Реализация параллельного метода LU-SGS для задач газовой динамики на кластерных системах с графическими ускорителями

П.В. Павлухин

Московский Государственный Университет им М.В. Ломоносова

В статье предлагается эффективный алгоритм параллельного расчета газодинамических течений на вычислительных кластерных системах с графическими ускорителями. В основе алгоритма лежит неявная схема, приводящая к большим линейным системам с сильно разреженной матрицей, которые решаются методом приближенной факторизации LU-SGS (Lower-Upper Symmetric Gauss-Seidel). Параллельный алгоритм точно воспроизводит работу последовательного и обладает высокой степенью масштабируемости.

1. Введение

Рост производительности вычислительных систем в последние годы все больше и больше определяется увеличением числа вычислительных ядер в узле системы (и числа самих узлов в системе) и во все меньшей степени – увеличением производительности одного ядра. Кроме того, все большее распространение получают системы с новыми SIMD-архитектурами (в частности, системы с использованием графических ускорителей), которые требуют от прикладного программиста дополнительных знаний по устройству таких систем для написания эффективных программ под них. Ясно, что эффективность использования систем с массивно-параллельными архитектурами будет определяться масштабируемостью параллельных алгоритмов и их конкретными реализациями для этих систем. Поэтому особую важность на сегодняшний день приобретают задачи построения таких алгоритмов.

Численные методы газовой динамики делятся на два класса – явные и неявные. Основное достоинство явных схем – относительно простое вычислительное ядро и относительная простота распараллеливания. Но существенным их недостатком является жесткое условие устойчивости, ограничивающее выбор временного шага, и низкая скорость сходимости в схемах с высоким порядком аппроксимации. Это значительно сужает применимость явных схем в вычислительной практике. Неявные методы в большей части лишены этих недостатков и позволяют для ряда задач (в частности, стационарных) значительно сократить время счета. Однако основная проблема, связанная с использованием этих методов, - сложность построения для них эффективных параллельных алгоритмов, особенно под архитектуры современных графических ускорителей. Распространен подход, при котором работа неявного метода в точности соблюдается не на всей расчетной области, а лишь на параллельно обсчитывающихся частях. При этом для улучшения сходимости решения на каждом временном шаге применяются дополнительные итерации. С увеличением числа процессоров, доступных для решения задачи, области, где будет соблюдаться точная работа метода, становятся все меньше. В получаемом решении будет накапливаться все больше ошибок, что может значительно снизить скорость сходимости решения. Ясно, что масштабируемость таких алгоритмов сильно ограничена.

В настоящей работе предлагается параллельный алгоритм для неявного метода LU-SGS (Lower-Upper Symmetric Gauss-Seidel) и его реализация с помощью технологий CUDA и MPI для кластерных систем, основанных на множестве графических ускорителей. Основная его особенность – точное соблюдение работы исходного последовательного алгоритма для всей расчетной области. В отличие от [1], где предлагаемый алгоритм следует работе последовательного лишь в определенных подобластях исходной расчетной области, предлагаемый подход приводит к одинаковому решению не зависимо от того, используется ли одно процессорное ядро, или же расчет ведется на нескольких графических ускорителях.. Следует отметить, что параллельные алгоритмы для рассматриваемого метода, предназначенные для реализации на

традиционных многопроцессорных системах были предложены, в частности, в [6] и [7]. Эффективность LU-SGS относительно других методов в частности оценивается в [2]. Показано, в частности, что данный метод в некоторых задачах обеспечивает скорость сходимости, более чем на порядок превосходящую таковую для явной 3-хэтапной схемы Рунге-Кутты.

2. Метод приближенной факторизации LU-SGS

Опишем коротко метод LU-SGS. Детали можно найти в работах [3, 4, 5]. Рассмотрим систему уравнений Навье-Стокса для сжимаемой жидкости в декартовой системе координат, записанную в форме законов сохранения, которая дискретизируется по пространственным переменным методом конечных объемов. Применяя затем неявную схему интегрирования по времени, в результате получаем систему дискретных уравнений, которая решается методом установления по псевдо-временной переменной с использованием неявной дискретизации и ньютоновских итераций. В результате необходимо решить линейную систему уравнений для определения итерационного инкремента $\delta^s \vec{q}$:

$$(D+L+U)\delta^s \vec{q}_i = -\vec{R}_i^{n+1,s} \tag{1}$$

Где D – блочно-диагональная матрица, а L и U - блочно-диагональные нижняя и верхняя треугольные. Уравнение (1) преобразуется к следующему виду:

$$(D+L)D^{-1}(D+U)\delta^s q_i = -\vec{R}_i^{n+1,s} - LD^{-1}U \tag{2}$$

Метод LU-SGS сводится к приближенной факторизации левой части уравнений (1), которая получается, если в этих уравнениях (2) пренебречь последним слагаемым правой части. Следует заметить, что этот член пропорционален Δt^2, и такое приближение видится вполне оправданным. Факторизованные таким образом уравнения распадаются на две подсистемы:

$$\begin{aligned} (D+L)\delta^s \vec{q}_i^{\,*} &= -\vec{R}_i^{n+1,s} \\ (D+U)\delta^s q_i &= D\delta^s \vec{q}_i^{\,*} \end{aligned} \tag{3}$$

Первая система уравнений в (3) имеет нижне-треугольную блочную матрицу, каждый элемент которой представляется блоком размерностью (5 x 5), а вторая – соответственно верхнетреугольную. Это позволяет эффективно вычислить их решения за два расчетных цикла по ячейкам: первый - в прямом направлении (от первой ячейки к последней), а второй - в обратном. При этом необходимо обращать только диагональные блоки, т. е., матрицы размером (5 x 5). Получающаяся при этом итерационная невязка $\delta^s \vec{q}$ служит для обновления итерационного вектора, после чего процедура (3) повторяется.

Таким образом, если рассматривать алгоритм LU-SGS как объект для распараллеливания, то его последовательный вариант выглядит как два прохода (прямой и обратный) по ячейкам расчётной области. Оба прохода осуществляются по одной последовательности ячеек, но в разных направлениях.

В общем случае значения искомых функций в ячейке на следующем временном слое зависят от всех ячеек расчётной области на текущем временном слое, что затрудняет построение параллельного алгоритма.

3. Параллельная версия LU-SGS

Построение параллельного алгоритма будет проводиться для двумерной расчетной области со структурированной сеткой. При этом данный алгоритм достаточно просто обобщается и на случай трехмерного пространства. В исходной последовательной версии метода фиксируется порядок ячеек (элементов сеточного разбиения), в соответствии с которым происходит прямой и обратный обход расчетной области (т.е. задается очередность вычислений в последовательности ячеек). Для вычисления изменения вектора решения в некоторой ячейке требуются данные от геометрически соседних ячеек, причем вычислительные операции с данными от ячеек-соседей, которые стоят в очереди обхода раньше текущей ячейки, отличаются от таковых для ячеек-соседей, которые расположены в очереди обхода позже. Для корректной работы метода необходимо, чтобы порядок обхода фиксировался неизменным, т.е. чтобы положение в очереди

обхода всех геометрически соседних ячеек («до/после») не менялось относительно текущей ячейки. Однако сам порядок обхода можно определять любым способом – не обязательно, чтобы следующая ячейка была геометрическим соседом предыдущей. По сути, задача построения параллельного алгоритма сводится к выбору такого порядка обхода в параллельном счете, который будет эквивалентен некоторому обходу в последовательном счете.

Поскольку алгоритм строится для системы многих графических ускорителей, распределенных по узлам кластера, можно выделить два уровня параллелизма: inter-GPU – согласованная работа между графическими ускорителями – и intra-GPU – параллельный счет внутри графического ускорителя. Параллельный счет на уровне inter-GPU полностью аналогичен версии многопроцессорного счета, представленной в [6], а именно, исходная расчетная область разбивается на блоки (по числу графических ускорителей), расположенные стык в стык и образующие на плоскости структуру двумерной решетки. Блоки делятся на два множества: два геометрически соседних блока всегда принадлежат разным множествам – таким образом, эти множества образуют «шахматное» разбиение блоков, каждый из них (блоков) считается на отдельном GPU. Внутри каждого блока выполняется обсчет его частей в соответствии с заданным порядком для множества (назовем их “black” и “white”), к которому принадлежит блок.

Для блоков из множества “black” вычисления проводятся в следующем порядке:

1. Из блоков отсылаются соседям в white копии К граничных частей ячеек (Рис 1. Слева - последовательность вычислений в блоках из множеств “black” и “white”, справа - обход области с 2x2 блоками., слева), которые являются соответствующими геометрическими соседями для ячеек из блоков white, при этом отсылаемые ячейки для white не являются обсчитанными. В это же время в блоках black также запускается обход по всем ячейкам, кроме граничных и кроме ячеек, являющихся соседними к граничным внутри блока, т. е. обход по всем ячейкам, кроме «двойного» периметра ячеек (поскольку модифицируются данные в необсчитанных еще ячейках, находящихся позже в очереди обхода).

2. В блоках black граничные ячейки обновляются из присланной копии К (операция корректна, т.к. никаких действий над граничными ячейками в black еще не производилось).

3. В блоках black по флагу проверяется, присланы ли ячейки из соседних white-блоков, которые уже являются обсчитанными, и обновились ли граничные ячейки в блоке. После подтверждения этого в блоках black выполняется обход по оставшимся приграничным ячейкам из «двойного» периметра блока с обращениями к полученным ячейкам из white.

Для множества “white” счет идет в таком порядке:

1. Во всех блоках из white запускается обход по первой половине внутренних ячеек, то есть обход будет сделан только по части внутренних ячеек. В момент счета из соседних блоков black присылаются необсчитанные ячейки, соседние к граничным в блоках white.

2. Затем в white по флагу проверяется, получены ли ячейки из соседних блоков (к этому моменту они будут доставлены, если время счета указанной выше части ячеек в white будет больше времени передачи ячеек из black), и выполняется обход по граничным ячейкам, причем используется копия К ячеек полученных из black, в этой копии также обновляются ячейки (полученные из black, они еще не обсчитаны). После завершения этого обхода из white соседям в black отправляются граничные ячейки блока white, которые являются соседями для соответствующих ячеек из black, и обновленная копия К ячеек из black.

3. Наконец, в блоках white выполняется обход оставшейся второй половины внутренних ячеек. Обратный обход в блоках строится в соответствии с прямым: в black он начинается с "двойного периметра" приграничных ячеек в обратном порядке и заканчивается на внутренней оставшейся их части (так же в обратном порядке); в white сначала выполняется обход второй половины внутренних ячеек, затем граничных и, наконец, оставшейся первой части (в обратном порядке).

Построенный выше алгоритм эффективен при параллельном счете блоков с примерно равным количеством ячеек, поскольку пересылка ячеек совмещена со счетом: передача данных идет одновременно с вычислениями. Корректность алгоритма подтверждает эквивалентный обход всей расчетной области, для которого результаты счета последовательного алгоритма совпадают с таковыми для рассмотренного параллельного. Он строится следующим образом: сначала выполняется обход внутренней части (без «двойного периметра») блоков “black”; затем в каждом блоке “white” обходится сначала первая половина внутренних ячеек, а потом гранич-

ная часть; далее, во всех блоках “black” обходятся «двойные» граничные части и, наконец, в блоках “white” обходятся оставшиеся вторые половины внутренних ячеек. Пример такого обхода для 2x2 блоков показан на Рис 1**.** Слева - последовательность вычислений в блоках из множеств “black” и “white”, справа - обход области с 2x2 блоками., справа (числами обозначен порядок в очереди обхода).

Выше был изложен алгоритм распределения вычислений между графическими ускорителями, следующий уровень – intra-GPU – счет внутри одного ускорителя. Архитектура GPU подразумевает одновременный счет многих ячеек, но здесь возникает проблема: как организо-

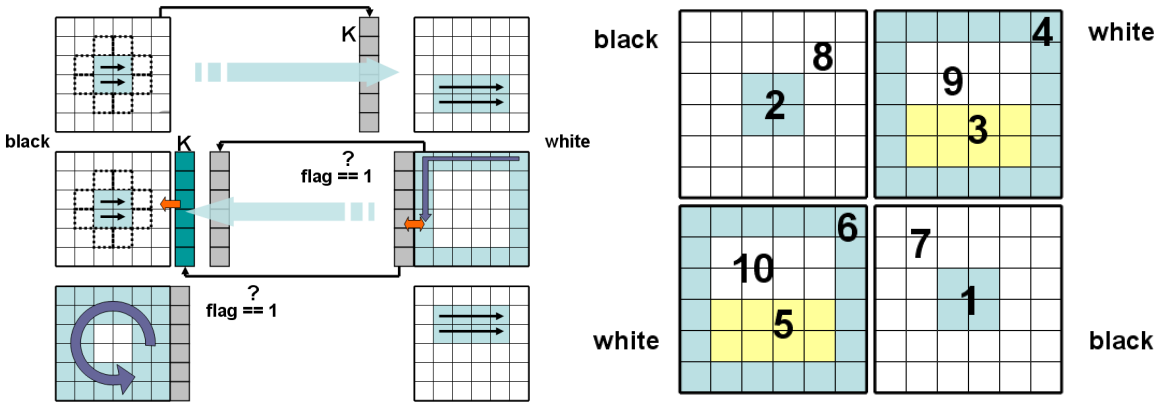

Рис 1. Слева - последовательность вычислений в блоках из множеств “black” и “white”, справа - обход области с 2x2 блоками.

вать очередь обсчета ячеек, чтобы положение в ней геометрически соседних ячеек оставалось неизменным относительно текущей, т.е. нужно гарантировать, что при обращении к каждой соседней ячейке она всегда была бы уже обсчитана (находится в очереди раньше текущей) или еще не обсчитана (в очереди – позже текущей). Средств глобальной синхронизации, которые помогли бы решить данную проблему, в современных GPU нет. Чтобы обеспечить корректность счета, можно разбить параллельный обсчет подобласти на два последовательных – сначала по одной части подобласти, затем – по другой. Ответ на вопрос о нужном разбиении подобласти по сути являет собой свойство автомодельности параллельного алгоритма – разделение ячеек внутри подобласти (внутренней или граничной части блока) на два множества происходит аналогично разбиению блоков исходной расчетной области: две геометрически соседних ячейки всегда принадлежат разным множествам. Таким образом, два этих множества вновь образуют «шахматное» разбиение ячеек. В итоге, счет каждой подобласти блока – внутренней или граничной его части – выполняется в два последовательных этапа: сначала на GPU запускается счет ячеек из “black” (при этом все соседние ячейки будут из множества “white”, которые в данной подобласти будут всегда еще не обсчитанными), затем, после его завершения, запускается счет ячеек из “white” (все соседние ячейки будут уже из “black”, которые в данной подобласти будут всегда уже обсчитанными). Обращение к ячейкам из другой подобласти всегда будет выполняться как к обсчитанным или необсчитанным – порядок в этом случае определяется порядком обсчета подобластей в блоке.

4. Реализация

Программный код был написан с использованием MPI и CUDA ToolKit 4.0. Для увеличения производительности счета и скорости копирования данных между графическими ускорителями внутренние ячейки блока хранились в виде двумерного массива, а граничные – в виде одномерного (в направлении обхода ячеек – сначала по «нижней», затем по «правой», «верхней» и «левой» границам блока). Для совмещения счета и передачи ячеек во времени использовались неблокирующие функции приема/передачи MPI и несколько CUDA-«потоков» (Stream): в одном из них выполнялось вычислительное ядро (kernel), в других параллельно шло копирование

данных. Для синхронизации вычислений использовались функции MPI_Wait и функции работы с событиями (event) в API CUDA. Все вычисления проводились на GPU, ядро центрального процессора использовалось лишь для копирования данных между оперативной памятью и памятью ускорителя и для обмена данными между процессами посредством MPI.

5. Результаты тестов

В качестве тестовой была выбрана задача о коническом теле, мгновенно помещенном в однородный сверхзвуковой поток газа с числом Маха M=1.6 (Рис. 2**.** Распределение плотности в задаче о коническом теле при t = 0.1 с.). Решалась нестационарная

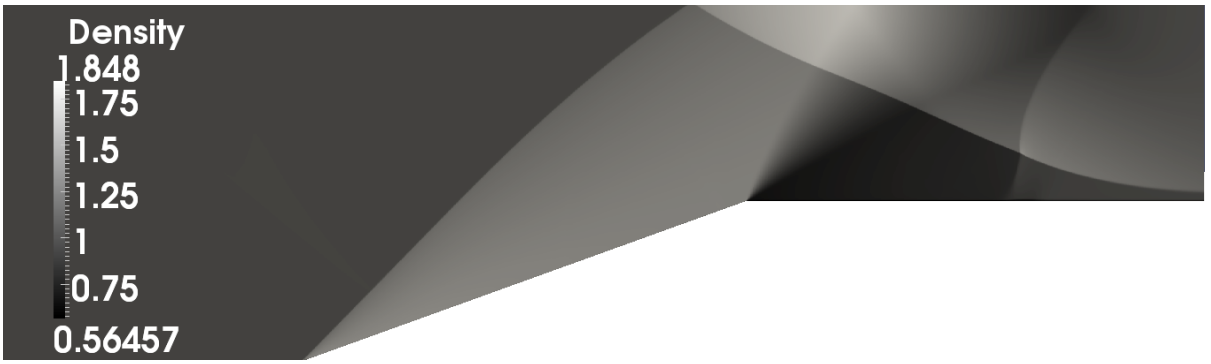

Рис. 2. Распределение плотности в задаче о коническом теле при t = 0.1 с.

задача об образовании ударно-волновой структуры в результате взаимодействия потока с поверхностью тела. Измерения проводились на суперкомпьютере К-100 в ИПМ им М. В. Келдыша. В каждом его узле установлены 3 ускорителя Nvidia Tesla C2070 и 2 процессора Intel Xeon X5670. Во всех измерениях использовалась двойная точность вычислений.

В первом тесте сравнивалась скорость работы одного ядра CPU и одного графического ускорителя. Расчет задачи с сеточным разрешением 3600x1080, 100 шагов по времени, длился на одном ядре 3580с, на одном GPU – 250с. Таким образом, использование одного ускорителя дает прирост производительности в 14.3 раза. Анализ кода с помощью дизассемблера (cuobjdump) и профилировщика CUDA показал, что производительность на GPU ограничена пропускной способностью памяти. Дело в том, что вычислительное ядро программы довольно сложное, и доступных 63 регистра на поток (thread) в архитектуре Fermi недостаточно, чтобы хранить все локальные переменные, поэтому возникает registers spilling – часть временно не используемых переменных записываются (а позднее считываются) в медленную глобальную память. В данной задаче spilling занимал бóльшую пропускную способность памяти GPU.

В следующем тесте исследовалась масштабируемость времени счета на одном GPU при использовании разных разрешений сеток. Базовое разрешение – 450x135 (≈ 60 тыс ячеек) заменялось на другие, которые содержали в 4^n (n = -2, ... 2, 3)раз больше/меньше ячеек. Результаты измерений представлены на Рис. 3 Слева - время счета на одном GPU с разными разрешениями расчетной сетки, справа - время счета с расчетной сеткой 3600x1080 с разным числом GPU., слева. Видно, что с увеличением разрешения относительно базового время счета растет линейно с увеличением числа ячеек расчетной области, однако эта линейность нарушается, когда разрешение сетки уменьшается: время счета уменьшается лишь в ≈ 2.5 раза при уменьшении разрешения в 4 раза (1/4 x – 15390 ячеек), при дальнейшем уменьшении разрешения (1/16 x – 3944 ячейки) счет сократился только в 1.75 раза. Это связано с тем, что в расчетных блоках с малым числом ячеек значительный вклад при счете граничной части начинает вносить время доступа к соседним ячейкам из внутренней части блока – запросы к ним в глобальную память GPU не объединяются в меньшее число транзакций (non-coalesced access), поскольку они происходят из соседних (в одномерном массиве) ячеек граничной части к геометрически соседним, но разреженным (разнесенным) в двумерном массиве ячейкам внутренней части.

Основной тест, демонстрирующий масштабируемость алгоритма с увеличением числа GPU, показан на Рис. 3 Слева - время счета на одном GPU с разными разрешениями расчетной

сетки, справа - время счета с расчетной сеткой 3600x1080 с разным числом GPU. справа. Расчет производился на сетке 3600x1080, всего 100 шагов по времени. Непропорциональное уменьшение времени счета при переходе от одного к двум GPU связано с появляющимися накладными расходами по обмену данными между графическими ускорителями. При дальнейшем же увеличении числа GPU наблюдается практически линейная масштабируемость времени счета. Так, например, при задействовании в 32 раза большего числа GPU (64 вместо 2) время счета сократилось в 26 раз.

6. Заключение

В работе предложен новый параллельный алгоритм, реализующий метод LU-SGS для задач газовой динамики на кластерной вычислительной системе распределенных графических ускорителей. Показана корректность этого алгоритма, описана его реализация с помощью технологий MPI и CUDA. Результаты вычислительных экспериментов подтвердили эффективность алгоритма на задачах с большими расчетными сетками, вплоть до таких, когда на один GPU

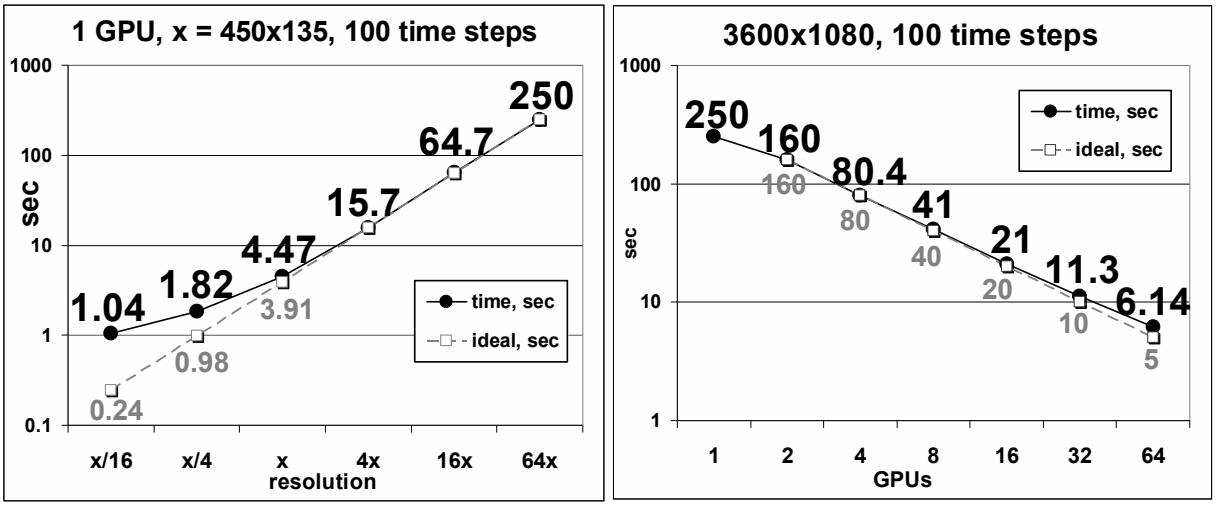

Рис. 3 Слева - время счета на одном GPU с разными разрешениями расчетной сетки, справа - время счета с расчетной сеткой 3600x1080 с разным числом GPU.

приходится порядка 60 тыс ячеек. Масштабируемость счета на таких задачах оказалась близкой к линейной. Такая эффективность достигается за счет совмещения счета и передачи данных между ускорителями: при достаточно больших разрешениях расчетной сетки время обмена данными полностью перекрывается временем счета. По результатам тестов производительность GPU более чем на порядок превосходит производительность процессорного ядра, что свидетельствует о целесообразности применения графических ускорителей в рассматриваемых задачах. В дальнейшем планируется реализовать алгоритм для трехмерного случая и провести оптимизацию самого вычислительного ядра под архитектуру GPU.

Литература

1. D. Sharov, H. Luo, J.D. Baum, R. Loehner Implementation of unstructured grid GMRES-SGS method on shared-memory, cache-based parallel computers // AIAA-2000-927 – 38th Aerospace Sciences Meeting and Exhibit. – 2000.
2. Y. Sun, Z.J. Wang, Y. Liu, C.L. Chen Efficient Implicit LU-SGS Algorithm for High-Order Spectral Difference Method on Unstructured Hexahedral Grids // AIAA 2007-313 – 45th Aerospace Sciences Meeting and Exhibit – 2000.
3. A. Jameson, E. Turkel Implicit schemes and LU decomposition // Math.of Comp., v.37, № 156, pp.385-397, 1981.

4. I. Menshov, Y. Nakamura On implicit Godunov's method with exactly linearized numerical flux //Computers & Fluids, 29 (6), pp. 595 – 616, 2000.
5. I. Menshov, Y. Nakamura Hybrid Explicit-Implicit, Unconditionally Stable Scheme for Unsteady Compressible Flows //AIAA Journal,vol. 42, № 3, pp. 551-559, 2004.
6. П.В. Павлухин, И.С. Меньшов. Эффективная реализация метода LU-SGS для задач газовой динамики. //Научный вестник МГТУ ГА, Т.165, 3, с.46 - 55, 2011
7. Семёнов И.В., Ахмедьянов И.Ф., Уткин П.С. Разработка вычислительного комплекса для решения двух- и трёхмерных задач газодинамики реагирующих течений на многопроцессорных ЭВМ // Материалы 6 Международного научно-практического семинара «Высокопроизводительные параллельные вычисления на кластерных системах». – 2007. – Т. 2. – С. 138 – 145.