УДК 519.6

ПРИМЕНЕНИЕ СРЕДСТВ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ МЕХАНИКИ ЖИДКОСТИ И ГАЗА НА МНОГОПРОЦЕССОРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

К. Н. Волков[1]

Рассматриваются вопросы, связанные с решением задач механики жидкости и газа на многопроцессорных вычислительных системах. Обсуждаются методы декомпозиции расчетной области, способы распределения данных по процессорам и особенности параллельной реализации численных методов, применяемых для решения подзадач. Приводятся примеры решения ряда частных задач гидрогазодинамики на структурированных и неструктурированных сетках и исследуется производительность программного кода в зависимости от размера задачи и числа процессоров.

1. Введение. Существует широкий круг задач механики жидкости и газа, требующих для своего решения значительно бóльших вычислительных ресурсов, чем может предоставить персональный компьютер. Многим их них необходимы высокое быстродействие, а также обработка и хранение большого объема информации, что предъявляет повышенные требования к оперативной памяти. Такие ресурсы имеются в распоряжении высокопроизводительных вычислительных систем или суперкомпьютеров (High Performance Computing, HPC).

В задачах механики жидкости и газа повышенные требования к производительности и памяти обусловлены сложными нелинейными моделями среды, описываемыми большим числом уравнений, пространственным характером задачи и нестационарностью протекающих процессов.

Решение задачи на многопроцессорных вычислительных системах связано с возможностью создания программы, выполняющей несколько подзадач одновременно и независимо друг от друга (параллельно) с последующим обменом информацией между ними. Технически это достигается за счет объединения нескольких процессоров, работающих над общей или распределенной памятью. На практике переход от однопроцессорной конфигурации к многопроцессорной системе из n процессоров не приводит к сокращению времени счета в n раз. Вследствие разделения ресурсов, добавление нового процессора может вызывать замедление работы остальных процессоров и снижение производительности. Решение задачи в параллельном режиме требует правильного выбора модели распараллеливания программного кода, тесно связанной с аппаратными и системными средствами параллельной обработки данных.

Подход к численному решению задач механики жидкости и газа при помощи современных вычислительных технологий и методов параллельного программирования основан на геометрической декомпозиции расчетной области на подобласти, количество которых равняется числу процессоров, расчете каждым процессором своей подобласти и обмене данными между ними на каждом шаге по времени. Связь подобластей обычно осуществляется при помощи введения фиктивных ячеек (их количество зависит от шаблона разностной схемы), которые находятся за границами подобластей и не обрабатываются кодом. Показатели производительности зависят от метода декомпозиции, способа распределения данных по процессорам и реализации численных методов, применяемых для решения подзадач.

В данной работе рассматривается ряд вопросов, связанных с решением задач механики жидкости и газа на многопроцессорных вычислительных системах с общей и распределенной памятью. Обсуждаются методы геометрической декомпозиции расчетной области и балансировки нагрузки процессоров, способы распределения данных по процессорам, а также особенности параллельной реализации численных методов, применяемых для решения подзадач. Приводятся примеры решения ряда частных задач гидрогазодинамики (решение уравнения Пуассона, течение в каверне с подвижной крышкой, течение в межлопаточном канале газовой турбины) на структурированных и неструктурированных сетках и исследуется производительность программного кода в зависимости от размера задачи и числа процессоров.

2. Схема решение задачи. Методологический подход к решению задачи при помощи средств параллельной обработки данных заключается в реализации следующих этапов [1].

[1] Балтийский государственный технический университет “Военмех” им. Д. Ф. Устинова, физико-механический факультет, 1-я Красноармейская ул., д. 1, 190005, Санкт-Петербург; e-mail: dsci@mail.ru

1. Декомпозиция (partitioning). Разбиение задачи вычислений и обработки данных на минимальные независимые подзадачи.

2. Коммуникации (communication). Определение структуры подзадач и установление необходимых связей между ними.

3. Кластеризация (agglomeration). Оценка структуры подзадач и их объединение с целью минимизации коммуникаций и повышения производительности.

4. Распределение (mapping). Назначение подзадач конкретным процессорам и обеспечение их равномерной нагрузки.

Представленная схема отражает отсутствие строго формализованного подхода к параллельному программированию. На шагах 1 и 2 выделяются параллельные ветви вычислительного алгоритма и закладывается его масштабируемость (гибкость по отношению к числу процессоров). Реализация шагов 3 и 4 требует учета архитектуры многопроцессорной системы для обеспечения мобильности и локальности алгоритма.

Различают способы геометрической (domain decomposition) и функциональной (functional decomposition) декомпозиции. Метод функциональной декомпозиции иногда разделяют на физико-математический и технологический [2].

Геометрическая декомпозиция заключается в разделении области интегрирования на множество подобластей и расчете решения в каждой подобласти с последующей сшивкой решений из подобластей.

Физико-математическая декомпозиция предполагает разбиение исследуемого физического процесса по составляющим его подпроцессам и разбиение алгоритма решения задачи на ряд алгоритмов решения составляющих его подзадач.

Технологическая декомпозиция подразумевает сегментирование программы на ряд физико-математических задач, каждая из которых состоит из алгоритмически независимых подзадач.

На практике процедура распараллеливания применяется к циклам, когда в качестве отдельных подзадач выступает тело цикла, выполняемое для различных значений переменной цикла.

Для параллельного решения задач механики жидкости и газа с неподвижными границами обычно используется метод геометрической декомпозиции с введением внутренних граничных условий. Расчетная область Ω разделяется на несколько подобластей по числу процессоров. Подобласти имеют ряд фиктивных ячеек (ghost cell), которые перекрываются с ячейками соседних подобластей и хранят граничные значения соседних блоков (рис. 1). Каждая подобласть обрабатывается одним процессором, а их взаимодействие требуется только при переходе к следующему временно́му слою. Связь подобластей осуществляется при помощи копирования значений искомых функций в фиктивные ячейки.

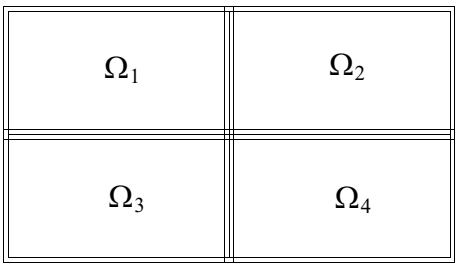

Рис. 1. Геометрическая декомпозиция расчетной области

Для получения решения в момент времени t_{n+1} в каждой из подобластей производятся вычисления в соответствии с разработанной разностной схемой. При декомпозиции, кроме внешних границ, возникают внутренние границы, разделяющие различные подобласти. Внутри каждой подобласти существуют узлы, для вычисления параметров потока в которых необходимо знать их значения в узлах, не принадлежащих данной подобласти.

Для того чтобы сумма решений из подобластей совпадала с решением, полученным на основе последовательного кода, для явных конечно-разностных схем необходимо и достаточно правильно задать граничные условия на внутренних границах. В качестве внешних (фиктивных) ячеек, находящихся за границами данной подобласти, берутся ячейки соседнего процесса, прилегающие к границе подобласти с другой стороны.

Шаблон разностной схемы определяет, какие и сколько внешних узлов оказываются необходимыми для получения корректного решения. В случае трехточечного шаблона для сшивки решений используются величины из одного, а в случае пятиточечного — из двух соседних узлов с каждой стороны [3]. Возникает необходимость в пересылке одного или двух слоев ячеек с каждой стороны подобласти (в зависимости от шаблона), граничащей с подобластями соседних процессов. В процессоре формируется буфер, в который заносятся значения, вычисленные в граничных ячейках подобласти. На каждом шаге по времени процессор обновляет буферные данные путем обменов с процессорами, обрабатывающими соседние подобласти.

Для минимизации коммуникаций между процессорами и обеспечения балансировки их нагрузки область разбивается таким образом, чтобы минимизировать площадь касания соседних подобластей и

уравновесить количество ячеек в них. Например, область $\{1,16\} \times \{1,10\} \times \{1,10\}$ разбивается на подобласти $\{1,8\} \times \{1,10\} \times \{1,10\}$ и $\{9,16\} \times \{1,10\} \times \{1,10\}$.

3. Хранение данных. Важное значение имеют распределение данных по процессорам, синхронизация вычислений и маршрутизация данных, неудачная организация которых приводит к неверному счету или большим накладным расходам из-за задержек счета, связанных с ожиданием и пересылкой данных и бездействием процессоров в некоторых сегментах.

Возможны следующие способы размещения массива данных в памяти процессоров.

1. Массив целиком хранится в каждом процессоре. В каждом процессоре хранится своя копия массива, в которой модифицируется только часть элементов. Задача сводится к вычислению начального и конечного значения переменной цикла для каждого процессора. В конце вычислений требуется сборка модифицированных частей массивов со всех процессоров.

2. Массив распределяется по процессорам. Требуется процедура установления связи индексов локального массива в некотором процессоре с индексацией всего массива. При отсутствии некоторых элементов массива в некотором процессоре необходима их пересылка из других процессоров.

Хранение копий массива во всех процессорах уменьшает накладные расходы на пересылку данных, однако не дает выигрыша в плане объема решаемой задачи и создает сложности синхронизации копий массива при независимом изменении его элементов различными процессорами. Распределение массива по процессорам позволяет решать более объемные задачи, но требует минимизации пересылок данных.

4. Способы разбиения. Сетку размерности $n_x \times n_y \times n_z$ можно разделить на подобласти различными способами. Применяются стратегии послойной (построчной или поколоночной) декомпозиции и поблочного разбиения области.

Самый простой способ декомпозиции состоит в разделении области на слои вдоль одной из координатных осей по числу процессоров. При наличии n_p процессоров каждая из подобластей имеет размерность $(n_x/n_p) \times n_y \times n_z$. Объем данных (в байтах), предназначенных для пересылки между двумя процессами на каждом шаге интегрирования по времени, равняется $B = 8q \times 2n_g \times n_y \times n_z$, где q — количество параметров, рассчитываемых в узле сетки (8 байт на число), n_g — число слоев фиктивных ячеек с каждой стороны подобласти ($n_g = 1$ для трехточечного и $n_g = 2$ для пятиточечного шаблона).

С точки зрения количества передаваемых данных между процессорами деление вдоль одной из осей не является оптимальным. При увеличении числа процессоров толщина слоя, выделенного каждому из них, убывает, в то время как количество передаваемых данных не изменяется. В конечном счете это приводит к эффекту насыщения, когда рост производительности замедляется и со временем выходит на постоянный уровень, при котором добавление новых процессоров не влечет за собой увеличения производительности. Кроме того, с ростом числа процессоров увеличиваются затраты времени на синхронизацию временно́го шага, вследствие чего добавление новых процессоров может приводить к падению производительности.

При двумерной декомпозиции область разделяется на слои вдоль одной из координатных осей, после чего получившиеся слои разделяются вдоль другой оси. Пусть p_x и p_y — число разбиений вдоль осей x и y. Тогда объем данных, предназначенных для передачи от процесса к процессу, составляет

$$B = 8q \times 4n_g \times n_z \times \left(\frac{n_x}{p_x} + \frac{n_y}{p_y}\right).$$

При таком способе разбиения количество передаваемых данных убывает с увеличением числа процессоров. Эффект насыщения также присутствует, что связано как с особенностями сети передачи данных (скорость передачи максимальна при пересылке больших объемов данных), так и с потерями на синхронизацию временно́го шага. Однако предел насыщения отстоит дальше, чем в случае разбиения вдоль одной из осей, что позволяет использовать большее число процессоров.

Трехмерная декомпозиция проводится таким же способом, что и двумерная. Объем данных, предназначенных для обмена между процессорами, получается равным

$$B = 8q \times 6n_g \times \left(\frac{n_x}{p_x}\,\frac{n_y}{p_y} + \frac{n_x}{p_x}\,\frac{n_z}{p_z} + \frac{n_y}{p_y}\,\frac{n_z}{p_z}\right).$$

Насыщение наступает при большем числе процессоров, чем при разбиении вдоль одной или двух осей.

5. Характеристики производительности. Для количественной оценки эффективности распараллеливания используются следующие характеристики.

1. Ускорение (speedup) $S_p = t_1/t_p$, где t_1 — время выполнения программы на одном процессоре, t_p — время выполнения программы в системе из p процессоров.

2. Коэффициент (эффективность, efficiency) масштабируемости $E_p = S_p/p$, причем $E_p = 1$ только тогда, когда достигается максимальное ускорение (при $S_p = p$).

3. Стоимость (cost) $C = pt_1/S_p = t_1/E_p$, связанная с ускорением и коэффициентом масштабируемости.

Для получения приближенной оценки производительности учитываются наиболее трудоемкие операции, такие как умножение двух чисел и пересылка данных между процессорами. Время t_m, необходимое для умножения пары чисел, на порядок превосходит время t_a, требуемое для их сложения. На практике $t_m/t_a = 10, \ldots, 1000$, поэтому при оценке времени выполнения программы операции сложения и вычитания не учитываются [4].

Пусть расчетная область представляет собой куб с ребром L и числом ячеек $C = L^3$. При этом L кратно числу процессоров n, а $n = m^2 = k^3$. Область разбивается $(n-1)$ параллельными плоскостями на n одинаковых слоев при одномерной декомпозиции, $2(m-1)$ плоскостями на m^2 областей при двумерной декомпозиции ($(m-1)$ плоскостями, параллельными плоскости xy, и $(m-1)$ плоскостями, параллельными плоскости xz), $3(k-1)$ плоскостями на k^3 областей при трехмерной декомпозиции.

В каждой из областей содержится одинаковое число ячеек C/n, поэтому время, затрачиваемое на вычисления, совпадает для всех трех видов декомпозиции. Число обменов данными N_s пропорционально общей площади поверхности плоскостей сечения с коэффициентом пропорциональности $2q^2L$, где q — число неизвестных, умноженное на число фиктивных ячеек вдоль одного направления, которое равняется $2qC^{2/3}(n-1)$ для одномерной декомпозиции, $4qC^{2/3}(n^{1/2}-1)$ для двумерной и $6qC^{2/3}(n^{1/3}-1)$ для трехмерной (учитываются обмены в обе стороны).

Время одной итерации складывается из времени вычислений во всех ячейках T_c и времени, затрачиваемого на межпроцессорные обмены данными T_s. Пусть время счета в одной ячейке равняется t_c, а время одного обмена — t_s. Тогда время работы на однопроцессорном компьютере составляет $T_1 = Ct_c$, а на компьютере с n процессорами $T_n = Ct_c/n + N_s t_s$.

Число обменов данными зависит от вида декомпозиции. Ускорение $S_n = T_n/T_1$ и коэффициент масштабируемости $E_n = S/n$ находятся из соотношений

$$S_n = \frac{n}{1 + kN_s/C} = \frac{n}{1 + wqk(n^{2/w} - 1)C^{1/3}}, \quad E_n = \frac{1}{1 + wqk(n^{2/w} - 1)C^{1/3}}.$$

Здесь $w = 2$ для одномерной, $w = 4$ для двумерной и $w = 6$ для трехмерной декомпозиции. Величина $k = t_s/t_c$ представляет собой отношение времени одного обмена ко времени одной итерации в одной ячейке. Наличие ненулевого коэффициента k приводит к невозможности достижения линейного ускорения по числу процессоров при постоянном размере сетки. При одномерной декомпозиции ускорение S_1 ограничено некоторой постоянной, зависящей от C. При малых k и на отрезках значений n и C, применяемых на практике, вполне может достигаться значительный рост ускорения, хорошо приближаемый линейной (теоретической) зависимостью.

Поскольку невозможно увеличивать количество подобластей и процессоров при постоянном размере сетки, говорят о зависимости ускорения от числа процессоров при росте размера сетки. При линейном росте размера сетки удается достичь линейного ускорения для трехмерной декомпозиции (в отличие от одно- и двумерной декомпозиции).

При $n \leqslant 10^2$ двух- и трехмерная декомпозиция показывают практически стопроцентное линейное ускорение, а одномерная декомпозиция отстает не более, чем на 20 %. При $n = 10^3$ ускорение одномерной декомпозиции падает на 70 % от теоретического значения, а двумерной — на 10 %. При $n = 10^4$ ускорение при одномерной декомпозиции приближается к своему пределу $S = C^{1/3}/(2qk)$, при двумерной — падает на 20 % от теоретически возможной, а при трехмерной — на 5 %.

Время счета пропорционально числу ячеек (сетка n^3 на p процессорах). Объем пересылаемой информации пропорционален площади поверхности куба вычислительной области процессора $T_{c1} = \alpha_1 n^3/p$, $T_{s1} = Cn^2/p^{2/3}$. В соответствии с определением шага по времени имеем $T_{c2} = \alpha_2 n^3/p$, $T_{s2} = A \log_2 p$. Время выполнения программы равняется сумме времени вычислений и коммуникаций $T = T_c + T_s$. Общее время вычислений и коммуникаций находится из соотношения

$$T_c = T_{c1} + T_{c2} = (\alpha_1 + \alpha_2)\frac{n^3}{p} = \alpha\frac{n^3}{p}, \quad T_s = T_{s1} + T_{s2} = C\frac{n^2}{p^{2/3}} + A\log_2 p.$$

При $p = 1$ получим, что $T_1 = \alpha n^3$. Выражение для эффективности алгоритма имеет вид

$$E = \frac{T_1}{pT} = \frac{\alpha}{\alpha + Cp^{1/3}/n + Ap(\log_2 p)/n^3}.$$

Алгоритм является масштабируемым, если эффективность остается постоянной при пропорциональном изменении размера задачи и числа процессоров. Полагая $p/n^3 = \mu = \text{const}$, получим

$$E = \frac{\alpha}{\alpha C \mu^{1/3} + A\mu \log_2 p}.$$

При $p \to \infty$ имеем $\dfrac{d \log_2 p}{dp} \sim \dfrac{1}{p} \to 0$. Время выполнения программы при $\dfrac{n^3}{p} = \nu = \text{const}$ определяется выражением $T = \dfrac{T_1}{pE} = \alpha\nu + C\nu^{2/3} + A \log_2 p$. Алгоритм и его реализация являются масштабируемыми при $p \gg 1$ и $\dfrac{n^2}{p^{2/3}} \gg 1$ (процессоры обмениваются информацией большими пакетами).

6. Декомпозиция области и методы балансировки. Под балансировкой нагрузки (load-balancing) подразумевается такое распределение подзадач по процессорам, которое обеспечивает максимальное ускорение (каждый процессор выполняет одинаковое количество полезной вычислительной и коммуникационной работы на каждом шаге).

Качество балансировки зависит как от топологии вычислительной системы, так и от способа программирования. Для явных конечно-разностных схем и реализующих их алгоритмов балансировка нагрузки является более важной задачей, чем минимизация коммуникаций, в то время как для неявных разностных схем ситуация оказывается обратной [5].

Статическая балансировка осуществляется до начала выполнения процессов. Динамическая балансировка выполняется в процессе решения задачи. При этом используются методы статической балансировки, применяемые на каждом шаге по времени [6], а для повышения эффективности используется их распараллеливание [5, 7].

Связи контрольных объемов или узлов сетки удобно представить в виде ненаправленного графа. В качестве вершин графа берутся узлы сетки, а его граней — грани контрольного объема. Задача состоит в разбиении области на множество непересекающихся подобластей, которое минимизирует число коммуникаций между процессорами.

Рассмотрим ненаправленный граф $G = (V, E)$, где V — множество вершин, а E — множество ребер. Для декомпозиции и обеспечения балансировки нагрузки процессоров необходимо найти такие множества V_1 и V_2, что $V = V_1 \cup V_2$ и $V_1 \cap V_2 = 0$, причем $|V_1| \simeq |V_2|$, если $|V_2|$ — четное, и $|V_1| \simeq |V_2| - 1$, если $|V_2|$ — нечетное (под $|V|$ понимается число элементов множества V). Требуется минимизировать число граней $|E_c|$ множества E, разрезаемых при разбиении расчетной области на подобласти:

$$|E_c| = \Big|\big\{e :\ e \in E,\ e = (v_1, v_2),\ v_1 \in V_1,\ v_2 \in V_2\big\}\Big|,$$

при этом $v_i = (x_i, y_i, z_i)$. Условие балансировки нагрузки процессоров приводит к минимизации числа граней, по которым происходит соприкосновение соседних подобластей.

Среди методов статической балансировки наиболее простым является метод рекурсивного деления пополам (Recursive Bisection, RB) [5, 7]. Исходная область сначала разделяется на две подобласти в направлении наиболее протяженной стороны. Узлы сортируются в порядке возрастания координат относительно серединного узла. Половина узлов присваивается одной подобласти, половина — другой. Разбиение повторяется для каждой из подобластей. За k итераций получается 2^k подобластей (рис. 2). Метод применим в случае 2^n процессоров (для вычислительной системы с топологией гиперкуба).

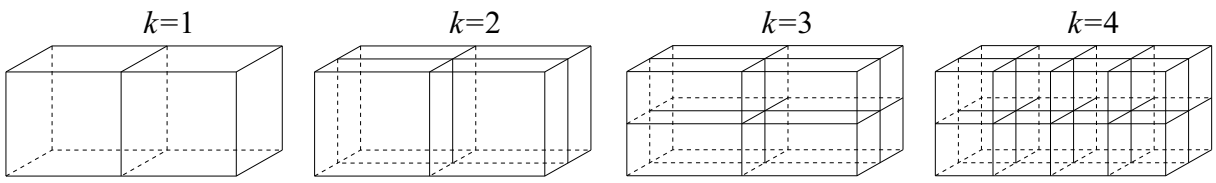

Рис. 2. Декомпозиция расчетной области при помощи метода рекурсивного деления пополам (четыре итерации)

Методы статической балансировки, основанные на методе рекурсивного деления пополам, различаются способом выбора метрики $d(v_i, v_j)$.

В методе рекурсивного деления по координатным направлениям (Recursive Coordinate Bisection, RCB) расчетная область сначала делится в самом протяженном координатном направлении, а все узлы сетки

сортируются по возрастанию относительно координаты серединного узла [8]. Половина узлов сетки присваивается одной подобласти, половина — другой. После этого процедура деления пополам повторяется нужное количество раз. Время работы оценивается как $O(n)$.

В методе RCB не выполняется оптимизация производительности, в результате чего получаются подобласти, вытянутые в одном из координатных направлений, что приводит к увеличению числа коммуникаций между процессорами. Для преодоления указанного недостатка используется метод, рассмотренный в [9], в котором вместо деления узлов области на равные подмножества и для получения подобластей приблизительно равной протяженности исходная область делится на подобласти из nk/p и $n(p-k)/p$ узлов, где n — число узлов, а p — число процессоров, $k \in \{1, 2, \ldots, p-1\}$. При помощи выбора подходящего значения k на каждой итерации метод приводит к подобластям с примерно равным числом узлов, но лучшим отношением сторон, чем метод RCB.

В методе рекурсивного деления пополам графа (Recursive Graph Bisection, RGB) в качестве объекта декомпозиции используется граф разностной сетки [10]. Расстояние между двумя вершинами графа определяется как наименьшее число граней, которые нужно пройти от узла i к узлу j (диаметр графа). Время работы оценивается как $O(n)$.

В инерционном методе рекурсивного деления пополам (Recursive Inertia Bisection, RIB) каждому узлу графа присваивается условная единичная масса и находится главная ось инерции полученной системы [11]. Деление пополам производится относительно оси, ортогональной главной оси инерции. Время работы оценивается как $O(n)$.

В отличие от методов RCB и RGB, спектральный метод рекурсивного деления пополам (Recursive Spectral Bisection, RSB) приводит к связным подобластям при условии, что исходный граф также является связным. С i-м узлом графа соотносится условный вес $w_i = -1$ или $w_i = +1$ в зависимости от того, к какой половине исходной области относится узел. Для обеспечения балансировки нагрузки процессоров необходимо минимизировать квадратичную форму $|E_c| = \dfrac{1}{4} \displaystyle\sum_{(i,j)\in V, i \leftrightarrow j} (w_i - w_j)^2$ [7]. Обозначение $i \leftrightarrow j$ используется для указания существования грани, соединяющей узлы i и j. Поскольку подобласти имеют одинаковое число узлов, из условия $w_i = \pm 1$ получим, что $\displaystyle\sum_{i=1}^{n} w_i = 0$, $\displaystyle\sum_{i=1}^{n} w_i^2 = n$. Задача декомпозиции сводится к задаче целочисленного программирования с ограничениями.

Для облегчения решения задачи целочисленные ограничения игнорируются, и вместо дискретной рассматривается решение непрерывной задачи, которая приводит к минимизации квадратичной формы $|E_c| = \dfrac{1}{4}\, \boldsymbol{w}'\, \mathrm{L}\, \boldsymbol{w} = \displaystyle\sum_{(i,j)\in E} (w_i - w_j)^2$. Элементы матрицы Лапласа $\mathrm{L}(G) = \{l_{ij}\}_{i,j=1,\ldots,n}$ имеют вид

$$l_{ij} = \begin{cases} -1, & \text{если } (v_i, v_j) \in E, \\ \deg\, v_i, & \text{если } i = j, \\ 0 & \text{иначе,} \end{cases}$$

причем $(v_i, v_j) \in E$, если $i \neq j$ и $i \leftrightarrow j$, а под весом $\deg\, v_i$ понимается число граней графа, связанных с данным узлом. Матрица Лапласа совпадает с дискретным представлением оператора Лапласа на пятиточечном шаблоне при использовании граничных условий типа Неймана (сетка 8×8):

$$\mathrm{L} = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

Матрица Лапласа представляется в виде $\mathrm{L}(G) = -\mathrm{D} + \mathrm{A}$, где A — матрица смежности графа, а D — диагональная матрица, состоящая из весов узлов графа. Наименьшее собственное значение матрицы Лапласа $\lambda_1 = 0$, а соответствующий ему собственный вектор $\boldsymbol{e}_1 = \{1, 1, \ldots, 1\}'$ не удовлетворяет исходным ограничениям и не является решением задачи. В то же время, наименьшее ненулевое собственное значение матрицы Лапласа λ_2 дает собственный вектор $\boldsymbol{e}_2$, который удовлетворяет всем ограничениям (Friedler

vector). Собственный вектор матрицы Лапласа, соответствующий ее наименьшему нетривиальному собственному значению, определяет связность графа и его протяженность, а также выступает в качестве метрики.

Для реализации метода RSB находится собственный вектор матрицы Лапласа $\boldsymbol{e}_2$. Вектор делится на две части, а элементы каждой части присваиваются различным подобластям. Для каждой подобласти шаги реализуются в такой же последовательности. Время работы оценивается как $O(mn)$, где m — число итераций для получения собственных значений матрицы Лапласа.

7. Вычисление шага по времени и его синхронизация. Сшивка решений, полученных различными процессами, приводит к требованию равенства используемых ими временны́х шагов, а выбор минимального временно́го шага — к необходимости обмена данными между процессами. Способ выбора временно́го шага влияет на производительность, особенно при большом числе процессоров и существенных задержках при передаче данных.

Каждый процесс вычисляет временной шаг, исходя из условия Куранта–Фридрихса–Леви (КФЛ) и значений параметров потока в своей подобласти. После этого процессы выбирают наименьшее значение временно́го шага. Выбор минимального значения среди всех Δt, вычисленных из условия КФЛ в подобластях, эквивалентен применению условия КФЛ ко всей области в целом.

Наиболее простой способ выбора временно́го шага состоит в передаче рассчитанных значений из всех процессов одному процессу, выбранному заранее, например процессу с номером 0. Процесс с номером 0, выбрав из всех значений Δt минимальное, рассылает его остальным процессам. При этом количество пересылок равняется удвоенному числу процессоров ($n_t = 2n_p$).

Используя независимую передачу данных между парами узлов системы, можно ускорить передачу данных за счет параллелизации пересылок на основе алгоритма двоичного дерева. Выделяется один процесс (процесс уровня 1), который ожидает поступления данных от двух подчиненных процессов (процессы уровня 2), выбирает наименьшее значение из своего значения временно́го шага и двух значений, полученных по сети, после чего передает результат обоим подчиненным процессам. Каждый процесс уровня 2, в свою очередь, имеет по два подчиненных процесса уровня 3. Значение временно́го шага, передаваемое процессу уровня 1, является минимальным из значений Δt данного процесса и значений, принятых от процессов уровня 3. Данная схема распространяется на процессы уровня 3, 4 и далее. В конечном счете, в древовидную структуру включаются все процессы задачи. Количество пересылок получается равным $n_t = 2\big(|\log_2 n_p| + 1\big)$ (с учетом обратных пересылок, связанных с распространением общего значения временно́го шага по сети). Замена линейной зависимости логарифмической позволяет увеличить порог насыщения по числу процессоров при любом типе разбиения.

8. Параллельные итерационные методы. Разработка параллельных методов решения систем разностных уравнений делает возможным решение задач большой размерности за приемлемое время.

8.1. Общая структура. Система разностных уравнений записывается в виде $\mathrm{A}\boldsymbol{x} = \boldsymbol{b}$ или $\displaystyle\sum_{j=1}^{n} a_{ij}x_j = b_i$, где $a_{ii} \neq 0\ \forall\, i = 1, \ldots, n$. Матрица $\mathrm{A} = \{a_{ij}\}_{i,j=1,\ldots,n}$ содержит коэффициенты, связанные с дискретизацией. Вектор $\boldsymbol{x} = \{x_i\}_{i=1,\ldots,n}$ состоит из неизвестных узловых значений искомой функции. Вектор $\boldsymbol{b} = \{b_i\}_{i=1,\ldots,n}$ составлен из коэффициентов, обусловленных дискретизацией, а также из известных значений $\boldsymbol{x}$, например из тех, которые задаются граничными условиями. Матрица A является симметричной и положительно определенной: $\mathrm{A} = \mathrm{A}'$, $\boldsymbol{x}'\mathrm{A}\,\boldsymbol{x} > 0$ для $\forall\ \boldsymbol{x} \neq 0$.

Общая структура итерационных методов связана с представлением матрицы коэффициентов в виде $\mathrm{A} = \mathrm{N} - \mathrm{P}$ и с видоизмененной формой исходного уравнения $(\mathrm{N} - \mathrm{P})\,\boldsymbol{x} = \boldsymbol{b}$. Матрица N близка к матрице A (при этом $\|\mathrm{N}\| \simeq \|\mathrm{A}\|$) и легко поддается численной факторизации (например, в случае трехдиагональной структуры). Итерационные методы отличаются друг от друга способом выбора матрицы N [12, 13].

Итерационный метод общего вида основан на последовательном улучшении начального приближения решения. Решение на итерации $k+1$ находится по формуле $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \mathrm{N}^{-1}\boldsymbol{r}^k$, где $\boldsymbol{r}^k = \mathrm{A}\boldsymbol{x}^k - \boldsymbol{b}$ представляет собой вектор невязки на k-й итерации. Итерации заканчиваются, когда $\max|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k| < \varepsilon_1$ и $\|\boldsymbol{r}^k\|_2/\|\boldsymbol{r}^0\|_2 < \varepsilon_2$.

Пусть система состоит из p процессоров, а вектор неизвестных, вектор невязки и матрица системы разбиты на блоки следующим образом:

$$\boldsymbol{x}^k = \begin{pmatrix} \boldsymbol{x}_1^k \\ \vdots \\ \boldsymbol{x}_p^k \end{pmatrix}, \quad \boldsymbol{r}^k = \begin{pmatrix} \boldsymbol{r}_1^k \\ \vdots \\ \boldsymbol{r}_p^k \end{pmatrix}, \quad \mathrm{N}^{-1} = \begin{pmatrix} \mathrm{N}_1 \\ \vdots \\ \mathrm{N}_p \end{pmatrix}.$$

При этом блоки $\boldsymbol{x}_i^k$, $\boldsymbol{r}_i^k$ и N_i приписываются i-му процессору ($i = 1, \ldots, p$).

При геометрической декомпозиции возникают трудности, связанные с рекурсивным характером вычислений при обращении матрицы предобусловленности и вычислении ее элементов. Для их преодоления используются методы переупорядочивания неизвестных и реконструирование матрицы предобусловленности [14].

Методы упорядочивания приводят к различным свойствам матрицы коэффициентов, в частности, к разным спектрам и спектральным радиусам, оказывая влияние как на скорость сходимости, так и на степень параллелизма итерационных методов. В общем случае используется такой метод упорядочивания (ordering), который приводит к матрице коэффициентов, наиболее близкой к исходной матрице (матрице, которой соответствует лексикографический порядок неизвестных). Близость матриц оценивается в смысле близости их спектров и спектральных радиусов.

8.2. Метод Якоби. В методе Якоби $\mathrm{N} = \mathrm{DI}$, $\mathrm{P} = \mathrm{L} + \mathrm{U}$, где $\mathrm{D} = \mathrm{diag}\{a_{11}, \ldots, a_{nn}\}$. Решение находится по формуле $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \mathrm{D}^{-1}(\boldsymbol{b} - \mathrm{A}\boldsymbol{x}^k)$, или $x_i^{k+1} = \left(b_i - \sum\limits_{i \neq j}^{n} a_{ij} x_j^k\right) \Big/ a_{ii}$. Вычисление решения x_i^{k+1} в узле i требует знания решения x_i^k на предыдущей итерации в том же самом узле. Пример распределения узлов сетки по процессорам и вид матрицы приведены на рис. 3 для дискретизации уравнения Пуассона на шаблоне типа "крест" (сетка 4×4).

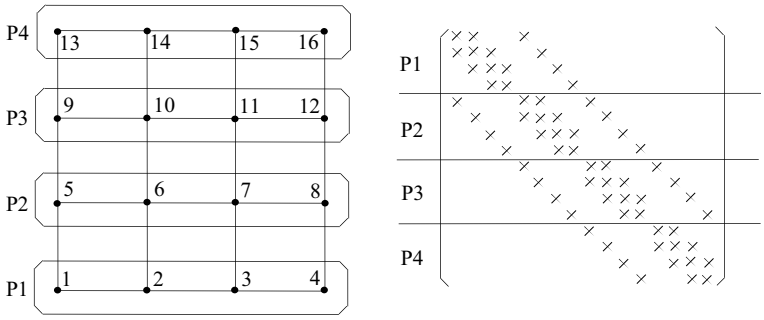

Рис. 3. Распределение узлов по процессорам и вид матрицы

Реализация метода Якоби условно представляется в следующем виде:

последовательный алгоритм	параллельный алгоритм
for $i = 1, \ldots, n$ вычислить x_i^{k+1} end	for $i = i^p_{\mathrm{start}}, \ldots, i^p_{\mathrm{end}}$ вычислить x_i^{k+1} end

Параллельная реализация метода Якоби заключается в выполнении следующих шагов.

1. Блоки $\boldsymbol{x}_i^0$, $\boldsymbol{r}_i^0$, N_i пересылаются на i-й процессор.

2. На k-й итерации процессор под номером i вычисляет x_i^{k+1}, пересылает результат другим процессорам, получает недостающие компоненты вектора решения, проверяет условие сходимости. При выполнении условия сходимости процессор устанавливает соответствующий флаг.

3. Если хотя бы один флаг не установлен, то осуществляется переход к следующей итерации.

4. Результат передается на основной процессор.

Глобальные коммуникации требуются только после окончания вычислений для проверки условия сходимости решения.

8.3. Метод Гаусса–Зейделя. В методе Гаусса–Зейделя $\mathrm{N} = \mathrm{DI} - \mathrm{L}$, $\mathrm{P} = \mathrm{U}$. Формула для нахождения решения $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + (\mathrm{D} - \mathrm{L})^{-1}(\boldsymbol{b} - \mathrm{A}\boldsymbol{x}^k)$, или $x_i^{k+1} = \left(b_i - \sum\limits_{j<i} a_{ij} x_j^{k+1} - \sum\limits_{j>i} a_{ij} x_j^k\right) \Big/ a_{ii}$. В случае, если $a_{ij} = 0 \ \forall\, j < i$, вычисления проводятся по сценарию метода Якоби. Решение на итерации $k+1$ зависит от значений, полученных как на итерации k, так и на итерации $k+1$.

Простейший способ распараллеливания в методе Гаусса–Зейделя состоит в том, чтобы хранить решения на итерациях k и $k+1$ в различных массивах.

Другой путь заключается в разделении узлов шаблона на два типа, например, красные/черные (двухцветная параллелизация, red/black). Сначала производится расчет решения в red-узлах, а затем в black-узлах, или наоборот (значения в red-узлах зависят только от значений в black-узлах, но не от самих себя). Коммуникации требуются на каждой итерации. Нумерация узлов разностного шаблона показана на рис. 4.

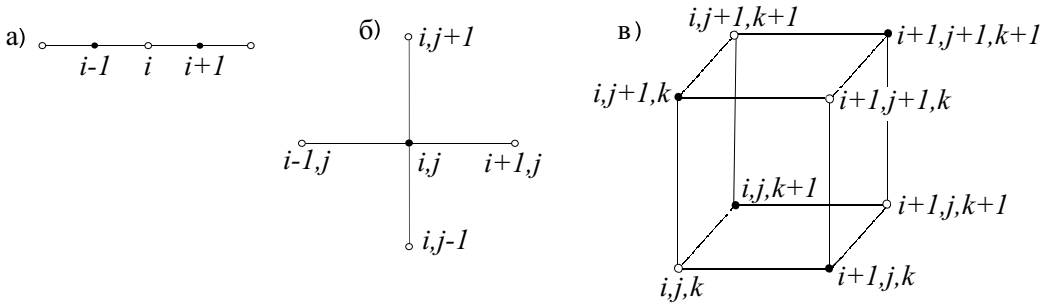

Рис. 4. Индексация узлов при двухцветной параллелизации метода Гаусса–Зейделя в одномерном (фрагмент а), двухмерном (фрагмент б) и трехмерном (фрагмент в) случае ($\circ$ — red-узлы, $\bullet$ — black-узлы)

Пример распределения узлов сетки по процессорам и вид матрицы приведены на рис. 5 для дискретизации уравнения Пуассона на пятиточечном шаблоне (сетка 4×4).

Используется также многоцветная параллелизация (Multi-Coloring, MC), позволяющая улучшить балансировку нагрузки процессоров, когда вычисления в узлах каждого цвета проводятся независимо.

Стратегия многоцветной параллелизации состоит в том, чтобы получить расщепление исходной системы на равные подмножества уравнений, которые не имеют внутренних зависимостей между неизвестными.

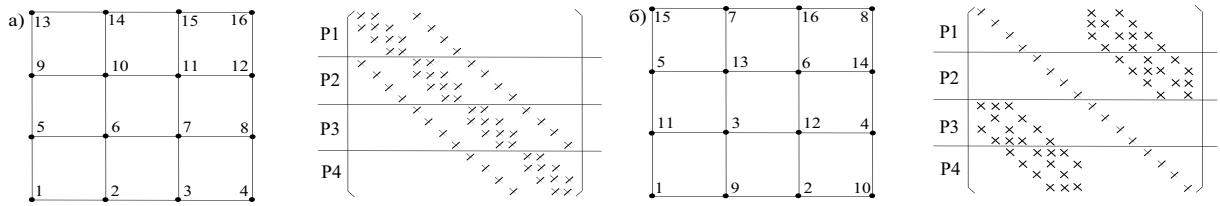

Рис. 5. Расположение узлов разностного шаблона и вид матрицы при последовательной (фрагмент а) и параллельной (фрагмент б) реализации метода Гаусса–Зейделя

$$\begin{pmatrix} 3 & 4 & 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 \end{pmatrix} \qquad \begin{pmatrix} 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{pmatrix}$$

а) б)

Рис. 6. Матрица индексов неизвестных для сетки 8×8

Пусть дано множество цветов $C = \{1, 2, \ldots, K\}$. Для упорядочивания узлов сетки необходимо найти отображение $\sigma : V \to C$, такое, что $\sigma(u) \neq \sigma(v)$ для $\forall\, u, v \in E$. Множество узлов V разбивается на независимые подмножества $(C_1, C_2, \ldots, C_N)$, в каждом из которых узлы оказываются не связанными какой-либо гранью графа исходной матрицы (смежные узлы имеют различный цвет). Минимальное число цветов $\chi = N$, необходимых для построения такого разбиения, называется хроматическим числом графа. Каждый узел сетки имеет цвет, отличный от цветов всех других узлов, с которыми он связан разностным шаблоном. Для сетки 8×8 матрица индексов неизвестных имеет вид, представленный на рис. 6 а.

Реализация метода многоцветной параллелизация проводится в следующей последовательности.

1. Из графа G матрицы A выбирается независимое подмножество узлов V_1 (множество, в котором нет двух соседних узлов), которое присваивается цвету 1, после чего удаляется из графа вместе с соответствующими гранями.

2. Выбирается независимое подмножество узлов V_2, присваивается цвету 2 и удаляется из графа.

3. Процесс выбора и удаления повторяется до тех пор, пока всем узлам не будет присвоен какой-либо цвет.

4. Строки и столбцы матрицы A переставляются таким способом, что все узлы цвета i предшествуют всем узлам цвета $i+1$. Такая перестановка изменяет зависимость между неизвестными (результат вычислений для узлов цвета j зависит только от результатов, полученных в узлах с цветом $i < j$).

5. Вычисления в узлах каждого цвета проводятся независимо.

При увеличении количества цветов возрастает скорость сходимости, но уменьшается степень параллелизма. При этом получаются вектора малой длины, что приводит к увеличению объема коммуникаций [12]. Метод имеет хорошие показатели производительности для симметричных матриц (или матриц, близких к ним).

Реализация метода Гаусса–Зейделя условно представляется в следующем виде:

двухцветная параллелизация	многоцветная параллелизация
for color={red, black} вычислить x_i^k end	for color={1, 2, ... } выбрать узел $x_i^k \in X$ в соответствии с процедурой упорядочивания; выбрать наименьший возможный цвет для x_i^k; $V = V \backslash \{x_i\}$ end

Узлы заносятся в множество X в порядке возрастания своего веса: $\deg x_1 \geqslant \deg x_2 \geqslant \ldots \geqslant \deg x_n$ (под весом узла x_i понимается количество смежных узлов того же цвета).

Для ускорения сходимости применяется метод циклической многоцветной параллелизации (Cyclic MC, CMC), в котором цвета узлам сетки присваиваются в циклическом порядке по диагонали. В случае сетки 8×8 матрица индексов неизвестных имеет вид, представленный на рис. 6 б. Всего имеется по 15 узлов каждого из четырех цветов. При этом узлы $\{1,5,9,13\}$, $\{2,6,10,14\}$, $\{3,7,11,15\}$, $\{4,8,12\}$ имеют одинаковый цвет.

При дискретизации уравнения Лапласа на пятиточечном шаблоне и регулярной сетке $n \times n$ используется двухцветная параллелизация, в которой присваивание цветов производится по диагонали ячейки сетки. Степень параллелизма оценивается как $O(\sqrt{n})$.

В случае девятиточечного шаблона используется четыре цвета (например, red, black, green, orange). Порядок, в котором раскрашиваются узлы сетки, может быть различным, например:

$$\begin{pmatrix} r & o & r & b \\ r & b & g & o \\ g & o & r & b \\ r & b & g & o \end{pmatrix} \quad \text{или} \quad \begin{pmatrix} g & o & g & o \\ r & b & r & b \\ g & o & g & o \\ r & b & r & b \end{pmatrix}.$$

При использовании восьмицветного шаблона на разнесенной сетке нумерация узлов контрольного объема показана на рис. 7. Такой подход применяется для распараллеливания схемы расщепления на разнесенной сетке (staggered grid) [14].

Для задач конвективно-диффузионного переноса стратегия многоцветной параллелизации часто приводит к неудовлетворительным результатам. В таких задачах используется метод, связанный с направлением потока (Flow Directed Point Iterations, FDPI). Множество узлов сетки разделяется на четыре подмножества в соответствии со знаком составляющих скорости:

$$N_1 = \{(i,j) \mid u_{ij} \geqslant 0,\ v_{ij} \geqslant 0\}; \quad N_2 = \{(i,j) \mid u_{ij} < 0,\ v_{ij} \geqslant 0\};$$
$$N_3 = \{(i,j) \mid u_{ij} \geqslant 0,\ v_{ij} < 0\}; \quad N_4 = \{(i,j) \mid u_{ij} < 0,\ v_{ij} < 0\}.$$

Для получения такого разбиения узлы заносятся в множество N_1 при помощи обхода узлов сетки слева направо и снизу вверх, в множество N_2 — справа налево и снизу вверх, в множество N_3 — слева направо и сверху вниз, в множество N_4 — справа налево и сверху вниз.

8.4. Метод последовательной верхней релаксации. В методе релаксации N = DI/(ω − L), P = U, где $\omega \in [1,2]$. Формула для нахождения решения будет иметь вид $\boldsymbol{x}^{k+1} = (1-\omega)\boldsymbol{x}^k + \omega \boldsymbol{x}_*^k$, или $x_i^{k+1} = (1-\omega)x_i^k + \omega\left(b_i - \sum\limits_{j<i} a_{ij}x_j^{k+1} - \sum\limits_{j>i} a_{ij}x_j^k\right)\Big/a_{ii}$. Значение x_*^k рассчитывается по методу Гаусса–Зейделя. Для распараллеливания используются те же подходы, что и для метода Гаусса–Зейделя (двух- и многоцветная параллелизация).

8.5. Многосеточный метод. Особенности реализации многосеточного подхода к решению системы разностных уравнений приведены в [15].

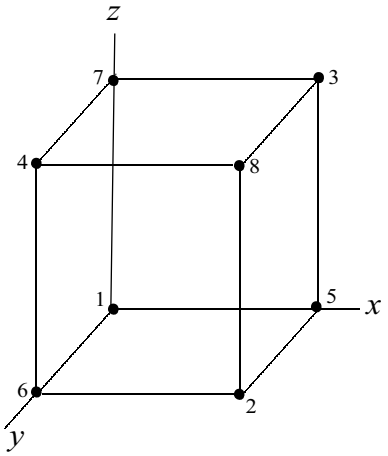

Рис. 7. Локальная индексация узлов контрольного объема

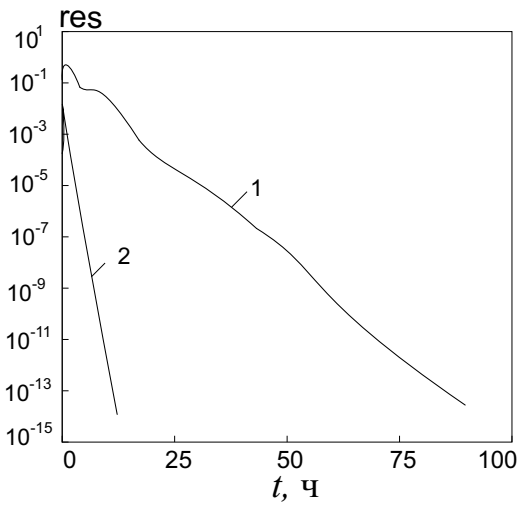

Рис. 8. Изменение невязки решения в случае одной сетки (кривая 1) и пяти вложенных сеток (кривая 2) в зависимости от времени счета

Для параллелизации вычислительного алгоритма расчетная сетка Ω разделяется на p перекрывающихся подсеток (по числу процессоров). В случае многосеточного подхода разделение расчетной области происходит как на подробной, так и на грубой сетке. При этом узел i грубой сетки Ω^H принадлежит подсетке процессора q только в том случае, если он принадлежит подробной сетке Ω^h.

Пусть T_i — число ячеек в сетке i, T_i^α — число ячеек в разбиении α. Тогда ускорение параллельного алгоритма для i-й сетки на системе с p процессорами составляет $L_i = T_i / \max\limits_\alpha \{T_i^\alpha p\}$.

Многосеточный подход дает ускорение [16]

$$L_M = \left(\sum_{i=1}^{N-1} c\gamma^{i-1} T_i + n_c T_N \right) \left(\sum_{i=1}^{N-1} c\gamma^{i-1} \max_\alpha \{T_i^\alpha\} + n_c \max_\alpha \{T_N^\alpha\}\, T_N \right)^{-1},$$

где n_c — число сглаживающих итераций на грубой сетке (обычно $n_c \sim 10$), $c = \mu_1 + \mu_2 + 1$, μ_1 — число приближений решения на подробной сетке при помощи метода Гаусса–Зейделя (предварительное сглаживание), μ_2 — число приближений решения на подробной сетке (постсглаживание). Число многосеточных итераций составляет γM_k. Под γ понимается рекурсивный параметр, влияющий на качество коррекции на грубой сетке (используются разные значения для сеток различного уровня). Обычно полагается $\mu_1 = \mu_2 = 1$ и используется либо V-цикл ($\gamma = 1$), либо W-цикл ($\gamma = 2$).

Таблица 1
Ускорение вычислительного алгоритма

L_i	L_1	L_2	L_3	L_4	L_5	L_M
S_i	6.6	6.2	6.2.	5.9	5.2	6.1

Значения ускорения вычислительного алгоритма при $p = 8$ на пяти вложенных сетках $L_1, \ldots, L_5$ (L_1 соответствует подробной сетке, а L_5 — самой грубой сетке), а также ускорение для многосеточного подхода L_M приведены в табл. 1. Изменение невязки решения показано ниже на рис. 11. Несмотря на увеличение вычислительной работы на каждом шаге по времени, многосеточный метод позволяет добиться заданного уровня сходимости за меньшее число шагов.

8.6. Сравнение различных подходов. Сравнительная характеристика различных итерационных методов и ресурсов (время счета и память), необходимых для реализации последовательного и параллельного кода, приведена в табл. 2 в зависимости от размера матрицы n. Время выполнения оценивается как произведение числа итераций на стоимость одной итерации [13]: $\mathrm{cost}_1 \simeq t_1 O(n/p) + t_2 + t_3 O(\sqrt{n}/p)$, где n — число неизвестных; p — число процессоров; t_1 — время, затрачиваемое на одну операцию; t_2 — время задержки посылки сообщения; t_3 — время, необходимое для пересылки одного числа между процессорами.

Особенности реализации различных итерационных методов приведены в [14, 15].

9. Программные и системные средства. На практике для создания параллельных приложений широкое применение находит интерфейс межпроцессорного взаимодействия MPI (Message Passing Interface).

Технология MPI предназначена для поддержки работы параллельных процессов в терминах передачи сообщений. Интерфейс MPI предоставляет единый механизм взаимодействия ветвей внутри параллельного приложения независимо от машинной архитектуры (однопроцессорные/многопроцессорные системы с общей/распределенной памятью) и взаимного расположения ветвей (на одном процессоре/на разных). Тем не менее во многих случаях предпочтительно использовать программные средства более высокого уровня, чем MPI, которые облегчают создание параллельного приложения для данной предметной области.

Таблица 2

Сравнительная характеристика различных итерационных методов

метод	последовательный код	параллельный код	память	число процессоров
метод Якоби	n^2	n	n	n
метод Гаусса-Зейделя (red/black)	n^2	n	n	n
метод SOR (red/black)	$n^{3/2}$	$n^{1/2}$	n	n
метод CG	$n^{3/2}$	$n^{1/2}\log n$	n	n
LU-разложение (плотная матрица)	n^3	n	n^2	n^2
LU-разложение (ленточная матрица)	n^2	n	$n^{3/2}$	n
LU-разложение (разреженная матрица)	$n^{3/2}$	$n^{1/2}$	$n\log n$	n
метод FFT	$n\log n$	$\log n$	n	n
многосеточный метод	n	$\log^2 n$	n	n
нижняя граница	n	$\log n$	n	

Для распараллеливания используется библиотека OPlus v2.12 (Oxford Parallel Library for Unstructured Solvers), реализованная на языке Fortran 77 в вычислительной лаборатории университета Оксфорда (Oxford University Computing Laboratory) для платформ Win32, Linux, SGI, AIX [17, 18]. К достоинствам библиотеки OPlus относятся: использование обобщенных структур данных (в качестве элемента данных рассматриваются грани и ячейки, которые имеют различный тип — шестигранники, призмы, параллелепипеды, тетраэдры); производительность (посылка сообщений происходит только при модификации данных, а для уменьшения задержек при посылках сообщений данные объединяются в пакеты); переносимость (для посылки сообщений используются функции MPI).

Использование библиотеки OPlus позволяет запустить программу как в последовательном режиме без вызова функций передачи сообщений (что удобно, например, для отладки кода), так и в параллельном режиме (парадигма master-slave, SPMD-концепция). Последовательная версия библиотеки содержит функции с теми же аргументами, что и ее параллельная версия, однако эти функции не выполняют действий по передаче сообщений (пустое тело функции).

Для проведения расчетов использовались следующие системы:

— 32-процессорный кластер на базе процессора Pentium IV 1.5 ГГц;

— 48-процессорный кластер на базе процессора Xeon 2.4 ГГц (24 узла);

— 60-процессорный кластер на базе процессора Opteron 2.4 ГГц (30 узлов);

— Silicon Graphics Origin 2000 с процессором R12000 350 МГц;

— IBM SP/1600 с узлами eServer pSeries 690 Regatta на базе процессора Power 4+ 1.7 ГГц.

Вычисления производились в режиме удаленного терминала. Кластеры, выпускаемые компаниями Streamline Computing Ltd и Cluster Vision Ltd, расположены в университете Суррея (University of Surrey, United Kingdom). Суперкомпьютерный центр находится в Daresbury Laboratory (Warrington, United

Kingdom); центр создан в рамках проекта HPCx (данные приводятся на 2003 год). Система поддерживается HPCx-консорциумом (UoE HPCx Ltd), в который входят университет Эдинбурга (University of Edinburgh), Центральная лаборатория совета по научным исследованиям (Central Laboratory for the Research Councils, CLRC) и компания IBM. Учебно-методическую поддержку осуществляет компьютерный центр университета Эдинбурга (Edinburgh Parallel Computer Center, EPCC).

Суперкомпьютер IBM SP/1600 состоит из 40 SMP-узлов pSeries 690 Regatta, каждый из которых содержит 32 процессора Power 4+ с 64-разрядной архитектурой и тактовой частотой 1.7 ГГц, обеспечивая теоретическую пиковую производительность 6.66 ТФлопс/с. Для оптимизации передачи данных между фреймами каждый из них разделяется на 4 узла по 8 процессоров (Logical Partitions, LPAR). С точки зрения программирования, система представляет собой 160 узлов SMP-типа по 8 процессоров каждый. Оперативная память всей системы достигает 1.28 ТФлопс, а дисковая — 18 ТФлопс.

Сетка и поле течения хранятся в формате библиотеки ADF Software Library (Advanced Data Format), которая является частью библиотеки CGNS (CFD General Notation System), разработанной сначала для внутреннего использования в корпорации Boeing, а затем получившей широкое распространение в NASA и компании McDonnel Douglas Aerospace. Для генерации файлов объемом, превышающим 2 Gb (это необходимо для проведения расчетов с использованием полной геометрической модели), на системе с 32-разрядной архитектурой используется расширенная версия библиотеки CGNS LFS (Large File System), разработанная корпорацией Rolls-Royce и университетом Суррея.

10. Результаты расчетов. Рассмотрим решение ряда задач механики жидкости и газа и исследуем производительность и эффективность программного кода в зависимости от размера задачи и числа процессоров. Разностные схемы для решения различных задач описаны в [3, 14, 15].

10.1. Решение уравнения Пуассона. Уравнение Пуассона описывает ламинарное течение вязкой несжимаемой жидкости в канале при малых числах Рейнольдса (течение Пуазейля). Правая часть уравнения при этом определяет перепад давления между входным и выходным сечениями канала. На стенках канала выставляются граничные условия типа Дирихле.

Решение уравнения Пуассона выполняется на прямоугольной равномерной сетке при помощи метода контрольного объема. Расчетная область представляет собой куб с ребром $L = 1$. Для дискретизации производных используются центрированные конечно-разностные формулы второго порядка, а для решения системы разностных уравнений — многосеточный метод [15]. При проведении расчетов меняется размерность сетки (указывается размерность сетки наилучшей разрешающей способности), число уровней сетки и тип многосеточного цикла (V или W). Сетки различной разрешающей способности строятся при помощи метода схлопывающихся граней [15]. Для декомпозиции расчетной области применяется метод RCB.

Расчет на сетке 64^3 (четыре уровня сетки, V-цикл) с использованием последовательного кода на компьютере Pentium IV 2.0 ГГц занимает 15 сек. Для достижения сходимости делается 100 итераций (последовательный и параллельный код дают одинаковый уровень невязки независимо от числа используемых процессоров).

Увеличение числа уровней сетки свыше 4 не дает выигрыша в производительности. В связи с этим на практике имеет смысл использовать не более четырех уровней сетки.

Применение W-цикла оказывается более затратным в вычислительном отношении, чем применение V-цикла, так как требуется на 50 – 88 % больше процессорного времени (оценки производились при $n_p = 1, \ldots, 16$ на сетке 256^3).

Зависимости ускорения от размера сетки (при фиксированном числе процессоров) и числа процессоров (при фиксированном размере сетки) приведены на рис. 9 и 10 (расчеты проводились на сетках 32^3, 64^3, 128^3 и 256^3). Полученные результаты свидетельствуют о масштабируемости вычислительного алгоритма как при изменении размера сетки, так и при изменении числа процессоров, участвующих в вычислениях.

На сетках достаточно большой размерности с числом ячеек 256^3 ускорение алгоритма превосходит соответствующее теоретической значение. Вместе с тем, алгоритм показывает практически одинаковое ускорение на сетках малой размерности с числом ячеек 32^3 и 64^3.

10.2. Течение в каверне. Рассмотрим нестационарное течение вязкой несжимаемой жидкости в кубической каверне, индуцированное движением с постоянной скоростью ее верхней стенки [3].

Расчетная область представляет собой кубическую полость со стороной $L = 1$ м. Верхняя стенка перемещается вдоль оси x с постоянной скоростью $U = 1$ м/с. На стенках каверны выставляются граничные условия прилипания и непротекания. Расчеты проводились для жидкости с молекулярной вязкостью μ, соответствующей заданному числу Рейнольдса и плотностью $\rho = 1.18$ кг/м^3.

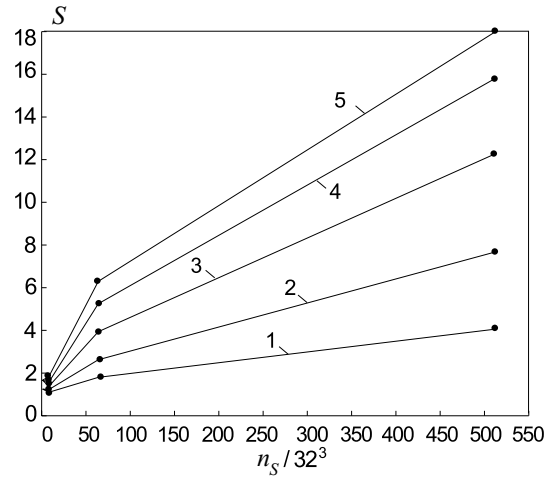

Рис. 9. Зависимость ускорения от числа ячеек сетки при $n_p = 2$ (1); 4 (2); 8 (3); 12 (4); 16 (8)

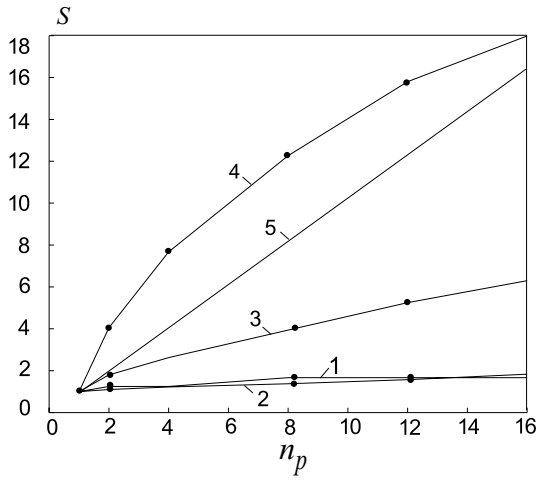

Рис. 10. Зависимость ускорения от числа процессоров для сетки $n_s = 32^3$ (1); 64^3 (2); 128^3 (3); 256^3 (4). Кривая 5 соответствует теоретической зависимости

Для дискретизации уравнений Навье–Стокса используется схема расщепления на разнесенной сетке [14], а для расчета конвективных потоков — схема SMART [3]. Система разностных уравнений решается при помощи методов SOR и BiCGStab [14]. Для реализации метода SOR используется двух- и восьмицветная параллелизация. Для декомпозиции расчетной области применяется метод RCB.

Основные затраты расчетного времени связаны с решением уравнения Пуассона для давления и с вычислением матрично-векторного произведения (при использовании метода BiCGStab).

Метод многоцветного упорядочивания узлов (при использовании метода SOR) сравним с матрично-векторным произведением как по вычислительной структуре, так и по коммуникационным требованиям. Однако число сообщений, необходимых для пересылки данных между процессорами в методе SOR, оказывается бóльшим, чем в случае матрично-векторного произведения, а масштабируемость метода SOR — хуже.

Метод многоцветной параллелизации обеспечивает более высокую производительность, чем метод двухцветного упорядочения неизвестных (особенно на больших сетках).

Показатели производительности достаточно слабо зависят от количества фиктивных ячеек (один слой для трехточечного и два слоя для пятиточечного шаблона), по которым производится стыковка соседних подобластей.

Зависимости ускорения от размера сетки приведены на рис. 11 для разных методов решения уравнения Пуассона для давления. На сетках большой размерности метод BiCGStab обеспечивает лучший прирост ускорения по сравнению с методом SOR.

10.3. Течение в межлопаточном канале. Рассмотрим турбулентное течение вязкого сжимаемого газа в межлопаточном канале газовой турбины и его взаимодействие с поперечным потоком из каверны (задача представляет интерес для охлаждения лопаток газотурбинной установки). Течение описывается полными осредненными по Рейнольдсу уравнениями Навье–Стокса, замкнутыми при помощи модели турбулентности Спаларта–Аллмараса [15]. Дискретизация уравнений проводится при помощи метода контрольного объема на гибридной сетке [15]. Для декомпозиции расчетной области используются методы RIB и RGB.

Расчеты проводились как для полной модели, так и ее части, показанной на рис. 12 (в окружном направлении выставляются периодические граничные условия). Расчетная область ограничена поверхностью профиля, стенками канала, периодическими поверхностями, входной и выходной границами канала, входной границей каверны и стенкой каверны. Стрелки показывают направление потока.

На входной границе межлопаточного канала задаются: полное давление $p_0 = 2.73 \cdot 10^5$ Па, полная температура $T_0 = 407$ К, угол закрутки $\alpha = 0°$, угол тангажа $\beta = 0°$, а также турбулентная вязкость $\widetilde{\nu}_0 = 10^{-3}$ м^2/с. На входной границе каверны $p_0 = 1.55 \cdot 10^5$ Па, $T_0 = 336$ К, $\alpha = 90°$, $\beta = 83°$, $\widetilde{\nu}_0 = 10^{-3}$ м^2/с. На выходной границе задается статическое давление $p = 1.61 \cdot 10^5$ Па. Граничные условия непротекания и прилипания, а также температура стенки $T_w = 400$ К выставляются на стенках канала и каверны.

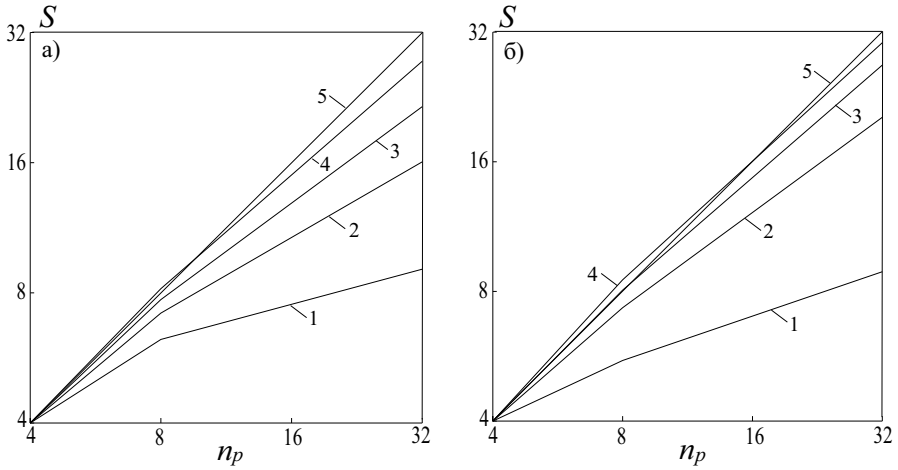

Рис. 11. Ускорение при использовании методов SOR (фрагмент а) и BiCGStab (фрагмент б) на сетке 10^3 (1), 20^3 (2), 30^3 (3), 40^3 (4). Кривая 5 соответствует теоретической зависимости

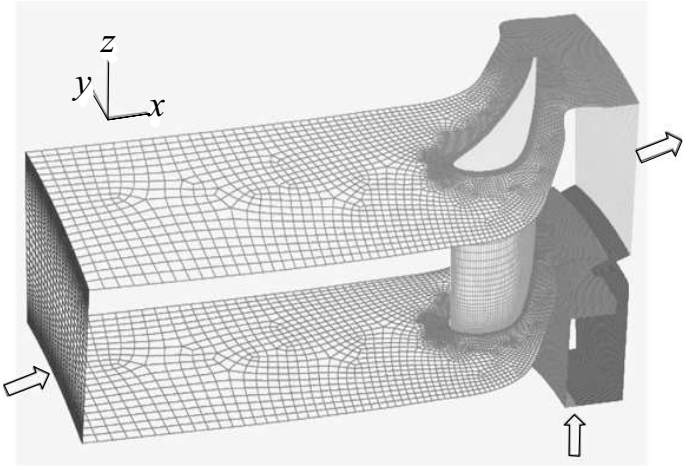

Рис. 12. Геометрия расчетной области и сетка

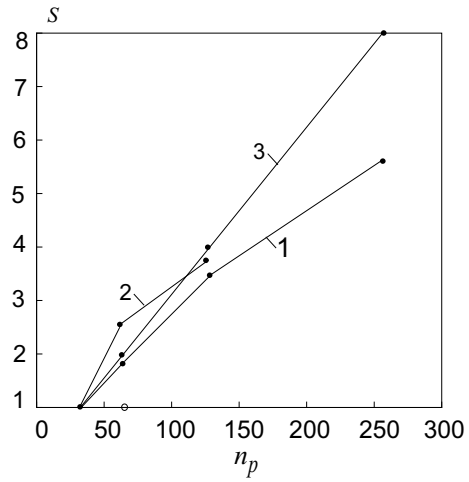

Рис. 13. Зависимость ускорения от числа процессоров

Стенка каверны вращается с угловой скоростью $\Omega = -1295$ 1/с.

В качестве начальных условий задаются составляющие скорости $v_x = 100$ м/с, $v_y = v_z = 0$ м/с, плотность $\rho = 1.18$ кг/м^3 и статическое давление $p = 2.2 \cdot 10^5$ Па.

Расчеты для модели, показанной на рис. 12, проводились на сетке, содержащей 839186 узлов и 795038 ячеек, из которых 87300 ячеек располагается на границе. Входная и выходная границы канала содержат по 1000 и 4000 ячеек, входная граница каверны — 880 ячеек, выходная граница каверны — 4000 ячеек, периодические границы — 11001 ячеек каждая, подветреная и надветреная поверхности профиля — 7950 и 6950 ячеек, нижняя и верхняя стенки канала — 21267 и 10276, стенка каверны — 12584 ячеек. В случае полной модели расчеты проводятся на сетке, содержащей $2.2 \cdot 10^7$ узлов и $6.6 \cdot 10^7$ граней. Для достижения сходимости делается 60000 итераций (в случае одного уровня сетки).

Зависимости ускорения от числа процессоров приведены на рис. 13 для полной модели (кривая 1) и для части расчетной области при использовании периодических граничных условий (значки $\circ$). Кривая 3 соответствует теоретической (линейной) зависимости. Ускорение оценивается как $S = t_p/t_{32}$, где t_p — время счета на p процессорах, t_{32} — время счета при использовании 32 процессоров (минимальное число процессоров, участвовавших в расчете). В идеальном случае $t_{32} = 32t_1$.

Полученные результаты для полной модели показывают хорошую производительность программного кода при пропорциональном изменении числа процессоров (максимальное число процессоров равняется 256). Ускорение в случае постановки периодических граничных условий значительно хуже (в расчетах использовалось от 32 до 64 процессоров) и код не является масштабируемым. Это связано с особенностями

реализации периодических граничных условий в методе контрольного объема [15].

Для улучшения характеристик производительности декомпозиция расчетной области производилась при помощи метода RGB. Архитектура системы состоит из 46 узлов по 32 процессора каждый с быстрым обменом данными между 32 процессорами в пределах узла и значительно более медленными коммуникациями между узлами. Метод RGB позволяет разместить процессы, которые интенсивно обмениваются данными, в пределах одного узла. Соответствующие результаты иллюстрируются кривой 2 на рис. 13, которая оказывается близкой к теоретической зависимости (кривая 3).

11. Заключение. Проведено тестирование программного кода и его составных частей на многопроцессорных вычислительных системах с общей и распределенной памятью. Установлена зависимость числа итераций и времени решения от количества подобластей при фиксированном размере задачи, а также зависимость числа итераций и времени решения от размера задачи, решаемой в каждой подобласти, при фиксированном числе разбиений. Выяснено влияние различных методов упорядочивания неизвестных на производительность и ускорение параллельных итерационных методов, а также их численная (зависимость числа итераций от размера задачи) и параллельная (зависимость ускорения от числа процессоров) масштабируемость.

СПИСОК ЛИТЕРАТУРЫ

1. *Foster I.* Designing and building parallel programs: concepts and tools for parallel software engineering. Boston: Addison-Wesley, 1995.
2. *Тарнавский Г.А., Шпак С.И.* Декомпозиция методов и распараллеливание алгоритмов решения задач аэродинамики и физической газовой динамики // Программирование. 2000. № 6. 45–57.
3. *Волков К.Н.* Дискретизация конвективных потоков в уравнениях Навье–Стокса на основе разностных схем высокой разрешающей способности // Вычислительные методы и программирование. 2004. **5**, № 2. 10–26.
4. *Белоцерковский О.М., Опарин А.М., Чечеткин В.М.* Турбулентность: новые подходы. М.: Наука, 2003.
5. *Karypis G., Kumar V.* A fast and high quality multilevel scheme for partitioning irregular graphs. University of Minnesota. Department of Computer Science. Technical Report TR95–035. Minneapolis, 1995.
6. *Walshaw C., Cross M., Everett M.G.* Parallel dynamic graph partitioning for adaptive unstructured meshes // Journal of Parallel and Distributed Computing. 1997. **47**, N 2. 102–108.
7. *Simon H.D.* Partitioning of unstructured problems for parallel processing // Computing Systems in Engineering. 1991. N 2. 135–148.
8. *Berger M.J., Bokhari S.H.* A partitioning strategy for nonuniform problems on multiprocessors // IEEE Transactions on Computers. 1987. **36**. 570–580.
9. *Jones M.T., Plassmann P.E.* Scalable iterative solution of sparse linear systems // Parallel Computing. 1994. **20**. 753–773.
10. *George A., Liu J.* Computer solution of large sparse positive definite systems. Englewood Cliffs: Prentice Hall, 1981.
11. *Nour-Omid B., Raefsky A., Lyzenga G.* Solving finite element equations on concurrent computers // Proc. of the Symposium on Parallel Computations and their Impact on Mechanics. Boston: American Society of Mechanical Engineers, 1986. 209–227.
12. *R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst.* Templates for the solution of linear systems: building blocks for iterative methods. Philadelphia: SIAM, 1994.
13. *Ортега Дж.* Введение в параллельные и векторные методы решения линейных систем. М.: Мир, 1991.
14. *Волков К.Н.* Реализация схемы расщепления на разнесенной сетке для расчета нестационарных течений вязкой несжимаемой жидкости // Вычислительные методы и программирование. 2005. **6**, № 2. 146–159.
15. *Волков К.Н.* Применение метода контрольного объема для решения задач механики жидкости и газа на неструктурированных сетках // Вычислительные методы и программирование. 2005. **6**, № 1. 47–64.
16. *Hackbusch W.* Multi-grid convergence theory // Lecture Notes in Mathematics. Vol. 960. Berlin: Springer-Verlag, 1982. 177–219.
17. *Burgess D.A., Crumpton P.I., Giles M.B.* A parallel framework for unstructured mesh solvers // Proc. of Working Conference on Programming Environments for Massively Parallel Distributed Systems (IFIP WG10.3.). Amsterdam: Elsevier, 1994. 97–106.
18. *Crumpton P.I., Giles M.B.* OPlus FORTRAN 77 library. Oxford, 2002.

Поступила в редакцию
27.01.2006