УДК 519.688, 004.428 DOI: 10.14529/cmse170103

ЭКСПЕРИМЕНТАЛЬНОЕ СРАВНЕНИЕ АЛГОРИТМОВ В ПАРАЛЛЕЛЬНОМ МЕТОДЕ ВЛОЖЕННЫХ СЕЧЕНИЙ*

© 2017 г. А.Ю. Пирова, Н.Ю. Кудрявцев, И.Б. Мееров

Нижегородский государственный университет им. Н.И. Лобачевского
(603022, г. Нижний Новгород, просп. Гагарина, д. 23)
E-mail: anna.pirova@itmm.unn.ru, n.yu.kudriavtsev@gmail.com,
iosif.meyerov@itmm.unn.ru

Поступила в редакцию: 20.10.2016

В прямых методах решения больших разреженных систем линейных алгебраических уравнений применяется процедура переупорядочения строк и столбцов исходной матрицы. Целью данной процедуры является сокращение числа ненулевых элементов в процессе последующей численной факторизации. Нахождение перестановки, минимизирующей число ненулевых элементов в факторе, является NP-полной задачей. Для решения этой задачи применяются эвристические методы. Результаты применения данных методов могут быть оценены как с точки зрения качества получаемых перестановок (заполнение фактора матрицы после переупорядочения), так и с точки зрения временных затрат на построение перестановок. Многоуровневый метод вложенных сечений, показывающий достаточно хорошие результаты по обоим критериям, является одним из наиболее распространенных методов переупорядочения. Метод имеет определенные ресурсы внутреннего параллелизма, активно используемые в ряде реализаций (ParMETIS, mtMETIS, PT-SCOTCH, PMORSy). Вместе с тем, низкая арифметическая интенсивность, нерегулярный доступ к памяти, дисбаланс вычислительной нагрузки и необходимость поиска компромисса между временем работы и качеством перестановок мотивируют дальнейшие исследования метода.

В данной работе выполняется сравнение ряда алгоритмов, применяемых на разных этапах метода вложенных сечений, с точки зрения их влияния на заполнение фактора и время работы в параллельном случае. Реализация алгоритмов и эксперименты выполнены в рамках ранее разработанной параллельной библиотеки переупорядочения матриц PMORSy, опережающей аналоги на ряде матриц коллекции университета Флориды. В результате выполненной работы удалось выделить наиболее перспективную комбинацию алгоритмов и улучшить качество перестановок и время работы PMORSy.

Ключевые слова: метод вложенных сечений, переупорядочение разреженных матриц, параллельный алгоритм.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Пирова А.Ю., Кудрявцев Н.Ю., Мееров И.Б. Экспериментальное сравнение алгоритмов в параллельном многоуровневом методе вложенных сечений // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2017. Т. 6, № 1. С. 38–55. DOI: 10.14529/cmse170103.

Введение

Решение систем линейных уравнений прямыми методами заключается в разложении (факторизации) матрицы системы на произведение верхне- и нижнетреугольной матриц и дальнейшего решения двух систем линейных уравнений с полученными треугольными матрицами. При этом, в случае разреженной матрицы системы, возникает проблема *заполнения*: число ненулевых элементов фактора матрицы в разы больше числа ненуле-

* Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии (ПаВТ) 2017».

вых элементов исходной матрицы. Чтобы снизить заполнение фактора, предварительно применяется процедура *переупорядочения* строк и столбцов исходной матрицы. От результатов переупорядочения зависят затраты памяти на хранение фактора и времени на выполнение дальнейшей факторизации. Поэтому под *качеством перестановки* чаще всего понимают число ненулевых элементов факторизованной матрицы (чем меньше, тем лучше). Кроме того, перестановка влияет на потенциал распараллеливания наиболее трудоемкого этапа решения — численной факторизации, поскольку она определяет взаимное расположение ненулевых элементов.

Нахождение перестановки, минимизирующей число ненулевых элементов в факторе, является NP-полной задачей [18]. На практике для ее решения применяются две группы эвристических методов — методы минимальной степени [17] и методы вложенных сечений [5, 6]. Известно, что перестановки, полученные методом вложенных сечений, имеют больший потенциал для параллелизма на этапе численной факторизации [4]. Поэтому в данной работе для реализации была выбрана модификация этого метода — многоуровневый метод вложенных сечений [1, 8], применяющийся с середины 1990-х годов.

В настоящее время в мире разработан ряд библиотек для переупорядочения разреженных матриц, преимущественно имеющих последовательные реализации или реализации для систем с распределенной памятью. Многоуровневый метод вложенных сечений, в частности, реализован в свободно распространяемых библиотеках METIS [10] и Scotch [12], широко используемых в решателях СЛАУ, а также в их аналогах для систем с распределенной памятью ParMETIS [7] и PT-Scotch [2]. Среди российских разработок отметим пакет МАТРУЗ [20]. В последние годы ведется работа по разработке реализаций многоуровневого метода вложенных сечений для систем с общей памятью. Так, в 2013 году авторами Scotch обсуждалось распараллеливание отдельных этапов многоуровневого метода [13], в 2015 году авторами METIS был представлен алгоритм переупорядочения для систем с общей памятью, реализованный в пакете mt-metis [11]. В то же время в Нижегородском государственном университете им. Н.И. Лобачевского была разработана библиотека PMORSy [15, 19], также ориентированная на многоядерные системы с общей памятью. На матрицах из разных предметных областей было показано [15], что реализация из PMORSy сопоставима по времени и качеству с реализациями из ParMETIS, mt-metis и PT-Scotch. При этом на ряде матриц PMORSy опережает аналоги, на других — проигрывает им в смысле указанных выше критериев.

Отдельного обсуждения заслуживает вопрос об эффективности распараллеливания. Известно, что алгоритмы переупорядочения обладают рядом особенностей, существенно затрудняющих возможность создания хорошо масштабируемого параллельного алгоритма. Среди таких особенностей можно отметить достаточно низкую арифметическую интенсивность, нерегулярный доступ к памяти, дисбаланс вычислительной нагрузки при решении независимых подзадач. Указанные причины приводят к недостаточной эффективности масштабируемости параллельных алгоритмов во всех указанных выше библиотеках, что обуславливает актуальность продолжения исследований в данной области.

Основная цель данной работы заключалась в изучении множества отдельных алгоритмов, предложенных в разное время для решения задач, возникающих на ряде этапов метода вложенных сечений. Исследовалось влияние данных алгоритмов на качество получаемых перестановок и время работы не только в последовательном [8, 9], но и, глав-

ное, в параллельном случае. В качестве базовой реализации использовалась библиотека PMORSy, тестовые задачи брались из коллекции университета Флориды.

Статья построена следующим образом. В разделе 1 приводится описание метода вложенных сечений и параллельного алгоритма, использующегося в PMORSy. В разделе 2 описаны отдельные этапы многоуровневого метода и особенности их программной реализации. В разделе 3 приводятся результаты сравнения времени и качества PMORSy в зависимости от выбора рассматриваемых алгоритмов, а также приводится сравнение с библиотекой mt-metis. В заключении обобщены результаты работы и приведены планы дальнейших исследований.

1. Параллельный многоуровневый метод вложенных сечений

Приведем краткое описание многоуровневого метода вложенных сечений. Базовым для метода является понятие *разделителя* — множества вершин, удаление которого расщепляет граф на две или более несвязные части. Процесс нахождения перестановки выполняется следующим образом. В графе, построенном по матрице СЛАУ, находят разделитель, который делит граф на две близкие по размеру части. Затем разделитель удаляется из графа, его вершины нумеруются последними свободными индексами, а процесс продолжается независимо на образовавшихся несвязных подграфах. Переупорядочение завершается, когда все вершины графа занумерованы. Качество переупорядочения зависит от того, насколько удачно были вычислен разделитель на каждом шаге. Чем меньше разделитель и дисбаланс образовавшихся после его удаления частей, тем лучше будет найденная перестановка.

При использовании многоуровневого метода вложенных сечений разделитель вычисляется в три этапа: огрубление, разделение, развертывание. На этапе огрубления по исходному графу с помощью паросочетаний строится последовательность графов меньшего размера, в которой каждый следующий граф представляет собой огрубленную структуру предыдущего графа. На этапе разделения выполняется нахождение разделителя и несвязных частей самого маленького в последовательности графа. Затем, в процессе развертывания, этот разделитель проецируется через всю последовательность графов на исходный граф. При этом используются алгоритмы улучшения разделения, которые уменьшают разделитель и сокращают дисбаланс образовавшихся несвязных частей. Псевдокод нахождения разделителя многоуровневым методом приведен на рис. 1. Подробное описание последовательного алгоритма переупорядочения, реализованного в PMORSy, приведено в [14].

Распараллеливание многоуровневого метода вложенных сечений в PMORSy использует принцип «разделяй и властвуй», заложенный в методе вложенных сечений. Алгоритм переупорядочения представляет собой параллельную обработку очереди независимых задач. Задачей, назначаемой потоку, является вычисление разделителя некоторого графа и получение новых несвязных подграфов. После того, как разделитель графа найден, поток создает новые задачи по обработке порожденных подграфов, если они достаточно большие, и выполняет рекурсивную обработку маленьких подграфов. Для реализации алгоритма использовался механизм логических задач OpenMP 3.0.

```
1.  (S0, G1*,..., Gk*) NDStep(graph G0(V0, E0), int nSt,
2.                             int sm, int r) {
3.    if |V0| < sm {
4.      S0 = AutomaticNestedDissection(G0);
5.      return (S0, NULL);
6.    }
7.    else {
8.      i = 0;
9.      while (i < nSt) && (Vi > sm) {
10.     Gi+1(Vi+1, Ei+1) = Coarse(Gi(Vi, Ei)); i++;
11.   }
12.   m = i;
13.   Pm(Sm, Vm,1, Vm,2) = InitializePartition(Gm, r);
14.   for(i = m; i >= 0; i--) {
15.     Pi-1 = ProjectPartition(Pi, Gi);
16.     RefinePartition(Pi-1);
17.   }
18.   (G1*, G2*, ..., Gk*) = FindSubgraphs(G0, P0);
19.   return (S0, G1*, G2*, ..., Gk*);
20. }
```

Рис. 1. Многоуровневый метод вложенных сечений в PMORSy [14]

Псевдокод параллельного алгоритма переупорядочения приведен на рис. 2. Основная функция *PMORSyOrdering* принимает на вход портрет графа матрицы G и параметры последовательного и параллельного алгоритмов *nSt*, *sm*, *r*, *b*. Задаче, назначаемой одному потоку, соответствует вызов функции *NDStepParallel*, которая принимает на вход подграф G_0 и параметры алгоритмов, и возвращает найденный разделитель S. После того, как для всех подграфов был найден разделитель, выполняется их слияние в массив перестановки.

```
1.  int* NDStepParallel(graph G0(V0, E0), int nSt, int sm,
2.                      int r, int b) {
3.    (S0, G1*, G2*, ..., Gk*) = NDStep(G0, nSt, sm, r);
4.    for (i = 0; i < k; i++) {
5.      #pragma omp task if (Gi* > b)
6.      NDStepParallel(Gi*, nSt, sm, r, b);
7.    }
8.    return S;
9.  }
10.
11. void PMORSyOrdering(graph G, int* iperm,
12.                     int nSt, int sm, int r, int b) {
13.   #pragma omp parallel
14.   #pragma omp single
15.     S = NDStepParallel(G, nSt, sm, r, b);
16.   iperm = Merge(S);
17. }
```

Рис. 2. Параллельный метод вложенных сечений в PMORSy [14]

2. Алгоритмы многоуровневого метода вложенных сечений

Рассмотрим подробнее алгоритм получения разделителя многоуровневым методом (рис. 1). Функция *NDStep* на вход получает граф G_0 и параметры алгоритма: *nSt* (мак-

симальное число шагов огрубления), *sm* (минимальный размер графа, к которому применяется многоуровневый метод вложенных сечений), *r* (число перезапусков алгоритма разделения). Выход функции *NDStep* — разделитель S_0 и новые подграфы G_1^*,..., G_k^*. Этапу огрубления соответствует вызов в цикле функции *Coarse* (строки 8–10), этапу разделения — вызов функции *InitializePartition* (строка 13), этапу развертывания — вызов в цикле функций *ProjectPartition* и *RefinePartition* (строки 14–17).

Для выполнения каждого из этапов многоуровневого метода разработан ряд последовательных алгоритмов, успешно применяющихся в рамках последовательного и параллельного метода вложенных сечений.

Так, для огрубления используются алгоритмы поиска максимального паросочетания. Для этого применяются алгоритмы случайных паросочетаний (random matching, RM), паросочетания тяжелых ребер (heavy edge matching, HEM), паросочетания легких ребер (light edge matching, LEM), паросочетания тяжелых клик (heavy clique matching, HCM) и др. Их описание можно найти, например, в работе [8]. Предварительные эксперименты показали, что паросочетания легких ребер или, аналогично, легких вершин, приводят к образованию слишком плотных графов, что снижает итоговое качество переупорядочения. Поэтому в данной работе были выбраны алгоритм случайных паросочетаний (RM), паросочетания тяжелых ребер (HEM), паросочетание тяжелых клик (HCM).

На этапе разделения выполняется определение вершинного разделителя графа и несвязных частей, образующихся после его удаления. Часто вершинный разделитель находят из реберного разделителя. Для этого изначально разделяют все вершины на два множества, близких по числу вершин или по их суммарному весу. Затем применяют алгоритм поиска минимальной вершинной оболочки либо включают в вершинный разделитель все вершины, инцидентные реберному разделителю.

Одним из простых методов получения вершинного разделителя является выделение среднего уровня смежности поиском в ширину, запущенным из псевдопериферийной вершины (level structure from pseudoperipheral vertex, LS) [6]. Для получения реберного разделителя также используются методы, основанные на поиске в ширину из некоторой вершины, например, алгоритм растущего разделения (graph growing partition, GG), алгоритм жадного растущего разделения (greedy graph growing partition, GGG). Алгоритм Кернигана—Лина (Kernighan–Lin partition, KL) выполняет минимизацию реберного разделителя для некоторого начального разделения, поэтому может использоваться как процедура, улучшающая свойства заданного разделения. Также для разделения графа применяются методы, использующие спектральную или геометрическую информацию о графе. Однако предварительные эксперименты показали, что их использование в рамках переупорядочения ведет к большим временным затратам при том, что качество полученных с их помощью перестановок не отличается значительно от качества перестановок, полученных другими методами. Поэтому в данной работе эти методы не рассматривались. Обзор алгоритмов реберного разделения графа можно найти в [8, 16].

Все алгоритмы, использованные в данной работе, были реализованы по описанию из [8]. Для алгоритмов, получающих реберный разделитель, в вершинный разделитель включались все вершины, инцидентные реберному разделителю. Кроме того, эти алгоритмы запускались несколько раз из случайных начальных вершин, чтобы улучшить качество получаемого разделения. Алгоритм Кернигана—Лина в качестве начального

разделения получал результаты растущего разделения графа. Ко всем алгоритмам на этапе развертывания графа применялась процедура улучшения вершинного разделения, описанная в [14].

3. Результаты вычислительных экспериментов

3.1. Методика проведения экспериментов

Основная цель, поставленная в данной работе, — определить, какие алгоритмы огрубления и разделения, необходимые для многоуровневого метода вложенных сечений, а также параметры самой многоуровневой схемы, целесообразно использовать в рамках имеющейся параллельной реализации переупорядочения в PMORSy. Результаты переупорядочения традиционно оцениваются с точки зрения качества перестановок (то есть числа ненулевых элементов фактора) и времени работы. При этом подбор параметров для улучшения одного из этих критериев, нередко приводит к ухудшению второго. В данной серии экспериментов не ставилось задачи найти лучшие перестановки по времени и (или) качеству для каждой из рассмотренных матриц. Напротив, для всех тестовых задач использовались одни и те же «компромиссные» значения основных параметров метода.

Вычислительные эксперименты проводились на 48 матрицах из коллекции Университета Флориды [3], из разных предметных областей. Все матрицы были симметричные, вещественные, порядком от $2{,}2 \times 10^6$ до $2{,}8 \times 10^8$, с числом ненулевых элементов от $7{,}6 \times 10^6$ до $7{,}6 \times 10^9$. В тестовый набор вошли 28 самых больших симметричных матриц коллекции. Все эксперименты проводились на узле суперкомпьютера Лобачевский, имеющем следующие характеристики: два 8-ядерных процессора Intel Sandy Bridge E5-2660, частота 2.2 GHz, память 64 GB, операционная система Linux CentOS 6.4. Использовался компилятор Intel C++ Compiler и генератор случайных чисел из библиотеки Intel MKL, входящие в пакет Intel Parallel Studio XE 2015 Cluster Edition.

Для контроля результатов PMORSy проводилось сравнение с библиотекой mt-metis v.0.5.0 [11]. Данная библиотека предназначена для систем с общей памятью, как и PMORSy, и входит в число лучших реализаций многоуровневого метода вложенных сечений. Более полное сравнение с библиотеками ParMETIS и mt-metis приведено в работе [15].

В рамках многоуровневого метода вложенных сечений в PMORSy сравнивались следующие алгоритмы огрубления: случайных паросочетаний (RM), паросочетания тяжелых ребер (HEM), паросочетание тяжелых клик (HCM). Для каждого из них были рассмотрены алгоритмы разделения: поиск в ширину из псевдопериферийной вершины (LS), алгоритм растущего разделения (GG), алгоритм жадного растущего разделения (GGG), алгоритм Кернигана—Лина (KL). Кроме того, в экспериментах варьировались параметры многоуровневого метода, связанные с числом запусков алгоритмов огрубления и разделения. Были использованы следующие значения:

- размер минимального графа (sm): 20, 100;
- максимальное число итераций алгоритма огрубления (nSt): 10, 20, 100;
- число запусков алгоритма разделения для алгоритмов GG, GGG, KL (r): 5, 10.

Всего для каждой матрицы было проведено 126 экспериментов для фиксированного числа потоков, запуски производились в 1 и 16 потоков. Библиотека mt-metis запускалась с параметрами по умолчанию. Отметим, что и PMORSy, и mt-metis в нескольких

запусках с одними и теми же параметрами переупорядочения дают близкие по времени, но разные по качеству результаты. Это связано с тем, что в алгоритмах используются псевдослучайные числа, генерируемые на разных потоках. Как правило, разница в качестве переупорядочения на одном графе составляет не более 2%, поэтому в данной серии экспериментов запуски проводились однократно. Для каждого запуска собирались следующие данные: время и качество работы переупорядочения, время обработки самого большого подграфа, отношение времени и качества переупорядочения к результатам, полученным mt-metis.

Результаты экспериментов далее представлены в следующем виде. Все эксперименты были сгруппированы по комбинациям параметров размера минимального подграфа и максимального числа шагов огрубления (*sm* и *nSt*). Результаты переупорядочения, в которых качество было хуже, чем у mt-metis, более чем на 60%, не участвовали в сравнении времени. Не анализировались и результаты экспериментов, занявших более 25 минут. Полученные значения времени и качества для каждой матрицы были отображены на цветовую шкалу, на которой минимуму среди всех 126 запусков соответствует зеленый цвет, максимуму — красный цвет, промежуточным значениям — желтый цвет. Исключенным значениям соответствует белый цвет. Во всех таблицах матрицы приведены в порядке увеличения числа ненулевых элементов. Таким образом, наиболее вычислительно трудоемкие задачи сгруппированы внизу таблицы.

3.2. Сравнение алгоритмов при получении первого разделителя

Рассмотрим работу многоуровневого метода сечений при получении первого разделителя. Этот этап занимает около 15% общего времени последовательной версии переупорядочения для большинства матриц. Из этого времени по 40–60% времени выполняются этапы огрубления и улучшения разделения, около 3% — этап разделения. В PMORSy вычисление первого разделителя выполняется последовательно, и в параллельной версии занимает 30–55% общего времени работы переупорядочения. Это является существенным ограничением для масштабирования приложения на большое число потоков, особенно для графов с числом вершин более 10^7. Для сокращения времени вычислений требуется распараллеливание отдельных этапов многоуровневого метода.

3.2.1. Алгоритмы огрубления

Рассмотрим работу алгоритмов огрубления при получении первого разделителя. Сравним графы, полученные различными алгоритмами огрубления после 10 шагов. Для большинства матриц (45 из 48) был получен граф с числом вершин, на три порядка меньшим исходного, и плотностью порядка 10^{-2}. В большинстве экспериментов граф меньшего размера был получен алгоритмами RM или HCM, при этом графы, полученные RM, были в 1,5–11,7 раз плотнее, чем полученные другими алгоритмами. Если продолжать огрубление далее, то для получения графа из 100–150 вершин алгоритмы выполняют почти одинаковое число шагов (разница 1–2 шага).

Во всех экспериментах 4 матрицы из 48 были недостаточно сжаты большинством алгоритмов. Это матрицы boyd2, HTC_336_9129, kkt_power, c_big. Причина такой работы алгоритмов — наличие в исходной матрице плотной строки, т.е. вершины графа матрицы, связанной почти со всеми другими вершинами.

Сравним время работы алгоритмов огрубления. При выполнении 10 шагов сжатия на большинстве графов наиболее быстро работал алгоритм HEM, наиболее медленно —

RM на графах размера до 10^7, HCM — на графах размера более 10^7. При огрублении до 100 вершин наиболее быстро работал алгоритм HEM, наиболее медленно — алгоритм RM (в 1,1–3 раза больше по сравнению с HEM). На четырех самых больших матрицах дольше всего работал алгоритм HCM.

Если сравнивать два наиболее часто используемых алгоритма — RM и HEM, то HEM позволяет за меньшее время получить граф большего, чем у RM, размера, но меньшей плотности. Соотношение числа вершин полученного графа составляет 1,2–2 раза в пользу RM, соотношение плотностей — 1,5–5 раз в пользу HEM для большинства графов. По времени HEM работает в среднем на 40% быстрее при 10 шагах огрубления, в 2 раза быстрее при 20 шагах огрубления и огрублении до размера 100. При этом для исходных графов порядка 10^6 не достаточно 10 итераций алгоритма огрубления для получения графа размером не более 1000.

3.2.2. Алгоритмы разделения

Сравним работу алгоритмов разделения в тех экспериментах, где граф был сжат до 100 вершин. В данном разделе рассмотрим алгоритмы GG, GGG, KL, которые изначально находят реберный разделитель.

По размеру вершинного разделителя худшие результаты были получены алгоритмом GG при 5 перезапусках, лучшие — GGG и KL при 10 перезапусках для графов, полученных HCM и HEM (наиболее разреженных). При этом для трети матриц дисбаланс разделений, полученный различными алгоритмами, различается в пределах 10%. Для большинства матриц лучший дисбаланс разделения был получен алгоритмами GGG и KL. Увеличение числа перезапусков в 2 раза меньше всего влияет на качество алгоритма GGG, для которого улучшение дисбаланса составляет в среднем 5%, улучшение размера разделителя — 1%. Наиболее целесообразно увеличивать число перезапусков алгоритма GG, для которого улучшение дисбаланса и размера разделителя в среднем составляет 5%.

Сравним время работы алгоритмов разделения на первой задаче. Алгоритмы GGG, KL работают на один порядок дольше, чем GG (4–60 раз в зависимости от плотности матрицы), при этом KL работает в 1,5–2,7 раз медленнее, чем GGG. При этом порядок времени работы алгоритмов разделения на сжатых графах 10^{-3}–10^{-4} с. Увеличение в два раза числа перезапусков алгоритма GG увеличивает время его работы в среднем на 70%, но не более 5,2 раз; алгоритмов GGG и KL — в среднем в 2 раза, но не более 3,5 раз.

Алгоритм улучшения разделения является алгоритмом локальной оптимизации: он ищет минимум функции, зависящей от характеристик разделения — веса разделителя и дисбаланса частей. Для большинства графов в результате данной процедуры начальные разделения, полученные различными алгоритмами, были сведены к одинаковым разделениям. Исключение в основном составляют графы с вершиной большой степени. Поэтому можно рассматривать время, необходимое алгоритму улучшения разделения, как характеристику качества начального разделения графа — чем оно меньше, тем лучше.

В большинстве экспериментов наиболее быстро улучшение разделения выполнялось после разделения алгоритмом KL, наиболее медленно — после разделения LS и GGG. В 2/3 запусков для графов, полученных одним методом огрубления, время улучшения разделения после различных алгоритмов разделения различалось не более чем в 2 раза, в среднем — на 30%. Значительные различия во времени работы алгоритма улучшения

разделения наблюдались в случаях, когда граф, полученный огрублением, содержал вершину большой степени или был близким к плотному. Если сравнивать время работы улучшения разделения после 5 и 10 перезапусков алгоритмов разделения, то из экспериментов видно, что для 90% запусков разница во времени работы улучшения разделения была не более 1,5 раз, при этом в среднем разница составляла 4–13%.

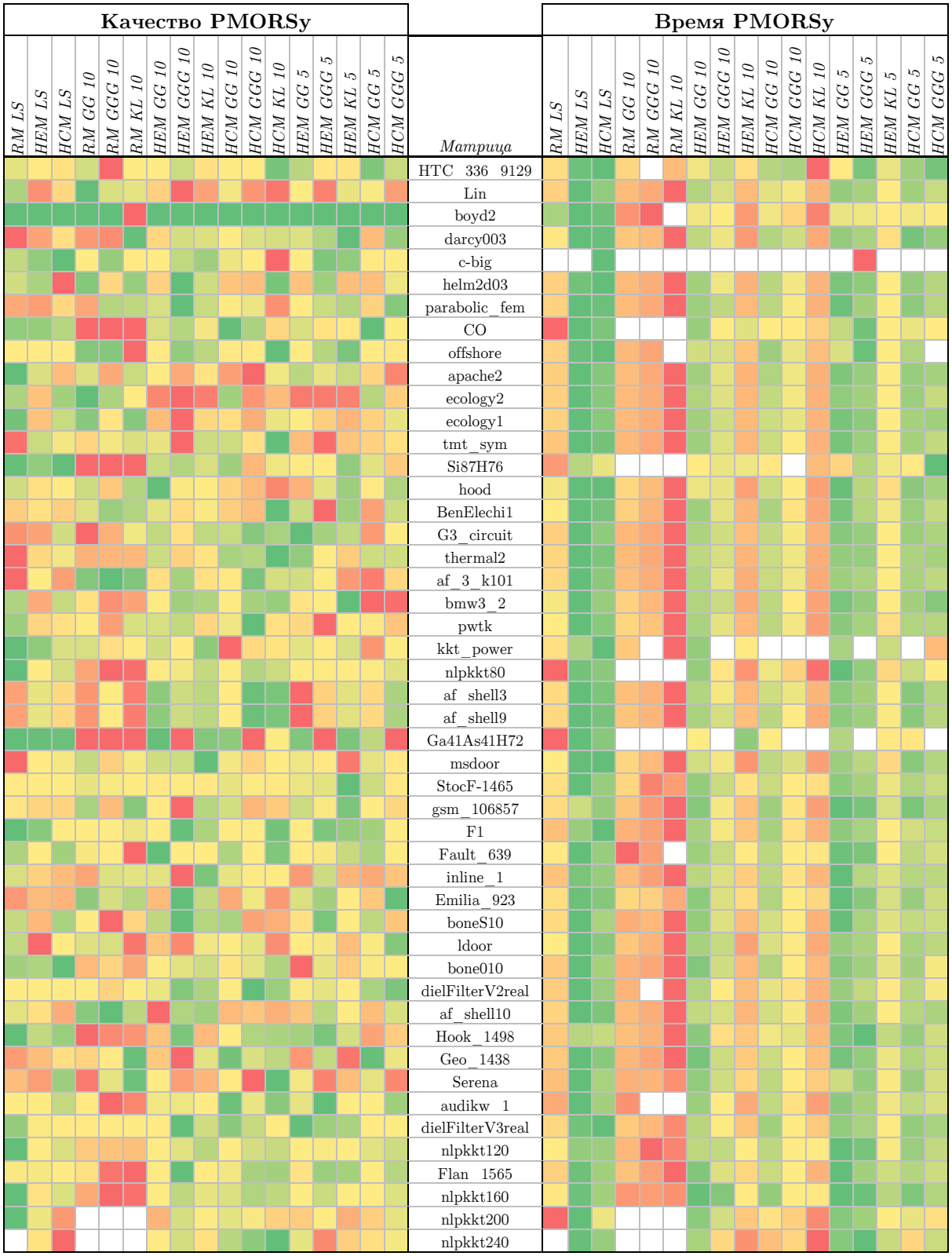
3.3. Сравнение алгоритмов в последовательной версии

Вычислительные эксперименты, поведенные для библиотеки PMORSy, запущенной в 1 поток, показали следующее:

1. Часть экспериментов привела к перестановкам с неприемлемым размером фактора (значительно большим, чем в других запусках). Основная причина такой работы переупорядочения — некорректная работа алгоритмов разделения и улучшения разделения на почти плотных графах и графах, содержащих вершину со степенью, значительно больше средней. Для таких графов не подходит использованный метод получения вершинного разделителя по реберному.
2. Перестановки с лучшим качеством были получены преимущественно при использовании алгоритмов огрубления HEM и HCM и любого метода разделения, а также огрублением RM с разделением LS. Алгоритмы HEM и HCM меньше других чувствительны к плотности начального графа и наличию вершин с большой степенью.
3. Для графов со степенями вершин менее 10 и небольшим разбросом степеней приемлема любая комбинация алгоритмов огрубления и разделения. Для большинства из них полученные перестановки различались по качеству в пределах 10%.
4. На качество получаемых перестановок в большей степени влияет алгоритм улучшения разделения, чем алгоритмы начального разделения. Это объясняется тем, что этот алгоритм выполняет дискретную оптимизацию, которая для разных начальных разделений находит близкие локально-оптимальные «улучшенные» разделения. При этом время работы улучшения разделения для различных начальных разделений примерно одинаково.
5. Графы с вершиной, степень которой значительно больше средней, а также графы большого порядка чувствительны к большому числу итераций огрубления и низкому ограничению по размеру графа, поскольку это приводит к слишком плотным графам в процессе огрубления. Для таких графов лучшие результаты были получены при минимальном графе в 100 вершин.
6. Использование большего числа перезапусков алгоритма разделения для большинства матриц незначительно улучшает конечное качество перестановок (около 5%). Среди рассмотренных алгоритмов более чувствительны к увеличению числа перезапусков алгоритмы GG и KL. При этом время работы переупорядочения увеличивается в среднем на 12%. Поэтому большое число запусков алгоритма разделения предлагается использовать для графов, содержащих вершины со степенью, значительно больше средней.
7. По соотношению общего времени работы переупорядочения быстрее всего работали запуски с алгоритмом LS, затем GG (+8%), GGG (+12%), KL (+20%).
8. Среди комбинаций параметров минимального размера графа и максимального числа шагов огрубления наиболее стабильными с точки зрения качества и небольшого времени работы были параметры $nSt = 10$ и $sm = 100$ (табл. 1).

Таблица 1

Время и качество переупорядочения PMORSy при работе в 1 поток с параметрами $nSt = 10$, $sm = 100$. Результаты приведены по цветовой шкале, где минимальному значению соответствует зеленый цвет, максимальному — красный цвет, промежуточным значениям — желтый цвет. Результаты, дающие неприемлемое заполнение фактора, отмечены белым



3.4. Сравнение алгоритмов в параллельной версии

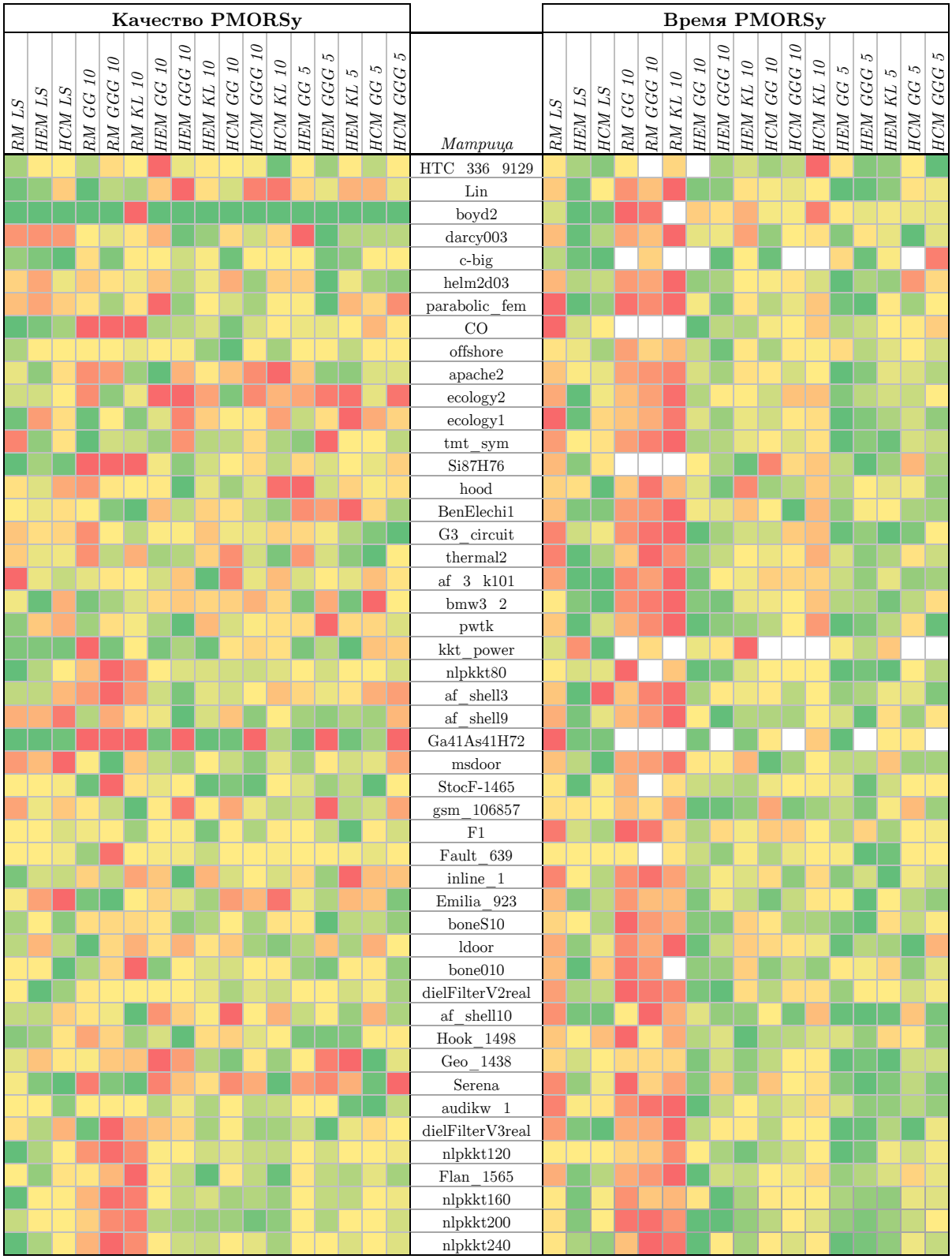
При запусках PMORSy в 16 потоков сохраняются тенденции по соотношению времени и качества, полученного разными алгоритмами, которые наблюдались при запусках в 1 поток. Дополнительно отметим следующее:

1. Перестановки с лучшим качеством были получены при использовании огрублений HEM и HCM, как и в однопоточном случае. Для матриц с числом ненулевых элементов в пределах $25{\times}10^6$ лучшими по качеству были эксперименты с минимальным графом размера 20, для матриц с большим числом ненулевых элементов — с минимальным графом размера 100.
2. Время работы PMORSy зависит в большей степени от примененного алгоритма огрубления, чем от алгоритма разделения. Так, во всех группах экспериментов быстрее всего работали запуски с огрублением HEM, затем — HCM (+ 6%), дольше всех — RM (+ 25% в среднем). Это связано с тем, что алгоритм RM требует дополнительной генерации массива случайных чисел по числу ребер графа.
3. Время работы переупорядочения при изменении алгоритма разделения графа в среднем увеличивается на 10–20%. В большинстве экспериментов быстрее всего работало переупорядочение с алгоритмом LS или GG с 5 перезапусками. Например, при использовании алгоритма огрубления HEM на большинстве матриц переупорядочение быстрее всего работало с разделением GG с 5 перезапусками, затем — GGG и KL с 5 перезапусками (+ 4–6%) и LS (+ 5%), дольше других работали эксперименты с KL с 10 перезапусками (+ 10%).
4. Использование большего числа перезапусков алгоритма разделения для большинства матриц также не дает значительного улучшения качества (средний выигрыш составляет 6%). При этом время работы переупорядочения для большинства матриц увеличивается, в среднем на 7%.
5. PMORSy при запусках в 16 потоков показывает масштабируемость 2–3,5 раза на матрицах порядка до $5{,}0{\times}10^6$; 3,5–4,5 раза на матрицах большего размера. Наибольшие значения масштабируемости достигаются на запусках с разделением KL, которые, однако, для большинства матриц работают дольше, чем запуски с другими разделениями.
6. Среди комбинаций параметров минимального размера графа и максимального числа шагов огрубления наиболее стабильными с точки зрения качества и небольшого времени работы были параметры $nSt = 10$ и $sm = 100$, как и при запусках в 1 поток, а также комбинация $nSt = 100$ и $sm = 100$.

Сравнение качества и времени работы для комбинации $nSt = 10$ и $sm = 100$ приведено ниже (табл. 2).

Таблица 2

Время и качество переупорядочения PMORSy при работе в 16 потоков с параметрами $nSt = 10$, $sm = 100$. Результаты приведены по цветовой шкале, где минимальному значению соответствует зеленый цвет, максимальному — красный цвет, промежуточным значениям — желтый цвет. Результаты, дающие неприемлемое качество, отмечены белым



3.5. Сравнение с библиотекой mt-metis

Сравним библиотеки PMORSy и mt-metis. По качеству переупорядочения для большинства матриц библиотеки показывают близкие результаты, отличающиеся в пределах 10% в пользу PMORSy или mt-metis, при этом соотношение качества близко для запусков с использованием разных алгоритмов. Исключение составляют матрицы, имеющие плотную строку (например, kkt_power, c-big и др.), на которых качество работы PMORSy не стабильно. Время работы PMORSy на 19 матрицах из 48 при этом меньше, чем у mt-metis, в 1,1–3 раза. На остальных матрицах в большинстве экспериментов PMORSy отстает от mt-metis в 1,1–3,5 раза. На самых больших матрицах из тестового набора PMORSy значительно отстает по времени от mt-metis (в среднем в 4,5 раза), однако при этом качество его переупорядочения на 30% лучше. Отметим, что результаты работы PMORSy на отдельных матрицах и классах матриц могут быть улучшены по времени и качеству при настройке параметров алгоритма улучшения разделения.

Сравнение качества и времени работы PMORSy и mt-metis при параметрах $nSt = 10$ и $sm = 100$ для алгоритмов разделения с 10 перезапусками приведено ниже (табл. 3).

Заключение

Переупорядочение строк и столбцов симметричных разреженных матриц — один из ключевых этапов в процессе прямого решения СЛАУ. Известно, что применяемые на данном этапе алгоритмы существенно влияют как на общее время решения задачи, так и на объем необходимой памяти. Проблема построения параллельных алгоритмов переупорядочения, сочетающих приемлемую эффективность масштабируемости с достаточным качеством получаемых перестановок, не является полностью решенной.

В данной работе изучается влияние разных алгоритмов огрубления и разделения графов на заполнение фактора и время работы параллельного метода вложенных сечений. В качестве экспериментальной платформы используется ранее представленная авторами параллельная библиотека PMORSy [15]. Данная библиотека реализует многоуровневый метод вложенных сечений, распараллеленный на основе логических задач OpenMP.

Одной из целей работы является повышение производительности используемого параллельного алгоритма при сохранении качества получаемых перестановок. Для этого был реализован ряд распространенных алгоритмов огрубления и разделения графов, изучены различные параметры алгоритмов, выполнено более 12 000 экспериментов на 48 симметричных матрицах из коллекции университета Флориды.

Эксперименты позволили сделать следующие выводы:

1. Среди алгоритмов огрубления наилучшие результаты по качеству и времени работы показал алгоритм паросочетания тяжелых ребер. Его использование в сравнении с другими алгоритмами позволило сократить время работы до 20% в зависимости от матрицы.
2. Рассмотренные алгоритмы разделения дают близкие по качеству результаты работы. Для большинства матриц переупорядочение выполнялось быстрее при использовании алгоритма жадного разделения или поиска в ширину из псевдопериферийной вершины. Для большинства экспериментов разница во времени работы при использовании разных алгоритмов разделения составляет около 10%.

Таблица 3

Отношение времени и качества работы PMORSy к работе mt-metis при запуске в 16 потоков. Параметры PMORSy $nSt = 10$, $sm = 100$. Наименьшие значения выделены зеленым цветом, наибольшие значения — красным цветом, промежуточные значения — желтым цветом. Результаты, дающие неприемлемое качество или время работы, отмечены белым

Качество PMORSy / mt-metis														**Время PMORSy / mt-metis**											
RM LS	*HEM LS*	*HCM LS*	*RM GG 10*	*RM GGG 10*	*RM KL 10*	*HEM GG 10*	*HEM GGG 10*	*HEM KL 10*	*HCM GG 10*	*HCM GGG 10*	*HCM KL 10*	*Матрица*	*RM LS*	*HEM LS*	*HCM LS*	*RM GG 10*	*RM GGG 10*	*RM KL 10*	*HEM GG 10*	*HEM GGG 10*	*HEM KL 10*	*HCM GG 10*	*HCM GGG 10*	*HCM KL 10*
1,2	1,4	1,3	1,3	1,7	1,3		1,3	1,4	1,3	1,3	1,2	HTC_336_9129	3,3	2,6	2,5	3,1		4,0	2,5	2,6	2,8	2,6	2,7	9,3
0,9	0,9	1,0	0,9	0,9	0,9	1,0	1,1	1,0	1,0	1,0	1,0	Lin	2,2	1,8	2,0	2,5	2,2	2,8	1,8	1,8	2,0	1,9	1,9	2,0
1,0	1,0	1,0	1,0	1,0		1,0	1,0	1,0	1,0	1,0	1,0	boyd2	0,9	0,6	0,6	1,2	1,2	0,7	1,0	1,0	1,1	1,0	1,0	1,1
1,4	1,4	1,4	1,3	1,3	1,3	1,3	1,3	1,3	1,3	1,3	1,3	darcy003	3,0	2,4	2,5	3,1	2,9	3,5	2,6	2,6	3,1	2,5	2,7	2,7
1,0	0,9	0,8		0,9		1,7	1,1	1,3	0,9	1,6	1,9	c-big	4,8	2,9	2,5					2,6	6,7	2,8		
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	helm2d03	3,2	2,7	2,7	3,3	3,2	3,6	2,6	2,7	2,9	2,9	2,9	2,8
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	parabolic_fem	3,7	2,7	2,8	3,7	3,5	3,7	3,0	2,7	3,1	2,9	3,0	3,2
0,7	0,7	0,8	2,3	2,3	2,3	0,8	0,8	0,8	0,7	0,8	0,9	CO		1,0	1,0				0,9	1,0	1,0	1,0	1,2	7,6
1,2	1,4	1,3	1,4	1,3	1,2	1,3	1,3	1,2	1,2	1,5	1,2	offshore	2,7	2,6	2,5	3,5	2,9	3,2	2,5	2,4	2,7	2,5	2,6	2,6
0,9	0,9	0,9	1,0	1,0	0,9	0,8	1,0	0,9	1,0	1,0	1,0	apache2	3,0	2,7	2,7	3,1	3,2	3,3	2,7	2,6	2,8	2,8	2,7	2,8
0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	ecology2	3,7	2,9	3,2	3,6	3,8	4,1	3,1	3,3	3,3	3,2	3,6	3,6
0,8	0,9	0,9	0,8	0,9	0,8	0,9	0,9	0,9	0,9	0,9	0,9	ecology1	4,4	2,8	3,4	3,5	3,7	4,2	3,1	2,9	3,1	3,1	3,2	3,3
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	tmt_sym	3,7	3,2	3,3	3,7	3,8	4,0	3,1	3,1	3,2	3,2	3,2	3,3
0,8	0,8	0,8	2,3	2,3	2,3	0,9	0,8	0,8	0,9	1,1	0,8	Si87H76	3,3	2,1	2,5				2,4	2,2	2,0	4,3	2,7	3,1
1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	hood	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3
0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	BenElechi1	0,4	0,3	0,3	0,4	0,4	0,4	0,3	0,3	0,4	0,3	0,3	0,4
1,0	1,0	1,0	1,1	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	G3_circuit	3,4	2,9	3,0	3,4	3,5	3,5	2,7	2,9	2,9	2,9	3,1	3,2
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	thermal2	4,0	3,0	3,2	3,6	4,1	3,9	3,2	3,3	3,4	3,4	3,5	3,8
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	af_3_k101	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
1,1	1,0	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	bmw3_2	0,5	0,5	0,5	0,6	0,6	0,7	0,5	0,5	0,5	0,5	0,5	0,6
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	pwtk	0,4	0,4	0,4	0,5	0,5	0,5	0,4	0,4	0,4	0,4	0,4	0,5
1,0	1,0	1,0		1,0	1,7	1,0	1,3	1,0	1,9	2,3	2,0	kkt_power	3,3	3,8	2,8	4,2	3,5		3,4	3,5	3,9			
0,7	0,7	0,7	1,0	1,7	1,4	0,7	0,7	0,7	0,7	0,7	0,7	nlpkkt80	3,9	3,1	3,2				3,0	3,0	3,2	3,2	3,2	3,4
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	af_shell3	0,5	0,4	0,6	0,5	0,6	0,6	0,5	0,5	0,5	0,5	0,5	0,5
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	af_shell9	0,6	0,5	0,5	0,6	0,6	0,6	0,5	0,5	0,5	0,5	0,5	0,5
0,8	0,8	0,9	2,4	2,4	2,4	0,9	2,4	0,8	0,9	2,4	1,4	Ga41As41H72		2,6	2,6				2,6		2,7	6,9		
1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	msdoor	0,3	0,3	0,3	0,3	0,3	0,4	0,3	0,3	0,3	0,3	0,3	0,3
1,0	1,0	1,0	0,9		1,3	1,0	1,0	0,9	0,9	0,9	1,0	StocF-1465	3,6	2,8	3,2	8,0	4,2	4,3	3,0	3,0	3,0	3,1	3,2	3,2
1,2	1,1	1,2	1,1	1,1	1,1	1,1	1,2	1,1	1,2	1,1	1,1	gsm_106857	1,5	1,5	1,5	1,6	1,5	1,6	1,3	1,3	1,4	1,6	1,3	1,4
1,2	1,2	1,1	1,2	1,2	1,1	1,2	1,2	1,1	1,2	1,1	1,2	F1	1,2	1,0	0,9	1,2	1,2	1,1	0,9	1,1	1,0	1,1	1,1	1,0
0,9	0,9	0,9	0,8	2,7	0,9	0,9	0,9	0,9	0,9	0,9	0,9	Fault_639	1,0	0,8	0,8	1,0		1,3	0,8	0,7	0,8	0,8	0,8	0,9
1,2	1,2	1,2	1,2	1,2	1,2	1,2	1,2	1,2	1,2	1,2	1,2	inline_1	0,9	0,7	0,7	0,9	1,0	0,9	0,7	0,7	0,7	0,7	0,6	0,7
0,9	0,9	0,9	0,8	0,8	0,9	0,9	0,9	0,8	0,9	0,9	0,9	Emilia_923	1,1	0,8	0,8	1,2	1,0	1,0	0,8	0,8	0,8	0,8	0,8	0,8
1,1	1,2	1,1	1,2	1,3	1,3	1,2	1,1	1,1	1,2	1,1	1,2	boneS10	1,0	0,9	0,9	1,2	1,1	1,0	0,9	0,9	1,0	0,9	0,9	0,9
1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	ldoor	0,4	0,3	0,4	0,4	0,4	0,4	0,3	0,3	0,4	0,4	0,4	0,4
0,9	0,9	0,9	0,9	1,0	1,7	0,9	0,9	0,9	0,9	0,9	0,9	bone010	1,1	0,8	1,0	1,2	1,2		0,8	0,8	1,0	0,8	0,9	0,8
1,1	1,0	1,0	1,3	1,1	1,1	1,1	1,1	1,0	1,0	1,0	1,0	dielFilterV2real	1,5	1,2	1,2	1,7	1,6	1,6	1,2	1,2	1,3	1,2	1,3	1,3
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	af_shell10	0,2	0,2	0,2	0,2	0,3	0,2	0,2	0,2	0,2	0,2	0,2	0,2
0,9	0,9	1,0	1,1	1,0	0,9	0,9	0,9	1,0	1,0	0,9	0,9	Hook_1498	1,2	1,1	1,2	1,3	1,1	1,2	1,0	1,1	0,9	1,0	1,0	1,1
0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	Geo_1438	1,0	0,8	0,9	1,0	1,0	1,0	0,8	0,8	0,8	0,8	0,9	0,9
1,0	0,9	0,9	1,0	0,9	0,9	1,0	1,0	1,0	1,0	1,0	0,9	Serena	1,3	0,8	0,9	1,4	1,0	1,2	0,8	1,1	0,8	1,1	0,9	1,0
1,0	1,0	1,0	1,0	1,4	1,0	1,0	1,0	1,0	1,0	1,0	1,0	audikw_1	0,9	0,8	0,8	0,9	0,9	0,9	0,7	0,7	0,8	0,7	0,8	0,8
1,1	1,1	1,1	1,0	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1	dielFilterV3real	0,7	0,6	0,6	0,7	0,7	0,7	0,6	0,6	0,6	0,6	0,6	0,6
0,7	0,7	0,7	0,9	1,0	0,9	0,7	0,7	0,7	0,7	0,7	0,7	nlpkkt120	4,3	3,8	3,8				4,0	3,6	3,4	3,6	3,7	4,0
0,8	0,8	0,8	0,8	0,9	0,9	0,8	0,8	0,8	0,8	0,8	0,8	Flan_1565	0,9	0,8	0,8	0,9	1,0	1,0	0,8	0,8	0,8	0,9	0,9	0,9
0,7	0,7	0,7	0,9	1,2	1,1	0,7	0,7	0,7	0,7	0,7	0,7	nlpkkt160	4,1	3,5	3,8	8,7	5,9	6,7	4,3	3,4	3,6	4,1	4,0	3,9
0,7	0,7	0,7	0,8	1,0	1,0	0,7	0,7	0,7	0,7	0,7	0,7	nlpkkt200	5,2	3,7	4,2				3,6	3,6	3,8	3,9	4,3	4,1
0,6	0,6	0,6	0,7	0,8	0,8	0,6	0,6	0,6	0,6	0,6	0,6	nlpkkt240	5,4	4,1	4,0	9,9			3,7	4,7	4,0	4,9	4,3	5,0

3. Работа алгоритма улучшения разделения оказывает большее влияние на качество и время переупорядочения, чем работа алгоритма разделения.
4. Для матриц с малым заполнением (до 25×10^6) целесообразно устанавливать малый порог на размер графа, к которому применяется многоуровневый алгоритм. Для матриц с большим заполнением использование малого порога приводит к существенной потере качества переупорядочения.
5. Существенным фактором, ограничивающим масштабирование текущей реализации PMORSy, является последовательная работа отдельных алгоритмов многоуровневого метода вложенных сечений на первых шагах работы метода.

В целом по результатам выполненной работы удалось выбрать комбинацию алгоритмов и их параметров, преимущественно дающую приемлемые результаты по времени и качеству получаемых перестановок. В дальнейшем планируется разработать двухуровневую схему распараллеливания, сочетающую параллелизм по задачам и внутреннее распараллеливание отдельных алгоритмов. Ожидаемый прирост производительности связан с лучшим использованием ресурсов параллелизма при решении наиболее трудоемких подзадач, возникающих в начале работы метода вложенных сечений.

Работа частично поддержана компанией Intel и Минобрнауки РФ (соглашение № 1.115.2014/K). Авторы благодарят Александра Калинкина за полезные обсуждения и внимание к работе.

Литература

1. Bui T., Jones C. A Heuristic for Reducing Fill in Sparse Matrix Factorization // 6th SIAM Conference Parallel Processing for Scientific Computing. 1993. P. 445–452.
2. Chevalier C., Pellegrini F. PT-Scotch: A Tool for Efficient Parallel Graph Ordering // Parallel Computing. 2008. Vol. 34, No. 6. P. 318–331. DOI: 10.1016/j.parco.2007.12.001.
3. Davis T. A., Hu Y. The University of Florida Sparse Matrix Collection // ACM Transactions on Mathematical Software (TOMS). 2011. Vol. 38, No. 1. P. 1. DOI: 10.1145/2049662.2049663.
4. George A. et al. Sparse Cholesky Factorization on a Local-memory Multiprocessor //SIAM J. on Scientific and Statistical Computing. 1988. Vol. 9, No. 2. P. 327–340. DOI: 10.1016/0167-8191(92)90014-x.
5. George A. Nested Dissection of a Regular Finite Element Mesh // SIAM J. on Numerical Analysis. 1973. Vol. 10, No. 2. P. 345–363. DOI: 10.1137/0710032.
6. George A., Liu J.W.H. An Automatic Nested Dissection Algorithm for Irregular Finite Element Problems // SIAM J. on Numerical Analysis. 1978. Vol. 15, No. 5. P. 1053–1069. DOI: 10.1137/0715069.
7. Karypis G., Kumar V. ParMetis: Parallel Graph Partitioning and Sparse Matrix Ordering Library. Tech. Rep. TR 97-060, University Of Minnesota, Department Of Computer Science. 1997.
8. Karypis G., Kumar V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs // SIAM J. on Scientific Computing. 1998. Vol. 20, No. 1. P. 359–392. DOI: 10.1137/s1064827595287997.
9. Karypis G., Kumar V. Analysis of Multilevel Graph Partitioning // Proceedings of the 1995 ACM/IEEE conference on Supercomputing. ACM, 1995. P. 29.

10. Karypis G., Kumar V. METIS. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-reducing Orderings of Sparse Matrices. Technical Report University of Minnesota, Department of Computer Science and Engineering. 1998.
11. LaSalle D., Karypis G. Efficient Nested Dissection for Multicore Architectures // Euro-Par 2015: Parallel Processing. 2015, Springer Berlin Heidelberg. P. 467–478.
12. Pellegrini F. Scotch and libScotch 6.0 User's Guide. Technical Report LaBRI, 2012.
13. Pellegrini F. Shared Memory Parallel Algorithms in Scotch 6 // MUMPS User Group Meeting. 2013. URL: http://graal.ens-lyon.fr/mumps/doc/ud_2013/pellegrini.pdf (дата обращения: 01.12.2016)
14. Pirova A., Meyerov I. MORSy — a New Tool for Sparse Matrix Reordering // An International Conference on Engineering and Applied Sciences Optimization M. Papadrakakis, M.G. Karlaftis, N.D. Lagaros (eds.) (Kos Island, Greece, 4-6 June 2014). P. 1952–1964.
15. Pirova A., Meyerov I., Kozinov E., Lebedev S. PMORSy: Parallel Sparse Matrix Ordering Software for Fill-in Minimization // Optimization Methods and Software. 2017. Vol. 32, No. 2. P. 274–289. DOI: 10.1080/10556788.2016.1193177.
16. Pothen A. Graph Partitioning Algorithms with Applications to Scientific Computing // Parallel Numerical Algorithms. Springer Netherlands, 1997. P. 323–368. DOI: 10.1007/978-94-011-5412-3_12.
17. Tinney W., Walker J. Direct Solutions of Sparse Network Equations by Optimally Ordered Triangular Factorization // Proceedings of the IEEE. 1967. Vol. 55, No. 11. P. 1801–1809. DOI: 10.1109/proc.1967.6011.
18. Yannakakis M. Computing the Minimum Fill-in is NP-complete // SIAM J. on Algebraic and Discrete Methods. 1981. Vol. 2, No. 1. P. 77–79. DOI: 10.1137/0602010.
19. Пирова А.Ю., Мееров И.Б., Козинов Е.А., Лебедев С.А. Параллельный алгоритм многоуровневого метода вложенных сечений для вычислительных систем с общей памятью // Вычислительные методы и программирование. 2015. Т. 16. № 3. С. 407–420.
20. Старостин Н.В. Многоуровневый итерационный алгоритм декомпозиции графа. // Системы управления и информационные технологии. 2015. Т. 61. № 3. С. 27–30.

Пирова Анна Юрьевна, ассистент, кафедра математического обеспечения и суперкомпьютерных технологий, Институт информационных технологий, математики и механики, Нижегородский государственный университет им. Н.И. Лобачевского (Нижний Новгород, Российская Федерация)

Кудрявцев Никита Юрьевич, магистрант, кафедра математического обеспечения и суперкомпьютерных технологий, Институт информационных технологий, математики и механики, Нижегородский государственный университет им. Н.И. Лобачевского (Нижний Новгород, Российская Федерация)

Мееров Иосиф Борисович, к.т.н., доцент, зам. заведующего кафедрой, кафедра математического обеспечения и суперкомпьютерных технологий, Институт информационных технологий, математики и механики, Нижегородский государственный университет им. Н.И. Лобачевского (Нижний Новгород, Российская Федерация)

DOI: 10.14529/cmse170103

EXPERIMENTAL EVALUATION OF ALGORITHMS IN THE PARALLEL MULTILEVEL NESTED DISSECTION METHOD

© 2017 A.Yu. Pirova, N.Yu. Kudriavtsev, I.B. Meyerov

Lobachevsky State University of Nizhni Novgorod

(23 Gagarin Avenue, Nizhni Novgorod, 603022 Russia)

E-mail: anna.pirova@itmm.unn.ru, n.yu.kudriavtsev@gmail.com, iosif.meyerov@itmm.unn.ru

Received: 20.10.2016

Direct methods for solving large sparse systems of linear equations make use of reordering of rows and columns of the original matrix. The goal of this procedure is to reduce the fill-in during the subsequent numerical factorization. Finding the ordering with the minimum fill-in is NP-complete. Heuristic methods are used to solve this problem. These methods can be evaluated for both quality (fill-in) and time to obtain an ordering. The multilevel nested dissection method performs reasonably well in terms of both criteria and is one of the most widely used reordering methods. The method has some parallelization potential, which is utilized in several implementations (ParMETIS, mtMETIS, PT-SCOTCH, PMORSy). However, low arithmetic intensity, irregular memory access pattern, workload imbalance and the trade-off between run time and quality motivates further investigation of the method.

This paper presents the comparison of the algorithms used on several stages of the multilevel nested dissection method in terms of fill-in and run time on a parallel system. The implementation and experiments are done using the parallel PMORSy library, which outperforms competitors on some matrices from the University of Florida sparse matrix collection. As the result we distinguish the most promising combination of the algorithms and improve the quality and performance of PMORSy.

Keywords: multilevel nested dissection, sparse matrix ordering, parallel algorithm.

FOR CITATION

Pirova A.Yu., Kudriavtsev N.Yu., Meyerov I.B. Experimental Evaluation of Algorithms in the Parallel Multilevel Nested Dissection Method. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2017. vol. 6, no. 1. pp. 38–55. (in Russian) DOI: 10.14529/cmse170103.

References

1. Bui T., Jones C. A Heuristic for Reducing Fill in Sparse Matrix Factorization. *6th SIAM Conference Parallel Processing for Scientific Computing*. 1993. pp. 445–452.

2. Chevalier C., Pellegrini F. PT-Scotch: A Tool for Efficient Parallel Graph Ordering. *Parallel Computing*. 2008. vol. 34, no. 6. pp. 318–331. DOI: 10.1016/j.parco.2007.12.001.

3. Davis T. A., Hu Y. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software (TOMS)*. 2011. vol. 38, no. 1. pp. 1. DOI: 10.1145/2049662.2049663.

4. George A. et al. Sparse Cholesky Factorization on a Local-memory Multiprocessor. *SIAM J. on Scientific and Statistical Computing*. 1988. vol. 9, no. 2. pp. 327–340. DOI: 10.1016/0167-8191(92)90014-x.

5. George A. Nested Dissection of a Regular Finite Element Mesh. *SIAM J. on Numerical Analysis*. 1973. vol. 10, no. 2. pp. 345–363. DOI: 10.1137/0710032.

6. George A., Liu J.W.H. An Automatic Nested Dissection Algorithm for Irregular Finite Element Problems. *SIAM J. on Numerical Analysis*. 1978. vol. 15, no. 5. pp. 1053–1069. DOI: 10.1137/0715069.

7. Karypis G., Kumar V. *ParMetis: Parallel Graph Partitioning and Sparse Matrix Ordering Library*. Tech. Rep. TR 97-060, University Of Minnesota, Department Of Computer Science. 1997.

8. Karypis G., Kumar V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. on Scientific Computing*. 1999. vol. 20, no. 1. pp. 359–392. DOI: 10.1137/s1064827595287997.

9. Karypis G., Kumar V. Analysis of Multilevel Graph Partitioning. *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*. ACM, 1995. pp. 29.

10. Karypis G., Kumar V. *METIS. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-reducing Orderings of Sparse Matrices*. University of Minnesota, Department of Computer Science and Engineering. 1998.

11. LaSalle D., Karypis G. Efficient Nested Dissection for Multicore Architectures. *Euro-Par 2015: Parallel Processing*. 2015, Springer Berlin Heidelberg. pp. 467–478.

12. Pellegrini F. *Scotch and libScotch 6.0 User's Guide*. Technical Report LaBRI, 2012.

13. Pellegrini F. Shared Memory Parallel Algorithms in Scotch 6. *MUMPS User Group Meeting*. 2013. Available at: http://graal.ens-lyon.fr/mumps/doc/ud_2013/pellegrini.pdf (accessed: 01.12.2016)

14. Pirova A., Meyerov I. MORSy — a New Tool for Sparse Matrix Reordering. *An International Conference on Engineering and Applied Sciences Optimization*. M. Papadrakakis, M.G. Karlaftis, N.D. Lagaros (eds.) Kos Island, Greece, 4–6 June 2014. pp. 1952–1964.

15. Pirova A., Meyerov I., Kozinov E., Lebedev S. PMORSy: Parallel Sparse Matrix Ordering Software for Fill-in Minimization. *Optimization Methods and Software*. 2017. vol. 32, no. 2, pp. 274–289. DOI: 10.1080/10556788.2016.1193177.

16. Pothen A. Graph Partitioning Algorithms with Applications to Scientific Computing. *Parallel Numerical Algorithms*. Springer Netherlands, 1997. pp. 323–368. DOI: 10.1007/978-94-011-5412-3_12.

17. Tinney W., Walker J. Direct Solutions of Sparse Network Equations by Optimally Ordered Triangular Factorization. *Proceedings of the IEEE*. 1967. vol. 55, no. 11. pp. 1801–1809. DOI: 10.1109/proc.1967.6011.

18. Yannakakis M. Computing the Minimum Fill-in is NP-complete. *SIAM J. on Algebraic and Discrete Methods*. 1981. vol. 2, no. 1. pp. 77–79. DOI: 10.1137/0602010.

19. Pirova A.Yu., Meyerov I.B., Kozinov E.A., Lebedev S.A. A Parallel Multilevel Nested Dissection Algorithm for Shared-memory Computing Systems. *Numerical Methods and Programming*. 2015. vol. 16, no. 3. pp. 407–420. (in Russian)

20. Starostin N.V. The Multilevel Iteration Algorithm for Graph Decomposition. *Sistemy upravlenija i informacionnye tehnologii* [Operation Systems and Information Technologies]. 2015. vol. 61, no. 3. pp. 27–30. (in Russian)