запускать на GPU параллельный счет сначала по ячейкам 1-го цвета, затем 2-го и т.д. – результат работы параллельного алгоритма будет всегда корректным, поскольку эквивалентный последовательный обход по ячейкам, дающий в точности такое же решение, выглядит следующим образом: сначала последовательно обходятся все ячейки 1-го цвета (в произвольном порядке), затем аналогично ячейки 2-го цвета и т.д.

Решение задачи о раскраске графа для декартовых (структурированных) сеток по сути являет собой свойство автомодельности предлагаемого параллельного алгоритма – разделение ячеек внутри подобласти (внутренней или граничной части блока) на два множества происходит аналогично разбиению блоков исходной расчетной области: две геометрически соседних ячейки всегда принадлежат разным множествам. Таким образом, два этих множества вновь образуют “шахматное” разбиение ячеек. В итоге, счет каждой подобласти блока – внутренней или граничной его части – выполняется в два последовательных этапа: сначала на GPU запускается счет ячеек из “black” (при этом все соседние

ячейки будут из множества “white”, которые в данной подобласти будут всегда еще не обсчитанными), затем, после его завершения, запускается счет ячеек из “white” (все соседние ячейки будут уже из “black”, которые в данной подобласти будут всегда уже обсчитанными). Обращение к ячейкам из другой подобласти всегда будет выполняться как к обсчитанным или необсчитанным – порядок в этом случае определяется порядком обсчета подобластей в блоке. Итоговый параллельный алгоритм для LU-SGS на системах с множеством графических ускорителей [19; 23] представлен на Рис. 3.7. Ключевой его особенностью является возможность полного совмещения по времени вычислений в GPU и обмена данными (в граничных ячейках) между ними, что в итоге позволяет достичь высокой масштабируемости на кластерных систмах, однако это требует соответсвующей поддержки в программной реализации алгоритма, детали которой рассматриваются в следующей главе.

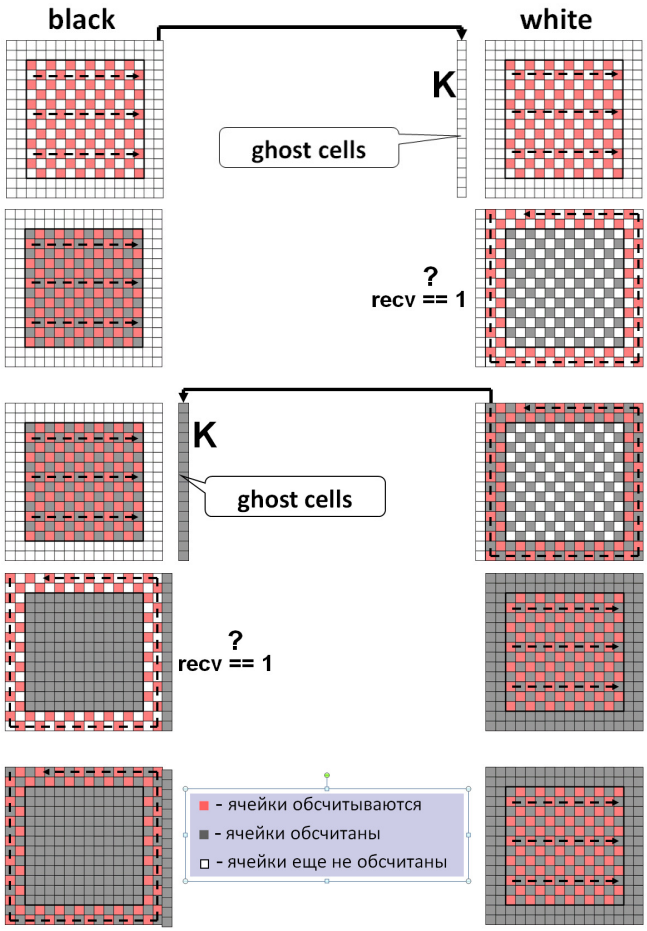


Рисунок 3.7 — Последовательность счета ячеек в black и white блоках при расчете на множестве GPU

Глава 4. Детали программной реализации

Для модели свободной границы §2.1 в каждой пересекаемой (с твердыми включениями) ячейке декартовой сетки необходимо определить следующие параметры: объемная доля ω_f твердого включения, площадь пересекаемой поверхности и нормаль к ней. Поскольку в текущей реализации твердые тела в расчетной области являются стационарными, данные параметры определяются на этапе старта задачи и не меняются в процессе ее решения. Геометрия твердых тел (например, профиля DLR F6 [70]), как правило, описывается набором параметрических поверхностей в одном из файловых форматов для CAD программ; для нее на этапе препроцессинга выполняется триангуляция, затем выполняется поиск пересечения отдельных полигонов (треугольников) с ячейками декартовой сетки. Для каждого такого пересечения определяется нормаль и площадь части полигона, оказавшейся внутри ячейки. В случае если одну сеточную ячейку пересекают несколько полигонов, выполняется процедура аппроксимации, в которой определяются нормаль и площадь для линейного приближения пересекаемой поверхности. Наконец, по сечению ячейки плоскостью (в которой расположены полигоны или которая является линейным приближением для них) определяется объемная доля твердого включения, расположенного в этой ячейке. Таким образом, геометрия твердых включений на декартовой сетке имеет кусочно-линейное представление.

Описанная адаптация математической и численной модели свободной границы под специфику архитектуры графических ускорителей позволяет создать эффективную программную реализацию, ведь для вычисления компенсационного потока F_w (2.26) используются лишь локальные значения в каждой ячейке, и нет необходимости учитывать множество возможных вариантов геометрии усеченных (по поверхности твердых включений) сеточных ячеек, которая приводила бы к значительным ветвлениям в коде, значительно снижающих производительность GPU, что шло бы вразрез со свойством вычислительного

“примитивизма” методов, крайне желательного для современных ускорителей. Данная адаптация модели свободной границы не приводит к появлениям дополнительных зависимостей по данным между ячейками сетки при вычислении решения, поэтому нет необходимости менять структуру предлагаемого в работе параллельного алгоритма (§3.3,3.4).

Описанный в предыдущей главе параллельный алгоритм (§3.3,3.4) был реализован с использованием моделей программирования на общей – CUDA – и распределенной – MPI – памяти. Все вычисления выполняются на ускорителях (GPU), центральный процессор (CPU) используется лишь для передачи данных между GPU. Решатель состоит из нескольких процедур, последовательно вызываемых для поиска решения на каждом временном шаге: *omega* – определение промежуточного ω-слоя в каждой ячейке (2.23); *slope* – вычисление частных производных для вектора состояния; *predictor* – вычисление значений предиктора (2.12); *forward, backward* – решение соответственно верхней и нижней подсистемы из 2.45; *updateiter, updatetime* – обновление вектора состояния при завершении ньютоновской итерации и переходе на новый шаг по времени соответственно. Каждая из этих процедур написана в виде функции, исполняемой полностью на GPU, но вызываемой на CPU; в терминах CUDA такие функции называются *ядрами* (*kernels*). Декартова сетка инициализируется на CPU и затем полностью копируется на GPU, общая схема программной реализации представлена на Рис. 4.1.

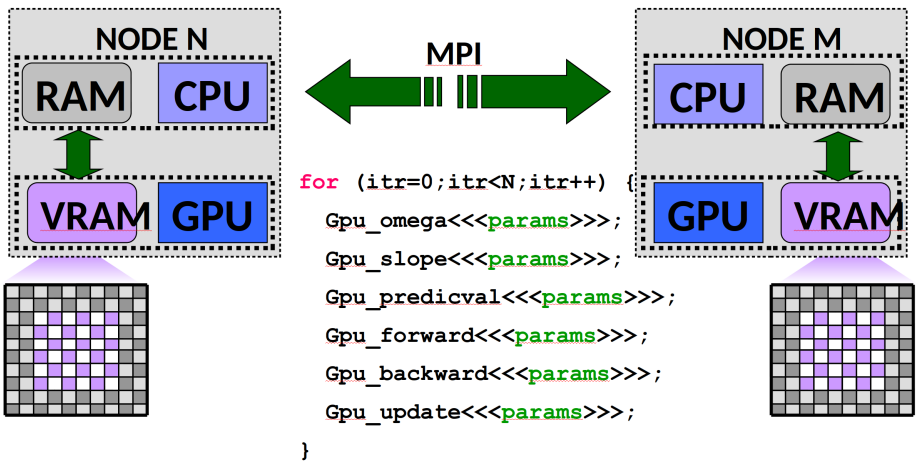


Рисунок 4.1 — Схема реализации программного комплекса

Сеточное разбиение изначально “нарезается” плоскостями на прямоугольные блоки, расположенные “стык-в-стык” (т.е. они также имеют топологию декартовой сетки), затем каждому GPU (MPI процессу) назначается по одному такому блоку. В соответстствии с параллельным алгоритмом каждое из указанных выше ядер запускается на отдельных подобластях каждого блока – внутренней его части и 6 “плитах”, состоящих из двойного слоя граничных ячеек, Рис. 4.2. Для повышения эффективности работы с памятью (минимизация разреженных обращений к ней из тредов внутри варпа) “black” и “white” ячейки хранятся в 2-х раздельных массивах, в них для поиска соседей используется индексная арифметика, позволяющая в линейно упорядоченном наборе ячеек каждого вида однозначно определять среди них смежные по грани.

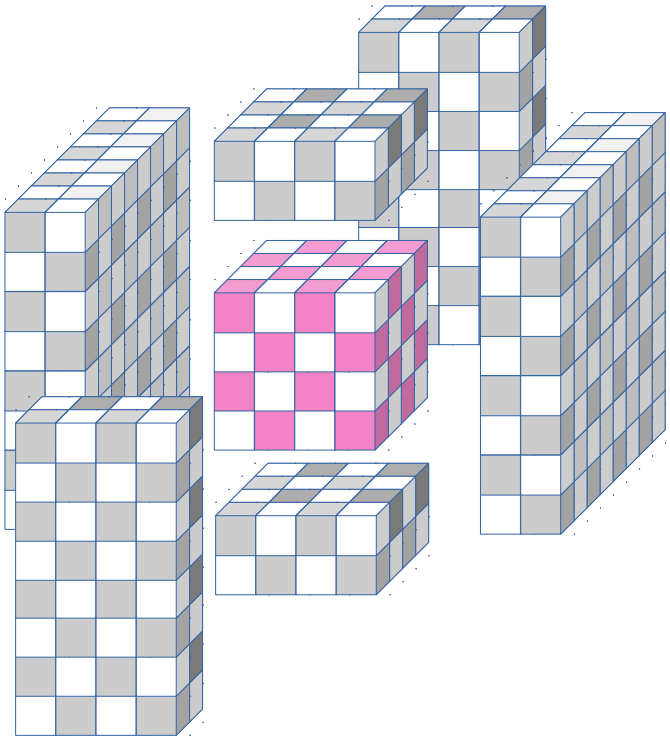


Рисунок 4.2 — Схема разделения подобласти в GPU на внутреннюю часть и 6 граничных “плит”.

Ключевой особенностью реализации является использование полностью асинхронной схемы с многоступенчатой передачей данных между GPU и одновременного выполнения вычислений в них [25]. Для этого в CUDA используется механизм очередей-потоков (streams), позволяющий совмещать во времени выполнение ядер (*kernels*) на GPU и обмен данными с CPU по шине

Pci-express. Для каждой из граничных “плит” подобласти определена отдельная очередь-поток, в которой упорядочиваются операции с GPU путем задания последовательности из следующих действий: копирования данных между CPU и GPU (с указанием направления), выполнение ядер. Для внутренней части подобласти также организуется своя очередь-поток, которая синхронизуется с другими очередями для граничных “плит” с целью задания обхода по внутренним/граничным ячейкам в нужном (для блока каждого типа) порядке. Фактически, система таких очередей (*streams*) представляет собой граф информационных зависимостей, который затем “раскладывается” runtime-системой CUDA на 3 вида ресурсов: 1) вычислительные ядра GPU, 2) канал передачи данных (pci-e) GPU $\rightarrow$ CPU и 3) канал передачи данных (pci-e) CPU $\rightarrow$ GPU. Действия в каждой очереди являются независимыми от действий в других очередях, если явно не задана соответствующая синхронизация, что в итоге и позволяет сократить простои в вычислениях на GPU за счет совмещения передачи данных для одной (из 6-ти) “плит” с вычислениями как в другой, так и во внутренней части. Еще одним механизмом, позволяющим выполнять такие совмещения, является использование т.н. *pinned* или *page-locked* RAM памяти CPU в качестве буферов для граничных ячеек. В отличие от “обычной” (*pageable*) *pinned* память имеет жестко зафиксированные адреса страниц в физической RAM памяти, к которым может напрямую обращаться DMA-контроллер GPU, позволяя тем самым вести обмен данными по Pci-e между памятью GPU и CPU без использования ресурсов последнего.

Несмотря на появление технологии GPU Direct RDMA [71], позволяющей напрямую передавать данные между GPU, расположенными на разных узлах, программная модель CUDA требует всегда явно инициировать такой обмен на CPU, используя, как правило, вызовы библиотеки MPI. Использование семейства технологий GPU Direct [72] накладывает ограничение как на аппаратное, так и программное обеспечение кластеров, тем самым сужая множество вычислительных систем, на которых программа, в которой они задействуются, может работать. Данные технологии нацелены на улучшение скоростных харак-

теристик обмена данными между GPU, позволяя совместно задействовать один общий буфер в памяти библиотекой CUDA и программным стеком OFED [73] для сети Infiniband, используемом в реализациях MPI, минимизируя таким образом число операций копирования при пересылке данных от одного ускорителя к другому, либо позволяя напрямую пересылать данные из памяти одного GPU в память другого, даже в случае, если ускорители расположены на разных узлах. В конечном итоге это позволяет уменьшить время коммуникаций между GPU и соответственно уменьшить отношение времени передачи данных ко времени счета, улучшив тем самым масштабируемость программной реализации. В данной работе GPU Direct не используются, но несмотря на это, программная реализация показала весьма хорошую масштабируемость (глава 6), которую в будущем можно будет еще улучшить, задействовав эти технологии.

Как было сказано, для обмена данными между GPU на кластерных системах приходится использовать функции передачи данных на CPU, как правило, – вызовы MPI. Чтобы совместить счет на GPU и обмен сообщениями по сети, в разработанном коде используются неблокируемые пересылки – *MPI_Isend* и *MPI_Irecv*. Для координации библиотек CUDA и MPI при многоступенчатой пересылке данных между GPU используются блокирующие функции *cudaStreamSynchronize* и *MPI_wait* таким образом, что в большинстве случаев они не приводят к простоям в GPU, т.е. к моменту их вызова соответствующие операции копирования данных по шине Pci-e и сети Infiniband, как правило, уже завершены.

Таким образом, использование описанной выше полностью асинхронной схемы с многоступенчатой передачей данных между GPU и выполнением вычислений в них позволяет во многих случаях *полностью* совместить во времени счет и обмен данными (Рис. 4.3), обеспечивая тем самым практически линейную масштабируемость солвера (глава 6); scatter*/gather* ядра обеспечивают разборку/сборку массива граничных ячеек, распределенных в линейном адресном пространстве памяти GPU, поскольку непрерывные блоки данных передаются по шине pci-e значительно эффективнее по сравнению с множеством копиро-

ваний отдельных сеточных ячеек, распределенных с большим шагом в памяти GPU. Особо стоит отметить, что данный результат достигнут для, строго говоря, *неявной* схемы 2.18 с *точным* соблюдением итеративного метода LU-SGS на *всей* расчетной области с весьма сложной структурой зависимостей по данным. Напротив, для реализаций более простых методов на вычислительных системах с графическими ускорителями, в частности, метода Якоби, часто используются более примитивные (и существенно более простые в реализации и отладке) схемы организации вычислений и обмена данными без их совмещения по времени, Рис. 4.4, которые в итоге значительно ограничивают масштабируемость решателя на большое число GPU.

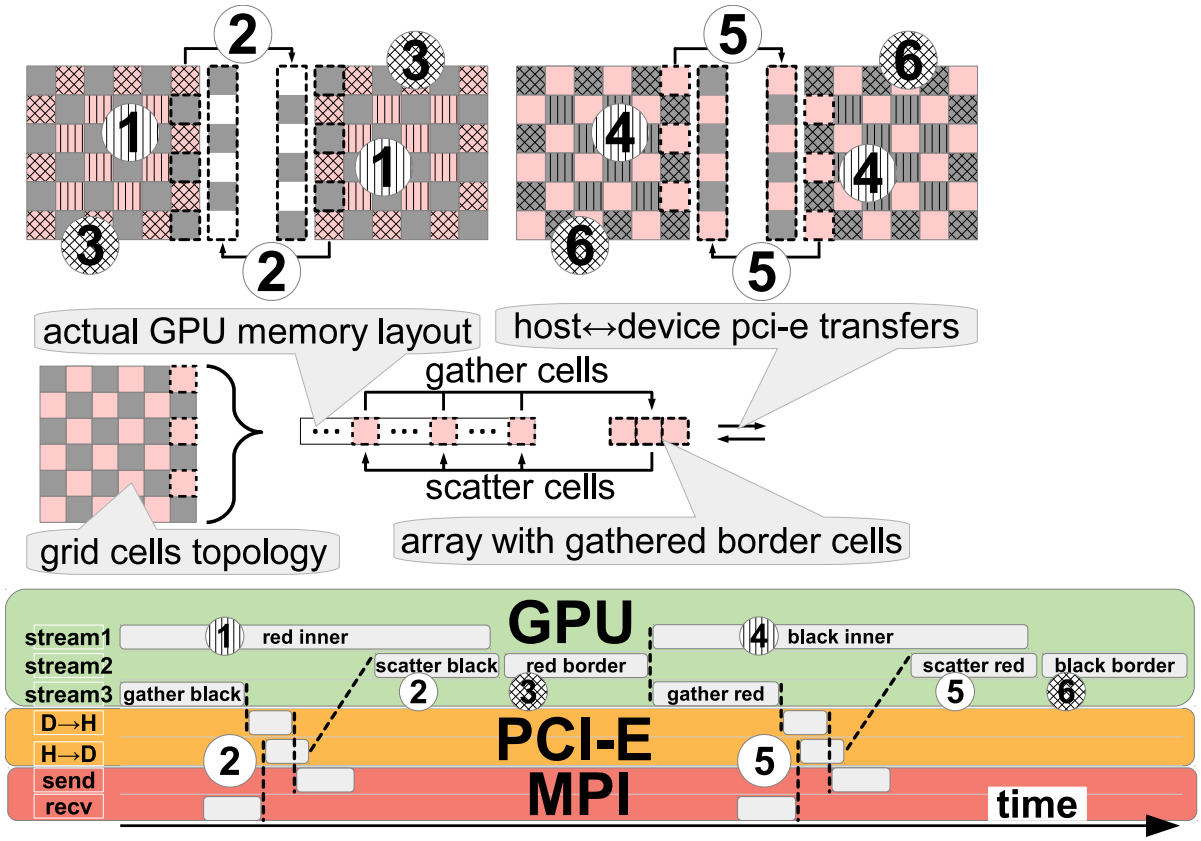


Рисунок 4.3 — Порядок счета и обмена ячеек между блоками (вверху); упаковка/распаковка ячеек в/из массива при копировании по pci-e (в центре); схема счета в GPU и обмена данными между ними с совмещением по времени (внизу).

Как будет показано в параграфе 6.2, даже в полностью отлаженной и оптимизированной программной реализации обнаружились проблемы с масштабируемостью, устранение которых потребовало весьма специфических настроек

окружения CUDA и MPI. Данный факт является дополнительным подтверждением того, что для высокоэффективной масштабируемой реализации решателей необходимо выполнение *всех* нижеследующих условий:

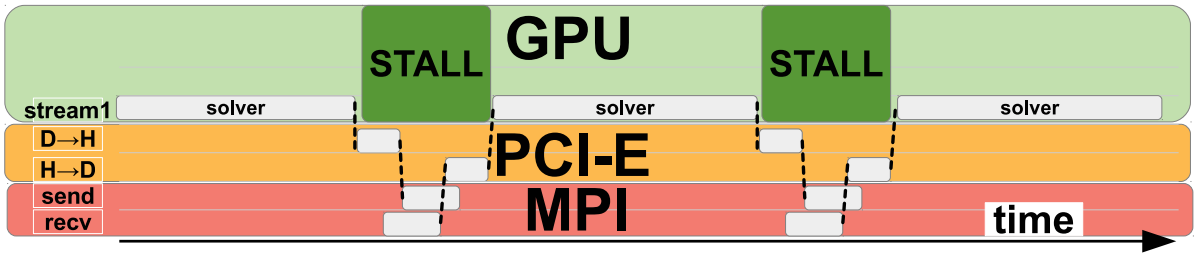


Рисунок 4.4 — Упрощенная схема организации вычислений на GPU без совмещения по времени с обменом данными между GPU, приводящая к простоям в них.

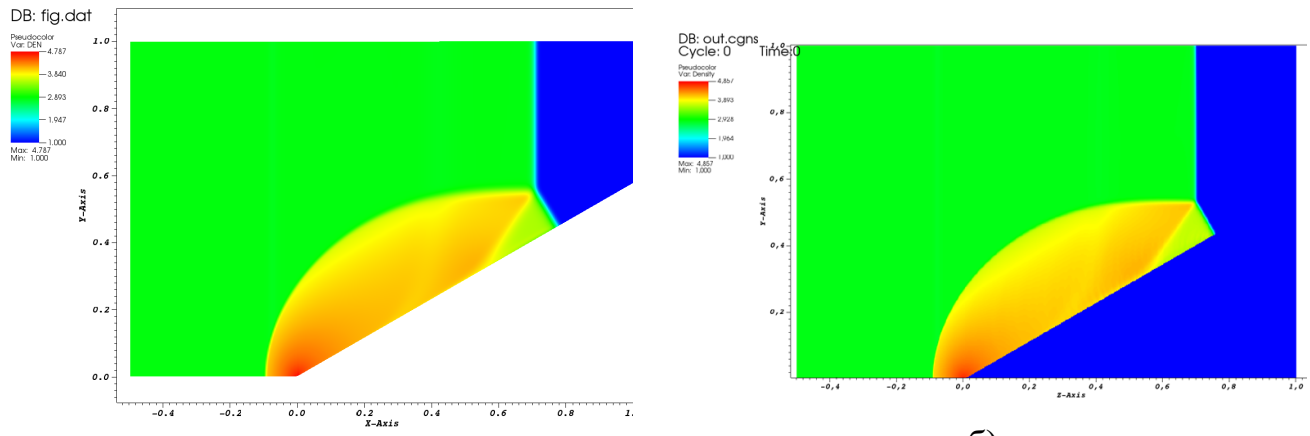
– выбор/адаптация численных моделей и методов с учетом специфики и архитектурных особенностей целевых вычислительных систем, на которых будет выполняться решение задач;
– разработка таких параллельных алгоритмов для численных методов, которые не содержат в себе неустранимые последовательные части и *позволяют* совмещать во времени вычисления и обмен данными между вычислителями;
– программная реализация с использованием достаточно низкоуровневых средств программирования, таких как CUDA C/OpenCL, MPI, которые предоставляют наибольший контроль над ресурсами вычислительной системы и позволяют организовывать достаточно сложные схемы совмещения вычислений и обмена данными в соответствии с разработанным параллельным алгоритмом. Использование высокоуровневых средств, таких как OpenACC [74], библиотека cuBLAS [75] во многих случаях не позволяют реализовать данные схемы;
– использование средств профилирования и анализа производительности для разработанной программной реализации с целью выявления и устранения проблем, приводящих к снижению скорости работы программы, либо подтверждения эффективности ее работы.

Глава 5. Результаты вычислительных экспериментов

В данной главе приводятся результаты численного моделирования, полученные с помощью разработанного программного комплекса, приводятся сравнения с решениями, полученными другими программными реализациями в стандартной постановке (на связной сетке без метода свободной границы) [1].

5.1 Дифракция ударной волны на клине

Для проверки корректности работы программной реализации было проведено моделирование движения ударной волны в области с клином с углом раствора $\theta = 30^\circ$. В начальный момент времени в области задавалась распределение с ударной волной с $p = 1, \rho = 1, \vec{v} = 0$ до фронта и условием Ранкина-Гюгонио после фронта. Расчет проводился с помощью разработанного программного комплекса методом свободной границы на GPU Nvidia Tesla K20c в трехмерной постановке с сеточным разрешением $600 \times 400 \times 6$; его результаты сравнивались (Рис. 5.1) с расчетом в стандартной постановке на связной сетке разрешением 300×200, полученном на центральном процессоре в однопоточном режиме со “стандартным” (слева–направо, снизу–вверх) обходом ячеек сетки. Как видно, оба решения дают очень близкие распределения давления во всей расчетной области, тем самым подтверждая корректность работы программной реализации на графическом ускорителе.



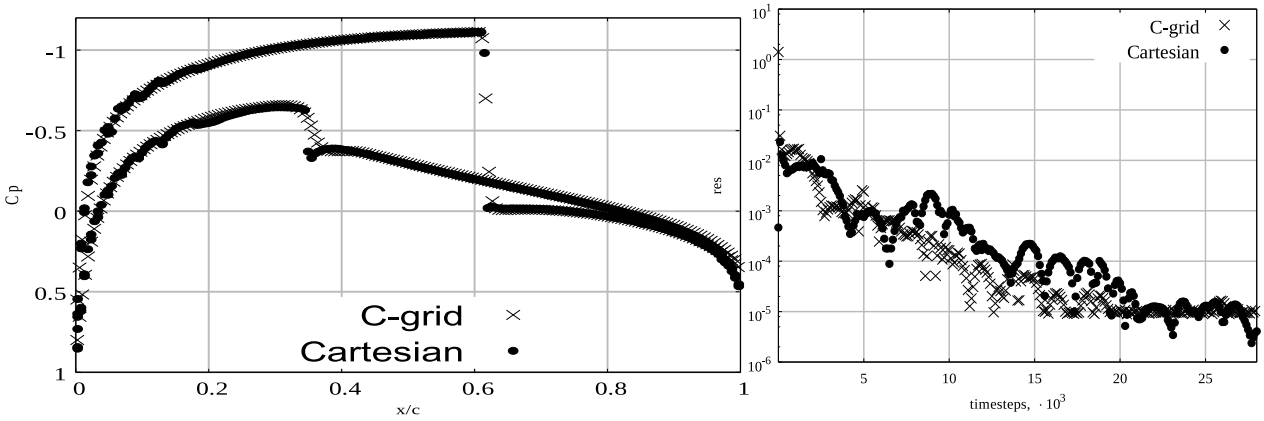
а) б)

Рисунок 5.1 — Распределение давления, $M = 2.12, t = 0.32c$; а) несвязная сетка 600×400, б) связная сетка 300×200.

5.2 Обтекание профиля NACA0012

Детальное тестирование предлагаемого метода было проведено на задаче обтекания крыловидного профиля NACA 0012. Угол атаки составлял $\alpha = 1.25°$, число Маха набегающего потока – $M = 0.8$. Использовались сетки 2 типов: связанная с поверхностью профиля структурированная C-сетка с разрешением 400 ячеек вдоль границы профиля и общим разрешением 1000×150 и несвязная декартова сетка с разрешением внутри описанного вокруг профиля прямоугольника 200×24 и общим разрешением 650×324. Расчеты на C-сетке проводились в последовательном однопоточном режиме на центральном процессоре, решения на декартовой сетке были получены с использованием графического ускорителя (Nvidia Tesla K20).

Расчеты на этих сетках проводились по гибридной явно-неявной схеме второго порядка точности с числом Куранта $C = 10$. Распределение коэффициента давления Cp и скорости сходимости к стационарному решению приведены на Рис. 5.2. Как видно, результаты на разных сетках, включая и скорость сходимости, хорошо согласуются. Характерной особенностью течения является небольшой локальный минимум в распределении давления, возникающий сразу за ударной волной на наветренной стороне профиля. Видно, что эта особен-



а) б)

Рисунок 5.2 — Распределение Cp (а) и скорость сходимости (б) на согласованной (C-grid) и несвязной (Cartesian) сетках, NACA0012, $M = 0.8$, $\alpha = 1.25°$, число Куранта 10

ность лучше воспроизводится на декартовой сетке, чем на согласованной, где она фактически отсутствует и начинает проявляться при увеличении сеточного разрешения. Это связано со свойством ортогональности, естественно присущим декартовой сетке, которое лучше обеспечивает выполнение повышенной точности схемы.

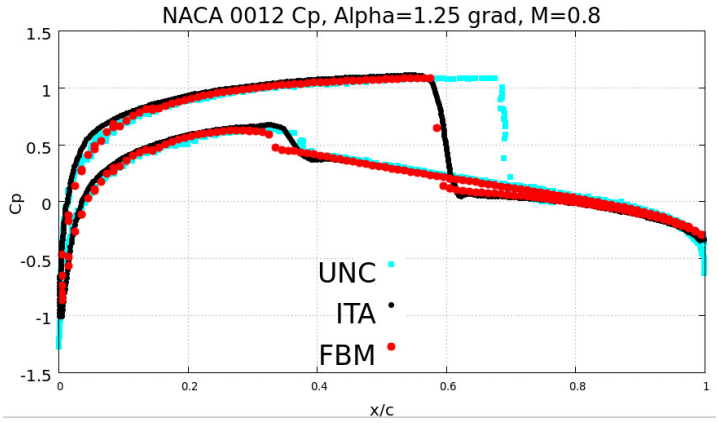
Расчетные значения коэффициентов подъемной силы и сопротивления, Cl и Cd, полученных на согласованной и несвязной декартовой сетках, приведены в Табл. 1. Разница значений этих коэффициентов составляет менее 1%.

Таблица 1 — Аэродинамические коэффициенты: NACA0012, $M = 0.8$, $\alpha = 1.25°$, число Куранта 10

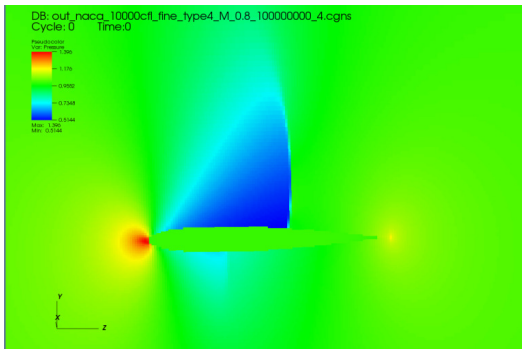
Коэффициент	Декартова сетка	C-сетка
Cl:	0.3012	0.3036
Cd:	0.02184	0.02199

На Рис. 5.3 показано сравнение с решениями, полученными другими кодами (UNC и ITA [76]) на C-сетках и распределение давления, полученное рассматриваемым в данной работе программным комплексом (FBM). Решатель UNC, в основе которого используется TVD схема типа Хартена [77], хоть и позволяет более точно захватывать скачок давления на подветренной стороне

профиля, дает в итоге бóльшую ошибку (характерную для него и на других задачах с трансзвуковыми течениями) в сравнении с ITA и FBM. ITA код, в котором использовалась схема Джеймсона [78], дает, также как и программный комплекс из данной работы (FBM), более точное решение.

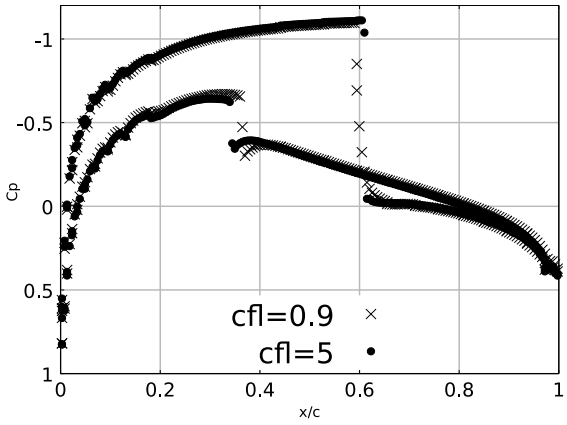


а)

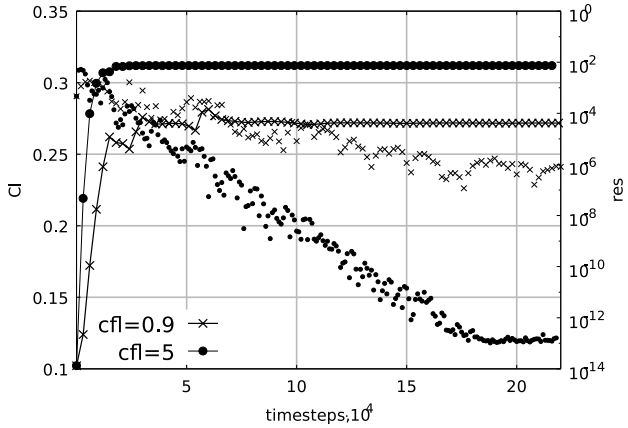


б)

Рисунок 5.3 — Распределение Cp (а), полученное различными решателями, и распределение давления (б), полученное с помощью метода свободной границы (FBM), NACA0012, $M = 0.8$, $\alpha = 1.25°$, число Куранта 10^4



а)



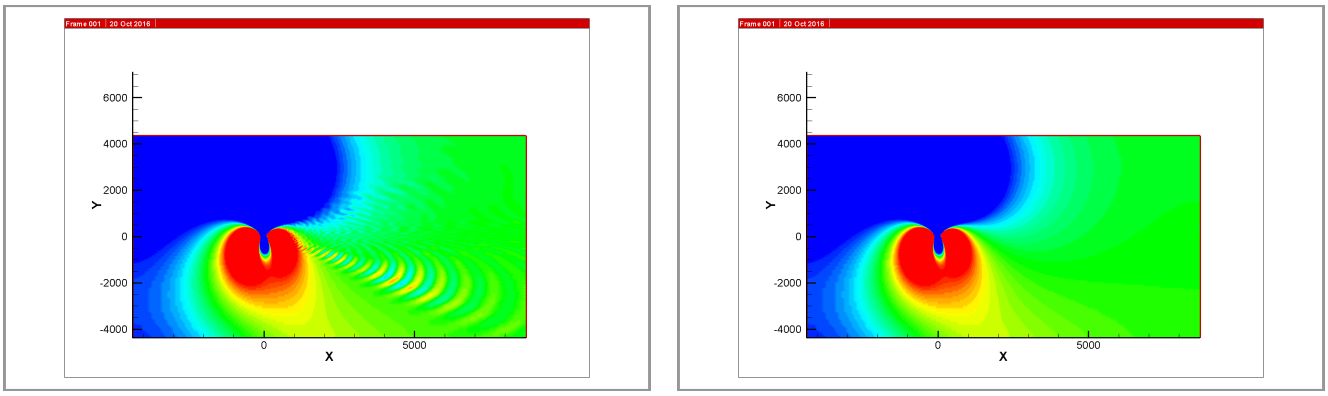
б)

Рисунок 5.4 — Распределение Cp (а), зависимость Cl от времени и скорость сходимости (б) для чисел Куранта 0.9 и 5 на несвязной сетке, NACA0012, $M = 0.8$, $\alpha = 1.25°$

Были проведены также расчеты на несвязной декартовой сетке с разными временными шагами, которые обнаружили известное свойство неединственности решений уравнений Эйлера. В этой серии расчетов применялась чисто неявная схема ($\omega = 0$), чтобы исключить зависимость стационарного решения

от шага по времени. На Рис. 5.4 а) приведены два распределения коэффициента давления Cp, полученные при числах Куранта 0.9 и 5. Сетка и метод решения абсолютно одинаковые. Видно, что это два разных решения, отличающихся положением и профилем ударных волн (в частности, на подветренной стороне ударная волна имеет более резкий профиль для решения с числом Куранта 5). На Рис. 5.4 б) приведены графики сходимости невязки и коэффициента подъемной силы. Как следует из этих данных, оба решения выходят на стационарный режим. Различие в значениях коэффициента подъемной силы при этом оказывается порядка 15%. Неединственность решения, обнаруженная в этих расчетах, наблюдалась и исследовалась также в работах [20-22] по трансзвуковому обтеканию крыловидных профилей.

При увеличении числа Куранта до значений порядка 10^5 решение в указанной ранее конфигурации переставало сходиться, Рис. 5.5 а). Примечательно, что понижение порядка аппроксимации по пространству со второго до первого или запуск решателя на CPU с последовательным “стандартным” (слева – направо, снизу – верх) обходом ячеек вновь позволили получать сходящееся решение. Исходя из вышеизложенного, стало понятно, что в данном случае на сходимость решения существенно влияет порядок обхода ячеек. В самом деле, “шахматный” порядок обхода ячеек, использующийся при расчетах на GPU, более напоминает явный метод Якоби, имеющий жесткие ограничения на выбор шага по времени, по сравнению с упомянутым “стандартным” обходом, где зависимость по данным между соседними ячейками носит более сложный характер, т.е. более выражена “неявность” сетода LU-SGS, что в итоге и позволяет в последнем случае получать сходящееся решение. Поэтому естественным подходом видится увеличение “неявности” для “шахматного” обхода расчетной области. Это было сделано в виде сохраненной глобальной структуры “шахматного” обхода, но клетка каждого цвета представляет собой уже макроячейку, состоящую из 2×2 ячеек, Рис. 5.6 б)



а) б)

Рисунок 5.5 — Распределение давления вокруг профиля NACA0012 для решений, получаемых с шахматным обходом с клеткой размером а) 1×1 ячейку и б) 2×2 ячейки

а) б)

Рисунок 5.6 — Схема разбиения расчетной сетки с макроячейками (клетками) размером а) 1×1 и б) 2×2 ячеек

Данная модификация “шахматного” обхода позволила получить сходящееся решение, Рис. 5.5 б). На Рис. 5.7 представлен график сходимости для двух обходов, который и подтверждает сходимость для нового обхода.

Значения компонент силы, действующей на профиль, в зависимости от номера итерации приведено на Рис. 5.8. Как видно, на разбиении с макроячейками 2×2 сила устанавливается значительно быстрее, без дальнейших колебаний.

Дополнительно были проведены расчеты и для “шахматных“ разбиений с клетками из 4×4, 16×16 сеточных ячеек, Рис. 5.9. Переход от 1×1 к 4×4 клеткам сокращает число необходимых итераций для получения стационарного решения в 1.5 раза. Дальнейший переход к клеткам 16×16 позволяет еще незначительно сократить нужное число итераций. Использование же последовательного “стандартного” (слева – направо, снизу – вверх) обхода по сравнению с

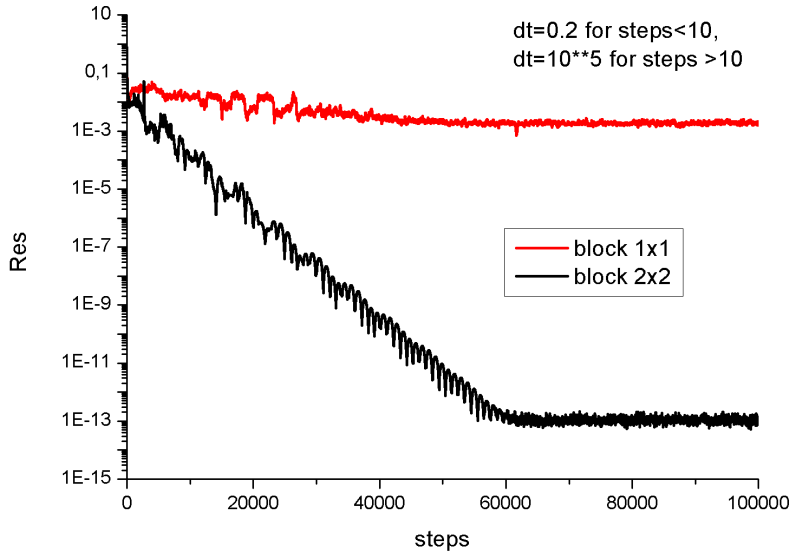
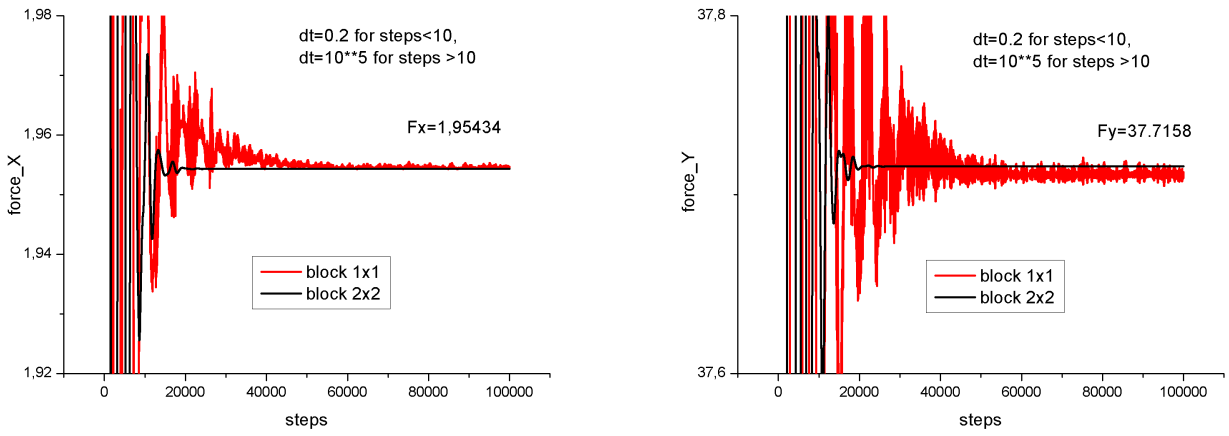


Рисунок 5.7 — Сходимость решения на разных шахматных разбиениях, “клетки” которых составляют ячейки сетки размерами 1×1 и 2×2



а) б)

Рисунок 5.8 — История изменения а) горизонтальной и б) вертикальной составляющей силы, действующей на профиль NACA0012

“шахматным” разбиением из макроклеток 16×16 уже практически не изменяет скорость сходимости решения.

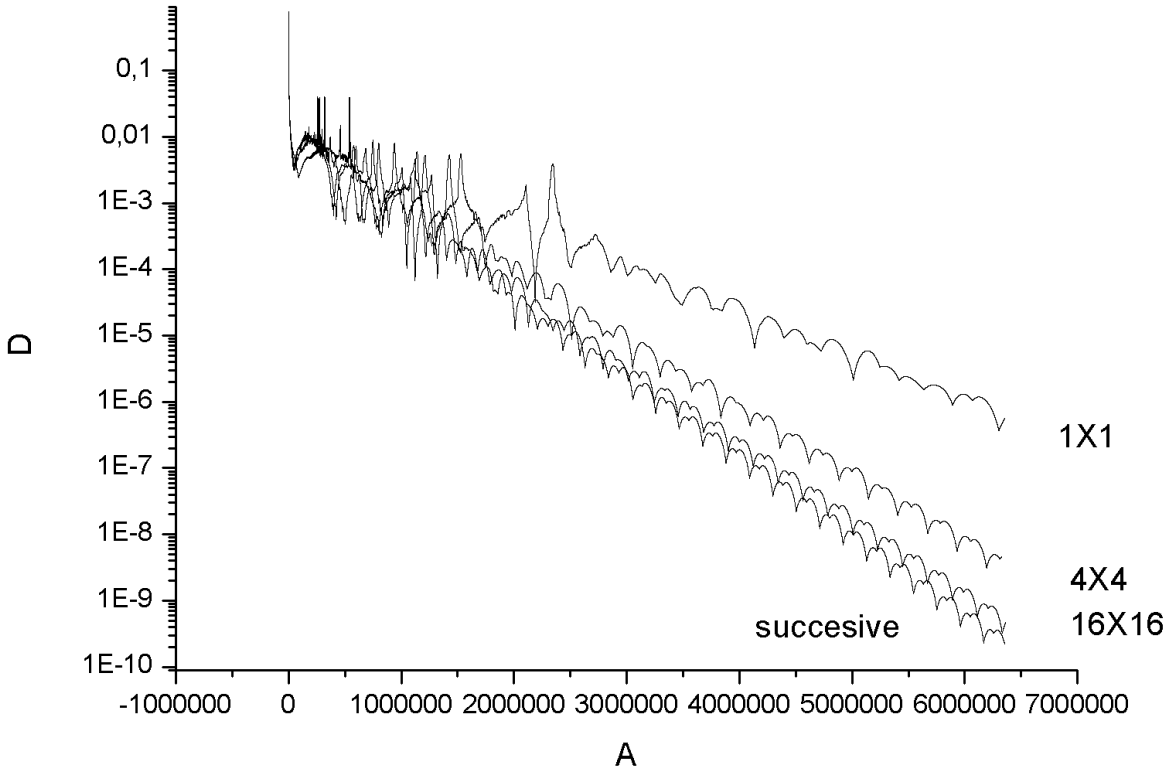


Рисунок 5.9 — Скорость сходимости решения с последовательным “стандартным” обходом и “’шахматным” обходом с различными размерами клеток

5.3 Сверхзвуковое обтекание клина

Корректность работы программного комплекса проверялась также на задачах взаимодействия сверхзвукового потока газа (число Маха $M = 3$) с клином (угол раствора 10°) на режимах с образованием ударной волны и волны разряжения. Использовалась равномерная несогласованная декартовая сетка с разрешением 1200×480 ячеек. Число Куранта в расчетах составляло $C = 4$. Расчеты проводились с использованием 9 GPU Nvidia Tesla C2050 на 3 узлах суперкомпьютера K-100. Расчетные значения угла ударной волны β и угла γ, внутри которого формируется веер волны разрежения, приведены в Табл. 2. Там же для сравнения указаны соответствующие аналитические значения. Как видно,

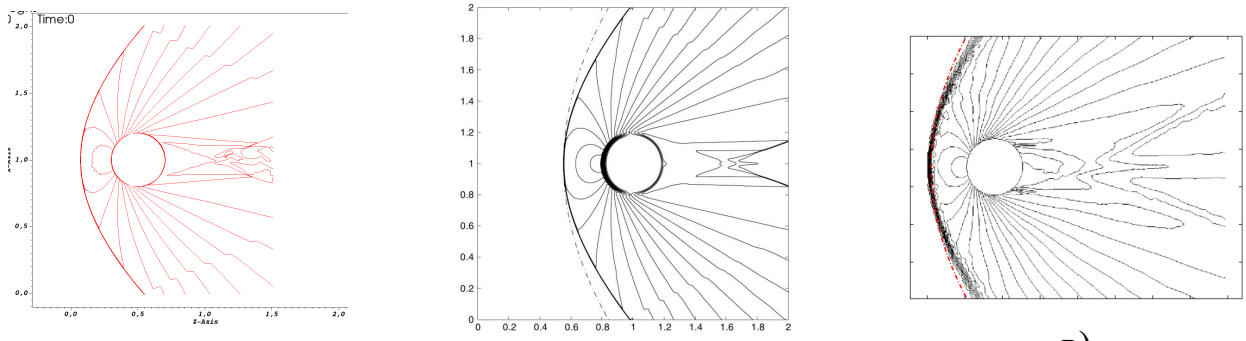
результаты для численного решения оказались очень близки к соответствующим аналитическим.

Таблица 2 — Сверхзвуковое обтекания клина: угол раствора 10°, $M = 3$

Численное решение β°	Аналитическое решение β°	Численное решение γ°	Аналитическое решение γ°
17.4	17.383	13.2	13.24

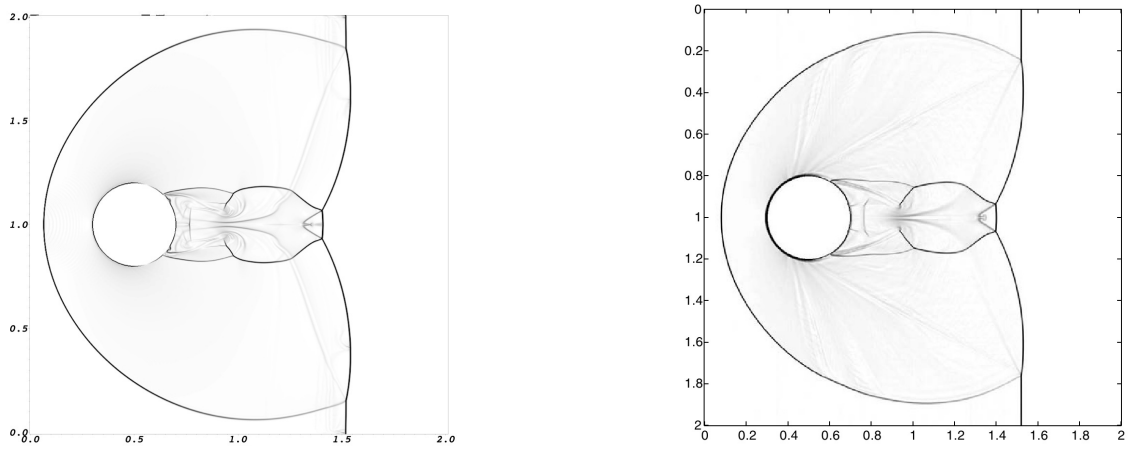
5.4 Течение вокруг системы цилиндров

Для проверки метода был выполнен расчет нестационарной задачи взаимодействия ударной волны с одним цилиндром и группой цилиндров из работы [60]. Расчет проводился на декартовой равномерной сетке разрешением 1024×1024 ячеек с использованием 32 GPU (Nvidia Tesla C2050) на суперкомпьютере K-100. Решения, полученные с помощью метода свободной границы, Ansys Fluent [79] (неструктурированная C-сетка) и одного из методов штрафных функций [60] приведены на Рис. 5.10. Также расчетные данные представлены на Рис. 5.11, 5.12, где изображены мгновенные численные шлирен-визуализации потоков. Решения оказались очень близки, но при этом метод свободной границы не дает нефизичных возмущений вблизи поверхностей цилиндров. Это связано с тем, что в методе штрафных функций [60] нет подсеточного разрешения геометрии; геометрия там представлена более грубо, с точностью до ячейки сеточного разбиения.



а) б) в)

Рисунок 5.10 — Изолинии плотности, $M = 2$. Метод свободной границы (а), метод штрафных функций (б); в Ansys Fluent (в) использована связная неструктурированная сетка.

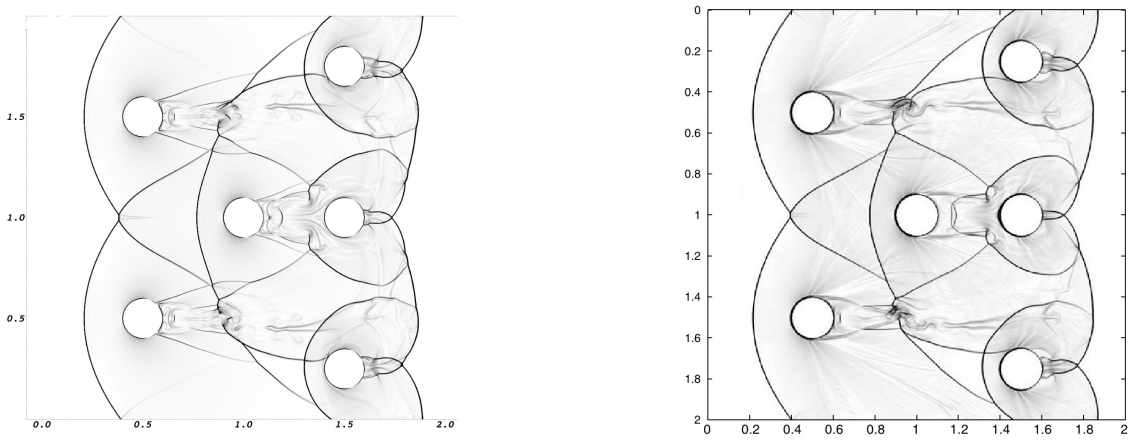


а) б)

Рисунок 5.11 — Численная визуализация течения около цилиндра, $M = 3$, $t = 0.4$; а) настоящий метод свободной границы, б) метод штрафных функций.

5.5 Моделирование макета тягового устройства

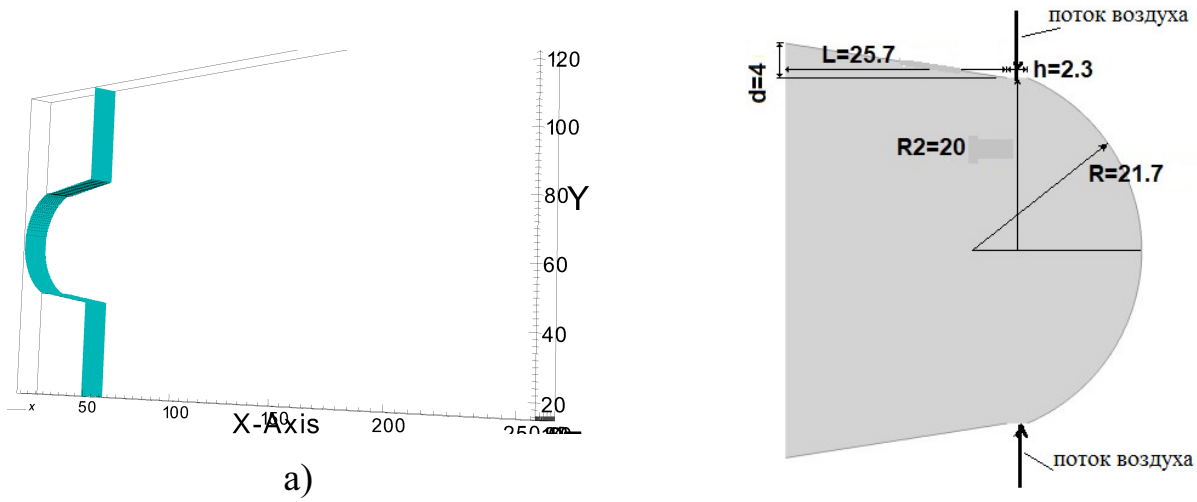
Было проведено моделирование макета двухщелевого тягового устройства в трехмерной пространственной постановке, геометрия которого представляет собой цилиндрическую поверхность (Рис. 5.13 а). Во внутреннюю его область через щели ширины $h = 2.3$ мм в начальный момент времени начинает поступать постоянный звуковой поток воздуха F с давлением и температурой торможения соответственно 21.8 атм и $308.5°K$ (Рис. 5.13 б). Расчет проводился на несвязной декартовой сетке разрешением $540 \times 400 \times 6$. Поскольку верхняя и нижняя щели пересекаются с внутренними ячейками сетки, к ком-



а) б)

Рисунок 5.12 — Численная визуализация течения около системы цилиндров, $= 3$, $t = 0.5$; а) настоящий метод свободной границы, б) метод штрафных функций.

пенсационному потоку F_w для них добавлялся поток F. Предполагалось, что изначально в устройстве и в пространстве вне его (во всей расчетной области) находится неподвижный воздух при нормальных условиях с давлением и температурой соответственно 1 атм и $300°K$. Расчет проводился на GPU Nvidia Tesla K20c с числом Куранта $C = 0.2$. На Рис.5.14 представлено поле давления, а на Рис.5.15,5.16 – линии тока, на которых хорошо видны вихревые структуры, образующиеся вблизи тяговой стенки, а течение подобно течению в сопловом устройстве с центральным телом.



а) б)

Рисунок 5.13 — а) Геометрия макета двухщелевого тягового устройства; б) Схема устройства с размерами в мм.

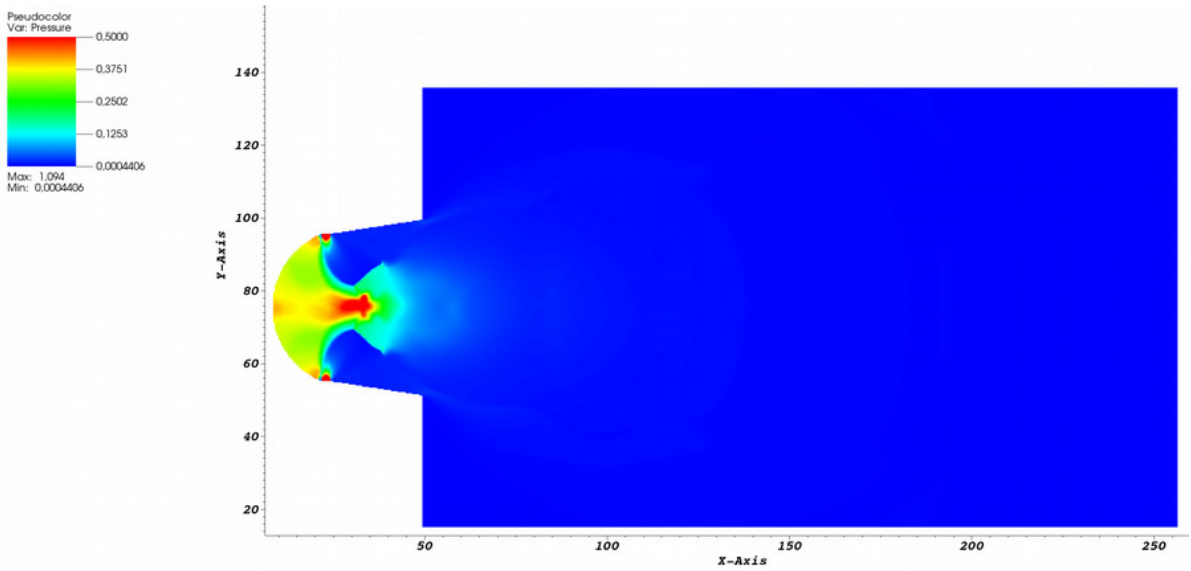


Рисунок 5.14 — Поле давления при $t = 3750$ с.

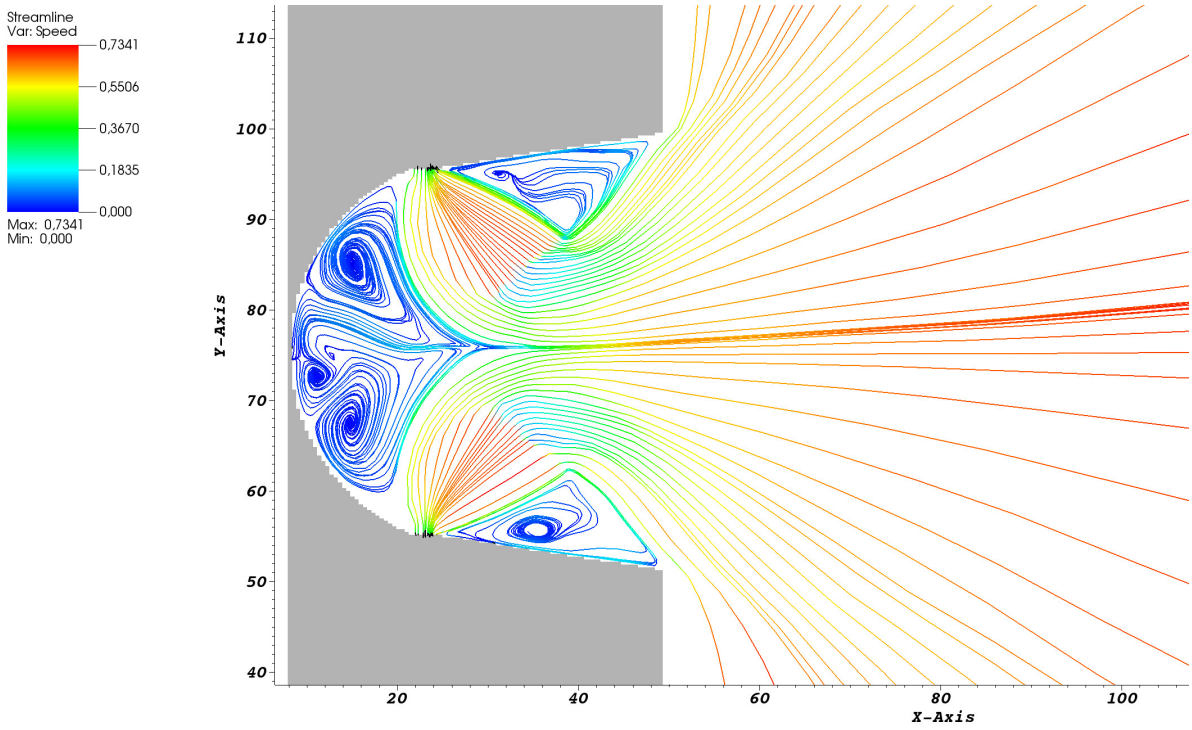


Рисунок 5.15 — Линии тока при $t = 3750$ с.

5.6 Моделирование течения вокруг профиля DLR F6

Проведено моделирование трехмерного обтекания модели пассажирского самолета DLR F6 [80]. Число Маха набегающего потока составляло M=0.75, угол атаки – $\alpha = 1°$, разрешение расчетной сетки – $408 \times 520 \times 1256$ ячеек.

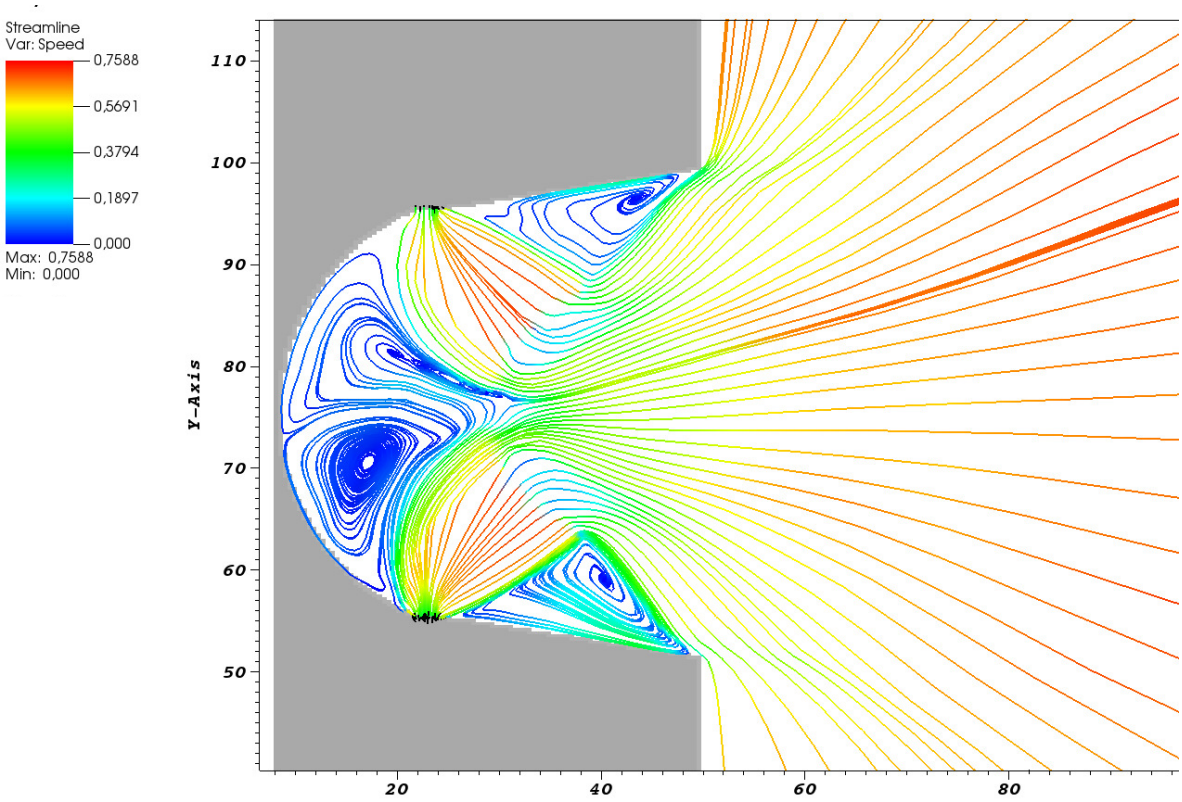


Рисунок 5.16 — Линии тока при $t = 3516$ с.

Сеточное разбиение и пересекаемые/внутренние ячейки для модели DLR F6 изображены на Рис. 5.17.

Расчет проводился с использованием 162 GPU СК “Лобачевский” по гибридной явно-неявной схеме с числом Куранта $C = 4$. Расчет 10^5 шагов по времени в указанной конфигурации занял 3 часа. Результаты расчета представлены на Рис. 5.18, 5.19. Следует отметить, что расчетное значение коэффициента подъемной силы оказалось несколько выше экспериментального ($C_{lexp} = 0.5$). Подобное превышение (порядка $0.5 - 0.6$) наблюдается также и в численных решениях, полученных на согласованных сетках [80] и объясняется недостаточностью модели Эйлера.

Были проведены расчеты также и на сетке с вдвое меньшим разрешением по каждому направлению – $204 \times 260 \times 628$ с числом Маха $M = 0.75$ и углами атаки $\alpha = -1, 0, 1°$. График зависимости коэффициента подъемной силы Cl от угла атаки представлен на Рис. 5.20. Отличие от экспериментальных данных и других численных решений объясняется как недостаточностью модели Эйлера, так и низким разрешением сетки, в крупных ячейках которой происходит доста-

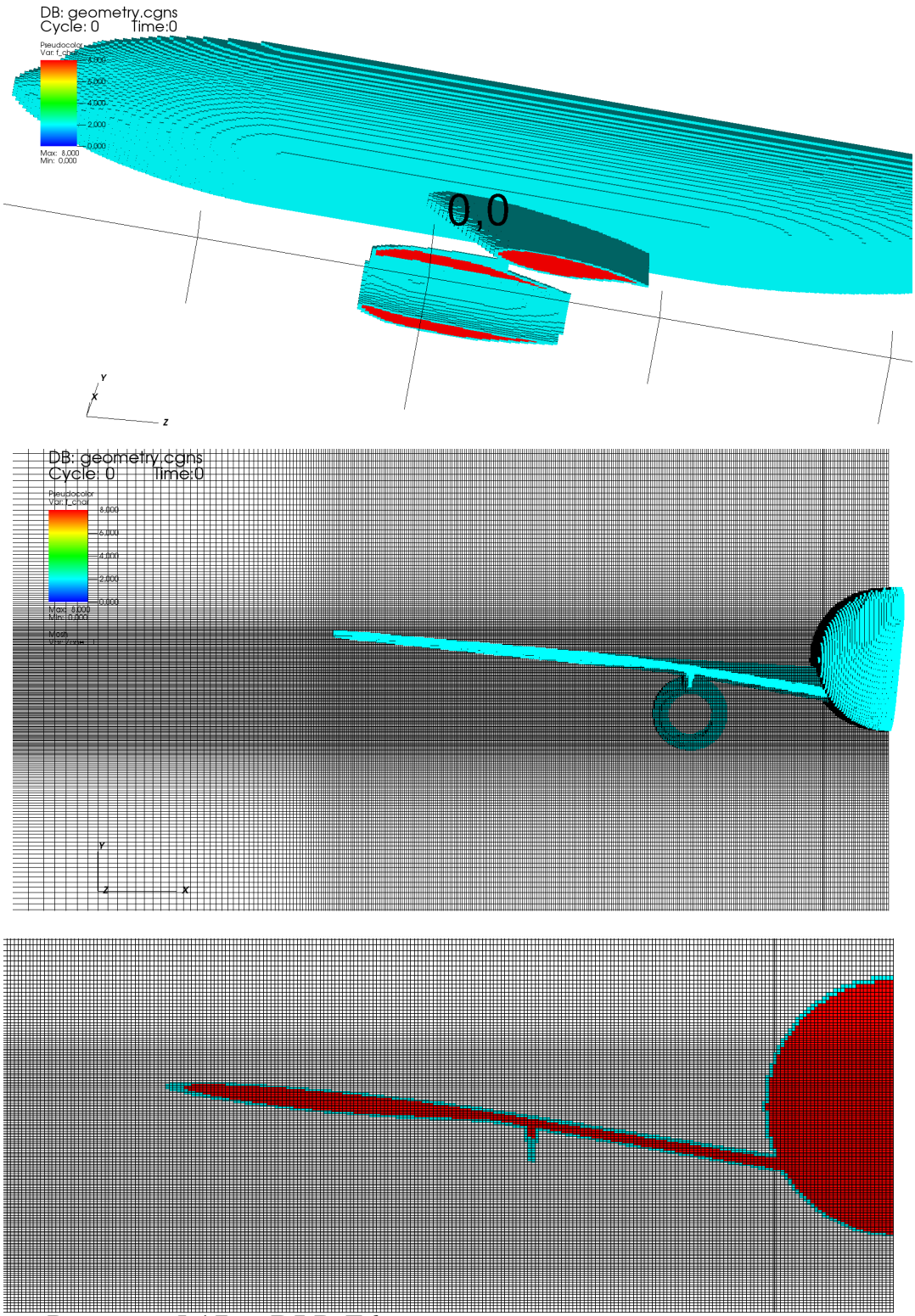
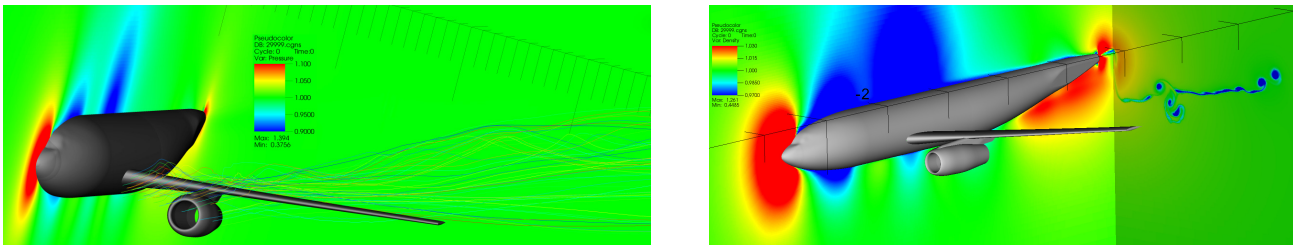


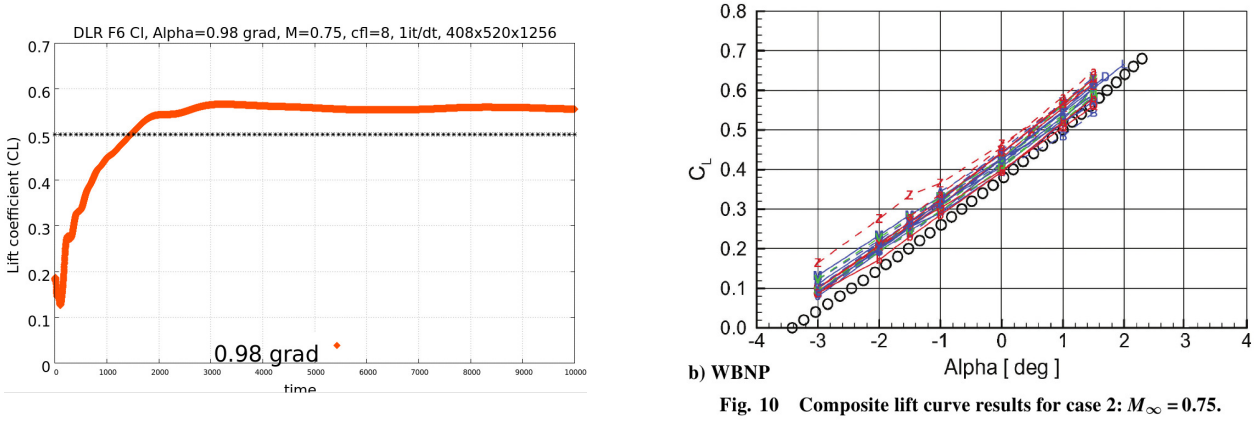
Рисунок 5.17 — DLR F6; декартова сетка с пересекающимися (■) и внутренними ячейками (■)

точно грубое линейное восполнение модели DLR F6. Последнее предположение подтверждается расчетом на более подробной сетке ($408 \times 520 \times 1256$), приведенным ранее: в нем значение $C_l = 0.56$ лучше согласуется с другими решениями по сравнению с $C_l = 0.44$ на грубой сетке ($204 \times 260 \times 628$).



а) б)

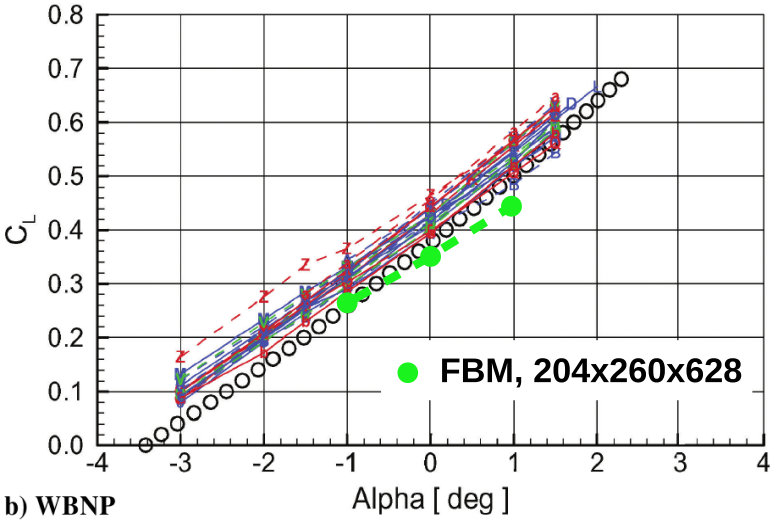
Рисунок 5.18 — Модель DLR F6, $M = 0.75$, $\alpha = 1°$; а) поле давления в плоскости симметрии и линии тока; б) поле плотности в плоскости симметрии и в перпендикулярной к ней (за моделью)



а) б)

Рисунок 5.19 — Модель DLR F6, $M = 0.75$; а) зависимость коэффициента подъемной силы C_l от времени, $\alpha = 1°$; б) экспериментальные данные (обозначены как “○”) и решения, полученные другими программами в [80]

Данный расчет демонстрирует, что для получения более точных решений методом свободной границы необходимо задавать достаточно высокое сеточное разрешение вблизи поверхностей твердых тел, и в случае декартовых сеток единственно доступный способ локального уменьшения размеров ячеек сетки - сгущение по каждой из координатных осей, которое все-таки приводит к избыточному разрешению и на отдалении от твердых включений, повышая тем самым вычислительную сложность задачи.



b) WBNP

Fig. 10 Composite lift curve results for case 2: $M_\infty = 0.75$.

Рисунок 5.20 — Модель DLR F6, $M = 0.75$; зависимость коэффициента подъемной силы C_l от угла атаки, сравнение с экспериментальными данными (обозначены как “$\circ$”) и решениями, полученными другими программами в [80]

.

Глава 6. Исследование производительности и масштабируемости программной реализации

Данный раздел посвящен детальному исследованию эффективности и масштабируемости разработанного программного комплекса [19; 25]. Под масштабируемостью здесь и далее понимается *сильная* масштабируемость (strong scalability), когда размер задачи (сеточное разрешение) фиксируется, а меняется лишь число используемых для расчета вычислительных узлов/ускорителей GPU и замеряется при этом соответствующее время работы программы (или производительность, как обратная величина).

Следует отличать *сильную* масштабируемость от *слабой*. Для оценки последней фиксируется уже не размер задачи, а объем данных, приходящийся на один вычислитель (GPU), т.е. с изменением их числа пропорционально увеличивается и размер всей задачи. Очевидно, что сильная масштабируемость является более строгой оценкой производительности, поскольку для нее, в отличие от слабой масштабируемости, с увеличением числа задействованных в задаче вычислителей обратно пропорционально сокращается объем данных, приходящийся на каждый из них, и всё бóльшую часть времени начинают занимать обмены данными (в граничных ячейках) между вычислителями. Вместе с тем, сильная масштабируемость является и более показательной характеристикой, позволяющей определить фактическую производительность программы с одними и теми же входными данными на разном числе вычислителей; по ней также определяется и максимальное число GPU, на которое программа масштабируется с заданной эффективностью распараллеливания.

Под эффективностью распараллеливания (масштабируемости) на k GPU относительно n GPU понимается величина eff_k,

$$eff_k(n) = \frac{t_k}{t_n} \cdot \frac{k}{n} \times 100\%, \tag{6.1}$$

где t_i – время счета задачи с использованием i ускорителей (GPU).

Тестирование проводилось на следующих суперкомпьютерах: К-100 (ИПМ им М.В. Келдыша РАН) [81], “Лобачевский” (ННГУ им Н.И. Лобачевского) [82], “Ломоносов” (МГУ им М.В. Ломоносова) [83].

6.1 Производительность расчетов на одном GPU, сравнение с CPU

В качестве тестовой была выбрана задача о коническом теле, мгновенно помещенном в однородный сверхзвуковой поток газа с числом Маха $M = 1.6$. Решалась нестационарная задача об образовании ударно-волновой структуры в результате взаимодействия потока с поверхностью тела (Рис. 6.1).

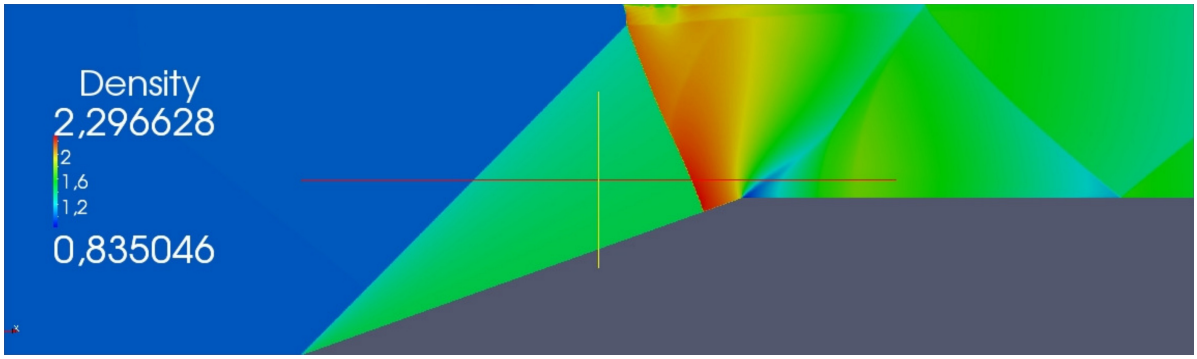


Рисунок 6.1 — Поле плотности, $M = 1.6$, сетка 1800×540, 5000 шагов по времени

Измерения проводились на суперкомпьютере К-100. Использовались 2 версии кода – для Nvidia Tesla C2070 и последовательная версия для процессора Intel Xeon X5670. Компиляция проводилась с помощью CUDA Toolkit 3.2 и Intel C compiler (icc) 11.1. Сравнение времени работы отдельных процедур солвера представлено на Рис. 6.2. На наиболее вычислительно сложные процедуры – forward и backward (решение подсистем 2.45) приходится основное время работы работы программы, причем forward занимает значительно бóльшее время по сравнению с backward, поскольку именно в первой процедуре происходит вычисление потоков на гранях ячеек. В зависимости от процедуры, ускорение на GPU относительно 1 ядра CPU меняется от 8.6 до 36 раз, общая производи-

тельность GPU при этом оказалась больше в 13 раз по сравнению с CPU, при увеличении разрешения сетки до 1800×540 оно повысилось до 15.5.

Следует отметить, что на других задачах, включая и трехмерные, указанное ускорение варьировалось от 13 до 30 раз в зависимости от моделей GPU и CPU, установленных на узлах.

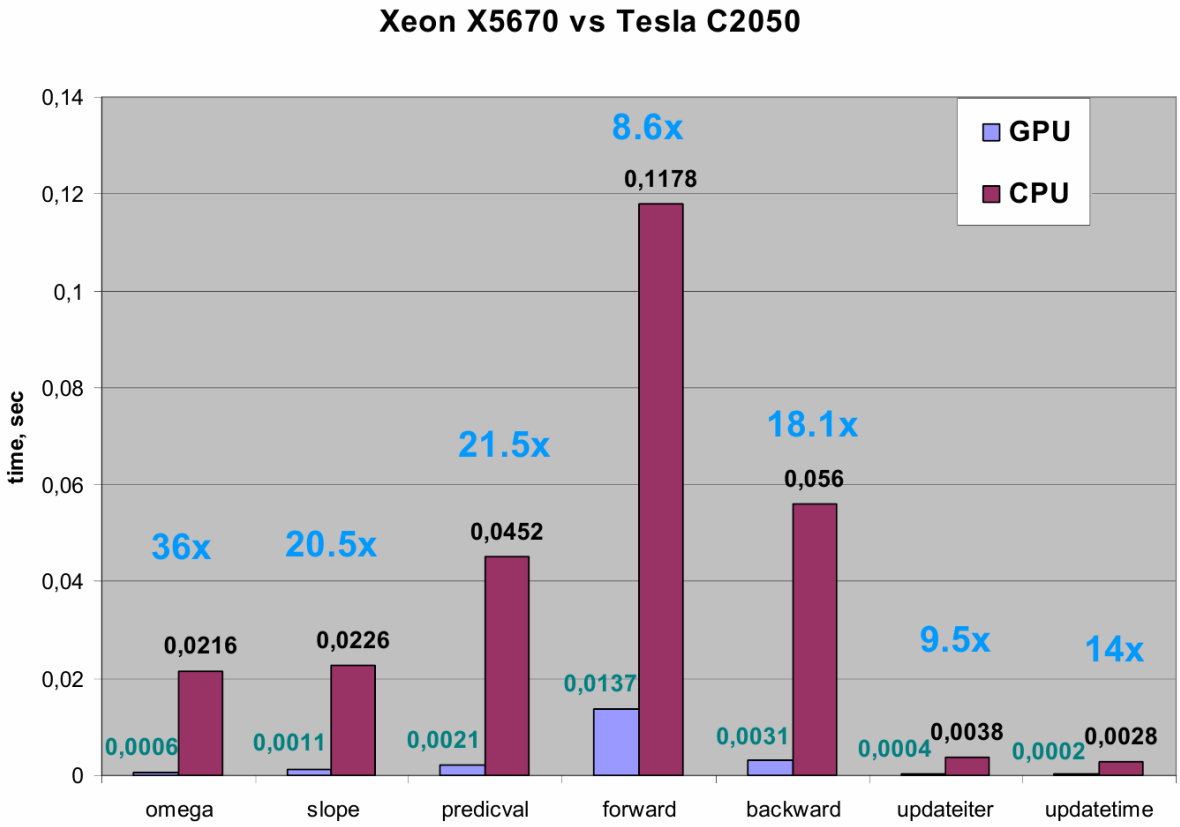


Рисунок 6.2 — Ускорение и время счета (в секундах) на CPU (1 ядро) и GPU, сетка 450×135, один шаг по времени с одной ньютоновской итерацией

Далее исследовалось изменение времени счета на одном GPU при использовании сеток разных разрешений. Базовое разрешение – $x = 450 \times 135$ (≈60 тыс. ячеек) заменялось на другие, которые содержали в 4^n ($n = -2, ...2, 3$) раз больше/меньше ячеек. Результаты измерений представлены на Рис. 6.3. Видно, что с увеличением разрешения относительно базового время счета растет линейно с увеличением числа ячеек расчетной области, однако эта линейность нарушается, когда разрешение сетки уменьшается: время счета уменьшается лишь в ≈2.5 раза при уменьшении разрешения в 4 раза ($1/4x - 15390$ ячеек), при дальнейшем уменьшении разрешения ($1/16x - 3944$ ячейки) счет сократился

только в 1.75 раза. Иными словами, для эффективной работы солвера на GPU необходимо использовать сетки с разрешением не менее 60 тыс. ячеек.

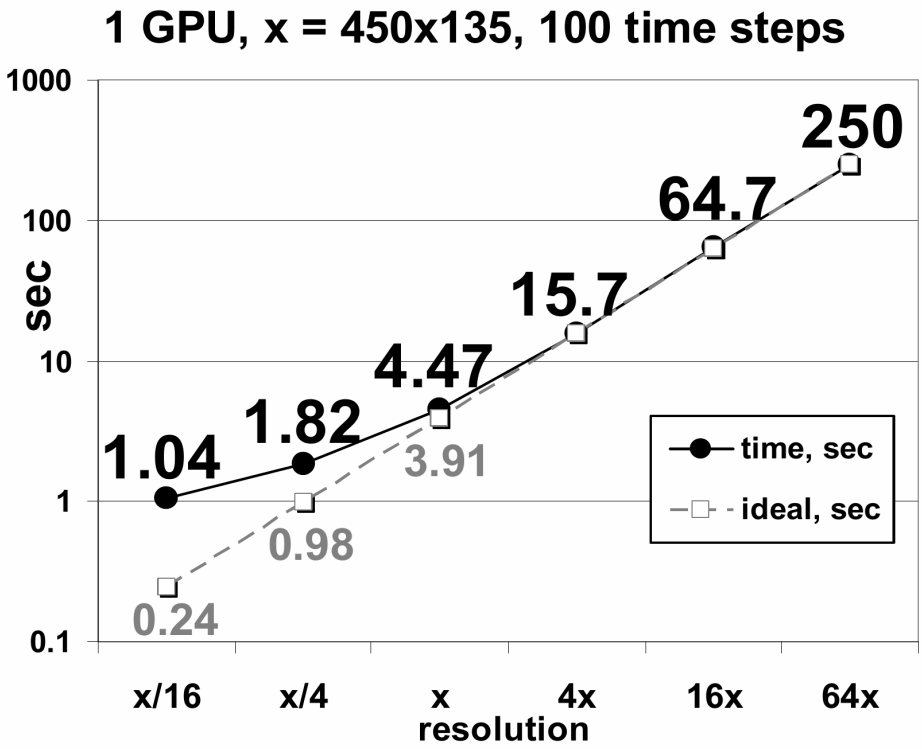


Рисунок 6.3 — Время счета на одном GPU с разными разрешениями расчетной сетки (базовое $x = 450 \times 135$)

6.2 Исследование взаимодействия библиотек CUDA и MPI

Задача из раздела 6.1 была запущена с сеткой разрешением 3600×1080 на разном числе GPU (Nvidia Tesla C2050, суперкомпьютер К-100), Рис. 6.4. Как видно, масштабирование, начиная с 2-х ускорителей, оказалось близко к линейному, и на 64 GPU (60.7 тыс ячеек / GPU) в соответствии с 6.1 составило 81.4 % (относительно 2 GPU).

Этот результат удалось достигнуть, как уже упоминалось ранее, благодаря двум ключевым факторам:

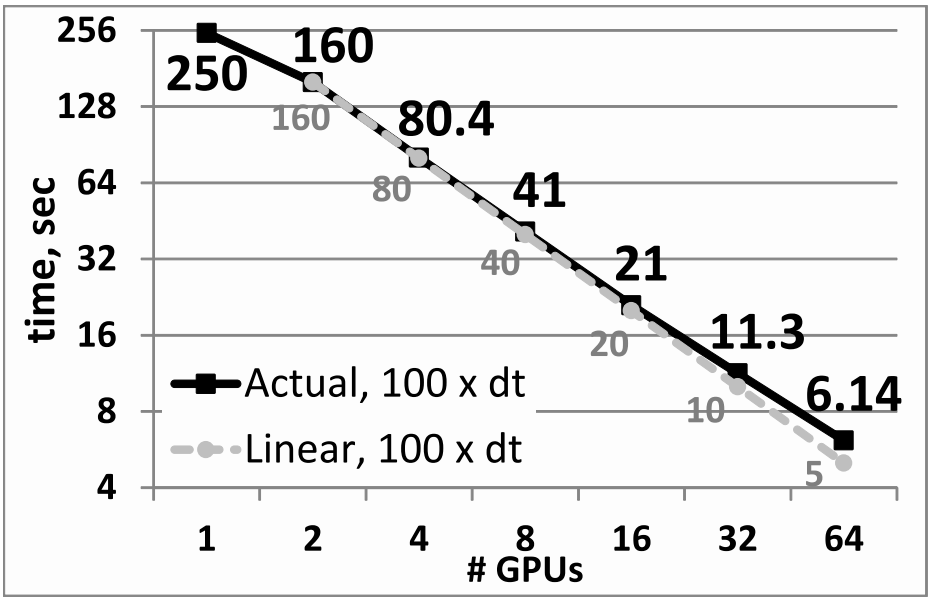


Рисунок 6.4 — Время счета на сетке 3600×1080 с разным числом GPU, 100 шагов по времени.

1. Параллельный алгоритм для метода LU-SGS не содержит в себе последовательных операций и позволяет совместить вычислительные операции с обменом данными в граничных ячейках подобластей.
2. Программная реализация для параллельного алгоритма в полной мере поддерживает совмещение по времени многоуровневого обмена данными(GPU VRAM $\rightarrow$ Pci-e $\rightarrow$ CPU RAM $\rightarrow$ Infiniband $\rightarrow$ CPU RAM $\rightarrow$ Pci-e $\rightarrow$ GPU VRAM) и вычислений, в значительной мере обеспечивая толерантность к задержкам передачи данных между ускорителями.

Однако при задействовании 2-х ускорителей вместо одного наблюдается значительная просадка в производительности – масштабируемость составила “лишь” 78.1 % (Рис. 6.4). Несмотря на это само по себе высокое значение (в сравнении с реализациями других решателей, упомянутых во введении), оно всё же является аномально низким, поскольку при переходе от 2 к 4, 8 GPU масштабируемость составила более 97 %. Данный результат является поводом для более детального анализа работы реализованного программного комплекса с помощью профилировщиков.

К сожалению, в большинстве работ, посвященных реализациям решателей на кластерных системах с графическими ускорителями, не приводятся

результаты подобного анализа, из-за чего сложно понять причины их весьма ограниченной масштабируемости.

Для рассматриваемой задачи запуск реализованной программы через CUDA профилировщик Nvprof [84] не прояснил причины просадки производительности при переходе от 1-го к 2-м GPU, а показал только, что на 2-х GPU есть простои между выполнением вычислительных ядер (kernels), при этом они не были вызваны использованием каких-либо функций из CUDA API. Стало понятно, что для дальнейшего анализа необходимо получить более сложные трассы, включающие как активность на GPU, так и операции на CPU (конкретно, вызовы функций MPI).

Одним из способов получения таких трасс является использование средств профилирования и анализа производительности – TAU [85], Vampir [86], Score-P [87]. Однако их использование требует отдельной настройки процедуры компиляции для исследуемой программы, а Vampir является коммерческим продуктом. Более простой и легковесной альтернативой является использование NVTX API [88] – интерфейса, позволяющего добавлять в трассы для GPU информацию о выполнении CPU кода путем “оборачивания” соответствующих (MPI) функций вызовами *nvtxRangePushA()/nvtxRangePop()* из этого API. Применив данный подход, для одной из тестовых задач была получена нужная трасса, Рис. 6.5, из которой стало ясно, что запуск ядер на GPU блокируется долгой работой *MPI_Wait()* (полоса “Markers and Ranges”), т.е. ожидание обновленных ghost ячеек в “white” блоках в процедуре решения второй подсистемы 2.41 (обратный обход).

Столь значительное время, занимаемое указанными *MPI_Wait()*, является аномальным, поскольку для других процедур (в частности, при прямом проходе по ячейкам) аналогичные вызовы занимают намного меньше времени и не приводят к простоям в GPU. Данный факт позволяет предположить, что причиной этому является неявная синхронизация в библиотеке MPI.

Для выяснения дальнейших деталей необходим учет особенностей внутренних механизмов работы библиотек MPI, задействуемых при пересылках

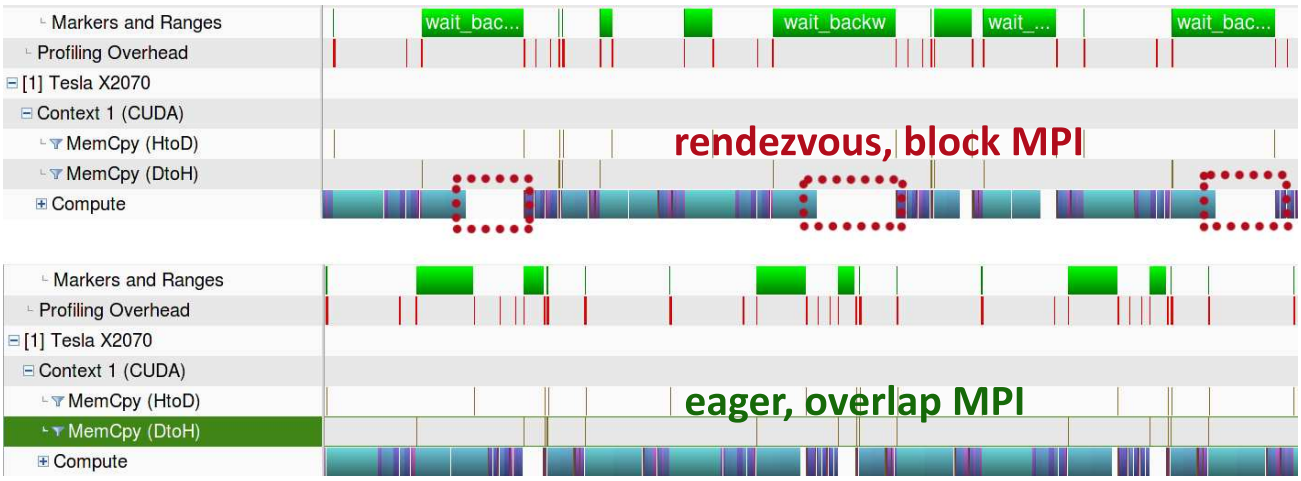


Рисунок 6.5 — Трасса для разработанного программного комплекса с протоколами *rendezvous* (вверху) и *eager* (внизу) передачи сообщений в MPI

“точка-точка”, т.е. в *Send/Recv* вызовах, включая и их неблокирующие разновидности [89]. Один из них – *rendezvous* (протокол обмена “с рукопожатием”) – используется для пересылки больших объемов данных между процессами, он не требует использования дополнительной памяти, кроме массивов, в которых содержатся соответственно передаваемые и принимаемые данные, но при этом в нем задействуется синхронизация между процессами, подтверждающая готовность принять данные в процессе получателе, причем она используется и для неблокирующих вызовов (*MPI_Isend()/MPI_Irecv()*). В другом механизме – eager (протокол обмена “без рукопожатий”) – передача данных между процессами происходит без указанной синхронизации, но при этом требуется дополнительная память под буфер-приемник сообщений в процессе-получателе, поскольку сообщение с принимаемыми данными может быть фактически получено до соответствующего вызова *MPI_*recv()*. Исходя из сказанного, в наиболее распространенных реализациях MPI сообщения до некоторого заданного размера передаются с помощью *eager* протокола (быстрый обмен данными с небольшим объемом дополнительной буферной памяти для этого), а свыше него – с помощью *rendezvous* (обмен с накладными расходами в виде дополнительной синхронизации между процессами, но без использования значительного по размеру памяти дополнительного буфера-приемника).

Рассматриваемые измерения проводились на суперкомпьютере “Лобачевский” с GPU Nvidia Tesla K20x, CUDA 6.5 и Intel MPI 4.1.3, в котором *rendezvous* протокол используется для сообщений размером более 256 KB. Поскольку размер сообщений в данном тесте составлял порядка 8 MB, то по умолчанию использовался именно этот протокол, т.е. между процессом-отправителем и процессом-приемником использовалась дополнительная синхронизация. Принудительное переключение на *eager* протокол посредством установки соответствующих переменных окружения при старте MPI программы позволило наконец избавиться от “долгих” вызовов *MPI_Wait()* и значительно сократить простои в GPU, Рис. 6.5. Как видно, Табл. 3, эффективность распараллеливания на 2-х GPU поднялась с 80–85 % до 95.2 % при принудительном использовании *eager* протокола вместо *rendezvous*. Приведенные данные демонстрируют, что особенности работы этих протоколов проявляются не только при межузловых взаимодействиях по сети Infiniband, но и при внутриузловых обменах с использованием только оперативной памяти.

Таблица 3 — Время работы (в секундах) разработанного программного комплекса с разными протоколами обмена данных в MPI

1 GPU	2 GPU на одном узле, *rendezvous*	2 GPU на разных узлах, *rendezvous*	2 GPU на одном узле, *eager*	2 GPU на разных узлах, *eager*
4.13	2.57	2.41	2.17	2.17

Выявленная выше проблема была не единственной, приводившей к простоям в GPU. На Рис. 6.6 (сверху) красным выделены ядра forward, запускавшиеся на отдельных “плитах” граничных ячеек подобласти (на соответствующих гранях) и очередной простой в GPU из-за вызова *MPI_Wait()* с уже включенным *eager* протоколом. Обнаруженные простои между выполнением указанных ядер привели к предположению о неявной активности библиотеки CUDA (cudart), которая проявляется на определенных ядрах (для других ядер, кроме forward, таких простоев не наблюдалось). Выяснилось, что в случае, если

используется достаточно большой объем локальной (в терминах CUDA) памяти (т.е. памяти, используемой под хранение локальных переменных в каждом треде) в ядрах, причем она в них отличается по размеру, то перед запуском каждого из ядер происходит соответствующее изменение объема этой памяти. Именно эта операция и приводит к простоям в GPU. Запрет на модификацию объема локальной памяти между запусками разных ядер путем вызова библиотечной CUDA функции *cudaDeviceLmemResizeToMax()* позволил выполнять одновременно несколько forward ядер для граничных ячеек (поскольку для этого было достаточно ресурсов GPU), сократив почти вдвое суммарное время их работы и выполнять на CPU вызовы MPI без блокировки запуска новых ядер на GPU, Рис. 6.6 (снизу).

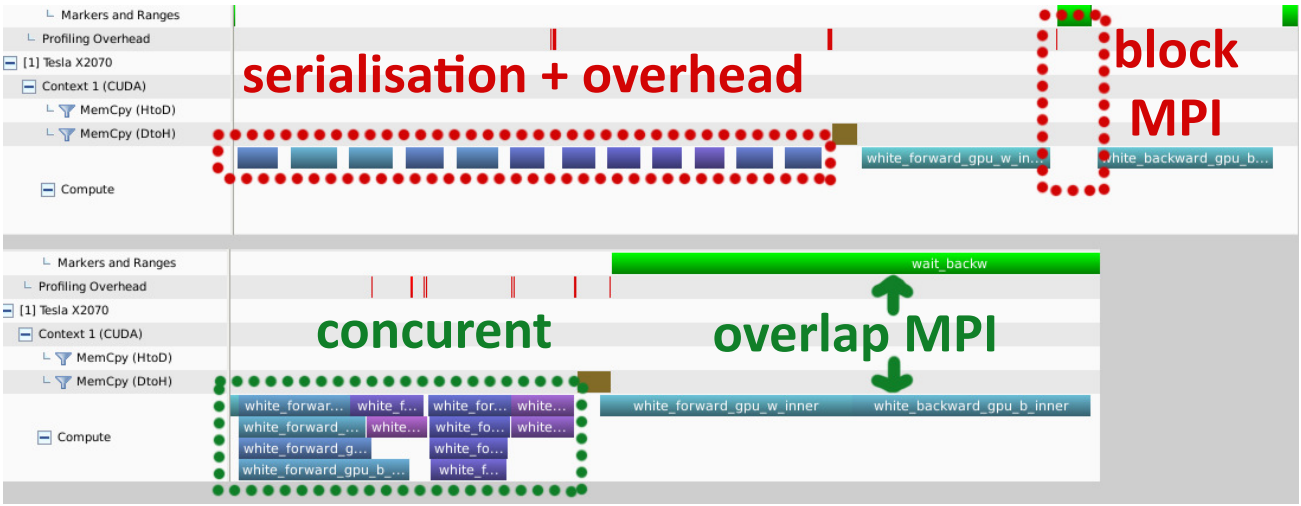


Рисунок 6.6 — Трасса для разработанного программного комплекса с настройками по умолчанию (вверху) и запретом уменьшения размера локальной памяти для треда в GPU (внизу)

Далее на примере крупномасштабных расчетов течения вокруг модели DLR F6 (6.7) показывается, как меняется производительность разработанного программного комплекса на нескольких десятках GPU с использованием оптимизаций, рассмотренных выше.

Для расчета течения использовалась сетка с 94 млн ячеек. Вычисления проводились на СК “Лобачевский” с ускорителями Nvidia Tesla K20X, минимальное число их в тесте – 15 (на меньшем числе ускорителей расчетная сетка не

умещалась в памяти GPU), Рис. 6.7. Как видно, масштабируемость комплекса на 180 GPU составила 81 % (относительно 15 GPU) с настройками MPI по умолчанию, а при принудительном использовании *eager* протокола – поднялась до 92 % с соответствующим 27 %-м ростом производительности на данном числе GPU.

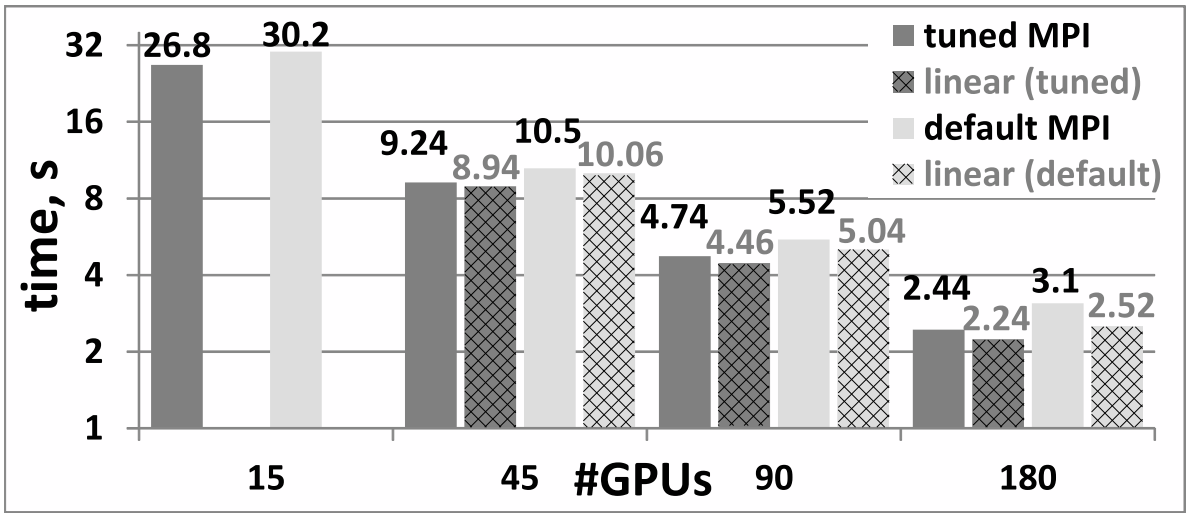


Рисунок 6.7 — Время счета (сек), DLR F6, $C = 8$, 94 млн ячеек, 10 шагов по времени с настройками MPI по умолчанию (default MPI) и оптимизированными настройками (tuned MPI)

Особо стоит отметить тот факт, что такой значительный рост был достигнут без модификации исходного кода разработанного программного комплекса, а лишь путем настройки параметров библиотеки MPI непосредственно перед запуском приложения, однако это потребовало нетривиального исследования особенностей взаимодействия библиотек CUDA и MPI и учета деталей реализации в последней из них.

6.3 Масштабируемость на большом числе GPU

Для исследования эффективности работы реализованного программного комплекса все ресурсы суперкомпьютера “Ломоносов” были выделены для проведения тестовых запусков в монопольном режиме. К сожалению, все его узлы с 1024 Nvidia Tesla X2070 не были доступны, удалось задействовать для расчетов

максимум 768 GPU. Как видно, Рис. 6.8, масштабируемость решателя на сетке с 150 млн ячеек оказалась достаточно близка к линейной и составила 75 % на 768 GPU (≈200 тыс ячеек/GPU) в соответствии с 6.1 (относительно 32 GPU – минимальное их число, в памяти которых помещалась расчетная сетка).

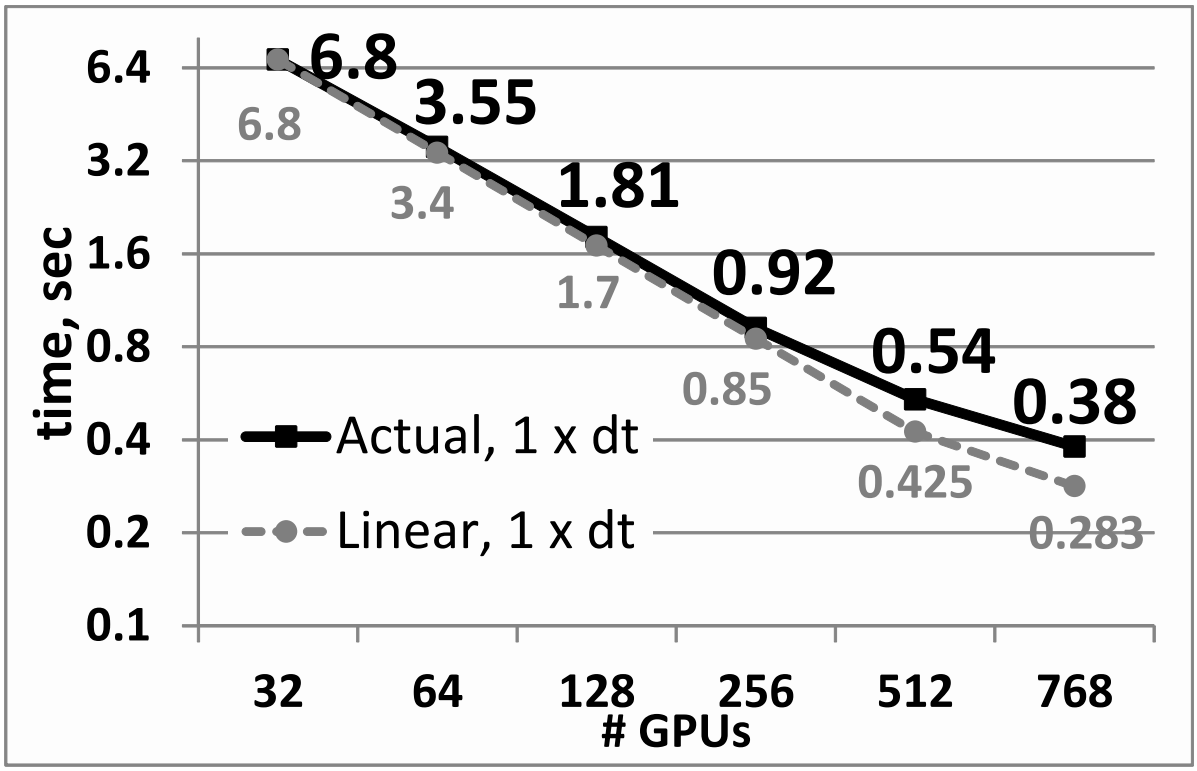


Рисунок 6.8 — Время счета (сек), 150 млн ячеек, 1 шаг по времени, СК “Ломоносов”

Глава 7. Заключение

В процессе работы над диссертацией получены следующие результаты:

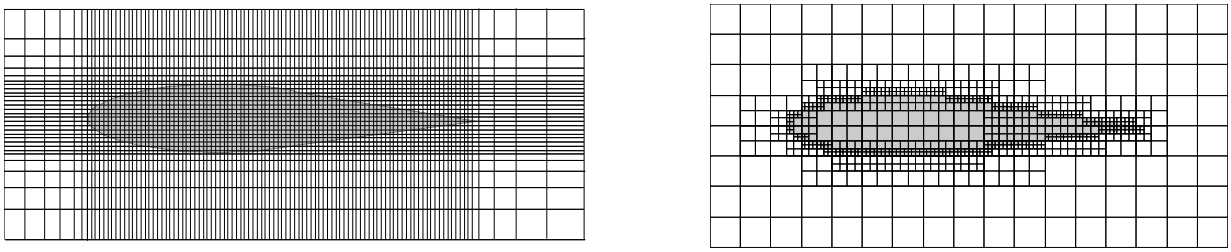
1. Предложено обобщение математической и численной модели свободной границы под специфику архитектуры графических ускорителей.
2. На основе модели свободной границы, гибридной явно-неявной схемы и итерационного метода LU-SGS разработан параллельный алгоритм для решения задач газовой динамики на вычислительных системах с графическими ускорителями.
3. Реализован программный комплекс на основе разработанного алгоритма с использованием технологий CUDA и MPI. Показана эффективность счета до 75% при использовании 768 GPU суперкомпьютера «Ломоносов».
4. С помощью разработанного программного комплекса проведены расчеты ряда газодинамических течений, включая моделирование обтекания вокруг профиля NACA0012 и DLR F6 на декартовой структурированной сетке, которые подтвердили высокую эффективность его работы.

7.1 Перспективы дальнейшей работы

Рассмотренная в данной работе программная реализация работает в рамках модели уравнений Эйлера, учет диссипативных эффектов возможен путем добавления дополнительного слагаемого в правую часть 2.25 – вязкого компенсационного потока, моделирующего работу сил трения на поверхностях твердых тел. Данная модификация не представляет значительных сложностей, поскольку параллельный алгоритм (3.3) при этом полностью сохраняет свою структуру, а

изменения в коде программной реализации будут носить лишь локальный (аддитивный) характер.

Основная сложность в дальнейшей работе связана с типом используемого сеточного разбиения – для более точного представления геометрии на сетке, а также для моделирования погранслоя необходимо использовать сгущения на декартовой сетке, которые приводят к итоговому ее избыточному разрешению, Рис. 7.1 а. Поэтому необходимо задействовать один из типов адаптивных сеток (AMR). Использование адаптивных неструктурированных [90], иерархических сеток на основе патчей (*patch-based, block-structured*) [91] на вычислительных системах с графическими ускорителями в большинстве случаев имеет весьма ограниченную *сильную* масштабируемость. Исключение составляют буквально несколько хорошо масштабируемых реализаций, например, [45], где также ключевое внимание уделено совмещению счета на ускорителях и обмена данными между ними, однако используемые в указанной работе неструктурированные сетки сложно сопрягаются с методом свободной границы, для них отсутствуют эффективные алгоритмы раскраски (что необходимо для параллельного LU-SGS) и динамической балансировки нагрузки.

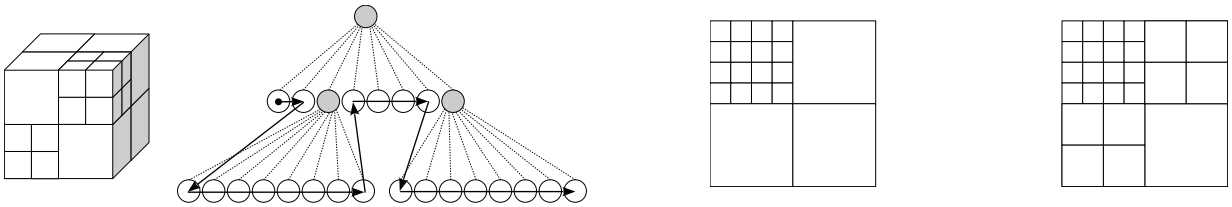


а) б)

Рисунок 7.1 — Расчетная область с а) декартовой сеткой и б) адаптивной сеткой на основе восьмеричных (квадратичных) деревьев

Промежуточное положение между неструктурированными и декартовыми сеточными разбиениями занимают сетки на основе восьмеричных (квадратичных) деревьев, когда ячейки исходной декартовой сетки могут рекурсивно делиться на 8 (4 в двумерном случае) подъячеек, Рис. 7.1 б. Каждая такая сетка однозначно представляется восьмеричным деревом, а ее ячейки линейно упорядочиваются в памяти с помощью кривой, заполняющей пространство (SFC),

Рис. 7.2 а. Данное упорядочивание, в отличие от неструктурированных сеток, отчасти обеспечивает локализацию обращений в память с учетом геометрического соседства ячеек, что позволяет более эффективно работать с памятью на графических ускорителях. Среди сеток на основе восьмеричных деревьев выделяется подтип т.н. $2:1$ сбалансированных сеток, Рис. 7.2 б, когда каждая ячейка имеет не более 4-х (2-х в двумерном случае) соседей по каждой из своих граней. Использование таких сеток значительно упрощает процедуры аппроксимации и позволялет избежать проблем со сходимостью решений.



а) б)

Рисунок 7.2 — а) Адаптивная сетка и соответствующее ее представление в виде восьмеричного дерева с кривой, заполняющей пространство; б) $2:1$ не сбалансированная (слева) и сбалансированная (справа) сетки

Для параллельного алгоритма LU-SGS требуется вновь решить задачу о раскраске графа (здесь и далее подразумевается использование только $2:1$ сбалансированных сеток). Очевидно, двух цветов для нее уже будет недостаточно, и в работах [20; 92] были предложены варианты ее решения, требующие соответственно 3 и 4 цвета для двумерного и трехмерного случая. С такой раскраской расчет выполняется сначала по ячейкам 1-го цвета, затем 2-го, 3-го (и 4-го в трехмерном случае) с соответствующими обменами граничными ячейками между подобластями, полученными после декомпозиции расчетной области, Рис. 7.3.

Глобальное упорядочивание всех ячеек с помощью SFC (z-кривой) Рис. 7.2 а позволяет эффективно и достаточно просто проводить как начальную декомпозицию области по процессам (ускорителям), так и динамическую балансировку (в случае сетки, адаптирующейся к решению/подвижной геометрии в процессе работы задачи), а алгоритмы перекраски для изменившихся ячеек [20; 92] выполняются локально с высокой эффективностью, не требуя

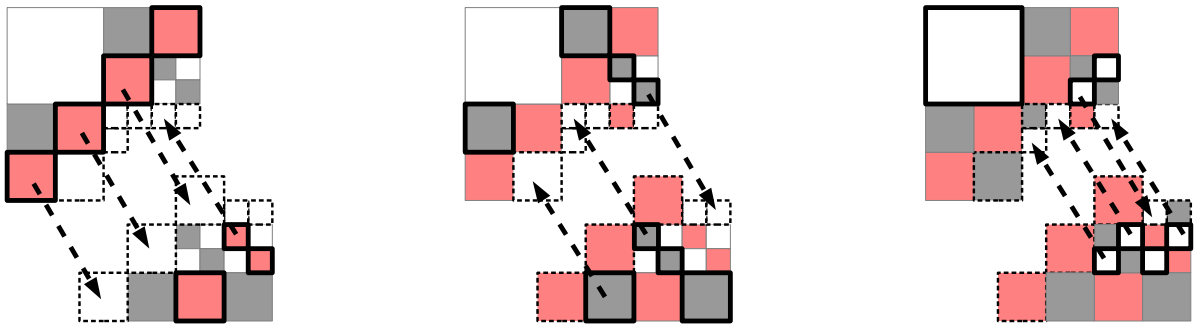


Рисунок 7.3 — Параллельный алгоритм для LU-SGS: обход ячеек (выделены жирными границами) с последующими обменами граничными ячейками между подобластями (указаны пунктирными стрелками) сначала для “красных”, затем “черных” и, наконец, “белых” ячеек

глобальной перекраски всех ячеек в отличие от многих других подобных алгоритмов. Всё это позволяет с высокой эффективностью и масштабируемостью реализовать параллельный алгоритм для LU-SGS на кластерных системах с распределенной памятью с использованием динамических адаптивных сеток на основе восьмеричных деревьев.

Очень высокая эффективность и масштабируемость до 262000 ядер CPU на рассматриваемых сетках была продемонстрирована библиотекой P4est [93]. Однако ее применимость на системах с графическими ускорителями, как и в подавляющем большинстве других подобных подходов, носит весьма ограниченный характер: в то время как решатель исполняется на GPU, все процедуры модификации сетки запускаются на CPU в *однопоточном* режиме (в рамках MPI-процесса), что требует регулярного копирования данных *всей* расчетной подобласти между GPU и CPU по шине Pci-express, которая имеет в разы меньшую пропускную способность по сравнению с подсистемой RAM памяти на GPU и CPU. Учитывая всё это, модификация сетки на CPU может занимать значительную часть от времени работы непосредственно решателя, поэтому было принято решение о разработке собственной библиотеки, которая позволяет производить все манипуляции с адаптивными сетками на основе восьмеричных деревьев *полностью* на GPU. Следует отметить, что в публичных источниках

найдена только одна работа [94], где реализуется аналогичный подход, но результаты которой еще не опубликованы.

В реализованной библиотеке в рамках продолжения данной работы адаптивная сетка хранится в памяти GPU в виде трех массивов (Рис. 7.4): базовые (макро)-ячейки – корни восьмеричных деревьев (□), узлы ветвления (•) и листовые узлы (непосредственно ячейки сеточного разбиения) (○) со ссылками между ними, определяющими структуру восьмеричного дерева. Каждый тред на GPU в параллель с остальными может измельчать/укрупнять свою ячейку, причем исходная(-ые) ячейка(-и) помечаются пустыми (×), а новые записываются в конец соответствующих массивов (Рис. 7.4). Очевидно, что в этом случае массивы могут бесконечно “разрастаться” с образованием множества пустых ячеек (“дыр”) внутри них, поэтому была реализована (также полностью на GPU) процедура дефрагментации с “уплотнением” массивов на основе z-кривой (SFC), проходящей по всем деревьям подобласти, (Рис. 7.5). После выполнения каждой из указанной процедур необходимо выполнять поиск геометрических соседей для листовых узлов (полностью на GPU), поскольку предыдущие ссылки на них могут оказаться недействительными. Предварительные результаты показали, что время работы процедур измельчения и укрупнения ячеек, Рис. 7.6, оказалось меньше или сравнимо с таковым в *P4est*. Иными словами, даже текущая неоптимизированная реализация на GPU позволяет модифицировать сетку *полностью* на GPU с производительностью, близкой к *P4est*, исключая значительный по времени этап обмена данными между CPU и GPU. Данный результат является очередным шагом к созданию высокомасштабируемого программного комплекса для решения задач газовой динамики с подвижной геометрией на динамически адаптируемых сетках, предназначенного для работы на системах с большим числом вычислителей с массивно-параллельными архитектурами.

Более подробное изложение деталей данного параграфа приведено в [21; 95].

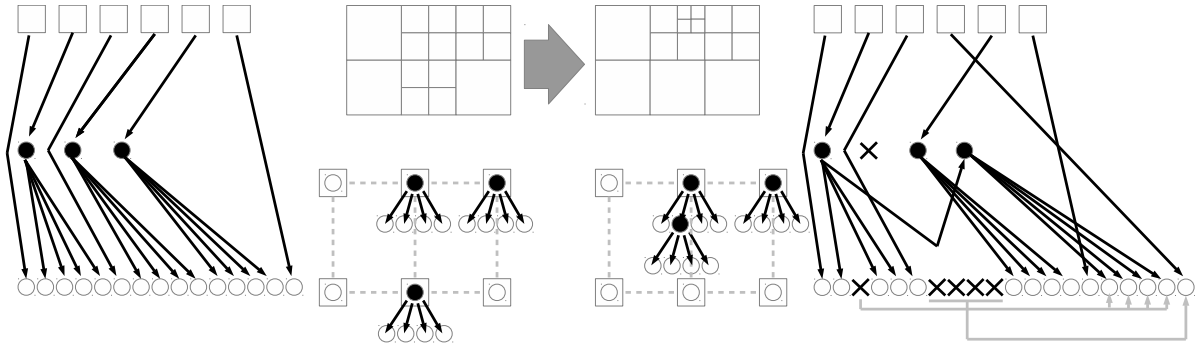


Рисунок 7.4 — Сетка, соответствующий граф и массивы для него в GPU до (слева) и после (справа) измельчения/укрупнения ячеек

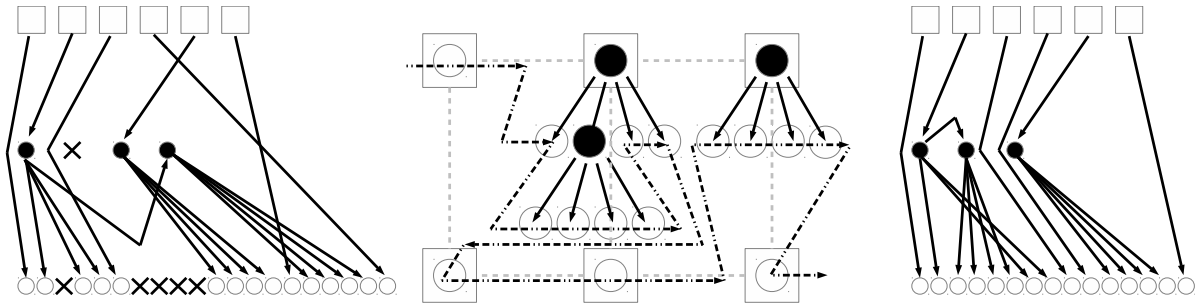


Рисунок 7.5 — Массивы для сетки в GPU до дефрагментации (слева), переупорядочивание ячеек на основе кривой, заполняющей пространство (в ценре), и массивы после дефрагментации (справа)

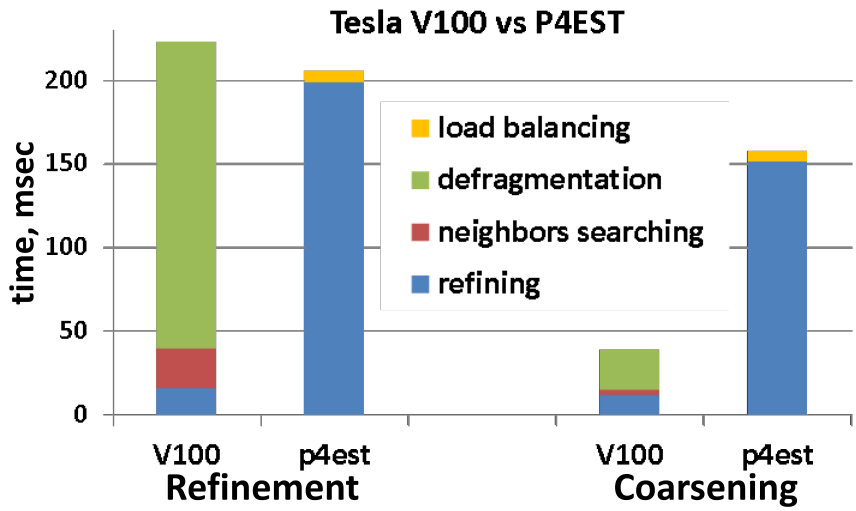


Рисунок 7.6 — Время выполнения процедур измельчения ячеек на сетке из 9×2^{20} ячеек (слева) и укрупнения ячеек на сетке из 9×2^{23} ячеек (справа).

Список литературы

1. *Меньшов, И. С.* Эффективный параллельный метод сквозного счета задач аэродинамики на несвязных декартовых сетках / И. С. Меньшов, П. В. Павлухин // Журнал вычислительной математики и математической физики. — 2016. — Янв. — Т. 56. — С. 1677—1691.
2. A multi-block viscous flow solver based on GPU parallel methodology / L. Fu [et al.] // Computers and Fluids. — 2014. — May. — Vol. 95. — P. 19—39.
3. *Jameson, A.* Implicit Schemes and LU Decompositions / A. Jameson, E. Turkel // Mathematics of Computation. — 1979. — Oct. — Vol. 37.
4. *Wright, M. J.* Data-parallel lower-upper relaxation method for the Navier-Stokes equations / M. J. Wright, G. V. Candler, M. Prampolini // AIAA Journal. — 1996. — Vol. 34, no. 7. — P. 1371—1377.
5. *Wissink, A. M.* Parallelization of a three-dimensional flow solver for Euler rotorcraft aerodynamics predictions / A. M. Wissink, A. S. Lyrintzis, R. C. Strawn // AIAA Journal. — 1996. — Vol. 34, no. 11. — P. 2276—2283.
6. LU-SGS versus DPLUR. — URL: https://www.aithercfd.com/2016/09/25/implicit-solver-comparison.html (дата обр. 01.07.2019).
7. Domain decomposition for an implicit LU-SGS scheme using overlapping grids / P. Stoll [et al.] // 13th Computational Fluid Dynamics Conference. — eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.1997-1896. — URL: https://arc.aiaa.org/doi/abs/10.2514/6.1997-1896.
8. Implementation of unstructured grid GMRES+LU-SGS method on shared-memory, cache-based parallel computers / D. Sharov [et al.] // 38th Aerospace Sciences Meeting and Exhibit. — eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2000-927. — URL: https://arc.aiaa.org/doi/abs/10.2514/6.2000-927.

9. Численное моделирование трехмерных течений с волнами детонации на многопроцессорной вычислительной технике / И. В. Семенов [и др.] // Вестник УГАТУ. — 2010. — Т. 14, № 5. — С. 140—149.

10. *Меньшов, И. С.* Метод свободной границы для численного решения уравнений газовой динамики в областях с изменяющейся геометрией / И. С. Меньшов, М. А. Корнев // Матем. моделирование. — 2014. — Т. 26, № 5. — С. 99—112.

11. *Jacobsen, D.* An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters / D. Jacobsen, J. Thibault, I. Senocak // Inanc Senocak. — 2010. — Jan. — Vol. 16.

12. Physis: An implicitly parallel programming model for stencil computations on large-scale GPU-accelerated supercomputers / N. Maruyama [et al.] //. — 11/2011. — P. 11.

13. CPU/GPU COMPUTING FOR AN IMPLICIT MULTI-BLOCK COMPRESSIBLE NAVIER-STOKES SOLVER ON HETEROGENEOUS PLATFORM / L. DENG [et al.] // International Journal of Modern Physics: Conference Series. — 2016. — Jan. — Vol. 42. — P. 1660163.

14. *Павлухин, П. В.* Реализация параллельного метода LU-SGS для задач газовой динамики на кластерных системах с графическими ускорителями / П. В. Павлухин // СБОРНИК ТРУДОВ. Международная научная конференция Параллельные вычислительные технологии 2012 / под ред. Л. Б. Соколинский, К. Пан. — Институт вычислительной математики и математической геофизики СО РАН, г. Новосибирск : Издательский центр ЮУрГУ (Челябинск), 2012. — С. 633—639.

15. *Павлухин, П. В.* Параллельный метод LU-SGS для трехмерных задач газовой динамики со сложной геометрией на вычислительных системах с многими графическими ускорителями / П. В. Павлухин // СБОРНИК

ТРУДОВ. Международная научная конференция Параллельные вычислительные технологии 2013 / под ред. Л. Б. Соколинский, К. Пан. — Южно-Уральский государственный национальный исследовательский университет, г.Челябинск : Издательский центр ЮУрГУ (Челябинск), 2013. — С. 483—488.

16. *Pavlukhin*, *P. V.* Parallel LU-SGS numerical method implementation for 3-dimensional gasdynamics problems on GPU-accelerated computer systems / P. V. Pavlukhin // Mathematical Modeling and Computational Physics (MMCP'2013): Book of Abstracts of the International Conference (Dubna, July 8-12, 2013). — Dubna: JINR, 2013. — P. 142.

17. *Павлухин*, *П. В.* ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ РЕШЕНИЯ ЗАДАЧ ГАЗОВОЙ ДИНАМИКИ НА GPU-СИСТЕМАХ ПЕТАФЛОПНОГО УРОВНЯ / П. В. Павлухин, И. С. Меньшов // НАУЧНЫЙ СЕРВИС В СЕТИ ИНТЕРНЕТ: МНОГООБРАЗИЕ СУПЕРКОМПЬЮТЕРНЫХ МИРОВ. Труды Международной суперкомпьютерной конференции. Российская академия наук Суперкомпьютерный консорциум университетов России. — Новороссийск, 22-27 сентября 2014 г., 2014. — С. 301—303.

18. *Павлухин*, *П. В.* Численное моделирование задач газовой динамики и аэроакустики на вычислительных системах с графическими ускорителями / П. В. Павлухин, И. С. Меньшов // Сборник тезисов Пятой всероссийской конференции "Вычислительный эксперимент в аэроакустике" 22 - 27 сентября 2014 года, г. Светлогорск Калининградской области. — ООО "МАКС Пресс" Москва, 2014.

19. *Pavlukhin*, *P.* On Implementation High-Scalable CFD Solvers for Hybrid Clusters with Massively-Parallel Architectures / P. Pavlukhin, I. Menshov // Parallel Computing Technologies / ed. by V. Malyshkin. — Cham : Springer International Publishing, 2015. — P. 436—444.

20. *Pavlukhin*, *P.* Parallel Algorithms for an Implicit CFD Solver on Tree-Based Grids / P. Pavlukhin, I. Menshov //. — 07.2017. — С. 151—158.

21. *Pavlukhin, P.* Parallel implicit matrix-free CFD solver using AMR grids / P. Pavlukhin, I. Menshov // Journal of Physics: Conference Series. — 2018. — Дек. — Т. 1141. — С. 012035.

22. *Павлухин, П. В.* Эффективная параллельная реализация метода LU-SGS для задач газовой динамики / П. В. Павлухин, М. И. С. // Научный вестник МГТУ ГА. — 2011. — № 165. — С. 46—55.

23. *Павлухин, П. В.* Реализация параллельного метода lu-sgs для задач газовой динамики на кластерных системах с графическими ускорителями / П. В. Павлухин // Вестник Нижегородского университета им. Н.И. Лобачевского. — 2013. — № 1—1. — С. 213—218.

24. *Меньшов, И. С.* Численное решение задач газовой динамики на декартовых сетках с применением гибридных вычислительных систем / И. С. Меньшов, П. В. Павлухин // Препринты ИПМ им.М.В.Келдыша. — 2014. — № 92. — URL: http://library.keldysh.ru/preprint.asp?id=2014-92.

25. *Menshov, I.* Highly Scalable Implementation of an Implicit Matrix-free Solver for Gas Dynamics on GPU-accelerated Clusters / I. Menshov, P. Pavlukhin // J. Supercomput. — Hingham, MA, USA, 2017. — Vol. 73, no. 2. — P. 631—638. — URL: https://doi.org/10.1007/s11227-016-1800-1.

26. *Dagum, L.* OpenMP: An Industry-Standard API for Shared-Memory Programming / L. Dagum, R. Menon // IEEE Comput. Sci. Eng. — Los Alamitos, CA, USA, 1998. — Jan. — Vol. 5, no. 1. — P. 46—55. — URL: https://doi.org/10.1109/99.660313.

27. *Nichols, B.* Pthreads Programming / B. Nichols, D. Buttlar, J. P. Farrell. — Sebastopol, CA, USA : O'Reilly & Associates, Inc., 1996.

28. Cilk: An Efficient Multithreaded Runtime System / R. D. Blumofe [et al.] // SIGPLAN Not. — New York, NY, USA, 1995. — Aug. — Vol. 30, no. 8. — P. 207—216. — URL: http://doi.acm.org/10.1145/209937.209958.

29. *Li, K.* Memory Coherence in Shared Virtual Memory Systems / K. Li, P. Hudak // ACM Trans. Comput. Syst. — New York, NY, USA, 1989. — Nov. — Vol. 7, no. 4. — P. 321—359. — URL: http://doi.acm.org/10.1145/75104.75105.

30. *Hoeflinger, J.* Programming with cluster openMP / J. Hoeflinger //. — 01/2007. — P. 270.

31. The TOP500: History, Trends, and Future Directions in High Performance Computing / H. W. Meuer [et al.]. — 1st. — Chapman & Hall/CRC, 2014.

32. CPU DB: Recording microprocessor history / A. Danowitz [et al.] // Communications of The ACM - CACM. — 2012. — Apr. — Vol. 55. — P. 55—63.

33. *Cook, S.* CUDA Programming: A Developer's Guide to Parallel Computing with GPUs / S. Cook. — 1st. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2013.

34. OpenCL Programming Guide / A. Munshi [et al.]. — 1st. — Addison-Wesley Professional, 2011.

35. *Forum, M. P.* MPI: A Message-Passing Interface Standard : tech. rep. / M. P. Forum. — Knoxville, TN, USA, 1994.

36. Running unstructured grid-based CFD solvers on modern graphics hardware / A. Corrigan [et al.] // International Journal for Numerical Methods in Fluids. — 2011. — Vol. 66, no. 2. — P. 221—229. — URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.2254.

37. Многофункциональный пакет программ ЛОГОС для расчета задачгидродинамики и тепломассопереноса на суперЭВМ. Базовые технологии и алгоритмы / А. Козелков [и др.] // Сборник трудов XIIIМеждународного семинара "Супервычисления и математическое моделирование". 11–15 октября 2010. Саров,Россия. — 2011. — С. 215—230.

38. Реализация параллельных вычислений на графических процессорах в пакете вычислительной газовой динамики ЛОГОС / К. Волков [и др.] // Выч. мет. программирование. — 2013. — Т. 14, № 3. — С. 334—342.

39. FlowVision software: Numerical simulation of industrial CFD applications on parallel computer systems / A. Aksenov [et al.]. — 2004. — Jan.

40. *Peskin*, *C. S.* The immersed boundary method / C. S. Peskin // Acta Numerica. — 2002. — Vol. 11. — P. 479—517.

41. *Saad*, *Y.* GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems / Y. Saad, M. H. Schultz // SIAM J. Sci. Stat. Comput. — Philadelphia, PA, USA, 1986. — July. — Vol. 7, no. 3. — P. 856—869. — URL: https://doi.org/10.1137/0907058.

42. *Vorst*, *H. A. van der*. BI-CGSTAB: A Fast and Smoothly Converging Variant of BI-CG for the Solution of Nonsymmetric Linear Systems / H. A. van der Vorst // SIAM J. Sci. Stat. Comput. — Philadelphia, PA, USA, 1992. — Mar. — Vol. 13, no. 2. — P. 631—644.

43. *Bahi*, *J. M.* Parallel GMRES Implementation for Solving Sparse Linear Systems on GPU Clusters / J. M. Bahi, R. Couturier, L. Z. Khodja // Proceedings of the 19th High Performance Computing Symposia. — Boston, Massachusetts : Society for Computer Simulation International, 2011. — P. 12—19. — (HPC '11).

44. GPU Accelerated Unconditionally Stable Crank-Nicolson FDTD Method for the Analysis of Three-Dimensional Microwave Circuits / K. Xu [et al.] // Progress In Electromagnetics Research. — 2010. — Vol. 102.

45. MPI-CUDA sparse matrix-vector multiplication for the conjugate gradient method with an approximate inverse preconditioner / G. Oyarzun [et al.] // Computers and Fluids. — 2014. — Mar. — Vol. 92. — P. 244—252.

46. Hiding Global Communication Latency in the GMRES Algorithm on Massively Parallel Machines / P. Ghysels [et al.] // SIAM Journal on Scientific Computing. — 2013. — Vol. 35, no. 1. — P. C48—C71.

47. *Wissink, A. M.* Parallelization of a three-dimensional flow solver for Euler rotorcraft aerodynamics predictions / A. M. Wissink, A. S. Lyrintzis, R. C. Strawn // AIAA Journal. — 1996. — Vol. 34, no. 11. — P. 2276—2283.

48. An efficient wavefront parallel algorithm for structured three dimensional LU-SGS / C. Gong [et al.] // Computers and Fluids. — 2016. — May. — Vol. 134.

49. *ANDERSON, W. K.* Comparison of finite volume flux vector splittings for the Euler equations / W. K. ANDERSON, J. L. THOMAS, B. VAN LEER // AIAA Journal. — 1986. — Vol. 24, no. 9. — P. 1453—1460.

50. *Leer, B. van.* Towards the Ultimate Conservative Difference Scheme V. A Second-order Sequel to Godunov's Method / B. van Leer // Journal of Computational Physics. — 1979. — July. — Vol. 32. — P. 101—136.

51. *Fromm, J. E.* A Method for Reducing Dispersion in Convective Difference Schemes / J. E. Fromm // J. Comput. Phys. — San Diego, CA, USA, 1968. — Oct. — Vol. 3, no. 2. — P. 176—189.

52. *Годунов, С. К.* Разностные схемы / С. К. Годунов, В. С. Рябенький. — М.: Наука, 1977. — С. 440.

53. *Калган, В. П.* Применение принципа минимальных значений производной к построению конечно-разностных схем для расчета разрывных решений газовой динамики / В. П. Калган // Ученые записки ЦАГИ. — 1972. — Т. 3, № 6. — С. 68—77.

54. *Albada, G. D. van.* A comparative study of computational methods in cosmic gas dynamics / G. D. van Albada, B. van Leer, W. W. Roberts Jr. // Astron. Astrophys. — 1982. — Apr. — Vol. 108. — P. 76—84.

55. *Годунов, С. К.* Разностный метод численного расчета разрывных решений уравнений гидродинамики / С. К. Годунов // Мат. сборник. — 1957. — Апр. — Т. 47, № 3. — С. 271—306.

56. Численное решение многомерных задач газовой динамики / С. К. Годунов [и др.]. — М.: Наука, 1976. — С. 440.

57. *Toro*, *E.* Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction / E. Toro //. — 01/2009.

58. *Русанов*, *В. В.* Разностные схемы третьего порядка точности для сквозного счета разрывных решений / В. В. Русанов // Докл. АН СССР. — 1968. — Т. 180, № 6. — С. 1303—1305.

59. *Menshov*, *I.* Hybrid Explicit-Implicit, Unconditionally Stable Scheme for Unsteady Compressible Flows / I. Menshov, Y. Nakamura // AIAA Journal. — 2004. — Vol. 42, no. 3. — P. 551—559.

60. *Boiron*, *O.* A High-Resolution Penalization Method for large Mach number Flows in the presence of Obstacles / O. Boiron, G. Chiavassa, R. Donat // Computers and Fluids. — 2009. — Mar. — Vol. 38.

61. *Ландау*, *Л. Д.* Гидродинамика. 3-е изд. / Л. Д. Ландау, Е. Лифшиц. — М.: Наука, 1986. — С. 736.

62. *Menshov*, *I.* A High-Resolution Penalization Method for large Mach number Flows in the presence of Obstacles / I. Menshov, Y. Nakamura // Collection of technical papers of 6th Int. Symp. on CFD, Lake Tahoe, Nevada. — 1995. — P. 815—821.

63. Parallel implementation of an implicit scheme based on the LU-SGS method for 3D turbulent flows / V. Borisov [et al.] // Mathematical Models and Computer Simulations. — 2015. — May. — Vol. 7. — P. 222—232.

64. *Titarev*, *V.* Construction and comparison of parallel implicit kinetic solvers in three spatial dimensions / V. Titarev, M. Dumbser, S. Utyuzhnikov // Journal of Computational Physics. — 2014. — Янв. — Т. 256. — С. 17—33.

65. ScaleMP - Virtualization for high-end computing. — URL: https://www.scalemp.com/ (visited on 07/01/2019).

66. Parallel algorithm for cfd lu-sgs time stepping with two dimensional structured meshes (in chinese) / C. Gong [и др.] // Journal of Frontiers of Computer Science and Technology. — 2013. — Т. 7, № 10. — С. 916—923.

67. *Уткин, П. С.* Численное моделирование инициирования и распространения волн газовой детонации в профилированных трубах : Дис. ... канд. физ.-мат. наук: 05.13.18; [Место защиты: Моск. физ.-техн. ин-т (гос. ун-т)] / П. С. Уткин. — М., 2010. — С. 167.

68. Parallel Versions of Implicit LU-SGS Method / A. Chikitkin [et al.] // Lobachevskii Journal of Mathematics. — 2018. — May. — Vol. 39, no. 4. — P. 503—512. — URL: https://doi.org/10.1134/S1995080218040054.

69. Многопоточная OpenMP-реализация метода LU-SGS с использованием многоуровневой декомпозиции неструктурированной расчетной сетки / М. Петров [и др.] // Ж. вычисл. матем. и матем. физ. — 2017. — Т. 57, № 11. — С. 1895—1905.

70. DLR-F6 Geometry - Drag Prediction Workshop - NASA. — URL: https://aiaa-dpw.larc.nasa.gov/Workshop2/DLR-F6-geom.html (дата обр. 01.07.2019).

71. Efficient Inter-node MPI Communication Using GPUDirect RDMA for InfiniBand Clusters with NVIDIA GPUs / S. Potluri [и др.] // 2013 42nd International Conference on Parallel Processing. — 10.2013. — С. 80—89.

72. NVIDIA GPUDirect. — URL: https://developer.nvidia.com/gpudirect (visited on 07/01/2019).

73. OpenFabrics Alliance. — URL: https://www.openfabrics.org/ (visited on 07/01/2019).

74. OpenACC. — URL: https://www.openacc.org/ (visited on 07/01/2019).

75. cuBLAS library. — URL: docs.nvidia.com/cuda/cublas/index.html (visited on 07/01/2019).

76. Finite volume simulation of a flow over a NACA 0012 using Jameson, MacCormack, Shu and TVD esquemes / O. Arias [и др.] // Mec. Comput. — 2007. — Янв. — Т. 26.

77. *Falcinelli*, *O.* Reducing the Numerical Viscosity in Nonstructured Three-Dimensional Finite Volume Computations / O. Falcinelli, S. Elaskar, J. Tamagno // Journal of Spacecraft and Rockets. — 2008. — Март. — Т. 45. — С. 406—408.

78. *Jameson*, *A.* Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time Stepping Schemes / A. Jameson, W. Schmidt, E. Turkel // AIAA Paper. — 1981. — Июль. — Т. 81.

79. ANSYS Fluent Software | CFD Simulation. — URL: https://www.ansys.com/products/fluids/ansys-fluent (visited on 07/01/2019).

80. Summary of the fourth AIAA CFD drag prediction workshop / J. Vassberg [и др.] //. — 06.2010.

81. Гибридный вычислительный кластер K-100. — URL: http://www.kiam.ru/MVS/resourses/k100.html (дата обр. 01.07.2019).

82. Суперкомпьютер “Лобачевский”. — URL: http://top50.supercomputers.ru/systems/511 (дата обр. 01.07.2019).

83. Практика суперкомпьютера “Ломоносов” / В. В. Воеводин [и др.] // Открытые системы. — Москва, 2012. — № 7. — С. 36—39. — описание суперкомпьютера “Ломоносов”.

84. NVProf - NVIDIA Developer Documentation. — URL: docs.nvidia.com/cuda/profiler-users-guide/index.html (дата обр. 01.07.2019).

85. *Shende*, *S. S.* The Tau Parallel Performance System / S. S. Shende, A. D. Malony // Int. J. High Perform. Comput. Appl. — Thousand Oaks, CA, USA, 2006. — Май. — Т. 20, № 2. — С. 287—311.

86. The Vampir Performance Analysis Tool-Set / A. Knüpfer [и др.] // Tools for High Performance Computing / под ред. M. Resch [и др.]. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. — С. 139—155.

87. Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir / A. Knüpfer [и др.] // Tools for High Performance Computing 2011 / под ред. H. Brunst [и др.]. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2012. — С. 79—91.

88. NVIDIA Tools Extensions. — URL: https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvtx (дата обр. 01.07.2019).

89. MPI Message Passing Protocols. — URL: https://computing.llnl.gov/tutorials/mpi_performance/#Protocols (дата обр. 01.07.2019).

90. ParFUM: a parallel framework for unstructured meshes for scalable dynamic physics applications / O. Lawlor [и др.] // Eng. Comput. (Lond.) — 2006. — Дек. — Т. 22. — С. 215—235.

91. Resident Block-Structured Adaptive Mesh Refinement on Thousands of Graphics Processing Units / D. Beckingsale [и др.] //. — 09.2015.

92. Coloring Octrees / U. Adamy [и др.] // Computing and Combinatorics / под ред. K.-Y. Chwa, J. I. J. Munro. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2004. — С. 62—71.

93. *Burstedde, C.* P4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees / C. Burstedde, L. Wilcox, O. Ghattas // SIAM Journal of Scientific Computing. — 2011. — Июль. — Т. 33, № 3. — С. 1103—1133.

94. *Brown, A.* Towards achieving GPU-native adaptive mesh refinement / A. Brown. — 2017. — URL: https://www.oerc.ox.ac.uk/sites/default/files/uploads/ProjectFiles/CUDA//Presentations/2017/A_Brown_8th_March.pdf (visited on 07/01/2019).

95. *Pavlukhin, P.* GPU-Aware AMR on Octree-Based Grids / P. Pavlukhin, I. Menshov // PaCT 2019 LNCS 11657. — 2019. —.