НАУЧНОЕ ИЗДАНИЕ МГТУ ИМ. Н. Э. БАУМАНА

НАУКА и ОБРАЗОВАНИЕ

Эл № ФС77 - 48211. Государственная регистрация №0421200025. ISSN 1994-0408

электронный научно-технический журнал

Решение систем линейных алгебраических уравнений методом предобуславливания на графических процессорных устройствах

01, январь 2013

DOI: 10.7463/0113.0525190

Карпенко А. П., Чернов С. К.

УДК 519.6

Россия, МГТУ им. Н.Э. Баумана

apkarpenko@mail.ru

lemonlime@yandex.ru

Введение

В настоящее время вычислительная гидродинамика находит широкое применение при решении, например, задач авиа-, судо- и двигателестроения. В силу высокой вычислительной сложности методов этой науки для решения указанных и других задач используют параллельные вычислительные системы различных классов и, прежде всего, кластерные системы. Для уменьшения расходов на вычисления в настоящее время в качестве альтернативы кластерам все чаще используют графические процессорные устройства (ГПУ). Статья подготовлена в рамках работ по адаптации программного комплекса *FlowVision* [1] российской компании «Тесис» к ГПУ.

На верхнем иерархическом уровне рассматриваемая в работе задача представляет собой задачу течения жидкости или газа в некотором заданном объеме. Имеется в виду, что для описания течения используются математически модели, основанные на дифференциальных уравнениях в частных производных (ДУЧП) Навье-Стокса, Эйлера, Рейнольдса и т. д. Для численного решения этих уравнений чаще всего применяют методы конечных разностей (КР), конечных элементов (КЭ) и конечных объемов (КО).

При использовании регулярной сетки метод КР прост в реализации,

эффективен, имеет наглядную процедуру дискретизации и позволяет легко получить высокий порядок точности. Существенным недостатком метода является то, что его достоинства проявляются только при использовании регулярной сетки, которая возможна лишь в случае простой геометрии расчетной области [2]. К достоинствам метода КЭ относится простота использования его для задач с произвольной формой области решения, к недостаткам — высокая вычислительная и алгоритмическая сложность [2].

И метод КР и метод КЭ обладают еще тем недостатком, что обычно не обеспечивают в явном виде выполнения законов сохранения [3]. От этого недостатка свободен метод КО [3]. По сравнению с методов КР, метод КО обладает более высокими возможностями использования в областях произвольной формы, а по сравнению с методом КЭ – меньшими вычислительными затратами. На этом основании в программном комплексе *FlowVision* применяется именно метод КО.

Численное решение краевых задач для ДУЧП является весьма ресурсоемкой задачей. Так, например, современные практически значимые задачи гидрогазодинамики требуют использования неструктурированных адаптивных расчетных сеток с числом ячеек порядка 100 млн. Поэтому одним из главных направлений развития *FlowVision* является повышение эффективности использования вычислительных ресурсов.

В настоящее время *FlowVision* ориентирован на кластерные вычислительные системы, процессоры которых являются многоядерными [1]. На каждом процессоре кластера *FlowVision* запускает один процесс, использующий библиотеки *Intel TBB* и *OpenMP* для создания некоторого динамически определяемого числа потоков (*threads*). Обмен данными между этими потоками происходит через общую память процессора. Обмен данными между процессами, запущенными на разных процессорах, реализуется с помощью библиотека *MPI*.

В связи с тем, что в настоящее время идет активное внедрение гетерогенных кластерных вычислительных систем, в состав которых входят

ГПУ, все более актуальной становится задача адаптации программного обеспечения и, в частности, *FlowVision*, к таким системам. Решение этой задачи, с одной стороны, может дать заметный выигрыш в производительности, с другой стороны, сложная многоуровневая гибридная архитектура таких вычислительных систем приводит к значительному усложнению программного обеспечения.

Современными производителями ГПУ являются, прежде всего, компании *AMD* и *NVIDIA*. Основу ГПУ обоих производителей составляет набор *SIMD* узлов (потоковых мультипроцессоров), связанных разделяемой памятью [4]. Каждый из потоковых мультипроцессоров включает в себя несколько потоковых процессоров, блок специализированных АЛУ, обеспечивающих вычисление наиболее часто встречающихся математических функций, блок вычислений с двойной точностью, разделяемую память и т.д.

Программирование ГПУ обеих указанных компаний осуществляют с помощью универсальной технологии *OpenCL* [5] и технологии *CUDA* [6], разработанной компанией *NVIDIA* для ее собственных ГПУ. Несмотря на универсальность технологии *OpenCL*, работа ориентирована на использование ГПУ *NVIDIA* и технологии *CUDA*. Данный выбор обусловлен следующими соображениями:

- неполная на момент начала работы поддержка обоими производителями стандарта *OpenCL*;
- более интенсивное развитие технологии *CUDA* по сравнению с технологией *OpenCL*;
- наличие у авторов опыта работы с технологией *CUDA*.

Компания *NVIDIA* предоставляет библиотеку *cuBLAS*, реализующую некоторые алгоритмы линейной алгебры [7]. Например, библиотека поддерживает умножение матрицы на матрицу, матрицы на вектор и т.д. Однако библиотека ориентирована только на операции с плотными матрицами и поэтому не может быть использована в нашем случае. Для работы с

разреженными матрицами предназначена ориентированная на ГПУ библиотека *cuSPARSE* [8], но в этой библиотеке в настоящее время реализована операция умножения блочно-разреженной матрицы только на один вектор, в то время как в *FlowVision* требуется умножение такой матрицы на набор векторов.

В первом разделе работы представлена постановка задачи. Во втором разделе обсуждаем алгоритм неполных треугольных факторизаций второго порядка точности *ILU2*, распараллеливанию которого и посвящена работа. В третьем разделе рассматриваем алгоритмическую и программную реализации на ГПУ двух основных операций алгоритма *ILU2* – операция умножения матрицы на блок векторов и операция решения блочно-треугольной системы линейных алгебраических уравнений (СЛАУ). В четвертом разделе представлены результаты исследования эффективности принятых алгоритмических и программных решений. В заключении сформулированы основные результаты работы и перспективы ее развития.

1 Постановка задачи

FlowVision позволяет производить полный комплекс действий для расчёта 3*D* стационарных и нестационарных течений газов и жидкостей в областях произвольной сложной формы с учетом теплообмена и химических реакций:

- импорт геометрической модели;
- автоматизированное построение расчетной сетки на основе модели;
- задание начальных и граничных условий;
- формирование математической модели в виде СЛАУ на основе метода КО;
- решение указанной СЛАУ;
- визуализация результатов моделирования.

Комплекс использует адаптивную локально измельчаемую расчетную сетку с прямоугольными ячейками и с подсеточным разрешением геометрии [9]. Последний механизм позволяет аппроксимировать

криволинейные границы без измельчения сетки и основан на том, что ячейка, через которую проходит граница, разбивается на несколько ячеек, часть из которых исключается из расчета. Оставшиеся ячейки представляют собой не прямоугольные параллелепипеды, а многогранники произвольной формы. Важно, что уравнения математической модели для этих многогранников *FlowVision* аппроксимирует без упрощений, что позволяет рассчитывать течения с высокой точностью даже на грубой сетке.

Метод КО, в конечном счете, сводит решение модельной краевой задачи для ДУЧП к решению СЛАУ вида

$$Ax = b, \tag{1}$$

где A - $(\nu \times \nu)$-разреженная квадратная матрица, b - $(\nu \times 1)$-вектор правой части, x - искомый $(\nu \times 1)$-вектор решения. Особенностями СЛАУ (1) являются высокая размерность и возможная плохая обусловленность.

Для решения СЛАУ разработано большое число различных методов, наиболее простыми из которых являются прямые методы (метод исключения Гаусса, Гаусса-Жордана и прочие). Хорошо известно, что методы этого класса не эффективны при решении плохо обусловленных систем и систем высокой размерности. Как правило, в вычислительной практике применяют итерационные методы и, в частности, методы, основанные на предобуславливателях [10].

Известно значительное число методов предобуславливания: приближенные обратные методы; приближенные факторизованные обратные методы; методы, основанные на неполной треугольной факторизации и т. д. Методы различаются затратами на построение предобуславливателя, свойствами сходимости и масштабируемости при параллельной реализации.

В комплексе *FlowVision* используется алгоритм неполных треугольных факторизаций второго порядка точности *ILU2*, являющийся несимметричным обобщением алгоритма *ICH2* [10]. Алгоритм *ILU2* показал высокую вычислительную эффективность для широкого класса задач, в том числе, для задач вычислительной гидрогазодинамики [11]. Ставится задача реализации этого алгоритма на ГПУ.

2. Распараллеливание алгоритма *ILU2*

Процесс факторизации в соответствии с алгоритмом *ILU2* основан на матричном соотношении

$$A + E = L\,U + L\,R + W\,U\,,$$

где A - масштабированная матрица системы (1), L и U — нижняя и верхняя треугольные части элементов первого порядка точности, W и R – аналогичные части элементов второго порядка точности, E - матрица ошибок.

В итерационной схеме алгоритма *ILU2* итерации производятся с матрицей $A(L\,U)^{-1}$. Алгоритм требует хранения в памяти ЭВМ всех элементов матриц A, L, U, небольшой части элементов матриц W и R и включает в себя следующие операции [11]:

1) построение предобуславливателя (неполная факторизация матрицы),

2) умножение матрицы на вектор,

3) решение систем уравнений с верхней и нижней треугольными матрицами,

4) векторные операции.

Для обеспечения высокой эффективности реализации на ГПУ алгоритма *ILU2* необходимо согласованное распараллеливание всех указанных операций. Рассмотрим первые три из них.

Построение предобуславливателя. В блочном виде рекуррентное выражение для операции треугольной факторизации имеет вид

$$\begin{bmatrix} A_{1,1} & a_{1,2} & A_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ A_{3,1} & a_{3,2} & A_{3,3} \end{bmatrix} = \begin{vmatrix} L_{1,1} & 0 & 0 \\ l_{2,1} & l_{2,2} & 0 \\ L_{3,1} & l_{3,2} & L_{3,3} \end{vmatrix} \times \begin{bmatrix} U_{1,1} & u_{1,2} & U_{1,3} \\ 0 & u_{2,2} & u_{2,3} \\ 0 & 0 & U_{3,3} \end{bmatrix}.$$

Здесь блоки $a_{2,2}, l_{2,2}, u_{2,2}$ имеет единичный размер; блоки $a_{2,1}, a_{2,3}, l_{2,1}, u_{2,3}$ являются строками; блоки $a_{1,2}, a_{3,2}, l_{3,2}, u_{1,2}$ - столбцами; $L_{i,j}, U_{i,j}$ - квадратные матрицы. На каждой итерации необходимы следующие операции:

1) на основе уравнения $l_{2,2}\, u_{2,2} = a_{2,2} - l_{2,1}\, u_{1,2}$ вычислить значения компонентов строк $l_{2,2}, u_{2,2}$;

2) из уравнений $u_{2,3} = \dfrac{a_{2,3} - l_{2,1}\, U_{1,3}}{l_{2,2}}$, $l_{3,2} = \dfrac{a_{3,2} - L_{3,1}\, u_{1,2}}{u_{2,2}}$ найти компоненты строки $u_{2,3}$ и столбца $l_{3,2}$.

Зависимости по данным для строки матрицы U представленного алгоритма иллюстрирует рисунок 1. Для столбца матрицы L зависимость по данным получается симметричным отражением представленных зависимостей относительно главной диагонали матрицы.

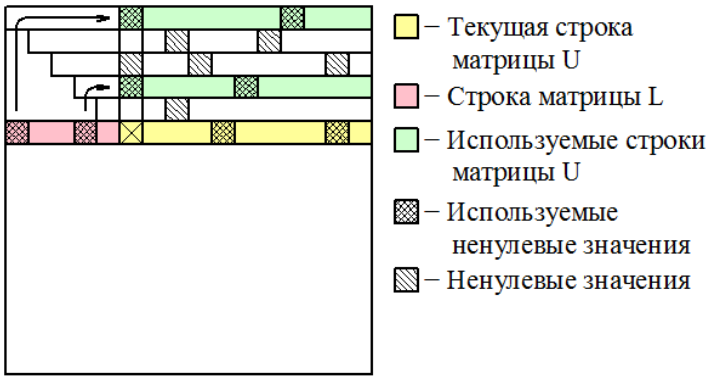


Рисунок 1 - Зависимости по данным при вычислении компонентов строки матрицы U в процессе факторизации

Различные варианты метода неполных разложений, в том числе алгоритмы *ICH2*, *ILU2*, представляют собой некоторые модификации шагов 1, 2 [11].

Умножение матрицы на вектор осуществляется по формуле

$$y_i = \sum_{j=0}^{\nu-1} A_{i,j}\, x_j, \quad i = 0,1,\ldots,(\nu-1),$$

из которой следует, что при выполнении данной операции рекуррентная зависимость по данным отсутствует.

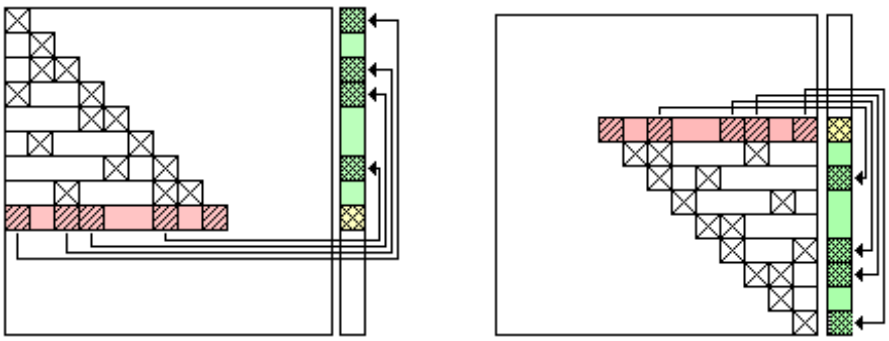
Решение систем уравнений. Рекуррентные уравнения для решения треугольных систем с матрицами L и U имеют, соответственно, вид

$$x_i = \frac{b_i - \sum_{j=0}^{i-1} l_{j,i}\, x_j}{l_{i,i}}, \; i = 1,\ldots,(\nu - 1), \tag{2}$$

$$x_i = \frac{b_i - \sum_{j=i+1}^{n-1} u_{j,i}\, x_j}{u_{i,i}}, \; i = (\nu - 2), (\nu - 3),\ldots,0. \tag{3}$$

Здесь $l_{j,i}, u_{j.i}$ - элементы матриц L, U.

Соответствующие зависимости по данным иллюстрирует рисунок 2.



а) б)

Рисунок 2 - Зависимости по данным: а) для рекуррентных уравнений (2); б) для рекуррентных уравнений (3)

Положим, что существует такая нумерация переменных, что матрица СЛАУ имеет вид

$$A = \begin{bmatrix} A_1 & \ldots & 0 & C_1 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & A_p & C_p \\ B_1 & \ldots & B_p & D \end{bmatrix}, \tag{4}$$

где A_i - квадратные блоки. Будем говорить, что эта матрица является *p*-блочно-окаймленной.

Легко видеть, что *p*-блочно-окаймленное представление матрицы A дает возможность организовать *независимую параллельную факторизацию* p диагональных блоков $A_1, A_2,..., A_p$. В то же время, факторизация

окаймляющих блоков не может быть выполнена независимо.

Существуют различные способы нумерации переменных, при которых матрица СЛАУ приобретает вид (4). Например, в методе вложенных сечений [11] строится расщепление вида (4) при p=2. Для каждого из полученных блоков снова строится расщепление вида (4) при p=2 и так далее (рисунок 3).

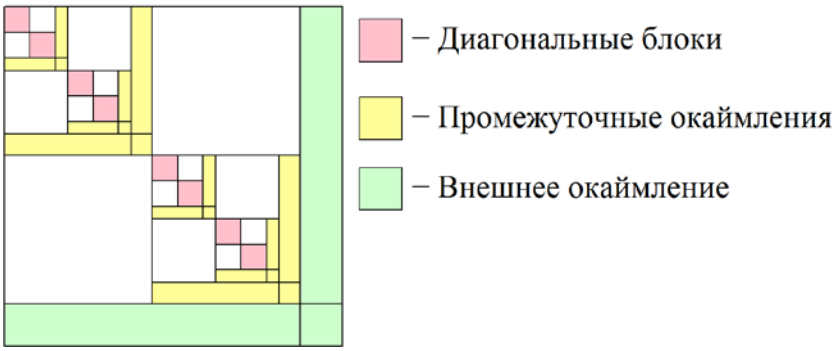


Рисунок 3. К методу вложенных сечений (четыре уровня расщепления)

Представление (4) можно получить на основе геометрической декомпозиции исходной краевой задачи для ДУЧП, когда расчетная область задачи разбивается на несколько подобластей, каждая из которых связна и имеет минимальную из возможных границу с другими подобластями [11]. На основе такой декомпозиции можно организовать двухуровневое распараллеливание при решении задачи на мультикомпьютере, узлами которого являются мультипроцессоры или многоядерные процессоры, по следующей схеме: на верхнем уровне иерархии каждую из подобластей ставим в соответствие одному узлу вычислительной системы; на втором уровне иерархии распараллеливание в пределах подобласти осуществляем, используя многопроцессорность или многоядерность узла.

Для ГПУ рассмотренная схема распараллеливания не может обеспечить высокую эффективность вычислений, поскольку позволяет реализовать параллелизм уровня мультипроцессоров (*cuda*-блоков), но не позволяет использовать параллелизм уровня потоковых процессоров (*cuda*-потоков). С целью обеспечения высокой эффективности параллельных вычислений на

ГПУ используем так называемое мелкоблочное (м-блочное) представление разреженной матрицы.

Мелкоблочная разреженная матрица A появляется при моделировании сложных физических процессов, имеющих место, в частности, в задачах гидрогазодинамики. Такие задачи требуют учета взаимовлияния различных физических величин, например, скорости, давления, температуры, плотности и т. д. Указанные величины можно пронумеровать таким образом, чтобы неизвестные, соответствующие одному узлу расчетной сетки, оказались в векторе неизвестных x рядом, а коэффициенты при них образовали плотные м-блоки в структуре матрицы A. В трехмерном случае для задачи гидродинамики без учета теплообмена м-блоки имеют размерность 4×4 (три декартовых компоненты скорости и давление), в той же задаче, но с учетом теплообмена, - размерность 5×5 и т. д.

Исходим из предположения, что все м-блоки матрицы A являются плотными и храним все компоненты каждого из этих блоков.

Недостаток м-блочного представления матрицы A заключается в том, что случае наличия в м-блоке всего одного ненулевого элемента, требуется хранить весь этот блок. Достоинством м-блочного представления является возможность увеличения числа однотипных вычислительных операций, что позволяет обеспечить более полную загрузку ГПУ на уровне потоковых процессоров.

В работе выполнена реализация на ГПУ двух основных операций алгоритма *ILU2*: умножение м-блочной матрицы на набор векторов (операция, в которой отсутствуют зависимости по данным); решение м-блочно-треугольной СЛАУ (операция, имеющая зависимости по данным).

Рекуррентные соотношения при решении м-блочно-треугольной системы с матрицами L, U аналогичны уравнениям (2), (3) и имеют вид

$$X_i = L_{i,i}^{-1}\left(B_i - \sum_{j=0}^{i-1} L_{j,i}\, X_j\right),\ i = 0, 2, \ldots, (N-1), \qquad (5)$$

$$X_i = U_{i,i}^{-1}\left(B_i - \sum_{j=i+1}^{N-1} U_{j,i}\, X_j\right),\ i = (N-1), (N-2), \ldots, 0. \qquad (6)$$

Здесь N - число м-блочных строк в матрице A; $L_{i,i}, L_{j,i}, U_{i,i}, U_{j,i}$ - м-блоки соответствующих матриц; X_i, B_i - м-подвекторы векторов x, b соответственно.

Из выражений (5), (6) следует, что схема зависимостей по данным для м-блочно-треугольных систем аналогична схеме зависимостей обычной треугольной системы (рисунок 2) с той разницей, что вместо скалярных величин в этой схеме фигурируют м-блоки матриц и м-подвекторы переменных.

Алгоритм (5), (6), во-первых, имеет имеется крупноблочную структуру типа (4), позволяющую осуществить эффективное распараллеливание на уровне ядер ЦПУ либо на уровне мультипроцессоров ГПУ. Во-вторых, этот алгоритм включает в себя операции умножения плотного м-блока на плотный м-подвектор, что дает возможность реализовать распараллеливание вычислений на уровне потоковых процессоров ГПУ.

3. Программная реализация

Представление матриц. В программной реализации используем квадратные матрицы с квадратными м-блоками постоянного размера. Разреженную м-блочную матрицу представляем в виде, основанном на йельском формате, который также известен как формат *CSR (Compressed Sparse Row)*.

Формат предусматривает использование массивов *A, I, J*. Здесь массив *A* имеет размерность $m \times (n \times n)$, где *m* - число м-блоков в матрице, а *n* - размер блока. Элементы исходной матрицы сгруппированы в массиве *A* по блокам, а внутри блока - по строкам (рисунок 4).

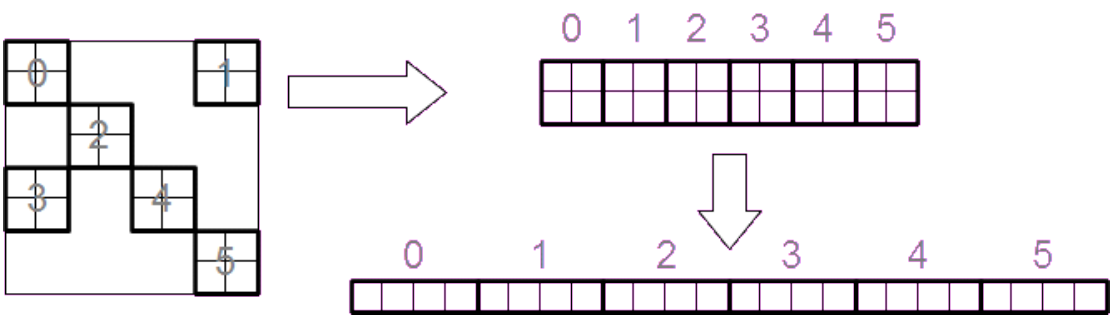


Рисунок 4 - Преобразование м-блочной матрицы в линейный массив А

Массив *J* имеет размерность *m* и в своем *j*-м элементе содержит номер м-блочного столбца элементов, записанных в позициях $(j\,n^2)$. $j+1)\,n^2-1)$ массива *A*. Массив *I* имеет размерность *N*+1. В *i*-й позиции массив содержит индекс, с которого в массиве *J* начинаются элементы, соответствующие *i*-й м-блочной строке матрицы. В последней *N*-й позиции массив содержит размерность вектора *J* (рисунок 5).

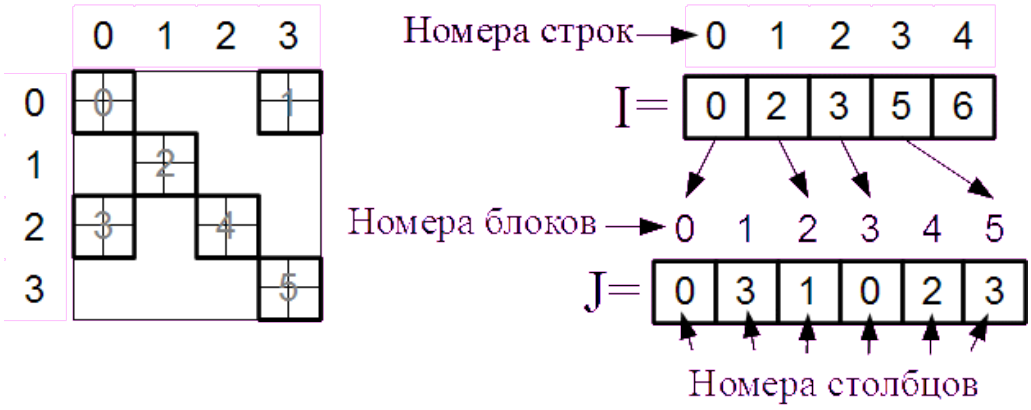


Рисунок 5 - Вспомогательные индексные массивы

Представление набора векторов. Набор из *k* плотных векторов размерности *v*=*Nn* естественно представить в виде *k* массивов размерности *v* или одного массива размерности $k\,v$, в котором векторы расположены один за другим (рисунок 6). Однако такое представление приводит к большому числу нелокальных обращений к памяти процессора и, как следствие, к

снижению производительности. Поэтому используем построчное хранение векторов. В данном формате векторы хранятся в массиве размерностью $k\,\nu$ в следующем порядке: первый элемент первого вектора; первый элемент второго вектора; … первый элемент k-го вектора; второй элемент первого вектора и т. д. (рисунок 7).

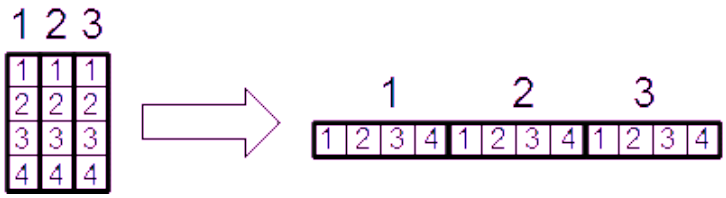


Рисунок 6 – «Естественный» формат хранения набора векторов

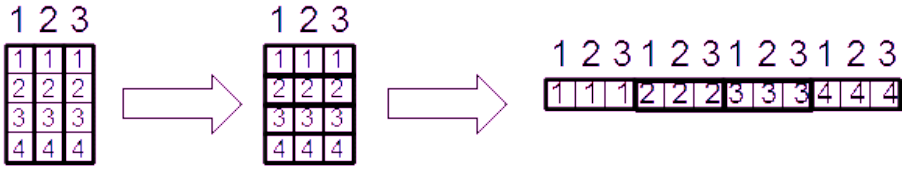


Рисунок 7 - Формат, обеспечивающий локальность обращений к памяти

3.1. Умножение матрицы на набор векторов

В рамках реализации умножения матрицы на набор векторов были разработаны четыре параллельных *CUDA*-алгоритма (алгоритмы *P*11 - *P*14), аналогичный алгоритм *S*1 последовательного умножения, а также алгоритм *T*1 для измерения производительности параллельных вычислений.

Алгоритм *P*11 каждой м-блочной строке ставит в соответствие один *CUDA*-поток. В этом потоке последовательно производится умножение всех элементов м-блочной строки на соответствующие элементы набора векторов. Параллелизм достигается за счет большого числа м-блочных строк. *CUDA*-потоки группируются в *CUDA*-блоки по 1024 потока в каждом (рисунок 8). Требуемое число *CUDA*-блоков определяется автоматически.

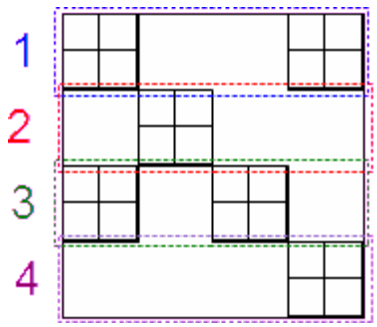


Рисунок 8 - Распределение м-блоков матрицы СЛАУ по *CUDA*-потокам: алгоритм $P11$

Алгоритм $P11$ не использует разделяемую память ГПУ, наиболее прост в реализации, обеспечивает отсутствие конфликтов при записи результатов вычислений в память. С другой стороны, алгоритм не может гарантировать высокую эффективностью из-за большого числа нелокальных несогласованных обращений к памяти из одной рабочей группы (*warp*). При малых размерностях матрицы СЛАУ алгоритм не обеспечивает сбалансированную загрузку ГПУ.

Алгоритм $P12$. В данном алгоритме каждой м-блочной строке ставим в соответствие один CUDA-блок, а каждой комбинации «строка м-блока – м-подвектор» - один CUDA-поток (рисунок 9). Таким образом, для умножения $n \times n$ м-блока на k м-подвекторов требуется nk CUDA-потоков. Умножение производим параллельно, так что в результате получается r блоков размерности $k \times n$ каждый, где r - число м-блоков в одной блочной строке матрицы СЛАУ, и алгоритм требует для работы knr CUDA-потоков. Перед умножением м-блоки загружаем в разделяемую память ГПУ для уменьшения задержек. После умножения элементы всех м-блоков суммируем с использованием атомарных операций (для обеспечения отсутствия конфликтов по записи).

Алгоритм $P12$ также достаточно прост в реализации, значительно лучше алгоритма $P11$ обеспечивает согласованность обращений к памяти и лучшую балансировку загрузки. Однако, в случае $knr > N_s$, где N_s - число *CUDA*-потоков в *CUDA*-блоке, алгоритм дает неверный результат. Данная ситуация

имеет место, например, при размерности блока $n = 10$, числе векторов $k = 10$ и числе м-блоков в блочной строке $r = 11$. Это связано с тем, что в данном алгоритме не предусмотрена организация цикла при превышении числом потоков величины N_s. Для оборудования, использовавшегося в ходе выполнения данной работы, $N_s = 1024$.

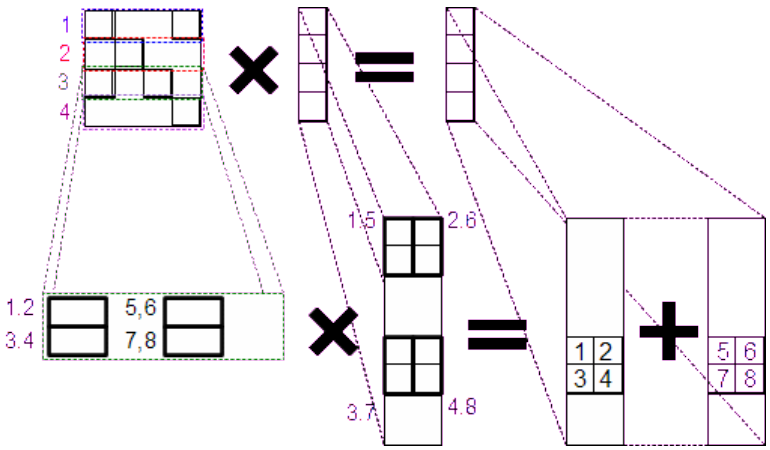


Рисунок 9 - Распределение м-блоков матрицы по *CUDA*-блокам, а также строк м-блоков матрицы и наборов векторов по *CUDA*-потокам: алгоритм *P*12

Алгоритм *P*13 идентичен алгоритму *P*12 за исключением того, что в случае превышения величиной knr значения N_s, он реализует вычисления в несколько итераций до тех пор, пока не будут обработаны все м-блоки строки.

Алгоритм *P*14. В данном алгоритме одному *CUDA*-блоку ставим в соответствие фиксированное число м-блочных строк матрицы. Это позволяет избежать ситуации, когда на один *CUDA*-блок приходится всего один – три м-блока матрицы, так что имеет место простой большого числа потоковых процессоров (рисунок 10). Таким образом, алгоритм *P*14 обеспечивает более равномерную загрузку ГПУ в случае, если в матрице присутствуют м-строки с малым числом блоков.

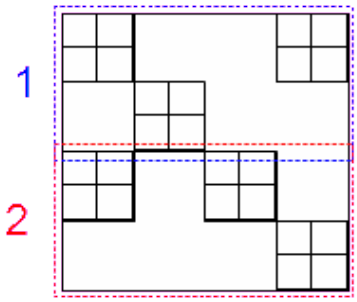


Рисунок 10 - Распределение м-блоков матрицы по *CUDA*-блокам: алгоритм $P14$

Алгоритм $S1$ представляет собой, по сути, классический последовательный алгоритм умножения блочно-разреженной матрицы размерности $(\nu \times \nu) = (N\ n) \times (N\ n)$ на плотную матрицу размерности $(k \times \nu)$.

Алгоритм $T1$ реализует следующие операции: умножение матрицы на вектор последовательным алгоритмом; определение времени исполнение алгоритма; умножение матрицы на вектор исследуемым параллельным алгоритмом; измерение времени исполнения алгоритма; вычисление ускорения параллельного алгоритма.

3.2. Решение блочно-треугольной СЛАУ

Для решения данной задачи также разработаны четыре параллельных *CUDA*-алгоритма (алгоритмы $P21 - P24$), аналогичный последовательный алгоритм $S2$, а также алгоритм $T2$ для измерения производительности параллельных вычислений

Алгоритм $P21$ реализует только те возможности распараллеливания, которые дает структура матрицы СЛАУ вида (4). Для матриц U и L, полученных при факторизации матрицы А, построенной в соответствии с методом вложенных сечений, алгоритм строит бинарное дерево, корень которого соответствует внешнему окаймлению, листья соответствуют блокам A_i, а промежуточные вершины - промежуточным окаймлениям (рисунок 3). Вычисление неизвестных, соответствующих вершинам, лежащим на одном

уровне дерева, производится параллельно. При этом существует зависимость неизвестных i-го уровня от неизвестных $(i+1)$-го уровня для системы с U матрицей и от неизвестных $(i-1)$-го уровня - для системы с L матрицей.

Каждому набору неизвестных, соответствующих какой-либо вершине дерева, назначаем одно *CUDA*-ядро, внутри которого вычисления организованы последовательно. Таким образом, одновременно выполняется число потоков, не большее числа потоковых мультипроцессоров используемого ГПУ.

Параллельные алгоритмы $P22$ - $P24$ построены на основе данного алгоритма.

Алгоритм $P22$ реализует распараллеливание на уровне м-блока. Матрично-векторные операции с м-блоками выполняем параллельно n *CUDA*-потоками в составе *CUDA*-ядра.

Алгоритм $P23$ использует распараллеливание циклов. На каждой итерации, обрабатывающей одну м-блочную строку матрицы СЛАУ, выполняем параллельные независимые умножения м-блоков на м-подвектор переменных. Каждому м-блоку ставим в соответствие один *CUDA*-поток. Поскольку в разных м-строках число блоков различно, при запуске вычислений *CUDA*-ядру назначаем максимально возможное число *CUDA*-потоков. При превышении этим числом числа м-блоков в строке лишние потоки отсекаем, а в противном случае организуем циклическую обработку м-блоков в строке.

Алгоритм $P24$ представляет собой комбинацию алгоритмов $P22$, $P23$. В каждом *CUDA*-ядре в этом случае выделяем $n\,m$ *CUDA*-потоков. Параллельно выполняем матрично-векторные операции с независимыми м-блоками в одной м-строке, а каждую из этих операций дополнительно распараллеливаем по строкам м-блока.

Алгоритм $S2$ реализует поиск неизвестных в соответствии с формулами (5), (6), то есть, по сути, представляет собой м-блочный вариант обратного хода метода исключения Гаусса.

Алгоритм $T2$ аналогичен алгоритму $T1$.

4. Тестирование и исследование эффективности алгоритмических и программных решений

Для проверки корректности принятых алгоритмических и программных решений использованы СЛАУ с известными точными решениями. Результаты тестирования показали погрешность порядка 10^{-14} при использовании чисел с плавающей запятой двойной точности и погрешность порядка 10^{-5} при использовании таких же чисел одинарной точности. Порядок системы варьировался от 10^3 до 10^6.

Вычислительные эксперименты выполнены на ПЭВМ с центральным процессором *Intel Core 2 Duo E*8600 с 2 ГБ ОЗУ и ГПУ *NVIDIA GeForce GTX* 460*SE* с 1 ГБ памяти. При определении ускорения S параллельных алгоритмов в качестве времени исполнения последовательного алгоритма использовано время решения соответствующей задачи на одном ядре ЦПУ. В качестве времени исполнения параллельного алгоритма использовано а) «чистое» время исполнения *CUDA*-ядра, б) время выполнения вместе с временем предварительного копирования матрицы в память ГПУ. Соответственно, для каждого из исследуемых параллельных алгоритмов были получены по два значения ускорения. При использовании этих алгоритмов в программном комплексе *FlowVision* предполагается, что на одну операцию копирования матрицы приходится несколько десятков умножений матрицы на набор векторов или решений СЛАУ и что новые векторы для очередной операции будут копироваться в память ГПУ параллельно с вычислениями. Поэтому далее представляем только ускорения, соответствующие варианту (а) исполнения параллельных алгоритмов.

Поскольку в вычислительных экспериментах в качестве *host*-ЭВМ использована ПЭВМ, функционирующая в мультипрограммном режиме, время исполнения последовательных алгоритмов оказывается случайной

величиной. В результате этого случайными оказываются и ускорения параллельных алгоритмов. На рисунке 11 в качестве примера показаны графики ускорений, полученных при одинаковых параметрах задачи в двух запусках одной из параллельных программ умножения матрицы на набор векторов.

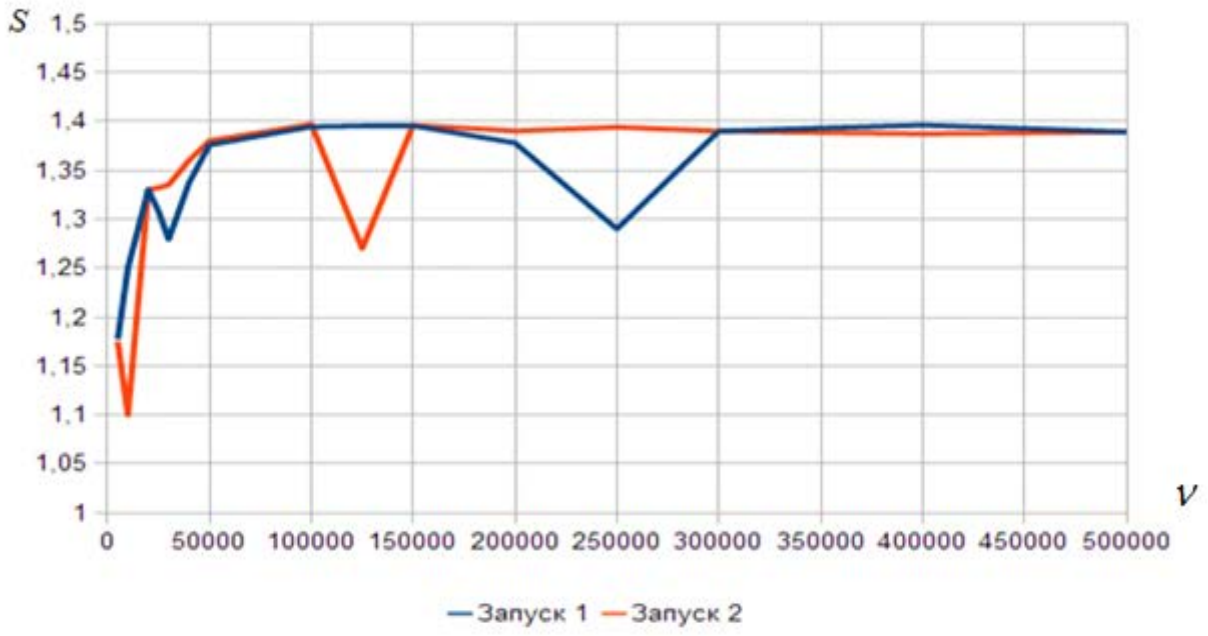


Рисунок 11 – К эффекту случайных флуктуаций ускорения

Для минимизации влияния случайных флуктуаций времени выполнения последовательной программы на ускорение каждый из вычислительных экспериментов проводился многократно, выбросы ускорения игнорировались, а оставшиеся значения усреднялись (рисунок 12).

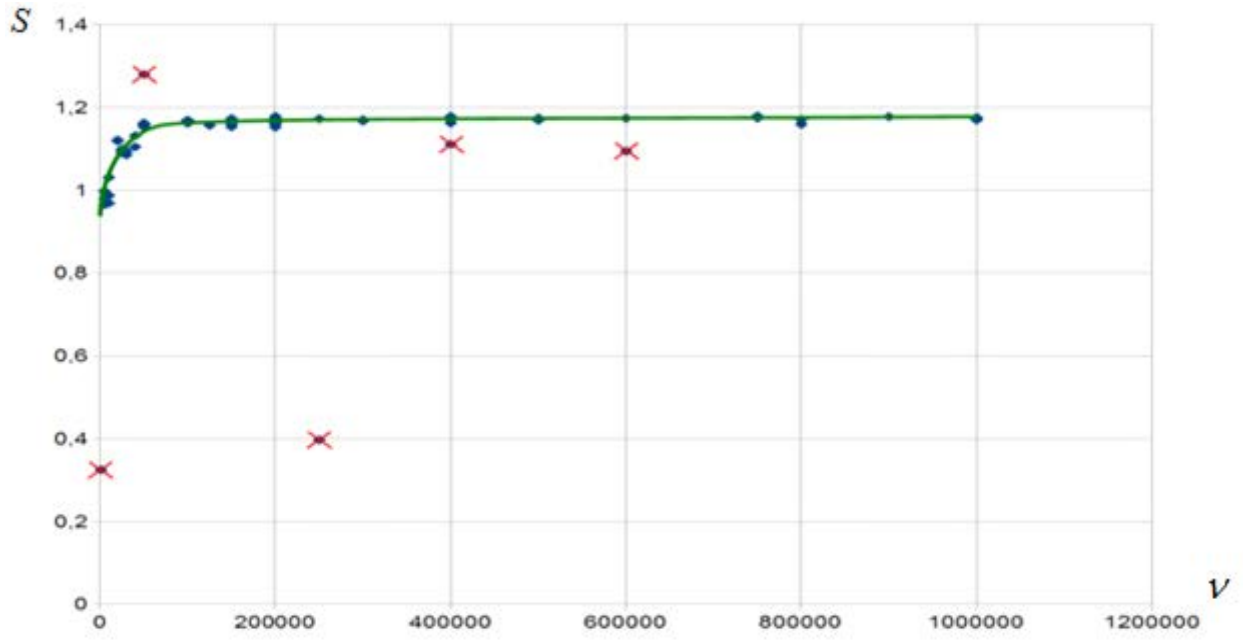


Рисунок 12 – К методике борьбы со случайными флуктуациями ускорения

4.1. Алгоритмы умножения матрицы на набор векторов

Алгоритм $P11$. Результаты исследования эффективности данного алгоритма иллюстрирует рисунок 13. Из рисунка следует, что алгоритм $P11$ в сравнении с алгоритмом $S1$ обеспечивает небольшой выигрыш во времени решения задачи, который уменьшается с увеличением размера м-блока. Такой результат объясняется тем, что алгоритм $P11$ противоречит концепции *SIMD*-вычислений, поскольку в данном случае в разных *CUDA*-потоках одного *CUDA*-блока приходится выполнять разные последовательности вычислений.

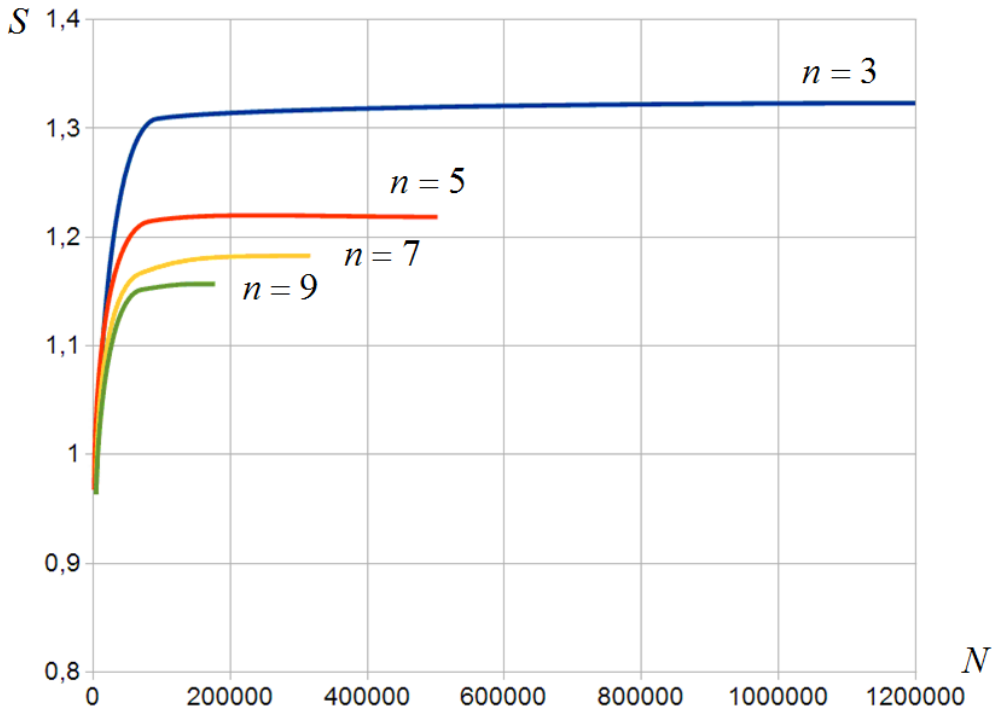


Рисунок 13 - Ускорение вычислений в функции числа м-блоков: алгоритм $P11$; $k = 5$

Алгоритмы $P12$, $P13$. Данные алгоритмы имеют близкую структуру и для них были получены близкие результаты. Поэтому эти алгоритмы рассматриваем вместе.

На рисунках 14, 15 представлены зависимости ускорения от числа м-блоков матрицы СЛАУ для пяти и девяти векторов соответственно. Рисунки показывают, что алгоритмы $P12$, $P13$ обеспечивают значительно более высокую эффективность по сравнению с алгоритмом $P11$. Так, при девяти векторах и размерности м-блока, равной 9×9, ускорения достигает величины ~17 (рисунок 15). В то же время, лишь двукратное ускорение при м-блоках размерности 3×3 и повышение ускорения в 7,5 раз при увеличении размерности м-блоков в три раза до 9×9, говорит о недостаточной загрузке ГПУ при малых размерах м-блоков.

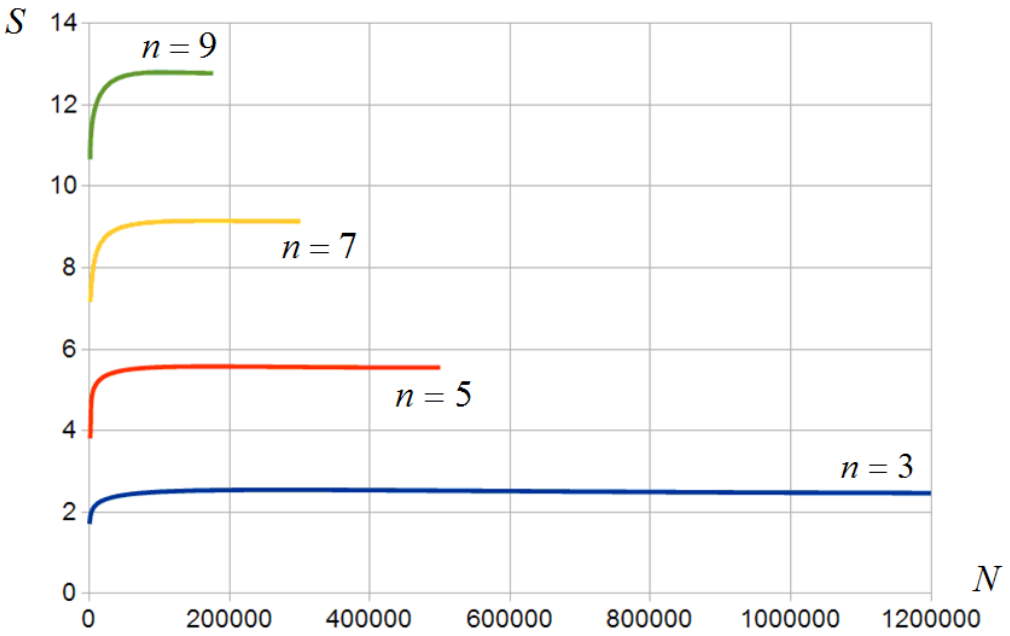


Рисунок 14 - Ускорение вычислений в функции числа м-блоков: алгоритмы $P12$, $P13$; $k = 5$

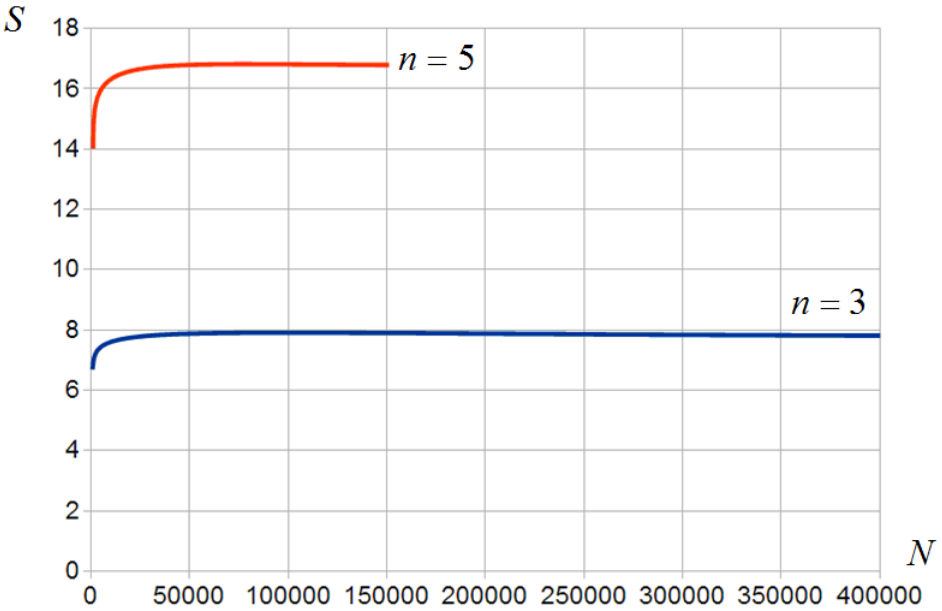


Рисунок 15 - Ускорение вычислений в функции числа м-блоков: алгоритмы $P12$, $P13$; $k = 9$

Алгоритм $P14$. Результаты исследования эффективности данного алгоритма представлены на рисунках 16, 17. Рисунки показывают, что по сравнению с алгоритмами $P12$, $P13$, алгоритм $P14$ обеспечивает рост минимального ускорения примерно в три раза и некоторое падение

максимального ускорения. Эффект объясняется тем, что, с одной стороны, при малом размере м-блоков работа с несколькими его строками в одном *CUDA*-блоке позволяет улучшить балансировку загрузки ГПУ. С другой стороны, при большом размере м-блоков существенными оказываются дополнительные затраты на организацию цикла по строкам.

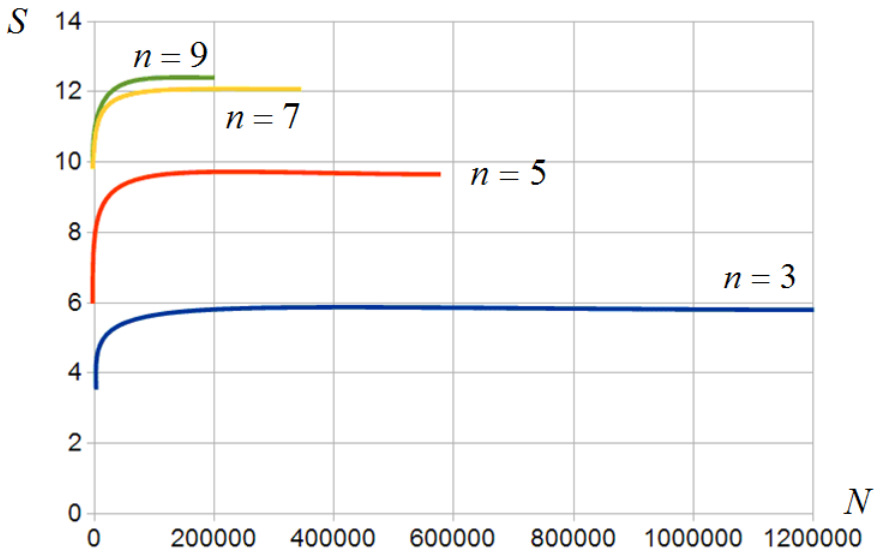


Рисунок 16 - Ускорение вычислений в функции числа м-блоков: алгоритм $P14$; $k = 5$

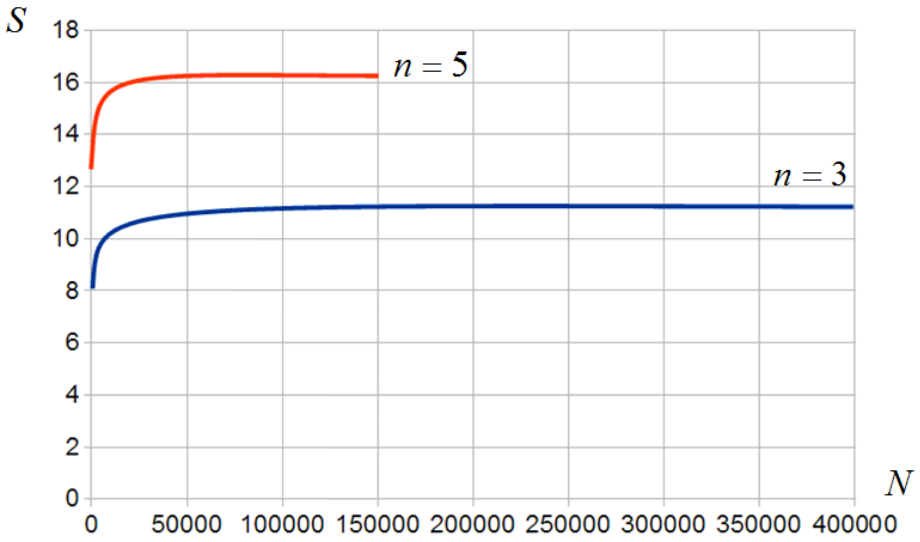


Рисунок 17 - Ускорение вычислений в функции числа м-блоков: алгоритм $P14$; $k = 9$

4.2. Алгоритмы решения блочно-треугольной СЛАУ

Результаты исследования эффективности алгоритма $P21$ не приводим, поскольку данный алгоритм проектировался лишь как основа для построения алгоритмов $P22 - P24$.

Алгоритм $P22$. Некоторые результаты исследования эффективности алгоритма для высоты дерева h, равной четырем и шести, представлены на рисунках 18, 19 соответственно. Рисунок 18 показывает, что в первом случае алгоритм обеспечивает замедление вычислений по сравнению с ЦПУ. Этот факт объясняется малой максимальной шириной дерева, составляющей в данном случае всего восемь узлов. В результате мультипроцессоры ГПУ оказываются загруженными далеко не полностью. Во втором случае, когда высота дерева равна шести, его максимальная ширина составляет 32 узла. Это позволяет загрузить ГПУ более полно и обеспечить меньшее падение его производительности (рисунок 19). Представленные далее результаты исследования эффективности алгоритмов $P23, P24$ относятся только к высоте дерева $h = 6$.

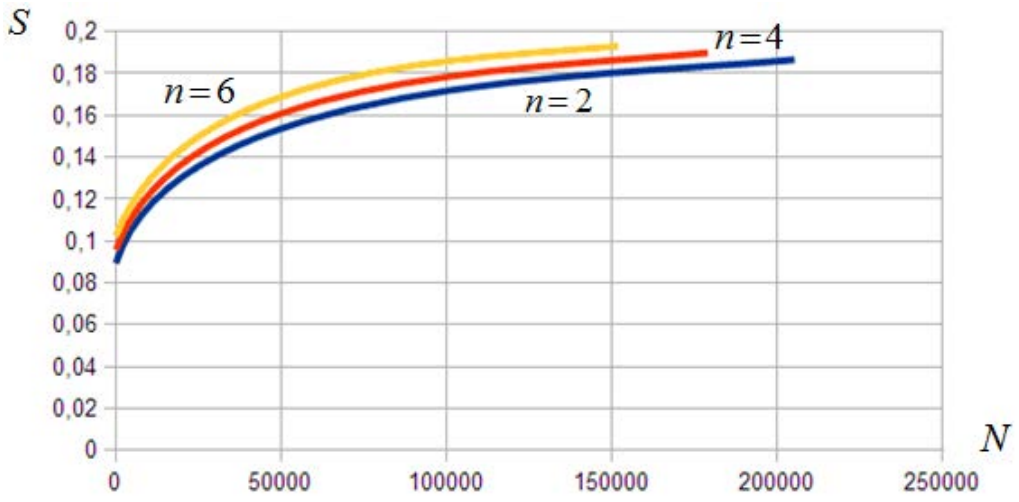


Рисунок 18 - Ускорение вычислений в функции числа м-блоков: алгоритм $P22$; $h = 4$

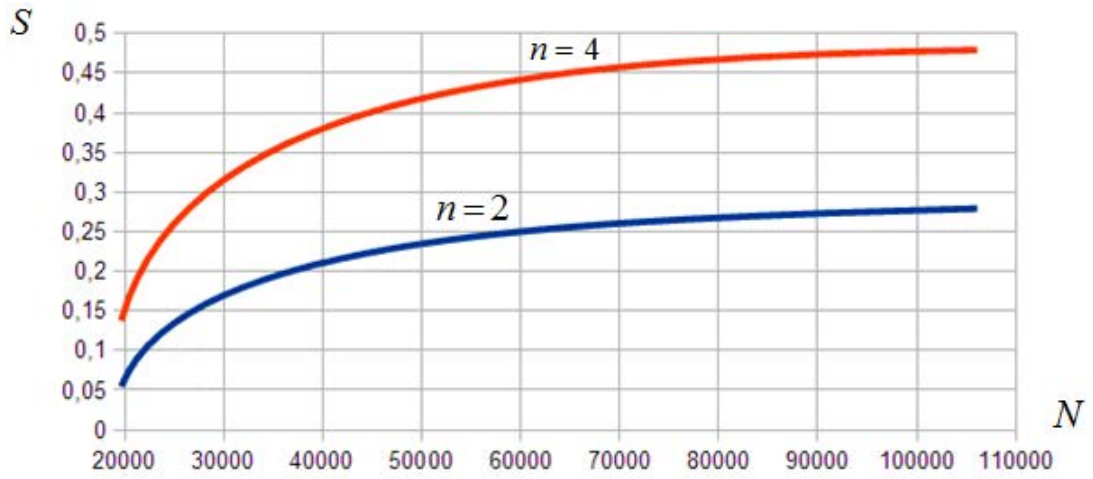


Рисунок 19 - Ускорение вычислений в функции числа м-блоков: алгоритм $P22$; $h = 6$

Алгоритм $P23$. Сравнение результатов, представленных на рисунках 19, 20, показывает, что распараллеливание цикла и распараллеливание матрично-векторных операций дают почти одинаково неудовлетворительные результаты – время решения СЛАУ на ГПУ оказывается более чем в два раза выше того же времени при использовании только ЦПУ.

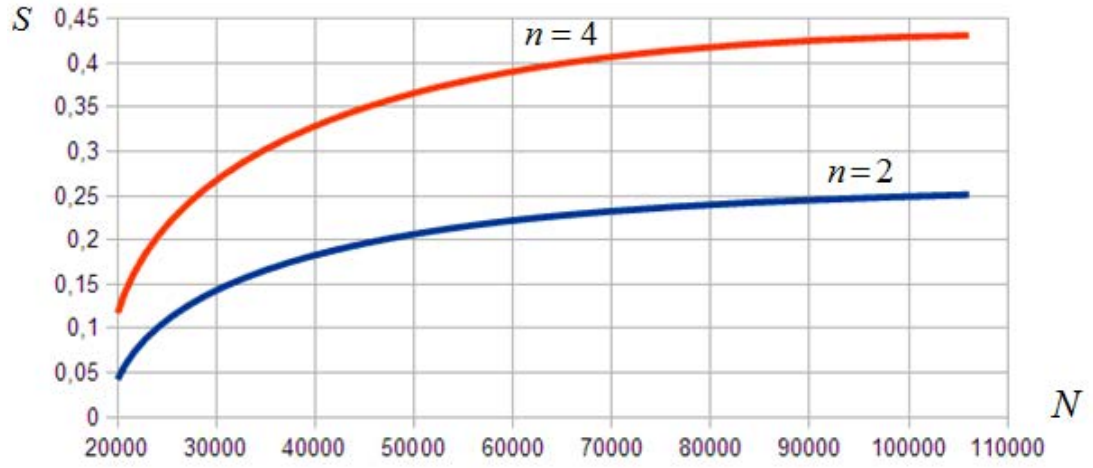


Рисунок 20 - Ускорение вычислений в функции числа м-блоков: алгоритм $P23$; $h = 6$

Алгоритм $P24$. Результаты исследования эффективности данного алгоритма представлены на рисунке 21, из которого следует, что этот

алгоритм обеспечивает наиболее высокую из числа рассмотренных алгоритмов производительность. Рисунок показывает быстрый рост ускорения с ростом размера м-блоков – если при блоках размером 2×2 ускорение лишь немногим превышает единицу, то для (4×4)-блоков оно становится почти двукратным. Таким образом, хотя для реальных задач требуется более высокая производительность, достигнутый результат можно считать удовлетворительным, поскольку в таких задачах, как правило, используются СЛАУ более высокой размерности, размеры м-блоков могут превышать указанные, объем памяти профессиональных ГПУ выше.

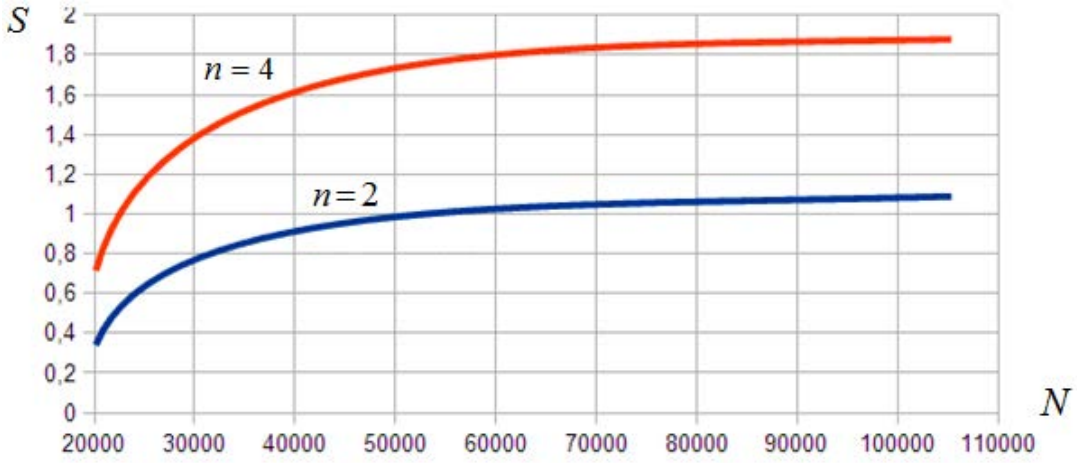


Рисунок 21 - Ускорение вычислений в функции числа м-блоков: алгоритм $P24$; $h = 6$

Заключение

Разработаны параллельные алгоритмы для ГПУ, реализующие основные операции алгоритма решения СЛАУ с предобуславливанием – операция умножения матрицы на набор векторов и операция решения блочно-треугольной СЛАУ. Разработано *CUDA*-программное обеспечение, реализующее указанные алгоритмы. Выполнено широкое исследование эффективности предложенных алгоритмических и программных решений.

Исследование показало достаточно высокую эффективность разработанных алгоритмов и программ для умножения матрицы на набор

векторов - достигнуто ускорение от четырех до шестнадцати раз (в зависимости от числа и размера м-блоков). Результаты исследования позволяют рекомендовать наиболее эффективный из рассмотренных алгоритмов и его программную реализацию к внедрению в промышленную версию комплекса *FlowVision*.

Разработанные алгоритмы и программы, предназначенные для решения блочно-треугольной СЛАУ, показали удовлетворительные результаты, которые позволяют ожидать приемлемого ускорения для практически значимых СЛАУ высокой размерности и при использовании профессиональных ГПУ. Для принятия окончательного решения о внедрении данных алгоритмов и соответствующих программ в комплекс *FlowVision* требуются дополнительные исследования.

В развитие работы авторы планируют разработку и исследование эффективности параллельных алгоритмов и программ, предназначенных для выполнения операций неполной факторизации матрицы и векторных операций. Результаты данной работы позволяют надеяться, что производительность векторных операций, в которых отсутствуют зависимости по данным, будет сопоставима с производительностью умножения матрицы на набор векторов, а производительности операций неполной факторизации, в которой такие зависимости присутствуют – с производительностью решения блочно-треугольной СЛАУ.

Список литературы

1. FlowVision. Режим доступа: http://www.flowvision.ru/)/ (дата обращения 28.12.2012).

2. Трудоншин В.А., Пивоваров Н.В. Системы автоматизированного проектирования. В 9 кн. Кн. 4. Математические модели технических объектов: учеб. пособие для втузов / Под ред. И.П. Норенкова. М.: Высшая школа, 1986. 160 с.

3. Смирнов Е.М., Зайцев Д.К. Метод конечных объемов в приложении к

задачам гидрогазодинамики и теплообмена в областях сложной геометрии // Научно-технические ведомости СПбГПУ. 2004. № 2 (36). С. 70-81.

4. Адинец А., Воеводин В. Графический вызов суперкомпьютерам // Открытые системы. 2008. № 4. Режим доступа: http://www.osp.ru/os/2008/04/5114497/ (дата обращения 28.12.2012).

5. OpenCL. Режим доступа: http://opencl.ru/ (дата обращения 28.12.2012).

6. NVIDEA. Режим доступа: http://www.nvidia.ru/object/cuda_home_new_ru.html (дата обращения 28.12.2012).

7. CUBLAS. Режим доступа: http://docs.nvidia.com/cuda/cublas/index.html (дата обращения 28.12.2012).

8. CUSPARSE. Режим доступа: http://docs.nvidia.com/cuda/cusparse/index.html (дата обращения 28.12.2012).

9. Дядькин А.А., Харченко С.А. Алгоритмы декомпозиции области и нумерации ячеек с учетом локальных адаптаций расчетной сетки при параллельном решении систем уравнений в пакете FlowVision // Материалы Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ» (Новороссийск, 24-29 сентября 2007 г.). М.: Изд-во Московского Университета, 2007. С. 201-206.

10. Kaporin I.E. High Quality Preconditioning of a General Symmetric Positive Definite Matrix Based on its UTU+UTR+RTU decomposition // Numer. Linear Algebra Appl. 1998. Vol. 5. P. 483-509.

11. Сушко Г.Б., Харченко С.А. Многопоточная параллельная реализация итерационного алгоритма решения систем линейных уравнений с динамическим распределением нагрузки по нитям вычислений // Параллельные вычислительные технологии (ПаВТ'2008): труды международной научной конференции (Санкт-Петербург, 28 января - 1 февраля 2008 г.). Челябинск: Изд. ЮУрГУ, 2008. С. 452-457. Режим доступа: http://omega.sp.susu.ac.ru/books/conference/PaVT2008/papers/Short_Papers/035.pdf (дата обращения 28.12.2012).

SCIENTIFIC PERIODICAL OF THE BAUMAN MSTU

SCIENCE and EDUCATION

EL № FS77 - 48211. №0421200025. ISSN 1994-0408

electronic scientific and technical journal

Solving systems of linear algebraic equations by preconditioning on graphics processing units

01, January 2013

DOI: 10.7463/0113.0525190

Karpenko A.P., Chernov S.K.

Bauman Moscow State Technical University, 105005, Moscow, Russian Federation

apkarpenko@mail.ru

lemonlime@yandex.ru

Algorithm of solving systems of linear equations with preconditioning was considered. Parallel algorithms and software that implements basic operations of this algorithm such as multiplication of a sparse matrix and a set of dense vectors along with solving a block-triangular system of linear equations were presented. Results of performance analysis of these algorithms and software were presented in this article. These results showed high efficiency of the developed algorithms and software for multiplication of a matrix and a set of vectors. Computational speedup in this case varied from four up to sixteen times. Algorithms and software designed to solve block-triangular system, showed satisfactory results, which allowed us to expect acceptable computational speedup for high dimension systems occurred in real-world problems and when using professional graphics processing units.

Publications with keywords: CUDA, system of linear equations, preconditioning, graphics processing units

Publications with words: CUDA, system of linear equations, preconditioning, graphics processing units

References

1. *FlowVision*. Available at: http://www.flowvision.ru/)/ , accessed 28.12.2012.

2. Trudonshin V.A., Pivovarov N.V. *Sistemy avtomatizirovannogo proektirovaniia. V 9 kn. Kn. 4. Matematicheskie modeli tekhnicheskikh ob"ektov* [Computer-aided design systems. In 9 books. Book 4. Mathematical models of technical objects]. Moscow, Vysshaia shkola, 1986. 160 p.

3. Smirnov E.M., Zaitsev D.K. Metod konechnykh ob"emov v prilozhenii k zadacham gidrogazodinamiki i teploobmena v oblastiakh slozhnoi geometrii [Finite volume method as applied to the tasks of fluid dynamics and heat transfer in the areas of complex geometry]. *Nauchno-tekhnicheskie vedomosti SPbGPU* [Scientific-technical Bulletin of Saint Petersburg state Polytechnical University], 2004, no. 2 (36), pp. 70-81.

4. Adinets A., Voevodin V. *Graficheskii vyzov superkomp'iuteram* [Graphics challenge to supercomputers]. *Otkrytye sistemy* [Open Systems Journal], 2008, no. 4. Available at: http://www.osp.ru/os/2008/04/5114497/ , accessed 28.12.2012.

5. *OpenCL*. Available at: http://opencl.ru/ , accessed 28.12.2012.

6. *NVIDEA*. Available at: http://www.nvidia.ru/object/cuda_home_new_ru.html , accessed 28.12.2012.

7. *CUBLAS*. Available at: http://docs.nvidia.com/cuda/cublas/index.html, accessed 28.12.2012.

8. *CUSPARSE*. Available at: http://docs.nvidia.com/cuda/cusparse/index.html , accessed 28.12.2012.

9. Diad'kin A.A., Kharchenko S.A. Algoritmy dekompozitsii oblasti i numeratsii iacheek s uchetom lokal'nykh adaptatsii raschetnoi setki pri parallel'nom reshenii sistem uravnenii v pakete FlowVision [Algorithms of domain decomposition and numbering of cells with allowance for local adaptations of the computational grid for a parallel solution of systems of equations in the package FlowVision]. *Materialy Vserossiiskoi nauchnoi konferentsii «Nauchnyi servis v seti INTERNET»* [Proc. of all-Russian scientific conference «Scientific service in network INTERNET»], Novorossiisk, 24-29 September 2007. Moscow, MSU Publ., 2007, pp. 201-206.

10. Kaporin I.E. High Quality Preconditioning of a General Symmetric Positive Definite Matrix Based on its UTU+UTR+RTU decomposition. *Numer. Linear Algebra Appl.*, 1998, vol. 5, pp. 483-509.

11. Sushko G.B., Kharchenko S.A. Mnogopotochnaia parallel'naia realizatsiia iteratsionnogo algoritma resheniia sistem lineinykh uravnenii s dinamicheskim raspredeleniem nagruzki po nitiam vychislenii [Multithreaded parallel implementation of the iterative algorithm for solving linear systems with dynamic load balancing to threads computing]. *Parallel'nye vychislitel'nye tekhnologii (PaVT'2008): trudy mezhdunarodnoi nauchnoi konferentsii*. [Parallel computational technologies (PCT'2008): proc. of international scientific conference], Sankt-Peterburg, 28 January - 1 February 2008. Cheliabinsk, SUSU Publ., 2008, pp. 452-457. Available at: http://omega.sp.susu.ac.ru/books/conference/PaVT2008/papers/Short_Papers/035.pdf , accessed 28.12.2012.