Введение

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-Принца. Подобные вычисления удобно автоматизировать, чтобы в дальнейшем иметь возможность быстро производить расчеты.

1 Постановка задачи

Требовалось создать программный комплекс, с помощью которого можно было бы автоматизировать моделирования конечно–разностных диффузионных задач на гибридных вычислительных кластерах.

Задача реакции диффузии имеет следующий вид.

$$\dot{u} = D \triangle u + F(u);$$

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

Точечная задача $\dot{u} = f(u)$, «определяющая реакцию», имеет устойчивый период $\dot{u}_0(f)$, который также !!!(слово) однородное решение задачи.

Для получения результата может быть применен один из следующих методов численного решения дифференциальных уравнений, а именно: метод Эйлера, четырехстадийный метод Рунге-Кутты, семистадийный метод Дормана-Принца.

Разрабатываемый программный комплекс должен поддерживать работу с уравнениями и системами уравнений, которые были бы распределены в одномерных, двумерных и трехмерных областях. Каждая из таких областей может быть представлена неким набором блоков. Для одномерного случая этими блоками являются отрезки, в случае плоскости - прямоугольники, параллелепипеды - если область трехмерна !!!примеры. Каждый из блоков характеризуется координатами в пространстве и размерами, а также информацией о своих границах. Границы блока могут состоять из нескольких частей, каждая из которых представлена !!! Нейман Дирихле , либо является местом соединения с другим блоком.

Кроме того, необходима поддержка современного оборудования с неоднородной архитектурой и иерархической организацией памяти. К системам с неоднородной архитектурой относятся, например, вычислительные кластеры, которые используют для расчетов мощности центрального процессора и видеокарт. Иерархическая организация памяти предполагает малые объемы высокоскоростной памяти и большие медленной. Подобный подход к реализации памяти подразумевает экономное расходование ресурсов в процессе выполнения с целью повешения общей производительности за счет ускоренного обращения к данным.

Программный комплекс должен иметь возможность по окончании вычислений сохранять полученный результат, а также выполнять сохранение текущего состояния в процессе выполнения для последующего возобновления вычислений с любого из сохраненных состояний. Кроме того, необходимо обладать инструментами для запуска расчетов до определенного момента времени с данным шагом или выполнять заданное количество итераций.

В качестве языка программирования используемого для реализации поставленных задач был выбран C++ как наиболее подходящий для

данных целей. Этот язык обладает необходимыми инструментами для удобной разработки подобного приложения, примером того может послужить механизм наследования. Кроме того, язык C++ дает возможность низкоуровневой работы с памятью, что позволять производить оптимизацию программного кода на высоком уровне. Также, для этого языка разработана библиотека MPI, которая осуществляет передачу данных между разными машинами, что было необходимо в рамках данного проекта.

2 Архитектура приложения

2.1 Класс Solver и его потомки

Класс Solver и его потомки обеспечивают работу различных численных методов, которые могут применяться для решения поставленной задачи. Сам класс Solver является абстрактным классом, который определяет интерфейс, необходимый для дальнейшей работы, содержит две переменных: матрица текущего состояния (mState) и количество элементов в этой матрице (mCount). !!!(см. Рис. 1).

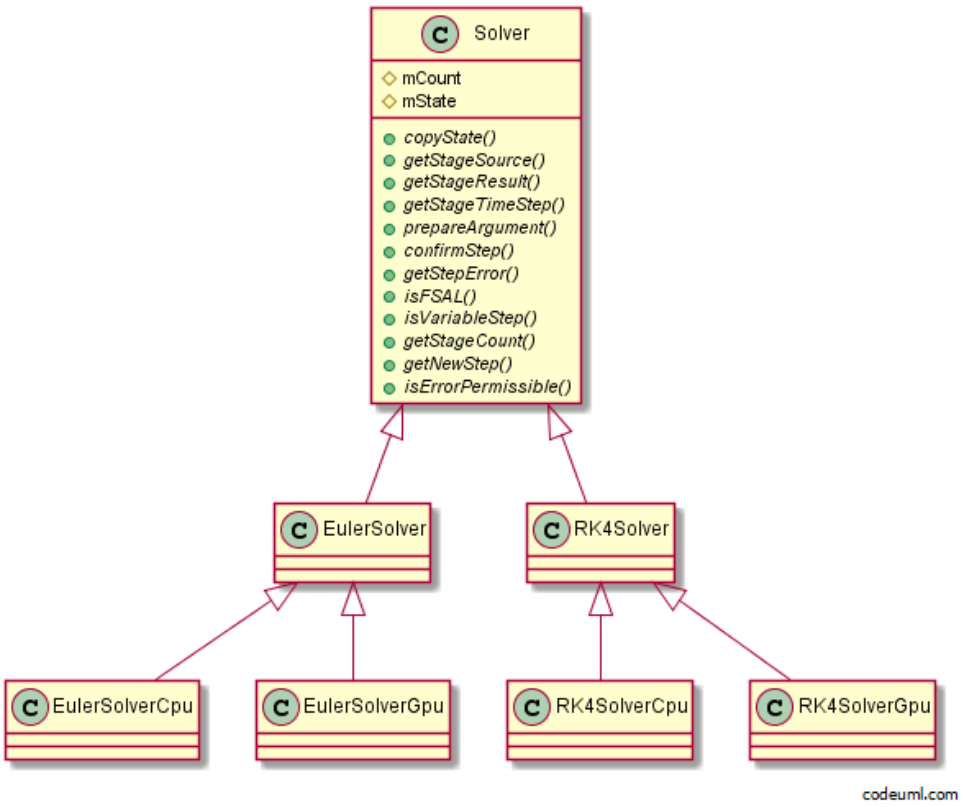

Рис. 1 Иерархия наследования для методов численного решения.

EulerSolver, RK4Solver, DP45Solver являются потомками класса Solver и описывают основную информацию о реализуемых методах. Например, является ли данный численный метод методом с изменяемым шагом или требуется ли ему предварительная подготовка перед первой итерацией, сообщает количество итераций которые необходимы для данного метода (для метода Эйлера это значение равно единице). Кроме того, они хранят набор констант необходимых конкретно для данного метода, например для метода Рунге-Кутты это числа, используемые при вычислении нового состояния. !!! информация о методе Все эти классы в полной мере реализуют все необходимые функции, описанные в классе предке, поэтому объект каждого из них можно создать. Именно они используются в качестве хранителей информации о выбранном методе решения задачи.

Классы EulerSolverCpu, EulerSolverGpu и другие уже являются потомками второго уровня, они наследуются от EulerSolver, RK4Solver,

DP45Solver соответственно и предназначены для работы на центральном процессоре и видеокарте соответственно. Именно эти классы в полной мере реализуют выбранный численный метод.

Рассмотрим подробнее работу данных классов на примере RK4SolverCpu. Фактически данный класс является большим хранилищем данных. Реализованный метод Рунге-Кутты предполагает четыре стадии, поэтому предок этого класса имеет четыре переменных для временного хранения данных (например mTempStore1, mTempStore2). Именно в них записывается информация о выделенных под хранилища ресурсах. В процессе работы от объекта данного класса можно получить источник и приемник информации в зависимости от стадии.

2.2 Класс Block и его потомки

Класс Block и его потомки обеспечивают непосредственно расчеты, производимые с данными. Сам класс Block является абстрактным и, как следствие, его создание невозможно. Он описывает интерфейс для блоков, который позволит полноценно работать с реальными блоками - потомками данного класса. Кроме того, он имеет переменные для хранения всей необходимой информации о блоке, например, его положение в пространстве, размеры, тип и номер устройства, на котором блок будет выполняться. !!!(см. Рис. 2).

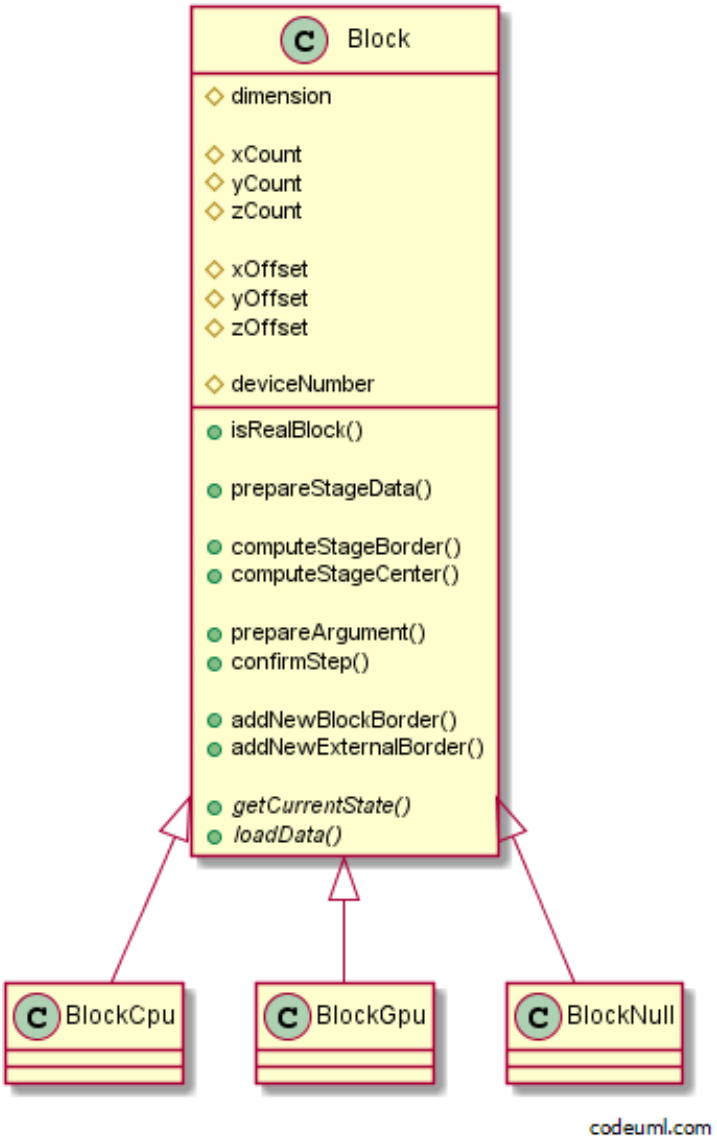

Рис. 2 Иерархия наследования для блоков.

Класс BlockCpu является наследником класса Block, предназначенным для работы на центральном процессоре. Он является реальным блоком, который хранит информацию и осуществляет ее обработку. Для осуществления распараллеливания используется библиотека OpenMP.

Класс BlockGpu является наследником класса Block, предназначенным для работы на видеокарте. Он также является реальным блоком,

но в отличии от класса BlockCpu всю информацию он хранит в памяти видеокарты, расчеты выполняются также на ней, для этого используется специальный язык CUDA.

Класс BlockNull единственный класс-наследник Block, который не является реальным блоком. Он нужен для того чтобы обозначить, что часть области, относящаяся к данному блоку на самом деле обрабатывается другим потоком и на другой машине. Но он позволяет выполнить все функции присущие блоку, но в данном случае ничего не будет выполнено. Кроме того, объект данного класса содержит основную информацию о блоке: его размеры и местоположение в пространстве. корректной работы.

2.3 Класс Domain

Главным управляющим классом приложения является класс Domain. Объект этого класса создается в единственном экземпляре каждым из потоков выполнения. Данный класс обладает широким набором возможностей. Чтение загрузочной информации из файла, во время которой фактически происходит инициализация всех переменных приложения. Вывод результатов вычисления как по завершению работы, так и во время выполнения. Также этот класс осуществляет сохранение текущего состояния вычислений и загрузку любого ранее сохраненного состояния, относящегося к той же задаче. !!!(см. Рис. 3).

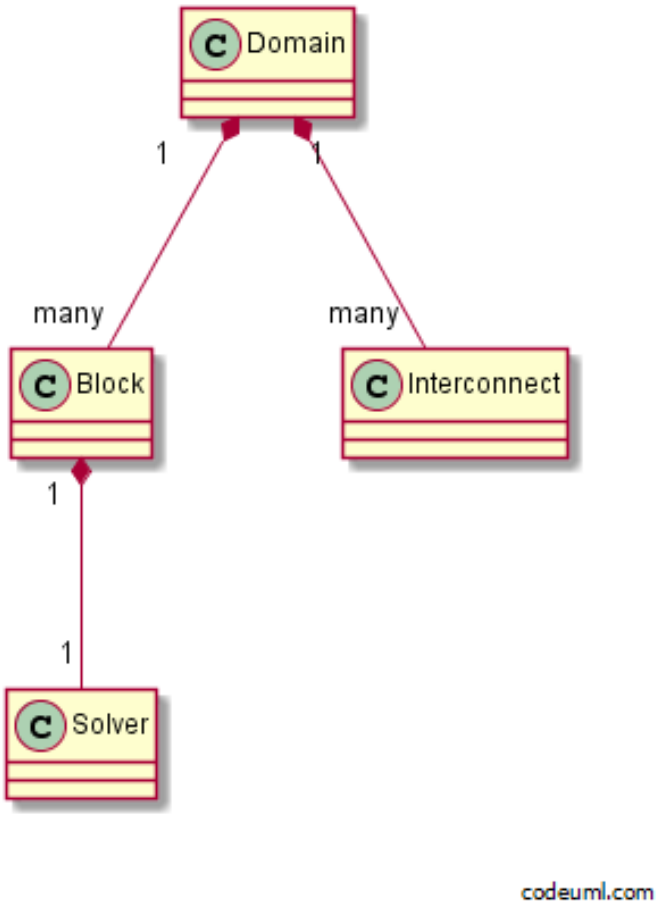

Рис. 3 Общая архитектура приложения.

2.4 Класс Interconnect

В процессе вычислений блокам требуется информация, которую необходимо получить от других составляющих области. Поэтому в контесте рассмотрния 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 вычислителном 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, складирумой 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 подобные манипуляции с памятью центрального процессора. Данная проблема решается путем выделяения памяти под данные, которые необходимы другим блокам, в специальной области, доступ к которой имеют оба вида вычислительных устройств.

Основные трудности связаны с передачай 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.

3 Компоненты программного комплекса

Приложение состоит из нескольких частей. Каждая из них отвечает за определенную часть работы, либо подготовительной, либо относящейся непосредственно к самим вычислениям, либо реализующей взаимодействие с пользователем.

Пользовательский интерфейс позволяет создавать и модифицировать проекты. Кроме того, он дает возможность управлять текущими вычислениями, а именно запускать, прекращать, приостанавливать, а также выполнять сохранение текущего состояния и его загрузку.

Часть предварительной обработки выполняет подготовку исходных данных. В ней происходит обработка геометрии и свойств области, производится определение размерности, расположения и размера блоков, граничных условий. Здесь же выполняется распределение блоков по вычислительным устройствам, формируется библиотека пользовательских функций.

Параллельный фреймворк и алгоритмы обработки данных составляют основу приложения и занимаются непосредственно самими вычислениями. К алгоритмам обработки данных относятся метод Эйлера, явные схемы Рунге-Кутты, методы Дормана-принца. Стоит отметить, что присутствует возможность расширения списка доступных методов решения.

4 Параллельный фреймворк

4.1 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 ком-

панией 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.

4.2 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 принципе, 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.

4.3 Пересылки информации

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.