ПРИМЕНЕНИЕ МЕТОДА ВСТРЕЧНЫХ ПРОГОНОК ДЛЯ СИНТЕЗА ПАРАЛЛЕЛЬНОГО АЛГОРИТМА РЕШЕНИЯ СЕТОЧНЫХ УРАВНЕНИЙ ТРЕХДИАГОНАЛЬНОГО ВИДА

Д.Л. Головашкин
Институт систем обработки изображений РАН
Самарский государственный аэрокосмический университет

Аннотация

Работа посвящена синтезу и исследованию параллельных алгоритмов (с локальными коммуникациями) для решения сеточных уравнений трехдиагонального вида. В данном классе алгоритмов разработан оптимальный по времени коммуникаций, простоя и объему занимаемой памяти алгоритм. Представлена экспериментальная методика повышения ускорения параллельного вычислительного процесса, порожденного оптимальным алгоритмом.

Введение

Моделирование физических процессов посредством численного решения дифференциальных уравнений находит все более широкое применение в различных отраслях науки. Этому способствует развитие вычислительной техники и численных методов, ориентированных на решение таких уравнений. Наиболее распространенные методы (конечных разностей и Галеркина) сводят дифференциальную задачу к системе линейных алгебраических уравнений (СЛАУ) вида $Ax=b$, где матрица A - ленточная. К прямым методам решения такой системы относят различные варианты прогонки и редукции. С появлением вычислительных систем, реализующих параллельные вычисления, связано создание параллельных алгоритмов, основанных на этих методах. Настоящая статья посвящена построению и исследованию параллельных алгоритмов, ориентированных на кластерную архитектуру ЭВМ и основанных на методе встречных прогонок для решения сеточных уравнений трехдиагонального вида. Известно несколько параллельных алгоритмов, основанных на методе трехдиагональной прогонки. В [1] представлен алгоритм, использующий распределение матрицы системы между задачами (в терминах модели канал/задача [2]), позволяющее каждой задаче алгоритма решать свою подсистему, выражая решение через значения на границах области данных, входящих в задачу. Для вычислительных систем с различной архитектурой синтезировано целое семейство параллельных алгоритмов, основанных на этом принципе. Недостатком такого подхода является наличие глобальных коммуникаций, нежелательных для кластерной архитектуры. В [3] предложено решение поставленной задачи с использованием метода редукции, к сожалению также не свободное от глобальных коммуникаций. Принципиально другой подход содержится в [4]. Автор алгоритма отказался от идеи одновременного решения одной СЛАУ несколькими задачами, применив вместо декомпозиции данных функциональную декомпозицию. Все задачи алгоритма одновременно решают разные подсистемы разных СЛАУ, общаясь посредством локальных коммуникаций. Использование функциональной декомпозиции ограничивает область применения данного подхода, не позволяя решать одномерные задачи математической физики. Другим недостатком алгоритма является неполная загруженность задач: на начальном и конечном этапах расчета часть задач не производит вычислений, ожидая получения необходимых данных. Вариант алгоритма, исключающий простои, требует передачи вдвое большего объема данных из-за неудачной декомпозиции сеточной области. Тем не менее, этот подход представляется наиболее перспективным. В данной работе он применяется к распараллеливанию метода встречных прогонок, что позволяет использовать его для решения одномерных задач математической физики, снизить вдвое время ожидания и улучшить декомпозицию сеточной области для варианта без простоев.

Описание параллельных алгоритмов

Алгоритм 1.

Рассмотрим простейший алгоритм, состоящий из двух задач. Функциональная декомпозиция задается спецификой метода встречных прогонок, в котором прогоночные коэффициенты (α, β и η, ζ) вычисляются с двух сторон по направлению к центру матрицы A [5]. В последовательном алгоритме этот прием используется, когда желают ограничиться нахождением не всего вектора x, а лишь его части. Так как вычисления двух пар коэффициентов прогонки независимы (информационно, логически и конкуренционно), то их можно осуществлять параллельно. Произведем разбиение одномерной сеточной области ω_1 на ω_1^1 и ω_1^2 (рис. 1). Вычисления разделим на три этапа, соответствующие прямому (рис. 1 а), обратному ходу (рис.1 в) встречных прогонок и обмену данными между задачами (рис. 1 б).

Во время прямого хода прогонок первая задача находит прогоночные коэффициенты α, β, вторая задача η, ζ. На этом этапе алгоритма (как и на остальных этапах) задачи производят независимые вычисления, следовательно, могут исполняться параллельно. Далее, первая задача передает второй пару чисел α, β, необходимых для запуска обратного хода прогонки во второй задаче, которая в это время производит аналогичную передачу, необходимую первой задаче. На третьем этапе обе задачи одновременно реализуют обратный ход прогонок, находя решение

СЛАУ. Ускорение такого алгоритма можно оценить как:

$$k = \frac{\tau_a(N)}{\mathrm{S}\,\tau_a(N) + 2\tau_k},$$

где N - размерность СЛАУ,
$\tau_a(N)=3N\tau_\times+(2N+1)\tau_\div+3N\tau_+$ - время расчета по последовательному алгоритму (τ_+ - длительность одной операции сложения (вычитание понимаем как сложение), $\tau_\times$ - длительность одной операции умножения, $\tau_\div$ - длительность одной операции деления), $\tau_к$ - время передачи или приема (считаем их равными) одного пакета по сети. При пакетной передаче данных время передачи разного объема информации будет оставаться константой до тех пор, пока объем передаваемого массива не превысит размер пакета. Здесь, и в дальнейшем, это условие соблюдается.

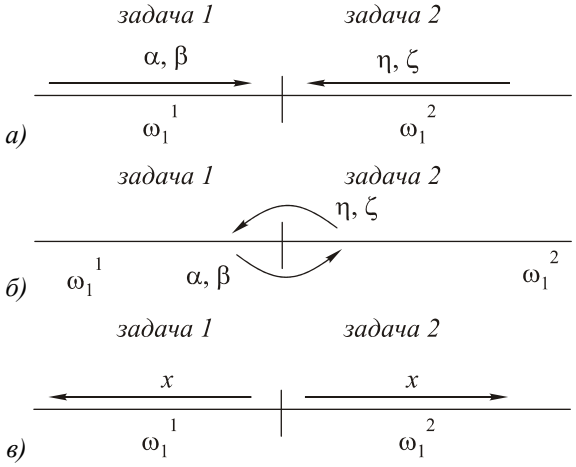

Рис. 1. Этапы вычислений по параллельному алгоритму, состоящему из двух задач и реализующему метод встречных прогонок для одной СЛАУ
а - прямой ход прогонок; б - обмен данными между задачами; в - обратный ход прогонок

Очевидно, при $N\to\infty$ $k\to2$. К сожалению, этот подход нельзя развить для большего количества задач, если сеточная область ω одномерна. Функциональная декомпозиция, как правило, ограничивает масштабируемость алгоритма. В нашем случае, алгоритм 1 не обладает свойством масштабируемости [2].

Алгоритм 2.

Для двумерной области ω_2 это ограничение снимается. Применительно к ней будем говорить о прогонке по строкам (продольной прогонке) и столбцам (поперечной прогонке) сеточной области, как это принято в методах расщепления и переменных направлений [6].

Если алгоритм состоит из двух задач, то модификации, по сравнению с предыдущим алгоритмом, не требуется, каждая СЛАУ в продольном направлении решается одновременно двумя задачами. Ускорения такого алгоритма и известного алгоритма из [4] составят

$$k_2 = \frac{2N\tau_a(N)}{N\tau_a(N) + 2N\tau_k}$$

$$\text{и } \tilde{k}_2 = \frac{2N\tau_a(N)}{N\tau_a(N) + 2N\tau_k + \frac{1}{2}\tau_a(N)}.$$

Третье слагаемое в знаменателе последней формулы есть время ожидания для задачи в стандартном алгоритме. Сформулированный выше вариант алгоритма лишен ожиданий.

Для алгоритма из четырех задач производится следующее разбиение сеточной области (рис. 2).

На рис. 2 (а-в) представлены начальные этапы вычислений по алгоритму. Задачи 1, 4 (рис. 2 а) начинают прямой ход прогонок для строки 1 и передают прогоночные коэффициенты задачам 2, 3, которые на первом этапе простаивают.

Далее не будем специально говорить об обмене данными между задачами, подразумевая, что перед прямым ходом они принимают прогоночные коэффициенты, после прямого хода передают их. Перед обратным ходом принимают значение сеточной функции, после - передают его. Первая и последняя задачи начинают прямой ход без принятия прогоночных коэффициентов, а задачи под номерами $L/2$ и $L/2+1$ (L - общее число задач) начинают обратный ход принятием прогоночных коэффициентов (друг от друга).

На втором этапе (рис. 2 б) задачи 2, 3 продолжают прямой ход для строки 1, задачи 1, 4 начинают прямой ход для строки 2. Третий этап (рис. 2 в) характеризуется простоем задач 1,4 и началом обратного хода прогонок для строки 1 задачами 2, 3. Четвертый этап (рис. 2 г) - прямой ход, задачи 1, 4 осуществляют его для строки 3, задачи 2, 3 для строки 2. Пятый этап (рис. 2 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 этапе задачи 1, 4 будут простаивать, задачи 2, 3 начнут обратный ход. Заключительный этап алгоритма характеризуется простоем задач 2, 3, в то время, как задачи 1, 4 завершат обратный ход прогонок для последней строки сеточной области. Прогонка по столбцам осуществляется каждой задачей самостоятельно, эти вычисления независимы. Пусть размер сеточной области ω_2 - $N\times N$ точек, тогда ускорение данного алгоритма и ускорение стандартного алгоритма оценим величинами

$$k_4 = \frac{2N\tau_a(N)}{\frac{1}{4}\mathrm{N}\,\tau_a(N) + 4N\tau_k + \frac{1}{4}\tau_a(N)}$$

$$\text{и } \tilde{k}_4 = \frac{N^2\tau_a(N)}{\frac{1}{2}\mathrm{N}\,\tau_a(N) + 4N\tau_k + \frac{3}{4}\tau_a(N) + 4\tau_k},$$

где $\frac{1}{4}\tau_a(N)$ - время ожидания в разработанном алгоритме, $\frac{3}{4}\tau_a(N)+4\tau_к$ - время ожидания в известном алгоритме.

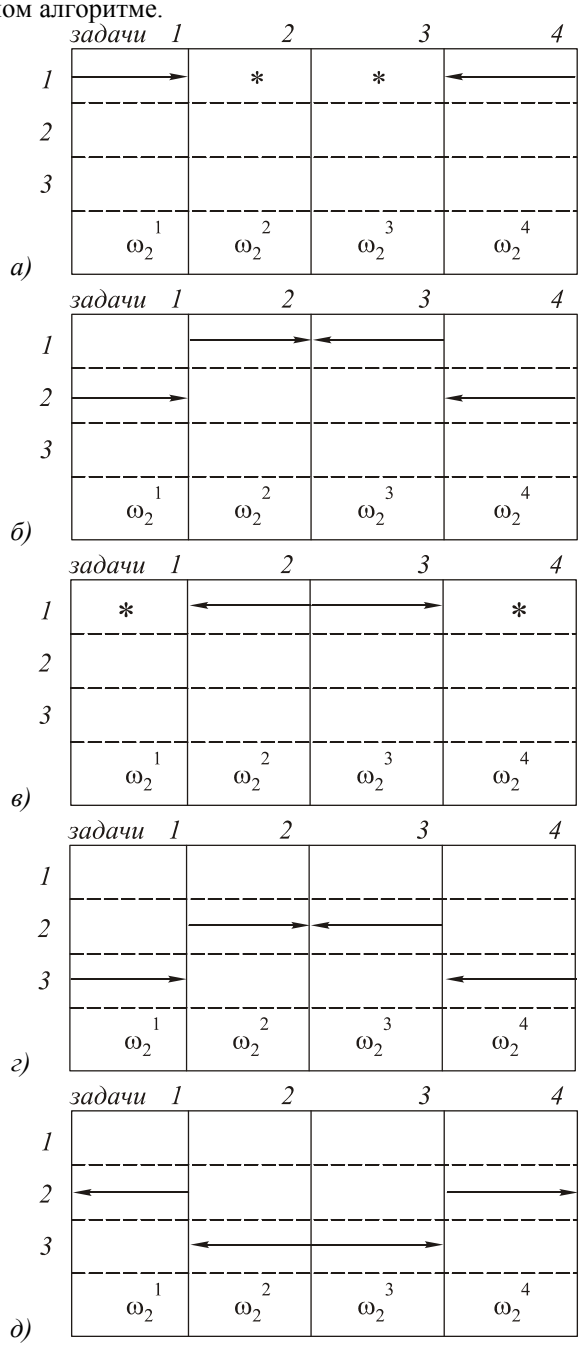

Рис. 2. Этапы вычислений по параллельному алгоритму, состоящему из четырех задач, для прогонки по строкам на ω_2. а,б,г - прямой ход прогонок; в,д - обратный ход прогонок. Задача, помеченная символом «», простаивает*

Очевидно развитие алгоритма для произвольного четного числа задач. Для L задач ускорения составят

$$k_L^{четн.} = \frac{2N\tau_a(N)}{\frac{2N}{L}\tau_a(N)+4N\tau_k+\frac{(L-2)}{2L}\tau_a(N)+(L-4)\tau_k}$$

$$\tilde{k}_L = \frac{2N\tau_a(N)}{\frac{2N}{L}\tau_a(N)+4N\tau_k+\frac{(L-1)}{L}\tau_a(N)+2(L-2)\tau_k}.$$

Время ожидания снижено более чем вдвое по сравнению со стандартным алгоритмом, остальные характеристики одинаковы. Следовательно, ускорение разработанного алгоритма превышает ускорение ранее известного алгоритма. В случае, когда алгоритм из двух задач производит вычисления на области, содержащей две строки, время вычислений сократится в полтора раза.

Для алгоритма с нечетным числом задач результат сравнения окажется скромнее. Поделим область ω_2 так, как это показано на рис. 3.

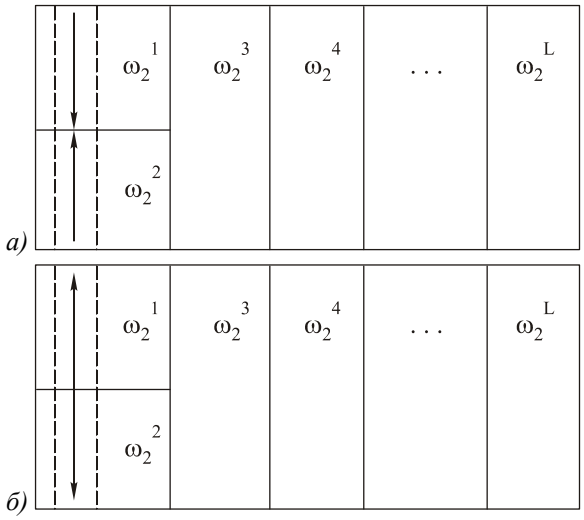

Рис. 3. Этапы вычислений по параллельному алгоритму, состоящему из нечетного числа задач, для прогонки по столбцам. а - прямой ход прогонки; б - обратный

Прогонка по строкам осуществляется так же, как при четном числе задач, ведь с каждой строкой работает L-1 задача. Появляется необходимость в распараллеливании прогонки по столбцам между двумя первыми задачами (рис.3). При реализации этой прогонки время коммуникаций увеличивается по сравнению с алгоритмом для четного числа задач.

Алгоритм 3.

Осуществим двумерную декомпозицию сеточной области ω_2 для построения параллельного алгоритма метода встречных прогонок. Очевидный вариант такой декомпозиции на четыре подобласти и этапы алгоритма представлены на рис. 4.

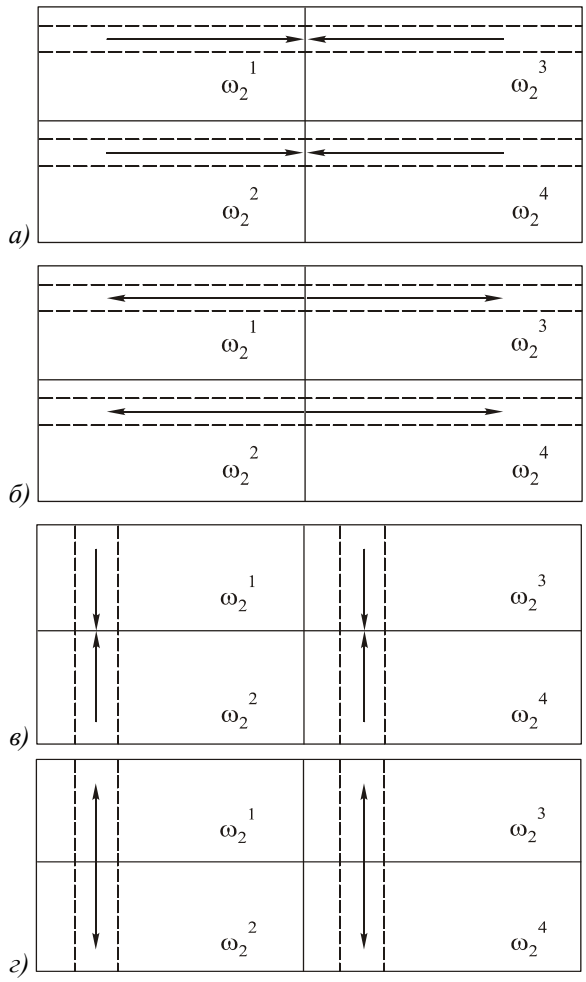

Рис. 4. Этапы вычислений по параллельному алгоритму, состоящему из четырех задач, с двумерной декомпозицией сеточной области. а - прямой ход прогонок по строкам, б - обратный ход прогонок по строкам, в - прямой ход прогонок по столбцам, г - обратный ход прогонок по столбцам.

Четырехзадачные алгоритмы с двумерной декомпозицией сеточной области (представленный на рис. 4 и известный) обеспечивают следующие ускорения

$$k_4^{2D} = \frac{2N\tau_a(N)}{\frac{1}{2}N^2\tau_a(N) + 2N\tau_k}, \quad \tilde{k}_4^{2D} = \frac{2N\tau_a(N)}{\frac{1}{2}N\tau_a(N) + 6N\tau_k}.$$

В отличии от четырехзадачного алгоритма (как изложенного выше, так и из [4]) с одномерной декомпозицией ω_2, алгоритм с двумерной декомпозицией не содержит простоев и оперирует вдвое меньшим объемом данных при установке коммуникаций между задачами. По сравнению с известным алгоритмом [4], использующим двумерную декомпозицию, алгоритм 3 тратит втрое меньше времени на коммуникации между задачами.

Декомпозиция области и действия по алгоритму из восьми задач иллюстрируются на рис. 5. Особенность разделения ω_2 между задачами состоит в принадлежности одной задаче двух квадратных подобластей. Их взаимное расположение позволяет распараллеливать вычисления прогонки как по строкам, так и по столбцам области.

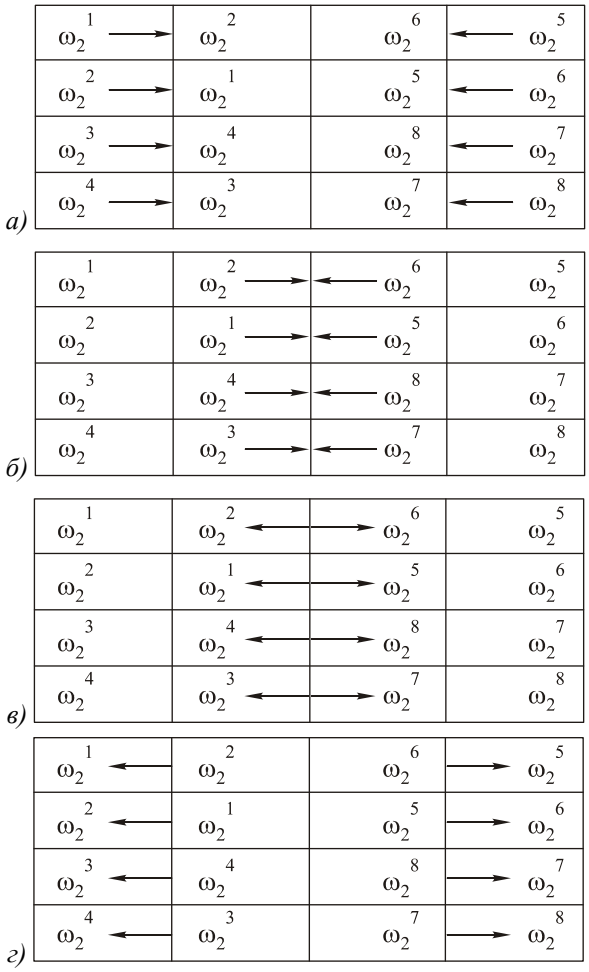

Рис. 5. Этапы вычислений прогонок по строкам (восьмизадачный алгоритм с двумерной декомпозицией сеточной области). а, б - прямой ход прогонок по строкам, в, г - обратный ход прогонок по строкам

На первом этапе алгоритма (рис. 5 а) производится прямой ход прогонок по строкам в левой и правой четвертях ω_2 . На втором этапе (рис. 5 б) прямой ход прогонок по строкам заканчивается, вычисления из левой четверти переносятся на одну подобласть вправо, из правой четверти на одну подобласть влево. Третий и четвертый этапы алгоритма (рис. 5 г, д) - обратный ход прогонок по строкам. Аналогично производятся прогонки по столбцам.

Сравним ускорения описанного алгоритма и алгоритма с двумерной декомпозицией ω_2 из [4]

$$k_8^{2D} = \frac{2N\tau_a(N)}{\frac{1}{4}N\tau_a(N) + 3N\tau_k}, \quad \tilde{k}_8^{2D} = \frac{2N\tau_a(N)}{\frac{1}{4}N\tau_a(N) + 7N\tau_k}.$$

Выигрыш по времени коммуникаций составляет 2,5 раза. По сравнению с алгоритмами, основанными на одномерной декомпозиции области, выигрыш по времени коммуникаций меньше (1,25 раза), за то отсутствуют простои, присущие этим алгоритмам.

Двумерная декомпозиция ω_2 при $L=2^q$ и $q>3$ производится аналогично разбиению на рис. 5. При этом ускорения разработанного и известного алгоритмов составят

$$k_L^{2D} = \frac{2N\tau_a(N)}{2N\tau_a(N)/L + 4(1-2/L)N\tau_k},$$

$$\tilde{k}_L^{2D} = \frac{N\tau_a(N)}{2N\tau_a(N)/L + 8(1-1/L)N\tau_k}.$$

Выигрыш по времени коммуникаций оценивается величиной $2(L\text{-}1)/(L\text{-}2)$. По сравнению с алгоритмом 2 этот выигрыш окажется значительно меньше и составит $\frac{1}{1-2/L}$, но при этом алгоритм 3 полностью лишен простоев.

Алгоритм 3 разработан для числа задач L - степени двойки. Является ли это ограничение существенным? Отметим локальность коммуникаций алгоритма 3. Каждая задача имеет $\log_2 L$ соседей, что соответствует степени связности гиперкуба с L вершинами. Вычисления по предложенному алгоритму разумно производить на ЭВМ с кластерной архитектурой, процессоры которой являются вершинами гиперкуба. Такая архитектура наиболее распространена в настоящее время, следовательно, ограничение алгоритма по числу задач несущественно.

Ранее, имея в своем распоряжении алгоритмы из [4], исследователь сталкивался с проблемой выбора декомпозиции сеточной области. Вариант с одномерной декомпозиции обеспечивал меньший объем коммуникаций, вариант с двумерной декомпозицией был свободен от простоев задач. Алгоритм 3, представленный в данной работе, для квадратной сеточной области обеспечивает, без простоев, наименьшее время коммуникации, по сравнению с известными параллельными алгоритмами из [4].

Численное исследование ускорения параллельных вычислительных процессов, порожденных разработанными алгоритмами

При аналитическом анализе ускорения параллельных алгоритмов сложно учесть все особенности конкретной вычислительной системы (работа с памятью, сетью и т.д.). Становится важным исследование ускорения параллельных вычислительных процессов (ПВП), порожденных синтезированными алгоритмами, позволяющее выявить ранее неучтенные эффекты, присущие практике параллельных вычислений. В дальнейшем, с учетом найденных закономерностей, можно повысить ускорение параллельного алгоритма, соответственно его модифицировав.

Изучим поведение ускорения s ПВП, порожденного алгоритмом 1, меняя размерность сеточной области (рис. 6). Все вычислительные эксперименты проводились на кластере из двух двухпроцессорных ЭВМ Pentium III, связанных сетью Mirinet.

При N=1000 (рис. 6 а) ускорение оказалось меньше 1, время вычислений по параллельному алгоритму больше времени вычислений по последовательному алгоритму.

Выясним, почему это происходит.

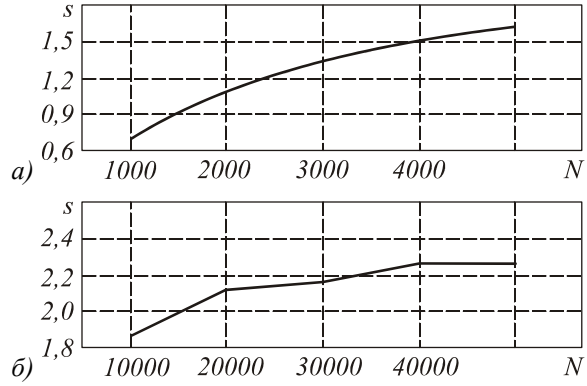

Рис. 6. Зависимость ускорения ПВП, порожденного алгоритмом 1, от размерности сеточной области, для разных диапазонов N

Длительность арифметических вычислений оценим величиной $O(N)$, длительность коммуникаций - константа $\tau_к$. Из определения функции «О большое» $O(N) \le C \times N$, где C - положительная константа. Для $N' < \frac{2\tau_k}{C}$ ускорение ПВП окажется меньше 1, что соответствует экспериментальным данным. Так как правая часть неравенства - константа, то начиная с некоторого $N'' > N'$ ускорение превысит 1 и по сравнению с последовательным вычислительным процессом получим ожидаемый выигрыш по времени (рис.6 а). Сверху ускорение ограничивается законом Амдала [2], по которому $s \le \frac{1}{q + (1-q)/P}$, где q - отношение времени исполнения части процесса, которую невозможно распараллелить, ко времени исполнения всего последовательного вычислительного процесса. Всегда найдется вычислительная операция, которую распараллелить нельзя, но, даже положив q=0, ускорение останется ограничено числом процессоров P вычислительной системы. Для исследуемого ПВП ускорение, согласно закону Амдала, не должно превышать 2. Однако, как следует из рис. 6б, при N=20000, s>2. Для изучения этого эффекта обратимся к табл.1.

Таблица 1

Зависимость объема оперативной памяти, занимаемой последовательным вычислительным процессом, от размерности ω_1

	Размерность сеточной области (N/1000)									
	1	2	3	4	5	10	20	30	40	50
Объем занимаемой оперативной	23,3	46,8	70,1	93,5	117,9	234,7	468,5	703,2	937,5	1171,7

памяти (кБ)										

Начиная с размерности N=2000 (именно с нее ускорение превышает 2), объем памяти, занимаемый последовательным вычислительным процессом, превышает кэш процессора (256кБ). Следовательно, последовательный и параллельный процессы оперируют областями памяти, характеризующимися разными скоростями доступа к данным. Область памяти, занимаемая последовательным процессом при N=50000, выходит за размеры кэша 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 (не важно 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.

Исследуем ускорение ПВП, порожденного двухзадачным алгоритмом 2. Предварительно модифицируем алгоритм, учитывая особенности пакетной передачи данных по сети. Выберем диапазон N таким, чтобы объем всех передаваемых данных не превышал объем пакета (рис. 7). Организуем обмен между задачами не парой прогоночных коэффициентов, относящихся к одной строке области ω_2, а коэффициентами сразу всех строк. Тогда объем коммуникаций уменьшится в N раз по сравнению с алгоритмом 2 и ускорение модифицированного алгоритма достигнет величины

$$k_2^{вект.} = \frac{2N\tau_a(N)}{N\tau_a(N)+2\tau_k}$$

Такой алгоритм назовем векторным, прежний алгоритм - скалярным.

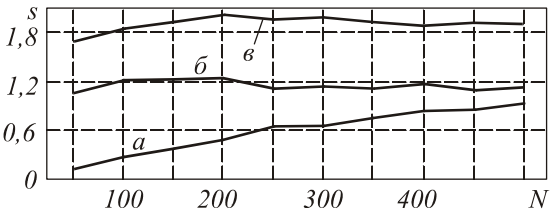

Рис. 7. Ускорения ПВП, порожденных алгоритмом 2.
а - ускорения для скалярного ПВП,
б - для векторного ПВП, в - векторного ПВП,
при соразмерном по занимаемой памяти
последовательном вычислительном процессе

В исследуемом диапазоне значений N ускорение скалярного ПВП растет, не достигая 1. Ускорение векторного ПВП, как и ожидалось, превышает ускорение скалярного. При N=50 отношение $\frac{s^{вект.}}{s^{скал.}}$=7,19; при N=500 $\frac{s^{вект.}}{s^{скал.}}$=1,22. Для выяснения причины, по которой ускорение векторного ПВП не достигает предельного значения, установленного законом Амдала, опять обратимся к анализу объема оперативной памяти, занимаемой процессами. За уменьшение объема передаваемых данных приходиться платить увеличением используемой области памяти. Если одна задача скалярного алгоритма хранит N^2+2N чисел, то одна задача векторного алгоритма - 3N^2. Ветвь векторного ПВП уже при N=150 занимает 263,67 кБ, что превышает кэш. При такой размерности ускорение равно 1,22. С увеличением N ускорение векторного ПВП не возрастает. Это не соответствует аналитической зависимости ускорения алгоритма от размерности задачи. Из-за увеличения времени доступа к памяти при исполнении ПВП, по сравнению с последовательным вычислительным процессом, векторный алгоритм имеет смысл только при N<150 (разумеется, для данной вычислительной системы), что серьезно ограничивает область его применения. Для подтверждения предположения о влиянии объема занимаемой памяти на ускорение ПВП, используем при определении ускорения векторного ПВП длительность последовательного процесса, занимающего тот же объем памяти, что и векторный (рис. 7). Ускорение векторного ПВП, определенное таким образом, достигает теоретического предела.

В качестве компромисса между скалярным и векторным способами передачи данных выберем блочный способ. Ограничим размером блока M $(M \le N)$ количество строк, для которых прогоночные коэффициенты передаются в одном пакете. При этом часть пакета не будет заполнена полезной информацией, зато объем используемой памяти сократится до N^2+2$N\times M$. Произведение 2$N\times M$ определяет объем памяти, отводящийся под прогоночные коэффициенты. На рис. 8 представлена зависимость ускорения блочного ПВП от M при N=500.

Применение блочного алгоритма позволило повысить ускорение с s=0,93 для скалярного ПВП и s=1,14 векторного ПВП до s=1,41 блочного ПВП (при M=40).

Автору не удалось аналитически выразить зависимость $s(M)$, поэтому, рекомендую поиск оптимального M производить экспериментально. Например, при разностном решении дифференциального уравнения параболического типа, следует, ограничившись несколькими шагами по времени, подобрать оптимальную величину M, для которой, затем, произвести необходимые расчеты по решению разностного уравнения с требуемым числом шагов по времени.

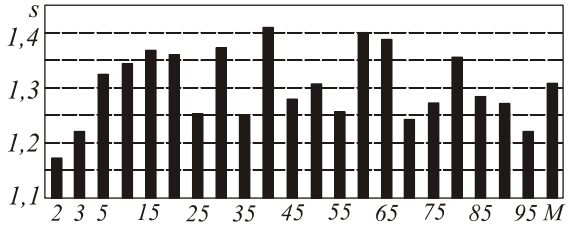

Рис. 8. Зависимость ускорения блочного ПВП от количества строк, для которых прогоночные коэффициенты передаются в одном пакете

При исследовании зависимости *s(N)* вышеизложенный подход применять нецелесообразно в силу большой трудоемкости. Для каждого значения *N* придется с помощью серии вычислительных экспериментов подбирать оптимальный размер блока, варьируя *M* в пределах от 1 до *N*, производя постановку *N×K* вычислительных экспериментов для определения вида зависимости *s(N)* по *K* отсчетам. На рис. 9 представлена зависимость ускорения блочного ПВП от размерности сеточной области ω_2 для таких *M*, при которых объем памяти, занимаемой прогоночными коэффициентами, не выходит за пределы кэша (для *M*+1 это условие уже не соблюдается). Такой подход к выбору величины *M* представляется автору наиболее разумным, если экспериментальное исследование зависимости *s(M)* затруднено.

При нем ускорение должно оставаться постоянной величиной, ведь отношение времени арифметических операций ко времени вычислений, с увеличением *N* (до тех пор, пока объем передаваемых по сети данных не превзойдет объем пакета), не изменяется. Как видно из рис. 9, ускорение блочного ПВП можно считать постоянным и равным 1,4. Возможно, экспериментально исследуя *s(M)* при *N*=4000, удастся существенно повысить ускорение блочного ПВП, однако в этом нет необходимости. При возрастании *N* ускорение скалярного ПВП стремится к пределу, установленному законом Амдала (рис. 9). Для любой строки ω_2 с увеличением размерности области, время коммуникаций остается постоянным, а длительность выполнения арифметических операций возрастает.

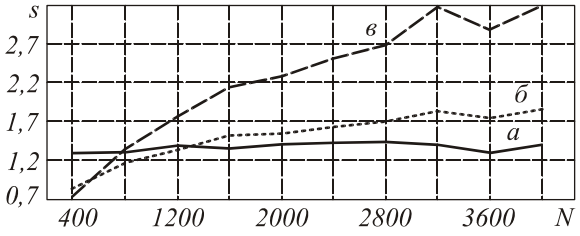

Рис. 9. Зависимость ускорения ПВП от размерности сеточной области. Кривая а - зависимость для блочного ПВП, порожденного двухзадачным алгоритмом 2, б - для скалярного, порожденного двухзадачным алгоритмом 2, кривая в - для скалярного ПВП, порожденного четырехзадачным алгоритмом 3

Следовательно, для значений $N<N^{'}$ (в данном случае $N^{'}$=1200) имеет смысл использовать блочный алгоритм, при $N>N^{'}$ - скалярный. Не следует полагать, что при небольших значениях *N* организация параллельных вычислений неуместна в виду малого размера сеточной области. Решая параболическое уравнение на сетке, содержащей число отсчетов по времени, превосходящее *N* на несколько порядков, при малой размерности ω_2, можно столкнуться с ситуацией, когда время работы последовательной программы окажется неприемлемо большим. Тогда, даже для *N*=50, следует обратиться к параллельному алгоритму. Так как при малых *N* блочный алгоритм предпочтительнее скалярного, то в таких ситуациях разумнее использовать именно его.

Отметим, что все известные алгоритмы из [4] подразумевают использование бо'льших объемов памяти, чем сформулированные здесь алгоритмы, которые и в этом смысле оказываются предпочтительнее.

Подтвердим экспериментально, что скалярный ПВП, порожденный четырехзадачным алгоритмом 3, развивает ускорение, превышающее единицу. Уже для *N*=800 (рис. 9) его ускорение выше, чем у скалярного и векторного ПВП, порожденных двухзадачным алгоритмом 2. При *N*=7200 исследуемый ПВП достигает предельного ускорения (рис.10).

Для алгоритма 3 имеет смысл та же блочная модификация, что и для алгоритма 2.

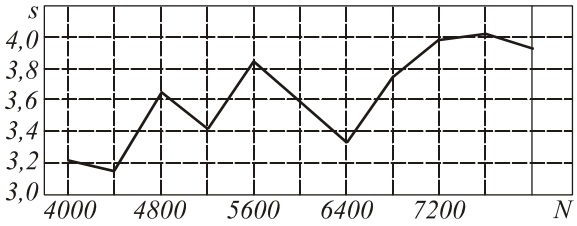

Рис. 10. Зависимость ускорения ПВП, порожденного четырехзадачным алгоритмом 3, от размерности сеточной области N

Выводы

В работе сформулированы параллельные алгоритмы для решения СЛАУ трехдиагонального вида методом встречных прогонок. Выбран оптимальный среди известных и представленных в настоящей работе алгоритмов с локальными коммуникациями между задачами. Предусматривая пересылку меньшего количества данных, будучи избавленным от простоев и требуя меньший объем памяти, он обеспечивает максимальное ускорение. Представлена экспериментальная методика, позволяющая повысить ускорение ПВП, используя блочную структуру коммуникаций.

Благодарность

Автор благодарит Владимира Александровича Шустова за помощь в интерпретации результатов вычислительных экспериментов.

Литература

1. Яненко Н.Н., Коновалов А.Н., Бугров А.Н., Шустов Г.В. Об организации параллельных вычислений и «распараллеливание» прогонки / Численные методы механики сплошной среды / ИТПМ СО АН СССР.- Новосибирск, 1978, т.9, с.139-146.
2. Braunl T. The art of parallel programming. - Prentice Hall International (UK) Limited, 1993. 378p.
3. Ортега Джеймс М. Введение в параллельные и векторные методы решения линейных систем /

перевод с англ. Икрамова Х.Д., Капорина И.Е.; под ред. Х.Д. Икрамова. - М.: Мир, 1991.-364.
4. Миренков Н.Н. Параллельные алгоритмы для решения задач на однородных вычислительных системах // Вычислительные системы. ИМ СО АН СССР. - Новосибирск, 1973, вып. 57, с. 3-32
5. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений / М.: Наука.- 1978. - 561 с.
6. Самарский А.А. Теория разностных схем / М.: Наука, 1989. -614 с.