РОССИЙСКАЯ АКАДЕМИЯ НАУК
СИБИРСКОЕ ОТДЕЛЕНИЕ

АВТОМЕТРИЯ

2007, том 43, № 2

МОДЕЛИРОВАНИЕ В ФИЗИКО-ТЕХНИЧЕСКИХ ИССЛЕДОВАНИЯХ

УДК 519.6

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ ДЛЯ БОЛЬШИХ ПРИКЛАДНЫХ ЗАДАЧ: ПРОБЛЕМЫ И ТЕХНОЛОГИИ *

В. П. Ильин

Институт вычислительной математики и математической геофизики СО РАН, г. Новосибирск
E-mail: ilin@sscc.ru

Рассматриваются актуальные проблемы распараллеливания вычислительных методов и технологий для реализации основных этапов математического моделирования при решении широкого круга больших прикладных задач. Описываются модели вычислительно-информационных процессов, классификация основных типов алгоритмов и принципы их эффективного отображения на архитектуру компьютерных систем с распределенной и общей памятью.

Введение. В последние десятилетия развитие вычислительной техники, с одной стороны, и методов математического моделирования, с другой – подтверждают две давно установленные характерные тенденции. Во-первых, экспоненциальный рост быстродействия компьютеров (закон Мура) – примерно трехкратное увеличение каждые 18 месяцев (через несколько лет прогнозируется рождение петафлопной вычислительной системы [1]). Во-вторых, независимость времени решения «большой задачи» на ЭВМ в разные годы от уровня текущих мощностей компьютеров (инвариант академика Н. Н. Яненко). Данное утверждение устанавливает тесную взаимосвязь между практическими вычислительными потребностями и темпами развития компьютерных средств. Общеизвестно, что появление первых ЭВМ было простимулировано в значительной степени необходимостью проведения многочисленных расчетов при создании ядерного оружия, а дальнейший рост скорости машинных операций привел к усложнению математических задач и алгоритмов. К настоящему времени свершившимся фактом стало понимание того, что наряду с теоретическим и экспериментальным изучением

* Работа выполнена при поддержке Российского фонда фундаментальных исследований (грант № 05-01-00487), а также NWO-RFBR (грант № 047.016.008).

процессов или явлений не менее важным и самостоятельным – третьим путем познания – является математическое моделирование [2].

Строго говоря, здесь надо дать определение понятию «большая задача». Не вдаваясь в излишний формализм, можно принять следующую классификацию задач по уровню их вычислительной сложности. Будем называть задачу «малой», если она решается несколько секунд или минут; «средней», если требует для завершения несколько десятков минут; «большой», когда время расчета занимает часы, десятки часов или более. Однако вычислительными ресурсами является не только скорость выполнения операций, но и объем оперативной памяти, а эти две важнейшие компьютерные характеристики обычно балансируют: быстродействию в мегафлопах или гигафлопах соответствует память в мегабайтах или гигабайтах.

Большие наукоемкие задачи математического моделирования связаны, как правило, с решением систем многомерных нелинейных дифференциальных уравнений, описывающих взаимосвязанные физико-химические процессы (например, тепло- и массоперенос в многофазных средах + термоупругие напряжения + кинетические реакции + электромагнитные поля – такой набор моделей характерен для ряда междисциплинарных проблем). Важно также отметить, что нередко конечной целью является решение обратных задач, связанных с поиском заранее неизвестных параметров модели, которые обеспечивали бы требуемые свойства искомых решений, формально описываемых каким-либо целевым функционалом. При этом необходимо проводить многовариантные расчеты прямых задач, исходные данные для которых варьируются в условиях поставленных ограничений по некоторому оптимизационному алгоритму.

Для соответствующих крупномасштабных вычислительных экспериментов разница во времени расчетов (например, час или несколько суток) может играть принципиальную роль. Кардинальным решением в таких случаях является распараллеливание алгоритмов на многопроцессорных вычислительных системах, которое в идеальном случае обеспечивает линейное ускорение суммарного быстродействия, пропорциональное числу процессоров (за счет использования неоднородности памяти иногда удается получить даже сверхлинейное ускорение).

Нельзя не отметить, что зачастую использование многопроцессорных ЭВМ для решения больших задач имеет главной целью не само ускорение, а вообще возможность их реализации на одном компьютере из-за малого объема памяти.

В мировой литературе распараллеливание алгоритмов нередко отождествляется с понятием «высокопроизводительные вычисления», что обусловлено чрезвычайной сложностью эффективной организации вычислительного эксперимента на многопроцессорной ЭВМ [3, 4]. В силу сложности архитектуры современной компьютерной системы, связанной с многообразием функциональных арифметических и логических устройств, разнородностью иерархической памяти (общей и/или распределенной), управление вычислительным процессом без возможных конфликтов и простоев оборудования представляет собой отнюдь не тривиальную проблему. На практике эффективность использования процессоров может составлять около 10 % или менее.

В последние десятилетия вычислительное сообщество живет в ожидании очередной компьютерной революции, когда «кремниевые компиляторы» и индустриальные технологии программируемых логических интег-

ральных схем (ПЛИС) позволят быстро (и дешево) конструировать новые машины под задачи и алгоритмы, однако в настоящее время специалистам по вычислительным наукам (Computer Science) приходится рассматривать существующие и быстро меняющиеся архитектуры как свыше данную реальность, которую надо тщательно изучать, чтобы добиться хорошего конечного результата. Следует отметить, что провозглашенное в 1970-е годы академиком Г. И. Марчуком направление «Отображение алгоритмов на архитектуру ЭВМ» сейчас очень актуально.

Очевидно, что чем сложнее архитектура многопроцессорной системы, тем на ней труднее программировать. Математическое обеспечение и теория параллельного программирования зарождались и активно развивались еще в 1970-е годы, однако производственных «коммерческих» систем автоматического распараллеливания алгоритмов пока не существует. Имеющиеся системы MPI и OpenMP, организующие работу по передаче сообщений между вычислительными узлами с разделенной или общей памятью, а также параллельные отладчики и профилировщики позволяют анализировать «узкие места» и существенно повышать производительность разрабатываемых прикладных программ.

Важно также отметить большую разницу между двумя такими понятиями, как распараллеливание задачи и алгоритма. Нередко численные методы, оптимальные для «классических» последовательных вычислений, плохо распараллеливаются, поэтому приходится конструировать специальные приемы, обеспечивающие успешное достижение конечной цели – быстрого решения больших задач.

В предлагаемой работе проведен концептуальный обзор проблем распараллеливания вычислительных методов и технологий для реализации основных этапов математического моделирования при решении широкого круга больших прикладных задач, но никак не выдача конкретных рекомендаций, которые в частных случаях требуют специальных исследований.

1. О моделях вычислений на многопроцессорной системе. Пусть $T_p(A)$ – время решения некоторой задачи A на p процессорах вычислительной системы. Тогда критериями эффективности распараллеливания являются ускорение и коэффициент использования (загрузки) процессоров

$$R_p(A) = T_1(A)/T_p(A); \qquad E_p(A) = R_p(A)/p. \tag{1}$$

В простейшем приближении время расчета T (индекс p и/или имя задачи A будем опускать, где это возможно по контексту) состоит из двух частей: суммарной длительности выполнения арифметических операций T_a и времени межпроцессорных обменов данными T_c, когда арифметические устройства простаивают:

$$T = T_a + T_c, \qquad T_a = N_a \tau_a, \quad T_c = N_c \tau_0 + N_d \tau_c. \tag{2}$$

Здесь N_a – количество исполненных арифметических операций; N_c – количество информационных обменов; N_d – средний объем одного передаваемого пакета данных. Величины

$$\tau_a \ll \tau_c \ll \tau_0 \tag{3}$$

представляют среднее время выполнения одного арифметического действия, время передачи одного числа и время задержки пересылаемого пакета соответственно.

Предполагается, что все арифметические операции выполняются над числами в стандартном представлении с плавающей запятой и с двойной точностью, необходимой (и в подавляющем большинстве случаев достаточной) при решении больших задач; объемы передаваемых данных измеряются в таких же числах. Величина задержки τ_0 обусловлена программируемой настройкой передающих устройств на реализацию конкретной пересылки.

Формула (2) содержит большое количество упрощений реальной модели вычислений. В действительности разные арифметические операции реализуются за различные времена. Например, на Pentium IV деление чисел с двойной точностью выполняется за 38 тактов, умножение – за 7 и сложение (вычитание) – за 5. Кроме того, необходимо иметь в виду, что современный процессор содержит несколько устройств для выполнения действий над числами с плавающей запятой и для быстрых логических операций. Работа сумматоров и умножителей может быть организована в конвейерном режиме, а также совмещена по времени с функционированием устройств обращения к памяти.

Реальная длительность выполнения алгоритма значительно зависит от эффективности компьютерной реализации доступа к данным, что вследствие неоднородности памяти представляет сложную проблему [5]. В простейшем случае следует выделить три уровня памяти, отличающиеся своими объемами и скоростями передачи информации: сверхбыстрые регистры, из которых непосредственно черпают свои операнды арифметические устройства, быстрый кэш, который играет роль промежуточного звена и в действительности имеет свою внутреннюю иерархическую структуру, а также главная оперативная память, через которую при необходимости больших информационных обменов осуществляется связь с внешним миром. В многопроцессорных системах память бывает двух основных типов: общая и распределенная, существенно различные в организации доступа. В новых многоядерных архитектурах, имеющих по несколько процессоров на одном кристалле, кэш также является общим для вычислительного узла.

Наихудшими с точки зрения производительности являются методы с сильной зависимостью от данных, которые для относительно малого объема вычислений требуют больших обменов информацией. При этом имеет важное значение, где расположить необходимые для реализации алгоритма данные: в кэш или в главной памяти. В силу неоднородности памяти можно получить кажущийся на первый взгляд парадоксальным эффект – сверхлинейное ускорение параллельных вычислений, когда коэффициент ускорения R_p растет быстрее, чем количество используемых процессоров p. Достигается это в случаях, когда при малых значениях p требуемые данные умещаются только в главной памяти, а при больших p их удается разместить в кэш соответствующих процессоров, что значительно сокращает время обменов T_c и может привести к значениям $E_p > 1$.

Как видно из приведенного краткого обзора особенностей машинных реализаций алгоритмов, формулы (2) являются слишком грубыми, а чтобы построить более или менее хорошую аппроксимацию времени выполнения конкретного алгоритма, в модель его вычислительной сложности надо закладывать слишком много параметров. Реально здесь сможет помочь только имитационное моделирование вычислительной системы, однако промоде-

лировать реализацию решения большой задачи на ЭВМ – это сама по себе еще бóльшая задача. Более того, время решения конкретного алгоритма на выбранном компьютере может существенно зависеть от языка, на котором он запрограммирован, какой компилятор при этом используется и какие опции компилятора (режимы оптимизации) применяет пользователь.

Таким образом, при исследовании или попытке оптимизации производительности вычислительной системы, тем более многопроцессорной, приходится опираться только на общие интуитивные соображения о влиянии тех или иных факторов. В этих условиях, естественно, главным методическим инструментом становится численный эксперимент, сводящийся к сравнительному анализу результатов измерений производительности при различных вариантах математической и/или программной реализации алгоритмов, а также режимов их исполнения в конкретной операционной среде.

2. Задачи и алгоритмы математического моделирования. Примером достаточно общей математической формулировки может быть начально-краевая задача

$$L\bar{u} \equiv D\frac{\partial \bar{u}}{\partial t} + \nabla A \nabla \bar{u} + B\nabla \bar{u} + C\bar{u} = f; \quad \bar{r} \equiv (x, y, z) \in \Omega, \ \ 0 < t \le T \le \infty, \tag{4}$$

где $\bar{u} = \{u_1, \ldots, u_m\}$, $\bar{f} = (f_1, \ldots, f_m)$ – неизвестная и заданная вектор-функции соответственно; t и x, y, z – временная и пространственные координаты; Ω – расчетная область с границей Γ; A, B, C, D – некоторые матрицы порядка m, элементы которых (как и f) могут зависеть и от независимых переменных, и от искомого решения $\bar{u}$. Расчетная область может состоять из подобластей $\left(\Omega = \bigcup_{s=1}^{N_s} \Omega_s\right)$ с существенно различными материальными свойствами, т. е. с разными представлениями коэффициентов или даже типов решаемых в них дифференциальных уравнений. Границу Γ можно представить состоящей из частей Γ_D и Γ_N, на которых заданы краевые условия различных типов:

$$\bar{u}(\bar{r}) = \bar{g}_D, \quad \bar{r} \in \Gamma_D; \qquad D_N \bar{u} + A_N \nabla_n \bar{u} = \bar{g}_N, \quad \bar{r} \in \Gamma_N, \tag{5}$$

где ∇_n – проекция градиента в направлении внешней нормали к Γ_N; $\bar{g}_D, \bar{g}_N$ и D_N, A_N – вектор-функции и матрицы соответственно, в общем случае зависящие от $\bar{u}$. Задача (1), (2) замыкается начальными данными

$$\bar{u}(\bar{r}, t = 0) = \bar{u}^0(\bar{r}), \quad \bar{r} \in \Omega. \tag{6}$$

Система соотношений (4)–(6) описывает непосредственно так называемую прямую математическую задачу. Более общими являются обратные задачи, в которых любые из исходных данных могут зависеть от вектора параметров $\bar{p} = (p_1, \ldots, p_M)$, значения которых надо оптимизировать по условию минимума целевого функционала

$$\Phi_0(\bar{u}(\bar{r}, t, \bar{p}_{\text{opt}})) = \min_{\bar{p}} \Phi_0(\bar{u}(\bar{r}, t, \bar{p})) \tag{7}$$

при дополнительных линейных или нелинейных ограничениях

$$p_k^{\min} \le p_k \le p_k^{\max},\ k=1,\dots,M_1; \quad \Phi_l(\bar{u}(\bar{r},t,\bar{p})) \le \delta_l,\ l=1,\dots,M_2, \quad (8)$$

где $p_k^{\min}$, $p_k^{\max}$, δ_l и $M_1 + M_2 = M$ суть заданные величины, а Φ_l – некоторые функционалы от решения. Типичным примером обратной постановки является идентификация параметров модели, например коэффициентов уравнений (4), (5) или геометрических характеристик подобластей Ω_s, на основе сравнения расчетных величин с результатами натурных измерений (в таком случае минимизируемый целевой функционал Φ_0 – это отклонение компьютерных и реальных данных [6]).

Реализация математического моделирования сложных процессов или явлений представляет собой многогранный вычислительный эксперимент, различные этапы которого имеют свои существенные особенности с точки зрения эффективности их распараллеливания [7]. Соответствующие стадии реализуются с помощью своих компонент пакета прикладных программ (ППП), однако анализ их структуры или требований к разработке не входит в задачу данной работы.

1. Решение задачи вида (4)–(8) начинается с формулировки исходной информации и ввода ее в компьютер с помощью каких-то выбранных программно-технических средств.

Этот этап включает в себя контроль и предварительную обработку (препроцессинг) пользовательских данных, которые можно разбить на две основные группы: геометрические и функциональные, отвечающие за конфигурацию расчетной области и описание решаемых уравнений. Анализ и модификация этой информации составляют содержание геометрического и функционального моделирования [8, 9], лежащего в основе управления вычислительным экспериментом для многовариантных и обратных задач и заканчивающегося принятием решений по результатам расчетов. Формально рассмотренный этап завершается созданием геометрической структуры данных (ГСД) и функциональной структуры данных (ФСД), обеспечивающих автоматизацию построения алгоритмов и полностью представляющих математическую постановку интересующего класса задач всем остальным вычислительным модулям ППП. Отметим, что некоторые задачи геометрического моделирования успешно решаются в системах автоматизации проектирования (CAD- и CAE-системы), однако многие актуальные проблемы здесь остаются открытыми.

В сложных краевых задачах с десятками и сотнями подобластей конфигурация расчетной области может быть представлена в виде макросети, состоящей из макроэлементов с допустимыми различными степенями вложенности, макрограней, макроребер и вершин, для которых должны быть заданы соответствующие геометрические и топологические характеристики. Типичные модификации отдельных объектов или их совокупностей – это сдвиги, повороты или сжатие (растяжение), которые и составляют главные операции геометрического моделирования.

2. Основные современные алгоритмы решения многомерных задач математической физики – сеточные, основанные на дискретизации расчетной области, т. е. разбиении ее на конечные объемы или элементы. За исключением простейших случаев равномерных или регулярных разбиений стандартного вида конфигураций, построение «хороших» сеток в сложных трехмерных

ситуациях, адаптированных под особенности расчетной области и искомых решений, представляет собой далеко не простую проблему как в математическом плане, так и в технологии реализации (см., например, работу [10] и библиографию в ней). Алгоритмически это сводится или к триангуляции Делоне, или к построению ячеек Дирихле – Вороного, или к другим геометрическим подходам, в любом случае связанным с удовлетворением жестких ограничений на характеристики базовых сеточных объектов: узлов, ребер, граней, объемов или элементов. В адаптивных сетках вершины углов расчетной области должны являться узлами сетки, а ребра и сегменты граничных поверхностей должны состоять целиком из совокупностей сеточных ребер и граней. Необходимо также выполнение определенных условий для углов сетки, для сравнительных размеров соседних ячеек и т. д. Конструирование в определенном смысле квазиоптимальных сеток может сводиться к решению нелинейных дифференциальных уравнений, представляющему собой не менее трудоемкую вычислительную проблему, чем исходная задача моделирования. Поэтому в практических (коммерческих) программных разработках – генераторах сеток – идут на палиативные решения, исходя из оптимизации соотношения стоимости и качества. По своей сложности сетки делятся на две группы: структурированные, в которых нумерация объектов описывается некоторой простой формулой, и неструктурированные. Компромиссным вариантом являются квазиструктурированные сетки, состоящие из достаточно простых сеточных подобластей [11], в каждой из которых конечные элементы или объемы могут иметь свои формы: тетраэдры, параллелепипеды или криволинейные фигуры.

С информационной точки зрения результатом этой стадии является сеточная структура данных (ССД), описывающая не только топологию всех сеточных объектов, но и необходимые ссылки на исходные геометрические и функциональные характеристики задачи. В зависимости от конкретных целей могут употребляться поузловые и/или поэлементные структуры данных, использующие различные упорядоченности и иерархии сеточных объектов.

3. На построенной сетке могут реализовываться аппроксимации исходных краевых задач, среди которых наиболее распространенными являются методы конечных разностей, методы конечных объемов и методы конечных элементов (МКЭ) (см. [12, 13] и библиографию в них), базирующиеся на дискретизации классических или обобщенных математических формулировок. Общим положительным свойством этих разных подходов является применение локально-элементных технологий, основанных на независимых вычислениях локальных матриц баланса или жесткости и сборке глобальных матриц и векторов правых частей для систем линейных алгебраических уравнений (СЛАУ). Этот подход позволяет идеально распараллеливать операции данного этапа.

Если говорить о задаче аппроксимации в общем виде, то она отличается большим разнообразием, определяемым как типом решаемых дифференциальных уравнений и граничных условий, так и классической или обобщенной формулировкой, а также применяемым порядком приближения, видом конфигурации конечных элементов или объемов и различными технологическими аспектами (сеточной структурой данных, упорядоченностью узлов сетки и т. д.).

Когда исходная задача является нелинейной, то решается она на основе последовательной реализации линейных задач, коэффициенты которых должны пересчитываться в процессе квазилинеаризации. При моделировании

нестационарных процессов после дискретизации временной переменной на каждом шаге будут формироваться соответственно линейные или нелинейные системы.

Порождаемые в многомерных задачах СЛАУ имеют сильно разреженные матрицы высокого порядка, для хранения которых используются различные сжатые форматы хранения коэффициентов и матричных портретов, составляющие в совокупности алгебраическую структуру данных (АСД). В вычислительно-алгебраическом мировом сообществе сложилось около десятка различных общепринятых форматов хранения больших разреженных матриц [14, 15]. В качестве примера поясним принцип разреженного строчного формата: ненулевые элементы матрицы $A = \{a_{i,\,j}\}$ порядка N располагаются последовательно построчно в вещественном массиве AA длины NA; для поиска в AA значений $\{a_{i,\,j}\}$ формируется целый массив указателей IA длины N, где $IA(i)$ – номер начала в массиве AA элементов i-й строки матрицы (в частности, $IA(N) = NA$); кроме того, задается целый массив столбцовых индексов IJ длины NA, как и AA, каждый элемент $IJ(k)$ которого дает номер столбца j для матричного коэффициента $a_{i,\,j}$, находящегося в соответствующей k-й позиции массива AA.

Если как иллюстрацию рассмотреть характерную для практики матрицу порядка $N = 10^7$ со средним числом ненулевых элементов в строке $m \approx 100$, то соотношение $NA = mN \approx 10^9 \ll N^2 = 10^{14}$ свидетельствует об огромной экономии памяти при использовании такого подхода в сравнении с объемом памяти для хранения плотной матрицы. Однако следует иметь в виду, что при этом существенно усложняются векторно-матричные и другие алгебраические операции, поскольку многоступенчатый доступ к элементам матрицы намного снижает производительность вычислений. Такая ситуация служит типичным примером необходимости поиска «золотой середины» при экономии оперативной памяти и/или достижения максимального быстродействия ЭВМ. Причем оптимальный компромисс будет зависеть как от конкретной решаемой задачи, так и от архитектуры компьютера.

4. Наиболее ресурсоемким этапом математического моделирования является решение СЛАУ, поскольку объем вычислений здесь растет нелинейно с увеличением числа узлов сетки. Вычислительные, в том числе параллельные, методы линейной алгебры – одна из областей математики, особенно активно развивающаяся в последние десятилетия [16]. Причем здесь важно отметить не только большое количество появившихся новых алгоритмов, теорем, статей и монографий, но также многообразное программное обеспечение, как свободно доступное, так и коммерческое. В сети Интернет можно найти информацию о хорошо известных библиотеках и пакетах программ по решению СЛАУ и других задач линейной алгебры: BLAS, LAPACK, ScaLapack (параллельная версия LAPACK), SparseKit, Petsc, Linpack и т. д. В частности, следует отметить такой самостоятельный и сложный класс проблем, как вычисление собственных чисел и векторов (до нескольких десятков и сотен) больших матриц, актуальных в электродинамике, механике и многих других приложениях, связанных с анализом устойчивости, когда решение СЛАУ также требуется на промежуточных этапах.

Методы решения алгебраических систем делятся на две основные группы: прямые алгоритмы и итерационные. Прямые алгоритмы обеспечивают нахождение точных решений за конечное число операций при отсутствии округлений в арифметических действиях и наиболее эффективны для систем

относительно небольшого порядка (до нескольких сот тысяч). Здесь высоко развиты технологические приемы работы с разреженными матрицами, описанные в монографиях [17–19] и в большом количестве специальных работ. Однако при размерах СЛАУ около миллиона и выше вне конкуренции оказываются итерационные алгоритмы как по числу действий, так и по объему используемой оперативной памяти, которые приближенно оцениваются формулами

$$Q_s \cong \alpha_s N^{\gamma_s}; \qquad P \cong \beta_s N^{\delta_s}, \tag{9}$$

где $s=1$ и $s=2$ относятся к прямым и итерационным алгоритмам соответственно; N – порядок СЛАУ; α_s, β_s, γ_s и δ_s – не зависящие от N коэффициенты, причем $\alpha_1 < \alpha_2$, $\beta_1 < \beta_2$. Если, например, алгебраическая система получена из аппроксимации дифференциальной задачи на сетке с числом узлов $N_x N_y N_z$, то типичными параметрами в (9) являются:

$$\gamma_1 \approx 7/3; \quad \gamma_2 \approx 1+\kappa, \ 1/6 \le \kappa \le 1/3; \quad \delta_1 \approx 5/3; \quad \delta_2 \approx 1. \tag{10}$$

Наиболее ограничивающим здесь оказывается тот факт, что при $N \ge 10^6$ для прямых методов не хватает оперативной памяти даже на современных компьютерных процессорах, поэтому остановимся на технологических особенностях реализации только итерационных методов. Оговоримся, что в оценках (10) не будем рассматривать сверхбыстрые прямые алгоритмы для решения слишком частных задач с разделяющимися переменными, а также оптимальные по порядку, но плохо приспособленные для распараллеливания многосеточные методы.

Большое разнообразие СЛАУ классифицируется по структурным, спектральным и другим признакам, определяющим самые экономичные для конкретных случаев алгоритмы. Наиболее характерные типы матриц – это вещественные и комплексные, эрмитовые и неэрмитовые, симметричные и несимметричные, положительно-определенные и неопределенные, ленточные с различной шириной ленты, специальные блочные матрицы и т. д.

Множество итерационных методов решения СЛАУ характеризуется двумя основными приемами: выбором предобусловливающих матриц и способом ускорения итерационного процесса. Если исходная линейная система имеет вид

$$Au = f, \tag{11}$$

то ее предобусловливание можно представить как сведение к другой, эквивалентной в некотором смысле, СЛАУ:

$$\widetilde{A}\widetilde{u} \equiv BACC^{-1}u = Bf \equiv \widetilde{f}, \quad \widetilde{A} = BAC, \ \ \widetilde{u} = C^{-1}u, \tag{12}$$

где невырожденные матрицы B и C называются левым и правым предобусловливателями соответственно и находятся по условию простой обратимости и возможной минимизации числа обусловленности предобусловленной матрицы $\mathrm{cond}\widetilde{A} = \left\|\widetilde{A}\right\|\left\|\widetilde{A}^{-1}\right\|$, что в конечном счете и определяет скорость сходимости и экономичность реализации каждого шага итераций.

Один из основных подходов к предобусловливанию СЛАУ представляет собой семейство методов неполной факторизации, или неполного LU-разло-

жения (ILU – incomplete LU-decomposition). Его можно представить в форме (12), положив $C = I$ (единичная матрица) и определив левый предобусловливатель как

$$B = (G - \overline{L})G^{-1}(G - \overline{U}). \tag{13}$$

Здесь G – некоторая диагональная или ленточная матрица, а $\overline{L}$ и $\overline{U}$ – строго нижняя и верхняя треугольные матрицы. Если исходную матрицу представить в виде $A = D - L - U$, где D, L и U суть ее диагональная, нижняя и верхняя треугольные части соответственно, то для одного из популярных методов – симметричной последовательной верхней релаксации (SSOR – Symmetric Successive Over Relaxation) предобусловливающая матрица принимает форму (13) с $G = \frac{1}{\omega}D$, $\overline{L} = L$, $\overline{U} = U$ (ω – итерационный (релаксационный) параметр).

Реализация методов ILU требует решения нижней и верхней треугольных систем на каждой итерации. Различные алгоритмы этого семейства связаны с определением матриц $\overline{L}$ и $\overline{U}$ большей или меньшей разреженности. При увеличении в них количества ненулевых элементов можно значительно уменьшить число обусловленности матрицы $\widetilde{A}$, но при этом «дорожает» реализация каждого итерационного шага. Важно отметить, что при использовании разреженных матричных форматов (в отличие от плотных) выполнение ILU-алгоритмов гораздо сложнее, поскольку включает непростую логическую задачу символьной факторизации.

Вычисление последовательных приближений уже для предобусловленной системы $\widetilde{A}\widetilde{u} = \widetilde{f}$ при заданном произвольном начальном приближении u^0 схематично можно выразить формулами

$$\begin{aligned} &\widetilde{r}^{\,0} = \widetilde{f} - \widetilde{A}\widetilde{u}^{\,0}; \\ &\widetilde{u}^{\,n} = \widetilde{u}^{\,n-1} + \alpha_n p^{n-1}; \quad \widetilde{r}^{\,n} = \widetilde{r}^{\,n-1} - \alpha_n \widetilde{A} p^{n-1}, \quad n = 1,2,\ldots, \end{aligned} \tag{14}$$

где p^0, p^1, ... – некоторые направляющие векторы, определяемые тем или иным образом через векторы предобусловленных невязок $\widetilde{r}^{\,0}$, $\widetilde{r}^{\,1}$, ...; α_n – соответствующие итерационные параметры. В простейшем случае чебышевского ускорения полагается $p^n = \widetilde{r}^{\,n}$, а значения α_n определяются через оценки границ спектра матрицы $\widetilde{A}$, который может быть как вещественным, так и комплексным.

Большое распространение получили градиентные, или вариационные, алгоритмы, в которых итерационные параметры вычисляются адаптивным образом по апостериорной информации из принципа минимизации какого-либо функционала, например нормы невязки. При этом оптимизация процесса производится в подпространствах Крылова, являющихся линейной оболочкой совокупности векторов $\widetilde{r}^{\,0}$, $\widetilde{A}\widetilde{r}^{\,0}$, ..., $\widetilde{A}^n\widetilde{r}^{\,0}$, а направляющие векторы p^n находятся как линейные комбинации только последних векторов последовательности через формулы вида

$$p^n = r^n + \sum_{k=n-m_n}^{n-1} \beta_{n,k}\, p^k; \quad p^0 = \widetilde{r}^{\,0}. \tag{15}$$

Здесь величины $\beta_{n,k}$, как и α_n из (14), рассчитываются с помощью скалярных произведений вспомогательных векторов, а целочисленные m_n определяют, сколько предыдущих направляющих векторов p^k необходимо сохранить в памяти. Если матрица $\widetilde{A}$ самосопряженная, то $m_n = 1$ и направляющие векторы в (15) определяются из коротких двучленных рекурсий.

Мы не будем рассматривать все многообразие существующих современных алгоритмов, тем более проводить их сравнительный анализ, поскольку нашей целью является выделение общих основных технологических принципов их распараллеливания. Поэтому ограничимся только перечислением основных методов пространств Крылова (самое общее название такого класса итерационных процессов) и их главных особенностей при решении вещественных СЛАУ.

Классические методы сопряженных градиентов (CG – Conjugate Gradient) и сопряженных невязок (CR – Conjugate Residual), применяемые для симметричных положительно-определенных (с.п.о.) и полуопределенных систем, являются самыми экономичными: они содержат двучленные рекурсии в формулах (15) для направляющих векторов и только одно векторно-матричное умножение на каждой итерации. Обобщением этих алгоритмов для несимметричных систем являются методы бисопряженных градиентов (BiCG) и невязок (BiCR), которые уже не имеют минимизирующих свойств, но сохраняют проекционные: векторы $\widetilde{r}^k$ и p^k биортогональны дополнительно вычисляемым векторам $\hat{r}^k$ и $\hat{p}^k$ соответственно. В этих процессах все рассчитываемые векторы также определяются из двучленных рекурсий, но реализация каждой итерации требует два векторно-матричных умножения.

Наиболее быструю скорость сходимости для несимметричных систем имеют обобщенные методы минимальных невязок (GMRES) и сопряженных невязок (GCR), которые минимизируют норму невязки, но имеют различные ортогональные свойства вычисляемых векторов. Отметим также метод полусопряженных невязок (SCR), который является устойчивой версией алгоритма GCR, использующей модифицированный процесс ортогонализации Грама – Шмидта для направляющих векторов. Эти алгоритмы требуют только по одному векторно-матричному умножению на итерационном шаге, но используют «длинные» рекурсии вида (15), в которых необходимо хранить все направляющие векторы p^k. Существуют различные подходы к ослаблению этого слишком жесткого ограничения, однако во всех случаях экономия памяти приводит к замедлению скорости сходимости итераций.

Проблема выбора оптимального алгоритма для конкретного класса СЛАУ далека от завершения. Пополнение знаний здесь происходит путем накопления экспериментальных результатов сравнительного тестирования всевозможных методов на коллекциях типовых матриц, например доступном в сети Интернет наборе MATRIX MARKET.

5. После завершения конечного или промежуточного этапа моделирования следует стадия постпроцессирования, т. е. предварительной обработки, визуализации и анализа полученных результатов, которые представляются значениями рассчитанных приближенных решений, определяемых или в узлах, или в серединах ребер, или в других точках сеточных элементарных объектов. Обычные наглядные формы представления трехмерных скалярных или векторных функций – это изолинии в задаваемых сечениях расчетной области, изоповерхности, силовые линии, графики, таблицы и т. д. [20]. Стан-

дартный технологический путь в данном случае – это выбор приемлемой графической структуры данных (ГрафСД) и существующей развитой системы визуализации. Идеальный формат конечных результатов вычислительного эксперимента должен согласовываться с информационной базой системы автоматизации проектирования или системы принятия решений, для целей которых и проводится математическое моделирование.

Следует отметить, что этап предварительной обработки массивов сеточных значений функций не требует слишком много арифметических операций. Однако непосредственная визуализация большого объема графического и особенно мультипликационного материала может потребовать огромных компьютерных ресурсов, поэтому проблема распараллеливания здесь особенно актуальна.

3. Стратегии и тактики распараллеливания. Проанализируем последовательно технологические особенности распараллеливания всех рассмотренных в разд. 2 стадий математического моделирования, выделим их «узкие места» и наметим общие принципы повышения эффективности решения больших задач на многопроцессорных вычислительных системах. Предлагаемое исследование достаточно абстрактно и не включает детальный анализ особенностей отдельных компьютерных архитектур или программных платформ, потому что эти вопросы требуют специального изучения. Остановимся на распараллеливании одной задачи (возможно, включающей многовариантные расчеты), так как решение различных задач на «своих» процессорах – это выполнение независимых вычислительных процессов, не требующее применения специальных алгоритмов их согласования.

Для распараллеливания задач рассматриваемых классов сложились следующие основные подходы, которые могут применяться или автономно, или в различных сочетаниях.

1. *Декомпозиция многомерных областей.* Расчетная область Ω и ее дискретный сеточный аналог Ω^h разбиваются на соответствующие пересекающиеся или непересекающиеся подобласти Ω_l и Ω_l^h, $l = 1, \ldots, N_\Omega$, для того чтобы исходную полную задачу свести к решению N_Ω подзадач, каждую из которых можно реализовывать одновременно и независимо (в определенном смысле) на своем процессоре. В конечном счете эти подзадачи являются взаимосвязанными, и поэтому между процессорами периодически необходимо осуществлять информационные обмены, что является платой за параллельные вычисления на системах с распределенной памятью, на которые исторически как раз были ориентированы алгоритмы декомпозиции.

Важно заметить, что в этом подходе распараллеливается именно задача, так как применяемый алгоритм становится существенно другим (как правило хуже, при решении на одном процессоре). Предтечей данного подхода является известный более 100 лет альтернирующий метод Шварца, который использовался при доказательствах теорем о существовании дифференциальных уравнений.

Разбиение на подобласти надо проводить исходя из принципа равномерности загрузки процессоров, т. е. одинаковой вычислительной сложности подзадач, и это никак не связано со свойствами физических подобластей исходной задачи.

2. *Расщепление по физическим процессам.* Этот способ используется для моделирования многоплановых процессов, описываемых системами дифференциальных уравнений, которые как-то разбиваются на подсистемы, реша-

емые на промежуточных шагах независимо и параллельно. При этом зачастую приходится «портить» исходную задачу и разбивать ради упрощения алгоритма одно уравнение на несколько. Например, перенос тепла вследствие диффузионного, конвективного и радиационного процессов может рассчитываться независимо по процессам.

3. *Расщепление по направлениям*. Многомерные задачи решаются «в лоб» гораздо тяжелее одномерных, поэтому один из популярных классов алгоритмов – методы переменных направлений – связан с именами Писмана, Речфорда, Дугласа, Н. Н. Яненко, Г. И. Марчука, А. А. Самарского и других. Эти методы применяются для решений как стационарных, так и нестационарных задач.

Существует также огромное количество приемов распараллеливания вспомогательных подзадач, составляющих необходимый математический инструментарий: векторно-матричные операции, быстрое преобразование Фурье и т. д., и их эффективное использование очень важно, но не это является сейчас нашей целью.

Реализация этапа препроцессирования, связанного с геометрическим и функциональным моделированием, не является ресурсоемкой, а его конечные результаты – ГСД и ФСД – не требуют для своего хранения большого объема памяти, так как количество геометрических и функциональных объектов исчисляется только десятками или сотнями.

Поэтому распараллеливание алгоритмов данной стадии фактически не стоит в повестке дня и может быть выполнено на одном головном компьютере, а результаты скопированы и разосланы по всем остальным процессорам. Главная технологическая задача в данном случае – это обеспечение пользователя дружественным графическим интерфейсом, позволяющим вводить изобразительную и текстовую информацию, контролировать ее и редактировать. И экономить здесь нужно не машинные, а человеческие ресурсы.

Дискретизация расчетной области и формирование сеточной структуры данных могут составлять или тривиальную, или очень сложную задачу. В любом случае это уже начало процесса распараллеливания, так как и исходные данные этого этапа, и тем более результаты располагаются в памяти разных процессоров. Естественно при этом, что логическая сеть из расчетных подобластей должна отображаться на виртуальную и/или физическую сеть из процессоров.

Здесь уместно рассмотреть важный методический вопрос: какое разбиение расчетной сеточной области на сеточные же подобласти является оптимальным? Для понимания сути возьмем простейший пример кубической сеточной области с числом узлов N^3, разбитой на m^3 одинаковых кубических подобластей, каждая из которых имеет $(N/m)^3$ узлов. Будем считать, что решение каждой из подзадач в подобласти на соответствующем процессоре требует выполнения $(N/m)^{3\alpha}$, $\alpha \geq 1$, арифметических действий, т. е. связано с «объемом» Ω_l^h. Количество же информационных межпроцессорных обменов для каждой подобласти пропорционально ее «поверхности», т. е. $(N/m)^2$. Таким образом, отношение времени решения каждой подзадачи к времени обменов пропорционально $(N/m)^{3\alpha-2}$. Отсюда следует, что с увеличением числа подобластей и процессоров эффективность распараллеливания (E_p в (1)) падает вследствие относительного роста коммуникационных потерь, хотя ускорение вычислений R_p может расти.

Пусть теперь расчетная область разбивается не на «кубики», а на m одинаковых сеточных «полос» с числом узлов в каждой N^3/m, а конфигурация вычислительной сети из m процессоров представляет собой одномерную «линейку». Оценивая аналогично предыдущему трудоемкость каждой подзадачи величиной $(N^3/m)^\alpha$ и учитывая, что площадь общих граней между полосами равна N^2, получаем отношение времени счета к времени обменов пропорциональным величине $N^{3\alpha-2}/m^\alpha$. Это означает, что такая декомпозиция гораздо хуже при $m \gg 1$.

Если же взять промежуточную компьютерную архитектуру в виде квадратной сети с числом процессоров m^2, то при соответствующем «двумерном» разбиении трехмерной расчетной области получаем времена счета, обменов и их отношение пропорциональными величинам $(N^3/m^2)^\alpha$, N^2/m и $N^{3\alpha-2}/m^{2\alpha-1}$ соответственно.

Многообразие математических и технологических проблем построения сеток можно представить из одного только перечисления их возможных конфигураций и характеристик: равномерные и неравномерные, структурированные и неструктурированные, согласованные и несогласованные, конформные и неконформные, статические и динамические, m-угольные ($m = 3, 4, \ldots$), адаптивные, вложенные, криволинейные, ортогональные, триангуляции Делоне и т. д. Один только анализ их особенностей и критериев качества достоин развернутого многостраничного обзора.

Формально весь этот комплекс алгоритмических вопросов можно закрыть следующим образом: в каждой из подобластей Ω_l^h сетка строится своей программой, совокупность которых составляет библиотеку сеточных генераторов. При этом согласование независимых дискретизаций может осуществляться предварительным разбиением макрограней сеточных подобластей. Разумеется, каждый сеточный объект должен специфицироваться своим региональным и глобальным номерами.

Тривиальная на первый взгляд проблема нумерации узлов и других элементарных объектов может оказать решающее влияние на эффективность реализации алгоритмов, а также их распараллеливание. Дело в том, что упорядоченность соответствующих компонент искомых функций определяет собой ленточную структуру СЛАУ, которая является главным фактором в объемах вычислений и требуемой памяти для разреженных матриц. Можно отметить следующую закономерность: чем сложнее и эффективнее сетка в «интеллектуальном» плане, тем труднее она реализуется в смысле технологии распараллеливания. Примерами могут служить адаптивные сетки, учитывающие свойства рассчитываемых решений, в том числе динамические, применяемые для сильно меняющихся во времени свойств нестационарных задач, а также многосеточные алгоритмы, основанные на последовательно вложенных дискретизациях и обеспечивающие оптимальные по порядку методы.

Итоговая распределенная ССД, сформированная на поэлементном принципе, может быть кратко описана следующим образом. Для каждой замкнутой сеточной подобласти $\overline{\Omega}_l^h$ формируется список конечных элементов $E_{l,m}$ ($m = 1, \ldots, M_l$) – тетраэдров, параллелепипедов, призм или других фигур, – для каждого из которых задаются: номер своей расчетной (физической) подобласти, по которому определяются типы и коэффициенты решаемых в ней

дифференциальных уравнений, номера инцидентных сеточных граней, по которым определяются краевые условия и уравнения граничных поверхностей, а также номера узлов-вершин, по которым могут быть найдены их координаты. При этом подразумевается, что все сеточные элементы, грани, узлы и ребра имеют свои независимые упорядоченности и нумерации. Информация по каждой сеточной подобласти размещается в памяти соответствующего процессора. При этом данные по смежным граням дублируются, что позволяет в дальнейшем наиболее эффективно осуществлять распараллеливание. Общий объем ССД, очевидно, пропорционален суммарному количеству сеточных элементов в расчетной области.

Рассматриваемая технология позволяет полностью распараллелить без всяких коммуникационных потерь этап аппроксимации решаемой краевой задачи. Построение локальных матриц жесткости в МКЭ определяется данными, хранящимися в своем процессоре. В методе конечных объемов около каждого сеточного узла P_k строится ячейка Дирихле – Вороного, интегрированием по которой формируются дискретные соотношения баланса. При этом каждый конечный элемент разбивается на подэлементы $E_m = \bigcup_k E_{m,k}$, $E_{m,k} = E_m \bigcap V_k$ и построение локальных матриц баланса также производится поэлементно.

Итоговая глобальная матрица СЛАУ в методах декомпозиции принимает блочный вид $A = \{A_{k,l}\}$, причем каждый диагональный блок $A_{k,k}$ отвечает за подзадачу, решаемую в k-й подобласти на соответствующем процессоре, а внедиагональные блоки представляют собой информационные связи между подобластями. Блочная структура матрицы A определяется топологией макросети из подобластей, которая, естественно отображается на архитектуру вычислительной сети. Для рассмотренных выше одно-, дву- и трехмерной конфигураций разбиений матрица будет блочно трех-, пяти- и семидиагональной, т. е. каждый процессор (и подобласть) имеет связи только с двумя, четырьмя или шестью соседями.

Если решается система дифференциальных уравнений, то локальные матрицы формируются независимо и, следовательно, параллельно. Однако формирование глобальной матрицы, ее блочной структуры и последующей вычислительной схемы зависит от стратегии упорядочения неизвестных. Рассмотрим, например, комплексное векторное уравнение Гельмгольца

$$\Delta \mathbf{E} + \kappa \mathbf{E} = \mathbf{f}, \tag{16}$$

описывающее распределение амплитуды гармонического по времени электромагнитного поля, компоненты которого имеют вид

$$E_x = E_x^{\mathrm{Re}} + iE_x^{\mathrm{Im}}, \quad E_y = E_y^{\mathrm{Re}} + iE_y^{\mathrm{Im}}, \quad E_z = E_z^{\mathrm{Re}} + iE_z^{\mathrm{Im}},$$

где i – мнимая единица, а Re и Im обозначают вещественные и мнимые части. Таким образом, уравнение (16) представляет собой систему относительно шести скалярных вещественных функций.

Рассмотрим простой случай, когда каждая из функций определена в узлах сетки. Тогда естественной является последовательная нумерация этих шести компонент подряд для k-го узла, в результате чего полный вектор неиз-

вестных размера $6N$ (N – количество расчетных узлов сетки) определяется в виде

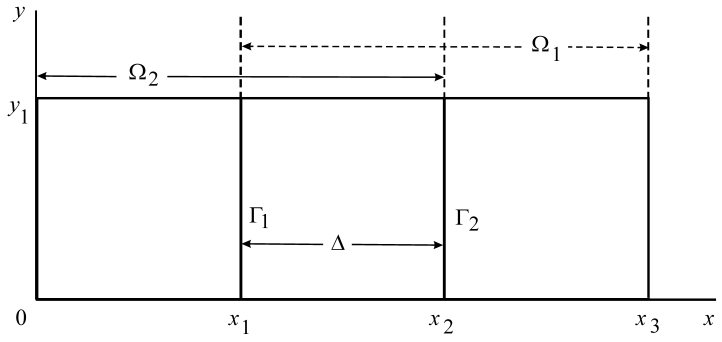
$$u = (\ldots, E^{\mathrm{Re}}_{x,k}, E^{\mathrm{Im}}_{x,k}, E^{\mathrm{Re}}_{y,k}, E^{\mathrm{Im}}_{y,k}, E^{\mathrm{Re}}_{z,k}, E^{\mathrm{Im}}_{z,k}, \ldots)^t.$$

Если при этом применить рассмотренные выше методы декомпозиции с теми же разбиениями на подобласти, то в каждой из них алгебраическая размерность подзадачи (и объем вычислений) увеличится в 6 раз, что положительно скажется на эффективности распараллеливания, так как коэффициент E_p в (1) возрастет вследствие относительного уменьшения коммуникационных потерь. Однако векторный характер уравнения (16) позволяет реализовывать и другую тактику распараллеливания: каждую из шести скалярных неизвестных функций рассчитывать на своем процессоре и не использовать принцип декомпозиции области

Аналогичным образом можно поступать при распараллеливании и других векторных систем, например уравнений термоупругости или уравнений Навье – Стокса, описывающих вязкое течение жидкости: различные сеточные функции группируются в блоки по их локальному расположению, а декомпозиция осуществляется общая по подобластям, или, наоборот, АСД для каждой из искомых функций (т. е. фактически глобальная матрица для соответствующей СЛАУ) формируется на отдельном процессоре для автономного решения.

Если коэффициенты исходной задачи зависят от времени и/или от самого решения, то естественным образом вычислительные аппроксиматоры пересчитывают АСД на каждом временном шаге и/или на каждой итерации по нелинейности. Данные факторы, конечно, увеличивают вычислительную сложность расчета, но, как правило, только повышают эффективность распараллеливания.

Особенности распараллеливания СЛАУ – «главного» ресурсоемкого расчетного этапа – это на алгебраическом языке есть реализация многоуровневого итерационного процесса с крупноблочными матрицами. Два основных уровня (нижний и верхний) – это итерационный алгоритм(ы) для решения «внутренних» задач в процессорах и итерации по подобластям с организацией межпроцессорных обменов. Вообще говоря, многоуровневость не есть универсальное средство ускорения решения задачи, хотя строгие теоремы об оптимальности алгоритмов (в однопроцессорном варианте) пока отсутствуют. Для понимания проблемы достаточно рассмотреть простой пример, приведенный на рисунке. Здесь расчетная прямоугольная область Ω представлена в виде двух пересекающихся подобластей $\Omega_1 = [x_1, x_3] \times [0, y_1]$ и $\Omega_2 = [0, x_2] \times [0, y_1]$, причем размер их общей части $\Delta = \Omega_1 \cap \Omega_2$ можно считать параметром декомпозиции. Классическая схема итерационных приближений следующая: на границах Γ_1 и Γ_2 выбираются произвольные начальные значения u_1^0, u_2^0 искомого решения, использующиеся как граничные условия Дирихле для подзадач в Ω_1 и Ω_2, которые реализуются на своих процессорах параллельно. Затем от полученных решений берутся следы на Γ_1 и Γ_2, принимаемые за новые приближения u_1^1, u_1^2 в итерационном процессе, который и заключается в последовательном вычислении новых значений u_1^n, u_2^n путем многократного решения однотипных задач в Ω_1 и Ω_2. Для них, в свою очередь, могут применяться как итерационные алгоритмы, так и пря-



Алгоритм декомпозиции областей

мые (последние будут предпочтительнее, если после дискретизации содержат не очень много узлов сетки).

На данном примере можно проследить возможные варианты применяемых методов и технологий вместе с их влиянием на эффективность распараллеливания. Общий объем вычислений рассмотренного алгоритма декомпозиции оценивается величиной

$$Q = c(N_1 + N_2)\sum_{k=1}^{n_l} n_{k,l}^{\text{in}}, \tag{17}$$

где $n_{k,l}^{\text{in}} \approx c_1 N_l^{\alpha}$ $(c_1 > 0,\ l = 1,2)$ – количество «внутренних» итераций в Ω_1 и Ω_2 (считаем их для простоты одинаковыми) на каждой k-й «внешней» итерации; N_1 и N_2 – количество узлов в Ω_1 и Ω_2 соответственно; c – число арифметических действий на одной внутренней итерации, а $\alpha > 0$ – некоторый показатель качества метода решения подзадач в Ω_1, Ω_2.

Рассмотрим различные факторы отдельно. Если размер общей подобласти Δ увеличивать, то количество внешних итераций n_l будет уменьшаться, но при этом будет расти трудоемкость каждой внутренней итерации, как и их количество $n_{k,l}^{\text{in}}$.

В частности, перспективным является разбиение на непересекающиеся подобласти Ω_1 и Ω_2, имеющие общую внутреннюю границу $\Gamma_{1,2}$, т. е. $\Delta = 0$. В этом случае для ускорения внешних итераций на $\Gamma_{1,2}$ следует чередовать различные типы граничных условий, например граничные условия Дирихле и Неймана. Это варьирование краевыми условиями является возможным средством оптимизации и для декомпозиции с пересечением подобластей.

Далее, при решении вспомогательных подзадач на первых внешних итерациях не требуется высокой точности и для экономии времени расчета надо уметь управлять числом внутренних итераций $n_{k,l}^{\text{in}}$, которое можно регулировать последовательностью допустимых погрешностей $\varepsilon_{k,l}^{\text{in}}$ для внутренних итерационных процессов. Наконец внешние итерации можно ускорять с помощью обобщенных методов сопряженных направлений. Перечисленные математические аспекты декомпозиции областей – это только одна сторона вопроса, который при отображении на компьютерную архитектуру дополняется многочисленными проблемами: сбалансированным формированием

размерностей подзадач во избежание простоев отдельных процессоров, совмещением вычислений с межпроцессорными обменами, разнообразной оптимизацией программ для каждого вычислителя и т. д.

Распараллеливание этапа постпроцессирования и визуализации результатов моделирования также имеет важное значение, поскольку при наличии больших объемов графической информации с анимацией и мультимедиа в режиме *on-line* именно обработка этих данных может составлять «львиную» долю общего времени вычислительного эксперимента. Здесь выделяются естественным образом две технологические части проблемы. Первая – это непосредственное вычисление координат точек различных геометрических интерпретаций (изоповерхностей, силовых линий, графиков и т. д.) рассчитанных полей, представляющих собой вещественные массивы значений функций, которые соотнесены с сеточными объектами, отображенными в ССД [20]. Поскольку полученные численные решения разнесены своими фрагментами по подобластям, то их постпроцессирование целесообразно осуществлять также одновременно на соответствующих процессорах.

Результаты данной стадии составляют распределенную ГрафСД, которая далее должна трансформироваться во внутренний формат какой-то системы, выбранной из штатных графических систем, обширная информация по которым имеется в сети Интернет. Непосредственная визуализация изображений представляет собой вторую часть технологической проблемы и может осуществляться декомпозицией по подобластям со сборкой результатов на головном мониторе.

Заключение. Главные выводы выполненного анализа заключаются в следующем: рассмотренные вопросы имеют комплексный характер, а создание эффективного программного обеспечения для высокопроизводительного вычислительного эксперимента требует умения выбрать современные алгоритмы, применения передовых информационных технологий и имеющихся инструментариев, а также глубокого знания архитектуры и оптимизационных средств многопроцессорных вычислительных систем (см., например, [21] и библиографию в ней). Организация и реализация такого проекта – это совокупность научно-исследовательских работ, рассчитанных отнюдь не на талантливого одиночку, а требующих согласованных действий команды разнопрофильных специалистов.

СПИСОК ЛИТЕРАТУРЫ

1. **Dongarra J. J., Walker D. W.** The quest for petascale computing // Comput. in Sci. Eng. 2001. N 3. P. 32.
2. **McCurdy C. W., Simon H. D., Kramer W. C. et al.** Future directions in scientific supercomputing for computational physics // Comput. Phys. Commun. 2002. **147**, N 1. P. 34.
3. **Воеводин В. В.** Математические модели и методы в параллельных процессах. М.: Наука, 1986.
4. **Воеводин В. В., Воеводин В. В.** Параллельные вычисления. С.-Пб.: БХВ-Петербург, 2002.
5. **Касперски К.** Техника оптимизации программ. Эффективное использование памяти. С.-Пб.: БХВ-Петербург, 2003.
6. **Ильин В. П.** О численном решении прямых и обратных задач электромагнитной георазведки // СибЖВМ. 2003. **6**, № 4. С. 381.

7. **Василевский Ю. В., Ильин В. П., Тыртышников Е. Е.** Вычислительные технологии // Современные проблемы вычислительной математики и математического моделирования. М.: Наука, 2005. Том 1. С. 100.
8. **Ильин В. П.** Геометрическое и функциональное моделирование в задачах математической физики // Вычисл. технологии. 2001. **6**, ч. 2. С. 315.
9. **Il'in V. P.** Geometric problems and algorithms in mathematical modeling // Proc. of the 15th Intern. Conf. of Computer Graphics and Applications (Graphicon'2005). Novosibirsk: ICM&MG, SB RAS, 2005. P. 289.
10. **Liseikin V. D.** Grid Generation Methods. Berlin: Springer-Verlag, 1999.
11. **Il'in V. P., Pavlov M. V., Volkov A. P.** On the finite volume approach to the 3-D quasi-structured grids // Bull. of the Novosibirsk Computing Center. Ser. Numer. Anal. 2005. N 13. P. 21.
12. **Axelsson O., Barker V. A.** Finite Element Solution of Boundary Value Problems: Theory and Computations. N. Y.: Academ Press, 1984.
13. **Ильин В. П.** Методы конечных разностей и конечных объемов для эллиптических уравнений. Новосибирск: Изд-во ИВМиМГ СО РАН, 2001.
14. **Джордж А., Лю Дж.** Численное решение больших разреженных систем уравнений. М.: Мир, 1984.
15. **Писанецки С.** Технология разреженных матриц. М.: Мир, 1988.
16. **Ортега Дж.** Введение в параллельные и векторные методы решения линейных систем. М.: Мир, 1990.
17. **Axelsson O.** Iterative Solution Methods. Cambridge: Univer. Press, 1994.
18. **Ильин В. П.** Методы неполной факторизации для решения алгебраических систем. М.: Наука, 1995.
19. **Saad Y.** Iterative Methods for Sparse Linear Systems. N. Y.: PWS Publishing, 1996.
20. **Il'in V. P., Solovev S. A.** U3D based computer graphics in PDE PTK // Proc. of the 16th Intern. Conf. of Computer Graphics and Applications (Graphicon'2006). Novosibirsk: NCC Publ., 2006. P. 222.
21. **Тарнавский Г. А., Вшивков В. А., Тарнавский А. Г.** Параллелизация алгоритмов и кодов вычислительной системы «Поток-3» // Программирование. 2003. № 1. С. 24.

Поступила в редакцию 3 ноября 2006 г.