ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 20 за 2015 г.

Краснов М.М.

Оптимальный параллельный алгоритм обхода точек гиперплоскости фронта вычислений и его сравнение с другими итерационными методами решения сеточных уравнений

Рекомендуемая форма библиографической ссылки: Краснов М.М. Оптимальный параллельный алгоритм обхода точек гиперплоскости фронта вычислений и его сравнение с другими итерационными методами решения сеточных уравнений // Препринты ИПМ им. М.В.Келдыша. 2015. № 20. 20 с. URL: http://library.keldysh.ru/preprint.asp?id=2015-20

Ордена Ленина
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В. Келдыша
Российской академии наук

М.М. Краснов

Оптимальный параллельный алгоритм обхода точек гиперплоскости фронта вычислений и его сравнение с другими итерационными методами решения сеточных уравнений

Москва — 2015

УДК 519.6

М.М. Краснов

Email: kmm@kiam.ru

Оптимальный параллельный алгоритм обхода точек гиперплоскости фронта вычислений и его сравнение с другими итерационными методами решения сеточных уравнений.

В данной работе предлагается алгоритм обхода точек гиперплоскости фронта вычислений, оптимальный для распараллеливания, в том числе для графических ускорителей CUDA. Во второй части работы сравниваются с точки зрения эффективности различные методы решения системы линейных уравнений на примере задачи теплопроводности (уравнение Пуассона). Кроме явной (метод Якоби) и неявной (методы Гаусса-Зейделя) схем, сравниваются также двухслойный метод простой итерации, Чебышевские двухслойный и трёхслойный методы и многосеточный метод. Методы Гаусса-Зейделя рассматриваются в простом виде и с добавлением последовательной верхней релаксации. Все алгоритмы были реализованы в последовательном варианте и в параллельном для CUDA. Для переноса программ на CUDA использовалась библиотека gridmath.

Ключевые слова: сеточные уравнения, фронт вычислений, Чебышевские итерационные методы, многосеточный метод, CUDA

M.M. Krasnov

Optimal parallel algorithm of calculation of points of a computational front hyperplane and its comparison with other iteration methods of solving of grid equations.

This paper offers an algorithm of calculation of points of a computational front hyper plane, optimal for parallelization, including CUDA GPUs. In the second part of the paper, different methods of solving of systems of linear equations are compared from the viewpoint of efficiency on example of Poisson's equation (heat conductivity task). Besides explicit (Jacobi method) and implicit (Gauss-Seidel methods) schemes, the paper also considers two-level simple iteration method, two and three-level Chebyshev methods and multi grid method. Gauss-Seidel methods were also considered together with successive over relaxation (SOR) method. All algorithms were implemented in serial version and in parallel version on CUDA GPUs. To simplify transformation on CUDA, gridmath library was utilized.

Key words: grid equations, computational front, Chebyshev iteration methods, multi grid method, CUDA

Работа выполнена при финансовой поддержке Программы № 18 фундаментальных исследований Президиума РАН.

1. Введение

При численном решении многих задач математической физики возникают (явно или неявно) большие системы линейных уравнений с тысячами и даже миллионами переменных. Прямое решение подобных систем неэффективно или даже невозможно. Поэтому для решения подобных систем уравнений часто применяются различные итерационные методы. Эти методы могут сильно различаться по своей эффективности. Ещё один немаловажный аспект – эффективность распараллеливания на современных вычислительных архитектурах, таких как NVidia CUDA или Intel Xeon Phi, на которых число одновременно работающих потоков на общей памяти может составлять десятки, сотни и даже тысячи. В итерационных методах на каждом последующем шаге итерации получается всё более точное решение исходной системы уравнений. Как правило, алгоритм завершает свою работу при достижении требуемой точности решения, т.е., например, когда норма невязки Au-b уменьшается в заданное число раз (для линейного уравнения Au=b, где A – заданная матрица, b – заданный вектор, u – вектор неизвестных, который требуется найти при заданных граничных условиях). Время работы того или иного алгоритма зависит, во-первых, от скорости сходимости метода, т.е. от того, как сильно уменьшается норма невязки за одну итерацию, и, во-вторых, от времени выполнения одной итерации. Время выполнения одной итерации можно рассматривать как один из критериев вычислительной сложности метода. При этом более сложные методы (т.е. методы, выполняющие одну итерацию за большее время) могут давать лучшую сходимость (сходиться с требуемой точностью за меньшее, а иногда за существенно меньшее число итераций) и в итоге работать быстрее. Одной из целей данной работы является сравнение по вычислительной эффективности разных методов решения систем линейных уравнений, причём рассматриваются как последовательные реализации методов, так и параллельные (на общей памяти), прежде всего на NVidia CUDA.

В итерационных методах решение на данном шаге итерации вычисляется на основе найденных решений на предыдущих шагах. Если используется только один предыдущий шаг, то такие методы называются двухслойными, а если два – то трёхслойными. Начальное приближение, как правило, может быть произвольным. В трёхслойных методах решение на первой итерации считается с помощью двухслойного метода. Двухслойные методы, в свою очередь, можно разделить на явные и неявные. В явных методах для вычисления значений на текущем шаге используются значения с предыдущего шага. При этом вычисления значений переменных на очередном шаге могут производиться параллельно для всех переменных. В неявных методах (например, методах типа Гаусса-Зейделя), помимо значений на предыдущем шаге, используются также ранее вычисленные значения на текущем шаге. Это значит, что на очередной итерации значения всех переменных считаются не одновременно, а они делятся на группы. Значения переменных из самой первой группы вычисляются на основе значений переменных с предыдущего шага, значения переменных из

второй группы вычисляются с использованием как значений переменных с предыдущего шага, так и значений переменных из первой группы с текущего шага и т.д. Значения переменных из одной группы могут быть вычислены параллельно, т.к. они не зависят друг от друга. Переход от группы к группе должен быть последовательным. Число групп – от двух и выше. Явные методы особенно хороши для распараллеливания. Неявные методы, как правило, сходятся быстрее (за меньшее число итераций) аналогичных явных методов при сравнимом времени выполнения одной итерации.

Среди неявных методов типа Гаусса-Зейделя для двух- и трёхмерных прямоугольных областей (регулярные равномерные сетки) при решении уравнения Пуассона можно выделить два – метод шахматной раскраски (называемый иногда также методом красно-чёрной раскраски) и метод гиперплоскости. Особенностью уравнения Пуассона, содержащего оператор Лапласа, является то, что в дискретном случае значение в данной точке зависит только от значений в соседних точках вдоль осей. В двухмерном случае таких точек четыре, в трёхмерном – шесть. В методе шахматной раскраски все точки делятся в шахматном порядке на две группы. Назовём одну группу, например, чёрной, а вторую – белой. Значения в точках из первой группы (чёрной) вычисляются на основе значений в соседних (белых) точках с предыдущей итерации, а значения в точках из второй группы (белых) – на основе только что вычисленных значений в точках из первой (чёрной) группы. Если сравнивать метод шахматной раскраски с явным методом Якоби, то, согласно теории, асимптотически (при стремлении точности решения к нулю) он должен сходиться за вдвое меньшее число итераций при примерно равном времени выполнения одной итерации, то есть должен быть примерно вдвое быстрее.

В методе гиперплоскости значения из соседних точек с большими индексами берутся с предыдущего шага, а с меньшими – с текущего. В этом случае все точки исходного параллелепипеда разбиваются на группы, в каждой из которых все точки лежат в трёхмерном случае на плоскости, а в двухмерном – на прямой, проходящей под углом 45° ко всем осям. Самая первая плоскость в трёхмерном случае проходит через точку с координатами (индексами) (0, 0, 0), вторая – через точку (1, 1, 1) и т.д. При решении других задач (не Пуассона) могут возникать плоскости (в многомерном случае – гиперплоскости), проходящие под другими углами к осям координат и начинающиеся из других вершин параллелепипеда. Все эти плоскости параллельны друг другу и определяют т.н. «фронт вычислений», который движется от некоторой начальной точки (вершины параллелепипеда) до некоторой конечной. Как уже говорилось выше, движение этой гиперплоскости должно осуществляться последовательно, а вычисления в точках гиперплоскости на каждом шаге её движения могут производиться параллельно. Однако встаёт вопрос – как найти все точки исходной прямоугольной области, лежащие на гиперплоскости фронта вычислений на данном шаге её движения? Форма области пересечения гиперплоскости с рассматриваемой прямоугольной многомерной областью на различных шагах движения гиперплоскости может быть весьма замысловатой,

и обход всех точек этой области пересечения становится нетривиальной задачей. Тем более если речь идёт о параллельном обходе этих точек. Большое внимание решению задач с сильно связанными компонентами, приводящими к появлению гиперплоскости фронта вычислений и обходу точек этой гиперплоскости, уделено в языке Норма (см., например, [1]). Автор данной статьи также рассматривал ранее данный вопрос (см. [2]).

Прежде всего нужно определить, что называется «параллельным обходом». Будем считать, что любой алгоритм, осуществляющий такой параллельный обход, состоит из внешнего последовательного цикла движения гиперплоскости фронта вычислений и одного внутреннего цикла по точкам этой гиперплоскости на данном шаге K. Во внешнем последовательном цикле индекс цикла (шаг гиперплоскости) K пробегает последовательно значения от некоторого начального значения $K_{нач}$ до конечного значения $K_{кон}$. Далее внутри этого цикла в зависимости от параметров гиперплоскости, размеров исходного параллелепипеда и текущего значения шага K определяется (оценивается сверху) общее количество N_K точек гиперплоскости, которые нужно обойти во внутреннем цикле. Эти точки некоторым образом упорядочиваются, т.е. каждой из них присваивается некоторый номер, по которому, зная шаг K, можно однозначно определить координаты этой точки. Затем запускается (последовательно или параллельно) внутренний цикл. На каждой итерации внутреннего цикла нам дополнительно к существующей информации (границы области, параметры гиперплоскости и шаг K последовательного движения гиперплоскости) становится известен текущий индекс внутреннего цикла. По всей этой информации определяются координаты точки гиперплоскости. При этом может оказаться, что точка гиперплоскости лежит внутри рассматриваемой области или выходит за её пределы. В первом случае в точке проводятся все требуемые вычисления, во втором случае точка игнорируется.

Одним из видимых недостатков этого общего описания схемы алгоритма параллельного обхода точек гиперплоскости является избыточность вычислений, т.е. допускается, что вместо точного определения количества точек будет сделана только некоторая оценка сверху этого количества. Но эта оценка может быть очень грубой или довольно точной в зависимости от конкретного алгоритма, и если, несмотря на эту избыточность, скорость работы алгоритма будет высокой, такой подход может оказаться приемлемым. Может, например, оказаться, что один алгоритм, точно вычисляющий число точек, на это вычисление затратит больше процессорного времени, чем другой алгоритм с избыточностью на просмотр лишних точек.

Главное преимущество алгоритмов, построенных по такой схеме, состоит в том, что они легко переносятся на современные параллельные архитектуры, в том числе CUDA, а также OpenMP, что даёт возможность запуска параллельных программ на процессоре Intel Xeon Phi.

Алгоритм, предложенный автором в статье [2], в целом соответствует только что описанной общей схеме. Основное отличие состоит в том, что количество точек гиперплоскости, которые нужно обойти на текущем шаге

движения гиперплоскости, определяется не внутри последовательного цикла, а перед ним и не зависит от шага K (число рассматриваемых точек одинаково для всех шагов). Это приводит к тому, что при значениях шага K, близких к крайним ($K_{нач}$ и $K_{кон}$), избыточность становится чрезмерной. Например, в крайних положениях гиперплоскости обрабатывается (как правило) только одна (угловая) точка области независимо от размера области. Описываемый в настоящей статье алгоритм даёт меньшую избыточность вычислений. Более того, в распространённом случае, когда рассматриваемая область представляет собой квадрат (одинаковые размеры во всех направлениях), а гиперплоскость проходит под углом 45° ко всем осям (все параметры гиперплоскости по модулю равны единице), число точек вычисляется точно. Однако сам алгоритм вычисления количества точек достаточно сложный и затратный по времени, поэтому для оценки его эффективности требуется проведение численных экспериментов.

Во второй части статьи описываются результаты численных экспериментов, в которых сравнивается эффективность различных алгоритмов решения систем линейных уравнений на примере задачи теплопроводности (решение стационарного уравнения Пуассона). Данная задача решается различными методами (Якоби, Гаусса-Зейделя, двухслойный метод простой итерации, Чебышевские двухслойный и трёхслойный методы, многосеточный метод и др.), причём каждый метод реализован как в последовательном варианте, так и в параллельном на графических ускорителях CUDA. Интересно не только сравнить эффективность различных методов, но и то, как хорошо они ускоряются на CUDA.

2. Постановка задачи

Пусть исходная n-мерная область, на которой проводятся вычисления, задана неравенствами:

$$M_j \le i_j \le N_j, j = 1 \dots n. \qquad (1)$$

Пусть также известны параметры гиперплоскости фронта вычислений, т.е. заданы целые числа p_j, j=1…n, причём хотя бы одно из них отлично от нуля, а сама гиперплоскость задаётся формулой:

$$\sum_{j=1}^{n} p_j i_j = K. \qquad (2)$$

Целое число K задаёт шаг последовательного движения гиперплоскости и пробегает значения от $K_{нач}$ до $K_{кон}$. Эти границы определяются следующим образом (см. [1]):

$$K_{нач} = \sum_{j,p_j>0} p_j M_j + \sum_{j,p_j<0} p_j N_j ; \quad (3)$$

$$K_{кон} = \sum_{j,p_j>0} p_j N_j + \sum_{j,p_j<0} p_j M_j. \quad (4)$$

Если некоторые из p_j равны нулю, то соответствующие им индексы i_j не входят в уравнение гиперплоскости (точнее, входят с нулевым коэффициентом). В этом случае гиперплоскость параллельна соответствующим осям координат, а индексы на каждом шаге движения гиперплоскости пробегают все допустимые значения $M_j \le i_j \le N_j$. При определении порядка

обхода точек гиперплоскости эти индексы можно не учитывать, а при обходе точек для каждой допустимой комбинации индексов с ненулевыми коэффициентами p_j дополнительно пробегать по всем допустимым значениям всех индексов, для которых $p_j=0$. Для простоты дальнейшего изложения будем считать, что все коэффициенты p_j отличны от нуля, или, другими словами, будем считать, что n – число ненулевых коэффициентов, а индексы перенумерованы так, чтобы индексы с ненулевыми коэффициентами шли первыми.

Если ввести обозначение

$$S_j = \begin{cases} M_j, \text{если } p_j > 0; \\ N_j, \text{если } p_j < 0; \end{cases} \quad (5)$$

тогда

$$K_{\text{нач}} = \sum_{j=1}^{n} p_j S_j. \quad (3')$$

Уравнение гиперплоскости можно переписать в виде:

$$\sum_{j=1}^{n} p_j (i_j - S_j) = K', \quad (6)$$

где $K'=K - K_{\text{нач}}$. Шаг K’ пробегает последовательно значения от нуля до $K'_{\text{кон}}$ ($0 \le K' \le K'_{\text{кон}}$), где

$$K'_{\text{кон}} = K_{\text{кон}} - K_{\text{нач}} = \sum_{j=1}^{n} |p_j| (N_j - M_j). \quad (7)$$

Числа S_j, j=1…n задают координаты начальной вершины при движении гиперплоскости.

С уравнением гиперплоскости в виде (6) нам в дальнейшем работать будет проще, т.к. в нём K’ – это неотрицательный порядковый номер (считая от нуля) шага движения гиперплоскости, и все слагаемые в сумме в левой части уравнения (6) также неотрицательные. Штрих у шага K для простоты в дальнейшем будем опускать.

Введём обозначение

$$a_j = p_j (i_j - S_j). \quad (8)$$

Для a_j справедливо неравенство

$$0 \le a_j \le \min(A_j, K), \quad (9)$$

где

$$A_j = |p_j| (N_j - M_j). \quad (10)$$

Кроме того, a_j должно быть кратно p_j

$$a_j \equiv 0 (mod\ |p_j|). \quad (11)$$

Тогда уравнение (6) можно переписать в виде

$$\sum_{j=1}^{n} a_j = K. \quad (12)$$

Координата точки i_j по известному a_j определяется так:

$$i_j = S_j + \frac{a_j}{p_j}. \quad (13)$$

Делить на p_j можно, т.к. выше мы условились, что все p_j отличны от нуля. Таким образом, задача поиска точек, принадлежащих гиперплоскости фронта вычислений на неотрицательном шаге K сводится к поиску всех комбинаций n неотрицательных целых чисел a_j, удовлетворяющих условиям (9), (10), (11), сумма которых равна K. В начале (перед исполнением внутреннего

параллельного цикла) требуется определить (или оценить сверху) общее количество таких комбинаций, затем для каждой итерации внутреннего цикла нужно по порядковому номеру итерации определить все значения a_j, затем проверить, что все эти числа удовлетворяют условиям (9), (10), (11). Если это так, то по известным значениям a_j вычислить по формуле (13) значения индексов i_j (определить координаты точки) и провести вычисления в ней. Если же хотя бы одно из условий не выполнено, то пропустить данную итерацию внутреннего цикла. Если есть нулевые коэффициенты p_j (с индексом $j > n$), то при проведении вычислений нужно пройти по всем допустимым значениям соответствующих индексов и провести вычисления во всех получившихся точках.

3. Оптимальный алгоритм обхода точек

Как уже говорилось выше, задача поиска точек гиперплоскости фронта вычислений, лежащих в рассматриваемой прямоугольной многомерной области на определённом шаге K движения гиперплоскости, сводится к вычислению всех комбинаций неотрицательных чисел a_j, удовлетворяющих условиям (9), (10), (11), сумма которых равна K. Очевидно, что можно написать переборный алгоритм, который в n вложенных циклах переберёт все возможные комбинации чисел a_j, удовлетворяющие условиям (9), (10), (11), и если сумма чисел в очередной комбинации равна K, то сохранит эту комбинацию. Затем можно посмотреть, сколько комбинаций чисел мы сохранили, это и будет точный результат. Во вложенном цикле алгоритма по индексу этого цикла легко восстановить комбинацию чисел из сохранённого массива комбинаций. Однако представляется, что такой алгоритм будет весьма затратным, прежде всего по времени. Затраты по памяти не очень существенны. Например, если исходная область – трёхмерный куб с размером ребра 10^3, то размер массива комбинаций будет порядка 10^7 байт, что при нынешних размерах оперативной памяти (10^{10} байт и выше) некритично. Тем не менее, если затраты по времени окажутся разумными, то такой алгоритм имеет право на существование. Его преимущество, во-первых, в простоте реализации и, во-вторых, в том, что он на каждом шаге даёт не верхнюю оценку числа точек, а точное значение.

Цель настоящей работы – построить более оптимальный по времени и памяти алгоритм, дающий разумную (не слишком большую) верхнюю оценку числа точек гиперплоскости.

Для начала заменим условие (9) на менее жёсткое:

$$0 \le a_j \le \min(A_{\max}, K), \qquad (14)$$

где

$$A_{max} = max_j(A_j). \qquad (15)$$

Кроме того, при оценке числа точек не будем использовать условие (11) (делимость на p_j). Условия (9) и (11) будут проверяться уже во вложенном цикле для определения допустимости полученных чисел a_j. Заметим, что такое ослабление условий несущественно в распространённом случае, когда исходная область – куб (все A_j равны между собой), а гиперплоскость проходит под

углом 45° ко всем осям (все p_j по модулю равны единице). В этом случае описываемый ниже алгоритм даст точный результат, а не верхнюю оценку.

Такое ослабление условий позволяет сделать следующее утверждение: если некоторая комбинация чисел является допустимой, то допустимой является любая комбинация чисел, являющаяся перестановкой чисел из данной комбинации. Без ослабления условий такого утверждения сделать нельзя: в результате перестановки для каких-то чисел a_j одно или оба из условий (9) и (11) могут перестать выполняться.

В качестве базовой будем искать комбинацию чисел, упорядоченных по невозрастанию, т.е. для $j>1$ $a_j \le a_{j-1}$. Тогда в любой перестановке чисел из найденной комбинации и отличающейся от неё этот принцип будет нарушен. Это, в свою очередь, означает, что такая перестановка не может оказаться среди других найденных комбинаций. Определим число различных перестановок из n чисел. Пусть среди них есть ровно k различных чисел, $1 \le k \le n$. Пусть каждое из этих k чисел встречается b_l раз, $1 \le l \le k$, $\sum_{l=1}^{k} b_l = n$. Тогда число N различных перестановок этих чисел равно

$$N = \frac{n!}{\prod_{l=1}^{k} b_l!}. \qquad (16)$$

Перейдём к описанию алгоритма.

Пусть имеется массив a из n чисел (a[0..n-1]). На вход алгоритму подаются числа K и A_{max}.

Шаг 0. Инициализация.

- Заполняем массив a нулевыми значениями.
- Обозначим $B=\min(A_{max}, K)$.
- Общее число точек C=0.

Шаг 1. Основной. Распределить число K начиная с числа B в массиве a начиная с ячейки 0. Результат – число точек.

Распределение числа по массиву делает основная рекурсивная функция. Назовём её distribute. Прототип этой функции на языке C/C++ следующий:

```
int distribute(int K, int B, int c);
```

Здесь K – число, которое нужно распределить, B – число, начиная с которого нужно распределить число K, c – начальный номер ячейки с массиве a, куда нужно поместить результат распределения. Функция возвращает число найденных вариантов распределения.

Таким образом, шаг 1 на языке C можно записать так:

```
N = distribute(K, B, 0);
```

Далее в качестве описания алгоритма приведём исходный текст функции distribute на языке C/C++ с комментариями. Считаем, что массив a и его размер n – это глобальные переменные. Функция строит все варианты невозрастающего массива a, сумма элементов которого равна K, и возвращает

сумму различных перестановок каждого из этих вариантов. Приведём в качестве примера список вариантов для K=B=4 и n=3:

- в нулевом элементе массива лежит число 4, в остальных нули, данный вариант даёт 3 перестановки;
- в нулевом элементе массива лежит число 3, в первом элементе лежит 1, во втором лежит 0, все числа различны, получаем 3!=6 перестановок;
- в нулевом и первом элементе массива лежит число 2, во втором лежит 0, данный вариант даёт 3 перестановки;
- в нулевом элементе массива лежит число 2, в первом и во втором элементах лежит 1, данный вариант также даёт 3 перестановки.

Мы суммируем все эти перестановки и получаем, что на 4 шаге движения гиперплоскости будет 15 точек.

```
int distribute(int K, int B, int c){
  // На входе всегда B<=K, 0<=c<n.
  if(K == 0) return permutations(a); // Для K=0 единственное
                                     // распределение – все нули.
  int result = 0; // результат, число комбинаций.
  // Пытаемся распределить, начиная с B и кончая 1.
  for(int b = B; b >= 1; --b){
     a[c] = b;       // Распределили первое число.
     int d = K - b; // Сколько ещё осталось распределить.
     if(d > 0){      // Ещё что-то осталось ?
       if(c < n - 1) // А есть ли куда распределять ?
          result += distribute(d, min(b, d), c + 1);
       else break;  // Если не смогли распределить, то дальше нет
                    // смысла продолжать, выходим из цикла.
     } else result += permutations(a);  // Больше нечего
     // распределять, правее все нули, считаем перестановки.
  }
  a[c] = 0;     // Обнуляем, чтобы контролировать содержимое.
  return result; // Возвращаем суммарное число перестановок.
}
```

Функция permutations считает число различных перестановок чисел из массива a в соответствии с формулой (16).

Последнее, что осталось сделать, – это описать алгоритм, который по порядковому номеру точки (индексу внутреннего параллельного цикла алгоритма) вычислит значения всех чисел a_j. Этот алгоритм (функция calc_items) похожа на предыдущий алгоритм (функцию distribute). Он также распределяет число по слагаемым, но заканчивает работу, как только дойдёт до нужного номера точки. Кроме этого, массив a передаётся как параметр, чтобы функцию можно было вызывать одновременно из разных потоков. Перед вызовом функции этот массив нужно обязательно обнулить. На выходе из функции массив содержит числа a_j. Размер массива n по-прежнему лежит в глобальной переменной. Прототип функции следующий:

```
bool calc_items(int K, int B, int c, int a[], int &index);
```

Функция возвращает true в случае успеха и false при неудаче. При вызове извне она всегда должна возвращать true, false возвращается только при внутренних рекурсивных вызовах. По сравнению с функцией distribute, имеется дополнительный параметр index, передаваемый по ссылке. На входе в ней передаётся порядковый номер точки, координаты которой нужно вычислить, на выходе она содержит порядковый номер перестановки в массиве a. В качестве описания алгоритма приведём исходный текст функции на языке C/C++:

```
bool calc_items(int K, int B, int c, int a[], int &index){
  // На входе всегда B<=K, 0<=c<n.
  if(K == 0){
     permutate_items(a, index); // переставляем элементы
     return true;
  }
  // Пытаемся распределить, начиная с B и кончая 1.
  for(int b = B; b >= 1; --b){
     a[c] = b;       // Распределили первое число.
     int d = K - b; // Сколько ещё осталось распределить.
     if(d > 0){      // Ещё что-то осталось ?
       if(c < n - 1){ // Да, а есть ли куда распределять ?
          if(calc_items(d, min(b, d), c + 1, a, index))
            return true; // Всё вычислили, возвращаемся.
       } else break; // Если не смогли распределить, то дальше нет
                     // смысла продолжать, выходим из цикла.
     } else {
       int perms = permutations(a); // Больше нечего распределять,
                    // правее все нули, считаем перестановки.
       if(index < perms){ // Нашли нужную комбинацию.
          permutate_items(a, index); // переставляем элементы
          return true;
       } else index -= perms;
     }
  }
  a[c] = 0;     // Обнуляем, чтобы контролировать содержимое.
  return false; // Не смогли.
    }
```

4. Тестирование

В качестве тестовой рассматривается трёхмерная задача теплопроводности. Находится стационарное решение уравнения теплопроводности

$$\frac{\partial u}{\partial t} = \Delta(u) + f(x, y, z). \qquad (17)$$

Находится приближённое решение для заданного точного решения

$$u_0(x, y, z) = \sin(lx) \times \sin(my) \times \sin(nz). \quad (18)$$

В уравнениях (17) и (18) l, m, n – константы. В этом случае

$$f(x,y,z) = (l^2 + m^2 + n^2) \times \sin(lx) \times \sin(my) \times \sin(nz). \quad (19)$$

Точное решение используется для задания граничных условий (типа Дирихле). Задача решается в единичном кубе, каждое ребро которого делится на N частей, т.е. шаг по направлению h=1/N. Шаг по времени τ полагается равным

$$\tau = \frac{h^2}{12}\ . \qquad (20)$$

Константы полагаются равными l=3, m=6, n=1. Число N в разных запусках полагалось равным 64, 128 или 256. Задача решалась с точностью ε, которая меняется от 0.1 до 1e-6.

Задача решалась разными методами и на разных процессорах, точнее на процессоре Intel Core i7 (последовательная версия программ) и на графическом ускорителе NVIDIA GeForce GTX 670 (параллельная CUDA версия). Задача решалась явным методом (метод Якоби) и двумя неявными методами (Гаусса-Зейделя) – шахматной раскраски и гиперплоскости фронта вычислений. Каждый из неявных методов также решался в двух вариантах – простом и последовательной верхней релаксации (successive over relaxation - SOR). Кроме того, для сравнения были реализованы двухслойный метод простой итерации, двухслойный и трёхслойный Чебышевские методы и многосеточный метод. Все методы, кроме метода обхода точек гиперплоскости, взятого из языка Норма (см. [1]), были реализованы в последовательном и параллельном (для CUDA) вариантах.

Общее описание итерационных методов для линейных систем, включая метод Якоби, Гаусса-Зейделя, последовательной верхней релаксации и других, можно прочитать в [3], глава 6. Метод Гаусса-Зейделя, сформулированный в самом общем виде (как в [3]), подразумевает, что все точки области делятся на несколько подобластей, выстроенных в определённом порядке. В первой подобласти вычисления производятся на основе значений в точках на предыдущей итерации (как в методе Якоби), а вычисления в каждой следующей подобласти – на основе только что вычисленных значений в предыдущих подобластях на текущей итерации. При этом вычисления внутри каждой из подобластей можно производить параллельно. Делить все точки области на подобласти можно по-разному, это приводит к разным вариантам метода Гаусса-Зейделя, в частности, к методу шахматной раскраски (две подобласти) и методу гиперплоскости (много подобластей).

Идея метода шахматной раскраски проста. Вся область разбивается на чередующиеся между собой «чёрные» и «белые» ячейки (трёхмерная шахматная доска). «Чёрные» ячейки считаются первыми и используют значения с предыдущей итерации. Затем считаются «белые» ячейки, которые используют только что посчитанные «чёрные» ячейки с текущей итерации.

В методе гиперплоскости фронта вычислений эта гиперплоскость последовательно передвигается от некоторой начальной вершины под некоторыми углами к осям, пока не дойдёт до противоположной вершины. Начальная вершина и углы к осям определяются параметрами гиперплоскости.

В данной работе гиперплоскость передвигается от вершины с координатами (1, 1, 1) под углом 45° ко всем осям, пока не дойдёт до вершины с координатами (N, N, N).

Обходить точки гиперплоскости на очередном шаге можно по-разному, соответственно, были реализованы 4 алгоритма обхода:

- алгоритм из языка Норма, взятый из работы [1];
- алгоритм, описанный в работе [2] («старый»);
- переборный алгоритм, упомянутый в настоящей статье;
- «новый» алгоритм, описанный в настоящей статье.

Алгоритм «Норма» реализовать для CUDA пока не удалось, он реализован только в последовательном варианте, остальные алгоритмы реализованы как в последовательном, так и в параллельном (CUDA) вариантах.

Идея метода последовательной верхней релаксации (обозначается через SOR(ω), где ω – *параметр релаксации*) состоит в том, чтобы улучшить цикл Гаусса-Зейделя подходящим взвешенным усреднением значений с предыдущей и текущей итераций, при этом значения с текущей итерации берутся с коэффициентом ω, а с предыдущей – 1-ω. Значение параметра релаксации ω=1 соответствует методу Гаусса-Зейделя, при ω<1 говорят о *нижней релаксации*, при ω>1 – о *верхней релаксации*. В методе последовательной верхней релаксации параметр релаксации лежит в интервале от 1 до 2. Подробно об этом методе можно прочитать, например, в [3], раздел 6.5.3. Оптимальное значение параметра релаксации подбиралось экспериментально при малой точности (ε=1e-2). При этом для метода шахматной раскраски оптимальное значение оказалось равным ω=1,18, а для метода гиперплоскости ω=1,83.

Метод простой итерации, двухслойный и трёхслойный Чебышевские методы описаны в [4]. Многосеточный метод описан в [5]. Все эти методы явные, т.е. значения во всех точках вычисляются на основе значений с предыдущей итерации. В этом они похожи на явный метод Якоби. Так же, как и в методе Якоби, значения во всех точках области в одной итерации можно вычислять одновременно. Это существенно упрощает (и, соответственно, ускоряет) алгоритм одного цикла и сильно ускоряет параллельную реализацию.

Общая идея двухслойных методов состоит в построении двухслойной итерационной последовательности вида:

$$u_{j+1} = u_j + \tau_{j+1} \cdot (\Delta u_j + f), j = 0,1, \dots \qquad (21)$$

с произвольным начальным приближением u_0. Здесь $\{\tau_j\}$ – последовательность итерационных параметров. В методе простой итерации все τ_j берутся одинаковыми и равными некоторому оптимальному значению. Это оптимальное значение зависит от минимального и максимального значений собственных чисел линейного оператора в решаемом уравнении. Для уравнения Пуассона (линейный оператор – оператор Лапласа) $\lambda_{min}=24$, $\lambda_{max}= 12/h^2$, где h – шаг сетки (см. [5]). При этом $\tau_j \equiv \tau_0 = 2/(\lambda_{min}+\lambda_{max})$. Используются также следующие обозначения (см. [4]):

$$\xi = \frac{\lambda_{min}}{\lambda_{max}}, \rho_0 = \frac{1-\xi}{1+\xi}, \rho_1 = \frac{1-\sqrt{\xi}}{1+\sqrt{\xi}}. \qquad (22)$$

В двухслойном Чебышевском методе итерационные параметры вычисляются на основе многочленов Чебышева. Число Чебышевских итераций определяется по формуле (см. [4], [5]):

$$n_0(\varepsilon) = \frac{\ln(1/\varepsilon + \sqrt{1/\varepsilon^2 - 1})}{\ln(1/\rho_0)} \approx \frac{\ln\left(\frac{2}{\varepsilon}\right)}{\ln(1/\rho_0)}.$$

Для устойчивости процесса вычислений итерационные параметры должны быть определённым образом упорядочены, алгоритм их упорядочения взят из [4], глава VI, §2, п.5, стр. 280-283. Особенностью этого метода является то, что количество итераций определяется заранее, исходя из размера сетки и требуемой точности вычислений. При этом для сходимости нужно выполнить все эти итерации. Во всех других методах, реализованных в настоящей работе (кроме Чебышевских и многосеточного), в конце каждой итерации вычисляется норма невязки и проверяется, стала ли она в нужное число раз меньше начальной невязки. Если нет, то выполняется следующая итерация. Таким образом, общее количество итераций становится известным только в конце вычислений.

В трёхслойном Чебышевском методе строится трёхслойная итерационная схема с произвольным начальным приближением u_0. Значение u_1 вычисляется как в методе простой итерации:

$$u_1 = u_0 + \tau \cdot (\Delta u_0 + f). \quad (23)$$

Следующие итерации вычисляются по формуле:

$$u_{j+1} = (1 - \alpha_{j+1}) \cdot u_{j-1} + \alpha_{j+1} \cdot \left(u_j + \tau \cdot (\Delta u_j + f)\right), j = 1,2, \ldots \qquad (24)$$

Здесь итерационный параметр τ такой же, как в методе простой итерации. Из формулы (24) видно, что при $\alpha_j \equiv 1$ трёхслойный метод превращается в метод простой итерации. Реальные итерационные параметры $\{\alpha_j\}$ вычисляются с использованием полиномов Чебышева первого рода и обеспечивают оптимальную сходимость метода. Оптимальные значения итерационных параметров вычисляются из рекуррентной формулы $\alpha_{j+1}=4/(4-\rho_0^2\alpha_j)$, $\alpha_1=2$ (см. [4]), где ρ_0 определяется по формуле (22). Число итераций, требуемых для достижения заданной точности ε, определяется как

$$n_0(\varepsilon) = \frac{\ln 0{,}5\,\varepsilon}{\ln \rho_1}.$$

Возможен также стационарный трёхслойный метод, в котором все α_j равны между собой и равны оптимальному значению, равному $\alpha_j \equiv \alpha = 1+\rho_1^2$, где ρ_1 определяется по формуле (22).

В многосеточном методе используется 5 сеточных уровней. Самый грубый уровень решается с помощью двухслойного Чебышевского метода. На других уровнях перед сборкой (переходом на грубый уровень) и после проектирования (перехода на подробный уровень) осуществляется сглаживание тем же

двухслойным Чебышевским методом, но с малым числом итераций (уменьшение нормы невязки в два раза), как правило, это 2 шага.

Все задачи решаются с использованием операторной библиотеки gridmath (см. [6]), которая позволяет, во-первых, компактно записывать алгоритмы с использованием операторов, а во-вторых, обеспечивает лёгкий перенос (практически простой перекомпиляцией) на CUDA.

Интересно посмотреть, с одной стороны, на то, за сколько шагов сойдётся с заданной точностью решение для разных алгоритмов, и, с другой стороны, измерить быстродействие разных алгоритмов. Для параллельной реализации того или иного алгоритма интересно посмотреть, насколько он ускорится по сравнению с последовательной версией того же алгоритма. Оценить сложность алгоритма можно по времени выполнения одной итерации. Простые алгоритмы (в которых время одной итерации мало) даже при большем числе итераций могут показать лучшее время.

Вначале приведём результаты для количества итераций:

Таблица 1. Число итераций

Размер	64				128			256		
Погрешность	1e-1	1e-3	1e-6	1e-12	1e-1	1e-2	1e-3	1e-1	1e-2	1e-3
Якоби	39	1671	6223	17686	29	1323	4961	28	1009	13061
Chess	40	910	3256	8987	25	914	2769	23	989	7653
Гиперплоскость	21	839	2891	8621	16	666	2489	15	510	6544
Chess+SOR	42	665	2323	6304	25	742	2022	22	992	5735
ГП+SOR	13	91	335	827	11	69	244	10	55	624
Чебышев 2 и 3	68	172	329	641	136	240	344	271	480	688
Прост. итерация	100	1700	6300	17700	100	1400	5000	100	1100	13100
Multigrid	2	5	10	18	2	3	5	2	3	5

Из приведённой таблицы видно, метод Якоби по количеству шагов существенно отстаёт от других методов. Метод простой итерации сходится примерно за то же количество шагов, что и метод Якоби. Метод шахматной раскраски асимптотически вдвое лучше, чем метод Якоби (см. [3]), что видно и из таблицы. Метод гиперплоскостей даёт лучшую сходимость, чем метод шахматной раскраски. Применение метода последовательной верхней релаксации к методу шахматной раскраски примерно на четверть уменьшает количество итераций. Применение же метода последовательной верхней релаксации к методу гиперплоскости уменьшает число итераций на порядок.

С ростом точности вычислений число итераций растёт примерно пропорционально точности, т.е. при увеличении точности на порядок (уменьшении ε в 10 раз) число итераций увеличивается в разы (5 – 10 раз). Это общая закономерность. Конкретное же число итераций предсказать трудно, ошибиться также можно в разы. В этом смысле выгодно отличается от других алгоритмов двухслойный и трёхслойный Чебышевский методы, а также многосеточный метод. Как уже говорилось выше, число итераций в этих методах определяется до начала вычислений, при этом это число пропорционально $\ln(\varepsilon^{-1})$. Хотя при малых ε число итераций довольно велико, с ростом точности оно растёт несильно. При увеличении точности вычислений на

порядок (уменьшении ε в 10 раз) количество итераций увеличивается примерно в 1,5 раза. При ε=1e-1 Чебышевские методы проигрывают почти всем методам, при ε=1e-3 проигрывают только методу гиперплоскости совместно с последовательной верхней релаксацией, а при меньших ε становятся лучше всех. Если учесть, что двухслойный Чебышевский метод вычислительно прост и одна итерация занимает мало (относительно других методов) времени (см. ниже), уже начиная с ε=1e-2 он становится самым быстрым. Кроме того, благодаря свой простоте, он очень хорошо ускоряется на CUDA: ускорение, по сравнению с последовательной версией того же алгоритма, достигает 10-12 раз. Особое место по своей эффективности занимает многосеточный метод. Это единственный метод, в котором число итераций не зависит от размера сетки, а только от точности вычислений, при этом оно очень мало.

Теперь результаты по времени. Полное время указано в секундах, а время одного шага – в миллисекундах.

Таблица 2. Время

Размер	64				128			256	
ε	1e-1	1e-3	1e-6	1e-12	1e-1	1e-2	1e-3	1e-1	1e-2
Якоби	0,16	5,71	21,41	65,38	0,99	37,47	140,75	6,57	234,13
1 шаг, мс	4,05	3,42	3,44	3,70	34,17	28,32	28,37	234,64	232,04
CUDA	0,05	1,55	5,49	15,34	0,12	4,46	16,64	0,64	22,39
Ускорение	3,04	3,69	3,90	4,26	8,54	8,40	8,46	10,27	10,46
1 шаг, мс	1,33	0,93	0,88	0,87	4,00	3,37	3,35	22,86	22,19
Chess	0,16	3,18	11,35	34,92	0,92	28,35	84,47	5,61	240,23
1 шаг, мс	3,90	3,50	3,49	3,89	36,68	31,02	30,51	244,04	242,90
CUDA	0,05	1,03	2,98	8,02	0,11	3,11	9,24	0,53	21,69
Ускорение	3,32	3,09	3,80	4,36	8,41	9,12	9,14	10,57	11,08
1 шаг, мс	1,18	1,13	0,92	0,89	4,36	3,40	3,34	23,09	21,93
Ст. метод	0,30	9,06	31,14		2,49	103,60	383,39	19,882	
1 шаг, мс	14,10	10,79	10,77		155,31	155,55	154,03	1325,47	
CUDA	0,22	8,39	28,91		0,71	28,75	106,32	6,22	211,19
Ускорение	1,36	1,08	1,08		3,49	3,60	3,61	3,20	
1 шаг, мс	10,38	10,00	10,00		44,56	43,17	42,71	414,47	414,09
Перебор	0,16	5,71	19,66	58,53	1,72	71,71	269,99	15,03	518,66
1 шаг, мс	7,48	6,80	6,80	6,79	107,38	107,67	108,47	1001,93	1016,98
CUDA	0,18	6,65	22,87	67,26	0,47	18,40	68,52	4,39	148,95
Ускорение	0,89	0,87	0,86	0,87	3,69	3,90	3,94	3,42	3,48
1 шаг, мс	8,38	7,93	7,91	7,80	29,13	27,63	27,53	292,87	292,05
Новый	0,11	3,93	13,45	40,22	1,45	60,03	223,14	12,38	419,09
1 шаг, мс	5,19	4,69	4,65	46,70	90,56	90,14	89,65	825,00	821,75
CUDA	0,19	6,78	23,31	69,54	0,47	18,02	66,92	4,28	145,48
Ускорение	0,58	0,58	0,58	0,58	3,12	3,33	3,33	2,89	2,88
1 шаг, мс	8,90	8,08	8,06	8,07	29,06	27,05	26,89	285,27	285,26
new+SOR	0,08	0,47	1,76	4,27	1,17	7,32	25,85	9,93	53,85
1 шаг, мс	6,00	5,15	5,25	5,16	106,55	106,13	105,95	993,40	979,07
CUDA	0,13	0,75	2,77	6,78	0,33	1,98	6,96	3,16	17,26
Ускорение	0,62	0,63	0,63	0,63	3,57	3,71	3,71	3,15	3,12
1 шаг, мс	9,62	8,24	8,27	8,19	29,82	29,62	28,53	315,70	313,82
Норма	0,11	4,09	13,69	40,81	1,44	59,63	221,99	12,98	499,57

1 шаг, мс	5,24	4,87	4,74	4,73	89,88	89,54	89,19	865,20	979,54
Чебышев	0,14	0,36	0,69	1,34	2,36	4,11	5,88	37,56	66,59
1 шаг, мс	2,06	2,09	2,10	2,09	17,38	17,12	17,08	138,58	138,73
CUDA	0,02	0,03	0,08	0,13	0,16	0,30	0,47	3,26	5,73
Ускорение	8,75	11,25	8,86	10,70	15,06	13,83	12,53	11,53	11,62
1 шаг, мс	0,24	0,19	0,24	0,20	1,15	1,24	1,36	12,01	11,94
Multigrid	0.03	0.09	0.19	0.31	0.27	0.39	0.66	2.15	3.23
1 шаг, мс	16,00	18,60	18,80	18,08	132,50	130,00	131,20	1074,00	1076,33
CUDA	0,02	0,03	0,05	0,08	0,03	0,05	0,08	0,19	0,30
Ускорение	2,00	3,00	4,00	4,00	8,55	8,30	8,41	11,43	10,87
1 шаг, мс	8,00	6,20	4,70	4,33	15,50	15,67	15,60	94,00	99,00

Эта таблица полная, но слишком большая. Для большей наглядности перевернём таблицу и оставим только часть данных. Кроме того, сделаем отдельные таблицы для разных размеров сетки и точности вычислений.

Таблица 3. Время, N=64, ε=1e-3

	Якоби	Chess	Ch+SOR	Гипер	Г+SOR	Чебышев	Multigrid
N итерац.	1671	910	665	839	91	172	5
Время, с	5,71	3,18	2,76	3,93	0,47	0,36	0,09
1 шаг, мс	3,42	3,50	4,15	4,69	5,15	2,09	18,60
CUDA	1,55	1,03	0,92	6,78	0,75	0,03	0,03
Ускорение	3,69	3,09	2,99	0,58	0,63	11,25	3,00
1 шаг, мс	0,93	1,13	0,38	8,08	8,24	0,19	6,20

Таблица 4. Время, N=64, ε=1e-12

	Якоби	Chess	Ch+SOR	Гипер	Г+SOR	Чебышев	Multigrid
N итерац.	17688	8987	6304	8621	827	641	18
Время, с	65,38	34,92	26,42	40,22	4,27	1,34	0,31
1 шаг, мс	3,70	3,89	4,19	4,67	5,16	2,09	18,80
CUDA	15,34	8,02	5,97	69,54	6,78	0,13	0,08
Ускорение	4,26	4,36	4,34	0,58	0,63	10,70	4,00
1 шаг, мс	0,88	0,89	0,95	8,07	8,19	0,20	4,33

Таблица 5. Время, N=128, ε=1e-3

	Якоби	Chess	Ch+SOR	Гипер	Г+SOR	Чебышев	Multigrid
N итерац.	4961	2769	2022	2489	244	344	5
Время, с	140,75	84,47	69,96	223,14	25,85	5,88	0,66
1 шаг, мс	28,37	30,51	34,60	89,65	105,95	17,08	131,20
CUDA	16,64	9,24	6,98	66,92	6,96	0,47	0,08
Ускорение	8,49	9,14	10,02	3,33	3,71	12,53	8,41
1 шаг, мс	3,35	3,34	3,45	26,89	28,53	1,36	15,60

Таблица 6. Время, N=256, ε=1e-1

	Якоби	Chess	Ch+SOR	Гипер	Г+SOR	Чебышев	Multigrid
N итерац.	28	23	22	15	10	271	2
Время, с	6,57	5,61	6,23	12,83	9,93	37,56	2,15
1 шаг, мс	234,64	244,04	283,05	825,00	993,40	138,58	1074,00
CUDA	0,64	0,53	0,53	4,28	3,16	3,26	0,19
Ускорение	10,27	10,57	11,73	2,89	3,15	11,53	11,43
1 шаг, мс	22,86	23,09	24,14	285,27	315,70	12,01	94,00

Таблица 7. Время, N=256, ε=1e-2

	Якоби	Chess	Ch+SOR	Гипер	Г+SOR	Чебышев	Multigrid
N итерац.	1009	989	992	510	55	480	3
Время, с	234,13	240,23	280,41	419,09	53,85	66,59	3,23
1 шаг, мс	232,04	242,90	282,67	821,75	979,07	138,73	1076,33
CUDA	22,39	21,69	22,41	145,48	17,26	5,73	0,30
Ускорение	10,46	11,08	12,51	2,88	3,12	11,62	10,87
1 шаг, мс	22,19	21,93	22,59	285,26	313,82	11,94	99,00

Таблица 8. Время, N=256, ε=1e-3

	Якоби	Chess	Ch+SOR	Гипер	Г+SOR	Чебышев	Multigrid
N итерац.	13061	7653	5735	6544	624	688	5
Время, с					610,21	95,31	5,83
1 шаг, мс					977,90	138,54	1076,60
CUDA	288,80	167,56	129,36	1863,91	195,63	8,18	0,50
Ускорение					3,12	11,66	10,77
1 шаг, мс	22,11	21,90	22,56	284,83	313,50	11,88	100,00

5. Заключение

Какие выводы можно сделать из этих таблиц?

Есть методы вычислительно (они же алгоритмически) простые и сложные. Сложные методы характеризуются большим (относительно простых методов) временем исполнения одного цикла итерации. При этом за счёт существенного сокращения числа итераций общее время исполнения сложного алгоритма может быть меньше, чем у простого. Из таблиц видно, что сложные методы – это методы, связанные с обходом гиперплоскости, а также многосеточный метод. Ещё одна неприятная особенность сложных неявных методов (прежде всего обход гиперплоскости) – плохое ускорение на CUDA (а на малых размерах сетки – даже замедление). Максимальное ускорение для сложных методов не превышает 4. В то же время для простых методов ускорение часто превышает 10.

Безусловным лидером по производительности является многосеточный метод. Хотя он алгоритмически и вычислительно самый сложный (наибольшее

время выполнения одной итерации), но за счёт очень малого числа итераций, требуемых для сходимости с заданной точностью, он является лидером для любых размеров сетки и любой требуемой точности. Кроме того, благодаря тому, что он явный (значения в точках на очередной итерации зависят только от значений в точках на предыдущей итерации), он прекрасно распараллеливается и очень хорошо ускоряется на CUDA.

Если стоит вопрос выбора наиболее быстрого метода (не считая многосеточного), то определяющим фактором для выбора служит требуемая точность вычислений. Если нужна небольшая точность (например, уменьшение нормы невязки в 10 раз), то можно выбрать или простейший явный метод (Якоби), или метод шахматной раскраски. При точности 10^{-3} и выше безусловным лидером является двухслойный Чебышевский метод.

- Явный метод, несмотря на большое количество шагов, благодаря свой простоте один из самых быстрых, а по времени одного шага – самый быстрый. Очень хорошо ускоряется на CUDA, на больших сетках более чем в 10 раз.
- Метод шахматной раскраски – лидер по производительности. По скорости одного шага он лишь немного уступает явному методу (примерно на 10%), а благодаря тому, что шагов примерно на 40% меньше, по суммарному времени он самый быстрый. Очень хорошо ускоряется на CUDA, на больших сетках примерно в 11 раз.
- Метод гиперплоскости самый эффективный по сходимости, в нём меньше всего шагов. Но он сложен в реализации и поэтому по производительности существенно (в разы) уступает явному методу и методу шахматной раскраски. Видимо, его следует использовать только в том случае, если он действительно необходим. Например, для данной задачи острой необходимости в нём нет, она прекрасно решается, например, методом шахматной раскраски.
- Если сравнивать разные алгоритмы обхода гиперплоскости, то выводы неоднозначные. Для последовательного случая самым быстрым является метод Норма, но пока его не удалось распараллелить для CUDA. На маленьких размерах сетки (64) он абсолютный лидер, даже в последовательном варианте он работает быстрее, чем другие алгоритмы на CUDA. Но на больших размерах сетки (128 и 256) абсолютным лидером становится новый алгоритм, запущенный на CUDA, благодаря тому, что он очень хорошо ускоряется (более чем в 9 раз).

6. Литература

1. Андрианов А.Н. Организация циклических вычислений в языке Норма. // Препринты ИПМ им. М.В. Келдыша АН СССР, 1986, № 171
2. Краснов М.М. Параллельный алгоритм вычисления точек гиперплоскости фронта вычислений. // Журнал вычислительной математики и математической физики, 2015, том 55, № 1, с. 145-152.
3. Деммель Дж. Вычислительная линейная алгебра. Теория и приложения. М.: «Мир», 2001, 430 с.
4. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. М.: «Наука», Главная редакция физико-математической литературы, 1978, 592 с.
5. 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. URL: http://library.keldysh.ru/preprint.asp?id=2012-30
6. Краснов М.М. Операторная библиотека для решения трёхмерных сеточных задач математической физики с использованием графических плат с архитектурой CUDA. // Математическое моделирование, 2015, № 3, т.27, с. 109-120.

Оглавление