УДК 519.178:519.612.2 doi 10.26089/NumMet.v16r339

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ МНОГОУРОВНЕВОГО МЕТОДА ВЛОЖЕННЫХ СЕЧЕНИЙ ДЛЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ С ОБЩЕЙ ПАМЯТЬЮ

А. Ю. Пирова[1], И. Б. Мееров[2], Е. А. Козинов[3], С. А. Лебедев[4]

Рассматривается задача переупорядочения строк и столбцов симметричной положительно определенной разреженной матрицы с целью уменьшения числа ненулевых элементов в факторе Холецкого. Данная задача является NP-полной. Для ее решения используются эвристические алгоритмы, основанные на применении методов теории графов. Предлагается параллельный алгоритм переупорядочения для вычислительных систем с общей памятью. В качестве базы для распараллеливания используется модификация многоуровневого метода вложенных сечений, ранее реализованная авторами в виде библиотеки с открытым исходным кодом MORSy. Основная идея распараллеливания заключается в организации и параллельной обработке очереди задач, которые могут быть решены независимо. В отличие от широко распространенных аналогов, применяющих MPI для организации параллелизма как на распределенной, так и на общей памяти, предложенный алгоритм использует возможности стандарта OpenMP 3.0. Вычислительные эксперименты выполнены на симметричных положительно определенных матрицах из коллекции университета Флориды. Показано, что параллельный код MORSy дает сходные или лучшие перестановки в сравнении с библиотекой ParMETIS для всех тестовых матриц, кроме одной, в большинстве случаев опережая ParMETIS по времени работы. Программная реализация выполнена в виде библиотеки с открытым исходным кодом и доступна для скачивания на сайте Приволжского научно-образовательного центра суперкомпьютерных технологий.

Ключевые слова: многоуровневый метод вложенных сечений, переупорядочение разреженной матрицы, разложение Холецкого, параллельные алгоритмы, вычислительные системы с общей памятью, высокопроизводительные вычисления.

1. Введение. При выполнении конечно-элементных расчетов нередко возникает задача решения систем линейных алгебраических уравнений (СЛАУ) $Ax = b$ с симметричными положительно определенными разреженными матрицами большого порядка. Использование прямых методов для решения таких СЛАУ часто сводится к треугольной факторизации матрицы системы и последующему решению двух треугольных систем. При построении фактора методом Холецкого число ненулевых элементов может значительно увеличиться. Для того чтобы снизить заполнение фактора ненулевыми элементами, к исходной матрице применяется процедура переупорядочения ее строк и столбцов. Применение указанной процедуры позволяет уменьшить затраты времени и памяти для построения и хранения фактора. Таким образом, число ненулевых элементов в факторе может рассматриваться в качестве одного из критериев успешности процедуры переупорядочения. Тем не менее, "качество переупорядочения", определяемое как число ненулевых элементов в факторе, не является единственным критерием. Так, требуется дополнительная проверка того, в какой степени перестановка строк и столбцов повлияла на потенциал параллелизма в рамках наиболее вычислительно трудоемкой численной фазы разложения Холецкого. Другим важным критерием является время, затраченное на само переупорядочение. Несмотря на существование задач, в которых данное время не является критичным (например, решение серии СЛАУ с матрицами одинаковой

[1] Нижегородский государственный университет им. Н. И. Лобачевского (ННГУ), факультет вычислительной математики и кибернетики, просп. Гагарина, 23, 603950, г. Нижний Новгород; мл. науч. сотр., e-mail: anna.yu.malova@gmail.com

[2] Нижегородский государственный университет им. Н. И. Лобачевского (ННГУ), факультет вычислительной математики и кибернетики, просп. Гагарина, 23, 603950, г. Нижний Новгород; зам. зав. кафедрой, e-mail: meerov@vmk.unn.ru

[3] Нижегородский государственный университет им. Н. И. Лобачевского (ННГУ), факультет вычислительной математики и кибернетики, просп. Гагарина, 23, 603950, г. Нижний Новгород; ассистент, e-mail: evgeniy.kozinov@gmail.com

[4] Нижегородский государственный университет им. Н. И. Лобачевского (ННГУ), факультет вычислительной математики и кибернетики, просп. Гагарина, 23, 603950, г. Нижний Новгород; программист, e-mail: sergey.a.lebedev@gmail.com

структуры, где переупорядочение может быть выполнено только один раз), сокращение затрат времени на рассматриваемую фазу решения является актуальным. В настоящей статье рассматривается вопрос о возможности ускорения фазы переупорядочения за счет использования многоядерных вычислительных систем с общей памятью.

Известно, что нахождение перестановки, минимизирующей число ненулевых элементов в факторе, является NP-полной задачей [1]. В практических приложениях для ее решения применяются две группы эвристических методов — методы минимальной степени [2] и методы вложенных сечений [3]. В данной работе рассматривается задача построения параллельного алгоритма вложенных сечений для систем с общей памятью. В настоящий момент существует ряд успешных программных реализаций (библиотеки ParMETIS и PT-Scotch), предназначенных прежде всего для систем с распределенной памятью и использующих те же принципы распараллеливания и технологию MPI, что и в системах с общей памятью. В нашей работе показано, что применение ориентированных на многоядерные системы методов и средств распараллеливания позволяет добиться лучших результатов. Предлагаемый нами подход основан на распараллеливании по задачам на верхнем уровне метода вложенных сечений. Программная реализация использует механизм логических задач, включенный в стандарт OpenMP 3.0, и планировщик, встроенный в компиляторы C++. При разработке параллельного алгоритма за основу взята ранее созданная авторами библиотека с открытым исходным кодом MORSy [4], в которой для переупорядочения разреженных матриц используется многоуровневый метод вложенных сечений с модификацией отдельных стадий.

Статья построена следующим образом. Раздел 2 содержит обзор методов переупорядочения разреженных матриц для минимизации заполнения фактора и обосновывает выбор метода вложенных сечений. В разделе 3 приведен обзор распространенных программных библиотек для решения рассматриваемой задачи и дана краткая характеристика параллельных алгоритмов, содержащихся в этих библиотеках. Раздел 4 содержит формальную постановку задачи минимизации заполнения фактора. В разделе 5 кратко описан последовательный алгоритм переупорядочения в библиотеке MORSy, его подробное описание может быть найдено в работе [4]. В разделе 6 приведено описание предлагаемого параллельного алгоритма переупорядочения. Раздел 7 содержит результаты экспериментов разработанной параллельной библиотеки на восьмиядерном вычислительном узле и их анализ. В разделе 8 обобщены результаты работы и приведены планы дальнейших исследований.

2. Обзор методов переупорядочения. Для переупорядочения разреженной матрицы с целью минимизации заполнения фактора при разложении Холецкого используют две группы методов — методы минимальной степени и методы вложенных сечений.

Метод минимальной степени (Minimum Degree, MD) был предложен Тиннеем и Уолкером [2] в 1967 г. Этот метод в некотором смысле моделирует процесс симметричного гауссова исключения. Выбор вершины, исключаемой из графа на каждом шаге алгоритма, соответствует выбору ведущего элемента в строках и столбцах, обеспечивающих добавление наименьшего числа ненулевых элементов. Для данного метода наиболее затратными операциями являются преобразование графа и пересчет степеней. С начала 1980-х гг. были разработаны модификации метода, направленные на сокращение объема вычислений при пересчете степеней, в том числе *множественный метод минимальной степени* (Multiple Minimum Degree) [5], *приближенный метод минимальной степени* (Approximate Minimum Degree) [6] и *столбцовый метод минимальной степени* (Column Approximate Minimum Degree) [7].

Метод вложенных сечений (Nested Dissection, ND) был предложен Джорджем [3] в 1973 г. для решения матричных задач, возникающих в конечно-разностных и конечно-элементных приложениях. В 1978 г. Липтон, Роуз и Тарьян [8] и Джордж и Лю [9] обобщили алгоритм для задач с нерегулярной сеткой. Метод вложенных сечений основан на многократном разбиении графа матрицы при помощи вершинных разделителей. На каждом шаге алгоритма вычисляется разделитель графа (т.е. множество вершин, после удаления которых граф распадется на две или более несвязные части). Вершины разделителя нумеруются и удаляются из графа. Затем данная процедура повторяется рекурсивно для каждой новой связной компоненты графа до тех пор, пока не будут пронумерованы все вершины. Модификации метода различаются, в первую очередь, способами выделения разделителя.

Так, в методе *автоматических вложенных сечений* (Automatic Nested Dissection) [3] разделитель выбирается из середины структуры уровней смежности графа, построенной с корнем в псевдопериферийной вершине. Потен, Симон и соавторы в 1992 г. предложили *спектральный метод вложенных сечений* (Spectral Nested Dissection) [10], который использует информацию о собственных значениях матрицы. Группа методов вложенных сечений использует геометрическую информацию о вершинах. Метод *декартовых вложенных сечений* (Cartesian Parallel Nested Dissection) [11] был предложен Хифом и Рагхаван в 1991 г. Авторы метода использовали двумерные декартовы координаты для нахождения разделителя.

Геометрический метод разделения графа был предложен Миллером и соавторами в 1991 г. [12]. Начиная с 1993 г. широко используются модификации метода вложенных сечений, основанные на многоуровневой процедуре разделения графа. Большинство из них используют специализированные алгоритмы для разделения графов и улучшения разделения. Многоуровневый метод вложенных сечений был предложен Бьюи и Джонсом в 1993 г. [13] и развит в работах Хендриксона и Леланда [14], Карписа и Кумара [15], Хендриксона и Ротберга [16] и др.

И метод минимальной степени, и метод вложенных сечений широко используются в последовательных библиотеках для решения систем линейных уравнений и в специализированных библиотеках для переупорядочения разреженных матриц. Однако распараллеливание метода минимальной степени затруднено ввиду последовательной структуры алгоритма. Напротив, метод вложенных сечений, основанный на принципе "разделяй и властвуй", имеет потенциал для параллельного выполнения и порождает перестановки, подходящие для дальнейшей параллельной обработки фактора матрицы. Известно, что перестановки, полученные с помощью метода вложенных сечений, обладают бóльшим потенциалом параллелизма на этапе численной факторизации, чем перестановки, полученные с помощью метода минимальной степени [17]. В связи с указанными причинами метод вложенных сечений является базовым для данной работы.

3. Обзор программных средств для переупорядочения разреженных матриц. Среди реализаций методов переупорядочения необходимо отметить программный пакет METIS [18], широко применяемый в научных и инженерных приложениях, включая Intel MKL PARDISO [19] — один из лучших прямых решателей разреженных СЛАУ. В качестве последовательного алгоритма переупорядочения METIS предоставляет разновидность многоуровневого метода вложенных сечений, описанную в работе [15]. В библиотеке реализован ряд методов для разделения графов и их модификации для поиска вершинных разделителей. По умолчанию, в методе вложенных сечений используются метод паросочетания тяжелых ребер на этапе огрубления, жадный метод наращивания графа (Greedy Graph Growing Partitioning) на этапе разделения и модификации метода Фидуччиа–Мэтьюза на этапе улучшения разделения [15].

В пакете ParMETIS [20] содержится параллельная редакция библиотеки METIS, описанная в работах [21, 22]. Авторы библиотеки используют MPI для распараллеливания алгоритмов, не предоставляя специальных решений для систем с общей памятью. Переупорядочение матрицы p процессами в библиотеке ParMETIS выполняется по следующему принципу. Вначале все процессы выполняют поиск разделителя исходной матрицы. В результате в графе матрицы выделяются два независимых подграфа, не считая самого разделителя. Далее одна половина процессов приступает к поиску разделителя в первом подграфе, а другая — во втором подграфе. Данная схема работает до тех пор, пока на каждом процессе не останется по одному подграфу. Начиная с этого момента, процессы переупорядочивают соответствующие им подграфы, используя последовательный алгоритм из METIS. Когда все процессы заканчивают обработку подграфов, происходит объединение найденных перестановок.

Другая широко используемая академическая разработка — библиотека Scotch [23]. В ней для переупорядочения разреженных матриц используется комбинация неполного многоуровневого метода вложенных сечений и модифицированного метода минимальной степени [24]. Так же как и METIS, пакет Scotch имеет параллельную версию — PT-Scotch [25]. Распараллеливание выполнено для систем с распределенной памятью с использованием методов, схожих с ParMETIS.

Обсудим вопрос о целях и перспективах разработки параллельных алгоритмов переупорядочения. Не вызывает сомнений тот факт, что для ряда приложений использование кластерных систем для переупорядочения разреженных матриц является целесообразным, а иногда и необходимым. При этом вопрос о возможности построения эффективных параллельных алгоритмов для современных систем с общей памятью тоже представляет интерес. Один из наиболее простых способов распараллеливания состоит в применении того же подхода, что и для систем с распределенной памятью. Благодаря тому, что в современных реализациях MPI работе процессов в системах с общей памятью уделяется значительное внимание, такой подход может привести к приемлемой масштабируемости. Тем не менее, одна из тенденций последнего десятилетия, состоящая в неуклонном увеличении числа ядер и оптимизации структуры подсистемы памяти, привела к усовершенствованию существующих и разработке новых технологий параллельного программирования (OpenMP, TBB, Cilk Plus и др.). В отличие от MPI, данные технологии ориентированы на системы с общей памятью и потенциально способны позволить достигнуть лучшей масштабируемости или времени работы параллельной программы, запущенной на одном узле кластера. В ряде случаев использование гибридного параллелизма, например комбинации MPI+OpenMP, позволяет добиться наилучших результатов.

Указанные выше соображения обусловливают интерес к разработке параллельных алгоритмов переупорядочения для систем с общей памятью. Последние работы по данному направлению посвящены

решению смежной задачи — разработкам параллельных алгоритмов для разделения графов, которые в настоящее время выполняются авторами METIS [26] и Scotch [27]. Они предлагают подходы к распараллеливанию на общей памяти отдельных этапов многоуровневого метода. Однако для задачи переупорядочения матрицы только распараллеливания отдельных этапов не достаточно, поскольку в большинстве случаев выполнение одного этапа многоуровневого метода — слишком "простая" с точки зрения вычислительной нагрузки задача. В связи с этим требуется разработка принципиально других параллельных алгоритмов. В настоящей статье предлагается использовать параллелизм на уровне подграфов, возникающих при упорядочении графа методом вложенных сечений.

4. Постановка задачи. Рассмотрим постановку задачи переупорядочения в следующем виде [1].

Пусть дана разреженная симметричная матрица размера $n \times n$. Построим граф матрицы $G = (V, E)$, в котором каждая вершина v_i $(i = 1, 2, \ldots, n)$ соответствует строке матрицы i, а ребро $(v_i, v_j) \in E$ соответствует внедиагональному ненулевому элементу матрицы $a_{i,j} \neq 0$ $(i,\ j = 1, 2, \ldots, n;\ i \neq j)$. Множество вершин, смежных с вершиной v, обозначим $\mathrm{Adj}(v)$.

Смоделируем процесс гауссова исключения. При выполнении исключения вершины v из графа G в граф добавляются ребра таким образом, чтобы множество смежных с v вершин стало кликой, сама вершина v удаляется из множества вершин вместе с инцидентными ей ребрами:

$$V = V \setminus v; \quad E = E\big\{(u, v) : u \in \mathrm{Adj}(v)\big\} \cup \big\{(u_1, u_2) : u_1, u_2 \in \mathrm{Adj}(v)\big\}.$$

Множество добавленных ребер соответствует новым ненулевым элементам, возникшим в процессе гауссова исключения. Пусть $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ — перестановка множества вершин v. Заполнение $F(\pi)$, порожденное перестановкой π, — это множество ребер, добавленных при последовательном исключении вершин $\pi_1, \pi_2, \ldots, \pi_n$. Рассмотрим задачу, состоящую в нахождении перестановки π^*, минимизирующей число добавленных ребер в заполнении $F(\pi^*)$: $\pi^* = \arg\min\Big\{\big|F(\pi)\big|\Big\}$.

Показано, что данная задача является NP-полной [1], что обусловливает целесообразность разработки эвристических алгоритмов для поиска псевдооптимального решения. В практическом смысле при создании таких алгоритмов приходится искать компромисс между качеством решения задачи (число ненулевых элементов в факторе матрицы) и временны́ми затратами для нахождения перестановки.

5. Многоуровневый метод вложенных сечений в MORSy. В основу параллельного алгоритма переупорядочения симметричных разреженных матриц легла библиотека MORSy [4]. Библиотека написана на языке C, работает под операционными системами Windows и Linux. MORSy доступен для скачивания с сайта Приволжского научно-образовательного центра суперкомпьютерных технологий [28].

Переупорядочение разреженной матрицы в MORSy выполняется с помощью многоуровневого метода вложенных сечений (см. алгоритм 1). Нахождение разделителя в этом случае выполняется в три этапа: огрубление, разделение, развертывание.

- *Огрубление графа (coarsening)*: построение последовательности сжимающихся графов $G_1(V_1, E_1)$, $G_2(V_2, E_2)$, …, $G_m(V_m, E_m)$ на основе паросочетания вершин. При этом число вершин каждого следующего графа меньше, чем у предыдущего: $|V_1| > |V_2| > \ldots > |V_m|$; структура каждого следующего в последовательности графа отражает структуру предыдущего.

- *Разделение графа (partitioning)*: поиск разделителя и выделение двух несвязных компонент наименьшего графа G_m.

- *Развертывание графа (uncoarsening)*: проекция разделения наименьшего графа G_m на исходный граф G через последовательность графов $G_m, G_{m-1}, \ldots, G_1$. На каждом шаге i выполняется улучшение разделения графа G_{m-i}, полученного проекцией разделения графа G_{m-i+1}. Для этого обычно используются модификации итерационного метода Кернигана–Лина [29] и Фидуччиа–Мэтьюза [30].

Перед началом работы многоуровневого метода выполняется точное сжатие структуры графа [16] для ускорения работы переупорядочения (строка 18 алгоритма 1). В алгоритме 1 многоуровневый метод вынесен в отдельную функцию NDStep(). На этапе сжатия графа (строки 5–7 алгоритма 1) используются методы случайных паросочетаний или паросочетаний тяжелых ребер [15]. На этапе разделения (строка 8 алгоритма 1) выполняется построение структуры уровней смежности графа [3]. На этапе улучшения разделения (строки 9–12 алгоритма 1) используется модифицированный метод *примитивных перемещений* (Primitive Moves) на основе работ Эшкрафта–Лю [31] и Хендриксона–Ротберга [16].

Алгоритм примитивных перемещений выполняет дискретную оптимизацию разделения графа на три части (разделитель и подграфы, образующиеся после его удаления), опираясь на эвристическую оценку

качества разделения. Алгоритм состоит из двух вложенных циклов. На каждой итерации внешнего цикла фиксируется разделение графа P на разделитель S и две несвязные части V_1 и V_2, образующиеся после его удаления. Затем, во внутреннем цикле, выполняются попытки улучшения этого разделения за счет последовательного перемещения вершин из разделителя S в одну из частей V_1 и V_2. При этом если после перемещения разделение P улучшилось, то оно запоминается в лучшее найденное разделение P^*. После выполнения серии перемещений внешний цикл алгоритма повторяется для лучшего найденного разделения P^*, иначе процесс останавливается. Суть модификации заключается в изменении условий выполнения внутренней итерации алгоритма (перемещения вершины из разделителя в одну из частей V_1 или V_2). В отличие от базовой версии авторов алгоритма, на одной итерации внешнего цикла алгоритма выполняется перемещение только в одну, меньшую на начало итерации, часть графа. Для оценки качества разделения используется функция из [32].

Алгоритм 1. Многоуровневый метод вложенных сечений в MORSy

Вход: Граф $G_A(V_A, E_A)$, построенный по симметричной разреженной матрице A. V_A — множество вершин графа, E_A — множество ребер графа. Параметр m определяет максимальное число шагов сжатия

Выход: $Iperm$ — новая нумерация вершин G_A (строк A)

function NDStep (граф $G_0(V_0, E_0)$, параметр m) \\ Шаг многоуровневого метода
  if $|V_0|$ достаточно мал **then**
    переупорядочить граф методом автоматических вложенных сечений; порядок вершин сохранить в S_0
  else
    for $i = 0$ **to** m **do**
      огрубить $G_i(V_i, E_i)$; получим $G_{i+1}(V_{i+1}, E_{i+1})$
    end for
    Найти разделение $P_m(S_m, V_{m,1}, V_{m,2})$ графа G_m, где S_m — вершинный разделитель, $V_{m,1}$, $V_{m,2}$ — множества вершин, образующие несвязные части
    for $i = m$ **downto** 1 **do**
      спроецировать разделение $P_i(S_i, V_{i,1}, V_{i,2})$ графа G_i на разделение $P_{i-1}(S_{i-1}, V_{i-1,1}, V_{i-1,2})$ графа G_{i-1};
      улучшить разделение $P_{i-1}(S_{i-1}, V_{i-1,1}, V_{i-1,2})$
    end for
    Найти подграфы $G_1^*, G_2^*, \ldots, G_k^*$, полученные после удаления разделителя S_0 из G_0
  end if
  return $\langle S_0, {G_1}^*, {G_2}^*, \ldots, {G_k}^* \rangle$
end function

function MORSyOrdering (граф $G_0(V_0, E_0)$, параметр m) \\ Основной алгоритм
  Сжать граф G_A. Получим граф $G(V, E)$
  Создать пустую очередь обрабатываемых графов Q. Добавить в нее G
  while очередь Q не пуста **do**
    Извлечь из очереди первый необработанный граф $G_0(V_0, E_0)$
    $\langle S_0, {G_1}^*, {G_2}^*, \ldots, {G_k}^* \rangle = \text{NDStep}\big(G_0(V_0, E_0), m\big)$
    Положить $\langle {G_1}^*, {G_2}^*, \ldots, {G_k}^* \rangle$ в очередь Q
    Сохранить S_0 в общий массив разделителей S
  end while
  Занумеровать вершины графа G согласно порядку разделителей в S, начиная с больших индексов
  Сформировать $Iperm$, спроецировав номера вершин графа G на номера вершин графа G_A
  return $Iperm$
end function

Подробное описание реализации многоуровневого метода вложенных сечений в библиотеке MORSy

приведено в работе [4].

6. Параллельный алгоритм переупорядочения для систем с общей памятью. Параллельный алгоритм переупорядочения в MORSy основан на распараллеливании по задачам. Одной логической задачей, которую можно выполнять независимо, является вычисление для графа разделителя и образующихся после его удаления частей. Это соответствует вызову потоком функции NDStep() алгоритма 1. Входные данные такой задачи — граф и параметры алгоритма, выходные данные — найденный разделитель и множество подграфов, образующихся после удаления разделителя. Указатели на входные и выходные данные задачи объединяются в логическую задачу, которая передается потоку. После того как разделитель графа найден, поток находит подграфы, образующиеся после исключения разделителя, и формирует новые логические задачи, которые можно выполнять независимо. Для реализации алгоритма использовался механизм логических задач OpenMP 3.0 (OpenMP task). Алгоритм переупорядочения был преобразован из итеративной формы в рекурсивную, в которой вызов функции NDStep() сопровождается директивой запуска логической задачи, при этом передача задач потокам обеспечивается автоматическим планировщиком OpenMP (алгоритм 2).

Алгоритм 2. Параллельный многоуровневый метод вложенных сечений в MORSy

Вход: Граф $G_A(V_A, E_A)$, построенный по симметричной разреженной матрице A. V_A — множество вершин графа, E_A — множество ребер графа. Параметр m определяет максимальное число шагов сжатия

Выход: $Iperm$ — новая нумерация вершин G_A (строк A)

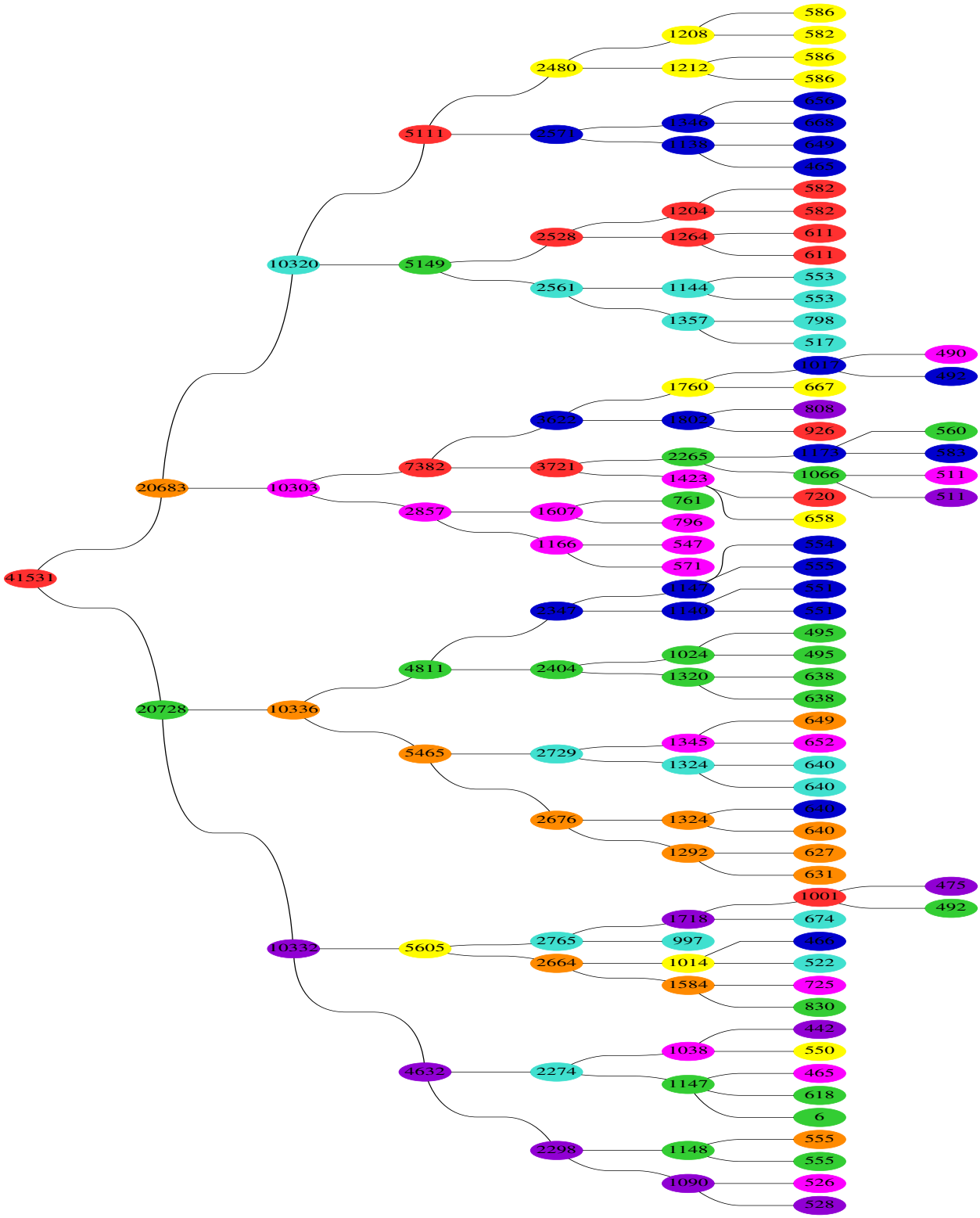
```
1:  function NDStepParallel (граф G_0(V_0, E_0), параметр m) \\ Шаг многоуровневого
                                                              \\ метода
2:    ⟨S_0, G_1*, G_2*, ..., G_k*⟩ = NDStep(G_0(V_0, E_0), m)
3:    for i = 0 to k do
4:      # pragma omp task
5:        NDStepParallel (G_i*(V_i*, E_i*), m)
6:    end for
7:  end function

8:  function MORSyParallelOrdering (граф G_0(V_0, E_0), параметр m) \\ Основной алгоритм
9:    Сжать граф G_A. Получим граф G(V, E)
10:   # pragma omp parallel
11:     # pragma omp single
12:       NDStepParallel(G(V, E), m)
13:   end omp parallel
14:   Сформировать массив разделителей S
15:   Занумеровать вершины графа G согласно порядку разделителей в S, начиная с больших индексов
16:   Сформировать Iperm, спроецировав номера вершин графа G на номера вершин графа G_A
17:   return Iperm
18: end function
```

Работа параллельного алгоритма проиллюстрирована на рисунке. На нем показано дерево задач, образовавшихся при переупорядочении матрицы pwtk (коллекция университета Флориды) восемью потоками.

7. Результаты вычислительных экспериментов.

7.1. Тестовые задачи. Для текущей программной реализации был проведен ряд вычислительных экспериментов по переупорядочению матриц из широко распространенной коллекции университета Флориды [33]. Эксперименты проводились на одном вычислительном узле со следующими характеристиками: два четырехъядерных процессора Intel Xeon L5630 (2.13 GHz), 24 GB памяти, операционная система Windows Server 2008 HPC Edition. Использовался компилятор Intel C++ Compiler, а также библиотека Intel MKL из пакета Intelo Parallel Studio XE 2013 SP1. Параметры тестовых матриц приведены в табл. 1.

Работа библиотеки MORSy сравнивалась с работой MPI-библиотеки ParMETIS v.4.0.3. Параметры



Работа параллельного алгоритма на матрице pwtk (коллекция университета Флориды) при запуске в 8 потоков. Узлом дерева обозначены вычислительные задачи, ребрами — зависимость между ними. Узлы-потомки соответствуют задачам, порожденным после выполнения узла-родителя. Узлы с одинаковым цветом обрабатывались одним потоком. Число в узле дерева означает количество вершин графа, для которого выполнялся поиск разделителя. Задачи для графов с числом вершин менее 1000 считались не порождающими других задач

переупорядочения в MORSy позволяют настроить работу алгоритма с приоритетом на минимизацию времени работы или на максимизацию качества в смысле числа ненулевых элементов фактора. Параметры переупорядочения подбирались для каждой матрицы. Для оценки эффективности реализации в смысле качества перестановок и времени работы в параллельном режиме в сравнении с ParMETIS были изучены две конфигурации параметров:

1) "лучшее качество" — параметры MORSy, дающие лучшее качество переупорядочения за приемлемое время; эта конфигурация показывает, какое качество перестановок в принципе является достижимым при использовании реализованных в MORSy алгоритмов;

2) "качество ParMETIS" — параметры MORSy, дающие качество переупорядочения, близкое к ParMETIS; эта конфигурация показывает, за какое время MORSy достигает качества перестановок ParMETIS, работающего в 8 процессов.

Таблица 1

Характеристики тестовых матриц. N — число строк матрицы, NZ — число ненулевых элементов матрицы

Матрица	N	NZ	Заполнение, %
pwtk	217 918	5 926 171	1.25E-04
msdoor	415 863	10 328 399	5.97E-05
parabolic_fem	525 825	2 100 225	7.60E-06
tmt_sym	726 713	2 903 835	5.50E-06
boneS10	914 898	28 191 660	3.37E-05
Emilia_923	923 136	20 964 171	2.46E-05
audkiw_1	943 695	39 297 171	4.41E-05
bone010	986 703	36 326 514	3.73E-05
ecology2	999 999	2 997 995	3.00E-06
thermal2	1 228 045	4 904 179	3.25E-06
StocF-1465	1 465 137	11 235 263	5.23E-06
Hook_1498	1 498 023	31 207 734	1.39E-05
Flan_1565	1 564 794	59 485 419	2.43E-05
G3_circuit	1 585 478	4 623 152	1.84E-06

7.2. Работа ParMETIS. Библиотека ParMETIS была собрана под ОС Windows с использованием CMake. При запусках ParMETIS матрица распределялась поровну между процессами, использовались настройки качества по умолчанию. В соответствии с руководством пользователя допустимо произвольное начальное разделение матрицы по процессам, поскольку матрица будет далее перераспределена согласно алгоритму k-разделения [34]. Результаты работы ParMETIS на тестовых матрицах приведены в табл. 2. Ускорение при работе в 8 процессов составило от 1.38 до 4.25 раза в зависимости от матрицы.

7.3. Сравнение ParMETIS и MORSy в конфигурации "Лучшее качество". Сравним время и качество работы MORSy с максимальными настройками качества и ParMETIS. В отличие от ParMETIS, качество переупорядочения параллельного MORSy не зависит от числа потоков, в которое было запущено приложение. Результаты работы MORSy относительно ParMETIS приведены в табл. 3, где NZ_MORSy ($NZ\_ParMETIS$) — число ненулевых элементов фактора матрицы после применения перестановки MORSy (ParMETIS) соответственно. T_MORSy — время работы MORSy, $T\_ParMETIS$ — время работы ParMETIS (в соответствующее число процессов).

В сравнении с ParMETIS, запущенным в один MPI-процесс, параллельный MORSy с настройками на лучшее качество дает лучшие перестановки на 5 матрицах из 14. На матрице ecology2 опережение составляет 17%, на других матрицах — 1–5%. На 6 из 14 матриц MORSy дает перестановки с размером фактора на 1–3% хуже, чем у ParMETIS. На 3 оставшихся матрицах отставание в размерах фактора составляет 5–9%.

В сравнении с ParMETIS, запущенным в 8 MPI-процессов, параллельный MORSy с настройками на лучшее качество дает лучшие перестановки на 11 матрицах из 14. Выигрыш в размерах фактора составляет 1–14% (в среднем 5%). На 2 из оставшихся 3 матриц отставание в размерах фактора составляет не более 2%, на матрице boneS10 — 9%.

Сравнение времени работы библиотек показывает, что MORSy с лучшими настройками качества на 8 матрицах из 14 работает быстрее, чем ParMETIS. Среднее опережение на этих матрицах составляет в среднем 2.2 раза при работе в один поток, в 3 раза — при работе в 8 потоков. На оставшихся 6 матрицах MORSy отстает по времени в среднем в 1.8 раз при работе в один поток, в 2.3 раза — при работе в 8 потоков. При этом для большинства матриц отставание или опережение относительно ParMETIS растет с увеличением числа потоков.

Таким образом, при лучших настройках качества MORSy дает перестановки, лучшие или сходные по качеству в сравнении с ParMETIS для большинства тестовых матриц. При этом по времени работы переупорядочения в параллельном режиме, позволяющего получить лучшую по качеству перестановку,

Таблица 2

Результаты работы ParMETIS. NZ — число ненулевых элементов фактора матрицы после применения перестановки. Время приведено в секундах

Матрица	NZ		Время работы, с	
	1 процесс	8 процессов	1 процесс	8 процессов
pwtk	47 518 408	48 907 025	1.74	1.14
msdoor	51 882 257	52 585 355	2.84	2.06
parabolic_fem	25 453 846	25 605 949	5.60	1.63
tmt_sym	28 657 617	30 052 013	7.60	1.87
boneS10	266 173 272	267 410 700	14.48	5.71
Emilia_923	1 633 654 176	1 808 078 468	13.72	6.12
audikw_1	1 225 571 121	1 327 766 831	21.96	13.72
bone010	1 076 191 560	1 133 242 535	18.58	8.55
ecology2	35 606 934	34 206 711	8.41	1.98
thermal2	50 293 930	50 465 040	14.87	3.61
StocF-1465	1 039 392 123	1 061 011 377	28.23	6.79
Hook_1498	1 507 528 290	1 615 437 773	23.43	8.92
Flan_1565	1 451 334 747	1 523 366 946	29.91	11.32
G3_circuit	90 397 858	94 800 689	17.01	4.38

Таблица 3

Результаты работы MORSy с лучшими настройками качества относительно результатов работы ParMETIS (табл. 2)

Матрица	NZ_MORSy / $NZ\_ParMETIS$		T_MORSy / $T\_ParMETIS$	
	1 поток	8 потоков	1 поток	8 потоков
pwtk	0.98	0.95	0.26	0.17
msdoor	1.00	0.99	0.40	0.21
parabolic_fem	0.98	0.97	1.31	1.50
tmt_sym	1.00	0.96	1.77	2.63
boneS10	1.09	1.09	0.44	0.44
Emilia_923	1.01	0.91	0.47	0.42
audikw_1	1.08	0.99	0.47	0.30
bone010	0.95	0.90	0.67	0.62
ecology2	0.83	0.86	2.03	2.60
thermal2	1.02	1.02	2.18	3.17
StocF-1465	1.03	1.01	1.95	2.42
Hook_1498	1.05	0.98	0.55	0.58
Flan_1565	0.98	0.93	0.58	0.50
G3_circuit	0.99	0.95	1.37	1.79

Таблица 4

Время работы MORSy относительно ParMETIS при близком качестве работы библиотек

Матрица	T_MORSy / $T\_ParMETIS$	
	1 поток	8 потоков
pwtk	0.25	0,16
msdoor	0.32	0.25
parabolic_fem	0.85	1.06
tmt_sym	0.95	1.36
boneS10	0.45	0.42
Emilia_923	0.33	0.33
audikw_1	0.43	0.27
bone010	0.33	0.28
ecology2	0.95	1.37
thermal2	1.72	2.43
StocF-1465	1.09	1.60
Hook_1498	0.45	0.46
Flan_1565	0.33	0.34
G3_circuit	0.83	1.13

MORSy опережает ParMETIS на 8 матрицах из 14.

7.4. Сравнение ParMETIS и MORSy в конфигурации "Качество ParMETIS". Неменьший интерес представляет конфигурация, в которой параллельный MORSy позволяет достигать того же качества, что и ParMETIS, работающий в 8 процессов. Для большинства матриц в этом случае время работы MORSy сокращается, так как разрешается получить большее число ненулевых элементов в факторе, чем при лучших настройках качества. Исключение составляют матрицы boneS10, thermal2, StocF-1465, на которых MORSy при максимальных настройках качества отстает от ParMETIS более чем на 5%. На остальных матрицах разница в размерах фактора составляет не более 1%. Время работы MORSy в данной конфигурации показано в табл. 4, где T_MORSy — время работы MORSy и $T\_ParMETIS$ — время работы ParMETIS, запущенного в соответствующее число процессов (табл. 2).

В этом случае при работе в один поток MORSy работает быстрее, чем ParMETIS, запущенный в один процесс, на 12 матрицах из 14. Среднее опережение составляет 2.3 раза. Отставание на оставшихся матрицах составляет 1.1 и 1.7 раз. При работе в 8 потоков MORSy работает быстрее, чем ParMETIS, запущенный в 8 процессов, на 8 матрицах из 14. Среднее опережение на этих матрицах составляет 3.5 раза. На 5 из оставшихся 6 матриц отставание не превосходит 1.6 раза, в среднем — 1.3 раза. Исключение составляет матрица thermal2, на которой MORSy работает медленнее, чем ParMETIS, в 2.4 раза.

7.5. Анализ производительности MORSy. Анализ профиля работы MORSy проводился с использованием Intel Amplifier XE. Для библиотеки наиболее затратным по времени является этап улучшения разделения, который занимает 35–55% времени всего переупорядочения при максимальных настройках качества решения задачи. Время работы этого этапа зависит от числа итераций алгоритма примитивных перемещений и определяет качество получаемых перестановок. Внесение дополнительных ограничений на число итераций метода позволило сократить время работы этого этапа до 2 раз. Второй по временны́м затратам — этап огрубления графа, занимающий 20–30% времени переупорядочения. Время работы этого этапа зависит от количества случайных чисел, используемых при построении паросочетания. Алгоритм паросочетания тяжелых ребер работает быстрее, чем алгоритм случайных паросочетаний. Кроме того, анализ профиля показывает, что существенный вклад в общее время работы вносят простои конвейера, причинами которых являются типичные для алгоритмов на графах факторы, а именно: значительный процент неправильно предсказанных ветвлений и нерегулярный доступ к памяти, приводящий к промахам в кэш последнего уровня и буфер ассоциативной трансляции.

Одним из главных недостатков текущей параллельной версии является то, что задача нахождения первого разделителя решается последовательно. При запусках в один поток время ее решения составляет около 10–15%, при запуске в 8 потоков — 20–25% общего времени решения. Распараллеливание нахождения первого разделителя является одним из направлений дальнейшей разработки библиотеки.

Результаты экспериментов показывают, что, в сравнении с ParMETIS, MORSy лучше работает на матрицах, допускающих сжатие структуры на этапе предобработки, а также имеющих большие степени вершин после сжатия (от 10 до 26 на рассматриваемых матрицах). На матрицах, не допускающих сжатие или имеющих небольшие степени вершин (2–4 на рассматриваемых матрицах), ParMETIS работает быстрее. В дальнейшем используемые алгоритмы будут доработаны и адаптированы для более эффективной работы на таких матрицах.

8. Заключение. В настоящей статье предлагается параллельная редакция многоуровневого метода вложенных сечений для систем с общей памятью. Программная реализация метода основана на представленной ранее последовательной библиотеке MORSy [4] с рядом модификаций, направленных на улучшение качества перестановок и скорости работы. В отличие от аналогов при распараллеливании используется традиционная для многоядерных систем технология распараллеливания OpenMP. По своей природе метод вложенных сечений относится к классу "разделяй и властвуй", что позволяет использовать механизм логических задач OpenMP для организации параллельной обработки графа матрицы. В качестве задачи рассматривается выполнение итерации метода вложенных сечений (нахождение разделителя в графе). Балансировка нагрузки выполняется динамически планировщиком OpenMP.

Результаты экспериментов на матрицах из коллекции университета Флориды показывают сравнимость выполненной реализации с широко используемой библиотекой ParMETIS. На большинстве тестовых матриц MORSy с максимальными настройками качества позволяет получить лучшие, чем ParMETIS, перестановки или незначительно уступающие по качеству. При этом на половине тестовых матриц время его работы тоже лучше. При близких к ParMETIS показателях качества перестановок MORSy работает быстрее на большинстве тестовых матриц.

Дополнительные эксперименты по использованию перестановок, полученных MORSy для решения разреженных СЛАУ с помощью Intel MKL PARDISO, показали, что решатель СЛАУ сохраняет естествен-

ное поведение: время его работы уменьшается на перестановках с меньшим числом ненулевых элементов, увеличивается — на перестановках с бóльшим числом ненулевых элементов в сравнении с работой на перестановках, полученных ParMETIS. Кроме того, параллельный решатель имеет близкое ускорение от распараллеливания при использовании перестановок ParMETIS и MORSy. Данные результаты говорят о применимости перестановок MORSy для последующего параллельного прямого решения разреженных СЛАУ.

Программная реализация выполнена в виде библиотеки с открытым исходным кодом и доступна для скачивания на сайте Приволжского научно-образовательного центра суперкомпьютерных технологий [28]. Разработанное программное обеспечение используется в многопоточном прямом решателе разреженных СЛАУ, разрабатываемом в ННГУ [35, 36]. Основное направление дальнейших исследований — улучшение времени работы и масштабируемости параллельного MORSy. Для этого, в частности, планируется выполнить распараллеливание отдельных этапов многоуровневого метода для графов с большим числом вершин. В результате предполагается построить двухуровневую схему распараллеливания, сочетающую параллелизм внутри этапов метода (для больших графов) и параллелизм по задачам (для средних и малых графов). Кроме того, будет разработана редакция алгоритма для систем с распределенной памятью.

Работа выполнена при частичной поддержке гранта РФФИ № 14–01–31455 и гранта МОН РФ (соглашение от 27 августа 2013 г. № 02.B.49.21.0003 между МОН РФ и ННГУ).

Статья рекомендована к публикации Программным комитетом Научно-технического семинара “Технологии параллельной обработки больших графов” (GraphHPC-2015, http://www.dislab.org/GraphHPC-2015).

список литературы

1. *Yannakakis M.* Computing the minimum fill-in is NP-complete // SIAM J. on Algebraic and Discrete Methods. 1981. **2**, N 1. 77–79.
2. *Tinney W.F., Walker J.W.* Direct solutions of sparse network equations by optimally ordered triangular factorization // Proceedings of the IEEE. 1967. **55**, N 11. 1801–1809.
3. *George A.* Nested dissection of a regular finite element mesh // SIAM J. on Numerical Analysis. 1973. **10**, N 2. 345–363.
4. *Pirova A., Meyerov I.* MORSy — a new tool for sparse matrix reordering // Proc. Int. Conf. on Engineering and Applied Sciences Optimization. Kos Island, Greece, 4–6 June 2014. Athens: National Tech. Univ. of Athens, 2014. 1952–1964.
5. *Liu J.W.H.* Modification of the minimum-degree algorithm by multiple elimination // ACM Trans. Math. Software. 1985. **11**, N 2. 141–153.
6. *Amestoy P.R., Davis T.A., Duff I.S.* An approximate minimum degree ordering algorithm // SIAM. J. on Matrix Anal. Appl. 1996. **17**, N 4. 886–905.
7. *Davis T.A., Gilbert J.R., Larimore S.I., Ng E.G.* A column approximate minimum degree ordering algorithm // ACM Transactions on Mathematical Software. 2004. **30**, N 3. 353–376.
8. *Lipton R.J., Rose D.J., Tarjan R.E.* Generalized nested dissection // SIAM J. on Numerical Analysis. 1979. **16**, N 2. 346–358.
9. *George A., Liu J.W.H.* An Automatic Nested Dissection Algorithm for Irregular Finite Element Problems // SIAM J. on Numerical Analysis. 1978. **15**, N 5. 1053–1069.
10. *Pothen A., Simon H.D., Wang L.* Spectral nested dissection. Technical Report CS-92-01. State College: Pennsylvania State Univ., 1992.
11. *Heath M.T., Raghavan P.* A Cartesian parallel nested dissection algorithm // SIAM J. on Matrix Analysis and Applications. 1995. **16**, N 1. 235–253.
12. *Miller G.L., Teng S.-H., Vavasis S.A.* A unified geometric approach to graph separators // Proc. of 32nd Annual Symposium on Foundations of Computer Science. New York: IEEE Press, 1991. 538–547.
13. *Bui T.N., Jones C.* A heuristic for reducing fill-in in sparse matrix factorization // Proc. of the 6th SIAM Conf. on Parallel Processing for Scientific Computing. Philadelphia: SIAM Press, 1993. 445–452.
14. *Hendrickson B., Leland R.* A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301. Albuquerque: Sandia National Laboratories, 1993.
15. *Karypis G., Kumar V.* A fast and high quality multilevel scheme for partitioning irregular graphs // SIAM J. on Scientific Computing. 1998. **20**, N 1. 359–392.
16. *Hendrickson B., Rothberg E.* Improving the runtime and quality of nested dissection ordering // SIAM J. on Scientific Computing. 1999. **20**, N 2. 468–489.
17. *George A. et al.* Sparse Cholesky factorization on a local-memory multiprocessor // SIAM J. on Scientific and Statistical Computing. 1988. **9**, N 2. 327–340.
18. *Karypis G.* METIS: a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Minneapolis: University of Minnesota, 2011.

19. Intel Math Kernel Library Reference Manual (http://software.intel.com/sites/default/files/managed/9d/c8/mklman.pdf).
20. *Karypis G., Kumar V.* ParMETIS: parallel graph partitioning and sparse matrix ordering library. Technical Report TR 97-060. Minneapolis: University of Minnesota, 1997.
21. *Karypis G., Kumar V.* A parallel algorithm for multilevel graph partitioning and sparse matrix ordering // J. of Parallel and Distributed Computing. 1998. **48**, N 1. 71–95.
22. *Schloegel K., Karypis G., Kumar V.* Parallel multilevel algorithms for multi-constraint graph partitioning // Lecture Notes in Computer Science. Vol. 1900. London: Springer, 2000. 296–310.
23. *Pellegrini F.* Scotch and libScotch 6.0 User's Guide. Technical Report LaBRI. Bordeaux: Université Bordeaux, 2012.
24. *Pellegrini F., Roman J., Amestoy P.* Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering // Concurrency: Practice and Experience. 2000. **12**, N 2–3, 68–84.
25. *Chevalier C., Pellegrini F.* PT-Scotch: a tool for efficient parallel graph ordering // Parallel Computing. 2008. **34**, N 6–8. 318–331.
26. *Lasalle D., Karypis G.* Multi-threaded graph partitioning // Proc. 27th Int. Symp. on Parallel and Distributed Processing. 2013. New York: IEEE Press, 225–236.
27. *Pellegrini F.* Shared memory parallel algorithms In Scotch 6 (http://graal.ens-lyon.fr/mumps/doc/ud_2013/pellegrini.pdf).
28. MORSy (http://hpc-education.unn.ru/research/overview/sparse-algebra/morsy).
29. *Kernighan B.W., Lin S.* An efficient heuristic procedure for partitioning graphs // The Bell System Technical Journal. 1970. **49**, N 2. 291–307.
30. *Fiduccia C.M., Mattheyses R.M.* A linear-time heuristic for improving network partitions // Proc. 19th IEEE Design Automation Conference. Piscataway: IEEE Press, 1982. 175–181.
31. *Ashcraft C., Liu J.W.H.* A partition improvement algorithm for generalized nested dissection. Technical Report BCSTECH-92-020. Seattle: Boeing Computer Services, 1994.
32. *Ashcraft C., Liu J.W.H.* Using domain decomposition to find graph bisectors // BIT Numerical Mathematics. 1997. **37**, N 3. 506–534.
33. *Davis T.A., Hu Y.* The University of Florida sparse matrix collection // ACM Transactions on Mathematical Software. 2011. **38**. doi 10.1145/2049662.2049663.
34. *Karypis G., Schloegel K.* ParMETIS Manual. Parallel Graph Partitioning and Sparse Matrix Ordering Library. Version 4.0. http://glaros.dtc.umn.edu/gkhome/fetch/sw/parmetis/manual.pdf. Cited August 11, 2015.
35. *Lebedev S., Akhmedzhanov D., Kozinov E., Meyerov I., Pirova A., Sysoyev A.* Dynamic parallelization strategies for multifrontal sparse Cholesky factorization // PACT-2015. Lecture Notes in Computer Science. 2015 (в печати).
36. *Козинов Е.А., Лебедев И.Г., Лебедев С.А., Мееров И.Б., Малова А.Ю., Сысоев А.В., Филиппенко С.С.* Новый решатель для алгебраических систем разреженных линейных уравнений с симметричной положительно определенной матрицей // Вестник Нижегородского университета им. Н.И. Лобачевского. 2012. № 5–2. 376–384.

Поступила в редакцию
29.05.2015

A Parallel Multilevel Nested Dissection Algorithm for Shared-Memory Computing Systems

A. Yu. Pirova [1], I. B. Meyerov [2], E. A. Kozinov [3], and S. A. Lebedev [4]

[1] *Lobachevsky State University of Nizhni Novgorod, Faculty of Computational Mathematics and Cybernetics; prospekt Gagarina 23, Nizhni Novgorod, 603950, Russia; Junior Scientist, e-mail: anna.yu.malova@gmail.com*

[2] *Lobachevsky State University of Nizhni Novgorod, Faculty of Computational Mathematics and Cybernetics; prospekt Gagarina 23, Nizhni Novgorod, 603950, Russia; Ph.D., Associate Professor, e-mail: meerov@vmk.unn.ru*

[3] *Lobachevsky State University of Nizhni Novgorod, Faculty of Computational Mathematics and Cybernetics; prospekt Gagarina 23, Nizhni Novgorod, 603950, Russia; Assistant, e-mail: evgeniy.kozinov@gmail.com*

[4] *Lobachevsky State University of Nizhni Novgorod, Faculty of Computational Mathematics and Cybernetics; prospekt Gagarina 23, Nizhni Novgorod, 603950, Russia; Programmer, e-mail: sergey.a.lebedev@gmail.com*

Received May 29, 2015

Abstract: This paper deals with the NP-complete problem of finding a symmetric positive definite sparse matrix ordering that minimizes the Cholesky factor fill-in. For this purpose, heuristic approaches based on graph

algorithms are applied. A new parallel ordering algorithm for shared-memory computing systems is proposed. The modified multilevel nested dissection algorithm from the recently presented MORSy library is used as a basis for ordering. The parallel processing is done in a task-based fashion. It uses the OpenMP 3.0 task parallelism relying on the dynamic load balancing implemented during the OpenMP runtime. The numerical experiments were performed using a number of symmetric positive definite matrices from the University of Florida Sparse Matrix Collection. The experimental results show the competitiveness of the proposed implementation on shared memory systems compared to the widely used ParMETIS library. In our experiments, the parallel MORSy version provides a better ordering than ParMETIS on all but one matrix in terms of the Cholesky factor fill-in and outperforms ParMETIS in most cases. The parallel MORSy version is publicly available from the Supercomputing Center of Lobachevsky State University of Nizhni Novgorod.

Keywords: multilevel nested dissection, sparse matrix ordering, Cholesky factorization, parallel algorithms, shared-memory computing systems, high-performance computing.

References

1. M. Yannakakis, "Computing the Minimum Fill-In is NP-Complete," SIAM J. Alg. Disc. Meth. **2** (1), 77–79 (1981).

2. W. F. Tinney and J. W. Walker, "Direct Solutions of Sparse Network Equations by Optimally Ordered Triangular Factorization," Proc. IEEE **55** (11), 1801–1809 (1967).

3. A. George, "Nested Dissection of a Regular Finite Element Mesh," SIAM J. Numer. Anal. **10** (2), 345–363 (1973).

4. A. Pirova and I. Meyerov, "MORSy — A New Tool For Sparse Matrix Reordering," in *Proc. Int. Conf. on Engineering and Applied Sciences Optimization, Kos Island, Greece, June 4–6, 2014* (National Tech. Univ. of Athens, Athens, 2014), pp. 1952–1964.

5. J. W. H. Liu, "Modification of the Minimum-Degree Algorithm by Multiple Elimination," ACM Trans. Math. Softw. **11** (2), 141–153 (1985).

6. P. R. Amestoy, T. A. Davis, and I. S. Duff, "An Approximate Minimum Degree Ordering Algorithm," SIAM. J. Matrix Anal. Appl. **17** (4), 886–905 (1996).

7. T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, "A Column Approximate Minimum Degree Ordering Algorithm," ACM Trans. Math. Soft. **30** (3), 353–376 (2004).

8. R. J. Lipton, D. J. Rose, and R. E. Tarjan, "Generalized Nested Dissection," SIAM J. Numer. Anal. **16** (2), 346–358 (1979).

9. A. George and J. W. H. Liu, "An Automatic Nested Dissection Algorithm for Irregular Finite Element Problems," SIAM J. Numer. Anal. **15** (5), 1053–1069 (1978).

10. A. Pothen, H. D. Simon, and L. Wang, *Spectral Nested Dissection*, Technical Report CS-92-01 (Pennsylvania State Univ., State College, 1992).

11. M. T. Heath and P. Raghavan, "A Cartesian Parallel Nested Dissection Algorithm," SIAM J. Matrix Anal. Appl. **16** (1), 235–253 (1995).

12. G. L. Miller, S.-H. Teng, and S. A. Vavasis, "A Unified Geometric Approach to Graph Separators," in *Proc. 32nd Annual Symp. on Foundations of Computer Science, San Juan, USA, October 1–4, 1991* (IEEE Press, New York, 1991), pp. 538–547.

13. T. N. Bui and C. Jones, "A Heuristic for Reducing Fill-In in Sparse Matrix Factorization," in *Proc. 6th SIAM Conf. on Parallel Processing for Scientific Computing, Norfolk, USA, March 22–24, 1993* (SIAM Press, Philadelphia, 1993), pp. 445–452.

14. B. Hendrickson and R. Leland, *A Multilevel Algorithm for Partitioning Graphs*, Tech. Rep. SAND93-1301 (Sandia National Labs., Albuquerque, 1993).

15. G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," SIAM J. Sci. Comput. **20** (1), 359–392 (1998).

16. B. Hendrickson and E. Rothberg, "Improving the Runtime and Quality of Nested Dissection Ordering," SIAM J. Sci. Comput. **20** (2), 468–489 (1999).

17. A. George, M. T. Heath, J. Liu, and E. Ng, "Sparse Cholesky Factorization on a Local-Memory Multiprocessor ," SIAM J. Sci. Stat. Comput. **9** (2), 327–340 (1988).

18. G. Karipis, *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, Version 5.0 (University of Minnesota, Minneapolis, 2011).

19. Intel Math Kernel Library Reference Manual.
http://software.intel.com/sites/default/files/managed/9d/c8/mklman.pdf. Cited August 11, 2015.

20. G. Karypis and V. Kumar, *ParMetis: Parallel Graph Partitioning and Sparse Matrix Ordering Library*, Technical Report 97-060 (University of Minnesota, Minneapolis, 1997).

21. G. Karypis and V. Kumar, "A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering," J. Parallel Distrib. Comput. **48** (1), 71–95 (1998).

22. K. Schloegel, G. Karypis, and V. Kumar, "Parallel Multilevel Algorithms for Multi-Constraint Graph Partitioning," in *Lecture Notes in Computer Science* (Springer, London, 2000), Vol. 1900, pp. 296–310.

23. F. Pellegrini, *Scotch and LibScotch 6.0 User's Guide* (Université Bordeaux, Bordeaux, 2012).

24. F. Pellegrini, J. Roman, and P. Amestoy, "Hybridizing Nested Dissection and Halo Approximate Minimum Degree for Efficient Sparse Matrix Ordering," Concurrency: Pract. Exper. **12** (2–3), 68–84 (2000).

25. C. Chevalier and F. Pellegrini, "PT-Scotch: A Tool for Efficient Parallel Graph Ordering," Parallel Comput. **34** (6–8), 318–331 (2008).

26. D. Lasalle and G. Karypis, "Multi-Threaded Graph Partitioning," in *Proc. 27th Int. Symp. on Parallel and Distributed Processing, Boston, USA, May 20–24, 2013* (IEEE Press, New York, 2013), pp. 225–236.

27. F. Pellegrini, "Shared Memory Parallel Algorithms in Scotch 6,"
http://graal.ens-lyon.fr/MUMPS/doc/ud_2013/Pellegrini.pdf. Cited August 11, 2015.

28. MORSy. http://hpc-education.unn.ru/research/overview/sparse-algebra/morsy. Cited August 11, 2015.

29. B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," The Bell Syst. Tech. J. **49** (2), 291–307 (1970).

30. C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," in *Proc. 19th IEEE Design Automation Conf., Las Vegas, USA, June 14–16, 1982* (IEEE Press, Piscataway, 1982), pp. 175–181.

31. C. Aschcraft and J. W. H. Liu, *A Partition Improvement Algorithm for Generalized Nested Dissection*, Technical Report BCSTECH-94-020 (Boeing Computer Services, Seattle, 1994).

32. C. Aschcraft and J. W. H. Liu, "Using Domain Decomposition to Find Graph Bisectors," BIT Numer. Math. **37** (3), 506–534 (1997).

33. T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," ACM Trans. Math. Softw. **38** (2011). doi 10.1145/2049662.2049663

34. G. Karipis and K. Schloegel, *ParMETIS Manual. Parallel Graph Partitioning and Sparse Matrix Ordering Library*, Version 4.0. http://glaros.dtc.umn.edu/gkhome/fetch/sw/parmetis/manual.pdf. Cited August 11, 2015.

35. S. Lebedev, D. Akhmedzhanov, E. Kozinov, et al., "Dynamic Parallelization Strategies for Multifrontal Sparse Cholesky Factorization," Lecture Notes in Computer Science (in press).

36. E. A. Kozinov, I. G. Lebedev, S. A. Lebedev, et al., "Novel Linear Equations Systems Solver for Sparse Symmetric Positive-Definite Matrix," Vestn. Univ. Nizhni Novgorod **5** (2), 376–384 (2012).