МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С. П. КОРОЛЕВА

*Д. Л. Головашкин, С. П. Головашкина*

# МЕТОДЫ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

(Часть 2)

*Учебное пособие*

САМАРА 2003

**УДК 681.324.006.3**

***Головашкин Д.Л., Головашкина С.П.*** **Методы параллельных вычислений (Часть 2):** Учеб. пособие. Самар. гос. аэрокосм. ун-т. Самара, 2003. 103 с.

**ISBN 5-7883-0247-1**

Излагаются методики построения параллельных алгоритмов, вычисления по которым могут быть реализованы на системах с мультипроцессорной архитектурой. Рассматриваются особенности, связанные с декомпозицией исходного численного метода, организацией коммуникаций и объединением задач параллельного алгоритма. Приведены 17 параллельных алгоритмов для задач линейной алгебры и теории разностных схем. На многочисленных примерах продемонстрированы приемы синтеза и анализа в теории параллельных вычислений.

Предназначено для студентов, обучающихся по специальности 010200 - прикладная математика, а именно специализации 010203 - математическое обеспечение компьютерных телекоммуникационных систем. Может быть полезно и для студентов других специализаций данной специальности. Разработано на кафедре "Техническая кибернетика".

Табл. 5. Ил. 65. Библиогр.: 12 назв.

Работа выполнена при поддержке Министерства образования РФ, Администрации Самарской области и Американского фонда гражданских исследований и развития (CRDF Project SA-014-02) в рамках Российско-американской программы "Фундаментальные исследования и высшее образование" (BRHE) и РФФИ (грант № 01-01-00097).

Печатается по решению редакционно-издательского совета Самарского государственного аэрокосмического университета имени академика С.П. Королева.

Рецензенты: канд. физ.-мат. наук С.И. Харитонов;
канд. техн. наук К.Е. Климентьев.

**ISBN 5-7883-0247-1**

© Д.Л. Головашкин, С.П. Головашкина, 2003
© Самарский государственный аэрокосмический университет, 2003

## СОДЕРЖАНИЕ

Предисловие . . . . . 4
3. Параллельные алгоритмы . . . . . 6
3.1. Методы синтеза параллельных алгоритмов . . . . . 6
3.2. Характеристики параллельных вычислительных процессов и алгоритмов . . . . . 35
3.3. Параллельные алгоритмы линейной алгебры . . . . 43
3.4. Параллельные алгоритмы решения разностных уравнений . . . . . 67
Заключение . . . . . 98
Список литературы . . . . . 99
Приложение. Параллельные алгоритмы генерации псевдослучайных чисел . . . . . 100

## ПРЕДИСЛОВИЕ

Настоящая работа является продолжением учебного пособия “Методы параллельных вычислений (Часть 1)”. Предисловие и введение к первой части работы можно отнести и к данной работе. Все, что касается особенностей изучаемой дисциплины и ее места в учебном процессе, верно применительно к следующему тексту. Как первое пособие, так и второе, предназначены для студентов одной специализации, изучающих предмет “Методы параллельных вычислений”. Совокупный материал обоих пособий и составляет содержание курса в полном соответствии с учебной программой по нему.

Касательно материала второй части следует отметить, что он имеет более специализированный характер: довольно подробно рассматриваются алгоритмы матричного перемножения и решения систем линейных уравнений (СЛАУ). Представленные читателю методы широко применяются на практике для численного решения различных задач линейной алгебры, дифференциальных и интегральных уравнений. Выбор авторов обусловлен не только популярностью алгоритмов, но и их методической ценностью. Предлагая семнадцать параллельных алгоритмов, от простейших до довольно сложных, авторы задавались целью привить читателю навыки синтеза и анализа параллельных алгоритмов. Хотя методам построения и исследования алгоритмов посвящены первые два параграфа пособия, авторы сочли необходимым проиллюстрировать этот материал в следующих параграфах многочисленными примерами разной степени сложности. Так, знакомясь с методами решения сеточных уравнений неявных разностных схем, читатель сначала исследует возможность распараллеливания хорошо известного ему метода правой прогонки, затем встречается с методом встречных прогонок и, наконец, узнает метод циклической редукции, наиболее часто употребляемый для решения таких задач.

Особенности вычислительной системы обычно серьезно влияют на характеристики параллельного вычислительного процесса, ускорение которого может оказаться существенно ниже или выше (редко) ожидаемого. Освещение практической реализации приведенных алгоритмов не входило в замысел авторов, но в пособии присутствует пример для метода встречных прогонок, дающий представление об эффектах, связанных со структурой оперативной памяти ЭВМ.

# 3. ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ

## 3.1. Методы синтеза параллельных алгоритмов

Под алгоритмом будем понимать точное предписание, определяющее вычислительный процесс, идущий от варьируемых начальных данных к исходному результату (Гост 19.781-74 - определение алгоритма). Под исходным подразумевается окончательный результат. Соответственно параллельный алгоритм определяет параллельный вычислительный процесс, о котором говорилось в парагр. 1.5.

### Свойства алгоритма

1. *Однозначность.* Алгоритм не должен допускать двух разных трактований.

2. *Результативность.* Для нахождения результата (или убежденности в его отсутствии) требуется исполнение конечного числа шагов алгоритма.

3. *Массовость.* Задачи, отличающиеся лишь начальными данными, могут решаться с помощью одного алгоритма.

4. *Дискретность.* Алгоритм должен допускать разбиение на части (шаги алгоритма).

Принято четыре основных формы представления алгоритмов:

1. *Графическая форма.* Последовательные алгоритмы представляются, как правило, в виде блок-схем. При графической записи параллельных алгоритмов также используют символы, определенные в модели канал/задача (парагр. 1.2).

2. *Словесное описание.* Если по словесному описанию можно заключить, что алгоритм удовлетворяет четырем перечисленным свойствам и подходит под определение, то оно может считаться формой представления алгоритма.

3. *Запись в виде пошаговых инструкций.* По свойству дискретности любой алгоритм должен допускать разбиение на шаги. Действия, производимые при исполнении любого шага, описываются раздельно от действий других шагов.

4. *Представление в виде текста программы на алгоритмическом языке.* Однако не всякая программа, написанная на таком



языке, будет алгоритмом. Например, программа, допускающая "зацикливание" вычислительного процесса, не удовлетворяет свойству результативности.

Распространенной ошибкой является непонимание отличия алгоритма от вычислительного процесса, им порождаемого. Алгоритм, даже оформленный в виде программы - это идея (в смысле Платона [7]), а вычислительный процесс - действия, происходящие в вычислительной системе (вполне материальной). Таким образом, алгоритм и вычислительный процесс располагаются в разных мирах (реальном и воображаемом), что обуславливает различие в подходах к их изучению. Сначала появляется идея (алгоритм), потом идея порождает действия (вычислительный процесс). Если алгоритм принято синтезировать, а затем анализировать, то с вычислительным процессом поступают наоборот. Моделирование вычислительного процесса производят исключительно для его анализа, а анализ алгоритма служит для задач синтеза. Поэтому, изучая вычислительные процессы, мы уделяем основное внимание анализу, говоря об алгоритмах, обращаемся к синтезу.

Приступая к синтезу параллельного алгоритма по известному последовательному, полезно оценить принципиальную возможность эффективного распараллеливания. Например, производя вычисления по разностным уравнениям явной схемы $L_hU_h=f_h$ на трехмерной сеточной области $D_h$, разобьем область на две части $D_1$ и $D_2$, организовав обмен значениями сеточной функции $U_h$ на границе разбиения Г (рис. 3.1). В подобластях $D_1$ и $D_2$ вычисление сеточной функции $U_h$ можно производить одновременно (параллельно), учитывая значения $U_h$ на границе соседней подобласти.

Пусть для некоторой вычислительной системы время вычислений по области $D_h$ составит $T_D$. За $T_{D1}=T_{D2}=?T_D$ примем время вычислений в подобластях $D_1$ и $D_2$. Если бы затраты времени ограничивались длительностью арифметических операций, то данная задача в параллельном варианте решалась бы вдвое быстрей, чем в последовательном. К сожалению, к этим затратам следует добавить время на передачу и прием значений функции в граничных узлах сетки. Тогда $T_{D1}=T_{D2}=?T_D+T_{перес}$. При $T_{перес}>?T_D$, что весьма возможно, $T_{D1}>?T_D$ и $T_{D2}>?T_D$, тогда время вычислений по параллельному

алгоритму превзойдет время вычислений по последовательному. В синтезе подобных алгоритмов нет необходимости.

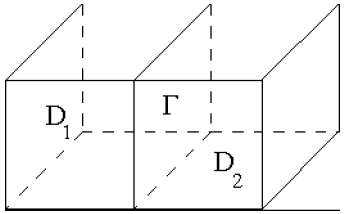



Рис. 3.1. Разбиение сеточной области $D_h$ на две подобласти $D_1$ и $D_2$, разделенные границей Г

Пока существует возможность производства вычислений с приемлемыми затратами времени по последовательному алгоритму, в параллельном алгоритме также нет необходимости.

### *3.1.1. Восходящий подход к синтезу параллельных алгоритмов*

Различают два подхода к синтезу параллельных алгоритмов. В восходящем подходе [12] объект исследования (последовательный алгоритм или численный метод) сразу дробится на мелкие части, которые в процессе построения параллельного алгоритма объединяются во все более крупные до тех пор, пока характеристики алгоритма не удовлетворят исследователя. Нисходящий подход [2] подразумевает разбиение исходной задачи на крупные блоки, которые в дальнейшем дробятся на более мелкие.

Этапы синтеза параллельного алгоритма восходящим методом представлены на рис. 3.2.

Остановимся подробно на трех из них.

#### *3.1.1.1. Разбиение*

Цель разбиения - формирование как можно большего числа задач параллельного алгоритма. Чем больше задач появится при разбиении, тем меньше ограничений стеснит фантазию разработчика на этапе сборки. Вместе с тем не стоит обращаться к крайностям, приписывая



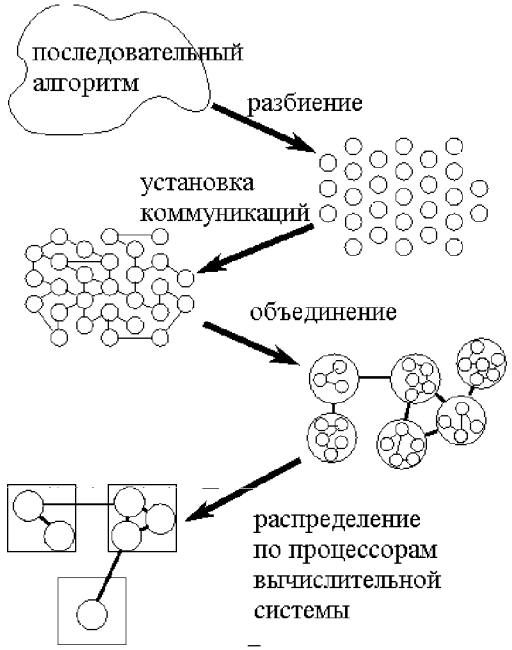



Рис. 3.2. Этапы синтеза параллельного алгоритма восходящим методом

каждой задаче, формируемой на данном этапе, одну арифметическую операцию (дальше разбивать некуда). Следует выделять в одну задачу наименьшую, но характерную именно для данного последовательного алгоритма часть. Если дальнейшее дробление этой части приведет к тому, что она станет присущей более широкому классу алгоритмов, то такое дробление производить не следует.

В качестве примера рассмотрим решение сеточных уравнений

$$\frac{U_i^{k+1} - U_i^k}{h_t} = \frac{U_{i-1}^k - 2U_i^k + U_{i+1}^k}{h_x^2}$$

простейшей явной разностной схемы для однородного одномерного линейного нестационарного уравнения теплопроводности на сетке $D_h=\{(t_k,x_i),\ t_k=kh_t,\ k=0,\ 1,\ \ldots,\ K=T/h_t\ ,\ x_i=ih_x,\ i=0,\ 1,\ \ldots,\ I=L/h_x\}$. Произведя максимально мелкое разбиение $D_h$ по направлению x, получим I задач (рис. 3.3).

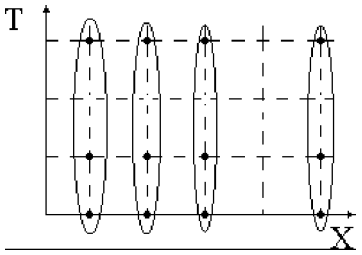



Рис. 3.3. Разбиение на задачи сеточной области $D_h$.
Обведены узлы сетки, принадлежащие одной задаче

Каждая задача решает одно разностное уравнение при фиксированном k. Увеличивая количество задач на одном временном слое, разобьем вычисления по одному разностному уравнению на несколько частей. Получим набор арифметических операций, по которым уже нельзя понять, что за алгоритм мы распараллеливаем.

Другое требование к образованным в результате разбиения задачам параллельного алгоритма - возможность их одновременного исполнения. В предыдущем примере нельзя увеличить число задач за счет разбиения по оси T, так как две задачи, характеризующиеся одним индексом i, не смогут исполняться одновременно.

На рис. 3.1 и 3.3 разбивалась область данных, с которыми работает последовательный алгоритм. В зависимости от этого разбиения распределялись вычисления между задачами параллельного алгоритма. Такой подход получил название декомпозиции данных. При функциональной декомпозиции сначала разбиваются вычисления, потом между задачами распределяются данные, с этими вычислениями связанные. Различия двух подходов к разбиению последовательного алгоритма на задачи иллюстрируются следующим примером.

Пусть на сеточной области $D_h=\{(t_k,x_i,y_j),\ t_k=kh_t,\ k=0,\ 1,\ \ldots,\ K=T/h_t,\ x_i=ih_x,\ i=0,\ 1,\ \ldots,\ I=L_x/h_x,\ y_j=jh_y,\ j=0,\ 1,\ \ldots,\ J=L_y/h_y\}$ записана система явных разностных уравнений

$$\begin{aligned} L_h^1 U_h &= f_h^1(V_n), \\ L_h^2 V_h &= f_h^2(U_n). \end{aligned} \qquad (3.1)$$



Для простоты ограничимся разбиением на две задачи параллельного алгоритма. Производя декомпозицию данных, разделим $D_h$ на $D_1=\{(t_k,x_i,y_j),\ t_k=kh_t,\ k=0,\ 1,\ \ldots,\ K=T/h_t,\ x_i=ih_x,\ i=0,\ 1,\ \ldots,\ I_1=L_x/(2h_x),\ y_j=jh_y,\ j=0,\ 1,\ \ldots,\ J=L_y/h_y\}$ и $D_2=\{(t_k,x_i,y_j),\ t_k=kh_t,\ k=0,\ 1,\ \ldots,\ K=T/h_t,\ x_i=ih_x,\ i=I_1+1,\ I_1+2,\ \ldots,\ L_x/h_x,\ y_j=jh_y,\ j=0,\ 1,\ \ldots,\ J=L_y/h_y\}$. Тогда первая и вторая задачи решают оба уравнения системы (3.1) для разных подобластей, обмениваясь значениями сеточных функций $U_h^\Gamma$ и $V_h^\Gamma$ на границе (рис. 3.4, а).

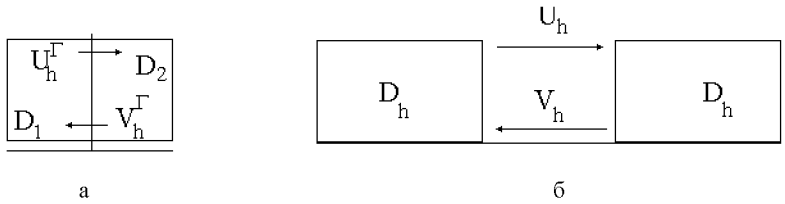



Рис. 3.4. Разбиение сеточной области и организация коммуникаций между задачами: а - при декомпозиции данных, б - при функциональной декомпозиции

При функциональной декомпозиции в первую очередь распределяются уравнения системы. Вычисления по первому уравнению включим в первую задачу, по второму - во вторую. На новом временном слое первая задача вычислит значение $U_h$, вторая задача значение $V_h$ на всей области $D_h$. При этом каждой задаче придется хранить обе сеточные функции на всей области, обмениваясь вычисленными значениями (рис. 3.4, б). Иначе невозможно продолжить вычисления по схеме (3.1), правая часть первого уравнения которой зависит от $V_h$ (следовательно, первая задача вынуждена хранить значения этой функции на $D_h$), второго уравнения от $U_h$ (вторая задача хранит $U_h$ на $D_h$).

Произведя такое же количество арифметических операций, одна задача алгоритма, полученного посредством функциональной декомпозиции, занимает вдвое больше памяти и передает/получает в I раз больше данных, чем одна задача алгоритма, записанного с помощью декомпозиции данных. Очевидно, декомпозиция данных для этого примера предпочтительней функциональной декомпозиции.

При функциональной декомпозиции наилучшим образом разбиваются вычисления (так как это разбиение происходит в первую очередь), распределение данных происходит менее удачно (оно целиком зависит от разбиения вычислений и носит подчиненный характер). Декомпозиция данных, наоборот, предусматривает оптимальное распределение данных между задачами и менее удачное распределение вычислений. Между ветвями вычислительного процесса, порожденного параллельным алгоритмом, происходит обмен данных, а не вычислений (при необходимости вычисления дублируются, а не пересылаются), поэтому оптимально распределить данные между задачами важнее, чем красиво разбить уравнения.

Зададимся целью разбить исследуемый последовательный алгоритм на максимально большое число задач L, положив для простоты J=1. Декомпозиция данных позволяет синтезировать I - задачный параллельный алгоритм (рис. 3.3). Осуществляя функциональную декомпозицию, при L>2 вступим в противоречие с требованием разбиения на характерные для данного алгоритма части, ибо распределять между задачами придется члены разностных уравнений (рис. 3.5).

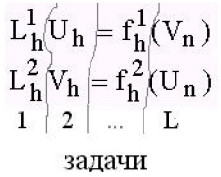



Рис. 3.5. Функциональная декомпозиция системы разностных уравнений (3.1) для произвольного числа задач параллельного алгоритма

При измельчении сетки $D_h$ (для повышения точности разностного решения) использование декомпозиции данных позволит увеличить количество задач параллельного алгоритма, а применение функциональной декомпозиции ограничит исследователя фиксированным числом задач при любой сеточной области.

Определим свойство масштабируемости параллельного алгоритма как возможность увеличивать число задач с возрастанием объема данных. Немасштабируемый алгоритм подобен последовательному, длительность вычислений по нему возрастает с увеличением области данных. Масштабируемый алгоритм позволяет компенсировать изменение объема данных таким изменением количества задач, при котором длительность вычислений не увеличивается. Декомпозиция данных, в отличие от функциональной декомпозиции, характеризуется масштабируемостью синтезируемых алгоритмов.



Несмотря на изложенные преимущества, не следует отдавать безусловного предпочтения декомпозиции данных. После применения функциональной декомпозиции нет необходимости в доказательстве сходимости и устойчивости используемого численного метода - он остался прежним. Проводя декомпозицию данных, приходится приспосабливать вычисления к разбиению данных, зачастую модифицируя численный метод, который уже может не обладать сходимостью и устойчивостью. Разработке и обоснованию новых вычислительных методов иногда удобнее предпочесть синтез алгоритма, основанного на функциональной декомпозиции (соответствующий пример приведен в парагр. 3.4).

Кратко повторим основные требования к этапу разбиения:

1) последовательный алгоритм следует разбивать на максимально большое число задач, каждая задача параллельного алгоритма содержит минимальный, но характерный участок последовательного;

2) задачи должны быть сбалансированными - содержать одинаковый объем вычислений и данных;

3) предпочтение отдается масштабируемым параллельным алгоритмам.

### *3.1.1.2. Установка коммуникаций*

Будем говорить, что задача $l_1$ данного параллельного алгоритма зависит от задачи $l_2$ того же алгоритма, если найдется хотя бы одно действие $d_1$ параллельной ветви, порожденной задачей $l_1$, которое в любом вычислительном процессе, определенном данным параллельным алгоритмом, всегда будет следовать за некоторым действием $d_2$ параллельной ветви, порожденной задачей $l_2$.

На этапе установки коммуникаций зависимые задачи связываются каналами, по которым передаются данные. Под коммуникациями

параллельного алгоритма будем понимать совокупность каналов и их топологию. Различают следующие типы коммуникаций.

*1. Локальные и глобальные коммуникации*

Если хотя бы одна задача параллельного алгоритма связана каналами со всеми остальными задачами, то уместно говорить об алгоритме с глобальными коммуникациями. В алгоритме с локальными коммуникациями любая задача связана с "небольшим" числом других задач. Эти задачи назовем соседями данной задачи, а "небольшим", для определенности, будем считать количество соседей, не превышающее размерность гиперкуба, вершины которого соответствуют задачам исследуемого алгоритма. Если число задач алгоритма не составляет степени двойки, используем для оценки локальности ближайшую большую степень двойки.

Построим параллельный алгоритм решения разностных уравнений

$$\frac{U_{i,j}^{k+1} - U_{i,j}^{k}}{h_t} = \frac{U_{i-1,j}^{k} - 2U_{i,j}^{k} + U_{i+1,j}^{k}}{h_x^2} + \frac{U_{i,j-1}^{k} - 2U_{i,j}^{k} + U_{i,j+1}^{k}}{h_y^2} \qquad (3.2)$$

явной схемы для однородного двумерного линейного нестационарного уравнения теплопроводности на сетке $D_h=\{(t_k,x_i,y_j), t_k=kh_t, k=0, 1, \ldots, K=T/h_t, x_i=ih_x, i=1, 2, 3, y_j=jh_y, j=1, 2, 3\}$. Произведя на этапе разбиения декомпозицию данных, поставим в соответствие каждому узлу сеточной области одну задачу алгоритма (рис. 3.6).

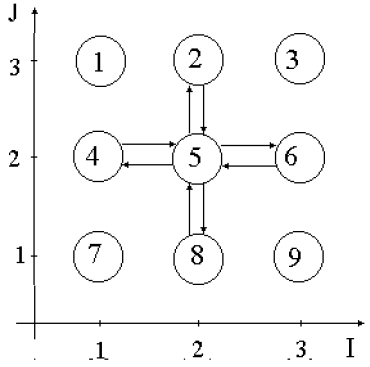



Рис. 3.6. Коммуникации 5-й задачи в параллельном алгоритме решения разностных уравнений (3.2)



В силу особенностей шаблона для (3.2) задачи с номерами 1 ,3, 7, 9 связываются каналами с двумя соседями, задачи 2, 4, 6, 8 - с тремя соседями, задача 5 - с четырьмя соседями (рис. 3.6). Например, задача 5 для решения уравнения (3.2) относительно $U_{2,2}^{k+1}$ нуждается в получении значений: $U_{2,3}^{k}$ от задачи 2, $U_{3,2}^{k}$ от задачи 6, $U_{2,1}^{k}$ от задачи 8 и $U_{1,2}^{k}$ от задачи 4. Перечисленные задачи, в свою очередь, принимают значение $U_{2,2}^{k}$ от 5-й задачи. Составленный алгоритм характеризуется локальными коммуникациями - максимальное количество соседей (4 у 5-й задачи) не превышает установленного предела.

Установим коммуникации параллельного алгоритма взятия определенного интеграла по восьми значениям подынтегральной функции. Восемь задач, хранящих значения известной сеточной функции в различных узлах, производят вычисления по квадратурной формуле прямоугольников, находя восемь интегральных сумм. Необходимо, связав каналами задачи параллельного алгоритма, сложить полученные числа.

Пусть задачи 2, 3, 4, 5, 6, 7, 8 передадут найденные интегральные суммы 1-й задаче, которая завершит вычисления по алгоритму (рис. 3.7). Коммуникации, организованные таким образом, глобальны. Рассмотрим присущие им недостатки.

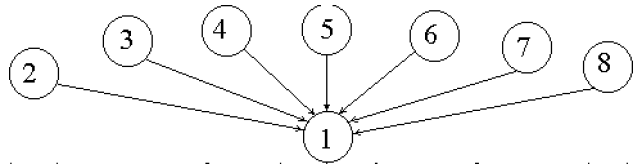



Рис. 3.7. Топология глобальных коммуникаций параллельного алгоритма взятия определенного интеграла по восьми значениям подынтегральной функции

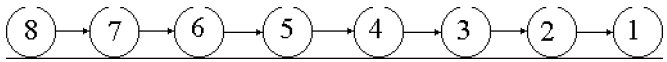
Первая задача, как и всякая другая задача параллельного алгоритма, порождает последовательный вычислительный процесс, в котором все действия, в том числе принятие по каналам интегральных сумм и их сложение, происходят последовательно. Если для расчета по квадратурной формуле прямоугольников требуется одна ариф-

метическая операция (ее одновременно произведут все задачи алгоритма), то для сложения интегральных сумм - семь. Таким образом, произведя одну арифметическую операцию, задачи 2, 3, 4, 5, 6, 7 в дальнейших вычислениях не участвуют, а задача 1 производит еще семь сложений. Для сокращения времени вычислений следует равномернее распределять операции между задачами алгоритма.

Более того, при реализации такого алгоритма на гиперкубе (наиболее распространенной кластерной конфигурации) две задачи из семи не смогут передать вычисленные значения интегральных сумм непосредственно первой задаче, между ними и первой задачей отсутствует прямая связь по сети, связывающей ЭВМ вычислительной системы.

Алгоритмы с локальными коммуникациями свободны от перечисленных недостатков. Например, при сгущении сетки и увеличении количества задач на рис. 3.6 число соседей для любой задачи алгоритма не превысит четырех. Следовательно, каждая задача произведет не более четырех последовательных приемов/передач по сети гиперкуба и решит свое сеточное уравнение одновременно с остальными задачами алгоритма.

На рис. 3.8 предложена схема локальных коммуникаций алгоритма численного интегрирования, которая позволяет реализовывать вычисления на гиперкубе, причем арифметические операции распределены равномерно между задачами. Задача 7, получая интегральную сумму от 8-й задачи, производит суммирование и передает результат задаче 6. Задача 6 получает интегральную сумму от 7-й задачи, производит суммирование, передает результат задаче 5 и так далее, до задачи 1.




3.8. Топология локальных коммуникаций параллельного алгоритма взятия определенного интеграла по восьми значениям подынтегральной функции

К сожалению, при сложении интегральных сумм всегда простаивают, не производя вычислений, семь задач из восьми. Например, когда сложение производит задача 5, задачи 8, 7, 6 уже передали



результаты своих вычислений и в дальнейшей работе алгоритма не участвуют, а задачи 4,3,2,1 еще не получили необходимых данных для начала суммирования. Хотя нам удалось адаптировать алгоритм к сетевой конфигурации гиперкуба, суммирование все еще производится последовательно.

В таких случаях принято использовать правило “разделяй и властвуй”. Его суть заключается в переходе от линейной структуры коммуникаций (рис. 3.8) к коммуникациям на дереве (рис. 3.9).

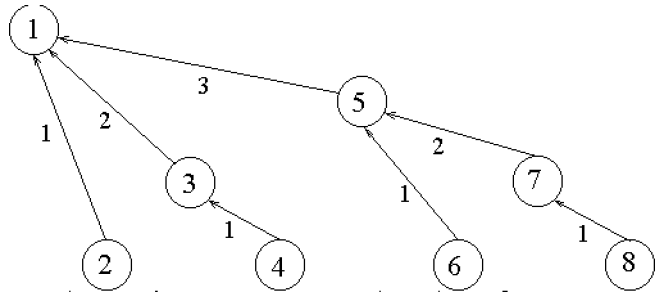



Рис. 3.9. Организация коммуникаций по правилу “разделяй и властвуй”. Под каналами цифрами обозначена очередность передачи данных

На первом шаге алгоритма все задачи вычисляют интегральные суммы. На втором шаге задачи 2, 4, 6, 8 передают задачам 1, 3, 5, 7 найденные значения (рис. 3.9), которые прибавляются к результатам, вычисленным задачами - получателями. Третий шаг алгоритма - передача данных от задач 3,7 к задачам 1,5 и аналогичное суммирование. На четвертом, последнем шаге алгоритма, задача 1, получая от задачи 5 необходимую информацию, находит значение определенного интеграла.

Максимальное количество соседей (три у 3-й задачи) не превышает размерность гиперкуба с восьмью вершинами - коммуникации предложенного алгоритма локальны. Более того, такая структура коммуникаций позволяет на втором шаге алгоритма произвести одновременно 4 сложения и 2 сложения на третьем шаге. Для аналогичного алгоритма, состоящего из L задач, сложение интегральных сумм потребует не L-1 последовательное применение операции суммирования, а $\log_2 L$ таких операций.

*2. Синхронные и асинхронные коммуникации*

Синхронные и асинхронные коммуникации различаются по аналогии с синхронным и асинхронным обменом сообщениями между параллельными ветвями вычислительного процесса (п. 1.5.5). К преимуществу асинхронных коммуникаций следует отнести возможность продолжения вычислений, если ожидаемое по каналу сообщение еще не поступило. Это полезно при вычислениях по несбалансированному алгоритму, синхронные коммуникации в этом случае увеличат время расчета. Использование асинхронных коммуникаций в хорошо сбалансированном алгоритме излишне.

*3. Статические и динамические коммуникации*

Если на множестве задач параллельного алгоритма при неизвестных начальных данных можно установить все отношения зависимости, то такой алгоритм характеризуется статическими коммуникациями. Для него топология каналов не меняется при варьировании начальных данных. Если все отношения зависимости до получения начальных данных однозначно установить нельзя - говорим об алгоритме с динамическими коммуникациями. Топология каналов такого алгоритма может меняться при различных входных данных. Наличие в алгоритме динамических коммуникаций серьезно осложняет его анализ. Перед реализацией вычислений по такому алгоритму следует с осторожностью выбирать сетевую конфигурацию вычислительной системы. Оптимальная при одних начальных данных конфигурация может оказаться неудачной при других.

*4. Регулярные и нерегулярные коммуникации*

Предположим, что все задачи исследуемого параллельного алгоритма, обозначив точками, удалось разместить в пространстве $R^n$ таким образом, что задачи, зависящие от любой заданной, составляют правильную геометрическую фигуру (n угольник, тор, сферу и т.д.), дерево или другую осмысленную структуру. Если это утверждение верно и для всех задач, от которых зависит любая заданная задача, то коммуникации такого алгоритма можно назвать регулярными. Когда получившиеся выше топологии нельзя описать аналитическими выражениями, говорят о нерегулярных коммуникациях параллельного алгоритма. Для реализации вычислений по алгоритму с нерегулярными коммуникациями сложно подобрать вычислительную систему с простой сетевой конфигурацией (гиперкуб, тор и т.п.).



Кратко остановимся на способе передачи группы данных по одному каналу. К особенностям передачи сообщений по сети вычислительной системы относятся: необходимость создания почтовых ящиков (п.1.5.6) и пакетный способ передачи. Ко времени передачи данных следует прибавить время, затрачиваемое на организацию этой передачи: создание почтового ящика, управление его работой и удаление ящика. Пакетный способ передачи заключается в посылке сообщения не сразу целиком, а по частям, пакетами. Если объем одного пакета превышает объем передаваемых данных, то время коммуникации будет определяться размером пакета, а не данных. С учетом этого зависимость времени передачи сообщения от его объема будет не линейной функцией, а ступенчатой (рис. 3.10).

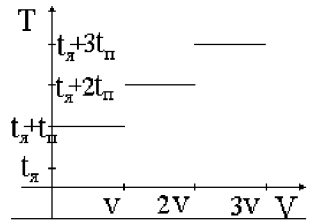



Рис. 3.10. Зависимость времени передачи сообщения T от его объема V: v-объем одного пакета, $t_я$ - время организации передачи (создание, работа и удаление почтового ящика), $t_п$ - время передачи одного пакета по сети

Представим ситуацию, когда по одному каналу необходимо передать два массива данных одинакового объема v/2. При раздельной передаче (каждый массив передается в виде отдельного сообщения) время пересылки составит $2t_я+2t_п$ ($t_я+t_п$ для каждого сообщения). Если поместить два массива в один, объемом v, и передать одним сообщением, то время пересылки сократится вдвое (при том что в обоих случаях пересылается одинаковый объем данных). Таким образом, следует отдавать предпочтение передаче

больших объемов данных в меньшем количестве сообщений над передачей меньших объемов в большем количестве.

Сформулируем основные рекомендации к этапу установки коммуникаций:

1) желательно избавляться от глобальных коммуникаций, переводя их в локальные по правилу “разделяй и властвуй”;

2) задачи хорошо сбалансированного алгоритма связываются одинаковым количеством каналов;

3) использование регулярных статических коммуникаций упрощает организацию вычислительного процесса (на большинстве вычислительных систем) по созданному алгоритму.

### *3.1.1.3. Объединение*

Вернемся к уравнениям (3.2), рассматривая их на сетке $D_h=\{(t_k,x_i,y_j), t_k=kh_t, k=0, 1, \ldots, K=T/h_t, x_i=ih_x, i=1, 2, .., 8; y_j=jh_y, j=1, 2, .., 8\}$. После этапов разбиения и установки коммуникаций получим 64 задачи параллельного алгоритма. Вычисляя $U_{i,j}^{k+1}$, отмеченная на рис. 3.11,а задача произведет 13 арифметических операций, на которые придется 4 приема значений $U_{i-1,j}^{k}$, $U_{i+1,j}^{k}$, $U_{i,j-1}^{k}$, $U_{i,j+1}^{k}$ и 4 передачи $U_{i,j}^{k}$ соседним задачам. Все значения передаются в разных пакетах, так как предназначены разным задачам и пойдут по разным каналам. На одну операцию посылки или приема пакета придется 1,625 арифметической операции. Предположим, что данный алгоритм реализован на вычислительной системе, процессоры которой производят 100 арифметических операций в секунду, а данные по сети передаются со скоростью 1 пакет в секунду. Тогда 1,6% от времени вычислений будет связано с производством арифметических операций, а 98,4% - с передачей данных по сети. Общее время вычислений на одном слое по k составит 8,13 с. Вычисления по последовательному алгоритму на данной вычислительной системе потребуют 8,32 с.

Объединим по 16 задач параллельного алгоритма в одну, как это показано на рис. 3.11,б, получив четырехзадачный алгоритм. Для него каждая задача производит 208 арифметических операций, связанных с пересылкой 4 пакетов (в каждом по 4 значения сеточной функции).



Для рассматриваемой вычислительной системы время работы по алгоритму составит 6,08 с. Объединение задач алгоритма позволило на четверть сократить время вычислений по нему.

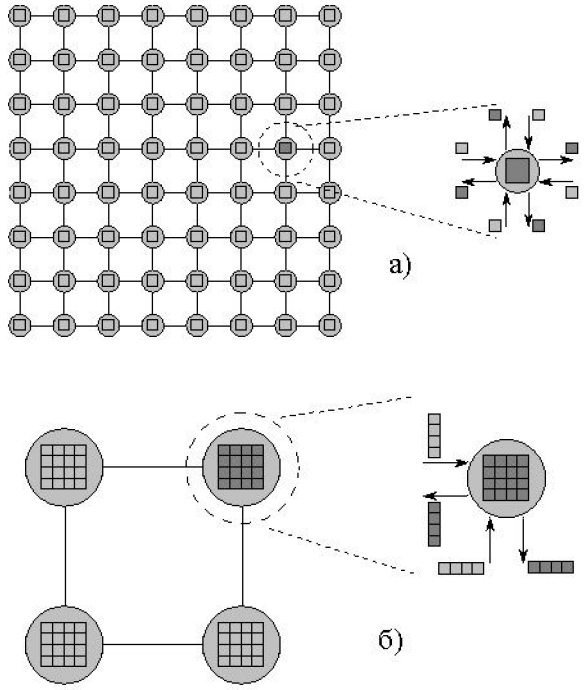



Рис. 3.11. Объединение задач при построении параллельного алгоритма решения разностных уравнений (3.2): а - алгоритм до объединения, б - после

Логично произвести объединение еще раз, до 2 задач параллельного алгоритма. Однако выигрыша времени по сравнению с четырехзадачным алгоритмом не получится, длительность расчетов составит 6,16 с. Уменьшение времени коммуникаций в два раза не позволит скомпенсировать двукратное увеличение времени арифметических операций.

Для произвольного параллельного алгоритма на этапе объединения, определяя оптимальное число задач L, необходимо искать экстремум функции

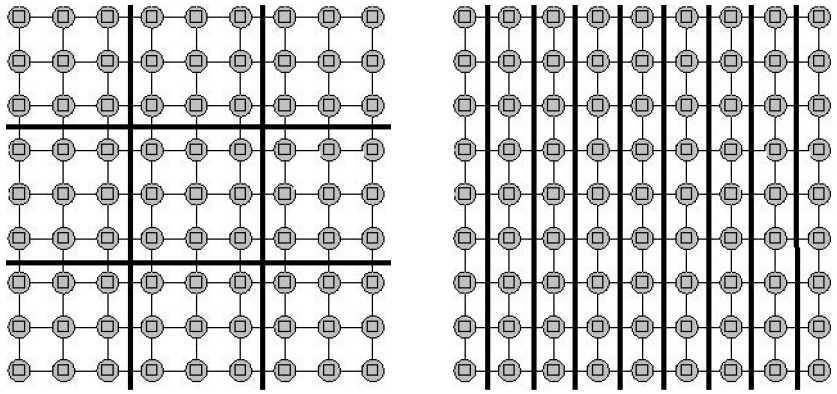
$$T(L)=T_a(L)+T_п(L), \qquad (3.3)$$

где T - время вычислений по алгоритму на данной вычислительной системе, $T_a$ и $T_п$ - длительность арифметических операций и передач по сети. В рассмотренном примере при L - степени двойки

$T_a(L)=\frac{13}{100}\frac{64}{L}$ с, $T_п(L)$=8 с для L>4, 4с для L=2 и 2с для L=2.

Учитывая, что отношение скорости производства арифметических операций к скорости передачи данных по сети для стандартных кластерных вычислительных систем колеблется незначительно, синтез алгоритма на этапе объединения не обязательно согласовывать с параметрами конкретной ЭВМ.

Желая сформулировать правила объединения, обратимся к алгоритмам для многомерных разностных уравнений. Существует n стратегий объединения задач, связанных с n-мерной сеточной областью $D_h$. На двумерной области можно объединять задачи в полосы вдоль одной из координатных осей (одномерное объединение), рис. 3.12,а, в блочные области (двумерное объединение), рис. 3.12, б.



а) б)

Рис. 3.12. Две стратегии объединения задач, связанных с двумерной сеточной областью: а - объединение в полосы вдоль координатной оси (одномерное объединение); б - объединение в блочные области (двумерное объединение)



Проведем сравнение получившихся полосного и блочного алгоритмов. Их задачи порождают вычислительные процессы с одинаковым количеством арифметических операций, но разной структурой коммуникаций. Исследуем коммуникации тех задач алгоритмов, что связаны каналами с максимальным числом соседей (время вычислений по алгоритму будет определяться такими задачами). Для полосного алгоритма это любая задача, кроме первой и последней, для блочного - задача, связанная с центральным блоком сеточной области. Характерная задача полосного алгоритма имеет двух соседей, блочного - четырех.

Для сеточной области $D_h=\{(t_k,x_i,y_j)$, $t_k=kh_t$, k=0, 1, …, $K=T/h_t$ , $x_i=ih_x$, i=0, 1, …, 8, $y_j=jh_y$, j=0, 1, …, 8$\}$ предпочтение следует отдать полосному алгоритму, вычисления по нему будут характеризоваться передачей и приемом 4 пакетов по сети, блочный алгоритм использует коммуникации с передачей 8 пакетов.

Рассмотрим квадратную сетку $D_h=\{(t_k,x_i,y_j)$, $t_k=kh_t$, k=0, 1, …, $K=T/h_t$ , $x_i=ih_x$, i=0, 1, …, I, $y_j=jh_y$, j=0, 1, …, I$\}$ такую, что для заданного I время передачи сообщения по сети будет характеризоваться в большей мере размером посылки, чем объемом пакета. Для достаточно больших I, когда объем передаваемой информации многократно превышает емкость пакета, зависимость T(V) на рис. 3.10 можно считать линейной (одно сообщение передается с помощью большого количества пакетов). Предположим, что на этапе объединения $I^2$ задач уже сведено в 81 задачу (рис. 3.12). Располагая вычислительной системой с 9 процессорами, произведем объединение еще раз, остановившись на одном из вариантов: одномерном или двумерном. При одномерном объединении характерная задача будет посылать I значений сеточной функции правому соседу, I значений соседу слева и столько же принимать от них, оперируя на своих коммуникациях с 4I числом. Задача блочного алгоритма, связанная с центральным блоком, посылает каждому их четырех соседей I/3 значений функции $U_h$, организуя посылку и прием 8I/3 чисел. Двумерное объединение в данном примере характеризуется пересылкой существенно меньшего объема данных (2/3 от полосного алгоритма) и является более приемлемым.

Когда речь идет о синтезе параллельного алгоритма, чаще всего подразумевается работа с большим объемом вычислений и данных

(иначе удовлетворяются известным алгоритмом, реализованным на персональной ЭВМ). Следовательно, производя объединение на области размерности n, отдают предпочтение n-мерному объединению над n-1-мерным, n-1-мерному над n-2-мерным так далее. Эта рекомендация по форме областей объединения в трехмерном случае получила название правила перехода от поверхности к объему. На рис. 3.13 стрелками указано направление перехода от объединения в область, протяженную вдоль одной координатной оси - прямую (одномерное объединение, наименее предпочтительное), через область, протяженную вдоль двух осей - поверхность (двумерное объединение), к области, протяженной вдоль трех осей - объему (трехмерное объединение, наиболее предпочтительное).

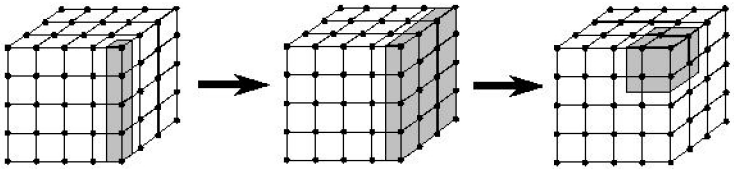



Рис. 3.13. Трехмерная иллюстрация правила перехода от поверхности к объему

Для прямоугольной сетки с разной дискретизацией правило перехода от поверхности к объему применяют с осторожностью. Например, если $D_h=\{(t_k,x_i,y_j)$, $t_k=kh_t$, k=0, 1, …, $K=T/h_t$, $x_i=ih_x$, i=0, 1, …, I, $y_j=jh_y$, j=0, 1, …, J}, где I>>J, его использование неуместно. Задача полосного алгоритма на рис. 3.14,а характеризуется пересылкой 12J значений сеточной функции, в то время как задача блочного алгоритма (рис. 3.14, б) пересылкой 4J/3+4I/3 значений. При I=9J блочный алгоритм проигрывает полосному в 3 раза по объему пересылок.

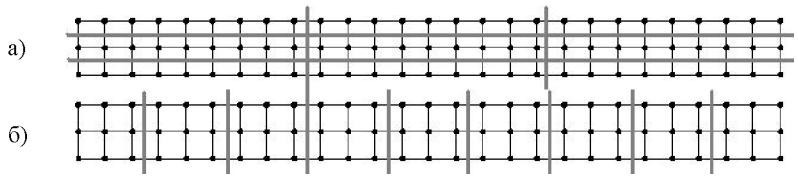



Рис. 3.14. Варианты объединения для прямоугольной сетки с разной дискретизацией: а - двумерное объединение, б - одномерное



Найдем такие параметры дискретизации I,J двумерной сеточной области и количество задач параллельного алгоритма L, при которых блочный алгоритм уступает полосному. Объединение задач в полосном алгоритме разумно проводить вдоль оси с меньшей дискретизацией (так минимизируется объем пересылок), пусть это будет ось Y с параметром дискретизации J. Тогда полосной алгоритм будет характеризоваться пересылкой 4J чисел. В блочном алгоритме организуется пересылка $\frac{4}{\sqrt{L}}$ J чисел по направлению Y и $\frac{4}{\sqrt{L}}$ I чисел по направлению X. При $4J<\frac{4}{\sqrt{L}}J+\frac{4}{\sqrt{L}}I$, то есть для $J<\frac{I}{\sqrt{L}-1}$, полосной алгоритм предпочтительнее блочного. В примере на рис.3.14 L=9 и одномерное объединение становится эффективнее двумерного при J<I/2.

На этапе объединения появляются одни задачи, исчезают другие, приходится заново устанавливать коммуникации параллельного алгоритма. При этом топология коммуникаций может претерпеть существенные изменения. Иногда, для сокращения времени вычислений по алгоритму, специально идут на увеличение объема вычислений и устанавливают новые каналы между задачами. Разберем этот прием на примере алгоритма взятия определенного интеграла.

Рассмотрим трехзадачный алгоритм численного интегрирования, каждая задача которого после последнего шага алгоритма должна содержать значение интеграла от функции на всей сеточной области. Коммуникации такого алгоритма состоят из 4 каналов. Пронумеруем каждый канал в соответствии с последовательностью передачи данных по нему (рис. 3.15).

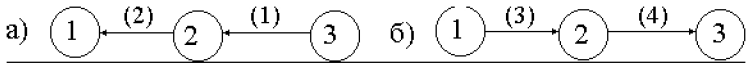



Рис. 3.15. Коммуникации трехзадачного алгоритма численного интегрирования

Будем рассматривать только те шаги алгоритма, которые следуют за расчетом интегральной суммы (предшествующие шаги не связаны с пересылкой данных и нам не интересны). Задача 3 посылает

найденное значение интегральной суммы $I_3$ задаче 2, которая, прибавляя его к своему значению $I_2$, отправляет сумму $I_2+I_3$ задаче 1. Та, найдя значение интеграла $I_s=(I_1+I_2+I_3)$ на всей сеточной области, посылает его задаче 2. В конце алгоритма 2 передает $I_s$ 3. Рассматриваемые шаги алгоритма характеризуются 8 пересылками (4 посылки и 4 приема) и 2 арифметическими действиями, причем как передачи, так и действия происходят последовательно. Время вычислений составит $T=2T_a+8T_п$.

Изменим структуру коммуникаций, связав задачи алгоритма тремя каналами, и увеличим количество арифметических операций в каждой задаче до двух (рис. 3.16).

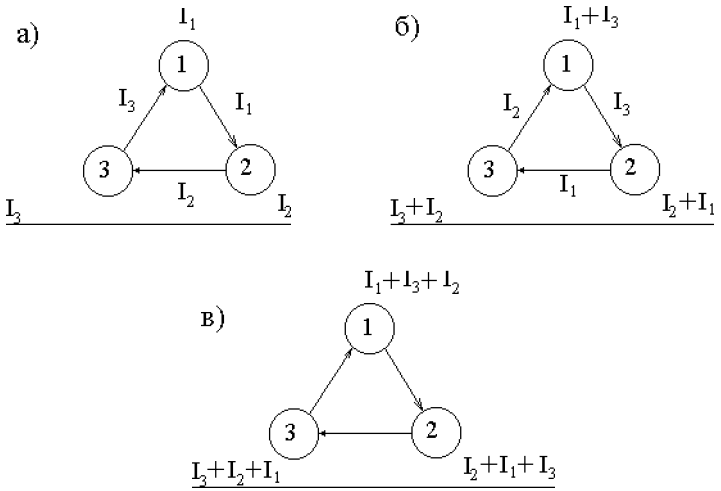



Рис. 3.16. Этапы модифицированного трехзадачного алгоритма численного интегрирования

Как и в предыдущем случае, ограничимся рассмотрением тех шагов алгоритма, что следуют за вычислением интегральных сумм. На шаге а) задача l, l=1,.., 3 посылает по выходящему из нее каналу значение $I_l$ и принимает сообщение по входящему каналу (рис. 3.16,а). Следующий шаг б) характеризуется суммированием в каждой задаче полученного по каналу и известного ранее значения. Посылается далее значение, полученное на шаге а), принимается новое (рис. 3.16,б).



На последнем шаге алгоритма каждая задача находит искомое значение $I_s$ определенного интеграла. За три шага произведено 12 передач (на 4 больше, чем в предыдущем случае) и 6 сложений (на 4 операции больше). Но при этом часть сложений и пересылок осуществлялась параллельно, таким образом, что время арифметических операций осталось тем же, а длительность пересылок данных сократилась вдвое. Учитывая, что 2 операции сложения производятся на несколько порядков быстрее, чем пересылка по сети вычислительной системы 4 пакетов, длительность вычислений, связанных с завершающими этапами алгоритма взятия определенного интеграла, удалось сократить вдвое.

Обратимся к более сложной топологии коммуникаций - дереву. Изучим возможность модификации аналогичного восьмизадачного алгоритма. Время вычислений по заключительным шагам известного алгоритма (см. рис. 3.9) составит $T=3T_a+12T_п$, производится 7 сложений и 28 пересылок (часть из них параллельно). Первая задача осуществит наибольшее количество сложений (три) и последовательно примет три переданных ей числа (3 приема, 3 посылки). На этапе рассылки значения $I_s$ до 8-й задачи ожидаемое число дойдет через 3 посылки и 3 приема (задача 1 посылает $I_s$ задаче 5, та пересылает задаче 7, от нее получает 8-я задача).

Сменим топологию дерева на гиперкуб (см. рис. 1.7,б; 3.17).

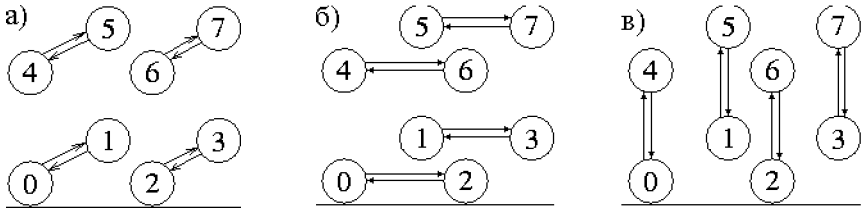



Рис. 3.17. Реализация на гиперкубе восьмизадачного алгоритма взятия определенного интеграла

На первом шаге модифицированного алгоритма каждая задача получает и передает сообщение от соседа по направлению Z (в правой декартовой системе координат, рис. 3.17, а). Далее используются каналы по направлениям X (рис. 3.17, б) и Y (рис. 3.17, в). С каждым

приемом связана одна операция сложения. Следовательно, алгоритм характеризуется 12 сложениями и 48 пересылками. По сравнению с исходным алгоритмом данный содержит почти вдвое больше арифметических операций и пересылок. Но, в отличие от сравниваемого алгоритма, в данном большая доля сложений и пересылок исполняется параллельно. Время вычислений по нему составит $T=3T_a+6T_п$, что вдвое меньше времени вычислений по алгоритму на дереве.

Учитывая изложенное, при осуществлении синтеза параллельного алгоритма на этапе объединения будем руководствоваться следующими рекомендациями.

1. Укрупняемые задачи должны оставаться сбалансированными.
2. Руководствоваться правилом перехода от поверхности к объему. Разумеется, только там, где его применение уместно. Понимать это правило как тенденцию, а не закон.
3. При изменении коммуникаций не ограничиваться старой топологией, изучать возможность установления каналов, использование которых позволит сократить время вычислений.
4. Применять прием тиражирования вычислений для уменьшения объема пересылаемых данных.

Коротко остановимся на этапе распределения задач параллельного алгоритма по процессорам вычислительной системы. Данный этап более связан с реализацией алгоритма на конкретной ЭМВ, чем с его синтезом, следовательно, относится к изучаемому студентами курсу “Параллельное программирование” и находится за рамками настоящего пособия. Все же считаем необходимым осветить характерные особенности этого этапа.

Ошибочно полагать, что при реализации любого алгоритма принято ставить в соответствие одному процессору вычислительной системы только одну задачу параллельного алгоритма. Рассмотрим случай, когда одному процессору можно поручать вычисления по нескольким задачам алгоритма.

Предположим существование алгоритма, задачам которого присущи простои. В некоторый момент времени часть задач такого алгоритма производит вычисления, другая часть ожидает передачи сообщений по каналам и вынужденно простаивает (без получения необходимых данных вычислений не производится). Например, в



алгоритме, коммуникации которого изображены на рис. 3.15, задача 3 участвует в вычислениях и пересылках только на первом и последнем шаге, простаивая во время исполнения остальных шагов алгоритма.

Если на один процессор вычислительной системы приходится две задачи, вычисления по которым одновременно не производятся (одна задача всегда простаивает), то общее время вычислений по алгоритму не возрастет.

При разработке параллельного алгоритма также следует иметь в виду, что современные многопроцессорные вычислительные системы позволяют одновременно на одном процессоре производить вычисления и осуществлять прием или передачу сообщений по сети. Эта особенность будет учтена нами в дальнейшем изложении (например при построении блочного алгоритма перемножения матриц).

В целом, касаясь восходящего метода синтеза параллельных алгоритмов, следует заметить, что последовательность прохождения этапов метода (от разбиения через установку коммуникаций и объединения к распределению по процессорам вычислительной системы) в практике построения алгоритмов выдерживается не всегда. Приходится возвращаться на предыдущие этапы, корректируя количество задач, их содержание и топологию коммуникаций. Синтез алгоритмов - процесс творческий, не следует ограничиваться строгим соблюдением последовательности этапов изложенного метода.

### *3.1.2. Нисходящий подход к синтезу параллельных алгоритмов*

Как уже отмечалось, нисходящий подход характеризуется разбиением последовательного алгоритма на как можно более крупные слабовзаимодействующие блоки (в литературе такой прием построения параллельных алгоритмов иногда называется блочно-иерархическим методом). Под блоками алгоритма понимают некоторую совокупность инструкций (как при функциональной декомпозиции) или данных (декомпозиция данных).

Иерархия, которую задают размеры блоков, при функциональной декомпозиции строится от арифметических операций (низший уровень иерархии) через все более сложные арифметические выражения до крупных блоков - сегментов алгоритма (высший уровень иерархии).

На рис. 3.18 представлена схема последовательного алгоритма с разбиением вычислений на блоки и с задаваемой этим разбиением иерархией. Стрелками показаны циклические конструкции, точками - арифметические выражения. Вычисления по схеме производятся слева направо.

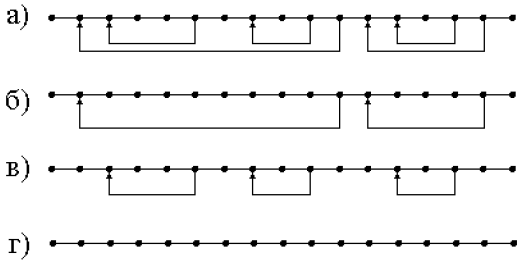



Рис. 3.18. Схема последовательного алгоритма: а - с разбиением вычислений на блоки; б - высший уровень иерархии, сегменты алгоритма - внешние циклы; в - второй уровень иерархии - вложенные циклы; г - третий уровень иерархии - арифметические выражения

При декомпозиции данных иерархическая классификация различает значения отдельных величин (низший уровень), векторов и матриц, многомерных массивов (высший уровень).

Блочно-иерархический метод можно рассматривать как модификацию этапов разбиения и установки коммуникаций восходящего подхода. Отличие заключается в выборе критерия окончания разбиения. Если в восходящем подходе разбиение заканчивается на самом мелком характерном фрагменте последовательного алгоритма, то применение рассматриваемого метода подразумевает проверку времени исполнения вычислений по алгоритму каждый раз после разбиения на более мелкие блоки при движении вниз по иерархической структуре вычислений или данных. При движении к иерархическим структурам низкого уровня увеличивается число задач параллельного алгоритма, большее количество арифметических операций выполняется одновременно, что сокращает длительность вычислений, связанных с исполнением арифметических операций. Однако при этом усложняется топология коммуникаций, задачам приходится обмениваться большим объемом данных - это, начиная



с определенного уровня иерархии, увеличивает общую длительность вычислений. Процесс разбиения в нисходящем подходе оканчивается на том иерархическом уровне $m_{opt}$, для которого время вычислений T по алгоритму характеризуется наименьшим значением $t_{min}$ (рис. 3.19).

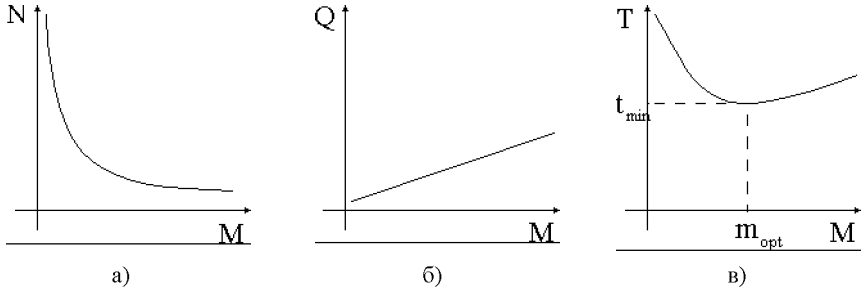



Рис. 3.19. Определение оптимального уровня блочной иерархической структуры при разбиении: а - зависимость количества арифметических операций N, производимых одной задачей параллельного алгоритма, от уровня иерархии M; б - зависимость от уровня иерархии объема данных Q, принимаемых и посылаемых по каналам одной (характерной) задачей алгоритма; в - аналогичная зависимость для времени вычислений по параллельному алгоритму

Проиллюстрируем применение блочно-иерархического метода на примере построения параллельного алгоритма перемножения квадратных матриц. Даны две матрицы A и B размерностью n×n. Элементы результирующей матрицы C=A×B находятся по формуле

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} .$$

На рис. 3.20 схематично записан последовательный алгоритм матричного перемножения.

Осуществляя функциональную декомпозицию, разобьем на блоки цикл $p_3$ - верхний уровень иерархии вычислений (рис. 3.21). Количество задач L параллельного алгоритма для простоты выберем таким, чтобы n делилось на L без остатка.

Данные между задачами алгоритма распределим так, как это показано на рис. 3.22.

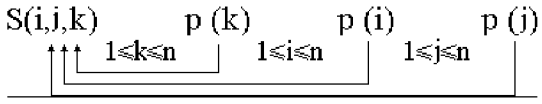



Рис. 3.20. Блочно-иерархическая структура вычислений по последовательному алгоритму матричного перемножения: S(i,j,k) - операция суммирования произведений $a_{ik}b_{kj}$ ; цикл $p_1(k)$ используется для вычисления одного элемента матрицы C; цикл $p_2(i)$ перебирает элементы матрицы C по строкам, цикл $p_3(j)$ по столбцам

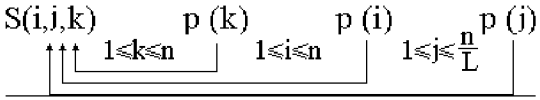



Рис. 3.21. Структура вычислений по параллельному алгоритму перемножения матриц без обмена данных между задачами

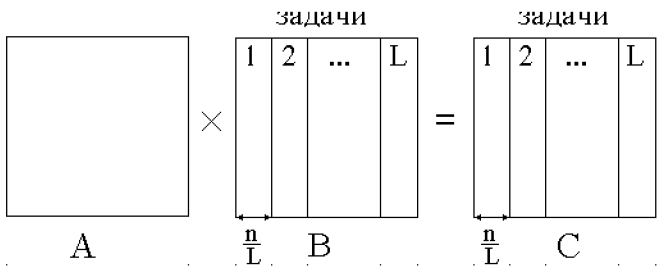



Рис. 3.22. Распределение данных между задачами алгоритма матричного перемножения без коммуникаций

Каждая задача алгоритма целиком хранит матрицу A и вертикальные полосы, состоящие из столбцов матриц B и C шириной n/L. Все данные, необходимые для вычисления любого элемента из полосы матрицы C, принадлежащей определенной задаче, наличествуют в памяти этой задачи. Таким образом, вычисления по алгоритму не связаны с установкой коммуникаций и обменом данными.

Часто при построении и исследовании параллельных алгоритмов рассматривают начальный этап, связанный с распределением данных. На этом этапе первая задача алгоритма считывает из файла начальные данные и распределяет их между остальными задачами. Здесь и далее будем предполагать, что рассматриваемые алгоритмы



входят в качестве модулей в более сложные алгоритмы, распределение данных в которых организовано таким образом, что на всех этапах (кроме первого и последнего) дополнительных пересылок, связанных с начальным распределением данных, не требуется.

Предложенный алгоритм матричного перемножения редко используется на практике из-за больших затрат памяти вычислительной системы. За увеличение скорости вычислений в L раз приходится платить увеличением объема занимаемой памяти (матрица A копируется L раз).

Запишем менее затратный алгоритм, изменив разбиение области данных (рис. 3.23).

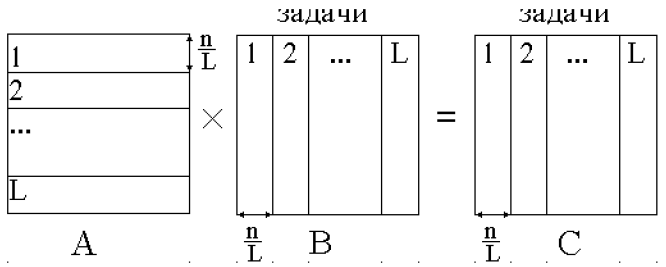



Рис. 3.23. Распределение матриц между задачами параллельного алгоритма матричного перемножения

Матрица A разделена между задачами на вертикальные полосы шириной n/L. При новой декомпозиции данных изменится структура вычислений по новому алгоритму (рис. 3.24).

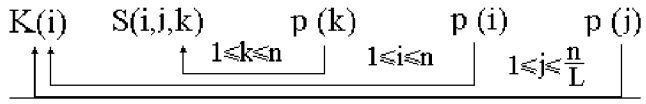



Рис. 3.24. Структура вычислений по параллельному алгоритму матричного перемножения с одномерной декомпозицией области данных

Операция K(i) - пересылка i-й строки матрицы A от задачи, в памяти которой эта строка содержится, всем остальным задачам алгоритма.

Первая задача перед вычислением элемента $c_{11}$ разошлет первую строку матрицы A остальным задачам. Второй задаче эта строка понадобится для нахождения элемента $c_{1,n/L+1}$, третьей - для нахождения $c_{1,2n/L+1}$, … , последней - для $c_{1,\frac{(L-1)n}{L}+1}$. Задачи со второй по последнюю не содержат в своей памяти первую строку матрицы A и вынуждены принимать ее от первой задачи. Строку A под номером n/L+1 будет рассылать вторая задача, а задачи 1,3,4, …,L эту строку принимают. И так далее, до последней строки матрицы, рассылкой которой займется последняя задача.

В новом алгоритме на вычисление одного элемента матрицы C приходится посылка одной строки матрицы A. На n умножений и n-1 сложение - пересылка n чисел. Независимо от характеристик вычислительной системы, на которой будут производиться вычисления по данному алгоритму, можно с уверенностью сказать, что большую часть времени ветви параллельного процесса будут обмениваться данными, а меньшую - производить арифметические операции. Время расчетов по представленному алгоритму может оказаться больше, чем по последовательному на однопроцессорной ЭВМ, если n/L достаточно велико.

В иерархической конструкции на рис. 3.24 переставим местами циклы p(i) и p(j) (рис. 3.25).

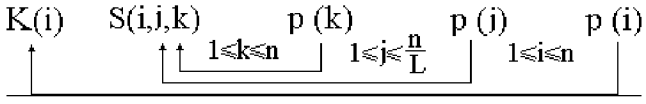



Рис. 3.25. Структура вычислений по модифицированному параллельному алгоритму с одномерной декомпозицией области данных

Такая перестановка допустима и в последовательном алгоритме, следовательно, иерархический принцип разбиения на блоки соблюдается.

Вычисляя элементы одной строки матрицы C (а значит, все элементы одной строки в вертикальной полосе матрицы C, принадлежащей каждой задаче), необходимо передать соответствующую строку матрицы A только один раз. После передачи n чисел



вычисляется n/L элементов C с использованием $n^2/L$ умножений и n(n-1)/L сложений. При таком соотношении объема передач и арифметических операций ветви параллельного вычислительного процесса на некоторых вычислительных системах большую часть времени вычислений будут уделять производству арифметических операций, а меньшую - передаче по сети строк матрицы A. Модифицированный алгоритм обеспечит сокращение длительности вычислений по сравнению с последовательным случаем.

## 3.2. Характеристики параллельных вычислительных процессов и алгоритмов

В предыдущем параграфе за меру эффективности параллельного алгоритма принималось время работы порожденного этим алгоритмом параллельного вычислительного процесса (ПВП). Действительно, при прочих равных показателях (количеству задач, максимальному числу соседей одной задачи, объему занимаемой памяти, точности полученного решения) предпочтительнее тот алгоритм, вычисления по которому производятся быстрее. Однако ПВП всегда связан с вычислительной системой, на которой протекает. Время его работы определяется быстродействием процессоров, конфигурацией сети, скоростью доступа к оперативной памяти и т.п. Сравнения двух алгоритмов на разных вычислительных системах могут дать разные результаты. Следовательно, говоря об эффективности, необходимо различать эффективность ПВП и алгоритмов.

Время вычислений, являясь абсолютной величиной, не всегда удобно для сравнения двух алгоритмов или процессов, когда принято пользоваться формулировками “во сколько раз” и “ограничено величиной”. В теории параллельных вычислений, сравнивая алгоритмы или процессы, пользуются относительными характеристиками: ускорением, эффективностью и масштабируемостью. Им и посвящен настоящий параграф.

### *3.2.1. Характеристики параллельных вычислительных процессов*

Измерив длительность вычислений на однопроцессорной ЭВМ и оставшись недовольным, исследователь может задаться вопросом:

во сколько раз ускорятся вычисления, окажись в его распоряжении вычислительная система, состоящая из N процессоров? Говоря языком теории параллельных вычислений, его интересует ускорение ПВП S, определяемое как отношение времени $T^1$ исполнения последовательного процесса ко времени $T^N$ исполнения ПВП на N процессорах:

$$S = \frac{T^1}{T^N}. \qquad (3.4)$$

Эффективностью ПВП называется производная от ускорения величина:

$$E = \frac{S}{N} = \frac{T^1}{NT^N}. \qquad (3.5)$$

Неопытный исследователь ожидает увидеть ускорение равным числу процессоров. Чаще всего на практике S значительно ниже N.

Остановимся на двух причинах этого явления, связав с ними два подхода к оценке ожидаемого ускорения ПВП. Первая причина кроется в структуре последовательного вычислительного процесса. Любой такой процесс содержит часть, которую нельзя исполнить параллельно, разделив между несколькими процессорами вычислительной системы. Существование такой части может быть связано с человеческим фактором (оператор последовательно вводит начальные данные, последовательно просматривает результаты), с особенностями алгоритма, порождающего данный процесс, с наличием "неделимых" операций (сложения, умножения, чтения одного числа из файла и т.п.). Обозначим время исполнения всех нераспараллеливаемых действий последовательного процесса за $t_н$ (рис. 3.26).

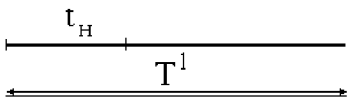



Рис. 3.26. Структура последовательного вычислительного процесса

Тогда $f = \frac{t_н}{T^1}$ есть доля длительности таких действий во времени всего процесса. При этом



$$T^N = fT^1 + (1-f)\frac{T^1}{N},$$

где первое слагаемое - длительность последовательной части ПВП, второе - время исполнения параллельной части.

Ускорение

$$S = \frac{1}{f + \frac{1-f}{N}}. \qquad (3.6)$$

Формула (3.6) выведена в предположении, что ветви ПВП не взаимодействуют друг с другом, что в общем случае неверно. Обмен данными увеличивает длительность ПВП, поэтому (3.6) следует обратить в неравенство

$$S \le \frac{1}{f + \frac{1-f}{N}}, \qquad (3.7)$$

известное как закон Амдала (по фамилии автора, опубликовавшего его в 1967 году).

Закон Амдала ограничивает сверху ускорение ПВП величиной, зависящей от количества процессоров вычислительной системы и структуры последовательного вычислительного процесса. При $N \to \infty$ $S \le 1/f$, для идеально распараллеливаемого последовательного процесса $S \le N$ (рис. 3.27).

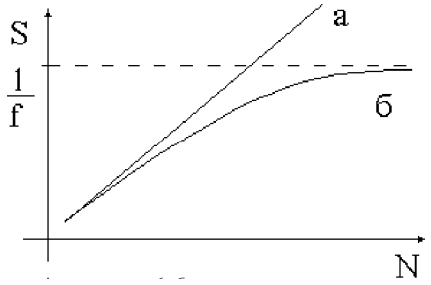



Рис. 3.27. Ограничения, налагаемые на ускорение ПВП законом Амдала:
а - ускорение при f=0, б - при f > 0

Если ускорение S=N принять за максимальное, то эффективность E показывает долю достигнутого ускорения от максимального.

Проиллюстрируем изложенное численными примерами. Пусть в нашем распоряжении имеется вычислительная система, состоящая из 1000 процессоров, а структура последовательного вычислительного процесса характеризуется величиной f=0,001 (нераспараллеливаемый фрагмент длится всего 0,1% от времени процесса). Тогда S≈500 и E≈0,5. Использование 1000 процессоров привело к ускорению всего в 500 раз. Для f=0,01 ускорение и эффективность снизятся до значений S≈91 и E≈0,091. Даже малые значения f при большом количестве процессоров серьезно ограничивают ускорение ПВП, снижая его эффективность.

Второй подход к оценке эффективности ПВП связан с учетом времени обмена сообщениями $t_{об}$ между ветвями процесса и времени простоев $t_{пр}$ ветвей, ожидающих приема сообщений. Синтезируя параллельные алгоритмы, $t_{пр}$ стремятся свести к нулю, что не всегда получается (например, при решении СЛАУ методом исключений Гаусса). Дальнейшие рассуждения верны в предположении, что ветви ПВП совершенно сбалансированы: все выполняют одинаковое количество арифметических операций (длительностью $t_{ар}$), обмениваются одинаковым объемом данных и вынуждены простаивать одно и то же время. При этом в один момент времени разные ветви могут производить различные действия. В рамках данного подхода предполагается, что, кроме арифметических операций, обменов и простоев, других действий не производится. Тогда

$$T^1=t_{ар}N,\ T^N=t_{ар}+t_{об}+t_{пр}$$

$$E=\frac{t_{ар}}{t_{ар}+t_{об}+t_{пр}}. \qquad (3.8)$$

Эффективность показывает долю, приходящуюся на время арифметических операций в длительности ПВП. Чем меньше времени ПВП тратит на обмены и простои и больше на арифметические операции - тем он эффективнее. Зная длительности последовательного и параллельного процессов, можно выразить совокупное время обменов и простоев:

$$t_{об}+t_{пр}=T^N-T^1/N.$$



В данном подходе раскрывается иной смысл ускорения. Величина N×E=S есть количество процессоров (из N), занятых исключительно арифметическими операциями; N(1-E)=N-S - количество процессоров, занятых только обменами и простоями. Эти числа условны, мы сводим все арифметические вычисления в одну группу процессоров (к тому же с нецелым числом таких процессоров), в то время как они распределены по всем процессорам вычислительной системы.

Если график зависимости ускорения от числа процессоров стагнирует начиная с некоторого N', значит, добавленные N-N' процессоров не производят полезных действий и их использование излишне (рис. 3.28).

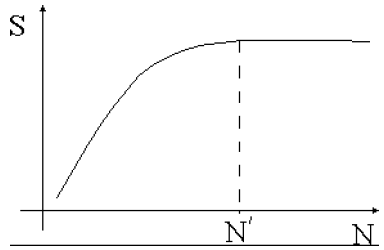



Рис 3.28. Определение оптимального числа процессоров для реализации ПВП

Ускорение оценивается величиной

$$S\le\frac{Nt_{ар}}{t_{ар}+t_{об}+t_{пр}}=\frac{N}{1+k}, \qquad (3.9)$$

где $k=\frac{t_{об}+t_{пр}}{t_{ар}}$.

На практике не удается добиться идеальной балансировки ветвей ПВП, этим объясняется знак неравенства в оценке (3.9). При $t_{об}+t_{пр}\to 0$ ускорение стремится к максимальному значению.

Во втором подходе к оценке ускорения ПВП (так же как и в первом) учитывается структура последовательного процесса. Величина f не входит в явном виде в выражение (3.9), но k является функцией от f. Параллельные ветви или дублируют нераспараллеливаемые вычисления (тогда $t_{ар}=t_{ар}(f)$), или ожидают, пока одна из ветвей производит такие вычисления ( $t_{пр}=t_{пр}(f)$).

На практике допустимо использование обеих оценок ускорения: если ПВП состоит из слабовзаимодействующих ветвей (3.7) и когда величину k можно оценить до реализации ПВП (3.9).

Зависимость ускорения от количества процессоров вычислительной системы не является исчерпывающей характеристикой семейства ПВП, порожденных одним алгоритмом. При увеличении объема данных, которым оперирует ПВП, значения f и k могут уменьшаться, ускорение на том же числе процессоров - увеличиваться. Изучая это свойство ПВП, исследователь иногда строит много графиков зависимостей S(N) для разного объема данных (рис. 3.29).

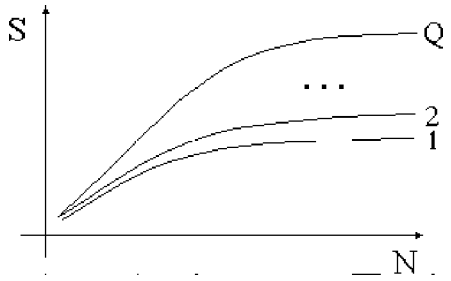



Рис. 3.29. Зависимости S(N) для разного объема данных V ПВП: $V=V_1$ (кривая 1), $V=V_2$ (кривая 2), ... , $V=V_Q$ (кривая Q), $V_1<V_2<\ldots<V_Q$

В теории параллельных вычислений принято пользоваться другой величиной - масштабируемостью M:

$$M=\frac{V^N}{V^1} \text{ при } T^1(V^1)=T^N(V^N).$$

Масштабируемость ПВП есть отношение объема данных $V^N$ параллельного процесса к объему данных $V^1$ последовательного процесса. Длительность обоих процессов, используемых для определения масштабируемости, должна совпадать. Масштабируемость показывает, во сколько раз увеличится объем данных при реализации ПВП на N процессорах по сравнению с однопроцессорной ЭВМ, если время реализации останется постоянным.



### *3.2.2. Характеристики параллельных алгоритмов*

Говоря об алгоритмах, в отличие от вычислительных процессов, мы не можем оперировать временем работы, выраженным в секундах, физическим временем обмена сообщениями, простоев и арифметических операций. В нашем распоряжении будут количество арифметических операций, исполняемых характерной задачей алгоритма, объем посылаемых и принимаемых ею данных. Простои некоторой задачи обычно связаны с арифметическими действиями другой задачи, результаты вычислений которой ожидает некоторая, и выражаются через количество соответствующих операций другой задачи.

Следовательно, ускорение сбалансированного алгоритма

$$S=\frac{L\times\tau_{ap}}{\tau_{ap}+\tau_{об}+\tau_{пр}}, \qquad (3.10)$$

где $\tau_{ap}$ - "условная" длительность арифметических операций характерной задачи алгоритма, выраженная через количество операций сложения, умножения, деления, и время одной такой операции ($\tau_\times$ ,$\tau_\div$ ,$\tau_+$); $\tau_{об}$ - "условная" длительность обменов алгоритма, выраженная через объем пересылаемых данных, и время пересылки по каналу одного числа ($\tau_к$); L - число задач алгоритма.

По аналогии с эффективностью ПВП за эффективность параллельного алгоритма примем величину

$$\eta=\frac{\tau_{ap}}{\tau_{ap}+\tau_{об}+\tau_{пр}}, \qquad (3.11)$$

показывающую долю времени, затрачиваемого на арифметические операции во всей длительности вычислений. В литературе встречается другое определение эффективности алгоритма:

$$\delta=\frac{\tau_{об}+\tau_{пр}}{\tau_{ap}}, \qquad (3.12)$$

обратное предыдущему. Отличие связано с разными областями применения критериев $\eta$ и $\delta$.

Рассмотрим параллельный алгоритм матричного перемножения из 3.1.2. Будем увеличивать число задач алгоритма, не меняя раз-

мерности матриц. Объем пересылок при этом останется постоянным. Каждая задача отправляет остальным свою часть матрицы A и принимает от них все другие части этой матрицы. С ростом L уменьшается ширина полосы матрицы A, принадлежащей каждой задаче (рис. 3.32), но объем пересылок, в которых участвует задача, не изменяется. Уменьшается количество посылаемых задачей чисел, увеличивается количество принимаемых (в сумме они составляют объем матрицы A). Общее время вычислений по алгоритму сокращается, эффективность по критерию $\delta$ возрастает, по критерию $\eta$ падает. Если более эффективным алгоритмом считать тот, вычисления по которому производятся быстрее, то критерий $\eta$ в данном случае, применять нельзя.

Решая этот парадокс, откажемся от общего понятия эффективности параллельного алгоритма, рассматривая отдельно эффективность по времени (чем быстрее производятся вычисления, тем алгоритм эффективнее) и эффективность по вычислениям (чем выше доля арифметических операций в общем количестве действий, тем лучше). Для алгоритма матричного перемножения из п. 3.1.2 эффективность по времени вычисляется по критерию $\delta$, по вычислениям - с помощью критерия $\eta$.

На рис. 3.30 предложен другой пример, для которого необходимо подобрать критерии эффективностей при сравнении двух алгоритмов, представленных временами арифметических операций и обменов с простоями.

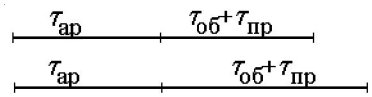



Рис. 3.30. Соотношение длительности арифметических операций, обменов и простоев двух сравниваемых алгоритмов

Если в качестве критерия эффективности по времени опять использовать $\delta$, то первый алгоритм, содержащий больше обменов и простоев при равном времени арифметических операций, окажется более эффективным, что неверно. Здесь для оценки эффективности по времени уместен критерий $\eta$.



Подбирая критерий эффективности по времени, сравнивая два разных алгоритма или один алгоритм с варьирующимся числом задач, будем руководствоваться следующими соображениями. При одинаковой для обоих алгоритмов величине $\tau_{ар}$ применяем критерий $\eta$ (эффективней по времени будет алгоритм, задачи которого меньше обмениваются данными и простаивают). Если у двух алгоритмов одинаковы времена $\tau_{об}+\tau_{пр}$, верен критерий $\delta$ (эффективней по времени алгоритм, производящий меньше арифметических вычислений). Если все "условные" времена для обоих алгоритмов различны, остается сравнивать напрямую длительность вычислений по алгоритму $\tau_{ар}+\tau_{об}+\tau_{пр}$, отказываясь от обоих критериев эффективности.

Определяя эффективность по вычислениям, пользуются исключительно критерием $\eta$, как точно соответствующим определению такой эффективности.

Хочется обратить внимание читателей на то, что в настоящем параграфе рассматривалась эффективность параллельной реализации вычислительного процесса и эффективность распараллеливания алгоритма. Под эффективностью не понималась точность полученного ПВП численного решения, его сходимость к аналитическому, корректность и т.д. Эффективно распараллеленный алгоритм может обладать худшими характеристиками по сравнению с неэффективно распараллеленным, когда речь идет о точности полученного решения, порядке его сходимости и т.д.

## 3.3. Параллельные алгоритмы линейной алгебры

Предваряя основной материал параграфа, коснемся проблемы распределения данных между задачами параллельного алгоритма. Предположим существование одномерного массива размерности n. Необходимо распределить все его элементы между L задачами ($L \le n$) как можно более равномерно, сохраняя балансировку задач. Но так, чтобы никакие две задачи не содержали элементы распределяемого массива с одинаковыми порядковыми номерами (исключить дублирование). Если n не делится на L без остатка, используют функцию

$$\gamma_{n,L}(l)=\begin{cases}\left[\frac{n}{L}\right]+1, \text{при } n\%L\geq l\\ \left[\frac{n}{L}\right], \text{при } n\%L<l\end{cases},$$

где l - номер задачи алгоритма ($1\le l\le L$); [..] - операция округления до ближайшего меньшего целого; с помощью бинарной операции a%b находится остаток от деления a на b, $n\%L=n-\left[\frac{n}{L}\right]\times L$.

Задаче с номером l отойдет $\gamma_{n,L}(l)$ элементов распределяемого массива, от элемента под номером $\sum_{g=1}^{l-1}\gamma_{n,L}(g)+1$ до элемента $\sum_{g=1}^{l}\gamma_{n,L}(g)$.

Например, для n=100 и L=15 $\left[\frac{n}{L}\right]$=6 n%L=10. В табл. 3.1 представлены значения функции $\gamma_{n,L}(l)$ для всех l.

Таблица 3.1

Значения функции $\gamma_{n,L}(l)$ при n=100 и L=15

| L | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\gamma_{n,L}(l)$ | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 6 | 6 | 6 | 6 | 6 |

Задачи с 1-й по 10-ю хранят по 7 элементов массива (например 10-я задача с 64-го по 70-й элемент), задачи с 11-й по 15-ю по 6 элементов (15-я задача с 95-го по 100-й элемент).

### *3.3.1. Алгоритмы матричного перемножения*

#### *3.3.1.1. Алгоритмы матричного перемножения с одномерной декомпозицией области данных*

Вернемся к алгоритмам матричного перемножения из п. 3.1.2, адаптировав их к прямоугольным матрицам A (размерностью I×Q), B (размерностью Q×J) и C (I×J), допуская деление с остатком I и J на L.



*Алгоритм 1*

В алгоритме без коммуникаций [2] принято хранить в памяти всех задач целиком ту матрицу, размерность которой меньше. Если I≤J, то декомпозиция данных остается такой, как изображено на рис. 3.22, иначе дублируем L раз матрицу B, а матрицу A распределяем между задачами горизонтальными полосами. При этом меняется структура хранения матрицы C - с вертикальных полос на горизонтальные (рис. 3.31).

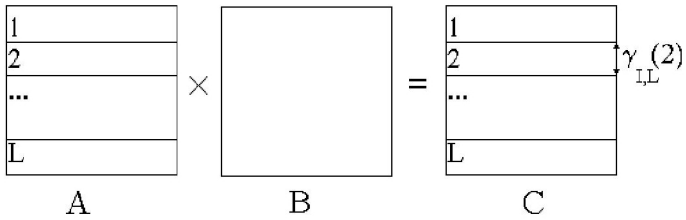



Рис 3.31. Декомпозиция данных в алгоритме 1 при I>J

Учтем, что $\max_{1\le l\le L}\gamma_{I,L}(l)=\gamma_{I,L}(1)$ (всегда). Тогда ускорение алгоритма 1 при I>J составит

$$S=\frac{I\times J\times(Q\tau_{\times}+(Q-1)\tau_{+})}{\gamma_{I,L}(1)\times J\times(Q\tau_{\times}+(Q-1)\tau_{+})}=\frac{I}{\gamma_{I,L}(1)}\approx L.$$

Здесь I×J - количество вычисляемых элементов матрицы C; $Q\tau_{\times}+(Q-1)\tau_{+}$ - длительность арифметических расчетов при нахождении одного элемента матрицы C; $\gamma_{I,L}(1)\times J$ - количество элементов матрицы C, вычисляемых первой задачей алгоритма.

*Алгоритм 2*

Следующий алгоритм [2] с одномерной декомпозицией данных характеризуется пересылками столбцов матрицы A. При I≤J используем декомпозицию данных, представленную на рис. 3.23, иначе (сокращая объем пересылок) уместней пересылать столбцы матрицы B, разбивая матрицу C на горизонтальные полосы (рис. 3.32).

После того как каждая задача (кроме передающей) примет j-й столбец матрицы B, начнутся вычисления элементов j-го столбца матрицы C в горизонтальных полосах всех задач.

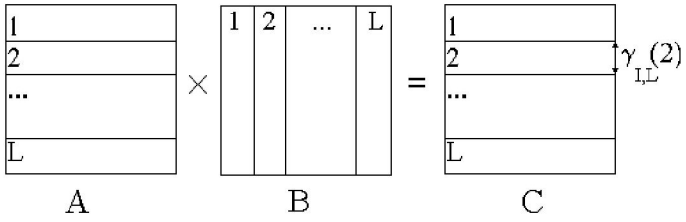



Рис. 3.32. Декомпозиция данных в алгоритме 2 при I>J

На рис. 3.33 представлена блок-схема алгоритма 2 для случая $I \le J$. Все задачи производят вычисления по записанной блок-схеме, каждая характеризуется только своим номером l. Задача под номером $\alpha$ хранит i-ю строку матрицы A, которую данной задаче необходимо пересылать остальным задачам алгоритма при расчете элементов i-й строки матрицы C.

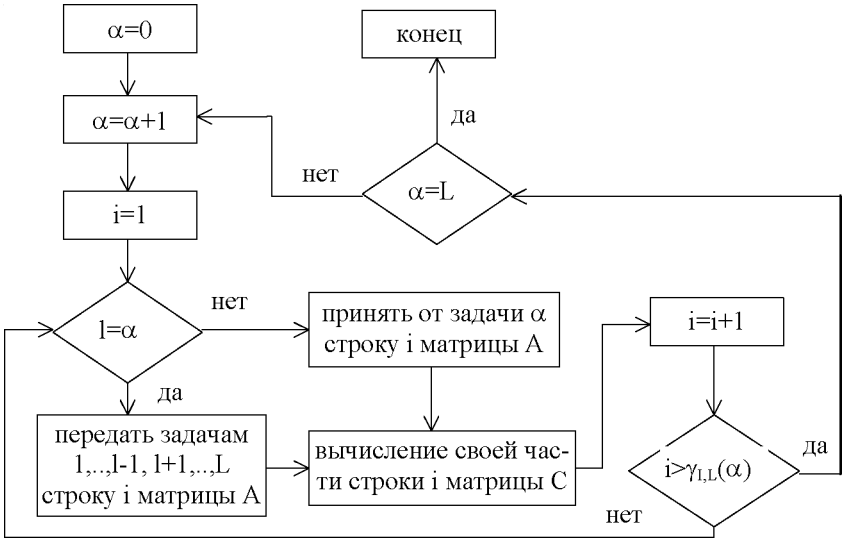



Рис. 3.33. Блок-схема алгоритма 2

Задача под номером l вычисляет элементы матрицы C со столбца $\sum_{g=1}^{l-1}\gamma_{J,L}(g)+1$ по столбец $\sum_{g=1}^{l}\gamma_{J,L}(g)$ и посылает элементы матрицы



A со строки $\sum_{g=1}^{l-1}\gamma_{I,L}(g)+1$ по строку $\sum_{g=1}^{l}\gamma_{I,L}(g)$ (остальную часть матрицы A она принимает от других задач). Когда номер строки i выходит за размер горизонтальной полосы матрицы A, принадлежащей задаче $\alpha$, число $\alpha$ увеличивается на единицу и рассылку строк продолжает другая задача. После перебора всех строк матрицы C алгоритм завершается.

Ускорение исследуемого алгоритма составит величину

$$S=\frac{I\times J\times(Q\tau_{\times}+(Q-1)\tau_{+})}{\gamma_{J,L}(l)\times I\times(Q\tau_{\times}+(Q-1)\tau_{+})+I\times Q\tau_{k}}\approx$$

$$\approx\frac{J}{\gamma_{J,L}(l)+k}\ \underset{\text{при } L\to J}{\to}\ \frac{J}{l+k}\ \underset{\text{при } k>>1}{\approx}\ \frac{J}{k},$$

где $k=\frac{\tau_k}{\tau_{\times}+\tau_{+}}$, а $I\times Q\tau_k$ - время пересылки и приема всей матрицы A.

Эффективность по времени для алгоритма 2 составляет

$$\delta=\frac{I\times Q\tau_k}{\gamma_{J,L}(l)\times I\times(Q\tau_{\times}+(Q-1)\tau_{+})}\approx\frac{k}{\gamma_{J,L}(l)}\ \underset{\text{при } L\to J}{\to}\ k,$$

по вычислениям

$$\eta=\frac{I\times\gamma_{J,L}(l)\times(Q\tau_{\times}+(Q-1)\tau_{+})}{\gamma_{J,L}(l)\times I\times(Q\tau_{\times}+(Q-1)\tau_{+})+I\times Q\tau_{k}}\approx$$

$$\approx\frac{\gamma_{J,L}(l)}{\gamma_{J,L}(l)+k}\ \underset{\text{при } L\to J}{\to}\ \frac{1}{l+k}\ \underset{\text{при } k>>1}{\approx}\ \frac{1}{k}.$$

Чем выше эффективность по времени, тем ниже она по вычислениям. Длительность вычислений по алгоритму минимальна при L=J, но доля пересылок при этом максимальна, процессоры большую часть времени занимаются организацией передачи и приема сообщений по сети. При малых значениях $\eta$ вычислительная система используется как почтовая служба. На практике находят компромисс между эффективностью по времени и вычислениям.

### *3.3.1.2. Алгоритмы матричного перемножения с двумерной декомпозицией области данных*

Упрощая изложение, ограничимся рассмотрением квадратных матриц размерности Q×Q, таковой, что Q делится на $m = \sqrt{L}$ ($m \in N$) без остатка. Интересующийся читатель самостоятельно, по аналогии с алгоритмами 1 и 2, повторит изложенное далее для более общего случая. Заметим, что количество задач L в алгоритме 4 обязательно должно быть квадратом целого числа.

*Алгоритм 3*

В настоящем параграфе двумерная декомпозиция данных соответствует матрицам с блочной структурой [3,10,11] (рис. 3.34).

$$\begin{pmatrix} c_{11} & c_{12} & \dots & c_{1Q} \\ c_{21} & c_{22} & \dots & c_{2Q} \\ \dots & \dots & \dots & \dots \\ c_{Q1} & c_{Q2} & \dots & c_{QQ} \end{pmatrix} ====> \begin{pmatrix} \mathbf{c}_{11} & \mathbf{c}_{12} & \dots & \mathbf{c}_{1m} \\ \mathbf{c}_{21} & \mathbf{c}_{22} & \dots & \mathbf{c}_{2m} \\ \dots & \dots & \dots & \dots \\ \mathbf{c}_{m1} & \mathbf{c}_{m2} & \dots & \mathbf{c}_{m,m} \end{pmatrix}$$

Рис. 3.34. Двумерная декомпозиция матрицы C. Блоки матрицы выделены жирным шрифтом

На рис. 3.34 блок $\mathbf{c}_{ij}$ ($1 \le i \le m$, $1 \le j \le m$) состоит из следующих элементов C:

$$\mathbf{c}_{ij} = \begin{pmatrix} c_{\frac{Q}{m}(i-1)+1,\frac{Q}{m}(j-1)+1} & c_{\frac{Q}{m}(i-1)+1,\frac{Q}{m}(j-1)+2} & \dots & c_{\frac{Q}{m}(i-1)+1,\frac{Q}{m}j} \\ c_{\frac{Q}{m}(i-1)+2,\frac{Q}{m}(j-1)+1} & c_{\frac{Q}{m}(i-1)+2,\frac{Q}{m}(j-1)+2} & \dots & c_{\frac{Q}{m}(i-1)+2,\frac{Q}{m}j} \\ \dots & \dots & \dots & \dots \\ c_{\frac{Q}{m}i,\frac{Q}{m}(j-1)+1} & c_{\frac{Q}{m}i,\frac{Q}{m}(j-1)+2} & \dots & c_{\frac{Q}{m}i,\frac{Q}{m}j} \end{pmatrix}.$$

Каждому блоку матрицы C поставим во взаимно однозначное соответствие одну задачу параллельного алгоритма, присвоив ей в качестве номера индекс блока (блоку $\mathbf{c}_{ij}$ соответствует задача (i,j)). Арифметические операции с блочными матрицами производятся аналогично операциям с обычными матрицами при условии



совпадения размерности блоков. Так, при перемножении блочных матриц $\mathbf{c}_{ij} = \sum_{g=1}^{m} \mathbf{a}_{ig}\mathbf{b}_{gj}$. Следовательно, для расчета блока $\mathbf{c}_{ij}$ задаче (i,j) необходимы блоки $\mathbf{a}_{i1}$, $\mathbf{a}_{i2}$, …, $\mathbf{a}_{i,m}$ матрицы A и блоки $\mathbf{b}_{i1}$, $\mathbf{b}_{i2}$, …, $\mathbf{b}_{i,m}$ матрицы B (рис. 3.35).

$$\begin{pmatrix} & & & \\ \mathbf{a}_{i1} & \mathbf{a}_{i2} & \dots & \mathbf{a}_{im} \\ & & & \end{pmatrix} \times \begin{pmatrix} \mathbf{b}_{j1} \\ \mathbf{b}_{j2} \\ \dots \\ \mathbf{b}_{jm} \end{pmatrix} = \begin{pmatrix} \\ \mathbf{c}_{ij} \\ \end{pmatrix}$$

Рис. 3.35. Область данных задачи (i,j) алгоритма 3

Организовав хранение всех требуемых блоков A и B в задаче (i,j), избавляемся от необходимости устанавливать каналы между задачами алгоритма 3. При этом каждый блок матриц A и B дублируется m задачами. Использование алгоритма 2, как и последовательного алгоритма, предполагает хранение 3 матриц размерности Q×Q (матриц A,B и C); использование алгоритма 1 - L+2 матриц (B,C и L раз матрицы A); алгоритма 3 - $2\sqrt{L}+1$ матриц (матрицы C и m раз матриц A и B). Для L=16 алгоритм 3 потребует в 2 раза меньше памяти, чем алгоритм 1, но в 3 раза больше, чем последовательный алгоритм (табл. 3.2).

Таблица 3.2

Зависимость объема требуемой памяти (в матрицах размерности Q×Q) от количества задач алгоритма

| Алго-ритм | Количество задач алгоритма | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 4 | 9 | 16 | 25 | 36 | 49 | 64 | 81 | 100 |
| 2, 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 1 | 6 | 11 | 18 | 27 | 38 | 51 | 66 | 83 | 102 |
| 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 |

При одномерной декомпозиции для вычисления строки матрицы C необходима вся матрица B, столбца - вся матрица A. Двумерная декомпозиция характеризуется вычислением фрагмента C, не

содержащего полностью ни строки, ни столбца. Для его определения требуются фрагменты матриц A и B. Поэтому алгоритм 3 нуждается в меньшей памяти, чем алгоритм 1, а алгоритм 4, как далее будет ясно, подразумевает пересылку меньшего объема данных, чем алгоритм 2.

Предел ускорения алгоритма 3

$$S = \frac{Q^2(Q\tau_\times + (Q-1)\tau_+)}{\frac{Q^2}{L}(Q\tau_\times + (Q-1)\tau_+)} = L \underset{L \to Q^2}{\to} Q^2$$

равен квадрату от максимального значения ускорения алгоритма 1. При разбиении и по строкам и по столбцам количество задач возрастает квадратично. Как и алгоритм 1, алгоритм 3 на практике не используется из-за "жадности" к памяти вычислительной системы.

*Алгоритм 4*

Установим коммуникации между задачами так, как это принято в сетевой конфигурации тор (рис. 3.36).

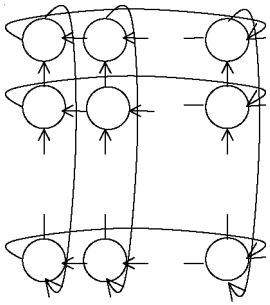


Рис. 3.36. Топология коммуникаций алгоритма 4

$\mathbf{a}_{11}\,\mathbf{b}_{11}$ $\mathbf{a}_{12}\,\mathbf{b}_{12}$ $\mathbf{a}_{13}\,\mathbf{b}_{13}$
$\mathbf{a}_{21}\,\mathbf{b}_{21}$ $\mathbf{a}_{22}\,\mathbf{b}_{22}$ $\mathbf{a}_{23}\,\mathbf{b}_{23}$
$\mathbf{a}_{31}\,\mathbf{b}_{31}$ $\mathbf{a}_{32}\,\mathbf{b}_{32}$ $\mathbf{a}_{33}\,\mathbf{b}_{33}$

Рис. 3.37. Неправильное распределение блоков матриц A и B между задачами алгоритма 4 при L=9



При вычислении $\mathbf{c}_{ij}$ задача (i,j) будет получать необходимые блоки матрицы A от соседа справа (понадобится вся строка i блочной матрицы A), матрицы B от соседа снизу (понадобится весь столбец j блочной матрицы B). Соседями снизу для задач (m,j) являются задачи (1,j), соседями справа для (i,m) являются (i,1).

Поместим при начальном распределении данных в задачу (i,j) блоки $\mathbf{a}_{ij}$ и $\mathbf{b}_{ij}$. Тогда только задачи (i,i) могут начать вычисления по алгоритму, остальные не содержат необходимых блоков. Например, при L=9 (рис. 3.37) сразу перемножить блоки **a** и **b** позволительно задачам (1,1), (2,2), (3,3). Задаче (2,1) для того, чтобы перемножить блок $\mathbf{a}_{21}$, которым она располагает, на соответствующий блок матрицы B, следует ждать передачи $\mathbf{b}_{11}$ от задачи (1,1) через (3,1). Для перемножения $\mathbf{b}_{21}$, который тоже имеется у (2,1), ей необходимо дождаться $\mathbf{a}_{22}$ от соседа справа. Путь $\mathbf{a}_{22}$ к (2,1) короче, чем $\mathbf{b}_{11}$, поэтому при начальном распределении данных первый столбец блочной матрицы B оставим на месте, а вторую строку блочной матрицы A циклически сместим на позицию влево. Помещая $\mathbf{a}_{33}$ к $\mathbf{b}_{31}$ в задаче (3,1), циклически сместим на две позиции влево третью строку блочной матрицы A (рис. 3.38). Теперь (1,1), (1,2), (1,3) могут начать вычисления. Обратимся к задаче (1,2). Ей необходим блок $\mathbf{b}_{22}$ или $\mathbf{a}_{11}$. Не нарушая декомпозиции для сформированных задач, циклически сместим второй столбец блочной матрицы B на позицию вверх. Задаче (3,1) необходим блок $\mathbf{b}_{33}$, сместим третий столбец на две позиции вверх (рис. 3.39). Все задачи готовы начать вычисления.

$\mathbf{a}_{11}\,\mathbf{b}_{11}$ $\mathbf{a}_{12}\,\mathbf{b}_{12}$ $\mathbf{a}_{13}\,\mathbf{b}_{13}$
$\mathbf{a}_{22}\,\mathbf{b}_{21}$ $\mathbf{a}_{23}\,\mathbf{b}_{22}$ $\mathbf{a}_{21}\,\mathbf{b}_{23}$
$\mathbf{a}_{33}\,\mathbf{b}_{31}$ $\mathbf{a}_{31}\,\mathbf{b}_{32}$ $\mathbf{a}_{32}\,\mathbf{b}_{33}$

Рис. 3.38. Декомпозиция со смещенными строками блочной матрицы A

$\mathbf{a}_{11}\,\mathbf{b}_{11}$ $\mathbf{a}_{12}\,\mathbf{b}_{22}$ $\mathbf{a}_{13}\,\mathbf{b}_{33}$
$\mathbf{a}_{22}\,\mathbf{b}_{21}$ $\mathbf{a}_{23}\,\mathbf{b}_{32}$ $\mathbf{a}_{21}\,\mathbf{b}_{13}$
$\mathbf{a}_{33}\,\mathbf{b}_{31}$ $\mathbf{a}_{31}\,\mathbf{b}_{12}$ $\mathbf{a}_{32}\,\mathbf{b}_{23}$

Рис. 3.39. Правильная декомпозиция блочных матриц A и B

В общем случае i-я строка блочной матрицы A циклически смещается на i-1 позицию влево, j-й столбец блочной матрицы B - на j-1 позицию вверх. Перед началом вычислений задаче (i,j) будут принадлежать блоки $\mathbf{a}_{ig}$ и $\mathbf{b}_{gj}$, где $g=\begin{cases} i+j-1, \text{ если } i+j-1\le m \\ i+j-1-m, \text{ если } i+j-1>m \end{cases}$.

Рассмотрим действия, производимые задачей (3,2) девятизадачного алгоритма. Перемножив $\mathbf{a}_{31}$ на $\mathbf{b}_{12}$, она передаст левому соседу (3,2) блок $\mathbf{a}_{31}$, приняв от правого соседа (3,3) $\mathbf{a}_{32}$. Блок $\mathbf{b}_{12}$ отправится соседу сверху (2,2), от нижнего соседа (1,2) придет $\mathbf{b}_{22}$ (рис. 3.40, а).

а)
$\mathbf{a}_{12}\,\mathbf{b}_{21}$ $\mathbf{a}_{13}\,\mathbf{b}_{32}$ $\mathbf{a}_{11}\,\mathbf{b}_{13}$
$\mathbf{a}_{23}\,\mathbf{b}_{31}$ $\mathbf{a}_{21}\,\mathbf{b}_{12}$ $\mathbf{a}_{22}\,\mathbf{b}_{23}$
$\mathbf{a}_{31}\,\mathbf{b}_{11}$ $\mathbf{a}_{32}\,\mathbf{b}_{22}$ $\mathbf{a}_{33}\,\mathbf{b}_{33}$

б)
$\mathbf{a}_{13}\,\mathbf{b}_{31}$ $\mathbf{a}_{11}\,\mathbf{b}_{12}$ $\mathbf{a}_{12}\,\mathbf{b}_{23}$
$\mathbf{a}_{21}\,\mathbf{b}_{11}$ $\mathbf{a}_{22}\,\mathbf{b}_{22}$ $\mathbf{a}_{23}\,\mathbf{b}_{33}$
$\mathbf{a}_{32}\,\mathbf{b}_{21}$ $\mathbf{a}_{33}\,\mathbf{b}_{32}$ $\mathbf{a}_{31}\,\mathbf{b}_{13}$

Рис. 3.40. Распределение блоков перед вторым (а) и третьим (б) перемножениями

По сравнению с предыдущим распределением все строки блочной матрицы A циклически смещаются на позицию влево, столбцы B - на позицию вверх.

Полученные блоки перемножаются, складываются с предыдущим результатом, пересылаются левому и верхнему соседям. От правого и нижнего соседей приходят $\mathbf{a}_{33}$ и $\mathbf{b}_{32}$ (рис. 3.40, б). В итоге $\mathbf{c}_{32}=\mathbf{a}_{31}\mathbf{в}_{12}+\mathbf{a}_{32}\mathbf{в}_{22}+\mathbf{a}_{33}\mathbf{в}_{32}$.

В общем случае каждая задача L задачного алгоритма произведет следующие действия:

for q=1 to m-1 do begin

**c**=**c**+**a**×**b**;

передать **a** соседу слева; принять **a** от соседа справа;

передать **b** соседу сверху; принять **b** от соседа снизу;

end;

**c**=**c**+**a**×**b**;

Все строки блочной матрицы A совершат круг по горизонтали, все столбцы блочной матрицы B - по вертикали. Каждая задача получит необходимые для перемножения блоки.



Сравним алгоритмы 2 и 4 при равном количестве задач L. Длительность арифметических операций, производимых двумя задачами разных алгоритмов, одинакова: $Q\times\frac{Q}{L}(Q\tau_\times+(Q-1)\tau_+)$ - для любой задачи алгоритма 2; $\frac{Q}{\sqrt{L}}\times\frac{Q}{\sqrt{L}}(Q\tau_\times+(Q-1)\tau_+)$ - для задачи алгоритма 4. Что неудивительно, одна матрица C, пусть и разными способами, разбивается на одинаковое количество равных частей. Объем посылаемых и принимаемых данных разный: $\frac{Q^2}{L}+\frac{L-1}{L}Q^2$ - пересылки алгоритма 2 (посылка полосы матрицы A - первое слагаемое, второе слагаемое - прием остальной части матрицы A); $\left(2\frac{Q^2}{L}+2\frac{Q^2}{L}\right)\left(\sqrt{L}-1\right)$ - пересылки задач алгоритма 4 (первое слагаемое первой скобки - посылки **a** и **b** левому и верхнему соседям, второе слагаемое - прием **a** и **b** от правого и нижнего соседей, сомножитель - количество итераций цикла по q). В табл. 3.3 представлено численное сравнение объема пересылок обоих алгоритмов.

Таблица 3.3

Отношение количества данных $V^4$, пересылаемых алгоритмом 4, к аналогичному параметру $V^2$ алгоритма 2

| Пересылаемые данные | Количество задач L | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 4 | 9 | 16 | 25 | 36 | 49 | 64 | 81 | 100 |
| $V^4/V^2$ | 1 | 0,89 | 0,75 | 0,64 | 0,56 | 0,49 | 0,44 | 0,4 | 0,36 |

Коммуникации алгоритма 4 имеют еще одно преимущество - они локальны. На наиболее распространенный для кластерных вычислительных систем сетевой топологии гиперкуб передача сообщений одного объема, связанная с вычислениями по алгоритму 4, произойдет быстрее. В алгоритме 2 любая задача связана каналами с L-1 задачей (коммуникации глобальны), в алгоритме 4 с четырьмя соседями, а один процессор в гиперкубе сообщается напрямую с $\log_2 L$ процессорами. Задачи четвертого алгоритма будут общаться напрямую, второго алгоритма - опосредованно.

Ускорение алгоритма 4

$$S=\frac{Q^2\left(Q\tau_{\times}+(Q-1)\tau_{+}\right)}{\sqrt{L}\,\frac{Q^2}{L}\left(\frac{Q}{\sqrt{L}}\tau_{\times}+\left(\frac{Q}{\sqrt{L}}-1\right)\tau_{+}\right)+4\frac{Q^2}{L}\left(\sqrt{L}-1\right)\tau_k}\approx$$

$$\approx\frac{Q}{\frac{Q}{L}+\frac{4}{\sqrt{L}}k}\underset{L\to Q^2}{\to}\frac{Q^2}{1+4k}\underset{k>>1}{\approx}\frac{Q^2}{4k}$$

в 4k раз меньше ускорения алгоритма 3. С ростом числа задач меняется как объем вычислений (для одной задачи алгоритма 4), так и объем пересылок, поэтому критерии $\delta$ и $\eta$ по смыслу не соответствуют эффективности по времени.

Эффективность по вычислениям

$$S=\frac{\sqrt{L}\,\frac{Q^2}{L}\left(\frac{Q}{\sqrt{L}}\tau_{\times}+\left(\frac{Q}{\sqrt{L}}-1\right)\tau_{+}\right)}{\sqrt{L}\,\frac{Q^2}{L}\left(\frac{Q}{\sqrt{L}}\tau_{\times}+\left(\frac{Q}{\sqrt{L}}-1\right)\tau_{+}\right)+4\frac{Q^2}{L}\left(\sqrt{L}-1\right)\tau_k}\approx$$

$$\approx\frac{Q}{Q+4\sqrt{L}k}\underset{L\to Q^2}{\to}\frac{1}{1+4k}\underset{k>>1}{\approx}\frac{1}{4k}$$

для предельного количества задач в 4 раза меньше соответствующей характеристики второго алгоритма. К преимуществам второго алгоритма над четвертым также следует отнести произвольность в выборе числа задач. Если в распоряжении исследователя имеется 8 однопроцессорных ЭВМ, связанных сетью, то, реализуя алгоритм 4, можно использовать только половину вычислительной мощности системы.

При кодировании алгоритмов 2,4 следует применять операторы асинхронной посылки и приема данных, совмещая во времени арифметические операции с пересылкой сообщений по сети. Ускорение ПВП, порожденных такими программами, может превзойти приведенные теоретические оценки.



В заключение разговора об алгоритмах матричного перемножения сообщим читателю о существовании “быстрых” последовательных методов Виноградова и Штрассена, позволяющих добиться существенного (в несколько раз) ускорения вычислений на однопроцессорной ЭВМ. Их распараллеливание представляет особый интерес.

### *3.3.2. Алгоритмы решения систем линейных алгебраических уравнений*

Из всех методов решения систем линейных алгебраических уравнений [1] (СЛАУ) выберем метод Крамера (первый, изучаемый студентами в курсе линейной алгебры), метод исключения Гаусса и метод простой однопараметрической итерации (типичные представители прямых и итерационных методов). Для них оценим принципиальную необходимость синтеза параллельных алгоритмов. Представим существование вычислительной системы, имеющей форму куба со стороной R=1 км и состоящей из процессоров с линейным размером $r=10^{-8}$ см (диаметр атома). Пусть время производства таким процессором одной операции равно времени прохождения луча света через него: $t_{опер}= r/c$, где c - скорость света в вакууме. Тогда быстродействие (число операций в секунду) вычислительной системы составит

$$G=\frac{R^3}{r^3}\frac{1}{t_{опер}}=\frac{R^3c}{r^4}\approx 3\cdot 10^{57,9}.$$

Сравним количество арифметических операций, характерных для изучаемых численных методов при решении СЛАУ с n неизвестными: $O(n!n)$ для метода Крамера; $O(n^3)$ для метода исключений Гаусса и $O(n^2)$ для метода простой однопараметрической итерации. Метод Крамера характеризуется очень большим объемом вычислений, например $100!=10^{158}>>G$. За время жизни Вселенной (15 миллиардов лет) на представленном вычислительном комплексе методом Крамера удастся решить СЛАУ с n=55. Очевидно, нет нужды в параллельном алгоритме для такого метода. Даже для небольших СЛАУ, используя мощную вычислительную технику, человечество может не дождаться результатов вычислений.

### *3.3.2.1. Параллельные алгоритмы метода простой однопараметрической итерации*

В методе простой однопараметрической итерации для решения СЛАУ Ax=b используется итерационный процесс $x^{q+1}=Cx^q+d$, где $C=E-\lambda A$, $d=\lambda b$, E - единичная матрица, $\lambda$ - параметр процесса. Вычисления заканчиваются, когда $\left\|x^{q+1}-x^q\right\|<\varepsilon$, где $\varepsilon$ - погрешность метода.

*Алгоритм 5*

Произведем декомпозицию области данных, как это показано на рис. 3.41.

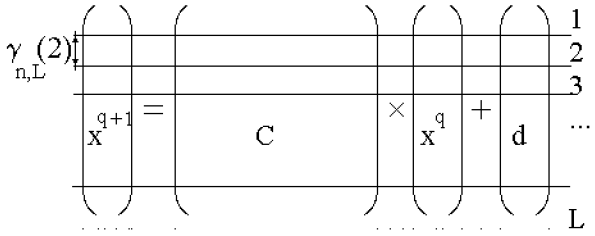



Рис. 3.41. Распределение матриц $x^{q+1}$, C, $x^q$ и d между задачами алгоритма 5

Каждая задача хранит $\gamma_{n,L}(l)$ элементов столбцов $x^{q+1}$, $x^q$, d и $\gamma_{n,L}(l)$ строк матрицы C с $r_l^l=\sum_{g=1}^{l-1}\gamma_{n,L}(g)+1$ по $r_l^l=\sum_{g=1}^{l}\gamma_{n,L}(g)$. Такая декомпозиция позволяет ограничиться пересылками только элементов $x^q$ [2].

На первом шаге алгоритма производится перемножение C на $x^q$. Задаче l следует отыскать элементы результирующего столбца с $r_l^l$ по $r_2^l$. Необходимые для этого строки матрицы C присутствуют в памяти задачи (рис. 3.41). Требуется полностью столбец $x^q$, имеется только его фрагмент ($r_l^l$, $r_2^l$). Организуем рассылку другим задачам (1,.., l-1, l+1,.., L) имеющегося фрагмента и прием от них недостающих частей $x^q$.



Второй шаг алгоритма не связан с пересылкой данных. Найденные на предыдущем шаге значения вектора $Cx^q$ (с $r_l^l$ по $r_2^l$) складываются с соответствующими значениями вектора d, которые присутствуют в памяти задачи l. Результат является фрагментом искомого вектора $x^{q+1}$.

На третьем шаге алгоритма проверяется критерий завершения итерационного процесса. Вычисление равномерной матричной нормы $\max_{r_l^l\le i\le r_2^l}\left|x_i^{q+1}-x_i^q\right|$ не требует дополнительных пересылок. Если хотя бы для одной задачи превышено допустимое значение погрешности, возвращаемся к первому шагу алгоритма. Иначе считаем найденное решение окончательным.

Использование нормы, предполагающей присутствие всех элементов $x^{q+1}$ в одной задаче (например, спектральной матричной нормы), не обязательно связано с дополнительными пересылками на третьем шаге. В этом случае проверку критерия следует осуществлять на первом шаге алгоритма, после получения каждой задачей вектора $x^q$ целиком. Тогда нас интересует норма разности $x^q$ и $x^{q-1}$.

Исследуем ускорение алгоритма 5:

$$s=\frac{n(n\tau_\times+(n-1)\tau_++\tau_+)}{\gamma_{n,L}(l)(n\tau_\times+(n-1)\tau_++\tau_+)+n\tau_k}=\frac{n}{\gamma_{n,L}(l)+k}\xrightarrow[L\to n]{}\frac{n}{1+k}\underset{k>>1}{\approx}\frac{n}{k}.$$

Вычисляя произведение матрицы на столбец, производим $n^2$ умножений и n(n-1) сложений. Сложение двух столбцов характеризуется n операциями. Для нахождения эффективности по вычислениям используется критерий $\delta$. С увеличением числа задач алгоритма объем пересылок одной задачи не изменяется, количество арифметических операций сокращается.

Стремясь увеличить ускорение алгоритма 5, нельзя перейти к двумерной декомпозиции области, столбец на клетки не разбивается. Нельзя организовать дублирование данных каждой задачей с целью уменьшения пересылок, значения вектора x изменяются другими задачами. Можно сменить глобальную топологию коммуникаций на

локальную. Применение правила “разделяй и властвуй” позволит реализовать алгоритм 5 на гиперкубе. В данном случае удобнее использовать не стандартный прием, а особенности структуры области данных, что позволит ограничиться самой простой сетевой конфигурацией - кольцом.

*Алгоритм 6*

Представим матрицу C в блочном виде:

$$C = \begin{pmatrix} \mathbf{c}_{11} & \mathbf{c}_{12} & \dots & \mathbf{c}_{1L} \\ \mathbf{c}_{21} & \mathbf{c}_{22} & \dots & \mathbf{c}_{2L} \\ \dots & \dots & \dots & \dots \\ \mathbf{c}_{L1} & \mathbf{c}_{L2} & \dots & \mathbf{c}_{LL} \end{pmatrix},$$

где $\mathbf{c}_{lj} = \begin{pmatrix} c_{r_1^l, r_1^j} & c_{r_1^l, r_1^j+1} & \dots & c_{r_1^l, r_2^j} \\ c_{r_1^l+1, r_1^j} & c_{r_1^l+1, r_1^j+1} & \dots & c_{r_1^l+1, r_2^j} \\ \dots & \dots & \dots & \dots \\ c_{r_2^l, r_1^j} & c_{r_2^l, r_1^j+1} & \dots & c_{r_2^l, r_2^j} \end{pmatrix}$.

Не меняя декомпозиции области полагаем, что l-я строка блочной матрицы C ($\mathbf{c}_{l1}, \mathbf{c}_{l2}, .., \mathbf{c}_{lL}$) принадлежит задаче l. Циклически сместим рассматриваемую строку на l-1 позицию влево:

$$C' = \begin{pmatrix} \mathbf{c}_{11} & \mathbf{c}_{12} & \dots & \mathbf{c}_{1L} \\ \mathbf{c}_{22} & \mathbf{c}_{23} & \dots & \mathbf{c}_{21} \\ \dots & \dots & \dots & \dots \\ \mathbf{c}_{LL} & \mathbf{c}_{L1} & \dots & \mathbf{c}_{LL-1} \end{pmatrix} = \begin{pmatrix} \mathbf{c}'_{11} & \mathbf{c}'_{12} & \dots & \mathbf{c}'_{1L} \\ \mathbf{c}'_{21} & \mathbf{c}'_{22} & \dots & \mathbf{c}'_{2L} \\ \dots & \dots & \dots & \dots \\ \mathbf{c}'_{L1} & \mathbf{c}'_{L2} & \dots & \mathbf{c}'_{LL} \end{pmatrix}.$$

На первом шаге нового алгоритма произведем вычисления по схеме:

for g=1 to L-1 do begin

$\mathbf{s}_l = \mathbf{s}_l + \mathbf{c}'_{lg} \times x^q_l$;

передача $x^q_l$ верхнему соседу; прием $x^q_l$ от соседа снизу (рис. 3.42);



end;

$\mathbf{s}_l = \mathbf{s}_l + \mathbf{c}'_{lg} \times x^q_l$,

где $x^q_l$ - фрагмент вектора $x^q$ с $r_1^l$ по $r_2^l$, принадлежащий l-й задаче на момент вычисления произведения $\mathbf{c}'_{lg}$ на $x^q_l$; $Cx^q=(\mathbf{s}^q_1, \mathbf{s}^q_2, \dots, \mathbf{s}^q_L)^T$.

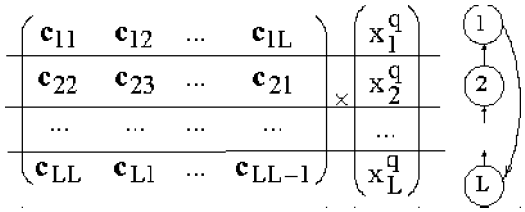



Рис. 3.42. Топология локальных коммуникаций алгоритма 6

Произведенное смещение блоков матрицы C обеспечивает корректное перемножение блока на соответствующий фрагмент столбца. Остальные шаги алгоритма 6 совпадают с шагами предыдущего алгоритма.

### *3.3.2.2. Параллельные алгоритмы метода Гаусса*

Вспомним простейшую вычислительную схему метода Гаусса. Пусть дана система уравнений

$$a_{11}x_1+a_{12}x_2+..+a_{1n}x_n=b_1;$$
$$a_{21}x_1+a_{22}x_2+..+a_{2n}x_n=b_2;$$
$$\dots\dots\dots\dots\dots\dots\dots$$
$$a_{n1}x_1+a_{n2}x_2+..+a_{nn}x_n=b_n.$$

Исключим из нее $x_1$. Для этого уравнение i делим на $a_{i1}$, затем вычитаем первое уравнение из всех остальных. Система принимает вид

$$x_1+a^1_{12}x_2+..+a^1_{1n}x_n=b^1_1;$$
$$a^1_{22}x_2+..+a^1_{2n}x_n=b^1_2;$$
$$\dots\dots\dots\dots\dots\dots\dots$$
$$a^1_{n2}x_2+..+a^1_{nn}x_n=b^1_n,$$

где $a^1_{1j}=a_{1j}/a_{11}$, $b^1_1=b_1/a_{11}$, $a^1_{ij}=a_{ij}/a_{i1}-a_{1j}/a_{11}$, $b^1_i=b_i/a_{i1}-b_1/a_{11}$ при $2 \le j \le n$.

Затем исключим $x_2$, для чего с уравнениями 2÷n преобразованной системы повторим предыдущую процедуру. И так далее, пока в результате прямого хода исключений Гаусса система не примет верхнетреугольный вид

$$x_1+a^1_{12}x_2+..+a^1_{1n}x_n=b^1_1;$$
$$x_2+..+a^2_{2n}x_n=b^2_2;$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$x_n=b^n_n.$$

Обратный ход метода характеризуется подстановкой вычисленных значений x в вышерасположенные уравнения [1].

*Алгоритм 7*

Произведем одномерную декомпозицию матрицы A [2] коэффициентов системы (рис. 3.43), предполагая, что n на L делится нацело. Столбец b распределяется между задачами аналогично.

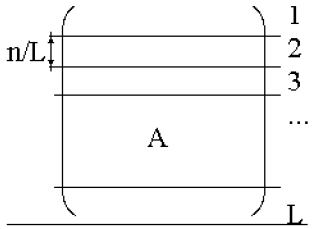



Рис. 3.43. Декомпозиция матрицы коэффициентов системы на горизонтальные полосы

Первая задача начинает процесс исключения, деля L/n первых уравнений системы на $a_{i1}$ ($1\le i\le n/L$), рассылая остальным задачам $a^1_{1j}$ и $b^1_1$, вычисляя затем $a^1_{ij}$ ($2\le j\le n$; $2\le i\le n/L$) и $b^1_i$. Задача под номером l ($2\le l\le L$) делит свои уравнения на $a_{i1}$ ($n(l-1)/L+1\le i\le nl/L$), принимает $a^1_{1j}$ и $b^1_1$, вычисляя $a^1_{ij}$ и $b^1_i$ ($2\le j\le n$). Аналогично происходят исключения $x_2$, $x_3$,… $x_{L/n}$. С увеличением индекса неизвестной первая задача алгоритма производит все меньше арифметических операций по сравнению с остальными задачами (количество уравнений, с которыми она работает, уменьшается). Начиная с $x_{L/n+1}$ первая задача перестает участвовать в вычислениях



по прямому ходу. Вторая задача делит L/n своих первых уравнений системы на $a^{L/n}_{i,L/n+1}$ ($n/L+1\le i\le 2n/L$), рассылая остальным задачам $a^{L/n+1}_{L/n+1,j}$ и $b^{L/n+1}_{L/n+1}$, вычисляя затем $a^{L/n+1}_{i,j}$ ($n/L+2\le j\le n$; $n/L+2\le i\le 2n/L$) и $b^{L/n+1}_i$. Теперь, с увеличением индекса неизвестной, вторая задача алгоритма производит все меньше арифметических операций. Начиная с исключения $x_{2L/n+1}$ она простаивает до конца прямого хода. В конце прямого хода задачи 1,2,3,…,L-1 простаивают в то время, когда последняя задача алгоритма производит исключение последних L/n-1 неизвестных системы. Во время прямого хода отправление данных производит задача с наименьшим номером (из работающих), остальные работающие задачи принимают посланные коэффициенты.

Обратный ход начинается рассылкой L-й задачей значения $x_n$, найденного из последнего уравнения системы, остальным задачам алгоритма. Приняв $x_n$, l-я задача умножает его на $a^i_{i,n}$ ($n/L+1\le i\le 2n/L$) и вычитает произведение из $b^{L/n+1}_i$, формируя новую правую часть уравнения i. Последняя задача в это время производит аналогичные операции над стоками $n(L-1)/L+1\le i\le n-1$. В дальнейшем, приняв значение $x_q$, l-я задача домножает его на коэффициент $a^i_{i,q}$ и вычитает из новой правой части уравнения i. С уменьшением индекса найденной неизвестной L-я задача производит все меньше арифметических операций по сравнению с остальными задачами. После нахождения $x_{n(L-1)/L+1}$ она не принимает участия в дальнейшей работе алгоритма. Рассылкой значения $x_{n(L-1)/L}$ занимается задача L-1. Теперь она начинает производить все меньше арифметических операций (с уменьшением индекса найденного значения x) по сравнению с остальными задачами. Во время обратного хода метода Гаусса очередность выхода задач из процесса вычислений противоположна очередности прямого хода. Отправление данных производит задача с наибольшим номером (из работающих), остальные работающие задачи принимают посланное значение x. Первая задача заканчивает обратный ход алгоритма вычислением $x_n$, остальные задачи в это

время уже простаивают. Алгоритм 7 - первый из изучаемых нами алгоритмов характеризуется простоями задач. Остановимся на его анализе подробнее.

Вычисляя ускорение параллельного алгоритма, подсчитаем количество арифметических операций в последовательном методе Гаусса. На i-м шаге прямого хода n+1-i уравнений делятся на $a_{qi}$ ($i \le q \le n$), причем в рассматриваемых уравнениях отличны от нуля n+1-i коэффициентов и правая часть. Следовательно, на i-м шаге производятся (n-i+1)(n-i+2) делений. Принимая вычитания за сложения, отметим, что на i-м шаге происходит (n+1-i) сложений в (n-i) уравнениях, всего (n-i)(n-i+1) сложений. Прямой ход состоит из n шагов ($1 \le i \le n$). Общее количество сложений прямого хода есть сумма ряда [5]:

$$S_1 = \sum_{i=1}^{n}(n-i)(n-i+1) = \sum_{i=1}^{n}\left(n^2+n\right) - (2n+1)\sum_{i=1}^{n} i + \sum_{i=1}^{n} i^2 =$$

$$= \left[\sum_{i=1}^{n} i = \frac{n(n+1)}{2};\ \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}\right] =$$

$$= n^2(n+1) - \frac{n(n+1)(2n+1)}{2} + \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n\left(n^2-1\right).$$

Количество делений

$$S_2 = \sum_{i=1}^{n}(n-i+1)(n-i+2) = \frac{1}{3}n\left(n^2+3n+2\right).$$

Обратный ход тоже состоит из n шагов, на i-м шаге производится (i-1) умножение и столько же делений. Общее количество сложений обратного хода

$$S_3 = \sum_{i=1}^{n}(i-1) = \frac{n(n-1)}{2} - n.$$

Тогда общая длительность вычислений по последовательному методу составляет

$$(S_1+S_3)\tau_+ + S_3\tau_\times + S_2\tau_\div.$$



Изучая прямой ход параллельного алгоритма, подсчитаем количество арифметических операций, произведенных одновременно несколькими задачами. Все L задач одновременно участвовали в исключении первых n/L неизвестных системы, осуществив параллельно $\sum_{i=1}^{n/L}\frac{(n-i)(n-i+1)}{L}$ сложений и $\sum_{i=1}^{n/L}\frac{(n-i+1)(n-i+2)}{L}$ делений. Следующие n/L неизвестных исключали уже L-1 задач, произведя одновременно $\sum_{i=n/L+1}^{2n/L}\frac{(n-i)(n-i+1)}{L-1}$ сложений и $\sum_{i=n/L+1}^{2n/L}\frac{(n-i+1)(n-i+2)}{L-1}$ делений. И так далее. Предпоследние n/L исключений приходятся на две задачи: $\sum_{i=n-2n/L+1}^{n-n/L}\frac{(n-i)(n-i+1)}{2}$ сложений и $\sum_{i=n-2n/L+1}^{n/L}\frac{(n-i+1)(n-i+2)}{2}$ делений. Последние исключения делаются L-й задачей в одиночестве при помощи $\sum_{i=n-n/L+1}^{n}(n-i)(n-i+1)$ сложений и $\sum_{i=n-n/L+1}^{n}(n-i+1)(n-i+2)$ делений. Сложив все суммы, получим оценку снизу количества операций прямого хода параллельного алгоритма. Неточность подсчета объясняется неравномерной загруженностью работающих задач. Когда мы делили количество операций на число исполняющих их задач, то не учитывали, что задача с меньшим номером работала с меньшим количеством уравнений, чем остальные непростаивающие задачи алгоритма.

Оценивая количество операций точно, обратимся к анализу работы последней задачи алгоритма, которая во время всего прямого хода не простаивает. Первые n-n/L исключений она, а следовательно и весь алгоритм, произведет с помощью $S_4 = \sum_{i=1}^{n-n/L}\frac{n}{L}(n-i+1)$ опера-

ций сложения и $S_5 = \sum_{i=1}^{n-n/L} \frac{n}{L}(n-i+2)$ деления над n/L уравнениями.

Заключительные вычисления проводятся L-й задачей без участия остальных (фактически это последовательный фрагмент параллельного алгоритма, анализируя только его, можно грубо оценить сверху ускорение прямого хода по закону Амдала) - $S_6 = \sum_{i=n-n/L+1}^{n}(n-i)(n-i+1)$ сложений и $S_7 = \sum_{i=n-n/L+1}^{n}(n-i+1)(n-i+2)$ делений. Тот же метод подсчета используем при анализе обратного хода параллельного алгоритма. Во время всего обратного хода не будет простаивать первая задача. За весь обратный ход она произведет $S_8 = \sum_{i=n-n/L+1}^{n}(i-1)$ операций сложения и столько же деления над первыми n/L уравнениями системы. Тогда общее время арифметических операций параллельного алгоритма есть $(S_4+S_6+S_8)\tau_+ + (S_5+S_7)\tau_\div + S_8\tau_\times$.

Говоря о коммуникациях алгоритма 7, предположим, что объем любого сообщения не превосходит размера пакета. При выводе ускорений алгоритмов 2,4,5,6 подразумевалось обратное, время пересылки считалось зависящим исключительно от объема пересылаемых данных. Здесь, независимо от размера посылки, длительность ее передачи принимается равной $\tau_П$. Это упрощает подсчет времени пересылок, так как в алгоритме предполагается обмен между задачами разными данными, разного объема, на разных шагах. Обмены обратного хода алгоритма описываются изложенной моделью точно, они связаны с пересылкой всего одного числа. Анализируя пересылки прямого хода при n<50000 для современных сетей (например Mirinet), можно утверждать, что наше предположение не внесет в расчеты никакой погрешности. В любом случае, задавшись достаточно большой величиной $\tau_П$, получим оценку ускорения сверху. В рамках изложенной модели вычисления по прямому и обратному ходу алгоритма сопровождаются пересылками общей длительностью $(n-n/L)\tau_П + (n-n/L)\tau_П$ - время приема данных первой и последней задачами.



Запись окончательного выражения для ускорения алгоритма 7 предоставим читателю, все необходимые для этого формулы выведены.

Алгоритму 7 присущи два недостатка: значительные простои задач и глобальность коммуникаций. Используя разбиение матрицы A на горизонтальные циклические полосы шириной в одно уравнение, сведем простои к минимуму.

*Алгоритм 8*

Произведем заново распределение строк матрицы A [2] между задачами алгоритма так, как это показано на рис. 3.44. Строки матрицы, принадлежащие одной задаче, разделены L-1 строками других задач. При таком разбиении простои начинаются не после n/L исключений, как в предыдущем алгоритме, а после n-L+1 исключений, то есть уже в конце прямого хода. Эти рассуждения верны и для обратного хода алгоритма. При n>>L простоями можно пренебречь, тем более что заключительные шаги прямого хода предполагают значительно меньшее количество арифметических операций, чем все остальные шаги.

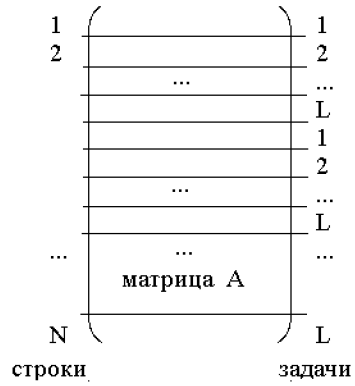



Рис. 3.44. Декомпозиция матрицы A на циклические полосы шириной в одну строку

Стремясь к точности, учтем простои алгоритма. До n-L+2 шага прямого хода все задачи принимали участие в вычислениях.

Следовательно, n-L+1 исключение связано с $\sum_{i=1}^{n-L+1}\frac{(n-i)(n-i+1)}{L}$ сложениями и $\sum_{i=1}^{n-L+1}\frac{(n-i+1)(n-i+2)}{L}$ делениями. Говорим об этих суммах, как об оценке снизу, ведь задачи не сбалансированы по вычислениям. Некоторая задача из L (в рассматриваемом случае одна и только одна) может работать с количеством строк на одну меньше, чем остальные задачи алгоритма. По сравнению с аналогичным подходом, применяемым в анализе предыдущего алгоритма, погрешность оценки снижена в n/L-1 раз и составляет $O(n^2)$ при общем количестве операций прямого хода $O(n^3)$. Такой погрешностью можно пренебречь. Последние L-1 исключений прямого хода выполняют не все задачи. Обратим внимание на задачу L, которая не простаивает. На этом этапе она произведет (точно) $\sum_{i=n-L+2}^{n}(n-i+1)$ операций сложения и $\sum_{i=n-L+2}^{n}(n-i+2)$ деления над последним уравнением системы.

Аналогично анализируем обратный ход алгоритма. До n-L+2 шага все задачи одновременно исполнят $\sum_{i=1}^{n-L+1}\frac{(i-1)}{L}$ сложений и столько же умножений. Неточность этой оценки вызвана несбалансированностью задач, погрешность составит O(n), по сравнению с $O(n^2)$ - общим количеством операций обратного хода - ею можно пренебречь. Начиная с шага n-L+2 ориентируемся на первую задачу, которая не простаивает. Для нее получим (точно) $\sum_{i=n-L+2}^{n}(i-1)$ операций сложения и столько же операций деления над первым уравнением системы.

По сравнению с предыдущим алгоритмом объем пересылок увеличился. Теперь их длительность оценивается величиной $2(n-1)\tau_п$. Читателю предлагается самостоятельно найти такие значения параметров n, L, $\tau_\times$, $\tau_{\times+}$, $\tau_\div$ и $\tau_п$, при которых ускорение алгоритма 8 превзойдет ускорение алгоритма 7.



Разбиение области данных на циклические фрагменты является стандартным приемом борьбы с несбалансированностью задач, вызванной простоями. При этом приходится мириться с возрастанием длительности пересылок. В каждом отдельном случае выгодность циклической декомпозиции следует рассматривать особо.

## 3.4. Параллельные алгоритмы решения разностных уравнений

В настоящем параграфе излагается методика построения параллельных алгоритмов решения разностных уравнений простейших явной и неявной схем. Для определенности рассматриваются схемы, применяемые при численном решении нестационарного линейного однородного уравнения теплопроводности $\frac{\partial U}{\partial t}-a\Delta U=0$, где $U=U(\bar{x},t)$ - температура среды, характеризующейся температуропроводностью a; t – время; $\bar{x}$ - пространственные координаты. На границе области на функцию U налагаются условия первого рода.

Описание алгоритмов ограничивается пояснением расчета сеточных функций на одном временном слое. Вычисления на других слоях полностью аналогичны. Все характеристики параллельных алгоритмов расчета на одном слое верны для произвольного количества слоев, так как длительность вычислений $t^q$ на q слоях линейно зависит от числа слоев и времени расчета сеточных функций на одном слое $t^1(t^q = q?t^1)$.

Представленные в параграфе алгоритмы могут использоваться для решения различных уравнений математической физики методами конечных разностей и Галеркина.

### *3.4.1. Параллельный алгоритм для явных разностных схем, одномерный случай*

Решая одномерное (по пространству) уравнение теплопроводности $\frac{\partial U}{\partial t}-a\frac{\partial^2 U}{\partial x^2}=0$, используем простейшую явную схему

$\frac{U_i^{n+1}-U_i^n}{h_t}-a\frac{U_{i-1}^n-2U_i^n+U_{i+1}^n}{h_x^2}=0$, записанную на равномерной сетке $\omega^1=\{(t_k,x_i),\ t_k=kh_t,\ k=0,\ 1,\ \ldots,\ K=T/h_t,\ x_i=ih_x,\ i=1,\ \ldots,\ I=L/h_x+1\}$ [8]. Такая схема позволяет выразить значение сеточной функции $U_i^{n+1}$ на новом временном слое n+1 через соседние функции $U_{i-1}^n, U_i^n, U_{i+1}^n$ на предыдущем слое n в явном виде:

$$U_i^{n+1}=a\frac{h_t}{h_x^2}\left(U_{i-1}^n-2U_i^n+U_{i+1}^n\right)+U_i^n. \qquad (3.13)$$

*Алгоритм 9*

Построение параллельного алгоритма для вычислений по (3.13) на $\omega^1$ начнем с декомпозиции сеточной области (рис. 3.45, б) [12].

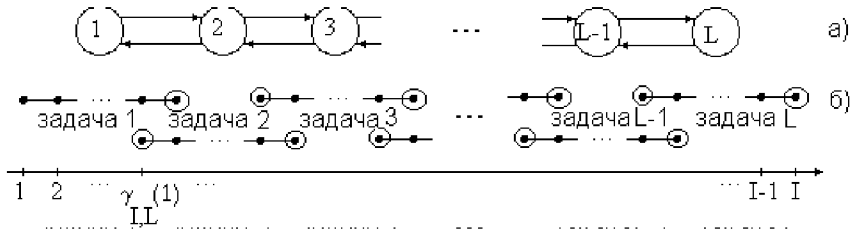



Рис. 3.45. Коммуникации алгоритма 9 (а) и разбиение сеточной области (б)

Кружком обведены узлы сетки, в которых сеточная функция не вычисляется, но используется

Задаче l (2≤l≤L-1) параллельного алгоритма принадлежит $\gamma_{LL}(l)+2$ значений сеточных функций и узлов сетки, ограниченных фрагментом области $\omega^1$ с индексами $r_1^l-1, r_2^l+1$. Первая задача содержит $\gamma_{LL}(l)+1$ значений в пределах $1\le i\le r_2^l+1$, последняя задача столько же значений в пределах $r_1^L-1\le i\le I$.

На первом шаге алгоритма задача l (1≤l≤L) вычисляет значения $U_i^{n+1}$ для $r_1^l\le i\le r_2^l$ по формуле (3.13). При расчете $U_{r_1^l}^{n+1}$ все задачи,



кроме первой, используют значение $U_{r_1^l-1}^n$, для хранения которого дублируется предпоследний узел задачи l-1. Вычисляя $U_{r_2^l}^{n+1}$, все задачи, кроме последней, используют значение $U_{r_2^l+1}^n$, для хранения которого дублируется второй узел задачи l+1. Поэтому на рис 3.46, б перекрываются фрагменты соседних подобластей $\omega^1$, принадлежащие разным задачам. Первая и последняя задачи не нуждаются в значениях $U_0^n$ и $U_{I+1}^n$, их подобласти меньше на один узел.

Второй шаг алгоритма связан с его коммуникациями (рис. 3.46,а). Каждой задаче l (2≤l≤L-1) на шаге n+2 понадобятся значения $U_{r_2^l+1}^{n+1}, U_{r_1^l-1}^{n+1}$, которые следует загодя получить от задач l+1, l-1.

Первая задача заинтересована в получении $U_{r_2^l+1}^{n+1}$ от задачи 2, последняя в получении $U_{r_1^l-1}^{n+1}$ от задачи L-1. Пересылку данных осуществим в два приема. Сначала задачи l (1≤l≤L-1) посылают правым соседям значения $U_{r_2^l}^{n+1}$ и принимают (кроме первой задачи) от левых соседей $U_{r_1^l-1}^{n+1}$. Задача L принимает от L-1 $U_{r_1^L-1}^{n+1}$. Затем задачи l (2≤l≤L) посылают левым соседям значения $U_{r_1^l}^{n+1}$ и принимают (кроме последней задачи) от правых соседей $U_{r_2^l+1}^{n+1}$.

Первая задача принимает от второй $U_{r_2^l+1}^{n+1}$.

Такой порядок пересылок позволяет реализовывать вычисления по алгоритму не только на кластерах, но и на векторных и конвейерных ЭВМ, для которых пересылки данных производятся всеми задачами одновременно, в одном направлении и только соседям (левому или правому). Однотипность вычислений в каждой задаче и наличие всего двух соседей (для первой и последней задач сосед один) позволяют отнести алгоритм 9 к классу векторных и конвейерных алгоритмов. Под собственно параллельными алгоритмами в литературе обычно понимают исключительно те алгоритмы, вычисления по которым реализуются на ЭВМ с кластерной архитектурой.

Оценивая ускорение алгоритма, длительность арифметических операций по нему положим равной $\gamma_{I,L}(l)\tau_1$, где $\tau_1=3\tau_+ +\tau_\times$ - время расчета одной сеточной функции по формуле (3.13). Коэффициент $a\frac{h_t}{h_x^2}$ вычисляется в начале алгоритма и далее используется как константа, поэтому операции деления и умножения, с ним связанные, в $\tau_1$ не входят. При смене дифференциального шаблона будем подставлять в выражение $\gamma_{I,L}(l)\tau_1$ другое значение $\tau_1$. Например, для шаблонов на рис. 3.46 $\tau_1$ увеличится, $\gamma_{I,L}(l)$ останется прежним.

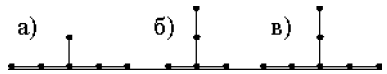



Рис. 3.46. Различные дифференциальные шаблоны для построения разностных схем

Для выражения времени пересылок в алгоритме 9 воспользуемся моделью, предложенной при описании алгоритма 8. Оценка, полученная с помощью этой модели, будет лишена погрешности. В рассматриваемом алгоритме все пересылки связаны с передачей или приемом одного числа (значения сеточной функции), следовательно, длительность одного приема или передачи равна $\tau_п$. Время всех пересылок алгоритма равно совокупному времени передач и приемов одной задачи l ($2\le l\le L-1$) и составит $4\tau_п$ (два приема и две передачи). Применяя алгоритм 9 для решения разностных уравнений, составленных по шаблонам на рис. 3.46, а, в получим длительность пересылок, равную $8\tau_п$, по шаблону на рис. 3.46, б - $4\tau_п$.



Ускорение алгоритма 9 примем равным

$$S=\frac{I\times\tau_1}{\gamma_{I,L}(l)\times\tau_1+4\tau_п}=\frac{I}{\gamma_{I,L}(l)+4k_1}\xrightarrow[L\to I]{}\frac{I}{1+4k_1}\underset{k_1>>1}{\approx}\frac{I}{4k_1},$$

где $k_1=\frac{\tau_п}{\tau_1}$.

С увеличением количества задач алгоритма объем пересылок остается постоянным, длительность вычислений снижается. Следовательно, критерием эффективности по времени следует считать критерий $\delta$.

### *3.4.2. Параллельные алгоритмы для явных разностных схем, двумерный случай*

Перейдем к рассмотрению двумерного уравнения теплопроводности

$$\frac{\partial U}{\partial t}-a\left(\frac{\partial^2 U}{\partial x^2}+\frac{\partial^2 U}{\partial y^2}\right)=0$$

и простейшей явной схемы для него

$$\frac{U_{i,j}^{n+1}-U_{i,j}^{n}}{h_t}-a\left(\frac{U_{i-1,j}^{n}-2U_{i,j}^{n}+U_{i+1,j}^{n}}{h_x^2}+\frac{U_{i,j-1}^{n}-2U_{i,j}^{n}+U_{i,j+1}^{n}}{h_y^2}\right)=0 \qquad (3.14)$$

на декартовой прямоугольной сетке $\omega^2=\{(t_k,x_i,y_j)$, $t_k=kh_t$, $k=0, 1, \ldots, K=T/h_t$, $x_i=ih_x$, $i=1, \ldots, I=L_x/h_x+1$, $y_j=jh_y$, $j=1, \ldots, J=L_y/h_y+1\}$, наложенной на область D, имеющую форму прямоугольника со сторонами $L_x$, $L_y$.

Одномерная и двумерная декомпозиции $\omega^2$ уже сравнивались в первом параграфе настоящей главы. Выбор декомпозиции в конечном счете определяется выбором модели передачи сообщений. При пересылке небольшого количества информации пакетами большого объема (пакетная модель) предпочтительнее одномерная декомпозиция области. Если же полагать длительность пересылки линейно зависящей от объема пересылаемых данных (линейная модель), то для квадратной сеточной области выгоднее двумерная декомпозиция.

В алгоритмах матричного перемножения пересылались целиком подобласти, принадлежащие задачам, - там мы придерживались линейной модели. Здесь пересылаются только границы подобластей, поэтому принимаем пакетную модель. Пытливый читатель самостоятельно повторит вывод ускорений алгоритмов для линейной модели передачи сообщений.

*Алгоритм 10*

Рассмотрим одномерную декомпозицию сеточной области $\omega^2$ (рис. 3.47).

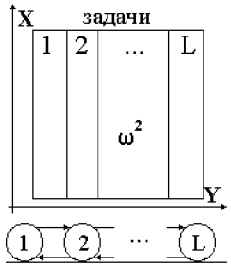



Рис. 3.47. Одномерная декомпозиция сеточной области $\omega^2$ с разбиением по оси Y и топология коммуникаций "линия" алгоритма 10, соответствующая такому разбиению

Используем вычислительную схему алгоритма 9 для построения нового параллельного алгоритма. На первом шаге алгоритма 9 вместо формулы (3.13) применим (3.14). Тогда для вычисления сеточной функции на новом временном слое потребуются значения

$$U^n_{i-1,j}, U^n_{i,j}, U^n_{i+1,j}, U^n_{i,j-1}, U^n_{i,j+1}.$$

for i=1 to I do begin

алгоритм 9 для строки i по формуле (3.14);

end.

Все задачи алгоритма 9 одновременно вычисляют значения сеточных функций на новом временном слое при i=const; 1≤j≤J. Если I>J, то выгоднее разбиение вдоль оси X, в этом случае задачи алгоритма 9 будут вычислять сеточные функции при j=const; 1≤i≤I.



Ускорение такого алгоритма составит

$$S = \frac{I \times J \times \tau_1}{\gamma_{J,L}(1) \times I \times \tau_1 + 4\tau_п I},$$

где 4I - количество принятых и отправленных задачей l (2≤l≤L-1) пакетов. Ускорение представленного алгоритма равно ускорению алгоритма 9.

При модификации предложенного алгоритма воспользуемся тем, что пакеты отправляются по каналам почти пустыми, в них находится всего по одному числу. Сократим количество пакетов за счет их более рационального наполнения. Если I мало настолько, что I значений сеточной функции поместятся в один пакет, то целесообразно использовать другую вычислительную схему.

for i=1 to I do begin

первый шаг алгоритма 9 для строки i по формуле (3.14);

end.

Затем задача l (1≤l≤L-1) передает одним пакетом соседу справа вектор значений $U^{n+1}_{i,j}$, в котором $j = r^l_2$, 1≤i≤I. Задача l (2≤l≤L) принимает от соседа слева вектор с индексами $j = r^l_1 - 1$, 1≤i≤I. Аналогично производится передача векторов от правых соседей - левым. При этом длительность пересылок алгоритма снижается в I раз.

Ускорение

$$S = \frac{I \times J \times \tau_1}{\gamma_{J,L}(1) \times I \times \tau_1 + 4\tau_п} = \frac{J \times \tau_1}{\gamma_{J,L}(1) \times \tau_1 + \frac{4}{I}\tau_п} \xrightarrow[L \to J]{} \frac{J}{1 + \frac{4}{I}k_1}$$

существенно возрастает. В случае, когда I значений сеточной функции в один пакет не помещаются, изложенную модификацию алгоритма 10 также следует применять. Длительность пересылок по сравнению с первым вариантом алгоритма сократится в $I/I_п$ раз, где $I_п$ - максимальное количество значений сеточной функции, помещающихся в один пакет.

*Алгоритм 11*

Двумерная декомпозиция сеточной области представлена на рис. 3.48.

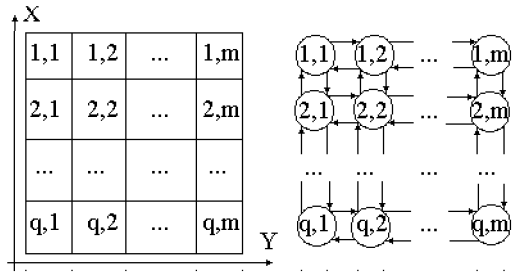



Рис. 3.48. Двумерная декомпозиция $\omega^2$ и топология коммуникаций "решетка", соответствующая такому разбиению

Задачи алгоритма 11 нумеруются двумя индексами $l_x$,$l_y$, как блоки матрицы. L=m×q.

Алгоритм 11, как и алгоритм 9, состоит из двух шагов: вычислительного и связанного с пересылкой сообщений. На первом шаге задача $l_x$,$l_y$ ($1 \le l_x \le q$, $1 \le l_y \le m$) находит значения $U_{i,j}^{n+1}$ для $r_1^{l_x} \le l_x \le r_2^{l_x}$, $r_1^{l_y} \le l_y \le r_2^{l_y}$. Второй шаг характеризуется пересылкой векторов сначала по направлению X (сверху вниз и снизу вверх), затем по направлению Y (слева направо и справа налево).

Наибольший объем данных пересылают задачи $l_x$,$l_y$ ($2 \le l_x \le q-1$, $2 \le l_y \le m-1$). Каждая из них передаст и примет от четырех соседей 8 пакетов, если $\max\{\gamma_{I,q}(1),\gamma_{I,m}(1)\} \le I_п$. Ускорение такого алгоритма

$$S = \frac{I \times J \times \tau_1}{\gamma_{I,q}(1) \times \gamma_{J,m}(1) \times \tau_1 + 8\tau_п}.$$

При равном числе задач эффективность по времени алгоритма 10 вдвое превысит эффективность по времени алгоритма 11. Исключение составляет случай, когда min{m,q}=2. Например, четырехзадачный алгоритм 11 по эффективности δ не уступает алгоритму 10, топология его коммуникаций соответствует четырех-



процессорному кольцу. Интересно, что фактическое количество значений сеточных функций, пересылаемое одной задачей такого алгоритма, составит всего половину от аналогичной характеристики четырехзадачного алгоритма 10, но в рамках пакетной модели длительность этих пересылок окажется одинаковой.

В общем случае использование алгоритма 10 предпочтительнее. Кроме меньшего объема передач он отличается простотой реализации, довольствуясь менее громоздкой сетевой конфигурацией.

Вкратце коснемся декомпозиции круговой области D. Сравним на прямоугольной сетке два варианта одномерного разбиения и вариант двумерного (рис. 3.49).

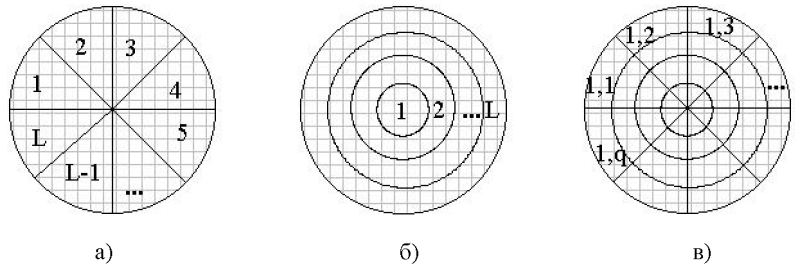



Рис. 3.49. Декомпозиции круговой области D с наложенной на нее прямоугольной сеткой: а - одномерная декомпозиция с разбиением по углу φ; б - одномерная декомпозиция с разбиением по радиусу ρ; в - двумерная декомпозиция с разбиениями по углу и радиусу

Одномерное разбиение по углу (рис. 3.49, а) удачно тем, что при равномерном шаге по φ нарезаются сектора с одинаковым количеством узлов сетки. Запись алгоритма с такой декомпозицией отличается простотой: все задачи производят одинаковое количество арифметических операций, обмениваясь равным количеством значений сеточных функций. К недостатку алгоритма следует отнести сложную топологию коммуникаций, связанную с вычислениями в центральных узлах сетки. Для нахождения сеточных функций в таких узлах могут потребоваться обмены не только с соседями по часовой стрелке и против нее, но и с другими задачами (рис. 3.50). При реализации такого алгоритма нельзя ограничиться использованием сетевой конфигурации "кольцо".

Этого недостатка лишен алгоритм, использующий одномерную декомпозицию с разбиением по радиусу (рис. 3.49, б). Все центральные узлы сеточной области принадлежат первой задаче такого алгоритма. Правда, при равномерном разбиении по ρ количество узлов сетки для каждой задачи окажется различным. Положение спасает неравномерное разбиение по радиусу.

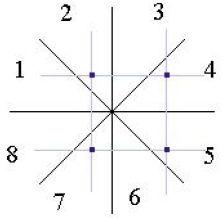


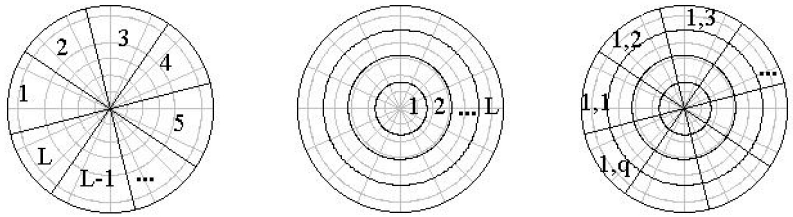

Рис. 3.50. Особенность вычисления функций в центральных узлах сетки при одномерной декомпозиции по углу

Для задачи 7, кроме каналов, связывающих ее с задачами 8 и 6, необходима установка дополнительных каналов с задачами 2 и 5.

Количество пересылаемых значений сеточной функции варьируется в зависимости от номера задачи, но в принятой нами пакетной модели при известных ограничениях это несущественно.

Алгоритм с двумерной декомпозицией на рис. 3.49, в характеризуется недостатками предыдущих алгоритмов, без их достоинств. К тому же его эффективность по времени окажется вдвое меньше.

При решении задач математической физики на круговую область обычно накладывают криволинейную сетку (рис. 3.51).




а) б) в)

Рис. 3.51. Декомпозиции круговой области D с наложенной на нее криволинейной сеткой: а - одномерная декомпозиция с разбиением по углу φ; б - одномерная декомпозиция с разбиением по радиусу ρ; в - двумерная декомпозиция с разбиениями по углу и радиусу



При одномерном разбиении по углу (рис. 3.51, а), проблема центральной точки становится особенно острой. Коммуникации такого алгоритма глобальны. Появляются простои задач, связанные с необходимостью решать сеточное уравнение для центрального узла в одной определенной задаче.

Коммуникации алгоритма с разбиением по радиусу (рис. 3.51, а) останутся локальными. Отпадет необходимость в неравномерном разбиении по ρ для равномерной сеточной области. Границы подобластей разных задач содержат одинаковое количество узлов. Если по ρ сетка гуще, чем по φ, то для линейной модели пересылки сообщений разбиение по радиусу позволит сократить время обменов по сравнению с разбиением по углу.

Двумерное разбиение на рис. 3.51,в будет полезным исключительно в рамках линейной модели, превосходя (на сетках с одинаковой дискретизацией по φ и по ρ) по эффективности оба варианта одномерного разбиения.

### *3.4.3. Параллельные алгоритмы для неявных разностных схем, одномерный случай*

Явные схемы, несмотря на простоту реализации, редко используются в силу жестких ограничений, налагаемых на шаги дискретизации для соблюдения устойчивости. Неявные схемы, как правило, лишены этого недостатка. Вычисления по ним сводятся к решению СЛАУ ленточного вида.

Уравнения неявной разностной схемы, построенной по четырехточечному дифференциальному шаблону (одномерный случай), на сетке $\omega^1=\{(t_k,x_i),\ t_k=kh_t,\ k=0,\ 1,\ldots,\ K=T/h_t,\ x_i=ih_x,\ i=0,\ 1,\ldots,\ I=L/h_x\}$ можно представить в виде

$$
\begin{aligned}
&c_0u_0 - b_0u_1 = f_0,\ i=0;\\
&-a_iu_{i-1} + c_iu_i - b_iu_{i+1} = f_i,\ 1\le i\le I-1;\\
&-a_Iu_{I-1} + c_Iu_I = f_I,\ i=I.
\end{aligned}
\qquad (3.15)
$$

Читателю хорошо известен метод правой прогонки [9], применяемый для решения этой системы и основанный на процедуре исключения Гаусса.

Во время прямого хода метода вычисляются прогоночные коэффициенты

$$\alpha_{i+1} = \frac{b_i}{c_i - a_i\alpha_i};\ i=1,2,..,I-1;\ \alpha_1 = \frac{b_0}{c_0};$$

$$\beta_{i+1} = \frac{f_i + a_i\beta_i}{c_i - a_i\alpha_i};\ i=1,2,..,I;\ \beta_1 = \frac{f_0}{c_0}.$$

Обратный ход прогонки производится по формулам $u_i = \alpha_{i+1}u_{i+1} + \beta_{i+1}$; i=I-1, I-2, .., 1; $u_I = \beta_{I+1}$, выражающим искомые значения сеточных функций.

Реализация таких вычислений требует применения 3I операции умножения, (2I+1) операции деления и 3I сложения (если не делать различий между сложениями и вычитаниями). Длительность вычислений по последовательному алгоритму составит $\tau_{ap}(I)=3I\tau_{\times}+(2I+1)\tau_{\div}+3I\tau_{+}$.

Изучая возможность распараллеливания алгоритма правой прогонки, произведем разбиение сеточной области (рис. 3.52) на подобласти

$\omega_1^1=\{(t_k,x_i), t_k=kh_t, k=0, 1, \ldots, K=T/h_t, x_i=ih_x, i=0, 1, \ldots, \gamma_{I+1,2}(1)-1\}$,

$\omega_2^1=\{(t_k,x_i), t_k=kh_t, k=0, 1,\ldots, K=T/h_t, x_i=ih_x, i=\gamma_{I+1,2}(1), \gamma_{I+1,2}(1)+1,\ldots, I\}$,

принадлежащие двум задачам параллельного алгоритма.

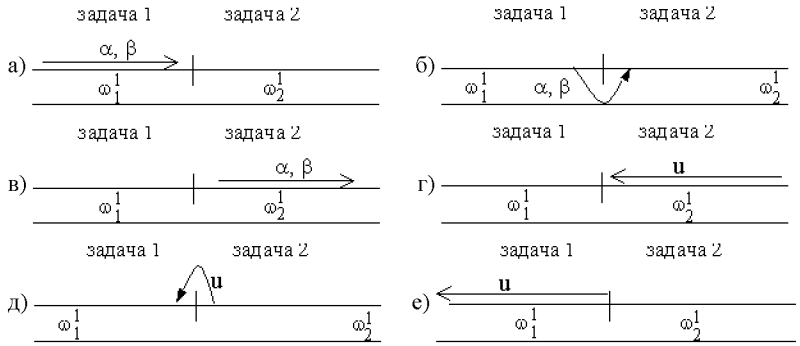



Рис. 3.52. Этапы вычислений по двухзадачному параллельному алгоритму, реализующему метод правой прогонки



На первом шаге алгоритма (рис. 3.52, а) задача 1 вычисляет прогоночные коэффициенты на подобласти $\omega_1^1$, задача 2 простаивает. Для начала работы ей необходимы коэффициенты $\alpha_{\gamma_{I+1,2}(1)}$, $\beta_{\gamma_{I+1,2}(1)}$, которые найдет первая задача в конце этого шага. Второй шаг (рис. 3.52, б) - передача необходимых коэффициентов от первой задачи второй. На третьем шаге (рис. 3.52, в) вторая задача продолжает прямой ход прогонки на подобласти $\omega_2^1$. Теперь простаивает первая задача в ожидании значения $u_{\gamma_{I+1,2}(1)}$, без которого она не может начать обратный ход прогонки на $\omega_1^1$. Простаивает первая задача и на четвертом шаге алгоритма (рис. 3.52, г), во время которого вторая задача осуществляет обратный ход прогонки на $\omega_2^1$. Пятый шаг (рис. 3.52, д) - передача необходимого первой задаче значения сеточной функции. Первая задача завершает обратный ход прогонки на последнем шаге алгоритма (рис. 5.52, е). Вторая задача в это время простаивает, все необходимые вычисления ею произведены на предыдущих шагах.

Итак, в предложенном алгоритме в любой момент времени, не связанный с передачей сообщений, простаивала или первая, или вторая задача. Следовательно, все арифметические операции производились последовательно, время работы алгоритма составляет $\tau_{ap}(I)+2\tau_{п}$.

Ускорения ($S = \frac{\tau_{ap}(I)}{\tau_{ap}(I) + 2\tau_{п}} < 1$) по сравнению с последовательным алгоритмом достигнуто не было. Более того, записанный параллельный алгоритм всегда будет работать медленнее, так как помимо арифметических операций содержит пересылки. Добавление любого количества задач к существующим лишь увеличит длительность пересылок до $2(L-1)\tau_{п}$.

Обратимся к методу встречных прогонок [9], также используемому для решения системы (3.15). Он применяется в случае, когда исследователю необходимо отыскать лишь часть неизвестных $u_i$.

Например, определяя $u_g$ ($2 \le g \le I-1$), достаточно вычислить коэффициенты прямого хода:

$$\alpha_{i+1} = \frac{b_i}{c_i - a_i\alpha_i};\ i=1,2,..,g-1;\ \alpha_1 = \frac{b_0}{c_0};\ \beta_{i+1} = \frac{f_i + a_i\beta_i}{c_i - a_i\alpha_i};\ i=1,2,..,g;\ \beta_1 = \frac{f_0}{c_0}$$

(совпадающие с коэффициентами прямого хода правой прогонки до узла g);

$$\zeta_i = \frac{a_i}{c_i - b_i\zeta_{i+1}};\ i=I-1, I-2, .., g;\ \zeta_I = \frac{a_I}{c_I};\ \eta_{i+1} = \frac{f_i + b_i\eta_{i+1}}{c_i - a_i\zeta_{i+1}};\ i=I-1, I-2, .., g;\ \eta_I = \frac{f_I}{c_I}.$$

Тогда $u_g = \frac{\eta_g + \zeta_g\beta_g}{1 - \zeta_g\alpha_g}$.

При необходимости остальные неизвестные находят по формулам $u_i = \alpha_{i+1}u_{i+1} + \beta_{i+1}$; $i=g-1, g-2, .., 0$; $u_{i+1} = \zeta_{i+1}u_i + \eta_{i+1}$; $i=g, g+1, .., I-1$. Длительность вычислений по методу встречных прогонок тоже составит $\tau_{ap}(I)$.

При построении параллельного алгоритма используем несколько измененный последовательный метод. Для симметрии $u_{g+1}$ будем рассчитывать не по формуле $u_{g+1} = \zeta_{g+1}u_g + \eta_{g+1}$, а как $u_g = \frac{\eta_{g+1} + \zeta_{g+1}\beta_{g+1}}{1 - \zeta_{g+1}\alpha_{g+1}}$, предварительно найдя $\alpha_{g+1} = \frac{b_g}{c_g - a_g\alpha_g}$ и $\beta_{g+1} = \frac{f_g + a_g\beta_g}{c_g - a_g\alpha_g}$. Дополнительно придется реализовать 4 умножения, 3 деления и 4 сложения, которыми, при достаточно большом I можно пренебречь, полагая $\tau_{ap}(I)$ прежней.

*Алгоритм 12*

Остановим свой выбор на функциональной декомпозиции. Пусть первая задача вычисляет прогоночные коэффициенты $\alpha,\beta$ и $u_i$ при $1 \le i \le g$, вторая $\eta,\zeta$ и $u_i$ при $g+1 \le i \le I$. Соответственно разобьем сеточную область $\omega^1$ на две подобласти: $\omega_1^1 = \{(t_k,x_i),\ t_k = kh_t,\ k=0, 1, \ldots, K=T/h_t,\ x_i = ih_x,\ i=0, 1, \ldots, g\}$ и $\omega_2^1 = \{(t_k,x_i),\ t_k = kh_t,\ k=0, 1, \ldots, K=T/h_t,\ x_i = ih_x,\ i=g+1, g+2, \ldots, I\}$, положив $g = \gamma_{I+1,2}(1)-1$.



Вычисления по алгоритму 12 разделим на три шага, соответствующие прямому ходу встречных прогонок (рис. 3.53, а), обратному ходу (рис. 3.53, в) и обмену данными между задачами (рис. 3.53, б).

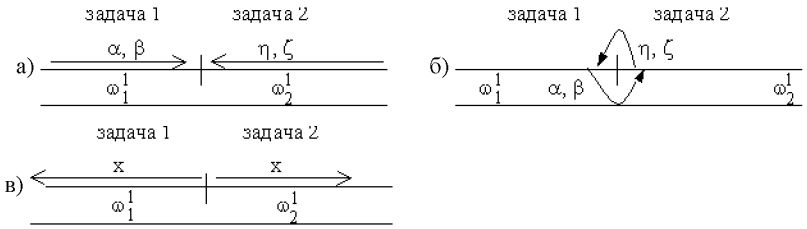



Рис. 3.53. Этапы вычислений по алгоритму 12: а - прямой ход встречных прогонок; б) - обмен прогоночными коэффициентами; в - обратный ход встречных прогонок

Во время прямого хода прогонок первая задача находит прогоночные коэффициенты $\alpha,\beta$, вторая $\eta,\zeta$. На этом шаге алгоритма (как и на остальных шагах) задачи производят независимые вычисления, следовательно, могут исполняться параллельно. Далее первая задача передает второй пару чисел $\alpha_{g+1}$, $\beta_{g+1}$, необходимых для запуска обратного хода прогонок во второй задаче, которая в это время производит аналогичную передачу $\eta_{g+1}$, $\zeta_{g+1}$, необходимую первой задаче. На третьем шаге обе задачи одновременно реализуют обратный ход прогонок, находя решение системы (3.15). Ускорение алгоритма 12 оценивается величиной

$$S = \frac{\tau_{ap}(I)}{1/2\tau_{ap}(I) + 2\tau_{\text{п}}} \xrightarrow[I\to\infty]{} 2.$$

К сожалению, этот подход в одномерном случае нельзя развить для большего количества задач. Как уже отмечалось, функциональная декомпозиция, как правило, ограничивает масштабируемость алгоритма. В нашем случае алгоритм 12 не обладает свойством масштабируемости.

Меняя число узлов сеточной области, изучим поведение ускорения ПВП, порожденного алгоритмом 12 (рис. 3.54).

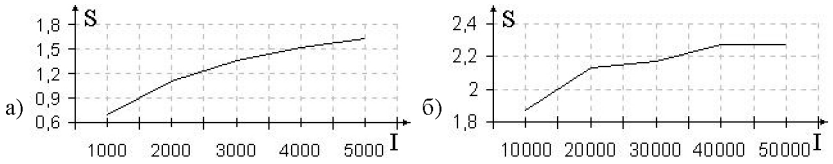



Рис. 3.54. Зависимость ускорений ПВП, порожденных алгоритмом 12, от числа узлов сеточной области для разных диапазонов I

При I=1000 (рис. 3.54, а) ускорение оказалось меньше 1, время вычислений по параллельному алгоритму больше времени вычислений по последовательному. Выясним, почему это происходит. Длительность арифметических операций оценивается величиной O(I), длительность обмена сообщениями - константа $2\tau_п$. Из определения функции "O большое" $O(I) \le C \times I$, где C - положительная константа. Для $I' < 2\tau_п/C$ ускорение ПВП окажется меньше единицы, что соответствует экспериментальным данным. Так как правая часть неравенства - константа, то начиная с некоторого $I'' > I'$ параллельный алгоритм обеспечит ускорение вычислений по сравнению с последовательным (рис. 3.54, б).

Ускорение исследуемого ПВП, согласно закону Амдала, не должно превышать 2. Однако, как видно из рис. 3.54, б, значение S может превосходить число процессоров. Изучая этот эффект, обратимся к табл. 3.4.

Таблица 3.4

Зависимость объема оперативной памяти ЭВМ, занимаемой одной ветвью ПВП, от значения I

| Объем занимаемой оперативной памяти (кБ) | Размерность сеточной области (I/1000) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 10 | 20 | 30 | 40 | 50 |
| | 23,43 | 46,88 | 70,31 | 93,75 | 117,19 | 234,37 | 468,75 | 703,12 | 937,5 | 1171,87 |

Начиная с размерности I=2000 (именно с нее ускорение превышает 2) объем памяти, занимаемый последовательным вычислительным процессом, превышает кэш процессора (256 кБ).



Следовательно, последовательный и параллельный процессы оперируют областями памяти, характеризующимися разными скоростями доступа к данным. Область памяти, занимаемая последовательным процессом при I=50000, выходит за размеры кэш на 915,87 кБ, а для ПВП этот показатель равен 329,93 и сравним с объемом кэш. Логично предположить, что как только объем памяти, занимаемой вычислительным процессом (неважно каким именно: последовательным или параллельным) превысит кэш на несколько порядков, ускорение опять подчинится закону Амдала. Действительно, при N=2000000 и занимаемой памяти 45,78 мБ s=1,9969.

В поисках подхода к построению масштабируемого алгоритма обратимся к методу циклической редукции [6]. Поясним его на примере решения СЛАУ трехдиагонального вида для I=6.

$$
\left.\begin{array}{l}
c_0u_0 - b_0u_1 = f_0; \\
-a_1u_0 + c_1u_1 - b_1u_2 = f_1; \\
-a_2u_1 + c_2u_2 - b_2u_3 = f_2; \\
-a_3u_2 + c_3u_3 - b_3u_4 = f_3; \\
-a_4u_3 + c_4u_4 - b_4u_7 = f_4; \\
-a_5u_4 + c_5u_5 - b_5u_7 = f_5; \\
-a_6u_5 + c_6u_6 = f_6;
\end{array}\right\}
\left.\begin{array}{l}
c'_1u_1 - b'_1u_3 = f'_1; \\
-a'_3u_1 + c'_3u_3 - b'_3u_5 = f'_3; \\
-a'_5u_5 + c'_5u_7 = f'_5;
\end{array}\right\}
u_3 = f'_3;
$$

Рис. 3.55. Редукция трехдиагональной СЛАУ для I=6

В отличие от метода исключения Гаусса, в методе циклической редукции процесс исключения неизвестных идет не от первых уравнений к последним (или наоборот), а посредством операций с перекрывающимися тройками соседних уравнений. С каждым четным уравнением системы складываются два его соседа (сверху и снизу) таким образом, чтобы исключить из рассмотрения неизвестные с четными индексами. Первое уравнение умножается на $a_1/c_0$ и складывается со вторым, третье домножается на $b_1/c_2$ и тоже складывается со вторым. Результирующее второе уравнение содержит только нечетные неизвестные. Аналогично поступаем с четвертым и шестым уравнением. Получившаяся новая система

уравнений сохранит трехдиагональную структуру. К ней применим описанную выше процедуру исключения, в общем случае получив третью систему уравнений относительно неизвестных $u_5$, $u_9$, $u_{13}$ .... В нашем примере (рис. 3.55) для выражения одного неизвестного достаточно двух этапов редукции, при большем количестве неизвестных процесс продолжается далее.

Обратный ход метода связан с подстановкой найденных неизвестных в сформированные системы. Первое значение подставляется в последнюю систему (из которой это значение было найдено), из первого и последнего уравнения которой определяются два значения сеточной функции. Затем подстановкой этих уравнений в предыдущую систему определяются еще четыре значения и т.д. Подстановкой нечетных неизвестных в исходную СЛАУ определяются значения четных сеточных функций.

Недостатком метода является ограничение на количество уравнений системы. Только для $I=2^q-2$, где $q\in N$, процесс редукции закончится выражением одного неизвестного через правую часть, для иных I получим систему из нескольких уравнений, редукции не подлежащую. Эту систему можно решать методом исключений Гаусса, подставляя во время обратного хода найденные значения в системы, образованные процессом редукции, или добавляя к исходной системе некоторое количество фиктивных уравнений вида $u_i=1$, дополнив ее до $2^q-2$ уравнений.

Изначально представленный метод был сформулирован не для параллельной реализации (тогда еще не было соответствующих вычислительных систем), но теперь может с успехом использоваться для синтеза параллельного алгоритма решения СЛАУ трехдиагонального вида.

*Алгоритм 13*

Разобьем систему (3.15) на L подсистем, распределив их между задачами параллельного алгоритма. Первое и последнее уравнения каждой подсистемы задачи l ($2\le l\le L$) дублируются в соседних задачах, где они окажутся соответственно последним уравнением задачи l-1 и первым уравнением задачи l+1.

В ходе циклической редукции параллельного алгоритма каждая задача произведет исключение неизвестных своей подсистемы по



описанному последовательному методу. Пройдя все возможные шаги редукции, соберем оставшиеся уравнения в любой задаче. Она закончит процесс исключения неизвестных и начнет обратные подстановки, рассылая всем задачам алгоритма необходимые значения, получив которые каждая задача для своей подсистемы закончит обратный ход метода.

Ускорение такого алгоритма при $I=2^q-2$ составит

$$S=\frac{\tau_{ap}(I)}{\frac{\tau_{ap}(I)-\tau_{ap}(L)}{L}+\tau_{ap}(L)+2(L-1)\tau_{п}},$$

где $(L-1)\tau_{п}$ - время приема некоторой (пусть первой) задачей коэффициентов оставшихся L-1 уравнений; $\tau_{ap}(L)$ - время решения подсистемы из L уравнений первой задачей; $(L-1)\tau_{п}$ - длительность рассылки этой задачей найденных значений сеточной функции; $\frac{\tau_{ap}(I)-\tau_{ap}(L)}{L}$ - время решения остальных I-L уравнений системы (3.15).

Применяя правило “разделяй и властвуй”, организуем передачу данных по каналам, связывающим задачи в дерево. Тогда длительность пересылок снизится до $2(\log_2 L)\tau_{п}$. Если после каждой пересылки принявшая задача продолжит процесс исключения, посылая затем коэффициенты оставшихся уравнений далее по дереву, то снизится и время арифметических операций. Правда, при этом возрастет длительность пересылок (принимающей задаче придется оперировать с тремя уравнениями, два из которых придут по каналам) и потребуется иная топология коммуникаций.

### *3.4.4. Параллельные алгоритмы для неявных разностных схем, двумерный случай*

Простейшая неявная разностная схема для двумерного уравнения теплопроводности на декартовой прямоугольной сетке $\omega^2$ имеет вид

$$\frac{U_{i,j}^{n+1}-U_{i,j}^{n}}{h_t}-a\left(\frac{U_{i-1,j}^{n+1}-2U_{i,j}^{n+1}+U_{i+1,j}^{n+1}}{h_x^2}+\frac{U_{i,j-1}^{n+1}-2U_{i,j}^{n+1}+U_{i,j+1}^{n+1}}{h_y^2}\right)=0$$

и решается посредством матричной прогонки. На практике стараются избежать вычислений по алгоритму матричной прогонки, который считается неэкономичным. Используются схемы метода расщепления

$$\frac{\tilde{U}_{i,j} - U^n_{i,j}}{h_t} - a\frac{\tilde{U}_{i-1,j} - 2\tilde{U}_{i,j} + \tilde{U}_{i+1,j}}{h_x^2} = 0;$$

$$\frac{U^{n+1}_{i,j} - \tilde{U}_{i,j}}{h_t} - a\frac{U^{n+1}_{i,j-1} - 2U^{n+1}_{i,j} + U^{n+1}_{i,j+1}}{h_y^2} = 0$$

и метода переменных направлений

$$\frac{U^{n+1/2}_{i,j} - U^n_{i,j}}{1/2h_t} -$$

$$-a\left(\frac{U^{n+1/2}_{i-1,j} - 2U^{n+1/2}_{i,j} + U^{n+1/2}_{i+1,j}}{h_x^2} + \frac{U^n_{i,j-1} - 2U^n_{i,j} + U^n_{i,j+1}}{h_y^2}\right) = 0;$$

$$\frac{U^{n+1}_{i,j} - U^{n+1/2}_{i,j}}{1/2h_t} -$$

$$-a\left(\frac{U^{n+1/2}_{i-1,j} - 2U^{n+1/2}_{i,j} + U^{n+1/2}_{i+1,j}}{h_x^2} + \frac{U^{n+1}_{i,j-1} - 2U^{n+1}_{i,j} + U^{n+1}_{i,j+1}}{h_y^2}\right) = 0,$$

знакомые читателю из курса разностных схем.

Оба метода предусматривают применение скалярной прогонки последовательно по направлениям X и Y. Являясь экономичными, они требуют $O(I^2)$ арифметических операций для нахождения $I^2$ неизвестных значений сеточных функций на одном слое по времени. Говоря о методике решения разностных уравнений экономичных схем, нам достаточно сформулировать параллельные алгоритмы скалярной прогонки по направлениям X и Y на сеточной области $\omega^2$.



*Алгоритм 14*

В отличие от одномерного случая, в данном можно построить параллельный алгоритм, основанный на методе правой прогонки. Применяя функциональную декомпозицию, будем следить за тем, чтобы в один момент времени разные задачи производили вычисления над разными сеточными уравнениями. Разобьем $\omega^2$ между двумя задачами так, как это показано на рис. 3.56. Вычисления по алгоритму разделим на три этапа: начальный, основной и заключительный.

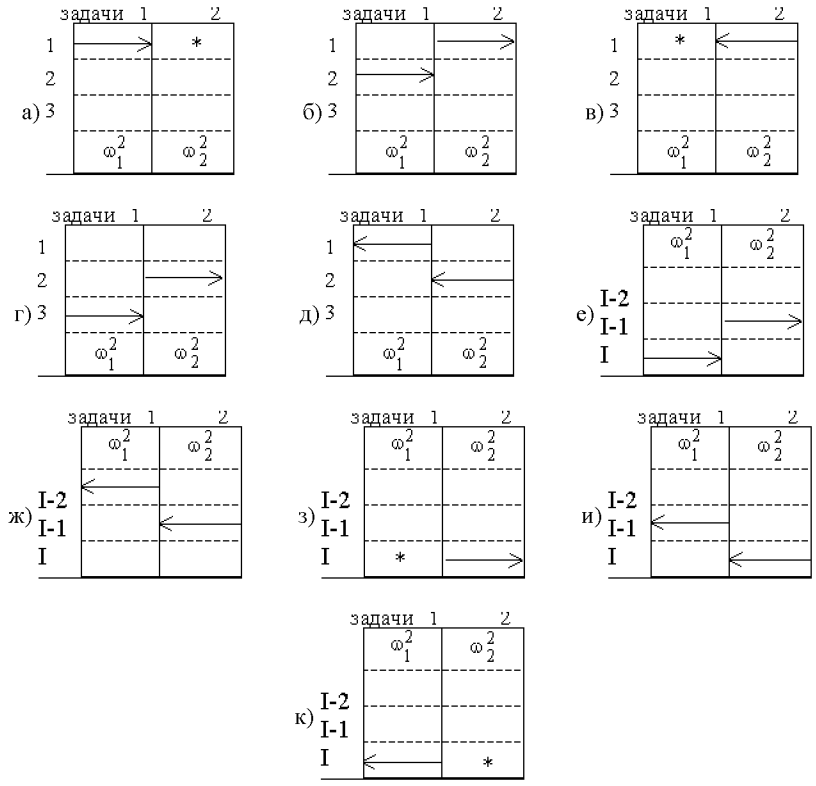



Рис. 3.56. Этапы алгоритма 14: а-в - начальный, г-ж - основной, з-к - заключительный. Звездочками обозначены простои задач

Далее не будем специально говорить об обмене данными между задачами, подразумевая, что перед прямым ходом прогонки они принимают прогоночные коэффициенты, после прямого хода передают их. Первая задача начинает прямой ход без принятия коэффициентов, последняя задача никому их не передает. Перед началом обратного хода задачи принимают соответствующие значения сеточных функций, после обратного хода передают их. Последняя задача ни от кого не принимает значение сеточной функции, первая задача никому не передает.

Начальный этап состоит из трех шагов алгоритма (рис. 3.56, а-в). На первом шаге задача 1 начинает прямой ход прогонки для строки 1 (рис. 3.56, а), задача 2 простаивает в ожидании прогоночных коэффициентов. Второй шаг обе задачи производят вместе (рис. 3.56, б), осуществляя прямой ход прогонки для разных строк: вторая задача для первой строки $\omega^2$, первая для второй. Третий шаг характеризуется обратным ходом прогонки (рис. 3.56, в), который начинает задача 2 для строки 1. Первая задача простаивает.

Основной этап (рис. 3.56, г-ж) заключается в чередовании двух следующих шагов для строк $2 \le i \le I-1$.

Прямой ход прогонки. Задача 1 для строки i+1, задача 2 для строки i.

Обратный ход прогонки. Задача 1 для строки i-1, задача 2 для строки i. Переход на следующую строку i=i+1.

Необходимость в заключительном этапе вызвана появлением простоев при i=I. Пока задача 2 производит прямой ход для i=I, задача 1 простаивает (рис. 3.56, з), так как не существует строки I+1 сеточной области. На последнем шаге алгоритма простаивает задача 2 (рис. 3.56, к), которая уже закончила вычисления.

Прогонки по направлению X для метода расщепления (будем ориентироваться на него) не требуют обмена сообщениями между задачами.

Ускорение алгоритма 14 определяется как

$$S = \frac{2I\tau_{ap}(I)}{I\tau_{ap}(I) + 2I\tau_{П} + 1/2\tau_{ap}(I)},$$

где $1/2\ \tau_{ap}(I)$ - время простоев одной задачи.



Распространим идею этого подхода на произвольное количество задач L. На начальном этапе такого алгоритма за первые L шагов производится только прямой ход прогонки. На первом шаге задача 1 работает со строкой 1, на втором задача 1 со строкой 2, задача 2 со строкой 1 и т.д., на L-м шаге задача 1 со строкой L, 2 с L-1, …, L-1 с 2, L с 1. В течение следующих 2L-3 шагов происходит чередование прямого и обратного ходов прогонки до тех пор, пока первая задача не прекратит простаивать во время обратного хода.

Основной этап характеризуется чередованиями прямого и обратного ходов без простоев. Если прямой ход задача L производит для строки i, то задача L-1 для i+1, …, задача 2 для i+L-1, задача 1 для i+L. Обратный ход задача L производит для строки i, задача L-1 для i-1, …, задача 2 для i-L+1, задача 1 для i-L.

Когда во время прямого хода начинает простаивать первая задача - наступает заключительный этап прогонки. Он состоит из 2L-2 шагов, на которых чередуются прямой и обратный ход, и L-1 шага обратного хода.

На любом шаге алгоритма все задачи производят однотипные вычисления по прямому или обратному ходу (кроме простаивающих задач). Данные одновременно пересылаются в одном направлении и только одному из двух соседей. Это позволяет причислить алгоритм 14 к классу конвейерных алгоритмов.

Изучая длительность простоев L-задачного алгоритма 14 обратим внимание на последнюю задачу. На начальном этапе она простаивает первые L-1 шагов, после которых работает без остановок. Длительность этих шагов составляет

$$\frac{L-1}{L}\tau_{ap}^{прям}(I) + (L-2)\tau_{П},$$

где $(L-2)\tau_{П}$ - время пересылок, в которых последняя задача не принимает участия, $\tau^{прям}_{ap}(I)$ - длительность прямого хода прогонки. На последнем этапе задача L простаивает заключительные L-1 шагов. Время этого простоя

$$\frac{L-1}{L}\tau_{ap}^{обр}(I) + (L-2)\tau_{П},$$

где $\tau^{обр}_{ap}(I)$ - длительность обратного хода прогонки.

Тогда ускорение L-задачного алгоритма

$$S = \frac{2I\tau_{ap}(I)}{\frac{2I}{L}\tau_{ap}(I) + 4I\tau_{п} + \frac{L-1}{L}\tau_{ap}(I) + 2(L-2)\tau_{п}}.$$

*Алгоритм 15*

Избавляясь от простоев, циклически разобьем область $\omega^2$ на блоки (рис. 3.57).

| | | | | | |
|---|---|---|---|---|---|
| $\omega_1^2$ | $\omega_L^2$ | $\omega_{L-1}^2$ | $\omega_{L-2}^2$ | … | $\omega_2^2$ |
| $\omega_2^2$ | $\omega_1^2$ | $\omega_L^2$ | $\omega_{L-1}^2$ | … | $\omega_3^2$ |
| $\omega_3^2$ | $\omega_2^2$ | $\omega_1^2$ | $\omega_L^2$ | … | $\omega_4^2$ |
| $\omega_4^2$ | $\omega_3^2$ | $\omega_2^2$ | $\omega_1^2$ | … | $\omega_5^2$ |
| … | … | … | … | … | … |
| $\omega_L^2$ | $\omega_{L-1}^2$ | $\omega_{L-2}^2$ | $\omega_{L-3}^2$ | … | $\omega_1^2$ |

Рис. 3.57. Циклическое разбиение области $\omega^2$ на блоки

Задаче 1 принадлежат блоки на главной диагонали блочной области, задаче 2 - блоки, циклически смещенные на одну позицию влево от главной диагонали, задаче l - циклически смещенные на l-1 позиций влево. Для двухзадачного алгоритма получим разбиение, представленное на рис. 3.58.

Прогонки по направлению Y для двух строк $\omega^2$ (i=1, 1+I/2) осуществляются за 4 первых шага алгоритма. На первом шаге (рис. 3.58, а) задача 1 производит прямой ход прогонки для первой строки с позиции j=1 до j=I/2, задача 2 для 1+I/2 строки для тех же позиций. На втором шаге строкой 1 займется вторая задача (рис. 3.58, б), первая задача закончит прямой ход для строки I/2. Обратный ход (рис. 3.58, в,г) проведут те же задачи с теми же строками. Далее работа продолжится над строками 2,2+I/2; 3,3+I/2; …;I/L,I. Прогонки



по столбцам $\omega^2$ пройдут аналогично (рис. 3.58, д,з). Хотя алгоритм 15 лишен простоев, длительность пересылок по сравнению с двухзадачным алгоритмом 14 возросла вдвое:

$$S = \frac{2I\tau_{ap}(I)}{I\tau_{ap}(I) + 4I\tau_{п}}.$$

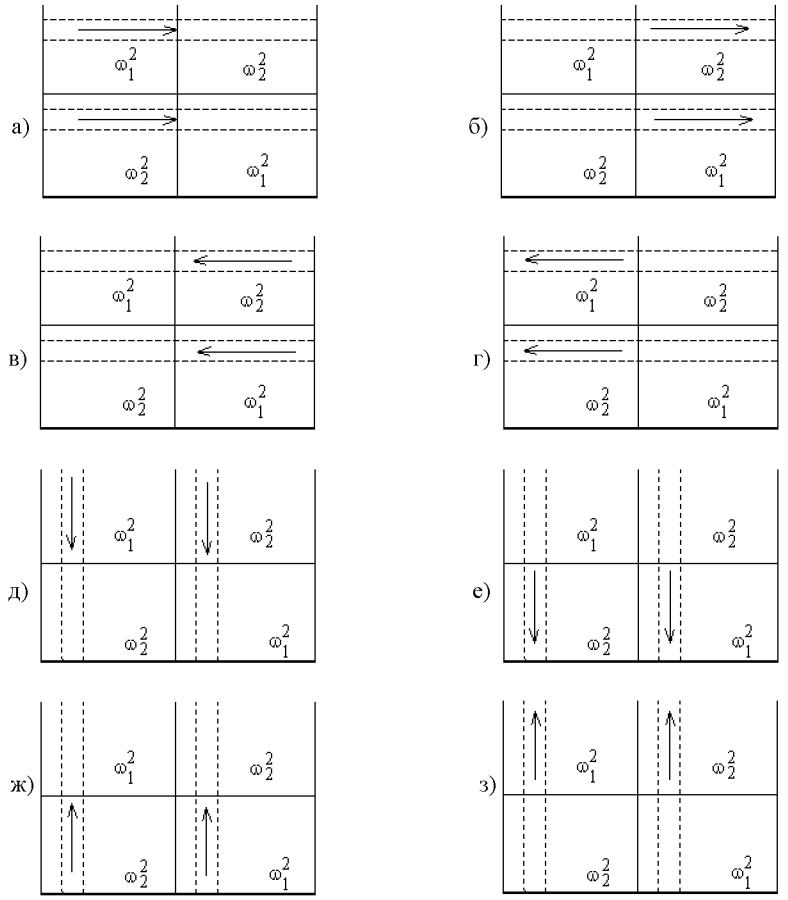



Рис. 3.58. Характерные шаги двухзадачного алгоритма 15

Идея циклического разбиения на блоки легко обобщается при синтезе L-задачного алгоритма 15. Прогонки по направлению Y для L строк $\omega^2$ i=1, 1+I/L, 1+2I/L, …,1+I-I/L осуществляются за 2L шагов. На первом шаге задача l (1≤l≤L) начинает прямой ход прогонки для строки 1+(l-1)I/L от j=1 до j=I/L. Далее, на втором шаге, для этой строки продолжит прямой ход прогонки задача l-1 (если l=1, то задача L-1) от столбца j=1+I/L до j=2I/L. В это время задача l работает со строкой 1+lI/L (если l=L, то со строкой 1). И так далее до шага L, когда задача 1 заканчивает прямой ход прогонки для строки I-I/L+1, задача 2 для строки 1, …, задача L для строки I-2I/L+1. Обратный ход прогонки каждая задача произведет над теми же фрагментами строк, для которых рассчитывали прямой ход. В результате для L строк $\omega^2$, отстоящих друг от друга на расстоянии I/L, найдены значения сеточных функций. Далее начинается прямой ход для строк i=2, 2+I/L, 2+2I/L, …,2+I-I/L. Последними будут найдены значения сеточных функций при i= I/L, 2I/L, …, I. Прогонки по направлению X производятся аналогично.

Ускорение L - задачного алгоритма 15

$$S=\frac{2I\tau_{ap}(I)}{\frac{2I}{L}\tau_{ap}(I)+8\left(1-\frac{1}{L}\right)I\tau_{п}}.$$

Существенно снизить объем пересылаемых данных и простои задач позволяет использование встречных прогонок.

*Алгоритм 16*

Если алгоритм состоит из двух задач, то модификации по сравнению с алгоритмом 12 почти не требуется, каждая СЛАУ в направлении Y решается одновременно двумя задачами (рис. 3.59) по вычислительной схеме

for i=1 to I do

алгоритм 12 для строки i;

end.

Продолжая не оговаривать порядок пересылок между задачами, учтем, что задачи L/2 и L/2+1 начинают обратный ход прогонок с обмена прогоночными коэффициентами α,β,η,ζ.

Рис. 3.59. Прогонки для одной строки области $\omega^2$ по двухзадачному алгоритму 16: а - прямой ход прогонок, в - обратный ход

Прогонки по направлению X производятся независимо, без обмена сообщениями. Ускорение $S=\frac{2I\tau_{ap}(I)}{I\tau_{ap}(I)+2I\tau_{п}}$. В отличие от двухзадачного алгоритма 14, двухзадачный вариант алгоритма 16 лишен простоев.

Для алгоритма с L=4 произведем следующее разбиение сеточной области (рис. 3.60).

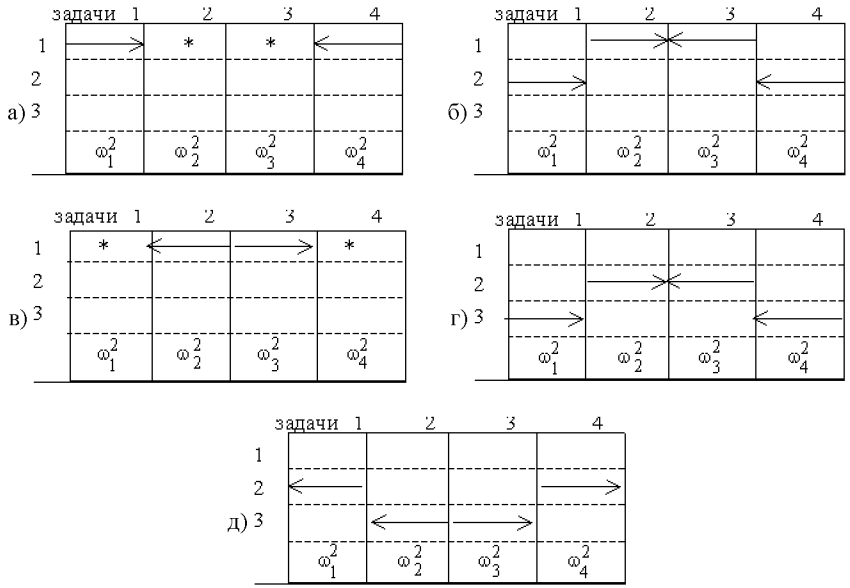



Рис. 3.60. Первые шаги четырехзадачного алгоритма 16: а, б, г - прямой ход прогонок; в, д - обратный ход

Задачи 1, 4 (рис. 3.60, а) начинают прямой ход прогонок для строки 1, задачи 2, 3 на первом шаге простаивают. На втором шаге

(рис. 3.60, б) задачи 2, 3 продолжают прямой ход для строки 1, задачи 1,4 начинают прямой ход для строки 2. Третий шаг (рис. 3.60, в) характеризуется простоем задач 1,4 и началом обратного хода прогонок для строки 1 задачами 2,3. Четвертый шаг (рис. 3.60, г) - прямой ход, задачи 1,4 осуществляют его для строки 3, задачи 2,3 - для строки 2. Пятый шаг (рис. 3.60, д) - обратный ход, задачи 1,4 осуществляют его для строки 2, задачи 2,3 - для строки 3. Далее прямой и обратный ходы чередуются, пока задачи 2,3 не произведут прямой ход прогонок для последней строки. На следующем после этого шаге задачи 1,4 будут простаивать, задачи 2,3 начнут обратный ход. Заключительный шаг алгоритма характеризуется простоем задач 2,3, в то время как задачи 1,4 завершат обратный ход прогонок для последней строки сеточной области. Прогонки по столбцам осуществляются каждой задачей самостоятельно, эти вычисления независимы.

Ускорение четырехзадачного варианта алгоритма 16 составит

$$S = \frac{2I\tau_{ap}(I)}{I\tau_{ap}(I)/2 + 4I\tau_{п} + \tau_{ap}(I)/4},$$

где $\tau_{ap}(I)/4$ - длительность простоев одной задачи алгоритма. Очевидно развитие этого подхода для произвольного четного числа задач. Для L - задачного алгоритма 16 ускорение составит

$$S = \frac{2I\tau_{ap}(I)}{\frac{2I}{L}\tau_{ap}(I) + 4I\tau_{п} + \frac{L-2}{2L}\tau_{ap}(I) + (L-4)\tau_{п}}.$$

Длительность простоев по сравнению с алгоритмом 14 снижена более чем вдвое.

*Алгоритм 17*

Осуществим двумерное разбиение $\omega^2$ на четыре части (рис. 3.61).

Четырехзадачный алгоритм 17 будет отличаться от четырехзадачного алгоритма 16 не только отсутствием простоев, но и вдвое меньшим объемом пересылок. Для него

$$S = \frac{2I\tau_{ap}(I)}{I\tau_{ap}(I)/2 + 2I\tau_{п}}.$$



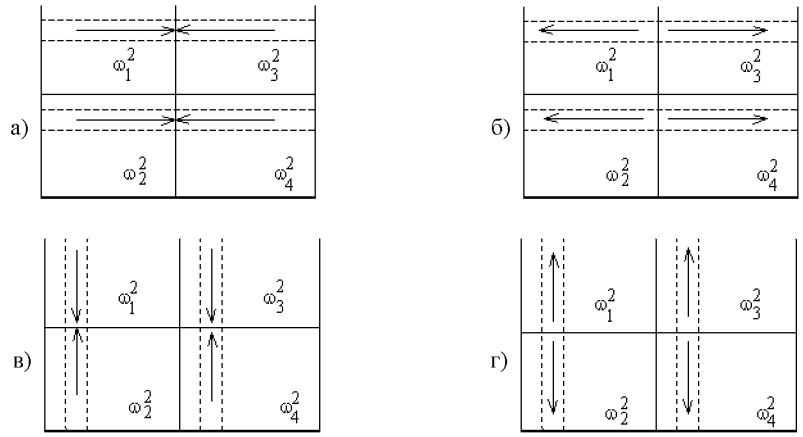



Рис. 3.61. Характерные шаги алгоритма с двумерной декомпозицией сеточной области. а - прямой ход прогонок по строкам; б - обратный ход прогонок по строкам; в - прямой ход прогонок по столбцам; г - обратный ход прогонок по столбцам

При выборе оптимальных параллельных алгоритмов для неявных разностных схем первостепенное значение приобретает сравнение протяженности границ сеточных подобластей, а не количество соседей характерной задачи (как для явных схем). Организуя передачу граничных значений одним пакетом, мы столкнемся с необходимостью хранить прогоночные коэффициенты для всей подобласти, что втрое увеличит объем памяти под одну задачу алгоритма (это неприемлемо). Поэтому объем пересылаемых данных в алгоритмах для неявных разностных схем всегда зависит от протяженности границ подобласти характерной задачи.

К сожалению, используя встречные прогонки для построения алгоритмов с L>4, нельзя организовать полноценную двумерную декомпозицию сеточной области, приходится обращаться к циклическому разбиению на блоки. Для восьми задач такое разбиение представлено на рис. 3.62.

Ускорение алгоритма 17 для произвольного числа задач $S = \frac{2I\tau_{ap}(I)}{\frac{2I\tau_{ap}(I)}{L} + 4I\left(1 - \frac{2}{L}\right)\tau_{п}}$ превышает соответствующие ускорения

алгоритмов 14,15,16. Это обусловлено отсутствием простоев (циклическое разбиение) и применением встречных прогонок (частичное двумерное распределение).

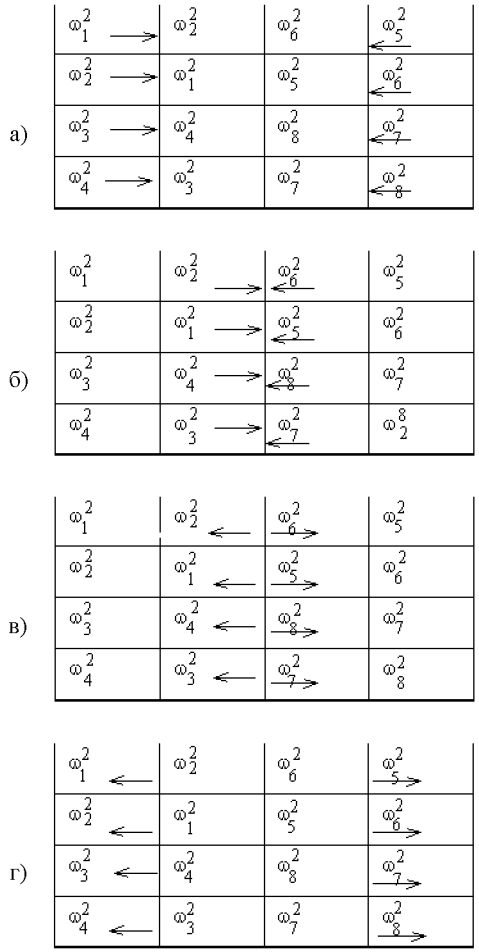



Рис. 3.62. Характерные шаги четырехзадачного алгоритма 17: а, б - прямой ход прогонок по строкам; в,г - обратный ход прогонок по строкам

Правда, задачи такого алгоритма будут иметь наибольшее число соседей ($\log_2 L$) по сравнению с остальными алгоритмами (2 соседа), а его реализация возможна лишь на сетевой конфигурации “гиперкуб”.



Алгоритм 13, использующий метод циклической редукции, легко адаптируется для расчетов на двумерной сеточной области. В этом случае декомпозиции $\omega^2$, одномерная и двумерная, полностью соответствуют аналогичным декомпозициям для алгоритмов явных разностных схем.

Ускорение при одномерной декомпозиции

$$S=\frac{2I\tau_{ap}(I)}{\frac{I}{L}\tau_{ap}(I)+\frac{I}{L}\left(\tau_{ap}(I)-\tau_{ap}(L)\right)+I\tau_{ap}(L)+2I(\log_2 L)\tau_{п}}$$

меньше ускорения аналогичных алгоритмов 14 и 16, так как необходимость решать конечную систему уравнений в одной задаче приводит к увеличению длительности пересылок данных между задачами пропорционально количеству узлов сетки и числу задач. В расчетах по алгоритмам 14 и 16 такая зависимость тоже присутствует, но не в виде произведения $2I\log_2 L$, а как сумма $4I+2(L-4)$ и $4I+(L-4)$. Поэтому, решая разностные уравнения на узких и длинных областях (Y>>X), используя вычислительную систему с небольшим числом процессоров (X>>L), предпочтительнее организовывать вычисления по алгоритму 16.

Возможность полноценной двумерной декомпозиции сеточной области ставит алгоритм, основанный на циклической редукции, вне конкуренции на квадратных областях. Ускорение по нему

$$S=\frac{2I\tau_{ap}(I)}{\frac{2I}{L}\left(\tau_{ap}(I)-\tau_{ap}(L)\right)+2\frac{I}{\sqrt{L}}\tau_{ap}(L)+4\frac{I}{\sqrt{L}}(\log_2 L)\tau_{п}}$$

превышает ускорения алгоритмов 15 и 17. Логарифмическая функция растет медленнее, чем степенная. Следовательно, организуя параллельные вычисления на квадратной сеточной области, следует применять алгоритм, основанный на циклической редукции.

## ЗАКЛЮЧЕНИЕ

В настоящем пособии авторы представили наиболее распространенные методы построения параллельных алгоритмов, проиллюстрировав их примерами из линейной алгебры и теории разностных схем. Читатель понимает, что не существует универсальных методов, удачно подходящих для всех случаев из вычислительной практики.

Например, можно разработать параллельный алгоритм вычисления степенной функции $x^n$, имеющий топологию коммуникаций "дерево" и состоящий из $\log_2 n$ шагов для n задач. Тогда, реализуя на каждом шаге одно умножение, можно свести n умножений последовательного алгоритма к $\log_2 n$ умножениям параллельного.

Подойдем к задаче с другой стороны, воспользовавшись известным соотношением

$$\sum_{i=1}^{n} \frac{r_i}{x - r_i} = \frac{n}{x^n - 1},$$

где $r_i$ - корни уравнения $x^n=1$.

Тогда, предварительно вычислив корни (они не зависят от значения x), построим другой параллельный алгоритм. На первом шаге найдем $a_i=x-r_i$ , что одно сложение в каждой задаче. Второй шаг - $b_i=r_i/a_i$ , одно деление. Далее с помощью дерева вычислим $f = \sum_{i=1}^{n} b_i$ за $\log_2 n$ сложений. На четвертом шаге вычислим c=n/f (одно деление в первой задаче). Последний шаг - нахождение результата: $x^n=c+1$ (одно сложение). В итоге новый алгоритм содержит $\log_2 n+2$ сложений и 2 деления. Длительность вычислений по нему сократится в $t_\times/t_+$ раз по сравнению с первым алгоритмом. В третьем параграфе упоминались аналогичные специализированные алгоритмы матричного перемножения: Виноградова и Штрассена.

Наиболее оптимальные параллельные алгоритмы строятся не по последовательным алгоритмам, для них разрабатываются специальные параллельные численные методы. Таким численным методам для задач линейной алгебры будет посвящена третья часть данного пособия.



## СПИСОК ЛИТЕРАТУРЫ

1. Бахвалов Н.С. Численные методы (анализ, алгебра, обыкновенные диф. ур-я.). - М.: Наука, 1973. - 632 с.

2. Вальковский В.А., Котов В.Е., Марчук А.Г. и др. Элементы параллельного программирования .- М.: Радио и связь, 1983.-239 с.

3. Голуб Д., Ван Лоун Ч. Матричные вычисления .- М.: Мир, 1999. - 548с.

4. Каханер Д., Моулер К., Нэш С. Численные методы и программное обеспечение. - М.: Мир, 2001. - 575с.

5. Корн Г., Корн Т. Справочник по математике для научных работников и инженеров. - М.: Наука, 1977.-832 с.

6. Ортега Д. Введение в параллельные и векторные методы решения линейных систем . - М.: Мир , 1991.-364 с.

7. Платон. Законы. Государство. Политик. - М.: Мысль, 1998.- 800 с.

8. Самарский А.А. Теория разностных схем.- М.: Наука, 1989. - 614 с.

9. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. М.: Наука, 1978.- 561 с.

10. Braunl T. The art of parallel programming. - Prentice Hall International (UK) Limited, 1993. 378 p.

11. Bruce P.L. Parallel programming an introduction - Prentice Hall International (UK) Limited, 1993. 270 p.

12. Foster J. Design and building parallel programs (http:// parallel.ru).

# ПРИЛОЖЕНИЕ

## Параллельные алгоритмы генерации псевдослучайных чисел

Излишне напоминать студенту старших курсов о важности генерации некоррелированной последовательности псевдослучайных чисел. Методы Монте-Карло и стохастической оптимизации довольно чувствительны к корреляции, поэтому качество псевдослучайной последовательности должно быть высоким. Кроме того, такие методы связаны с большим объемом вычислений, что до недавних пор ограничивало их применение. Использование параллельных алгоритмов и соответствующей вычислительной техники снимает это ограничение.

Для порождения последовательности псевдослучайных чисел традиционно применяется линейный конгруэнтный генератор [4] $x_{k+1}=(ax_k+b)\%m$ (3.16), в котором каждое следующее значение последовательности $x_{k+1}$ линейно выражается через предыдущее $x_k$. Элемент $x_0$ называют зерновым. Период такой последовательности (количество неповторяющихся элементов) не превышает m.

На настоящий момент известно весьма ограниченное количество значений a,b и c, обеспечивающих порождение некоррелированной последовательности (табл. П1)

Таблица П1

Три генератора псевдослучайных чисел

| Номер генератора | Параметры генератора | | |
|---|---|---|---|
| | a | B | M |
| 1 | 69069 | 0 | $2^{32}$ |
| 2 | 16807 | 0 | $2^{31}$-1 |
| 3 | 8z+5; z∈N | 0 | $2^g$; g∈N |

Зерновой элемент принято выбирать целым. При необходимости x приводят к диапазону [0,1) делением на m. Далее будем пользоваться правилом $x_{k+g}=(a^g x_k)\%m$ (если b=0) (3.17), по которому элемент последовательности выражается через своего g-го предшественника.

Читатель может выразить удивление тем, что псевдослучайные числа генерируются совершенно неслучайным образом по детерминированному правилу (3.16). Посоветуем ему представить



бильярдный стол, над которым закреплена фотокамера. Пусть по столу быстро катается шар, отталкиваясь от бортиков, а сверху это фотографирует камера с периодом в 3 секунды. Если в кадр не входят бортики стола, то рассматривая последовательность фотографий, неискушенному человеку сложно будет понять закономерность в перемещениях шара. Позиция шара может считаться случайной, хотя, строго говоря, таковой не является, будучи вполне детерминированной. Бортики стола в нашем примере играют роль операции %, не позволяя шару выйти за установленные пределы.

Существует три подхода к параллельной генерации псевдослучайных чисел [12]. Первый заключается в создании последовательности с помощью одного генератора в одной задаче. Для кластерной модели такой способ генерации не подходит, он требует пересылки большого объема данных по каналам с глобальной топологией. Второй подход предусматривает создание по одному генератору в каждой задаче алгоритма. Как уже отмечалось, хороших генераторов немного и для многозадачного алгоритма не хватит. Третий подход (наиболее распространенный) состоит в генерации последовательности разными задачами по одному правилу.

Пусть алгоритм состоит из фиксированного количества задач L, каждая из которых нуждается в определенном (заранее известном) количестве n псевдослучайных чисел. Подпоследовательности, принадлежащие разным задачам, являются участками одной общей последовательности, разбитой на L частей (рис. П1).

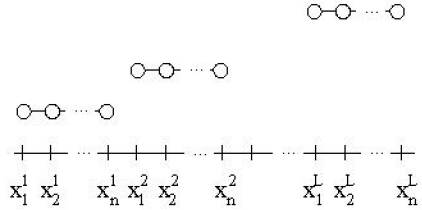



Рис. П1. Декомпозиция псевдослучайной последовательности на L участков длины n

Тогда l-я задача ($1\le l\le L$) производит вычисления по следующему алгоритму. На первом шаге получает от первой задачи зерновое значение $x_0$ и вычисляет $x^l_1=(a^{(l-1)n+1}x_0)\%m$ по формуле (3.17). На втором шаге, действуя по правилу (3.16), генерирует остальные элементы подпоследовательности, от $x^l_2$ до $x^l_n$.

В случае, когда n заранее неизвестно или может меняться, применим циклическое разбиение исходной последовательности (рис. П2).

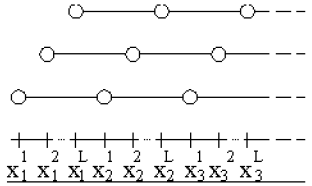



Рис. П2. Циклическая декомпозиция исходной последовательности

Алгоритм изменится. Получив на первом шаге зерновое значение от задачи 1, задача l вычисляет $x^l_1=(a^l x_0)\%m$. Далее, действуя по правилу (3.17), полагаем $x^l_{k+1}=(a^L x^l_k)\%m$, где $a^L$ следует вычислить один раз заранее и далее использовать как константу. Количество элементов подпоследовательности для каждой задачи ограничено лишь периодом используемого правила (m/L).

Отметим еще раз, что оба алгоритма предусматривают генерацию разными задачами непересекающихся частей одной последовательности, гарантируя тем самым некоррелированность подпоследовательностей между собой.

Если в алгоритме задачи порождаются и исчезают динамически, их количество непостоянно - применяется метод дерева случайных чисел (рис. П3).

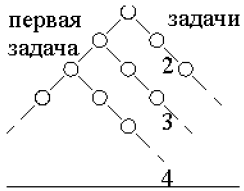



Рис. П3. Дерево случайных чисел

В первой задаче работает генератор $x^1_{k+1}=(ax^1_k)\%m$, поставляющий для остальных задач зерновые значения. Например, задача l в качестве зернового значения примет $x^l_0=x^1_l$. Далее все задачи, отличные от главной, используют другой генератор $x^l_{k+1}=(qx^l_k)\%m$. В отличие от предыдущих методов, метод дерева случайных чисел допускает генерацию двумя задачами l и l' коррелированных подпоследовательностей, если значения $x^l_l$ и $x^{1}_{l'}$ близки.



Учебное издание

*Головашкин Димитрий Львович*
*Головашкина Светлана Петровна*

**МЕТОДЫ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ**

(Часть 2)
*Учебное пособие*

Редактор Т. К. К р е т и н и н а

Компьютерная верстка Т. Е. П о л о в н е в а

Подписано в печать 26.05.03 г. Формат 60x84 1/16.
Бумага офсетная. Печать офсетная.
Усл. печ. л. 5,85. Усл. кр.-отт. 5,97. Уч.-изд.л. 6,5.
Тираж 100 экз. Заказ . Арт. С-22/2003.

Самарский государственный аэрокосмический
университет им. академика С. П. Королева.
443086 Самара, Московское шоссе, 34.

РИО Самарского государственного
аэрокосмического университета.
443001 Самара, ул. Молодогвардейская, 151.