УДК 519.6

Оптимизация компактной схемы Гаусса для многоядерных процессоров

В. О. Мищенко, Б. В. Паточкин
Харьковский национальный университет имени В.Н. Каразина, Украина

Какой экономии времени расчётов можно достичь за счёт распараллеливания стандартных вычислений на многоядерном процессоре? Эффект во многом зависит от рациональности обращения параллельных процессов к общей памяти через кэш. В статье развивается известный подход к улучшению локализации памяти, известный как «тайлинг». Применительно к задачам класса решения линейных систем разработан экспериментальный метод оптимизации параметров управления тайлингом при распараллеливании алгоритмов. Этим методом получена параллельная модификация компактной схемы метода Гаусса решения линейных систем, ориентированная на современную архитектуру персональных компьютеров.

Ключевые слова: *многоядерный процессор, параллельный алгоритм, оптимизация по времени, кэш памяти, тайлинг, компактная схема метода Гаусса.*

Якої економії часу, потрібного на розрахунки, можна досягти за рахунок розпаралелювання стандартних обчислень на багатоядерних процесорах? Ефект багато в чому залежить від раціональності звернення паралельних процесів до спільної пам'яті через кеш. У статті розвивається відомий підхід до поліпшення локалізації пам'яті, відомий як «тайлінг». У застосуванні до таких задач як розв’язок лінійних систем створено експериментальний метод оптимізації параметрів управління тайлінгом при розпаралелюванні алгоритмів. За його допомоги отримана паралельна модифікація компактної схеми методу Гауса розв’язку лінійних систем, що зорієнтована на сучасну архітектуру персональних комп'ютерів.

Ключові слова*: багатоядерний процесор, паралельний алгоритм, оптимізація часу, кеш пам'яті, тайлінг, компактна схема методу Гауса.*

What time-saving for calculations can be achieved by parallel transformation of the regular computations for multi-core processors? The effect depends largely on the rationality of concurrent processes requests to a shared memory through the cache. In this paper we expand a well-known approach of memory localization improving, known as "tiling". Experimental method of tailing control parameters optimization has been developed for solving problems of linear systems with parallel algorithms. Using this method we obtained a parallel modification of the compact scheme of the Gauss algorithm for solving linear systems, focused on modern architecture of personal computers.

Key words: *multi-core processor, parallel algorithm, the optimization time, the cache memory, tiling, Gaussian compact scheme.*

1. Введение и постановка задачи

В настоящее время при разработке программного обеспечения проблема ускорения машинных вычислений на основе распараллеливания алгоритмов выступает, прежде всего, как техническая. Математические результаты в предположениях «неограниченного параллелизма» не могут, как правило, служить практическим руководством для программистов [1]. Для достижения высокого эффекта от параллельности приходится учитывать архитектуру

используемого компьютера, особенности операционной системы, технику программирования и сочетать математические идеи с экспериментом.

Не являются в этом отношении исключением и вычислительные методы дискретных особенностей (МДО), которые при трёхмерной постановке задач моделирования электродинамических явлений (см., например, [2-5]) достаточно громоздки в вычислительном отношении для того, чтобы при их реализации можно было обойтись без распараллеливания вычислений [6-8]. При этом распределённое решение больших (и хорошо обусловленных в применениях МДО) систем линейных алгебраических уравнений (СЛАУ), которое обычно выполняется по методу Гаусса, естественно приводит [7] к выбору для этого метода компактной схемы (описанной, например, в [9]). Но в последние годы не только все узлы вычислительных кластеров, но и рядовые персональные компьютеры стали многоядерными. Это важно учитывать при решении вычислительных задач МДО (как, впрочем, и многих других).

В отличие от распределения параллельных вычислений на разные компьютеры (или узлы вычислительного кластера), использование нескольких процессорных ядер в составе одного компьютера (узла кластера) сталкивается с возможностью конфликта по разделяемым ресурсам, прежде всего по памяти. Поэтому время выполнения одной и той же программы с помощью многоядерного процессора не всегда будет меньше, чем с помощью одноядерного процессора такой же тактовой частоты. Дело ещё осложняется многоуровневым характером памяти. Чем дольше процессор работает с кэшем оперативной памяти, не нуждаясь в его обновлении, тем, при прочих равных условиях, быстрее идут вычисления. Этот эффект принято рассматривать как локализацию памяти, и существует общая теория, как им следовало бы управлять (этим вопросам посвящены недавние работы [10-11]). Но даже при хорошей локализации разные ядра, в принципе, могут задерживать друг друга, используя общий кэш. Необходимо, очевидно, чтобы каждое ядро было занято собственным вычислительным процессом, который хорошо локализуется по памяти. Но априори не ясно, что лучше, чтобы множества данных, с которыми в один и тот же момент работают в кэше разные ядра, не пересекались или же, напротив, были общими? Или требуется какой-то «средний» вариант? Далее, неверен принцип — по одной задаче на одно ядро. Как видно из исследования [12], лучший результат может дать организация в программе значительно большего числа раздельных процессов, чем имеется процессорных ядер.

Актуальным является выяснение этих вопросов для алгоритмов, которые часто используются. При реализации МДО это схемы формирования СЛАУ, моделирующих гиперсингулярные интегральные и псевдодифференциальные операторы [3-4], методы решения СЛАУ (прежде всего, как следует из предыдущих замечаний, — компактная схема метода Гаусса), методы расчёта полей по рассчитанным дискретным приближениям к распределениям токов на проводящих поверхностях, визуализация полей.

В данной работе с указанной точки зрения изучается компактная схема метода Гаусса. Целью является решение вопроса: оправдана ли вообще забота о локализации памяти при распараллеливании вычислений на разные ядра, и

каким методом можно подбирать оптимальные варианты алгоритмических схем распараллеливания, ориентированных на так называемый тайлинг?

Отметим, что для таких важных (применяемых в кодировании и сжатии данных), но простых алгоритмов, как перемножение прямоугольных матриц, аналогичная задача ранее была успешно исследована на базе тестовой программы, разработанной на языке Ада [12]. Для нашего исследования выбрана та же инструментальная база — встроенные средства параллельного программирования языка Ада. Однако, хотя теоретически метод Гаусса и можно свести к последовательному перемножению матриц специального вида, практическое программирование алгоритмов этого метода имеет дело с треугольными частями квадратных матриц и соответственно вложенные суммы имеют переменные пределы. В этом случае не очевидна целесообразность подбора тайлов на основе квадратных подматриц, и не применимо асинхронное суммирование в противоположность [12]. Отсюда особенность постановки — требуется следить за тем, чтобы некоторое нарушение регулярности тайлов и дополнительные затраты времени на необходимую синхронизацию раздельно выполняемых процессов не «съели» тот выигрыш от тайлинга, который можно предполагать в алгоритмах, родственных умножению матриц.

2. Воплощение тайлинга и локализация памяти

Под тайлом - зерном вычислений в литературе (см. [10-13]) понимается множество операций алгоритма, выполняемых атомарно. В частности, вычисления, относящиеся к одному какому-то зерну, производятся на одном процессоре (в нашем случае - ядре), не прерываются синхронизацией и на их фоне с помощью этого процессора (ядра) не производится обмен данными.

Целью тайлинга (т.е. организации подходящих зерен вычислений - тайлов) является уменьшение накладных расходов на обмен данными между процессорами (ядрами) и (или) на использование общей иерархической памяти. Тайлинг всегда предполагает [10-11] выбор подструктур обрабатываемых структур данных (с расчётом спровоцировать локализацию работы процессорного ядра на соответствующих участках памяти), выбор последовательности выполнения операций, относящихся к тайлу, выбор последовательности выполнения тайлов.

Рассмотрим систему линейных уравнений

$$Ax = f \tag{1}$$

или в подробной записи

$$\begin{pmatrix} a_{11} + a_{12} + \ldots + a_{1n} \\ a_{21} + a_{22} + \ldots + a_{2n} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1} + a_{n2} + \ldots + a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \ldots. \\ x_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \ldots. \\ f_n \end{pmatrix} \tag{1´}$$

Для неё в так называемой компактной схеме метода Гаусса сначала по явным формулам рассчитываются элементы разложения матрицы в произведение левой и правой треугольных матриц (по другой терминологии — нижне и верхнетреугольных). Обозначим ненулевые элементы этих матриц как l_{ij} и u_{ij} соответственно. Тогда, как известно [9], их расчёт можно произвести по схеме:

$$u_{11} = a_{11}, \quad u_{1j} = a_{1j}\ , \quad l_{j1} = \frac{a_{j1}}{u_{11}}, \quad j = 2,3,...,n\ ,$$

$$l_{ii} = 1, \quad u_{ii} = a_{ii} - \sum_{p=1}^{i-1} l_{ip} u_{pi} \quad , \quad i = 2,3,...,n\ ,$$

$$u_{ij} = a_{ij} - \sum_{p=1}^{i-1} l_{ip} u_{pj}\ , \quad l_{ji} = \frac{a_{ji} - \sum_{p=1}^{i-1} l_{jp} u_{pi}}{u_{ii}}, \tag{4}$$

$$i = 2,3,...,n, \quad j = i+1, i+2,...,n$$

После этого решение системы (1), где $A=LU$, завершается по алгоритму, отражающему процесс последовательного решения двух треугольных систем:

$$L\ y = f\ , \tag{5}$$

$$U\ x = y\ , \tag{6}$$

что занимает немного времени в сравнении с предвычислением матриц L и U. Отметим также, что эти матрицы, как правило, хранятся на месте матрицы A.

Нами было выдвинуто предположение о том, что для вычислительного процесса, который определяется формулами (2)-(4), для структуры данных, имеющей форму массива (см. (1´)), можно успешно обеспечить тайлинг (зернистость), предусмотрев отдельные процессы для обработки подматриц вида

$$\begin{pmatrix} a_{i,j} & a_{i,j+1} & \cdots & a_{i,j+m-1} \\ a_{i+1,j} & a_{i+1,j+1} & \cdots & a_{i+1,j+m-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i+m-1,j} & a_{i+m-1,j+1} & \cdots & a_{i+m-1,j+m-1} \end{pmatrix}\ , \tag{7}$$

$$i = m \cdot k,\ k = 1..\frac{n}{m}, \quad j = m \cdot k,\ k = 1..\frac{n}{m}, \tag{8}$$

$$i \neq j \tag{9}$$

Впоследствии вычислительные эксперименты подтвердили оправданность этой гипотезы. Действительно, нашлись значения параметров, при которых вычислительный процесс значительно ускоряется, а это в данной ситуации возможно только за счёт того, что при таких значениях параметров обработка данных вида (7) действительно становится тайлом.

Однако такой эффект зависит, как отмечено выше, не только от выбора для тайлинга определённых подструктур данных, но и от выбора порядка операций, который обычно можно варьировать в некоторых рамках, не меняя (с точностью до различия вычислительных погрешностей) конечные результаты вычислений. Цель такого варьирования — улучшить локализацию вычислений по памяти. В прозрачных ситуациях на оптимальный выбор варианта может указать результат вычисления так называемых векторов локальности [10-12]. В любом случае локальность улучшается преобразованиями циклов, которые позволяют процессорному ядру повторно использовать те же элементы данных на большем числе итераций. Это возможно двумя путями - за счёт улучшения временной и (или) пространственной локальности. Временная локальность означает, что данные повторно используются прежде, чем они перемещаются из памяти с быстрым доступом в память более низкого уровня. Этого можно добиться, сосредотачивая, по возможности, все необходимые операции над порцией данных на итерациях самого внутреннего цикла. Пространственная локальность означает использование в смежных по времени звеньях вычислений данные, расположенные смежно в основной памяти компьютера. Улучшение пространственной локальности повышает вероятность того, что очередная требуемая порция данных окажется в памяти быстрого доступа, попав туда ранее при загрузке «большого» блока ради другой порции данных.

Эти известные теоретические представления напоминают задачи оптимизирующего преобразования программ при компиляции с языков высокого уровня. Но рассчитывать на автоматическое обеспечение высокой локальности программ штатными компиляторами не приходится. Тем более, что регулярность системы тайлов в непрямоугольных структурах нарушается.

Задача полного перебора вариантов с реализацией даже на таком удобном языке, как Ада, не тривиальна, тестирование - длительно. Изложим, как нам удалось выявить варианты с относительно лучшей локальностью теоретически.

Перестановка индексов суммирования по (*i, j, p)* в формулах (4) порождает множество 3!•3! вариантов, которые мы идентифицируем с помощью пометки подматриц и - по существующей в литературе традиции (см., например, [13, 10,12]) - упорядоченных троек индексов. Рассмотрим, например, тайл вида (7) для *U*, который должен сохраняться на месте правой верхней треугольной подматрицы (1´). Если его расчёт производится с сохранением вытекающего из формулы (4) порядка перебора индексов, то эта обработка обозначается

$$U\,(i, j, p) \qquad . \tag{10}$$

Если же мы в соответствии с идеями [10] хотим изменить обработку так, чтобы индексы перебирались в другом порядке, например,

$$U\,(i,\ p,\ j) \tag{11}$$

с сохранением конечного результата, то эта обработка будет соответствовать алгоритму:

$$\begin{array}{l} for\ i\ \ in\ \ 2..n\ \ loop \\ \quad for\ k\ \ in\ \ 1..i-1\ \ loop \\ \quad\quad for\ \ j\ in\ \ k+1..n\ \ loop \\ \quad\quad\quad u_{ij} = u_{ij} - l_{ik}u_{kj} \\ \quad\quad end\ \ loop; \\ \quad end\ \ loo; \\ end\ \ loop; \end{array} \tag{12}$$

Для всех оставшихся 36-2 = 34 вариантов алгоритм обработки тайлов (без изменения конечных результатов) был выписан аналогично, что позволило доказать следующие два утверждения.

Варианты перестановок циклов в нижней треугольной подматрице:

$$L\,(i,\ p,\ j),\ L\,(j,\ p,\ i),\ L\,(p\ ,i,\ j),\ L\,(p,\ j,\ i)\quad ,\text{-} \tag{13}$$

заведомо требуют каждый выполнения более, чем на 30 процентов больше операций, чем в каком-то из других вариантов.

Для *U* менее перспективны по сравнению с другими перестановки:

$$U\,(j,\ p,\ i),\ U\,(p,\ i,\ j),\ U\,(p,\ j,\ i) \qquad . \tag{14}$$

Это вытекает из соображений, аналогичных тем, которые в похожих ситуациях приводились в исследованиях [13, 12].

Остаются 6 перспективных для экспериментального исследования вариантов:

$$U\,(i,\ j,\ p)\ L\,(j,\ i,\ p)\ , \tag{15}$$

$$U\,(j,\ i,\ p)\ L\,(i,\ j,\ p)\ , \tag{16}$$

$$U\,(i,\ j,\ p)\ L\,(i,\ j,\ p)\ , \tag{17}$$

$$U\,(i,\ p,\ j)\ L\,(i,\ j,\ p)\ , \tag{18}$$

$$U\,(j,\ i,\ p)\ L\,(j,\ i,\ p)\ , \tag{19}$$

$$U\,(i,\ p,\ j)\ L\,(j,\ i,\ p)\ . \tag{20}$$

Теперь – об организации параллельных вычислений. Один последовательный процесс (он же – поток, он же – задача языка Ада) обрабатывает непустое

множество тайлов: матрица (которая вначале хранит исходные данные, а в конце результат разложения) разрезается на равные по ширине полосы (числом N), каждая из которых распадётся ещё на L–часть и U–часть.

3. Метод организации вычислительных экспериментов

3.0. Перед тем как приступать к проведению основных экспериментов, по которым можно было бы сделать окончательные выводы, необходимы вспомогательные тесты. Эти эксперименты требуются, помимо того, что нужно убедиться в функциональной правильности кода, преимущественно с целью предварительно оценить возможность получения прироста производительности за счёт тайлинга, причём на разных компьютерах. Нами, как вспомогательные, так и основные тесты, проводились на компьютерах intel atom 330 с 512 kb кэша 2 уровня, Amd Phenom 9550 с 4*512 Kb кэша 2 уровня и 2 Mb кэша 3 уровня и Amd Phenom 8650 с 3*512 Kb кэша 2 уровня и 2 Mb кэша 3 уровня.

В нашем случае предварительное тестирование позволило:

- убедится в том, что само по себе количество k задач языка Ада, которые обрабатывают тайлы, не влияет на производительность s ядер,если $k \geq s$;
- подобрать достаточный диапазон варьирования размерностей тайлов.

Следствием первого факта является возможность загружать процессор по максимуму, следуя простому правилу. Например, если число ядер не больше 4 или размерность тайла (7) намного меньше размерности матрицы A (1´), можно полагать $k = 2 \cdot N$ (см. конец предыдущего раздела).

Схема основных вычислительных экспериментов такова.

3.1. Тестирование начинать на одном, практически наиболее доступном для этого процессоре (у нас это был компьютер с intel atom 330) на матрицах с возрастающими размерностями (мы последовательно генерировали случайные матрицы с 1000x1000, 1731x1731, 2996x2996 и 5186x5186 элементами). Экспериментально оценить зависимость продолжительности вычислений от размера тайла и перестановки циклов. При этом необходимо варьировать перестановки циклов (8), проверяя, конечно только те из них, которые рассматриваются как перспективные на основе аналитического исследования локальности (у нас это (10)-(15)). На выходе первого этапа ожидается, что можно будет выявить наиболее «медленные» (с точки зрения экстраполяционного прогноза) кривые роста времени счёта, как функции размерности. Соответствующие значения параметров (размера тайла и индекса перестановки (10)) будут переданы на следующие этапы тестирования для подтверждения или уточнения их оптимальности. Отметим, что в соответствии с результатами предварительного этапа мы рассматривали

$$\text{тайлы (7) с размерностями } \quad m = 20, 40, 80, 150 \tag{21}$$

(число вариантов в намеченном диапазоне вытекает из оценки ресурса времени).

3.2. На втором этапе при малом, оставшемся после первого этапа, числе параметров (параметры здесь – это размер тайла и перестановка) необходимо убедиться (или опровергнуть), что выявленные на первом этапе закономерности для времени счёта сохраняются на других процессорах. У нас на этом этапе

рассматривалось два варианта перестановки циклов (наиболее оптимальный на первом этапе и следующий за ним), но все варианты m (21) мы из осторожности сохранили. На выходе этапа следует сохранить не более двух перспективных пар параметров и выбрать наиболее производительный компьютер.

3.3. Попытаться подтвердить оптимальность выбранной пары параметров в рамках избранных дискретных диапазонов их изменения (или, если таких пар две, то окончательно определиться с оптимальной парой, проводя эксперимент с матрицами наибольшей размерности, доступной на наиболее производительном компьютере). В нашем случае треугольное разложение случайных матриц найбольшего из рассмотренных размеров 8977 проводилось на Amd Phenom 9550. Существенно большие размерности были недоступны с точки зрения условия завершения вычислений за 12 часов.

3.4. Если значительный эффект тайлинга подтвердился, на заключительном этапе целесообразно провести сравнение минимального из наблюдавшихся (при каждом значении размерности) времен счёта с показателем времени для реализаций без тайлинга. Для этих тестов следует подбирать «свой» наиболее производительный компьютер. Мы в этой части эксперимента произвели тесты, использовав процессоры Amd Phenom 9550 и Amd Phenom 8650.

4. Результаты вычислительных экспериментов и выводы

По результатам первого этапа 3.1 выяснилось, что размер тайла не влияет на оптимальную последовательность обработки циклов и оптимальная последовательность для подматрицы L есть (i, j, p), а для U , начиная с размерностей несколько больше 1000, оптимальна последовательность (i, p, j) (для малых же размерностей - (j, i, p)). Поэтому в дальнейшем для L всегда использовалась последовательность (i, j, p), а для U - два варианта (i, p, j) и (j, i, p). Выбор вариантов иллюстрирует рис. 1.

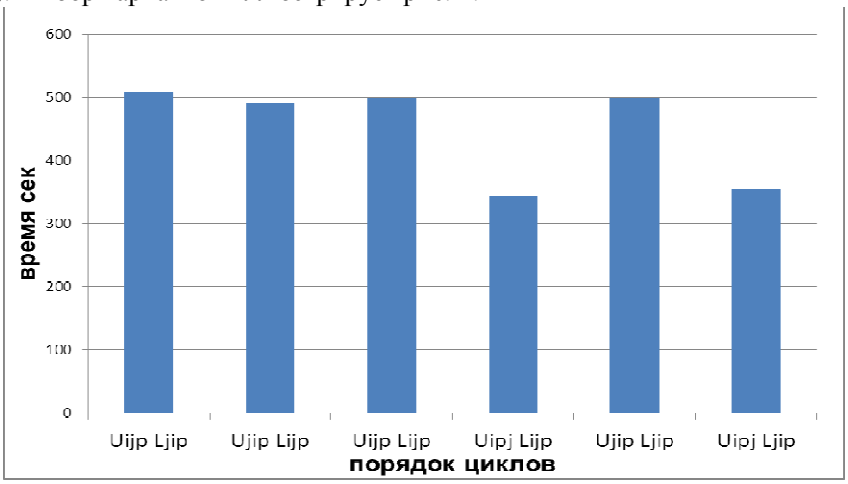

Рис.1. Время выполнения программы в зависимости от перестановки циклов на процессоре atom 330, размер матрицы 2996x2996.

С другой стороны, наилучший размер тайла для каждого значения размерности матрицы свой, причём выбор тестовой матрицы на продолжительность выполнения алгоритма практически не влияет (см. табл. 1).

Табл.1. Наилучший размер тайла в шкале (21) для случайных матриц разного размера, процессор atom 330.

Размер матрицы	1000	1731	2996
Размер тайла	20	40	80
Коэффициент вариации времени счёта при случайном выборе матрицы	0.028	0.013	0.009

На этапе 3.2 эксперименты на всех компьютерах подтвердили, что оптимальными являются схемы обработки:

$$L\,(i, j, p)\; U\,(i, p, j)\;. \qquad (22)$$

Однако размеры тайлов, оказавшиеся наилучшими, для некоторых компьютеров отличались, как показано в табл.2 (и соответственно на рис. 2). Поэтому на следующий этап мы перешли, имея, вообще говоря, по две пары параметров, которые можно было бы рекомендовать как оптимальные (например, для размерности 2996 это 80 и 40).

Табл.2. Наилучший размер тайла в шкале (21) для матриц разной размерности (обработка тайлов по схеме (22), данные на примере процессора Phenom 8650).

Размер матрицы	1000	1731	2996	5186	8977
Размер тайла	20	20	40	80	150
Вариация времени счёта при варьировании размерностей тайлов (21)	1.54	0.75	0.15	0.05	0.10

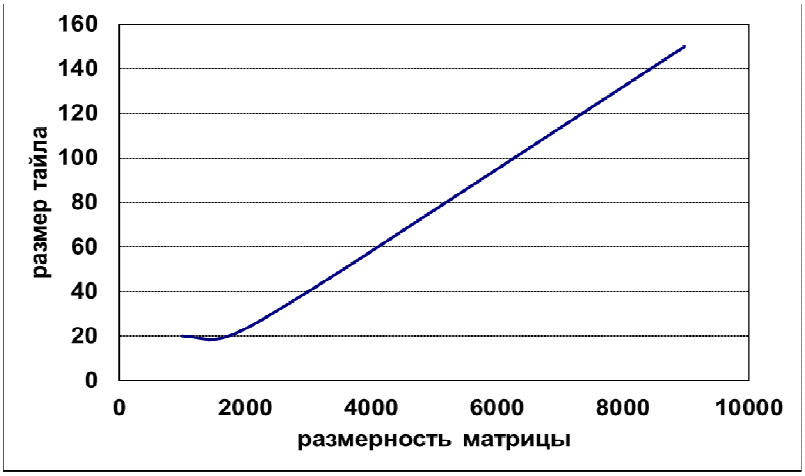

Рис.2. Наглядная зависимость оптимального размера тайла от размера матрицы согласно данным табл. 1.

Для третьего этапа был выбран процессор Amd Phenom 9550 (суммарная производительность всех ядер у него выше). Результаты показаны в табл. 3. Отметим, что хотя для размерности 2996 в соответствии с нашей схемой экспериментов нужно было выбрать в качестве рекомендуемого варианта размер тайла 40, время вычислений для наиболее мощного выбранного процессора практически не изменится и при выборе m=20.

На основе таблиц 1-3 можно сделать вывод о том, что при увеличении размера матрицы (1´) начиная с некоторого значения, увеличивается и размер рекомендуемого тайла, но чем больше кэш памяти и количества ядер у процессора, тем более медленным становится это увеличение.

Табл.3. Наилучший размер тайла в шкале (21) для случайных матриц разной размерност в экспериментах на компьютере с процессором Phenom 9550.

Размер матрицы	1000	1731	2996	5186	8977
Размер тайла	20	20	40	40	80
Относительное время счёта	1.00	6.02	36.3	207	1300

На заключительном этапе 3.4 проведилось тестирование многопоточных программ без тайлинга и с тайлингом, а также однопоточной программы (без тайлинга), результаты показаны на рис. 3.

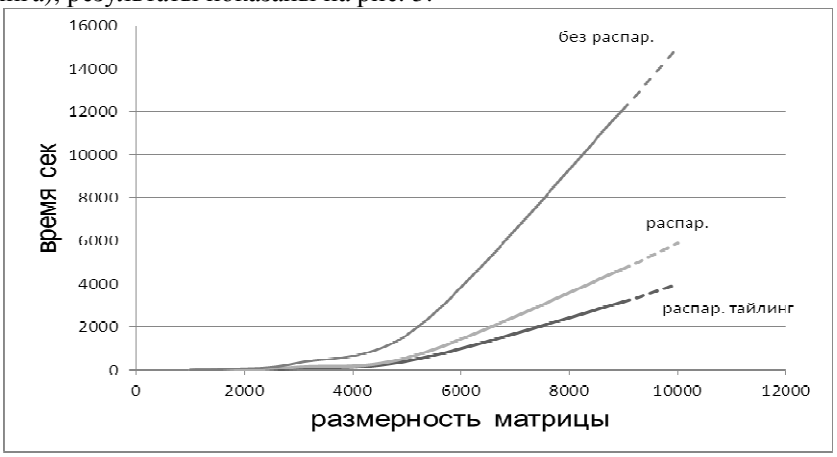

Рис.3. Время расчета однопоточной программы и распараллеленных программ с тайлингом и без него на компьютере с процессором Phenom 9550.

Основной вывод состоит в том, что, хотя распараллеливание вычислительного процесса треугольного разложения матриц по компактной схеме Гаусса за счёт многопоточности и даёт значительную экономию времени на многоядерном процессоре (примерно в 2.5 согласно рис. 3), она еще примерно в 1.5 раза увеличится, если задействовать тайлинг. В практике решения больших задач, требующих суток времени на счёт (размерность около 10^4), а также при решении большого числа задач такая дополнительная экономия, безусловно, существенна.

Другой важный вывод относится к методу исследования. В настоящее время в следствие того, что многоядерные процессоры устанавливаются на

персональные компьютеры, прикладным программистам нередко приходится сталкиваться с задачами распараллеливания вычислений, относящихся к той или иной прикладной области (например, авторы данной статьи работают над проблемой реализации вычислений МДО в области дифракционных задач акустики и электродинамики). Популярный подход состоит в том, чтобы использовать дополнительные инструментальные средства (компиляторы, библиотеки), создаваемые для инициации и управления несколькими (иногда многими) вычислительными процессами на компьютере. При этом код первоначально однопоточной программы, реализующей заданный алгоритм, анализируется на наличие «узких мест», которые пордлежат «расшивке» за счёт параллельности. Окончательная параллельная версия программы создаётся фактически вручную за счёт искусства и опыта разработчика ad hoc. Примером может служить недавняя работа [14] тоже посвященная численной реализации МДО, но в области фильтрационной гидродинамики с использованием классической схемы метода Гаусса и известной библиотеки Open MP. В ней на 4-ядерном процессоре в диапазоне размерностей до $0.4 \cdot 10^2$ достигнуто ускорение в решении СЛАУ не меньше, чем примерно в 3.5 раза (со увеличением этого показателя с уменьшением размерности). Мы добились аналогичного результата для размерностей от 10^3 до 10^4, применив один из альтернативных подходов, и даём рекомендацию об устройстве программы с единственным, в конечном итоге, параметром (размер тайла), практический выбор которого можно производить в соответствии с табл. 1-3. Альтернативность состоит в следующем: использован стандартный компилятор (за счёт того, что выбранный язык программирования Ада сам имеет высокоуровневые средства создания параллельных процессов и защиты операций от множественного изменения), применялась известная теория [10-11] и прозрачная общая схема проведения численных экспериментов.

5. Заключение

Создан метод, который позволяет для алгоритма обработки матриц на основе циклов по многим индексам (не обязательно изменяющихся в постоянных пределах) экспериментально подбирать наилучшую версию параллельной реализации на многоядерных процессорах за счёт локализации вычислений.

На основе вычислительных экспериментов в соответствии с этим методом разработан и практически оптимизирован по своим параметрам параллельный вычислительный метод треугольного разложения матриц на основе компактной схемы метода Гаусса.

Ада программа, реализующая этот метод, доступна под лицензией GPL [15].

ЛИТЕРАТУРА

1. Воеводин В.В., Воеводин Вл.В., Параллельные вычисления, БХВ-Петербург, С-Пб, 2002.– 608 с.
2. Давыдов А. Г., Захаров Е. В., Пименов Ю. В. Метод численного решения задач дифракции электромагнитных волн на незамкнутых поверхностях произвольной формы // ДАН СССР, 1984,- т.286, вып. 1., С. 96-100.

3. Вайникко Г.М., Лифанов И.К., Полтавский Л.Н. Численные методы в гиперсингулярных интегральных уравнениях и их приложения. – М.: Янус-К.– 2001.– 508 с.
4. Гандель Ю.В. Парные и гиперсингуляные интегральные уравнения задач дифракции электромагнитных волн на плоских решетках и экранах // Труды XI Международного симпозиума МДОЗМФ-2003.-Харьков-Херсон,2003.-С.53-58.
5. Гандель Ю.В., Мищенко В.О. Псевдодифференциальные уравнения электромагнитной дифракции на плоскопараллельной структуре и их дискретная модель // Вестник Харк. нац. ун-та., – 2006. – № 733. Сер. «Математическое моделирование. Информационные технологии. Автоматизированные системы управления», вып. 6. – С. 58-75.
6. Мищенко В.О. К моделированию электромагнитных явлений на базе использования методов дискретных особенностей для решения гиперсингулярных интегральных уравнений //Труды международной конференции по вычислительной математике МКВМ-2004.– Ч. II, Новосибирск: Изд. ИВМиМГ РАН, 2004.– с. 555-560.
7. Mishchenko V.O. Ada programming language and specifying, modeling and distributed computing for the implementation of the discrete singularities methods // Procedins of SCALNET-2004, KSPU. – Kremenchug. – 2004. – P. 110-112.
8. Гахов А.В., Мищенко В.О. Схема параллельной модификации систем компьютерного моделирования, использующих методы дискретных особенностей // Радиоэлектронные и компьютерные системы. – 2010. – № 6 (47). – С. 143 – 147.
9. Воеводин В.В., Вычислительные основы линейной алгебры, НАУКА, Москва, 1977. – 155с.
10. Лиходед Н. А. Методы распараллеливания гнезд циклов : курс лекций – Минск : БГУ, 2008. -100 с.
11. Лиходед Н. А., Пашкович А. К. О выборе зерна вычислений при реализации алгоритмов на параллельных компьютерах с распределенной памятью // Весці НАН Беларусі. Сер. фіз.-мат. навук. 2008. № 2. С. 121—123
12. Л.С. Киркоров, С.И, Киркорова Параллельные алгоритмы математиеских моделей: исследование локальности и применение языка Ада. // Вестник Харк. нац. ун-та., – 2009. – № 863. Сер. «Математическое моделирование. Информационные технологии. Автоматизированные системы управления», вып. 12. – С. 129-142.
13. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем, М.: Мир, 1991 – 367 с.
14. Дорофеева В.И., Матвеев И.Е. Применение параллельных технологий для решения задачи растекания бугра грунтовых вод под действием силы тяжести //Труды Международных школ-семинаров «МДОЗМФ». Выпуск 8. – Орёл: Из-во ГОУ ВПО «ОГУ», полиграфич фирма «Картуш», 2010, с. 48-53.
15. Parallel Gaussian compact scheme // [Электрон. ресурс]. – http://www.mediascan.by/index.files/parallel_gaussian_compscheme_win32.zip .

Надійшла у першій редакції 03.07.2011, в останній – 07.10.2011.