Операторная библиотека для решения задач математической физики на трёхмерных локально-адаптивных сетках с использованием графических ускорителей CUDA

М.М. Краснов

ИПМ им. М.В. Келдыша РАН

Описывается библиотека операторов для работы с сеточными функциями, определёнными на трёхмерных локально-адаптивных сетках. Библиотека спроектирована так, чтобы детали ее реализации были скрыты от пользователя, что позволяет эффективно реализовать ее на машинах с различной архитектурой (параллельных, гибридных и т.п.). При создании библиотеки ставилось несколько целей: приближение внешнего вида программ к формулам в теоретических работах, относительная простота использования, в том числе при работе на графических платах с архитектурой CUDA, вычислительная эффективность.

1. Введение

В задачах математического моделирования широко используются разнообразные сетки и сеточные функции – величины, определённые на этих сетках. Для численного решения таких задач с этими сеточными функциями делаются определённые преобразования. В литературе эти преобразования часто описываются уравнениями, содержащими операторы, такие, например, как операторы Лапласа, градиента, дивергенции. Кроме того, уравнения могут содержать линейные комбинации и композиции операторов. При описании векторных полей (таких как поля скоростей или сил) сами уравнения также могут быть векторными и содержать, например, скалярные и векторные произведения векторов. Уравнения могут оперировать с размерными величинами. В результате запись уравнений выглядит компактно и в то же время ясно и прозрачно. Приведём пару примеров. Первый – это уравнение сохранения количества движения для ньютоновской несжимаемой жидкости без действия массовых сил:

$$\frac{\partial \rho \boldsymbol{U}}{\partial t} + \nabla \cdot (\rho \boldsymbol{U}\boldsymbol{U}) - \nabla \cdot (\mu \nabla \boldsymbol{U}) = -\nabla p.$$

Здесь ρ – плотность, p – давление, **U** – вектор скорости, μ – вязкость, ∇ - оператор набла (вектор), **UU** –скалярное произведение векторов, ∇**U** – дивергенция скорости, а ∇p – градиент плотности.

Второй пример – это уравнение Эйлера в гидродинамике идеальной жидкости в консервативной векторной форме:

$\partial \mathbf{m}/\partial t + \partial \mathbf{f}_x/\partial x + \partial \mathbf{f}_y/\partial y + \partial \mathbf{f}_z/\partial z = 0,$

где

$\mathbf{m} = (\rho, \rho u, \rho v, \rho w, E)^T,$

$\mathbf{f}_x = (\rho u, p+\rho u^2, \rho uv, \rho uw, u(E+p))^T,$

$\mathbf{f}_y = (\rho v, \rho vu, p+\rho v^2, \rho vw, v(E+p))^T,$

$\mathbf{f}_z = (\rho w, \rho wu, \rho wv, p+\rho w^2, w(E+p))^T.$

Здесь ρ – это плотность жидкости, u, v, w – компоненты скорости, p – давление, E – полная энергия единицы объёма жидкости, $E=\rho e+\rho(u^2+v^2+w^2)/2$, где e – внутренняя энергия единицы массы жидкости. Величины $\mathbf{f}_x$, $\mathbf{f}_y$, $\mathbf{f}_z$ задают потоки жидкости в направлениях трёх осей. Каждая из этих векторных величин состоит из пяти компонент, имеющих разные размерности. Уравнение Эйлера в данной форме задаёт законы сохранения массы, момента импульса и энергии.

Несмотря на свою относительную сложность, оба уравнения записываются весьма компактно. При записи тех же самых формул в программах они превращаются в многострочные операторы с вложенными циклами и сложными выражениями, разобраться в которых может быть непростой задачей.

Существуют системы программирования, позволяющие записывать подобные выражения в упрощённом виде с использованием шаблонов (stencils), например, система Blitz++ (см. [1]). Однако, эта система не решает другой важной задачи – облегчение переноса программ на но-

вые параллельные архитектуры, например, на графические ускорители (например, CUDA, см. [2]) и новые процессоры Intel Xeon Phi (см. [3]). Существует ряд систем, облегчающий подобный перенос. Среди них можно отметить разработанную в ИПМ им. М.В. Келдыша систему DVM (см. [4]), системы OpenMP (см. [5]) и OpenACC (см. [6]). С использованием этих и им подобных систем можно относительно легко преобразовать существующую последовательную программу, написанную на языках C/C++ или Fortran для работы на графических ускорителях или процессорах Intel Xeon Phi. Для этого в программе перед участками кода (как правило, циклами), которые должны исполняться параллельно, нужно вставить специальные инструкции компилятору, которые позволят ему сгенерировать параллельный код. В языках C и C++ это, как правило, делается с помощью т.н. прагм (#pragma dvm, #pragma omp или #pragma acc). В программах на языке Fortran используются псевдокомментарии (например, !$acc). Если скомпилировать такую программу обычным компилятором, который «не понимает» этих инструкций, то он их просто проигнорирует и сгенерирует последовательный код. Всё это позволяет относительно легко перенести существующую программу на новые архитектуры. Если программа компилируется для гетерогенной архитектуры с отдельным ускорителем (например, с графическим ускорителем, таким, как CUDA или с процессором Intel Xeon Phi), то нужно иметь в виду, что у отдельного ускорителя своя оперативная память, и он может обрабатывать только данные, размещённые в ней. Поэтому компилятор, генерирующий параллельный код, как правило, хранит две копии данных – в памяти основной (host) системы и в памяти ускорителя и при необходимости синхронизует их (копирует в ту или другую сторону). Синхронизацией данных (когда и что копировать) также можно управлять с помощью специальных инструкций (прагм или псевдокомментариев).

Однако, всё это никак не сокращает размеров формул в программах.

Описываемая библиотека стоит на стыке этих двух направлений. С одной стороны, она позволяет использовать сокращённую запись выражений с использованием операторов, аналогичных математическим, а с другой стороны, позволяет путём простой перекомпиляции генерировать код, исполняемый на различных архитектурах.

2. Библиотека gridmath

Для того, чтобы внешний вид уравнений в программах на языке C++ можно было приблизить к внешнему виду тех же уравнений в математической литературе, написана операторная библиотека gridmath (см. [7]). В ней сеточная функция – это класс, для которого переопределены стандартные математические операции. Кроме того, оператор (например, оператор Лапласа) превращён из абстрактного математического понятия в конкретный программный объект. Например, уравнение

$$u = \Delta(\mu v)$$

с использованием библиотеки gridmath может быть записано, например, так: `u=laplas(mu*v);`

В правой части оператора присваивания может стоять выражение любой сложности, результатом этого выражения будет сеточный вычислитель, который запомнит всю цепочку вычислений, не производя их. В момент присваивания вычислителя сеточной функции произойдёт запуск вычислений, и для всех точек сеточной функции в левой части оператора присваивания будут произведены вычисления в соответствие с запомненной цепочкой.

Библиотека gridmath решает ещё одну важную задачу. Как известно, на многих современных суперкомпьютерах (кластерах) стоят графические ускорители CUDA или ускорители Intel Xeon Phi. Например, в списке top500 самых мощных суперкомпьютеров (см [8]) за ноябрь 2014 года на первом месте стоит суперкомпьютер Tianhe-2 (Китай - NUDT), основанный на процессорах Intel Xeon Phi 31S1P, а на втором месте – Titan (США - Cray Inc), основанный на графических ускорителях NVIDIA K20x. Многие Российские суперкомпьютеры, например, самый мощный Российский суперкомпьютер «Ломоносов» (см. [9]) и вычислительный кластер К-100 в ИПМ им. М.В. Келдыша РАН (см. [10]) также содержат графические ускорители. Например, на К-100 на каждом из 64-х узлов, помимо двух основных процессоров Intel Xeon X5670 стоят по три платы NVIDIA Fermi C2050 (по 448 GPU и 2,8 Gb памяти на каждом). Эффективное же использование этих весьма внушительных вычислительных мощностей затруднено, т.к. требует

освоения большого объёма новой и непривычной информации о методах программирования на них. Одной из целей при создании библиотеки ставилось облегчение использования подобных гетерогенных систем. В частности, все результаты, изложенные в данной статье, получены на кластере К-100.

Как показано выше, основной запуск вычислений в библиотеке производится в операторе присваивания вычислителя сеточной функции. То, как этот оператор реализован, полностью скрыто от пользователя и может производиться как последовательно, так и параллельно для точек сеточной функции в левой части. Порядок обхода точек – внутреннее дело библиотеки и пользователь не должен закладываться на тот или иной порядок обхода точек. Всё это даёт библиотеке существенную гибкость. В частности, если компиляция осуществляется компилятором nvcc (CUDA), то, во-первых, все данные сеточных функций располагаются в памяти графического ускорителя, а не host-системы, и, во-вторых, оператор присваивания реализован путём вызова ядра (kernel) в графическом процессоре и осуществляется параллельно. В последовательном случае обход точек в случае трёхмерной регулярной сетки осуществляется в трёх вложенных циклах по трём осям. При работе на CUDA следует обратить внимание на то, что, в отличие от упомянутых во введении систем, нацеленных на распараллеливание циклов (DVM, OpenACC), библиотека на хранит копию данных в памяти основного (host) процессора. Все данные сеточных функций хранятся только в памяти CUDA. Для хранения данных сеточных функций при компиляции «обычным» компилятором используется класс std::vector из стандартной библиотеки C++, а при компиляции с помощью nvcc используется класс thrust::device_vector из CUDA SDK.

Таким образом, при использовании библиотеки gridmath перенос программы на графические ускорители CUDA является относительно простой задачей и не требует от пользователя библиотеки (а им чаще всего является математик, специалист по численным методам) глубокого знания методов программирования для CUDA. В простейшем случае программу достаточно просто перекомпилировать. Немного сложнее получается при использовании MPI. В этом случае надо учитывать, что данные сеточных функций расположены в памяти CUDA, а MPI работает с памятью host-системы, поэтому нужно ещё добавить копирование данных из CUDA в host-систему и обратно.

Перенос последовательной программы, написанной с использованием библиотеки gridmath, на ускоритель Intel Xeon Phi также является простой задачей. Для этого используется OpenMP (см. [7]). Для того, чтобы запустить программу на этом ускорителе в native режиме, нужно скомпилировать программу обычным компилятором от Intel (например, icpc или mpiicpc) с опциями –mic и –openmp. В этом случае, как уже говорилось выше, обход точек осуществляется в трёх вложенных циклах по трём осям. Но перед этими циклами стоит прагма:

```
#pragma omp parallel for private(i,j,k)
```

Если не указана опция –openmp, то компилятор игнорирует эту прагму и получается последовательный код. Если же эта опция указана, то получается параллельный код на общей памяти (распараллеливание цикла), использующий все имеющиеся вычислительные возможности системы. Опцию –openmp можно, естественно, использовать и на обычных процессорах. При этом получится параллельный код, использующий все ядра обычного процессора (или процессоров, если их несколько).

3. Перенос библиотеки на локально-адаптивные сетки

Первоначальная версия библиотеки написана для регулярных двух и трёхмерных прямоугольных сеток. С её помощью успешно перенесён на CUDA многосеточный метод (см. [11]). Проведён сравнительный анализ эффективности многосеточного метода на различных архитектурах вычислительных систем. Результаты изложены в работе [12].

Затем встал вопрос о переносе библиотеки на нерегулярные сетки. Основная идея такого переноса состоит в том, чтобы некоторым образом упорядочить ячейки сетки, а затем, при присваивании выражения сеточной функции, делать их одномерный обход, последовательный (в одном цикле) или параллельный (для CUDA или OpenMP). При переносе библиотеки на другие типы сеток по сути, создаётся новая библиотека, а не новая версия существующей библиотеки.

В результате создаётся целое семейство библиотек, отличающихся типом сетки, но с общей внутренней идеологией и с многочисленными общими частями.

Важное отличие в случае нерегулярных сеток состоит в том, что сетка явно отделяется от сеточной функции. В простейшем случае в этом не было необходимости, т.к. регулярная прямоугольная сетка устроена очень просто и её вынос в отдельный объект не имеет смысла. В случае же нерегулярной сетки это не так. Есть целый ряд нетривиальных операций с сеткой, таких как определение соседей ячейки, вычисление площадей граней, объёмов ячеек, векторов нормалей к граням, не имеющих непосредственного отношения к данным, которые хранит сеточная функция в каждой ячейке. Поэтому вынос сетки в отдельный самостоятельный объект в этом случае оправдан. Сетка не хранит данных, не имеющих к ней непосредственного отношения, этим занимается сеточная функция. Сеточная функция определяется на сетке, т.е. объект-сетка передаётся сеточной функции как параметр создания и может впоследствии использоваться. Сеточная функция по желанию программиста может быть определена в ячейках сетки или в её узлах (вершинах). Сеточная функция может хранить для каждой ячейки сетки одно или несколько скалярных или векторных значений. Например, если нужно для каждой ячейки сетки хранить некий трёхмерный вектор (например, вектор скорости), то нет необходимости заводить три сеточных функции для хранения каждой из координат. Можно завести одну сеточную функцию и хранить в ней для каждой ячейки сетки сразу все три координаты. На одной и той же сетке можно определить любое количество сеточных функций. Одна из них может, например, хранить скорость (все три координаты), другая – давление, третья – плотность, четвёртая – температуру и т.д. Библиотека позволяет хранить и размерные величины, при этом размерность переменной – это параметр времени компиляции и никак не влияет ни на объём данных, занимаемой переменной в оперативной памяти, ни на скорость обработки данных. С другой стороны, использование размерных величин, во-первых, повышает наглядность программы, т.к. уже из определения переменной (её размерности) ясно, что в ней содержится, и, во-вторых, позволяет часть программных ошибок (таких, как сложение и вычитание величин, имеющих разную размерность) отловить уже на стадии компиляции.

В качестве первого шага при переносе библиотеки на произвольные нерегулярные сетки было принято решение перенести библиотеку на локально-адаптивные сетки.

Локально адаптивные сетки формируются на основе регулярных равномерных сеток. Физический шаг исходной регулярной сетки вдоль каждой из осей постоянный, но вдоль разных осей может быть разным. Сетка строится самой библиотекой или может быть загружена из файла. После построения сетку можно сохранить в файл. Формат файла бинарный и нестандартный. Возможен экспорт сетки в файл формата TecPlot (одно зонный и многозонный), а также в формат ig и METIS (см [13]).

Сетка строится следующим образом. Внутрь исходной регулярной сетки помещается тело произвольной формы, например, шар или цилиндр так, чтобы был простой алгоритм, позволяющий для любой точки внутри исходной физической области ответить на вопрос, находится ли эта точка внутри тела или снаружи. Затем строятся несколько (например, 5) уровней локально-адаптивной сетки, причём каждый следующий уровень строится на основе предыдущего. Исходная регулярная сетка рассматривается как нулевой уровень. До построения сетки формируются глобальные (общие для всех уровней) массивы физических координат по каждому из трёх направлений. Первоначальное заполнение этих массивов простое – значение по i-му индексу равно шагу по данному направлению, помноженному на этот индекс. Эти массивы динамические – по мере построения сетки они расширяются. При делении ребра пополам может возникнуть новая точка с координатой, которой ещё нет в массиве координат. В этом случае координата с заданным значением добавляется в конец массива координат. Это, в свою очередь, означает, что массивы координат не упорядочены по возрастанию (кроме начала этих массивов), значения координат могут идти в произвольном порядке.

Алгоритм построений следующего (начиная с первого) уровня следующий. Изначально для строящегося уровня список рёбер, граней и ячеек пустой. Рассматриваются все рёбра сетки предыдущего уровня. У каждого ребра есть две вершины. Если оказывается так, что одна из вершин ребра лежит внутри тела, а вторая – снаружи, то это ребро делится пополам, и получившиеся два коротких ребра добавляются к списку рёбер строящегося уровня, а само ребро с предыдущего уровня помечается как поделенное. В описании каждого из объектов сетки (вер-

шины, грани или ячейки) хранится индекс соответствующего объекта со следующего уровня. Если этот индекс равен -1 (недопустимое значение для индекса), то это означает, что данный объект сетки не поделён, иначе в нём хранится индекс первого из объектов, из которых состоит данный объект. После того, как все рёбра рассмотрены (и некоторые из них поделены), рассматриваются все грани сетки предыдущего уровня. Если оказывается, что для очередной грани хотя бы одно из четырёх рёбер было поделено на предыдущем шаге, то оставшиеся не поделёнными рёбра этой грани также делятся пополам, а сама грань делится на 4 маленьких равных грани, которые добавляются к списку граней строящегося уровня, а грань предыдущего уровня помечается как поделенная. Затем аналогичная процедура повторяется для ячеек. Если хотя бы одна из шести граней ячейки была поделена на предыдущем шаге, то оставшиеся не поделёнными грани также делятся (вместе со своими рёбрами), а сама ячейка делится на 8 маленьких равных ячеек, которые добавляются к списку ячеек строящегося уровня, а ячейка предыдущего уровня помечается как поделенная.

После построения очередного уровня в том случае, если номер построенного уровня больше 1, осуществляется сглаживание предыдущих уровней в обратном порядке (вначале сглаживается пред-предыдущий уровень, затем пред-пред-предыдущий и т.д.). Сглаживание делается для того, чтобы с двух сторон одной грани не было ячеек «через уровень», т.е. это должны быть или ячейки одного уровня, или соседних уровней. Алгоритм сглаживания следующий. Просматриваются все ячейки сглаживаемого уровня. Если какая-то ячейка не была поделена, а хотя бы одна из её граней поделена дважды (поделена грань и хотя бы одна из её дочерних граней), то ячейка делится (вместе со всеми своими гранями и рёбрами).

Формат внутреннего представления информации обо всех элементах сетки (рёбрах, гранях, ячейках) одинаковый и содержит следующие поля:

```
int child;
int boundary_type;
int map[6];
```

Поле `child` задаёт индекс первого дочернего объектов следующего уровня или -1.

Поле `boundary_type` содержит признак граничного элемента. 0 – внутренний элемент, 1 – граничный с граничным условием типа Дирихле, 2 - граничный с граничным условием типа Неймана.

В поле `map` хранится информация об объектах того же уровня, составляющих данный элемент (вершинах для ребра, рёбер для граней, граней для ячеек). Для рёбер хранятся координаты (глобальные индексы) двух вершин. Для граней гранятся индексы 4 рёбер, окружающих данную грань. Последние два элемента массива map в этом случае не используются. Для ячеек хранятся индексы 6 граней, окружающих данную ячейку.

Для сеточных функций, определённых на локально-адаптивных сетках, определены такие же операции, как и для сеточных функций на регулярных сетках (сложение, вычитание, умножение и пр. со скалярами и вычисляемыми объектами) и также можно создавать операторы.

4. Решение параболических уравнений

Рассмотрим параболическое уравнение

$$\frac{\partial u}{\partial t} = div(\kappa\ grad\ u) - a\ u + f$$

Будем искать решение этого уравнения в трёхмерной области $\Omega = [x_0 ; x_1] \times [y_0 ; y_1] \times [z_0 ; z_1]$ (прямоугольный параллелепипед) при условии, что на всех гранях расчётной области задан нулевой поток $\kappa \frac{\partial u}{\partial \vec{n}} = 0$, где $\vec{n}$ – внешняя нормаль к границе расчётной области. Функции $\kappa(r)$, $f(r)$, $a(r) \geq 0$ являются заданными, $r=(t,x,y,z) \in G = [t_0 ; T] \times \Omega$, $[t_0 ; T]$ – заданный интервал времени.

Предполагается, что расчётная область может содержать зоны со сложной геометрией, отличной от прямоугольных параллелепипедов. Эти зоны могут иметь различные физические свойства, на границах зон тензор диффузии κ может быть разрывным и его скачок на поверхно-

сти разрыва может быть большим. Тогда в окрестности поверхности разрыва решение может иметь большие градиенты, для сеточного воспроизведения которых обычно требуются подробные сетки. Но построение подробной сетки на всей расчётной области приведёт к значительному росту вычислительной сложности задачи. Выборочное измельчение элементов сетки может оказаться компромиссным решением. Локально-адаптивные сетки как раз и помогают решить такую задачу.

Рассмотрим уравнение теплопроводности в следующей форме:

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(a_{xx}\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(a_{yy}\frac{\partial u}{\partial y}\right) + \frac{\partial}{\partial z}\left(a_{zz}\frac{\partial u}{\partial z}\right) - a \cdot u + f$$

Для получения дискретных уравнений используется метод конечных объёмов. Сеточная функция задана в центрах ячеек, и ячейка консервативности совпадает с ячейкой сетки:

$$\frac{\hat{u}_i - u_i}{\tau} = \frac{1}{V_i}\sum_{face=1}^{6}\Delta_{face}^{i} S_{face}^{i} - a_i\hat{u}_i + f_i.$$

Индекс i нумерует сеточную функцию в центре ячейки сетки,

$\hat{u}_i$ – искомое значение функции на верхнем временном слое $\hat{t} = t + \tau$,

S_{face}^{i} – площадь боковой грани ячейки,

Δ_{face}^{i} – удельный диффузионный поток через боковую грань ячейки в направлении внешней нормали, V_i – объём i-ой ячейки.

Для параболических уравнений универсальной является явная схема, но она используется крайне редко из-за известного ограничения на шаг по времени. В данном примере используется явно-итерационная схема ЛИ-М, которая специально разработана для решения параболических уравнений (см. [14]).

Данная задача решена ранее Феодоритовой О.Б. без использования библиотеки gridmath. Интересно сравнить по быстродействию старую последовательную версию с последовательной же версией, но с использованием библиотеки, а также с CUDA версией (также с использованием библиотеки). Как уже говорилось выше, CUDA версия программы получается практически простой перекомпиляцией компилятором nvcc. Решалась также распределённая задача с обменом по MPI. Декомпозиция сетки на несколько узлов была сделана Головченко Е.Н. с помощью разработанного ею метода (см. [15]).

Для решения задачи разработан разностный оператор диффузии. На вход этому оператору подаются значения тензора диффузии a_{xx}, a_{yy}, a_{zz} и функции a во всех ячейках сетки. На основе этих коэффициентов и с использованием структуры сетки в конструкторе оператора для всех ячеек рассчитываются разностные коэффициенты. В зависимости от того, с ячейками какого уровня (предыдущего, текущего или следующего) граничит данная ячейка, шаблон разностной схемы для неё может содержать от 7 до 25 точек.

Расчёты проведены на сетках 32 × 32 × 32 и 64 × 64 × 64 с 3, 4 и 5 уровнями адаптации. Решается нестационарное уравнение теплопроводности в кубе $[0 ; 1]^3$ на промежутке времени $t \in [0 ; 1]$. Внутрь исходного куба помещается цилиндр вдоль оси z радиуса 0,2 с осью вдоль прямой (x=0,5; y=0,5). Внутри и вне цилиндра коэффициент диффузии к полагается одинаковым и равным 1. Параметр a полагается постоянным и равным нулю. Возьмём точное решение в виде $u_{exact}(t;x,y,z) = e^{-t}\cos\pi x \cos\pi y \cos\pi z$, тогда правая часть f(t; x,y,z) имеет вид

$f(t; x,y,z) = (3\pi^2-1)\cdot u(t; x,y,z)$.

Шаг по времени взят фиксированный. Тогда число явных шагов схемы ЛИ-М прямо пропорционально числу шагов по одному направлению (см. [14]), то есть при переходе с сетки 32 × 32 × 32 на сетку 64 × 64 × 64в стандартном варианте без адаптации время счёта возрастает в 16 раз. Ниже в таблицах приведено время счёта (в секундах) на указанных сетках при разном числе уровней адаптации.

Таблица 1. Время счёта при трёх уровнях адаптации.

Основная сетка	«Старая» программа	gridmath	gridmath +CUDA
32 × 32 × 32	4.99	1.92	1.21
64 × 64 × 64	90.31	39.38	21.69

Таблица 2. Время счёта при четырёх уровнязх адаптации.

Основная сетка	«Старая» программа	Gridmath	gridmath +CUDA
16 × 16 × 16	4.24	1.64	1.05
32 × 32 × 32	65.83	28.97	15.90
64 × 64 × 64	1282.60	490.74	258.97

Таблица 3. Время счёта при Пяти уровнях адаптации.

Основная сетка	«Старая» программа	gridmath	gridmath +MPI	gridmath +CUDA	gridmath+ CUDA+MPI
16 × 16 × 16	73.33	28.60	6.45	15.59	13.29
32 × 32 × 32	1140.23	454.34		237.04	

Последовательная версия программы примерно в 2,5 раза превосходит по производительности «старую» программу, написанную также на языке C++, но без использования данной библиотеки. Перекомпиляция для CUDA даёт ускорение ещё примерно в два раза. Также производилась декомпозиция сетки на несколько областей, каждая область обсчитывалась на отдельном узле, обмен граничными ячейками производился по MPI.

Не очень большое ускорение на CUDA объясняется спецификой задачи. CUDA даёт существенное ускорение, если удаётся сделать так, чтобы потоки с последовательными номерами обращались к соседним ячейкам памяти, что в случае локально-адаптивных сеток не выполняется. Этим же объясняется слабое ускорение при декомпозиции сеток. Основное время уходит на построение массива граничных ячеек, которые собираются по всей области. Также, в отличие от последовательной версии, появляется этап копирования данных из памяти CUDA в память host процессора.

5. Заключение

Написана операторная библиотека для решения задач математической физики на локально-адаптивных сетках, позволяющая компактно записывать в программах математические формулы, связывающие сеточные функции, определённые на таких сетках. Операторы в этой библиотеке превращены из абстрактных математических понятий в конкретные программные объекты. Перенос программ, использующих данную библиотеку, на графические ускорители CUDA, осуществляется простой перекомпиляцией.

С помощью библиотеки решена задача теплопроводности, заданная в виде параболического уравнения. Задача решается внутри прямоугольного параллелепипеда, в который помещено тело, на границе которого коэффициент диффузии может претерпевать сильный разрыв. Для повышения точности решения первоначально равномерная сетка измельчается в окрестности границы внутреннего тела. Перенос программы на CUDA (перекомпиляцией) даёт ускорение примерно в два раза по сравнению с последовательной версией.

Автор выражает ей искреннюю благодарность Феодоритовой О.Б. за консультации при переносе написанной ею программы, решающей параболическое уравнение на локально-адаптивных сетках, на данную библиотеку.

Литература

1. The Blitz++ library. Official website. URL: http://blitz.sourceforge.net/
2. NVidia CUDA.URL: http://www.nvidia.com/object/cuda_home_new.html
3. Intel Xeon Phi. URL:http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html
4. DVM система. URL: http://www.keldysh.ru/dvm/
5. OpenMP. URL: http://openmp.org

6. OpenACC URL: http://www.openacc-standard.org/
7. Краснов М.М. Операторная библиотека для решения трёхмерных сеточных задач математической физики с использованием графических плат с архитектурой CUDA.// Математическое моделирование, 2015, т. 27, № 3, с. 109-120.
8. TOP500 Supercomputer Sites. URL: http://www.top500.org
9. Суперкомпьютер "Ломоносов". URL: http://parallel.ru/cluster/lomonosov.html
10. Гибридный вычислительный кластер К-100
URL: http://www.kiam.ru/MVS/resourses/k100.html
11. Жуков В.Т., Новикова Н.Д., Феодоритова О.Б. Параллельный многосеточный метод для разностных эллиптических уравнений. Часть I. Основные элементы алгоритма.// Препринты ИПМ им. М.В.Келдыша. 2012. № 30. 32 с.
12. Жуков В.Т., Краснов М.М., Новикова Н.Д., Феодоритова О.Б. Сравнение эффективности многосеточного метода на современных вычислительных архитектурах.// Программирование, 2015, № 1, с. 21-31.
13. METIS – Family of Graph and Hypergraph Partitioning Software
URL: http://glaros.dtc.umn.edu/gkhome/views/metis
14. Жуков В.Т. О явных методах численного интегрирования для параболических уравнений.// Математическое моделирование, 2010, т. 22, № 10, с. 127-158.
15. Головченко Е.Н. Параллельный пакет декомпозиции больших сеток.// Математическое моделирование. 2011. Т. 23, № 10. с. 3-18

Operator library for solving of mathematical physics problems on locally adaptive grids using CUDA

Michael Krasnov

Keywords: Locally adaptive grids, Grid functions, Grid operators, Heterogeneous systems, CUDA

This paper describes a library of operators to work on grid functions defined on three dimentional locally adaptive grids. The functions is designed so that the details of its implementation are hidden from the user, allowing effective implementation on machines with different implementation (hybrid, parallel, etc.), including CUDA. The library creation raises several goals: the approximation of the appearance of the program to the theoretical formulas, the relative ease of use, including the work on the graphics card with CUDA architecture, computational efficiency.