Математические структуры и моделирование
2007, вып. 17, с. 72–93

УДК 519.6:004.272.2

ОБЗОР МЕТОДОВ РАСПАРАЛЛЕЛИВАНИЯ АЛГОРИТМОВ РЕШЕНИЯ НЕКОТОРЫХ ЗАДАЧ ВЫЧИСЛИТЕЛЬНОЙ ДИСКРЕТНОЙ МАТЕМАТИКИ

С.С. Ефимов

В статье рассматриваются представители параллельных алгоритмов, относящихся к решению задач таких разделов вычислительной дискретной математики, как числовые ряды, комбинаторные задачи, линейная алгебра, вычисления в узлах сеток и решеток, обработка графов.

Дана характеристика параллельных методов сортировки чет-нечетной перестановкой, подсчетом сравнений, двухпутевым слиянием, Шелла, расширения Брюса-Вагара для быстрой сортировки, метода параллельного двоичного поиска, алгоритмов Фокса и Кэннона умножения матриц, методов Гаусса и простой итерации решения систем линейных уравнений, параллельного решения краевой задачи, «жадного» алгоритма Прима построения минимального охватывающего дерева, алгоритм Дейкстры поиска кратчайшего пути и др.

Рост вычислительной мощности компьютерных систем, появление современных суперкомпьютеров, кластеров рабочих станций сделали возможным решение многих задач дискретной математики, требующих выполнения больших объемов вычислений за приемлемое расчетное время.

Существует целый ряд отраслей, в которых возникает необходимость решения подобных задач [1, 2]: квантовая физика (физика элементарных частиц, ядерная физика, квантовая теория поля); статистическая физика; физика молекул, физика плазмы; квантовая химия; науки о Земле (физика атмосферы, метеорология, климатология, геофизика, физика океана); биология, экология; экономика и эконометрия (вычислительная экономика, макроэкономика, теория массового обслуживания и теория оптимального управления, финансовая деятельность); социальные науки; математическая лингвистика (распознавание речи, анализ текста и автоматический перевод); информатика (ведение баз данных, распознавание образов, распределенные вычислительные системы); механика сплошных сред (гидродинамика и газодинамика, теория сопротивления

Омский государственный университет.
E-mail: s_efimov@mail.ru

материалов); баллистика; медицина, фармацевтика; промышленность, в том числе автомобиле- и авиастроение, нефте- и газодобыча, дизайн и другие.

Приведенный список не является исчерпывающим. Постоянное повышение мощности компьютерных систем приводит к тому, что задачи, которые еще в недалеком прошлом не могли быть решены в реальном масштабе времени, успешно решаются благодаря использованию параллельных алгоритмов, реализуемых на многопроцессорных системах, обладающих высоким быстродействием.

Попытки выполнения обычных последовательных алгоритмов решения известных задач дискретной математики на многопроцессорных вычислительных системах во многих случаях не приводят к повышению быстродействия. Для реального уменьшения времени решения требуется применение специальных параллельных вычислительных алгоритмов, учитывающих архитектурные особенности многопроцессорных систем. В связи с тем, что решения в прикладных областях, подобных приведенным в списке, сводятся к ряду известных задач дискретной математики, большой практический интерес представляют параллельные алгоритмы их решения.

Последовательным реализациям методов решения многих задач дискретной математики уделено достаточно много внимания в академической науке, в опубликованных статьях и монографиях. Параллельные аналоги известных последовательных алгоритмов к настоящему времени не были должным образом освещены в публикациях и не привлекали к себе необходимого внимания математиков и исследователей. Это объяснялось сначала отсутствием, а затем недостаточным развитием многопроцессорных вычислительных систем.

На современном этапе аппаратное обеспечение многопроцессорных систем развивается быстрыми шагами: в 2007 г., по сообщению Government Computer News, предполагается появление в США (компания Cray) суперкомпьютера производительностью более 1 петафлопса (10^{15} операций с плавающей запятой в секунду). В настоящее время предел быстродействия суперкомпьютера достигает 280×10^{12} флопсов (количество его процессоров превышает 130 тысяч) [3]. Поэтому любые исследования, связанные с развитием параллельных алгоритмов для многопроцессорных быстродействующих систем, а также популяризация и освещение в печати подобных алгоритмов представляются актуальными.

В статье будут рассмотрены преимущественно алгоритмы, предназначенные для выполнения на распределенных вычислительных системах типа кластеров рабочих станций. Параллельным алгоритмам, ориентированным на выполнение в вычислительных системах с общей памятью, также будет уделено некоторое внимание в тех случаях, когда это необходимо для сравнения с алгоритмами, предназначенными для выполнения в многопроцессорных системах с распределенной памятью.

В связи с тем, что полный список задач дискретной математики является весьма обширным, а объем настоящей публикации ограничен, в ней представлен обзор параллельных алгоритмов задач, относящихся к числу типовых и наиболее часто используемых. Приводимый материал позволит читателю получить

представление о приемах, используемых при распараллеливании алгоритмов решения различных задач дискретной математики, а при необходимости более детального знакомства – обратиться к соответствующим первоисточникам, на которые в работе имеются ссылки.

В статье рассматриваются представители параллельных алгоритмов, относящихся к решению задач таких разделов вычислительной дискретной математики, как:

— числовые ряды;

— комбинаторные задачи;

— линейная алгебра;

— вычисления в узлах сеток и решеток;

— обработка графов.

Рассмотрим представителей каждой из названных групп.

1. Числовые ряды

К этому разделу относятся задачи определения различных констант (числа π, числа и др.), представляемых в виде числового ряда [4], нахождение определенных интегралов, суммирование большого количества констант и подобные этому задачи, определение всех частичных сумм, выполнение операций параллельного префикса (сканирования). Здесь же рассматриваются задачи, решаемые аналогично перечисленным выше: поиска глобального максимума, минимума, вычисления значений логических выражений над длинными последовательностями величин и т. п.

Алгоритмы, позволяющие выполнять параллельное суммирование с помощью многопроцессорных систем с общей памятью, могут выполняться с помощью каскадной схемы суммирования, модифицированной каскадной схемы [5]. В названном первоисточнике [5] приводится также схема вычисления всех частных сумм.

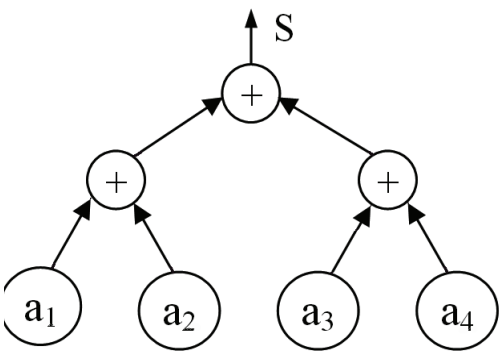

Рис. 1. Каскадная схема

Каскадная схема суммирования имеет вид бинарного дерева, в листьях которого записаны члены ряда, а во всех остальных узлах – операции сложения.

Например, чтобы найти сумму $S = a_1 + a_2 + a_3 + a_4$, потребуется схема, представленная на рис. 1.

Эффективность рассмотренной схемы определяется как

$$S_p = T_1/T_p = (n-1)/\log_2 n \ ,$$

где T_1 - трудоемкость алгоритма для одного процессора, T_p - для сети из p процессоров.

Эффективность использования процессоров вычисляется по формуле

$$E_p = T_1/pT_p = (n-1)/((n/2)\log_2 n) \ .$$

Для данной схемы $E_p \to 0$ при $n \to \infty$.

Если допустить, что каскадная схема вычисления выполняется с помощью кластера, каждый узел которого делает одну операцию сложения, то время на пересылку промежуточных результатов будет неизмеримо больше времени вычисления. Применение кластера по такой схеме оказывается нецелесообразным. Однако при использовании многопроцессорного суперкомпьютера с общей памятью каскадная схема является очень эффективной. Ограничением будет являться максимально допустимое на современном этапе количеством процессоров в одном суперкомпьютере – примерно 130 тыс. И то это в предположении, что все процессоры компьютера можно задействовать для вычислений по каскадной схеме. Решение задачи суммирования большого количества слагаемых, измеряемого сотнями тысяч и миллионами чисел, потребует использовать другие вычислительные схемы.

В этом случае для решения задачи с помощью многопроцессорной вычислительной сети с распределенной памятью (кластера) можно предложить, например, использовать каскадные схемы в комбинации с операцией редукции по сумме (см. рис. 2), если многопроцессорные узлы кластера реализуют каскадную схему, а редукцию по сумме выполнит обычный последовательный процессор. Схема обработки для такой реализации получается, если лес, состоящий из бинарных деревьев, соединить связями от корней образующих этот лес деревьев с новым узлом – корневым узлом сформированного дерева. Фрагменты части рисунка, расположенные ниже штрих-пунктирной линии, выполняются на многопроцессорных узлах кластера. Фрагмент, который расположен выше штрих-пунктирной линии, соответствует однопроцессорному корневому узлу.

Аналогом этой схемы является схема, в которой вместо редукции выполняется обычное последовательное суммирование (рис. 3). На этом рисунке фрагменты части рисунка, расположенные ниже штрих-пунктирной линии, также выполняются на многопроцессорных узлах кластера. Фрагмент, который расположен выше штрих-пунктирной линии, соответствует однопроцессорному корневому узлу. Однако вычислительная трудоемкость данной схемы примерно такая же, как и у схемы на рис. 2, так как редукция по сумме выполняется на обычном последовательном компьютере.

Выбор одной из двух схем зависит от программного обеспечения, используемого для последовательного суммирования, от эффективности его реализации в конкретной программной системе.

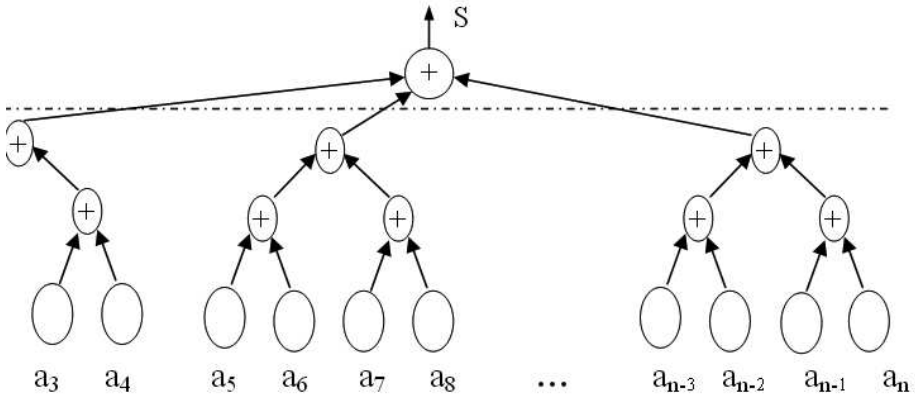

Рис. 2. Каскадная схема с редукцией по сумме

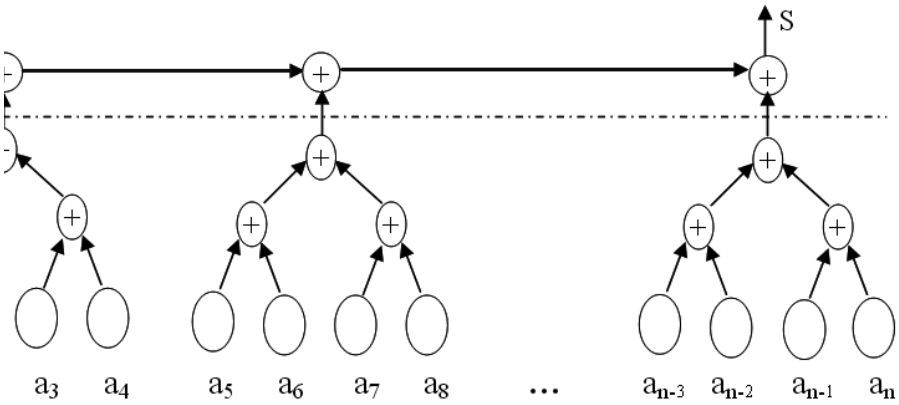

Рис. 3. Каскадно-последовательная схема

Если же в распоряжении имеется кластер, позволяющий лишь на одном узле реализовать каскадную схему, а на других – только последовательное вычисление, то можно использовать другую схему из [5] (см. рис. 4). Фрагменты части рисунка, расположенные ниже штрих-пунктирной линии, выполняются на однопроцессорных узлах кластера. Фрагмент, который расположен выше штрих-пунктирной линии, соответствует многопроцессорному корневому узлу.

Время сортировки рассмотренного метода определяется как

$$T_p = 2\ \log_2 n\ \ ,\ \ p = (n/\ \log_2 n)\ .$$

Ускорение и эффективность определяются формулами

$$S_p = (n-1)/2\ \log_2 n\ \ ,\ \ E_p = (n-1)/\ 2n\ .$$

По сравнению с обычной каскадной схемой ускорение уменьшается в два раза, однако оно стремится к 0.5 при росте n ($E_p \to 0.5$ при $n \to \infty$).

Последний вариант (рис. 5) соответствует случаю, когда все узлы кластера, включая корневой, являются многопроцессорными. Он позволяет ускорить вычисление результата на заключительном этапе объединения частичных резуль-

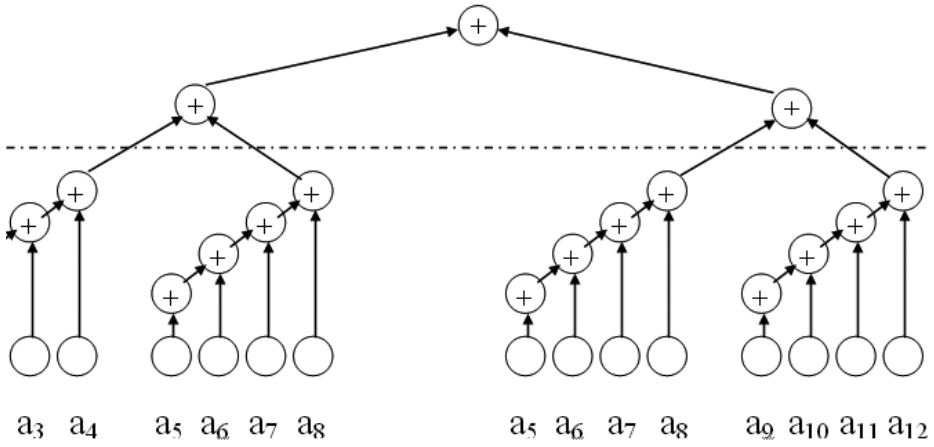

Рис. 4. Последовательно-каскадная схема

татов, а также использовать узлы с ограниченным количеством процессоров в каждом из них.

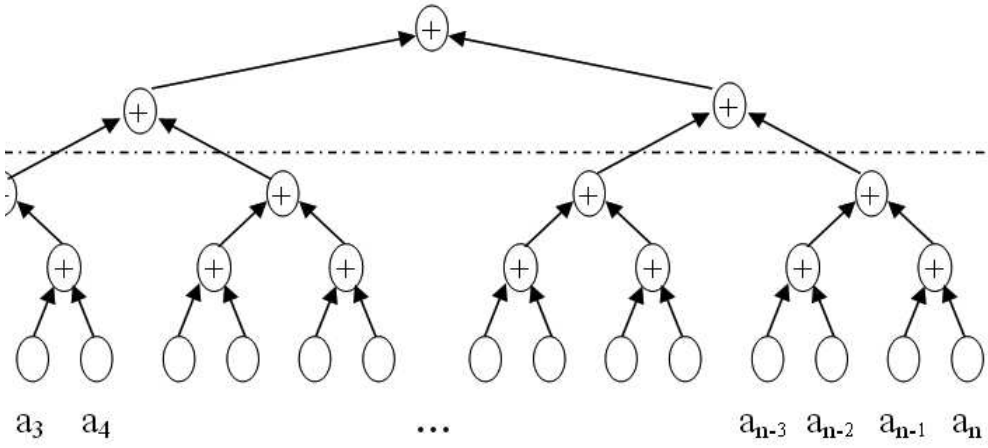

Рис. 5. Каскадно-каскадная схема

Алгоритм вычисления всех частичных сумм [5] рассматривать не будем, так как в представленном виде он не предназначен для распределенной обработки.

Для понимания схемы вычисления определенного интеграла с помощью кластера следует воспользоваться свойством его аддитивности [6]

$$\int_a^b f(x)\,dx = \sum_{i=0}^{p-1} \int_{a_i}^{b_i} f(x)\,dx \ ,$$

где $a_i = i * d$, $b_i = a_i + d$, $d = (b-a)/p$, p - число процессоров (не считая корневого). Тогда с учетом сказанного схема вычисления будет иметь вид, представленный на рис. 6.

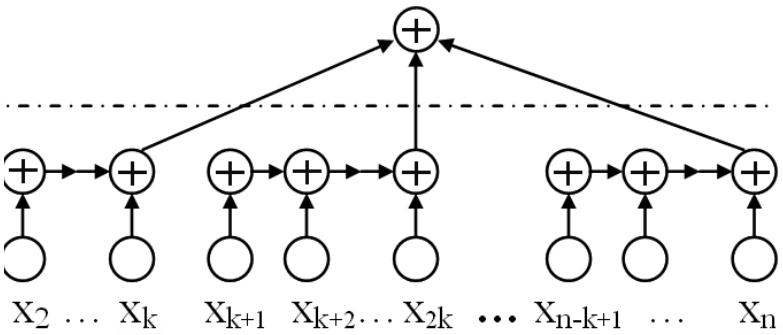

Рис. 6. Схема вычисления определенного интеграла

По данной схеме суммируется n значений с помощью $p+1$ узла кластера. Каждому процессору, кроме корневого, достается по $k = n/p$ слагаемых. После нахождения всех последовательных сумм (ниже пунктирной линии) корневой процесс выполняет редукцию по сумме этих значений. Если корневой узел кластера является многопроцессорным, то он может вместо редукции выполнить сложение по каскадной схеме.

Для вычисления суммы числового ряда, например числа π [7], можно воспользоваться схемой, аналогичной той, которая представлена на рис. 6, изменив в ней порядок группирования слагаемых следующим образом. Первый узел кластера будет вычислять сумму $S_1 = x_1 + x_{k+1} + \cdots + x_{n-k+1}$, второй процессор найдет сумму $S_2 = x_2 + x_{k+2} + \cdots + x_{n-k+2}, \ldots$, p-й процессор найдет $S_p = x_k + x_{2k} + \cdots + x_n$. После этого корневой процессор выполнит редукцию и вычислит $\pi = S = S_1 + S_2 + \ldots + S_p$. Такой порядок группировки слагаемых легче программируется, позволяет более равномерно загрузить вычислительной работой процессоры в том случае, если сложность вычисления последних членов ряда (если они представлены в виде некоторого выражения) существенно отличается от сложности вычисления начальных.

Таким образом, идея распараллеливания вычислений при подсчете значений числовых рядов заключается в том, что все слагаемые делятся некоторым образом в зависимости от метода на определенное количество промежуточных групп, после чего подсчет сумм в группах выполняется одновременно разными процессорами вычислительной системы. Затем значения частичных сумм складываются для получения конечного результата. Способы вычисления сумм в промежуточных группах и в конечной группе зависят от используемого метода.

Эту же идею можно использовать при поиске максимальных, минимальных значений среди последовательности величин, подсчете значений логических выражений с операциями И, ИЛИ и другими бинарными функциями, как и с функциями от большего числа аргументов.

2. Комбинаторные задачи

Среди задач, относящихся к данной группе, можно обратить внимание, в первую очередь, на параллельные методы сортировки и поиска данных.

Суть многих параллельных алгоритмов сортировки [8] состоит в том, что упорядочиваемые данные разбиваются на какие-либо части в зависимости от метода, каждая из которых сортируется по отдельности разными процессорами. Затем упорядоченные части объединяются некоторым образом, опять же в зависимости от метода. Эти два этапа могут повторяться необходимое число раз.

Из методов параллельной сортировки в качестве примеров рассмотрим наиболее типичных представителей: метод пузырька, сортировку Шелла, быструю сортировку.

Для параллельной реализации метода пузырьковой сортировки используется модификация, называемая методом чет-нечетной перестановки [9]. В простейшем случае сортировка заключается в чередовании двух этапов. На первом этапе массив разбивается на пары $(a_0, a_1), (a_2, a_3), (a_4, a_5), \ldots, (a_{n-2}, a_{n-1})$, начинающиеся с четных индексов. В каждой паре слева помещается наименьший элемент из двух, справа – наибольший из них. На втором этапе аналогичная обработка выполняется в парах $(a_1, a_2), (a_3, a_4), (a_5, a_6), \ldots, (a_{n-3}, a_{n-2})$, начинающихся с нечетных индексов. Максимум через n повторений массив оказывается упорядоченным. Иллюстрация приведена на рис. 7. Стрелками обозначены пары элементов, которые могут поменяться местами.

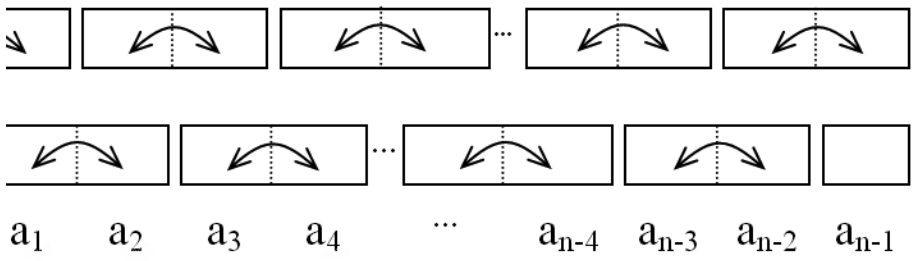

Рис. 7. Чет-нечетная сортировка

В рассмотренном методе подразумевается, что число процессоров равно числу n элементов массива A. Такая реализация может быть использована в многопроцессорном компьютере, но для кластера рабочих станций она не годится, так как затраты на пересылку соседних элементов будут превышать время на выполнение операций по сортировке.

Чтобы приспособить метод к реализации в распределенной вычислительной системе, состоящей из p процессоров, все n сортируемых элементов делятся на части, содержащие по n/p элементов. Каждая из частей рассылается в локальную память соответствующего процессора и предварительно сортируется в нем каким-либо быстрым методом. Затем чередуются два этапа:

1) Обмен фрагментами, состоящими из n/p элементов, между соседними процессорами (рис. 8). Этот обмен выполняется по очереди:

1а) внутри четных пар процессоров с номерами (0,1), (2,3), (4,5) ...,
1б) внутри нечетных пар процессоров с номерами (1,2), (3,4), (5,6) ...

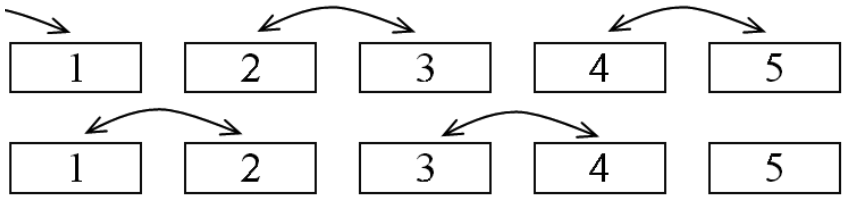

Рис. 8. Порядок обмена фрагментами в чет-нечетной сортировке

2) В каждом процессоре выполняется операция «сравнить и разделить» (compare-split). При ее выполнении процессор объединяет два упорядоченных фрагмента в один упорядоченный удвоенной длины $2n/p$. Для этого можно использовать метод сортировки слиянием («слить» два в один). Затем левый процессор пары оставляет у себя левую половину результата, а правый процессор – правую половину (рис. 9).

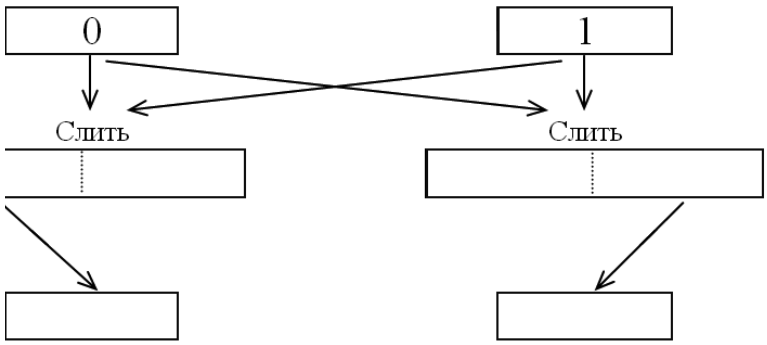

Рис. 9. Операция «Сравнить и разделить»

Для выполнения слияния необязательно выполнять слияние элементов до получения результата длиной $2n/p$. Достаточно в левом процессоре сливать фрагменты, начиная слева, до получения результата длиной n/p, а в правом процессоре сливать, начиная справа, также до получения результата длиной n/p. С более подробной иллюстрацией чет-нечетной сортировки можно ознакомиться в [10].

Вычислительная трудоемкость метода равна

$$T_p = (n/p)\ \log(n/p) + 2n\ .$$

Показатели ускорения и эффективности для p процессоров определяются как

$$S_p = n\ \log\ n/((n/p)\ \log(n/p) + 2n)\ ,$$

$$E_p = n\ \log\ n/(p((n/p)\ \log(n/p) + 2n))\ .$$

Рассмотрим теперь метод сортировки Шелла [11]. При последовательной реализации он имеет то преимущество, что перестановки выполняются между элементами, расположенными друг от друга на большом расстоянии. Для этого в массиве из n элементов на первом этапе сортируются элементы внутри групп, включающих элементы с номерами $[0, n/2], [1, n/2+1], [2, n/2+2], \ldots, [n/2-1, n-1]$. На втором этапе группы образуются из элементов, расположенных в 2 раза ближе друг к другу. В каждой группе находится в 2 раза больше элементов, то есть: $[0, n/4, n/2, 3n/4], [1, n/4+1, n/2+1, 3n/4+1], \ldots, [n/4-1, n/2-1, 3n/4-1, n-1]$. На следующем этапе расстояние между элементами уменьшается еще в 2 раза, а число элементов в группе увеличивается в 2 раза и т.д. На последнем этапе сортируется весь массив как одна группа.

На рис. 10 представлена иллюстрация сортировки методом Шелла для массива из 8 элементов. Верхние связи соответствуют четырем группам 1-го этапа, нижние связи – двум группам 2-го этапа.

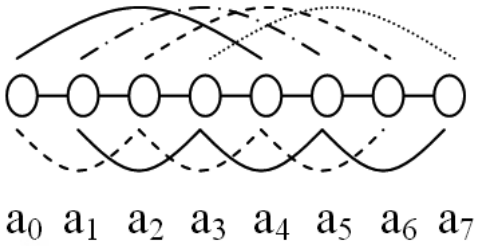

Рис. 10. Метод Шелла

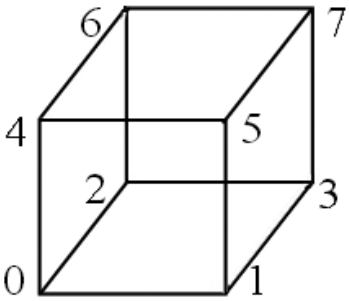

Рис. 11. Топология куба

Чтобы получить параллельную реализацию метода Шелла, необходимо объединить процессоры коммуникационной сети в топологию N-мерного гиперкуба [5]. Для этого в сети должно быть 2^N процессоров. Тогда максимальное расстояние между элементами будет равно N (2^N узлов, связанных последовательно, имели бы наибольшее расстояние $2^N - 1$). На рис. 11 представлен пример 3-мерного гиперкуба (имеет вид обычного куба), у которого последовательность нумерации вершин (узлов сети) позволяет использовать код Грея при обходе в порядке $0-1-3-2-6-7-5-4$. Код Грея, который используется для установления соответствия двух систем нумерации, характеризуется тем, что номера соседних процессоров отличаются только одним битом двоичного представления.

На предварительном этапе выполняется деление исходного массива на 2^N частей и рассылка каждой из них соответствующему узлу кластера. Затем все узлы кластера сортируют свои части массива каким-либо быстрым методом.

Перед началом процесса сортировки между узлами, являющимися соседними в структуре гиперкуба, организуются коммуникационные связи. На первом этапе в последовательной реализации метода Шелла обмен должен выполняться между удаленными друг от друга парами узлов с номерами: $0-4, 1-5, 2-$

$6, 3 - 7$. В параллельной реализации операция «сравнить и разделить» также будет выполняться между этими же парами узлов. Однако расстояние между узлами в структуре гиперкуба будет равно не четырем, а единице (см. рис. 10). На следующем этапе, как и в последовательной реализации метода, в группах с номерами $0-2-4-6$ и $1-3-5-7$ выполняются этапы обычной чет-нечетной сортировки до тех пор, пока не прекратится изменение порядка значений. Операция «сравнить и разделить» всегда проводится таким образом, что меньшая часть блока остается в процессоре, соответствующем вершине гиперкуба с меньшим номером, а большая часть – в процессоре с большим номером.

На заключительном этапе упорядоченные части собираются в единый упорядоченный массив. Обмен фрагментами на всех этапах (предварительном, основном и заключительном) в топологии гиперкуба выполняется существенно быстрее, чем в линейной топологии. Чем больше размерность гиперкуба, тем более эффективно использование метода Шелла.

Трудоемкость рассмотренного метода при числе итераций L (от 2 до p) определяется значением

$$T_p = (n/p)\,\log(n/p) + (2n/p)\,\log\,p + L(2n/p)\;.$$

Перейдем к рассмотрению метода быстрой сортировки. Последовательная реализация [12] заключается в неоднократном разделении упорядочиваемых значений на два фрагмента, таких, что элементы первого меньше всех элементов второго. К полученным частям снова применяется этот же прием и т.д. Для того чтобы распределить элементы левой и правой части на два фрагмента, в каждой из них выбирается ведущий элемент, после чего элементы, которые меньше его значения, переносятся в один фрагмент, а большие – в другой. При удачном выборе ведущих элементов трудоемкость алгоритма T_1 пропорциональна выражению $n\,\log\,n$. Среднее значение [11] определяется формулой $T_1 = 1.4n\,\log\,n$.

Для параллельной реализации метода быстрой сортировки (расширение Брюса Вагара [8] – т. н. гипербыстрая сортировка) так же, как и в случае метода Шелла, создается коммуникационная топология в виде N-мерного гиперкуба для кластера, содержащего 2^N процессоров. Для сортировки n значений предварительно каждому узлу рассылается блок из n/p элементов. Затем N раз выполняется следующая последовательность действий:

— выбирается ведущий элемент с рассылкой его значения всем процессорам гиперкуба;

— блок в каждом процессоре разбивается с учетом значения ведущего элемента на две части;

— организуются связи между парами процессоров, двоичные коды номеров которых отличаются только в N-ой позиции;

— выполняется взаимный обмен частями внутри каждой пары процессоров так, чтобы у процессоров, содержащих 0 в двоичном представлении своего номера, осталось по две части с меньшими значениями элементов, а у процессоров с 1 в номере – две части с большими значениями.

Каждый раз после однократного выполнения перечисленных действий гиперкуб порядка N оказывается разбитым на два гиперкуба порядка $N-1$, таких, что у гиперкуба с нулем в старших разрядах двоичных номеров его процессоров все значения упорядочиваемых элементов меньше всех значений другого гиперкуба, старший разряд двоичных номеров процессоров которого содержит единицу.

Чтобы облегчить удачный выбор ведущих элементов, желательно после рассылки каждому узлу гиперкуба блока значений выполнить их предварительную сортировку, а также обеспечить однородное распределение данных между процессорами.

Вычислительная сложность алгоритма, определяемая временем сортировки в процессорах, временем выбора ведущих элементов и сложностью разделения блоков, вычисляется по формуле:

$$T = (n/p)\,\log(n/p) + \log p + \log(n/p)\,\log p\ .$$

Можно предложить также параллельную реализацию сортировки подсчетом сравнений. Последовательная реализация метода [11] заключается в том, что в массиве A из n элементов каждый элемент A_i $(i = \overline{1, n-1})$ сравнивается с элементами A_j $(j = \overline{i+1, n})$. Если $A_i > A_j$, то соответствующий элемент вспомогательного массива B_i увеличивается на единицу, иначе $B_j = B_j + 1$. Элементам массива B до начала обработки присваиваются единицы. После выполнения всех сравнений он используется для указания местоположений элементов исходного массива в массиве-результате : $C_i = A(B_i)$ $(i = \overline{1, n})$.

Для параллельной реализации массив A, состоящий из n элементов, рассылается всем p узлам кластера. Затем каждый k-й узел $(k = \overline{0, p-1})$ реализует в своей локальной памяти метод подсчета сравнений между элементами k-го фрагмента массива A и всеми элементами массива B , то есть a_i $(i = kn/p, \ldots, (k+1)n/p-1)$ и a_j $(j = \overline{i+1, n})$. На следующем этапе выполняется суммирование одноименных элементов массива B из всех узлов с накоплением результата в одном узле, то есть операция редукции по сумме MPI_Reduce, если говорить в терминах интерфейса с передачей сообщений MPI.

Сортировка простым двухпутевым слиянием [11] также допускает параллельную реализацию. Предположим, что для сортировки используется сеть из двух процессоров, а числа в каждом из двух сливаемых фрагментов упорядочены по возрастанию. Перед началом работы оба фрагмента рассылаются каждому из процессоров сети. Первый процессор будет выполнять слияние фрагментов, начиная слева, второй – начиная справа. В средней части фрагментов слияние прекращается, а оба слитых фрагмента-результата после пересылки одного из них в какой-либо из двух процессоров объединяются в средней части (с учетом значений граничных элементов). Чтобы процесс объединения не требовал существенных затрат времени, числа в сливаемых фрагментах не должны существенно различаться по значениям, по крайней мере в средней части.

Изложенный в предыдущем абзаце прием можно обобщить на 4 процессора, разбив сливаемые фрагменты на две части и разослав левые части на 1-й и 2-й

процессоры, а правые части – на 3-й и 4-й. Затем каждая пара процессоров реализует рассмотренный выше алгоритм слияния двух фрагментов. Полученные фрагменты аналогичным образом сливаются вместе. Можно реализовать метод на 8-ми, 16-ти и т.д. процессорах.

Рассмотрев методы сортировки данных, остановимся на некоторых методах параллельного поиска. Алгоритм двоичного поиска среди упорядоченных величин в параллельной реализации [13] предполагает разбиение ключей, среди которых выполняется поиск, на последовательные группы по N/p элементов. Каждый из p процессоров сети производит двоичный поиск в своей группе элементов и, в случае обнаружения искомого ключа, передает сведения о нем основному процессору.

Время выполнения такого поиска имеет порядок $\mathrm{O}(\log(N/p))$, а стоимость – $\mathrm{O}(p\ \log(N/p))$. Если число процессоров равно $p = \log N$, то время выполнения имеет порядок $\mathrm{O}(\log N)$, а стоимость имеет порядок $\mathrm{O}(\log^2 N)$.

Параллельная реализация метода последовательного поиска среди неупорядоченных данных в сети из процессоров также может быть выполнена путем разбиения N значений на группы из N/p элементов с последующим поиском искомой величины каждым процессором в своей части и пересылкой результата центральному процессору. Время выполнения метода будет иметь порядок (N/p), а стоимость – порядок (N). То есть порядок стоимости остается таким же, как и для последовательного поиска на одном процессоре, однако время поиска удается уменьшить (по сравнению с (N)) примерно в p раз.

3. Задачи линейной алгебры

Типичным представителем задач, относящихся к линейной алгебре, является умножение матрицы на матрицу. При последовательной реализации используется формула

$$C_{ij} = \sum_{k=1}^{m} A_{ik}\, B_{kj} \quad (i = \overline{1,n}; j = \overline{1,m}).$$

Методы решения этой задачи зависят от выбранной топологии вычислительной сети и от способа распределения матриц A, B, C по процессорам. Если сеть содержит p процессоров, то каждую из трех матриц можно распределить между ними одним из 4-х способов: на p горизонтальных полос, на p вертикальных полос, на сетку размера p на p, совсем не разбивать на части. Тогда, в зависимости от способов распределения матриц, получается $4^3 = 64$ варианта решения задачи [14]. Однако большинство из этих способов являются неэффективными.

Рассмотрим в качестве примера так называемый ленточный алгоритм. Если учесть специфику задачи, формулу последовательного алгоритма, то более логично разбить матрицы A и C на p горизонтальных полос, матрицу B – на p вертикальных полос [15]. Схема разбиения представлена на рис. 12.

Сеть в этом случае будет иметь топологию «кольцо». При использовании топологии «кольцо» предполагается, что горизонтальные полосы матрицы A, содержащие по n/p строчек, перед началом вычислений рассылаются в локаль-

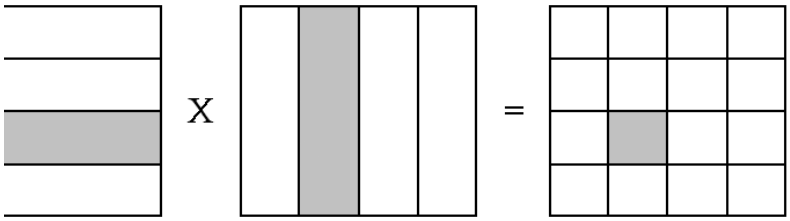

Рис. 12. Схема разбиения матриц при умножении по ленточному алгоритму

ную память соответствующих процессоров и постоянно там находятся. Вертикальные полосы матрицы B, содержащие по m/p столбцов, рассылаются в локальную память соответствующих процессоров. В процессе вычислений каждый i-й процессор ($i = \overline{1,p}$) находит произведение элементов i-й горизонтальной полосы на элементы находящейся у него вертикальной полосы. В первом процессоре на этом же этапе в массиве C получается фрагмент, расположенный на одну полосу ниже и правее и т.д. На втором этапе вертикальные полосы массива B циклически сдвигаются на одну полосу, то есть перемещаются в локальную память соседних процессоров. После этого вычисляются элементы соответствующих фрагментов в массивах C каждого процессора. Затем снова выполняется сдвиг полос массива B и т.д. p раз. На последнем этапе производится сбор горизонтальных полос массива C из всех процессоров в корневой процессор.

Если матрицу B не разрезать на вертикальные полосы, то сдвиг полос не потребуется. Однако в этом случае понадобится в p раз больше локальной памяти для каждого процессора. Для организации вычислений, при которой схема разбиения матриц A и B представляет собой прямоугольную решетку, используются методы Фокса и Кэннона [9,12]. Такой прием называют геометрическим способом распараллеливания. В данных методах матрицы A, B, C разбиваются, как показано в правой части рис. 12. Каждый из процессоров, которые можно объединить по топологии тора или иным удобным способом, отвечает за вычисление только своего прямоугольного фрагмента матрицы C. В процессе вычислений происходит обмен фрагментами в горизонтальном направлении для матрицы A и в вертикальном направлении для матрицы B.

Вычислительную трудоемкость рассмотренного блочного метода можно оценить как $T_p = 2n^3/p$.

Представление матриц в виде решетки, с одной стороны, увеличивает объем информации, пересылаемой между процессорами, однако, с другой стороны, обмен может быть совмещен с обработкой данных. Ленточный метод умножения позволяет снизить объем пересылаемых данных. Но разница в объемах пересылаемых данных уменьшается при увеличении числа процессоров в соответствии с выражением $(1+1/\sqrt{p})$. Кроме того, использование геометрического способа распараллеливания позволяет помещать прямоугольные блоки данных в процессорах с учетом близости их физического расположения. Это также позволяет ускорить пересылку данных. Разбиение матриц по принципу прямоугольной решетки дает возможность строить эффективные алгоритмы для

обработки разреженных матриц.

В качестве примеров решения систем линейных уравнений рассмотрим две модификации метода Гаусса и метод простой итерации [15]. Требуется решить систему линейных уравнений вида $AX = F$, где A – матрица коэффициентов системы, X – вектор неизвестных, F – вектор свободных членов.

Модификации метода Гаусса различаются способом распределения частей матрицы A и вектора F в локальной памяти узлов кластера.

В первой модификации матрица A и вектор F разрезаются горизонтальными полосами, количество которых определяется числом компьютеров в кластере: 0-я полоса помещается в локальную память 0-го компьютера, 1-я полоса – в память 1-го компьютера, ..., $(p-1)$-я полоса – в память $(p-1)$-го компьютера.

Прямой ход метода Гаусса, выполняющий преобразование матрицы A к треугольному виду, заключается в следующем. На первом шаге верхняя строка 0-го компьютера рассылается всем компьютерам с 1-го по $(p-1)$-й. Затем во всех полосах с ее помощью исключается крайний слева столбец во всех строках всех полос, кроме нее самой (в этой строке коэффициент при неизвестном x_0 должен стать равным единице). Очевидно, что обработка полос в каждом компьютере будет выполняться одновременно.

На втором шаге следующая сверху строка из 0-го компьютера рассылается всем остальным узлам. Затем во всех строках всех узлов, кроме 0-го, исключается следующий слева столбец. В 0-м узле это выполняется для строк, расположенных ниже текущей, а в ней самой коэффициент при x_1 должен стать равным единице. На последующих шагах аналогичная рассылка оставшихся в матрице A строк и обработка выполняется для всех строк всех узлов кластера в направлении сверху вниз.

Вид матрицы A размера 16×16, получившейся после преобразования к треугольному виду, приведен на рис. 13 (для 4-х узлов). Символом v на рисунке обозначены вещественные числа.

Обратный ход метода Гаусса выполняется аналогично, но начинается с нижней строки последнего компьютера.

Недостатком рассмотренной модификации метода Гаусса является то, что по мере увеличения номера рассылаемой строки верхние относительно нее компьютеры при прямом ходе метода и нижние при обратном начинают простаивать, пока остальные не завершат обработку своих строк. То есть вычислительная работа между компьютерами оказывается распределенной неравномерно.

Во второй модификации метода Гаусса матрица A и вектор F распределяются между узлами следующим образом: 0-я строка заносится в 0-й компьютер, 1-я – в 1-й, ..., $(p-1)$-я – в $(p-1)$-й. Затем p-я трока снова располагается под 0-ой в 0-ом компьютере, $(p+1)$-я – в 1-ом и т.д.

Обработка строк при таком распределении выполняется следующим образом. Сначала текущей для рассылки и обработки является 0-я строка в 0-ом узле, затем 0-я в 1-ом узле и т.д. вплоть до 0-ой в $(p-1)$-м узле. Затем текущими являются первые строки последовательно из 0-го, 1-го, ... $(p-1)$-го узлов и т.д. Вид матрицы A после прямого хода метода для данной модификации представлен на рис. 14.

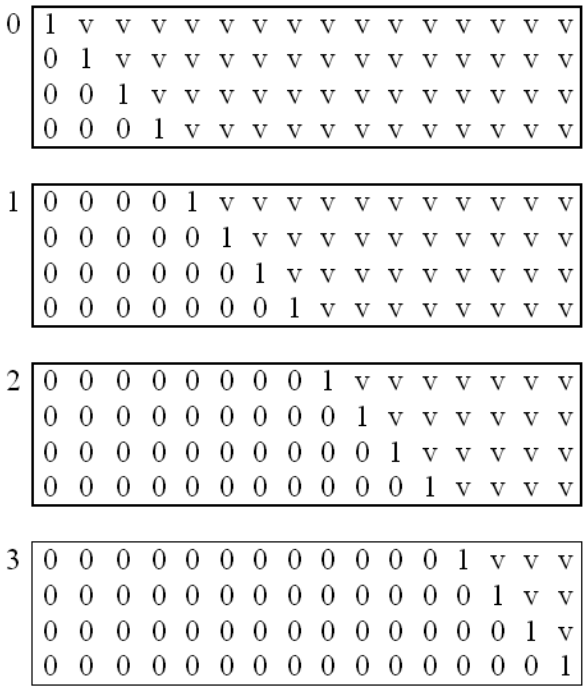

Рис. 13. Вид полос матрицы A после прямого хода 1-й модификации метода Гаусса

Для обратного хода метода порядок выбора текущих строк выполняется аналогично, но в направлении снизу вверх.

Достоинством второй модификации метода Гаусса является то, что как при прямом, так и при обратном ходе метода процессоры более равномерно загружены вычислениями. Время простоя узла, разославшего свою строку остальным узлам, во второй модификации равно времени обработки одной строки, а не целой полосы.

При сравнении двух модификаций метода Гаусса следует учитывать, что преимущество второй модификации начинает сказываться лишь при сравнительно больших матрицах – не менее нескольких сотен тысяч элементов. Это объясняется тем, что время на рассылку строк во второй модификации превышает аналогичную величину первой модификации. Во втором случае на каждом шаге строка рассылается всем узлам кластера, а в первом случае – только оставшимся, число которых с каждым шагом уменьшается (от $p-1$ до нуля).

Метод простой итерации предполагает преобразование системы линейных уравнений вида $AX = B$ к виду

$$x_i^{k+1} = \frac{1}{a_{ii}}[b_i - \sum_{j \neq i}^{1 \div N} a_{ij} x_j^k] \ \ (i = \overline{1, N}),$$

где k – номер шага итерации, i – номер уравнения, j – номер неизвестной.

Для использования метода матрица коэффициентов A и вектор B разрезаются на p горизонтальных полос по числу узлов кластера. Каждая полоса

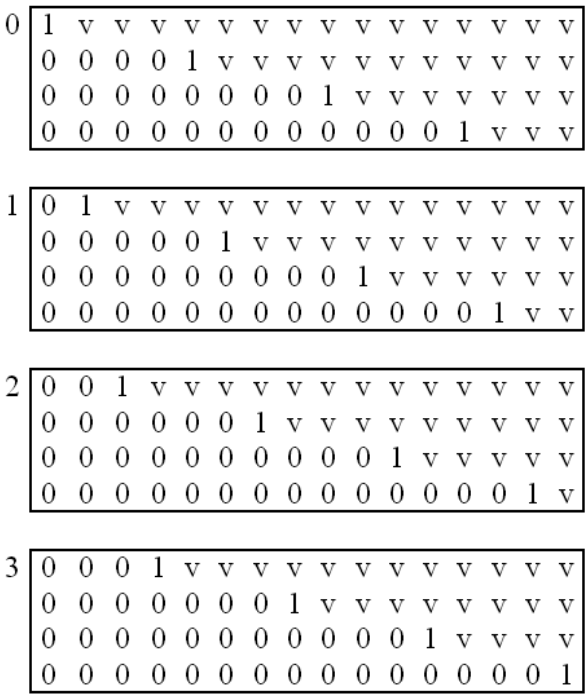

Рис. 14. Вид полос матрицы A после прямого хода 2-й модификации метода Гаусса

рассылается соответствующему узлу. Вектор неизвестных X должен присутствовать полностью во всех узлах. На следующем шаге каждый узел вычисляет свою часть вектора неизвестных X, рассылая ее после этого всем остальным узлам для выполнения нового шага итерации. Кроме того, в узлах вычисляется максимальное отклонение новых значений вектора неизвестных от старых в соответствующем номеру узла подмножестве вектора X. Процесс вычислений прекращается, когда во всех узлах кластера это отклонение станет меньше заданной точности. В библиотеке MPI для этого можно, например, использовать функцию MPI_Reduce с функцией MPI_MAX.

4. Вычисления в узлах сеток и решеток. Краевые задачи

Задачи, связанные с вычислениями в узлах сеток и решеток, решение краевых задач относятся к числу задач, для которых технология параллельного программирования также хорошо разработана [5].

Для иллюстрации рассмотрим численное решение двумерной краевой задачи для уравнения теплопроводности методом простой итерации с использованием явного итерационного алгоритма. Перед началом вычислений задаются граничные условия (значения на границе плоской области – прямоугольной сетки) и начальные значения во внутренних точках в начальный момент времени. На каждом этапе вычислений определяются приращения значений функции во внутренних точках сетки в предположении, что новые значения функций равны одной четверти от суммы значений в 4-х соседних точках (рис. 15).

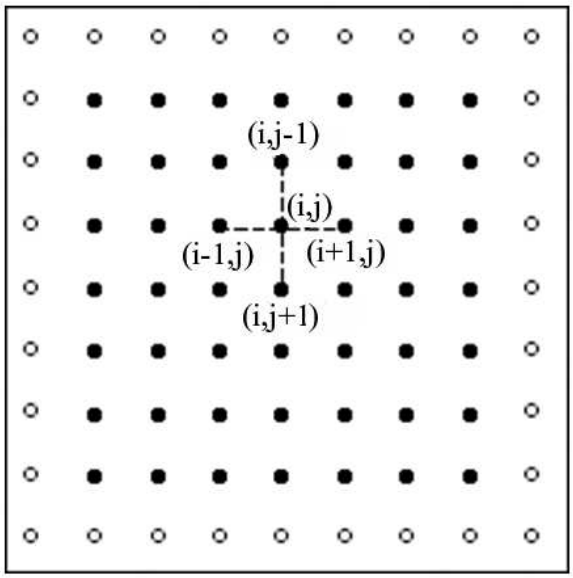

Рис. 15. Иллюстрация двумерной краевой задачи

В отличие от задачи вычисления произведения матриц, в данном случае разбиение области на p подобластей (по числу процессоров) выполняется с перекрытием соседних подобластей (рис. 16). В связи с тем, что при расчете значений точек крайнего (не граничного ряда) требуются точки, хранящиеся в локальной памяти соседнего процессора, добавляются граничные (заштрихованные) ряды, в которые через сеть передаются необходимые значения. Во внутренних точках каждой подобласти расчет выполняется по формуле

$$M_{ij} = 0.25(M_{i-1,j} + M_{i,j-1} + M_{i+1,j} + M_{i,j+1}) \ \ (j = \overline{1, 1000}).$$

Номер горизонтального ряда принимает значения в диапазоне $i = 1 \ldots 250$ для процессора $P0$, $i = 251 \ldots 500$ для процессора $P1$ и т.д. После выполнения очередного итерационного шага определяется максимальное значение погрешности во всех областях и, если необходимая точность не достигнута, выполняется обмен граничными строками и повторяется очередной шаг итерационного процесса.

При увеличении числа процессоров и, соответственно, подобластей время решения задачи с помощью рассмотренного метода уменьшается. Ускорение по закону Амдала определяется не только числом процессоров, но и величиной доли последовательных операций. Кроме того, при дальнейшем росте числа процессоров и увеличении времени на обмен между узлами кластера начинает проявляться действие сетевого закона Амдала. В результате этого рост ускорения не только прекращается, но и начинает уменьшаться.

5. Параллельные алгоритмы на графах

Многие реальные задачи прикладного характера, как и теоретические задачи, могут быть решены с использованием математических моделей различных про-

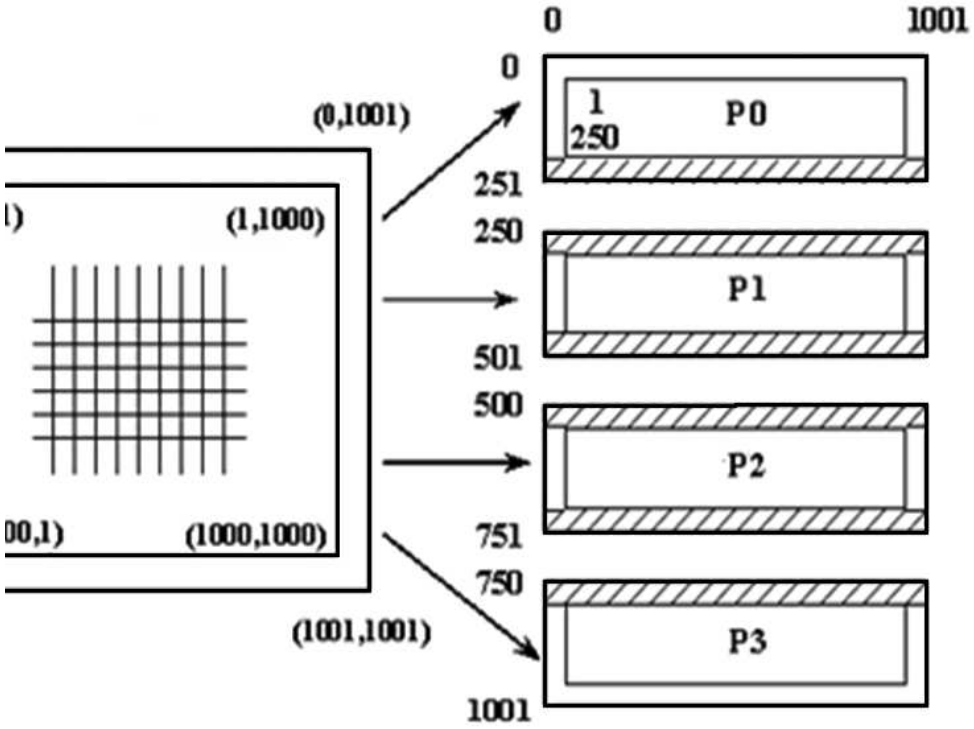

Рис. 16. Разбиение расчетной области на подобласти с перекрытием

цессов, явлений, систем, представленных в виде взвешенных графов. В качестве примеров можно привести линии телефонной связи, электрические сети, задачи составления расписаний, маршрутов самолетов. Для иллюстрации алгоритмов на графах рассмотрим два параллельных алгоритма из данной области: задачи поиска минимального охватывающего дерева и задачи поиска кратчайшего пути [13, 16].

В качестве практического применения алгоритма построения минимального охватывающего дерева можно назвать решение задачи построения сети персональных компьютеров с прокладыванием наименьшего количества соединительных линий связи.

Алгоритм Прима построения минимального охватывающего (остового) дерева относится к так называемым «жадным» алгоритмам. Подобные алгоритмы, базируясь на ограниченном наборе данных, выбирают лучшее решение, которое только возможно при наличии этого неполного набора данных. По данному алгоритму сначала выбирается произвольный начальный узел. Затем формируется начальная «кайма», включающая только узлы, непосредственно связанные с узлами, имеющимися в построенной части охватывающего дерева (то есть на первом шаге – с начальным узлом). На следующем этапе из каймы извлекается узел с наименьшим весом и присоединяется к остовому дереву (остову). После этого создается новая кайма за счет добавления к предыдущей кайме узлов, связанных с только что добавленным к остову. Опять извлекается узел с минимальным весом для добавления к остову и т. д., пока не закончатся все

вершины графа.

Параллельная реализация рассмотренного алгоритма для графа из N узлов в сети из p процессоров предполагает выделение для обработки каждому из процессоров по N/p узлов. Перед началом построения охватывающего дерева центральный процессор сети рассылает всем процессорам сведения о структуре графа и весах узлов, перечень выделенных конкретным процессорам узлов и номер начального узла остова.

На каждом проходе алгоритма процессоры определяют среди узлов своей группы номер узла, имеющего дугу с наименьшим весом, связанную с остовом. Эти номера узлов отправляются от процессоров сети центральному процессору, который выбирает из них узел графа с наименьшим весом, пополняет этим узлом остов и рассылает узлам сети сведения о новом остове. Затем повторяется очередной проход алгоритма до тех пор, пока не будут исчерпаны все узлы графа. Общее число проходов равно $N - 1$.

Сложность данного алгоритма оценивается величиной (N^2/p) при стоимости (N^2). Это намного меньше, чем сложность (2^N) метода «грубой силы», то есть метода сплошного перебора. Оптимальным для рассмотренного алгоритма считается число процессоров, равное $N/\log N$.

Задача поиска кратчайшего пути имеет важное практическое значение в приложениях, для которых веса дуг означают время, стоимость, расстояние, затраты и т. п.

Алгоритм Дейкстры отличается от рассмотренного алгоритма тем, что минимизируется не расстояние от группы узлов каждого процессора до остова, а расстояние от них до корня остова. При числе процессоров $N/\log N$ и использовании топологии гиперкуба трудоемкость метода имеет порядок $N \log N$.

Еще один вариант решения рассмотренного алгоритма поиска кратчайшего пути [13] заключается в следующем. Строится матрица связности графа размера $N \times N$. На пересечении i-ой строки и j-го столбца записан 0, если $i = j$; символ ∞, если узлы не связаны; значение веса, если узлы связаны. Данная матрица указывает веса между узлами, находящимися на расстоянии в одну дугу, и обозначается как A^1. После этого создается матрица A^2, которая хранит сумму весов дуг между узлами, удаленными друг от друга на одну или две дуги. Затем по этим двум матрицам уже можно получить матрицы A^3 или A^4 и т. д. до A^{N-1}. На каждом шаге получения из матрицы A^1 матриц $A^2, A^4 \ldots$ выполняются операции поиска минимума из суммы элементов матриц (сумм весов дуг, соединяющих узлы).

Если в известном алгоритме умножения матриц трактовать сложение как операцию взятия минимума, а умножение как операцию сложения, то алгоритм поиска кратчайшего пути сводится к алгоритму умножения матриц. Следовательно, распараллеливание алгоритма решения задачи поиска кратчайшего пути может быть выполнено так же, как распараллеливание алгоритма умножения матриц рассмотренными выше методами (ленточным, Фокса, Кэннона).

Для многих из рассмотренных параллельных алгоритмов разработаны программы, исходные тексты которых можно найти в сети Интернет. Ряд программ, реализующих данные алгоритмы, входит в состав специализированных

параллельных библиотек. В качестве примера можно назвать библиотеки ATLAS, Aztec, BlockSolve95, Distributed Parallelization at CWP, DOUG, GALOPPS, JOSTLE, NAMD, P-Sparslib, PIM, ParMETIS, PARPACK, PBLAS, PETSc, PGAPack, PLAPACK, ScaLAPACK, SPRNG [17]. Они содержат программный код, реализующий параллельные алгоритмы линейной алгебры, сеточных методов, методов Монте-Карло, генетических алгоритмов, рендеринга изображений, квантовой и молекулярной химии и другие.

Сложность и трудоемкость разработки распределенных программ с использованием интерфейса передачи сообщений долгое время тормозили широкое развитие многопроцессорных систем с распределенной памятью. Появление в этой области библиотек подпрограмм позволило избавить программистов от рутинной работы. Тем не менее наличие таких библиотек предполагает четкое понимание механизмов работы параллельных алгоритмов, заложенных в программные модули этих библиотек. С другой стороны, далеко не все известные методы используются в программных модулях, входящих в состав библиотек.

Таким образом, в работе выполнен обзор методов распараллеливания алгоритмов решения ряда типовых задач, относящихся к области вычислительной дискретной математики, преимущественно на основе распределенных вычислительных систем, не имеющих общей памяти.

Рассмотрены такие задачи, как определение значений числовых рядов (вычисление определенных интегралов; каскадные схемы суммирования: каскадно-последовательная, последовательно-каскадная, каскадно-каскадная; определение максимальных и минимальных значений среди больших последовательностей величин; определение значений длинных выражений, построенных на основе логических и других операций), комбинаторные задачи (методы сортировки: чет-нечетной перестановки, Шелла, расширение Брюса-Вагара для быстрой сортировки; методы параллельного двоичного поиска среди упорядоченных данных и последовательного поиска среди неупорядоченных величин), задачи линейной алгебры (умножение матриц: ленточный алгоритм, алгоритм Фокса, алгоритм Кэннона; две модификации метода Гаусса решения систем линейных уравнений; метод простой итерации решения СЛУ), краевая задача (метод простой итерации с использованием явного итерационного алгоритма), параллельные алгоритмы на графах («жадный» алгоритм Прима построения минимального охватывающего дерева, алгоритм Дейкстры поиска кратчайшего пути, алгоритм поиска кратчайшего пути сведением задачи к умножению матриц).

Литература

1. Корнеев В.В. Параллельные вычислительные системы. М.: Нолидж, 1999. 320 с.
2. Проект: ИММ УрО РАН / Моделирование.
   – http://www.parallel.ru/russia/map/data/project29.html.
3. Список TOP500 наиболее мощных компьютеров мира. – http://www.parallel.ru/.
4. Фихтенгольц Г.М. Основы математического анализа (Т.2). СПб.: изд-во «Лань», 2001. 464 с.

5. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. Учебное пособие: Изд. 2-е, доп-е. Н.Новгород: изд-во ННГУ, 2003.
6. Богачев К.Ю. Основы параллельного программирования. М.: БИНОМ. Лаборатория знаний, 2003. 342 с.
7. Дацюк В.Н., Букатов А.А., Жегуло А.И. Методическое пособие по курсу «Многопроцессорные системы и параллельное программирование». Ростов-на-Дону: РГУ, 2000.
8. Миллер Р., Боксер Л. Последовательные и параллельные алгоритмы: Общий подход. М: Бином. Лаборатория знаний, 2006. 406 с.
9. Библиотека параллельных алгоритмов ParaLib. Руководство пользователя. Нижний Новгород: НГУ, 2004. 29 с.
10. Ефимов С.С. Параллельные вычисления. Методические указания к лабораторным работам. Омск: Изд-во: Наследие. Диалог-Сибирь, 2006. 32 с.
11. Кнут Д.Э. Искусство программирования. Т.3. Сортировка и поиск: Уч. пособие. М.: Изд. дом «Вильямс», 2000. 832 с.
12. Kumar V., Grama A., Gupta A., Karypis G. Introduction to Parallel Computing. Second Edition. Addison Wesley, 2003. 856 с.
13. Макконелл Д. Основы современных алгоритмов. М: Техносфера, 2004. 368 с.
14. Букатов А. А., Дацюк В. Н., Жегуло А. И. Программирование многопроцессорных вычислительных систем. Ростов-на-Дону: Изд-во ООО «ЦВВР», 2003. 208 с.
15. Корнеев В.Д. Параллельное программирование в MPI. Москва-Ижевск: Институт компьютерных исследований, 2003. 304 с.
16. Кормен Т.Х., Лейзерсон С.Е., Ривест Р.Л. Алгоритмы: построение и анализ. М: БИНОМ: МЦНМО, 2004.
17. Специализированные параллельные библиотеки.
– http://www.parallel.ru/tech/tech_dev/par_libs.html.