О РАСПАРАЛЛЕЛИВАНИИ ВЫЧИСЛИТЕЛЬНЫХ АЛГОРИТМОВ

В.А. Вшивков

Институт вычислительных технологий СО РАН

Рассмотрим проблемы параллельной реализации численных алгоритмов. В настоящее время существует огромное количество алгоритмов решения задач, которые эффективно реализуются на однопроцессорных ЭВМ. Но по мере развития задачи рано или поздно рамки однопроцессорных машин становятся тесными и возникает необходимость использования более мощной техники. Причин, по которым возникает такая необходимость, по крайней мере две. Первая – мы хотим, чтобы задача, которая успешно решается на однопроцессорной машине, решалась быстрее. И если у нас есть возможность считать на 100-процессорном компьютере, неплохо было бы в 100 раз быстрее получить результат. Это важно, например, в задачах прогноза погоды, в управлении какими-либо технологическими процессами и т.п. Существуют и другие задачи, в которых своевременное получение результатов имеет решающее значение. Это важная причина, но для научных исследований она не является главной. Как правило, нет большой разницы, получится результат через час или через десять часов. И было время, когда на БЭСМ-6 отдельные варианты считались месяцы. Вторая причина заключается в том, что на многопроцессорном компьютере мы можем решить более сложную задачу, и, на мой взгляд, эта причина гораздо важнее, чем первая. Например, многие задачи физики плазмы стали численно решаться только с появлением мультикомпьютеров. Как известно, популярный в физике плазмы метод частиц в ячейках предъявляет особенно большие требования к памяти и быстродействию ЭВМ [1-4].

Некоторые задачи легко поддаются распараллеливанию, т.е. для них легко найти алгоритмы, эффективно реализуемые на мультикомпьютерах, поскольку они требуют выполнения большого числа независимых вычислений. Для решения других же приходится создавать по существу новые алгоритмы. На первый взгляд может показаться, что путь для разработки самого быстрого параллельного алгоритма должен начинаться с самого быстрого существующего последовательного алгоритма. И на практике так чаще всего и бывает. Но это, вообще говоря, неверно. Фактически наилучший параллельный алгоритм нередко не имеет даже отдаленного сходства с наилучшим последовательным алгоритмом. Покажем

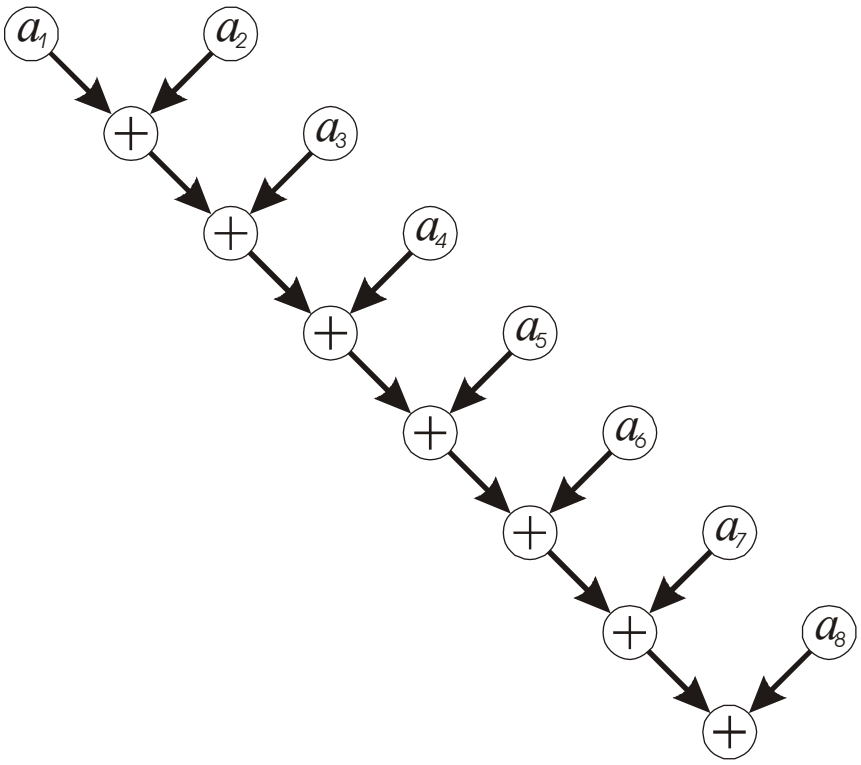

Рис. 1

на простом примере, что существуют как однопроцессорные, так и многопроцессорные алгоритмы, которые не получаются один из другого.

Рассмотрим известный пример сложения 8 чисел. Однопроцессорный алгоритм сложения этих чисел показан на рис. 1. Он эффективно реализуется с помощью оператора цикла и требует 7 операций сложения. Лучшего здесь ничего не придумаешь. Но, имея 8 процессоров и размещая каждое число на одном из процессоров, мы можем сложить эти числа за 3 операции, как показано на рис. 2. Легко видеть, что реализовать этот алгоритм на однопроцессорной машине более сложно, чем первый алгоритм, и очевидно, что он не будет более быстрым. Этот пример наглядно показывает разницу между последовательным и параллельным алгоритмами. В этом же примере можно увидеть и некоторые проблемы, возникающие при распараллеливании алгоритмов. Первая – это то, что возникают обмены данными между процессорами. И очевидно, что скорость передачи данных играет большую роль в оценке эффективности распараллеливания. Понятно, что чем больше происходит вычислений на процессоре и чем меньше обменов данными между процессорами, тем параллельный алгоритм эффективнее. Вторая проблема заключается в том, что хотя мы имеем 8 процессоров, мы не можем

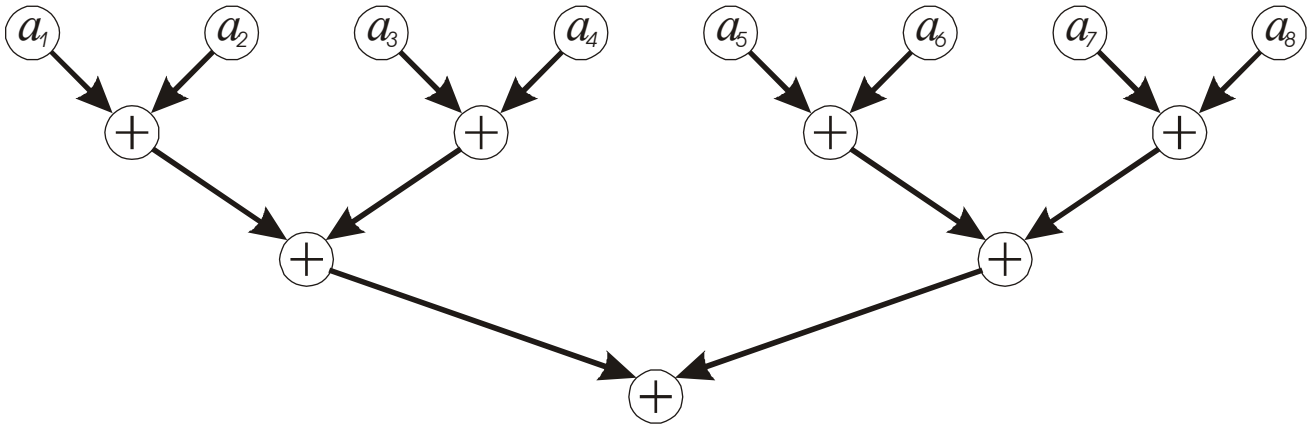

Рис. 2

сложить 8 чисел за одну операцию. В данном случае легко подсчитать, что для сложения *N* чисел необходимо порядка $\log_2(N)$ операций.

Возьмем более сложный пример. Рассмотрим итерационные методы решения системы линейных алгебраических уравнений:

$$Ax = b.$$

Здесь A – матрица размера $n \times n$; x и b – векторы размерности n. Как известно, решение многих дифференциальных уравнений после дискретизации на сетке сводится к решению системы линейных алгебраических уравнений большой размерности. И чем выше необходимая точность решения, тем больше размерность системы, что приводит к необходимости использования мультикомпьютеров. Итерационных методов решения систем линейных алгебраических уравнений в настоящее время существует довольно много, поэтому остановимся на двух из них, не самых лучших, но самых простых, для того, чтобы показать, какие появляются проблемы при распараллеливании алгоритмов. Первый из них – метод Якоби, который является примером явного алгоритма. В векторной форме он имеет вид

$$x^{m+1} = (I - D^{-1}A)x^m + D^{-1}b.$$

Здесь D – диагональная часть матрицы A; m – номер итерации. Реализация этого метода проста. Из первого уравнения системы находим первую компоненту вектора x^{m+1}, используя все остальные компоненты на итерационном слое m, из второго – вторую и т.д. Компоненты вектора x на слое $m+1$ вычисляются независимо друг от друга, поэтому их распределение между процессорами может быть произвольным. Например, на первый

процессор помещаем компоненты вектора x с номерами $1,...,n/p$, где p – количество используемых процессоров. На второй процессор поместим компоненты вектора с номерами $n/p+1,...,2n/p$ и т.д. Вычисление всех компонент может производиться одновременно на всех процессорах. После каждой итерации вычисленные на каждом процессоре компоненты вектора x рассылаются на все процессоры, а элементы матрицы $(I - D^{-1}A)$ и вектора $D^{-1}b,$ хранящиеся на каждом процессоре, не меняются. После этого цикл повторяется до тех пор, пока решение не будет получено с достаточной степенью точности.

Метод Якоби прост, но требует большого количества итераций. Поэтому возникла идея при вычислении компонент вектора x^{m+1} использовать уже вычисленные к этому моменту предыдущие компоненты вектора x^{m+1}. Эта идея приводит к методу Зейделя:

$$x^{m+1} = -D^{-1}Lx^{m+1} - D^{-1}Ux^{m} + D^{-1}b,$$

где L – нижняя треугольная часть матрицы A, а U – верхняя треугольная. Сходимость метода Зейделя, как правило, лучше, чем у предыдущего метода, но теперь мы вынуждены вычислять компоненты вектора x^{m+1} в строгом порядке, один за другим. Хотя реализация метода Зейделя простая, из вида формулы видно, что он относится к классу неявных алгоритмов. И печальным следствием этого является то, что он не поддается распараллеливанию. Если распределить компоненты вектора x^{m+1} между процессорами, как в предыдущем случае, то никакого ускорения не получится. Пока первый процессор насчитывает первую порцию компонент вектора x^{m+1}, остальные процессоры вынуждены простаивать, дожидаясь окончания его работы. Потом та же история происходит со вторым процессором и т.д. Но теперь обратимся к конкретным задачам.

Рассмотрим двумерное уравнение Пуассона в прямоугольной области S с граничными условиями первого рода на всей границе

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y).$$

В области S введем двумерную равномерную сетку с шагами h по каждому направлению и запишем простейшую разностную схему для решения уравнения Пуассона на заданной сетке. Она имеет вид системы линейных алгебраических уравнений:

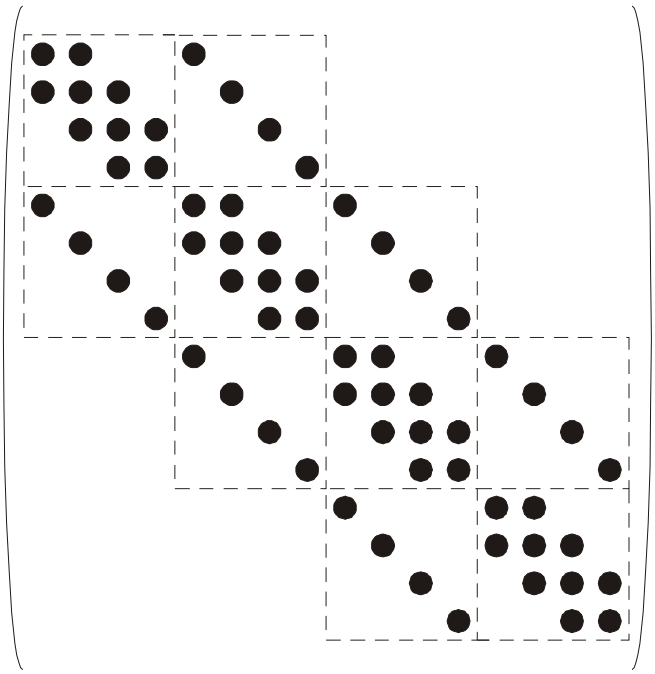

Рис. 3

$$\frac{u_{i+1,k} - 2u_{i,k} + u_{i-1,k}}{h^2} + \frac{u_{i,k+1} - 2u_{i,k} + u_{i,k-1}}{h^2} = f_{i,k},$$
$$i = 1,\ldots,I-1; \quad k = 1,\ldots,K-1,$$

где h – шаг сетки; i, k – индексы, указывающие, в какой точке находятся сеточные функции (i – по оси x, k – по оси y). Если теперь перенумеровать узлы сетки последовательно слева направо и снизу вверх, то матрица A в уравнении $Ax = b$ будет иметь характерный пятидиагональный вид. На рис. 3 приведен вид этой матрицы для $I = K = 5$. Черными кружками обозначены ненулевые элементы, остальные элементы равны нулю.

Метод Якоби для этой системы уравнений легко распараллеливается, а метод Зейделя – нет. Но тем не менее идею метода Зейделя здесь можно использовать. Для этого разобьем все узлы в шахматном порядке на красные и черные (произведем так называемое красно-черное упорядочивание узлов), как на рис. 4 (здесь белыми кружками обозначены красные узлы, черными – черные, а прямоугольниками – известные граничные значения). А сам метод разобьем на 2 этапа. На первом этапе будем вычислять значения сеточной функции $u_{i,k}$ в красных узлах, используя значения в черных узлах на предыдущей итерации:

$$u_{R,i,k}^{m+1} = \frac{1}{4}(u_{B,i+1,k}^{m} + u_{B,i-1,k}^{m} + u_{B,i,k+1}^{m} + u_{B,i,k-1}^{m} - h^2 f_{i,k}).$$

На втором этапе вычисляются значения $u_{i,k}$ в черных узлах, используя значения в красных узлах, но уже новые, вычисленные на первом этапе:

$$u_{B,i,k}^{m+1} = \tfrac{1}{4}(u_{R,i+1,k}^{m+1} + u_{R,i-1,k}^{m+1} + u_{R,i,k+1}^{m+1} + u_{R,i,k-1}^{m+1} - h^2 f_{i,k}).$$

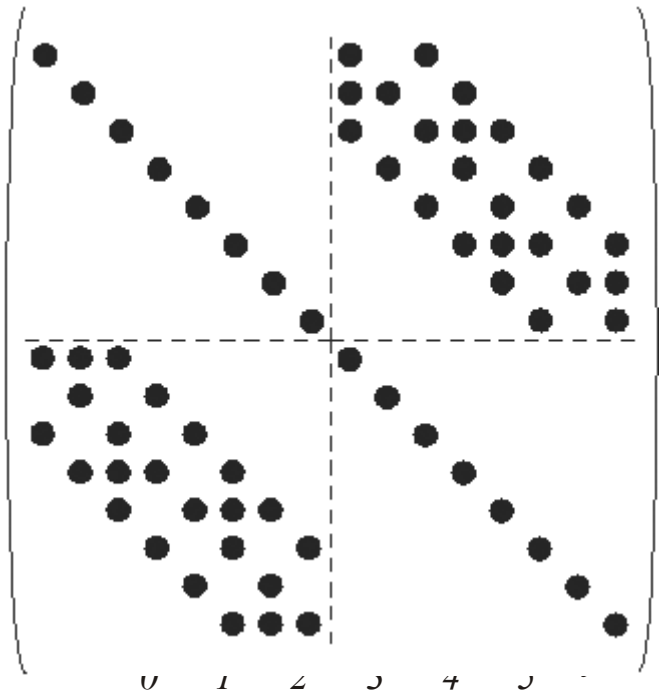

Рис. 4

Полученный метод распараллеливается так же просто, как и метод Якоби. Как легко видеть, компоненты вектора x^{m+1} на первом этапе вычисляются независимо друг от друга; то же самое происходит и на втором этапе. Сходимость метода лучше, чем у метода Якоби. Но, с другой стороны, это уже и не метод Зейделя. Если перенумеровать сначала красные узлы, а

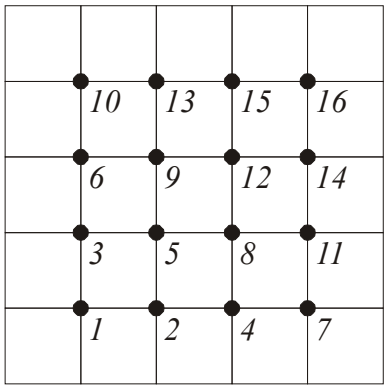

Рис. 6

затем черные последовательно слева направо и снизу вверх, то матрица A будет иметь вид, показанный на рис. 5. Еще следует заметить, что если в методе Якоби, имея n процессоров, мы получаем результат за один этап, то в методе Зейделя при красно-черном упорядочивании – за 2 этапа, и, кроме того, не можем использовать больше чем $n/2$ процессоров. Эта разница, конечно, чисто теоретическая, поскольку на практике, как правило, $n >> p$, тем не менее это иногда надо иметь в виду.

К этому можно еще добавить, что в данном конкретном случае можно полностью сохранить исходный метод Зейделя и при этом распараллелить его. Перенумеруем внутренние узлы сетки так, как показано на рис. 6. Тогда все 16 узлов разобьются на неравные по размеру группы, вычисления компонент вектора в которых уже легко распараллеливаются. Матрица A примет вид, изображенный на рис. 7. Как видно из рисунка, получается 7 групп с количеством элементов от 1 до 4. Следовательно, вычисления можно провести за 7 этапов и более 4 процессоров не понадобится. Коэффициент ускорения при этом получается равным $K_4 = t_1/t_4 = 16/7 = 2{,}29$, а эффективность распараллеливания $E_4 = K_4/4 = 0{,}57$.

Важную роль при получении последних методов сыграли большая разреженность матрицы системы линейных алгебраических уравнений и простота шаблона схемы. При более сложных матрицах усложняется и анализ возможности распараллеливания метода. Существуют алгоритмы,

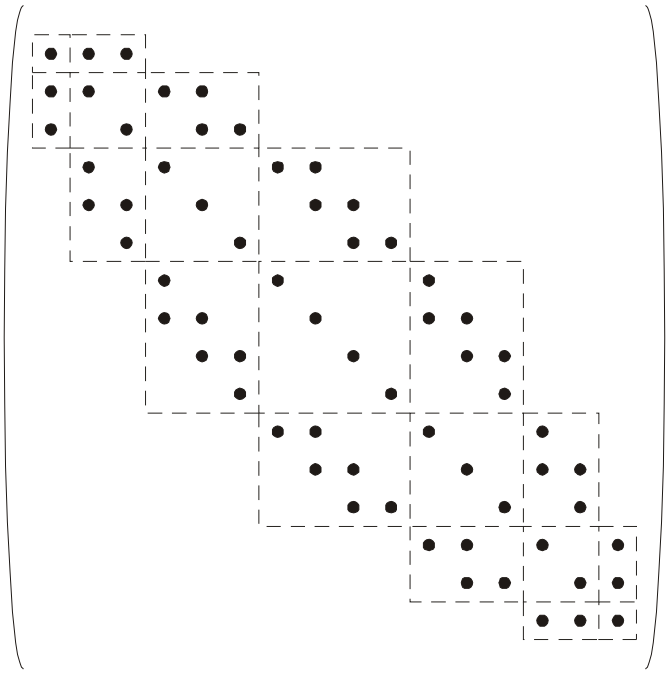

Рис. 7

позволяющие установить, на сколько групп независимых переменных можно разбить вектор решения, но изложение этой теории требует специального рассмотрения. Во всяком случае, учет конкретной задачи, конкретной системы линейных алгебраических уравнений позволяет найти наилучший вариант распараллеливания алгоритма.

Очень важным алгоритмом при решении дифференциальных уравнений является метод прогонки. На его основе разработан целый класс так называемых экономичных разностных схем. Эти алгоритмы неявные, следовательно, более устойчивые, чем явные схемы, иногда абсолютно устойчивые, поэтому они широко применяются при решении различных задач. На основе таких схем разработаны большие пакеты прикладных программ, которые хорошо оттестированы, показали свою надежность и универсальность. Поэтому при распараллеливании таких комплексов нежелательно отказываться от апробированных методов. Но, к сожалению, метод прогонки плохо поддается распараллеливанию.

Рассмотрим, к примеру, двумерное уравнение теплопроводности

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

и выберем для его решения метод продольно-поперечной прогонки. Схема метода двухэтапная и имеет следующий вид:

Первый этап:

$$\frac{u^{m+1/2} - u^m}{\tau/2} = \Lambda_1 u^{m+1/2} + \Lambda_2 u^m.$$

Второй этап:

$$\frac{u^{m+1} - u^{m+1/2}}{\tau/2} = \Lambda_1 u^{m+1/2} + \Lambda_2 u^{m+1}.$$

Здесь τ – временной шаг; Λ_1 – оператор разностной производной второго порядка по x

$$\Lambda_1 u_{i,k} = \frac{u_{i+1,k} - 2u_{i,k} + u_{i-1,k}}{h_x^2},$$

а Λ_2 – по y

$$\Lambda_2 u_{i,k} = \frac{u_{i,k+1} - 2u_{i,k} + u_{i,k-1}}{h_y^2}.$$

Метод абсолютно устойчив, имеет второй порядок точности по времени и пространственным переменным. Метод экономичен, т.е. количество арифметических операций на шаге по времени пропорционально количеству

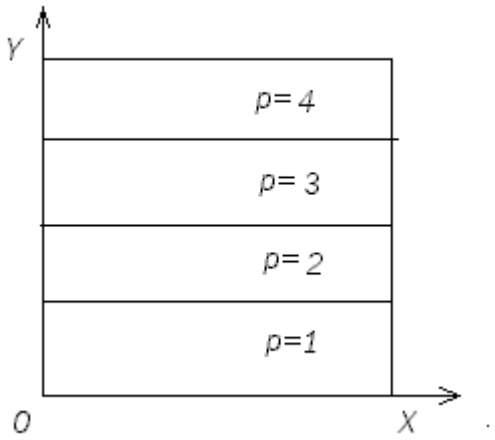

Рис. 8

узлов сетки. Но он требует использования метода прогонки. На первом этапе производится прогонка в направлении оси x, а на втором – в направлении оси

y. Допустим, область решения имеет вид прямоугольника и разбита на одинаковые подобласти вдоль осу *y*. Количество подобластей равно количеству используемых процессоров (рис. 8). Данные распределены по процессорам в соответствии с разбиением на подобласти. Следовательно, все необходимые данные для осуществления прогонки по *x* содержатся в этом процессоре.

Приведем для конкретности обсуждения формулы прогонки в общем виде. Пусть мы имеем следующую систему линейных алгебраических уравнений:

$$A_i y_{i-1} - C_i y_i + B_i y_{i+1} + F_i = 0, \quad i = 1,\ldots,N-1,$$

$$y_0 = \mu_1 y_1 + \nu_1, \quad y_N = \mu_2 y_{N-1} + \nu_2.$$

Коэффициенты $A_i, B_i, C_i, F_i, \mu_1, \mu_2, \nu_1, \nu_2$ считаются заданными. Формулы решения этой системы уравнений с помощью метода правой прогонки имеют следующий вид (номера указывают номера этапов решения):

1) $\alpha_1 = \mu_1, \quad \beta_1 = \nu_1,$

2) $\alpha_{i+1} = \dfrac{B_i}{C_i - A_i\alpha_i}, \quad \beta_{i+1} = \dfrac{A_i\beta_i + F_i}{C_i - A_i\alpha_i}, \quad i = 1,\ldots,N-1,$

3) $y_N = \dfrac{\mu_2\beta_N + \nu_2}{1 - \mu_2\alpha_N},$

4) $y_i = \alpha_{i+1} y_{i+1} + \beta_{i+1}, \quad i = N-1,\ldots,0.$

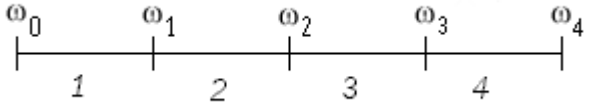

Рис. 9

Здесь α_i, β_i – векторы прогоночных коэффициентов. Легко подсчитать, что количество арифметических операций, требующихся для получения результата, будет порядка *8N*.

Если же необходимо осуществить прогонку в направлении оси *y*, то, поскольку данные для нее находятся на разных процессорах, использование вышеприведенных формул невозможно. В работе [5] предложен алгоритм распараллеливания прогонки, позволяющий получить тот же результат, что и однопроцессорная прогонка. Приведем соответствующие формулы. Разобьем область решения на K частей пропорционально количеству имеющихся

процессоров так, чтобы последний узел предыдущей подобласти совпадал с первым узлом следующей. Значение сеточной функции в этом узле обозначим ω_k, где k – номер левой подобласти (рис. 9). Далее будем считать, что обозначения в каждой подобласти одинаковы и расчеты в каждой из подобластей проводятся на своем процессоре.

Вначале на каждом из *K* процессоров поэтапно выполняем следующие операции:

1) $\alpha_1 = \beta_1 = 0, \quad \gamma_1 = 1,$

2) $\alpha_{i+1} = \dfrac{B_i}{C_i - A_i\alpha_i}, \quad \beta_{i+1} = \dfrac{A_i\beta_i + F_i}{C_i - A_i\alpha_i}, \quad \gamma_{i+1} = \dfrac{A_i\gamma_i}{C_i - A_i\alpha_i},$

$i = 1,\ldots,N-1,$

3) $p_N = r_N = 0, \quad q_N = 1,$

4) $p_i = \alpha_{i+1}p_{i+1} + \gamma_{i+1}, \quad q_i = \alpha_{i+1}q_{i+1},$

$r_i = \alpha_{i+1}r_{i+1} + \beta_{i+1}, \quad i = N-1,\ldots,0.$

По сравнению с предыдущим методом прогонки здесь появляются дополнительные одномерные массивы γ_i, p_i, q_i, r_i. После выполнения вышеприведенных четырех этапов на каждом процессоре мы связываем результаты, полученные на разных процессорах:

5) $A_k p_{N-1}^{(k)}\omega_{k-1} - (C_k - A_k q_{N-1}^{(k)} - B_k p_1^{(k+1)})\omega_k + B_k q_1^{(k+1)}\omega_{k+1} +$

$+ F_k + A_k r_{N-1}^{(k)} + B_k r_1^{(k+1)} = 0, \quad k = 1,\ldots,K-1.$

В этой формуле *k* – номер стыковочного узла. Значения сеточной функции в стыковочных узлах обозначены ω_k. ω_k совпадает с y_N для левого интервала (процессора) с номером *k* и с y_0 для правого интервала (процессора) с номером *k+1* (рис. 10).

Верхний индекс у функций p, q, r (k или $k+1$) показывает, из какого интервала берутся значения функции. Система уравнений для ω_k есть снова система линейных алгебраических уравнений с трехдиагональной матрицей, которая должна решаться методом прогонки. Размер системы уравнений совпадает с количеством используемых процессоров. Она может быть решена на одном процессоре, а затем найденные значения ω_k должны быть разосланы по соответствующим процессорам, либо, предварительно разослав исходные данные по всем процессорам, можно решить данную систему на каждом из процессоров.

После этого на каждом процессоре выполняем последний, шестой этап и получаем значения решения

$$6) \quad y_i = p_i \omega_{k-1} + q_i \omega_k + r_i, \quad i = 0,1,\ldots,N.$$

Если не считать пятый этап, то количество вычислений равно $17N$, где N – полное число узлов. По сравнению с обычной прогонкой количество вычислений выросло чуть больше чем в 2 раза, и на первый взгляд это приемлемая цена за возможность распараллеливания.

В работе [6] экспериментально была исследована эффективность распараллеливания прогонки. Для этого было рассмотрено решение уравнения

$$\frac{\partial^2 \Phi}{\partial y^2} = \rho(x, y)$$

в прямоугольной области с необходимыми граничными условиями. Для решения уравнения использовалась неявная схема, реализация которой

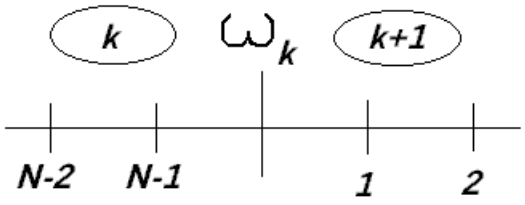

Рис. 10

требовала применения метода прогонки. Были решены 2 задачи, отличающиеся друг от друга только сегментацией полной расчетной области:

а) сегментация в виде P одинаковых вертикальных полос, прогонки ведутся вдоль направления y;

б) сегментация в виде P одинаковых горизонтальных полос, прогонки ведутся по-прежнему вдоль направления y (см. рис. 8).

Результаты расчетов на ЭВМ Siemens RM600 (ИВМиМГ СО РАН, Новосибирск) на сетке (3000,1000) узлов приведены в таблице.

Тип сегмен-тации	Запуск						
	Однопро-цессор-ный	Двухпроцессор-ный		Четырехпроцессорный			
	p=1	p=1	p=2	p=1	p=2	p=3	p=4
а	2,81	2,20	1,87	1,26	0,55	0,57	1,09
б	2,81	15,76	9,26	9,03	7,72	6,83	12,30

В таблице приведены времена (с) выполнения расчетов на каждом из использованных процессоров. Сразу бросается в глаза, что распараллеливание прогонки крайне неэффективно, время ее выполнения не только не уменьшается, а, наоборот, существенно превышает время выполнения на одном процессоре. Кроме того, интересно обратить внимание на то, что одинаковое количество операций на одинаковых процессорах выполняется за разное время.

Так как же быть с прогонкой? Использовать или не использовать? Здесь все зависит от конкретной задачи. Если можно обойтись без нее, то, конечно, ее не надо применять.

Но вернемся к уравнению теплопроводности. Очевидно, явные схемы для нее легко распараллеливаются. Вычисления на каждом шаге по времени на процессорах выполняются независимо друг от друга, обмен граничными значениями между соседними процессорами выполняется в конце каждого шага. А неявная схема требует использования прогонки, количество арифметических операций, приходящихся на каждый узел, больше, чем в явной схеме. Но она абсолютно устойчива, позволяет считать с существенно большим временным шагом, чем явная схема, и поэтому количество пересылаемой между процессорами информации существенно меньше. И здесь вступает в игру фактор, обычно не рассматриваемый в методах вычислений, – архитектура машины. Как связаны между собой процессоры? Какова скорость пересылки данных между процессорами? Каково быстродействие процессоров и количество операций, выполняемых на них между обменами с другими процессорами? И вот эти, казалось бы, чисто технические вопросы, могут играть решающую роль при выборе того или иного алгоритма. И может случиться, что неявные алгоритмы с применением распараллеливания прогонки окажутся более эффективными, чем более простые для распараллеливания явные.

Существует мнение, что при разработке алгоритма решения какой-либо большой задачи надо сразу ориентироваться на решение ее на многопроцессорной машине, т.е. сразу создавать параллельные алгоритмы. На мой взгляд, это чистая фантастика. Ведь алгоритмы появляются при решении каких-либо практических задач, и уже в самой задаче порой лежит возможность или невозможность разделения ее решения на несколько независимых подзадач. И перед специалистом-вычислителем возникает порой небогатый выбор между явными и неявными алгоритмами, о преимуществах и недостатках которых уже сказано выше. Чтобы глубже понять задачу, построить эффективный алгоритм, необходимо провести много предварительных тестовых расчетов, и в этом случае вешать на задачу еще и сложности пропуска ее на многопроцессорных ЭВМ крайне неэффективно. Поэтому решение практической задачи всегда будет начинаться с однопроцессорного алгоритма.

Список литературы

1. *Березин Ю.А., Вшивков В.А.* Метод частиц в динамике разреженной плазмы. – Новосибирск: Наука, 1980. – 94 с.
2. *Хокни Р., Иствуд Дж.* Численное моделирование методом частиц. – М.: Мир, 1987. – 638 с.
3. *Григорьев Ю.Н., Вшивков В.А.* Численные методы "частицы-в-ячейках". – Новосибирск: Наука, 2000. – 184 с.
4. *Вшивков В.А., Краева М.А., Малышкин В.Э.* Параллельная реализация метода частиц //Программирование. – 1997. – № 2. – С. 39-51.
5. *Яненко Н.Н., Коновалов А.Н., Бугров А.Н., Шустов Г.В.* Об организации параллельных вычислений и «распараллеливании прогонки» //Численные методы механики сплошной среды. – 1978. – № 7. – С. 136-139.
6. *Вшивков В.А., Тарнавский Г.А., Неупокоев Е.В.* Параллелизация алгоритмов прогонки: многоцелевые вычислительные эксперименты //Автометрия. – 2002. – Т.4. (в печати).