Министерство образования и науки Российской Федерации

Государственное образовательное учреждение
высшего профессионального образования
«Ярославский государственный университет им. П.Г. Демидова»
(ЯрГУ)

Кафедра компьютерных сетей

«Допустить к защите»

Заведующий кафедрой
д. ф.-м. н., профессор
_____Глызин С. Д.
«__»_____20__ г.

Выпускная квалификационная работа бакалавра
по направлению 010500.62 Прикладная математика и информатика

Фреймворк для конечно-разностного моделирования диффузионных задач на гибридных вычислительных кластерах

Научный руководитель
д. ф.-м. н., профессор
_____Глызин С. Д.
«___»_____2015 г.

Студент группы ИВТ-43-БО
_____Фролов Д. А.
«___»_____2015 г.

Ярославль 2015

Содержание

Введение

К классу диффузионных относятся такие задачи как уравнение теплопроводности или реакция Белоусова-Жаботинского.

Развитие современного общества зачастую ставит перед наукой цели, решение которых требует решения самых разнообразных систем дифференциальных уравнений. В том числе и систем диффузионных уравнений. К таким задачам можно отнести уравнение теплопроводности !!! (примеры других задач). Каждую из них можно решать с помощью численных методов, например, методом Эйлера, многошаговыми методами Рунге-Кутты или методами Дормана-Принса. Подобные вычисления удобно автоматизировать, чтобы в дальнейшем иметь возможность быстро производить расчеты.

1 Постановка задачи

1.1 Теоретические основы решения диффузионных задач

Как известно, в общем случае задачи реакции диффузии представимы в виде системы дифференциальных уравнений:

$$\frac{\partial u}{\partial t} = D \triangle u + F(u);$$

$$\frac{\partial u}{\partial v} = 0, F(0) = 0.$$

Для их решения применяются различные численные методы, в том числе метод Эйлера, метод Ругне-Кутты, Дормана-Принса. Все эти методы в той или иной степени могут быть использованы для численного решения задачи реакции диффузии. Кратко рассмотрим каждый из вышеперечисленных методов.

Метод Эйлера считается наиболее простым численным методом решения дифференциальных уравнений. Он относится к явным, одношаговым методам первого порядка точности и фактически основан на аппроксимации интегральной кривой кусочно-линейной функцией. Упрощенно метод Эйлера может быть представлен в виде формулы:

$$y_i = y_{i-1} + \triangle t \cdot f(t, y_{i-1}).$$

Второй метод, который может быть использован для решения поставленной задачи, – метод Рунге-Кутты. Данный метод кратко описывается следующей схемой.

Пусть дана начальная задача Коши

$$\dot{y} = f(x, y)$$

и некоторое начальное условие

$$y(x_0) = y_0.$$

Тогда каждую последующую точку решения можно вычислить с помощью предыдущей, используя следующее равенство:

$$y_{n+1} = y_n + \sum_{i=0}^{s} b_i k_i.$$

Коэффициенты k_i вычиляются, исходя из следующих соотноше-

ний:

$$k_1 = hf(x_n, y_n),$$
$$k_2 = hf(x_n + c_2 h, y_n + a_{21} k_1),$$
$$\dots$$
$$k_s = hf(x_n + c_s, y_n + a_{s1} k_1 + a_{s2} k_2 + \dots + a_{s,s-1} k_{s-1}).$$

Количество коэффицентов k_i определяет порядок метода: к примеру, для четырехстадийной схемы Рунге-Кутты необходимо вычислить четрые таких коэффициента.

Для параметров a_{ij}, b_i, c_i, которые используются при расчеты k_i, должны выполняться следующие равенства:

$$\sum_{j=1}^{i-1} a_{ij} = c_i,$$

$$\sum_{j=1}^{s} b_i = 1,$$
$$0 \leq (b_i, c_i, a_{ij}) \leq 1,$$
$$b_i > 1.$$

Именно параметры a_{ij}, b_i и c_i задают конкретный метод Рунге-Кутты.

Как правило, для решения дифференциальных уравнений используются схемы невысоких порядков точности, поэтому в нашем случае достаточно воспользоваться четырехстадийной реализацией метода Рунге-Кутты, то есть взять $s = 4$.

Для решения задачи диффузии, как уже было сказано выше, также может быть использован метод Дормана-Принса, который фактически является модификацией метода Рунге-Кутты. По сравнению с последним он является более точным за счет большего числа стадий вычисления нового состояния. Среди особенностей метода также стоит отметить изменяемость шага вычислений.

Очевидно, что применение того или иного метода, оказывает влияние на точность и скорость производимых вычислений и, естественно, конечный результат. В связи с этим, выбор конкретного метода решения следует оставить за пользователем.

1.2 Основные требования к приложению

33

2 Архитектура приложения

2.1 Класс Solver и его потомки

Автоматизация вычислений дифференциальных уравнений, в том числе и диффузионных, требует использования численных методов решения. В данном проекте было реализовано несколько подобных методов:

1. метод Эйлера;
2. метод Рунге-Кутта;
3. метод Дормана-Принса.

Первый методов является самым простым из всех перечисленных. Включает в себя одну стадию вычислений и имеет постоянный шаг.

Метод Рунге-Кутты, представленный в проекте, аналогично первому методу имеет неизменяемый шаг, но использует четырехстадийную схему решения.

Последний метод использует семистадийную схему расчетов и отличается от двух ранее рассмотренных наличием изменяемого шага, а также необходимостью предварительной подготовки перед началом работы.

Для реализации всего вышеизложенного используются класс Solver и его наследники. Сам класс Solver лишь определяет интерфейс, которым должны обладать классы, реализующие методы численного решения.

В процессе работы необходимо знать некоторую общую информацию об используемом методе, для того чтобы корректно продолжать вычисления. Например, нужно знать сколько стадий необходимо методу или нужно ли выполнять изменение шага по времени. Подобную информацию содержать классы EukerSolver, RK4Solver и DP45Solver, которые являются наследниками Solver'а. Каждый из них обладает всей информацией о методе, которая может потребоваться в процессе вычислений: количество стадий, постоянный или изменяемый шаг, необходима ли методу предварительная подготовка. Кроме того здесь же содержаться различные константы необходимые методу, например в DP45Solver это могут быть минимальный и максимальный коэффициенты изменения шага. Все эти классы не хранят данные о текущем состоянии решения, они предназначены исключительно для предоставления справочной информации.

Полноценными классами, способными хранить данные являются наследники вышеописанных классов, а именно EulerSolverCpu, EulerSolverGpu, RK4SolverCpu и так далее. Как видно из названий этих классов каждая

пара предназначена для работы на центральном процессоре и видеокарте соответственно. Подобное решение продиктовано разным способами выделения и работы с памятью на двух этих типах устройств.

Общую схему классов можно увидеть на Рис. 1.

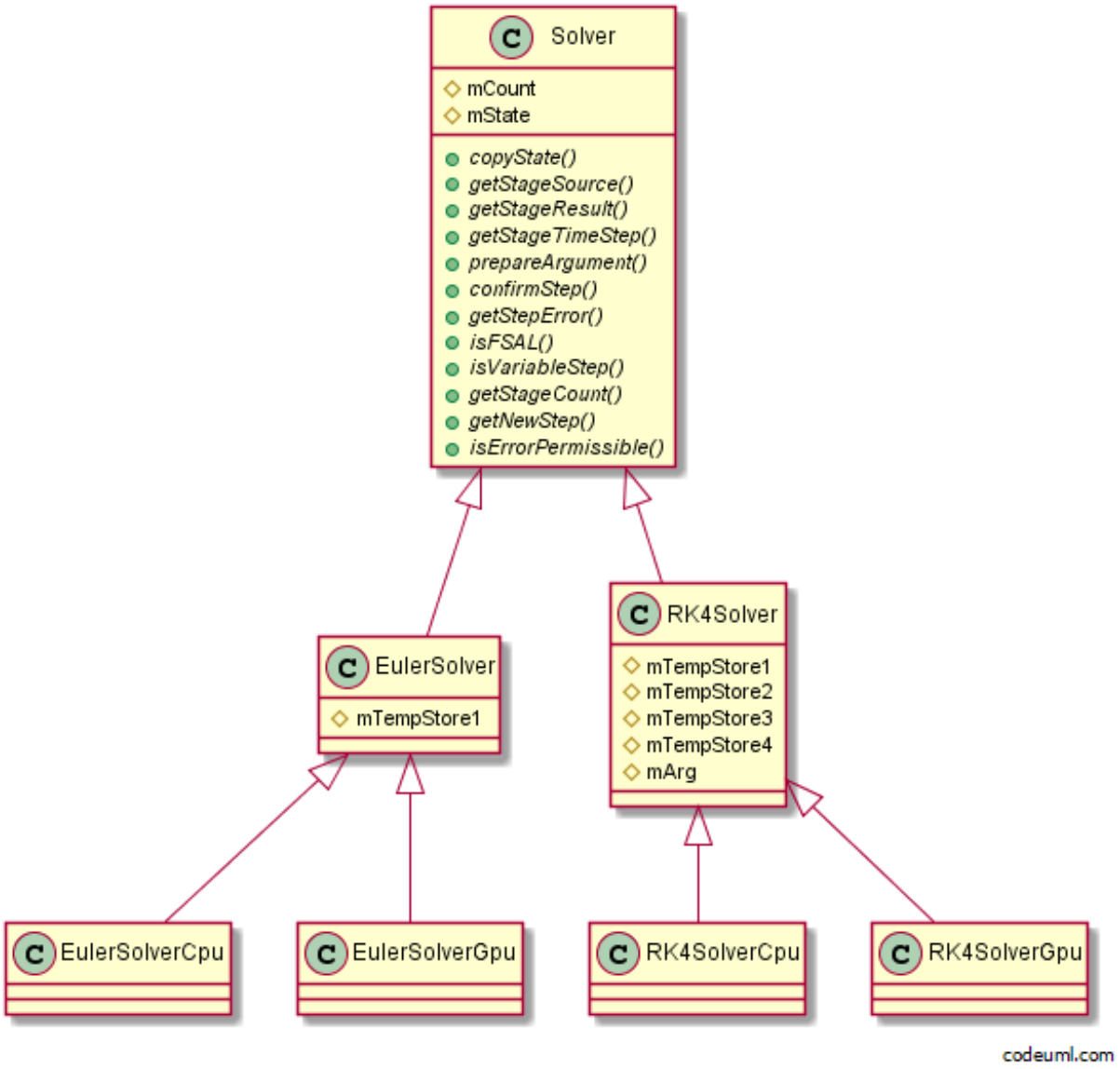

Рис. 1 Иерархия наследования для методов численного решения

Рассмотрим подробнее работу данных классов на примере RK4SolverCpu. Экземпляр данного класса фактически является «хранилищем» данных. Текущее состояние хранится в mState, данное поле имеют все без исключения классы, реализующие различные численные методы. Кроме того существует несколько дополнительных «хранилищ» (mTempStore1, mTempStore2, mTempStore3, mTempStore4), они используются для сохранения результатов вычисления на разных стадиях. Последний подобный «склад» это mArg, он необходим при получении нового состояния с использованием результатов, полученных ранее при выполнении четырех стадий метода.

В процессе вычислений объект данного класса выдает источник данных и их приемник, для этого используются описанные ранее переменные. Например, на первой стадии в качестве источника информации будет предоставлено mState, а запишется информация в mTempStore1. Стоит отметить, что сам объект не занимается непосредственно вычислениями, это работа другого класса. После каждой стадии метода выполняется функция prepareArgument() она занимается необходимой обработкой полученных данных перед следующей итерацией метода.

После того, как шаг метода выполнен выполняется его подтверждение (confirmStep()), метод Рунге-Кутты является методом с постоянным шагом и никогда не отвергает полученные результаты, поэтому данная функция просто меняет местами содержимое mState и mArg, так как в последнем будет храниться новое состояние.

2.2 Класс Block и его потомки

Обсуждая вопрос архитектуры системы невозможно обойти одну из самых важных составляющих приложения - блоки. Область, над которой производятся вычисления, разбивается на составные части, которые и называются блоками. Для их представления используются класс Block и его потомки.

Класс Block является абстрактным классом и необходим для описания свойств, которые присущи всем блокам независимо от того, на каком типе устройства они будут выполняться. Например, к таким свойствам можно отнести размерность. каждый блок может быть одномерным, двумерным или трехмерным в зависимости от поставленной задачи. Кроме того каждый блок обладает определенными координатами в пространстве и размерами. Важной информацией о блоке также является тип устройства, на котором будет производится вычисление конкретно данного блока, и номер устройства, хранение этих данных тоже предусмотрена в данном классе.

Каждый блок должен обладать определенным набором методов, которые описаны в классе Block. К таким методам относится, например, вычисление нового состояния (computeStageCenter() и computeStageBorder()), подготовка к новой стадии численного метода (prepareArgument()), подтверждение выполненного шага и несколько других. Все эти методы позволяют полноценно работать с блоком и осуществлять вычисления.

Всего у класса Block существует три наследника (см. Рис. 2):

1. BlockCpu;
2. BlockGpu;
3. BlockNull.

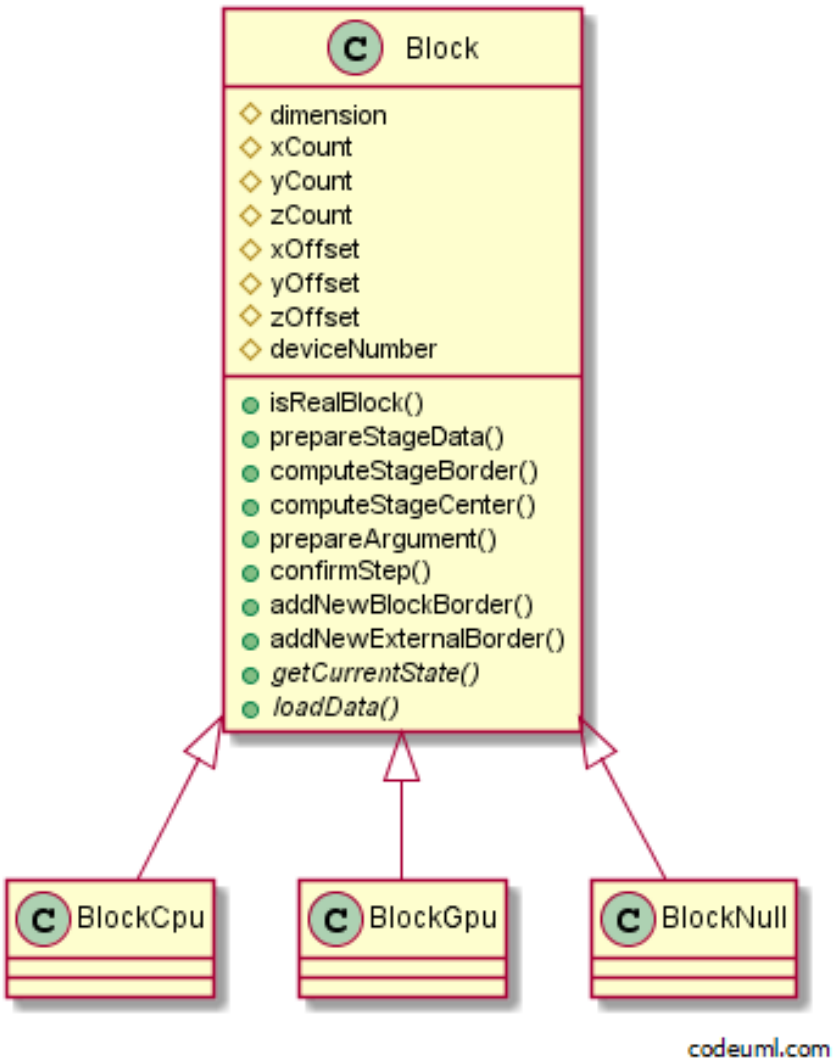

Рис. 2 Иерархия наследования для блоков

Первые два класса-наследника работают на центральном процессоре и видеокарте соответственно. Как и в случае с Solver'ами подобное разделение обусловлено различными способами работы с памятью, но в данном случае это не единственная причина. В зависимости от типа устройства меняется и способ осуществления параллельных вычислений: для центрального процессора используется библиотека OpenMP, а для видеокарты язык CUDA. Подробнее это будет рассмотрено в одной из следующих глав. Объекты этих классов создают себе «хранилища» (наследников класса Solver) для работы с данными и реально занимаются вычислениями.

BlockNull класс-наследник Block, который никаким образом не хранит данные текущем состоянии той части области, за которую он по идее отвечает. На самом деле он нужен для того чтобы обозначить, что составляющая области, относящаяся к данному блоку на самом деле обрабатывается другим потоком и на другой машине. Но он позволяет выполнить все функции присущие блоку: получение координат и размеров,

даже выполнение расчетов, правда в данном случае ничего рассчитано не будет.

Отдельно стоит осветить вопрос пользовательских функций. Задача реализации высокого уровня абстракции предполагала генерацию программного кода по пользовательским функциям. Наряду с областью, над которой выполняются вычисления, пользователь задает систему уравнений, которую необходимо моделировать, Кроме того задаются граничные условия. Исходя из полученных данных формируется набор пользовательских функций, которые необходимы для работы приложения. В этот набор входят функции, которые задают начальное состояние области, а также функции, позволяющие рассчитать новое значение в определенной ячейке блока.

Каждый блок имеет специальную матрицу, которая совпадает по размеру с количеством ячеек, относящихся к блоку. В этой матрице лежать индексы функций, которые ответственны за перерасчет конкретно данной ячейки. На этапе подготовки выполняет компиляция пользовательских функций и формирования списка указателей на них. Индексы из этого списка и заносятся в матрицу. Функция для любой ячейки имеет унифицированную сигнатуру, что позволяет удобным образом осуществлять ее вызов, не задумываясь об особом статусе ячейки, если таковой имеет место быть.

2.3 Класс Interconnect

В процессе вычислений блокам требуется информация, которую необходимо получить от других составляющих области. Поэтому в контексте рассмотрения класса для передачи информации между блоками полезно заострить внимание на особенностях, связанных с их размещением. Очевидно, что существует три варианта размещения каждой пары блоков друг относительно друга:

1. блоки располагаются на одном вычислительном устройстве;
2. блоки размещаются на разных устройствах в пределах одной машины;
3. блоки находятся на разных машинах.

В случае расположения блоков на одном вычислительном устройстве проблема обеспечения доступа одного блока к данным, складируемой другим, фактически отсутствует, так как в данной ситуации нет ограничения для доступа к памяти.

Во втором случае существуют ограничения вызванные тем, что центральный процессор не имеет возможности напрямую читать из памяти видеокарты, ровно как и видеокарта не может осуществлять подоб-

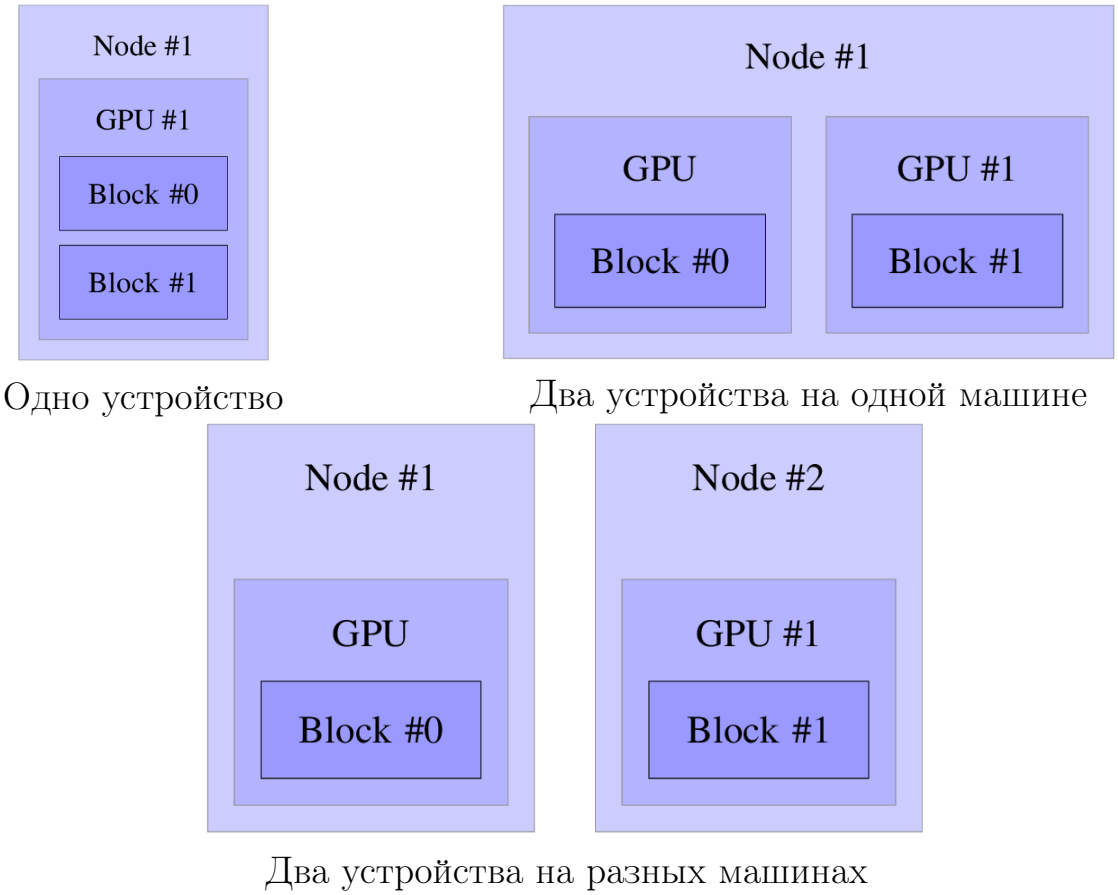

Рис. 3 Варианты расположения блоков

ные манипуляции с памятью центрального процессора. Данная проблема решается путем выделения памяти под данные, которые необходимы другим блокам, в специальной области, доступ к которой имеют оба вида вычислительных устройств.

Основные трудности связаны с передачей информации в третьем случае. Расположения блоков на разных машинах гарантирует невозможность прямого обращения к памяти и получения информации. Для решения данной проблемы и создан класс Interconnect. Объекты данного класса используются для пересылки данных между блоками, который располагаются на различных узлах вычислительного кластера. Каждый экземпляр хранит номер потока, являющегося источником информации, то есть потока, в которому реально приписан интересующий нас блок. Кроме того хранится номер потока приемника информации, такой потом тоже содержит реальный блок, которому необходима данная информация. Также каждый объект класса Interconnect знает длину массива передаваемой информации, это необходимо для корректной работы библиотеки MPI, которая используется для передачи данных.

Экземпляры данного класса создаются на каждую связь между блоками всеми потоками исполнения независимо от того относятся ли зависимые блоки к данным потокам. При вызове функции передачи/-

приема экземпляр класса выполнит либо отправку сообщения с данными, либо их прием в зависимости от номера потока, который вызвал данную функцию у себя, либо не будет выполнено ничего, если пересылка осуществляться не должна, например поток не имеет к ней вообще никакого отношения, не является ни источником, ни приемником, либо пересылка для данной связи не требуется - она относится к первым двум случаям.

2.4 Класс Domain

Главным управляющим классом приложения является класс Domain. Объект данного класса создается в единственном экземпляре каждым потоком. Предполагается, что на одном вычислительном узле не запускается более одного потока. Подобное ограничение введено с целью исключить следующую ситуацию. На одной машине одновременно выполняются два или более потоков. Каждый из них имеет некоторое количество блоков и вполне вероятно, что некоторые блоки, принадлежащие разным потокам, окажутся на одном вычислительном устройстве. Кроме того очевидно, что все блоки располагаются в пределах одной машины. Как было описано ранее в подобной ситуации передача данных между блоками средствами библиотеки MPI не требуется, так как ее можно организовать проще и. что еще более важно, она может осуществляться быстрее. Но в случае, который сейчас рассматривается, передача информации будет осуществлять именно через стороннюю библиотеку. С целью предотвращения подобной ситуации и было введено данное ограничение, которое фактически не мешает полноценному функционированию приложения.

Объекты класса Domain, как было сказано ранее, осуществляют управление вычислениями, каждый на своем потоке. В начале выполнения считывается файл, содержащий всю необходимую информацию о решаемой задаче: количество блоков, их размерность, размеры и координаты, а также данные о связях между блоками и другие сведения. Все это позволяет полностью сформировать набор переменных для дальнейшего решения (см. Рис. 4). Основную часть этих переменных, конечно, составляют блоки и связи между ними. Кроме того здесь же выполняется присвоение значений таким переменным как номер потока выполнения, количество потоков в целом, количество блоков и соединений. Также создается объект класса-наследника Solver, который будет предоставлять необходимую информацию о численном методе, применяющемся для решения задачи.

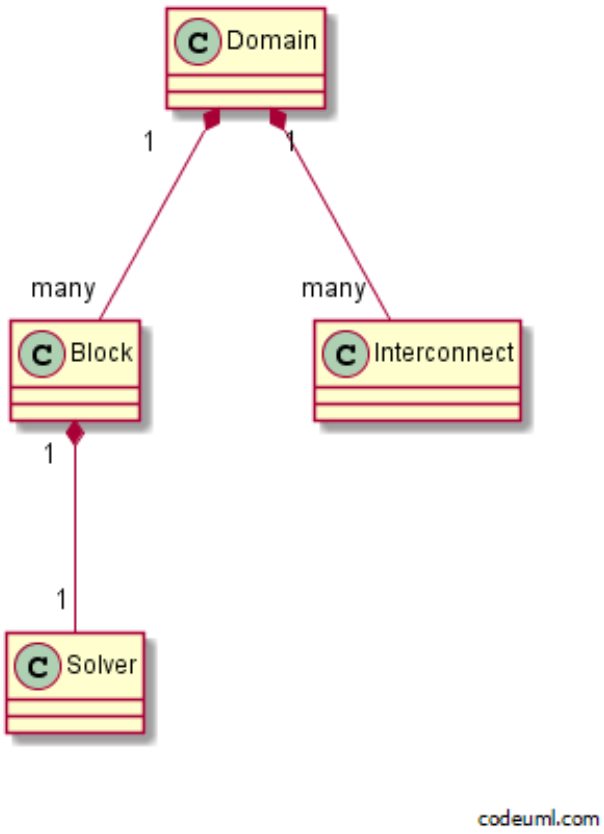

Рис. 4 Общая архитектура приложения

2.5 Схема работы приложения

Прежде чем переходить к рассмотрению схемы работы приложения (см. Рис. 5) необходимо ввести понятия центральных и граничных элементов блока.

Центральными элементами будем называть такие элементы, для вычисления новых значений которых не требуется данных от других блоков.

Граничные элементы - это элементы, для вычисления новых значений которых могут потребоваться данные от других блоков, следовательно не исключена возможность пересылки данных между различными вычислительными узлами кластера.

После того, как выполнена полная инициализация задачи можно начинать вычисления. На каждом потоке объект управляющего класса вызывает функцию compute(), которая начинает вычисления. До тех пор, пока не будет выполнено необходимое условие будут выполнять действия представленные на Рис. 5. Сначала каждый блок выполняет подготовку данных, то есть копирует часть своей информации, которая нужна его «соседям» - блокам, с которыми у него есть связь. Подобные действия выполняет каждый блок на каждой машине.

Важной опорной элементом алгоритма является точка синхрони-

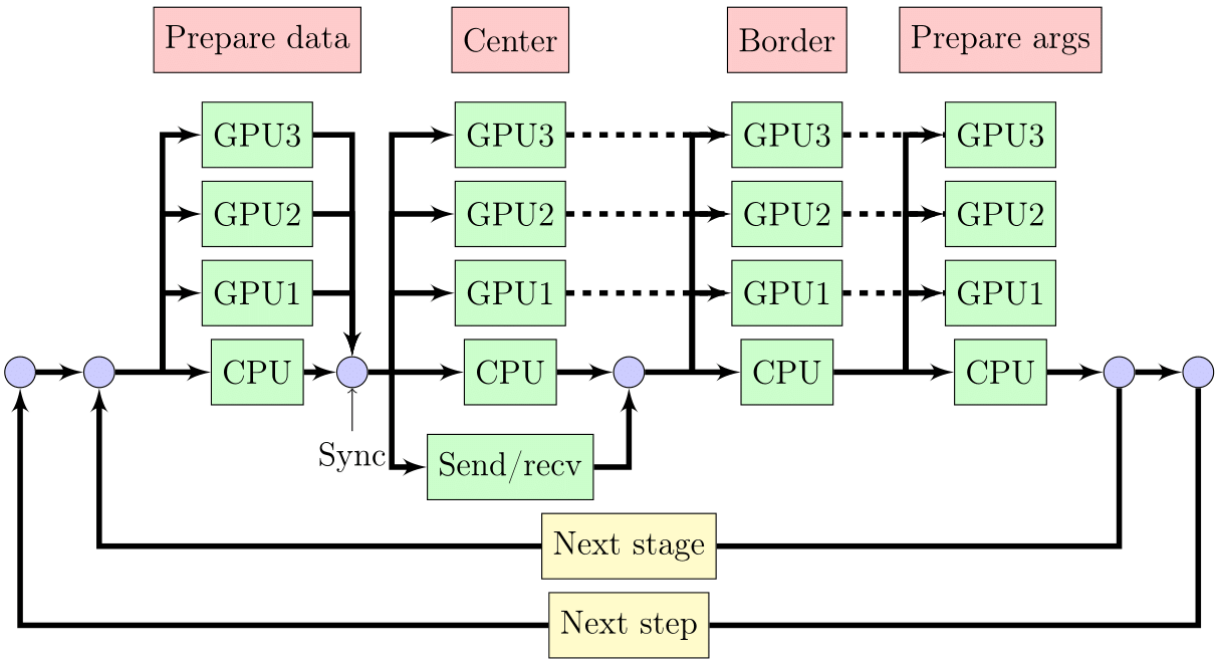

Рис. 5 Схема вычислений

зации. В этом момент времени можно быт уверенным, что все потоки пришли в эту точку и дождались друг друга. Это необходимо для того чтобы потоки не обгоняли друг друга. Например, область состоит из двух блоков, но один из них оказался значительно больше другого. В такой ситуации поток. работающий с блоком меньшего размера, будет выполнять вычисления значительно быстрее, будет предоставлять некорректные данные для пересылки и решение в конечном итоге окажется неверным. Для исключения подобной ситуации введена синхронизация потоков. Важно отметить, что инициализация новых вычислений или подготовки данных на центральном процессоре невозможно до завершения предыдущих действий, а каждый возов вычислений на ядре видеокарты всегда дожидается завершения предыдущего вызова на этой видеокарте.

После завершения этих действий мы имеем полностью подготовленные массивы, которые и будут пересылаться между вычислительным узлами кластера. Если пересылка не требуется, блоки расположены в пределах одной машины, то брать информацию будут именно из этих массивов напрямую, иначе инициализируется процесс пересылки данных. Данное действие требует значительных временных затрат, которые вполне разумно использовать для вычисления той части блоков, которым пересылаемые данные не могут потребоваться в Принсипе - центральных блоков. В связи с этим передача данных выполняется асинхронным образом, то есть после вызова команды «отправить» процесс, вызвавший ее, не дожидается завершения операции, а продолжает работу. Аналогичным образом поступает и поток, который информацию будет принимать, он просто сигнализирует о том, что готов это сделать,

но не дожидается окончания передачи данных. Пока информация пересылается и ведется расчет центральных элементов. Вычисления запускаются одновременно на центральном процессоре и видеокартах. Каждое устройство будет выполнять вычисления только с блоками, которые принадлежать этому устройству. Если окажется, что таких блоков нет, то устройство просто завершит выполнения данной части алгоритма.

После того, как процессор завершил расчет центральных элементов, относящихся к его блокам, он запускает ожидание окончания пересылки. Так как передача данных выполнялась асинхронным образом обязательно нужно убедиться, что действие завершено, прежде чем приступать к вычислению граничных элементов, новые значения которых могут зависеть от этих данных. Данные вычисления осуществляются аналогично вычислениям центральных элементов.

Далее необходимо выполнить подготовку аргументов, ведь если мы используем многостадийный метод нам вполне может потребоваться, например, сумма матриц, полученных на предыдущих стадиях, а это дополнительные вычисления, которые можно и нужно распараллелить. Затем мы либо отправляемся выполнять следующую стадию метода, либо переходим к следующей итерации - выполняем шаг по времени. И все это длится до тех пор, пока либо мы не достигнем заданных условий, либо пользователь не остановит процесс принудительно.

3 Компоненты программного комплекса

3.1 Общие сведения

Важной составляющей программного комплекса, ориентированного на конечного пользователя является пользовательский интерфейс. В его задачи входит предоставление удобных инструментов для управления задачами и вычислениями, а именно: создание и сохранение проектов, запуск, приостановка и завершение вычислений, визуализация полученных результатов.

Значимую роль в работе приложения играет программа предварительной обработки. Данная часть комплекса ответственна за преобразование пользовательских функций в код, который впоследствии будет скомпилирован и использован при расчетах. Кроме того данная часть приложения разбивает область задачи на блоки и осуществляет их распределение по вычислительным устройствам и узлам кластера, а также создает бинарный файл со всей необходимой информацией, который передается вычислительному ядру для дальнейшей работы и вычислений на его основе.

Наконец, параллельный фреймворк или вычислительное ядро занимается непосредственно распределенными вычислениями. В задачи этой части программного комплекса входит непосредственно осуществление параллельного выполнения расчетов на вычислительном кластере.

3.2 Параллельный фреймворк

Поговорим чуть подробнее о механизмах и инструментах, которые используются при распределенных вычислениях. Фактически распараллеливание проводится в два этапа:

1. разделение задачи на блоки - крупнозернистый параллелизм;
2. параллельные вычисление на устройстве - мелкозернистый параллелизм.

3.2.1 Крупнозернистый параллелизм

Крупнозернистый параллелизм осуществляется путем разбиения области на блоки и распределения блоков по вычислительным устройствам еще на этапе подготовки, о чем было изложено ранее. Данный подход позволяет использовать все вычислительные мощности имеющиеся в наличии и равномерно распределять нагрузку на устройства. На одном устройстве может располагаться несколько блоков одновременно, в таком случае их вычисление будет осуществляться в порядке очереди.

Немаловажным фактом является то, что в случае если количество блоков значительно (в несколько раз) превышает количество вычислительных устройств, то блоки, которые имеют общие границы разумно располагать на одном устройстве, либо в пределах одной машины, а блоки не имеющих таких границ - на разных, сохраняя равномерность распределения блоков по устройствам в целом. Такой подход позволяет увеличить скорость расчетов для небольших, по количеству используемых узлов, областей. Достигается это уменьшение времени, которое необходимо на пересылку данных между блоками между итерациями вычислений, так как если блоки расположены на одном вычислительном устройстве или одной машине, то пересылка не будет выполняться в Принсипе, потому что данные уже доступны блоку. Очевидно, что в общем случае невозможно распределить все блоки таким образом чтобы пересылок не было вообще, но такой подход позволяет минимизировать количество пересылаемой информации, что положительно сказывается на производительности.

3.2.2 Мелкозернистый параллелизм

Мелкозернистый параллелизм осуществляется непосредственно на вычислительных устройствах. В зависимости от типа устройства будет использована библиотека OpenMP, для работы на центральном процессоре, и CUDA, в случае работы на видеокарте. Увеличение производительности достигается путем параллельного вычисления частей блока.

OpenMP реализует параллельные вычисления с помощью многопоточности, в которой «главный» (master) поток создает набор подчиненных (slave) потоков и задача распределяется между ними. Предполагается, что потоки выполняются параллельно на машине с несколькими процессорами (количество процессоров не обязательно должно быть больше или равно количеству потоков). Количество создаваемых потоков может регулироваться как самой программой при помощи вызова библиотечных процедур, так и извне, при помощи переменных окружения.

Данная библиотека работает в системах с общей памятью, поэтому каждая нить имеет доступ ко всем данным блока. Например, при расчете двумерного случая прямоугольник «разрезается» на полосы, обработка которых осуществляется одновременно. Библиотека OpenMP самостоятельно определяет количество таких полос и их ширину.

В качестве аналога данной библиотеки можно было использовать стандартные потоки языка C++, но подобный подход значительно усложнил бы разработку и увеличил вероятность ошибки. OpenMP обладает всеми необходимыми инструментами для реализации многопоточности на центральном процессоре в рамках одной машине и достаточно проста в использовании.

Как уже сообщалось ранее, при осуществлении вычислений на видеокарте используется CUDA - (Compute Unified Device Architecture) программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы Nvidia.

CUDA SDK позволяет программистам реализовывать на специальном упрощённом диалекте языка программирования Си алгоритмы, выполнимые на графических процессорах Nvidia, и включать специальные функции в текст программы на Си. Архитектура CUDA даёт разработчику возможность по своему усмотрению организовывать доступ к набору инструкций графического ускорителя и управлять его памятью.

При обработке блока на видеокарте каждая ячейка блока обрабатывается отдельным потоком. Каждый такой поток имеет необходимый доступ к данным.

В качестве альтернативы CUDA можно рассмотреть OpenCL, данный фреймворк для работы с графическими процессорами также предоставляет необходимые инструменты. Но так как CUDA разработана компанией Nvidia специально для своих видеокарт и, как следствие, она демонстрирует лучшие показатели производительности именно на видеокартах данной торговой марки. На вычислительный кластеры ставят подобные видеокарты, поэтому в качестве инструмента работы с видеокартами была выбрана именно CUDA.

3.2.3 Передача информации между узлами

Пересылка данных между разными машинами (вычислительными узлами) осуществляется с помощью библиотеки MPI. Message Passing Interface (MPI, интерфейс передачи сообщений) - программный интерфейс (API) для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу. MPI является наиболее распространённым стандартом интерфейса обмена данными в параллельном программировании, существуют его реализации для большого числа компьютерных платформ. Используется при разработке программ для кластеров и суперкомпьютеров. Основным средством коммуникации между процессами в MPI является передача сообщений друг другу. В стандарте MPI описан интерфейс передачи сообщений, который должен поддерживаться как на платформе, так и в приложениях пользователя. В первую очередь MPI ориентирован на системы с распределенной памятью, то есть когда затраты на передачу данных велики.

4 Тестирование

33333

5 Список литературы

Список литературы

[1] *Круглов В. В., Борисов В. В.* Искусственные нейронные сети. Теория и практика. М.: Горячая линия - Телеком, 2001. — 382 с.

[2] *Кохонен Т.* Ассоциативные запоминающие устройства. М.: Мир, 1982. — 384 с.

[3] *Колесов А. Ю., Колесов Ю. С.* Релаксационные колебания в математических моделях экологии. М., 1993 (Тр. МИАН. Т. 199).

[4] *Глызин С. Д., Колесов А. Ю., Розов Н. Х.* Релаксационные автоколебания в нейронных системах. I // Дифференциальные уравнения. 2011. Т. 47, № 7. С. 919 – 932.

[5] *Глызин С. Д., Колесов А. Ю., Розов Н. Х.* Релаксационные автоколебания в нейронных системах. II // Дифференциальные уравнения. 2011. Т. 47, № 12. С. 1675 – 1692.

[6] *Глызин С. Д., Колесов А. Ю., Розов Н. Х.* Релаксационные автоколебания в нейронных системах. III // Дифференциальные уравнения. 2012. Т. 48, № 2. С. 155 – 170.

[7] *Кащенко С. А., Майоров В. В.* Модели волновой памяти. М.: Либроком, 2009. — 288 с.

[8] *Глызин С. Д., Колесов А. Ю., Розов Н. Х.* Дискретные автоволны в нейронных системах. Журнал вычислительной математики и математической физики. 2012. Т. 52, № 5. С. 840–858.

[9] *Глызин С. Д., Колесов А. Ю., Розов Н. Х.* Моделирование эффекта взрыва в нейронных системах. // Математические заметки. 2013. Т. 93, № 5. С. 682–699.

[10] *Боресков А. В., Харламов А. А.* Основы работы с технологией CUDA. М.: ДМК Пресс, 2010. — 232 с.

[11] *Антонов А. С.* Параллельное программирование с использованием OpenMP. М.: Изд-во МГУ, 2009. - 77 с.

[12] *Сандерс Дж., Кэндрот Э.* Технология CUDA в примерах. Введение в программирование графических процессоров. М.: ДМК Пресс, 2013. — 232 с.

[13] *Горелов Ю. Н.* Численные методы решения решения обыкновенных дифференциальных уравнений (метод Рунге-Кутта). Самара: Самарский университет, 2006. — 48 с.

[14] *Ивановский Л. И., Самсонов С. О.* Фазовые перестройки одной двумерной динамической системы с импульсным воздействием. // Модел. и анализ информ. систем, 2014. Т. 21, №6. С. 179 – 181.

[15] *Гукенхеймер Дж., Холмс Ф.* Нелинейные колебания, динамические системы и бифуркации векторных полей. Москва-Ижевск.: Институт компьютерных исследований, 2002. – 560 с.

[16] *Арнольд В. И.* Дополнительные главы теории обыкновенных дифференциальных уравнений. М.: Наука, 1978. — 304 с.