# Программный комплекс DMORSy для переупорядочения разреженных матриц на кластерных системах

А.Ю. Пирова, И.Б. Мееров, Е.А. Козинов

Нижегородский государственный университет им. Н. И. Лобачевского

Перестановка строк и столбцов разреженных матриц с целью сокращения заполнения фактора является важным этапом в процессе прямого решения больших разреженных систем линейных алгебраических уравнений. Задача нахождения оптимальной перестановки является NP-полной, для ее решения на практике используются эвристические методы. В работе рассматривается схема распараллеливания многоуровневого метода вложенных сечений для систем с распределенной памятью. Приводятся первые результаты экспериментов. Дается сравнение с библиотеками ParMETIS и PT-Scotch на матрицах из коллекции SuiteSparse.

*Ключевые слова:* переупорядочение разреженных матриц, метод вложенных сечений, параллельные вычисления.

## 1. Введение

Прямые методы широко используются при решении разреженных систем линейных алгебраических уравнений (СЛАУ). В ходе прямого решения СЛАУ матрица системы раскладывается в произведение двух треугольных матриц, после чего дважды применяется процедура обратного хода метода Гаусса. Одной из основных проблем, возникающих при использовании прямых методов, является существенное (речь может идти о нескольких порядках) увеличение числа ненулевых элементов в процессе факторизации. Для преодоления этой проблемы применяется перестановка столбцов и строк исходной матрицы. Известно, что в случае построения подходящей перестановки для многих матриц удается сократить число ненулевых элементов фактора и, соответственно, уменьшить затраты памяти и времени.

Разработка методов построения таких перестановок ведется на протяжении нескольких десятков лет. Показано, что задача поиска оптимальной перестановки является NP-полной [1]. В связи с этим усилия исследователей сосредоточены в области создания различных эвристик, позволяющих находить приемлемое компромиссное решение в смысле двух ключевых критериев – времени на построение перестановки и числа ненулевых элементов в факторе. Наибольшее распространение получили схемы, основанные на методе минимальной степени, методе вложенных сечений и их многочисленных модификациях.

Распространение параллельных архитектур и систем, а также развитие технологий их использования привело к возникновению интереса к методам распараллеливания вычислений в задаче о построении перестановок разреженных матриц. Значительные результаты в данной области достигнуты группами Дж. Карписа и Ф. Пеллегрини при разработке параллельных программных пакетов с открытым исходным кодом ParMETIS [2, 3] и PT-Scotch [4]. Данные пакеты основаны на методе вложенных сечений и использовании технологии MPI для распараллеливания вычислений. Несколько лет назад авторами пакета ParMETIS была представлена библиотека Mt-METIS [5], предназначенная для параллельной работы в системах с общей памятью. Среди других важных результатов необходимо отметить реализацию методов переупорядочения разреженных матриц, разработанных корпорацией Интел в составе прямых решателей разреженных СЛАУ MKL PARDISO [6] (для систем с общей памятью) и Cluster PARDISO [7] (для систем с распределенной памятью). Данные реализации являются коммерческими и могут быть использованы только в составе решателя Интел.

В 2015 году авторами данной статьи была представлена библиотека с открытым исходным кодом PMORSy [8], реализующая модифицированную схему метода вложенных сечений, распараллеленную для систем с общей памятью. Было показано, что на большинстве рассмотренных матриц из коллекции университета Флориды [9] (в настоящее время коллекция SuiteSparse)

и коллекции матриц, сгенерированных пакетом «ЛОГОС» в ходе конечно-элементных расчетов задач прочности, реализация PMORSy показывает сопоставимые с аналогами результаты по времени работы и заполнению фактора, в некоторых случаях опережая их. В данной статье мы делаем следующий шаг в направлении расширения функциональности PMORSy. В работе представлено описание параллельного алгоритма для систем с распределенной памятью (DMORSy). Этот алгоритм основан на разработанной авторами ранее последовательной реализации многоуровневого метода вложенных сечений с модификациями ряда стадий и модели распараллеливания для систем с распределенной памятью, предложенной авторами PT-Scotch. Представлены первые результаты экспериментов, сравнивающие результаты с основными аналогами. Конечной целью является создание гибридной версии библиотеки, основанной на использовании связки технологий MPI + OpenMP для оптимального использования возможностей параллелизма современных многоуровневых вычислительных систем.

## 2. Параллельный алгоритм

Рассмотрим процесс переупорядочения разреженной матрицы методом вложенных сечений. Предварительно по матрице строится граф, вершины которого соответствуют строкам (столбцам) матрицы, а ребра – ее ненулевым элементам. На каждом шаге алгоритма в графе выделяется разделитель – множество вершин, удаление которых делит граф на два или более несвязных подграфа. Вершины разделителя нумеруются и удаляются из графа; процесс продолжается аналогично на образовавшихся подграфах. Переупорядочение завершается, когда все вершины графа занумерованы. Качество переупорядочения понимается в смысле числа ненулевых элементов в факторе матрицы и зависит от характера найденных разделителей: чем меньше вершин содержит разделитель и чем ближе по размеру образующиеся после его удаления подграфы, 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 $G_0$ строится последовательность графов $G_1$, $G_2$, …, $G_m$, в которой каждый следующий граф меньше предыдущего и представляет собой огрубление его структуры. На этом этапе используются алгоритмы построения максимального паросочетания, в которых для каждой вершины по некоторому правилу выбирается парная вершина. В PMORSy для этого используется метод паросочетания тяжелых ребер (HEM), паросочетания тяжелых клик (HCM) или метод случайных паросочетаний (RM). Огрубление выполняется, в зависимости от настроек алгоритма, определенное число раз, либо пока не будет получен граф достаточно малого размера. На этапе разделения выполняется вычисление разделителя и несвязных подграфов самого маленького в последовательности графа, $G_m$. На этапе развертывания выполняется проекция разделителя и несвязных частей графа $G_m$, на граф $G_0$ через всю последовательность графов $G_{m-1}$, $G_{m-2}$, …, $G_1$. При этом используются алгоритмы улучшения разделения, которые уменьшают разделитель и сокращают дисбаланс образовавшихся несвязных частей. Подробное описание последовательного алгоритма переупорядочения, реализованного в PMORSy, приведено в [10], а его многопоточной версии – в [8].

При разработке распределенной версии (DMORSy) принято предположение, что граф исходной матрицы хранится распределенно на нескольких процессах. На каждом процессе хранятся списки смежности относящихся к нему вершин (назовем их локальными вершинами), а также веса вершин и другая вспомогательная информация об этих вершинах и связанных с ними вершинах с других процессов. Библиотека DMORSy не выполняет перераспределение исходного графа по процессам. Предполагается, что изначально граф распределен по процессам так, что каждый их них хранит близкое число вершин.

В основу параллельного многоуровневого метода вложенных сечений для систем с распределенной памятью был взят алгоритм из [3]. Пусть граф хранится на P процессах. Тогда поиск первого разделителя выполняется параллельно P процессами. После того, как разделитель найден, в графе выделяется 2 несвязных подграфа, каждый из которых распределяется по P / 2 процессам (рис. 1). Для этих подграфов поиск разделителя будет выполняться P / 2 процессами. Процедура выполняется до тех пор, пока на каждом процессе не окажется по целому подграфу.

После этого переупорядочение продолжается последовательно и независимо на каждом процессе.

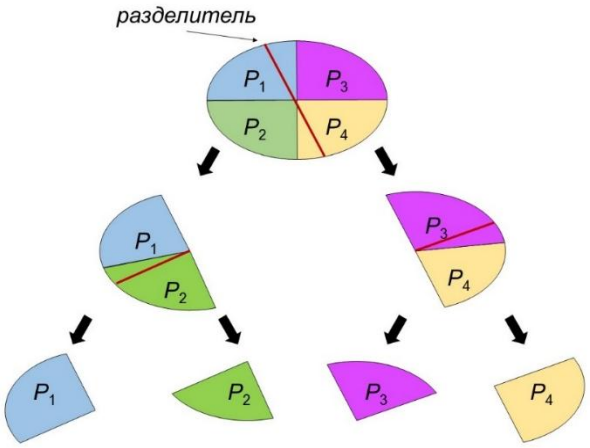


**Рис. 1**. Переупорядочение графа, хранящегося на 4 процессах, методом вложенных сечений.

Рассмотрим параллельное выполнение этапов многоуровневого метода вложенных сечений P процессами.

1. *Огрубление*.
   - 1.1. На каждом процессе составляется список «свободных» вершин, то есть вершин, у которых пара еще не найдена. Для каждой «свободной» вершины определяется пара-кандидат по алгоритму HEM, HCM или RM. Если вершина-кандидат является локальной для процесса, то обе вершины помечаются как «занятые» и объединяются в пару. Иначе обе вершины помечаются как «заблокированные» и такая пара будет рассмотрена позже. При этом пара «заблокированных» вершин помещается в список-«запрос» для другого процесса.
   - 1.2. Процессы обмениваются списками-«запросами». Запрос на объединение принимается, если вершина с чужого процесса была свободна или она отправила такой же запрос на объединение. Иначе запрос отклоняется. Результаты обработки запросов рассылаются процессам-отправителям, которые корректируют свое паросочетание.
   - 1.3. Процесс объединения вершин 1.1 – 1.2 может повторяться определенное число раз или до тех пор, пока есть свободные вершины.
   - 1.4. Одним процессом выполняется распределение вершин нового, огрубленного графа, по процессам. При этом, если вершина огрубленного графа получена в результате объединения вершин с одного процесса, то она будет локальной для того же процесса. Иначе вершина будет локальной для того процесса из пары, на котором меньше своих новых локальных вершин.
   - 1.5. Каждый процесс собирает с других процессов информацию о вершинах, пары с которыми станут локальными для данного процесса (списки смежности, веса, вспомогательная информация).
   - 1.6. На каждом процессе выполняется построение локальной части огрубленного графа.
   - 1.7. Если на каждый процесс приходится слишком малая часть графа, то выполняется процесс «сложение и дублирование» (рис. 2). Процессы, на которых хранился граф, разделяются на две группы, и граф перераспределяется целиком между процессами в каждой группе. Таким образом, получается два экземпляра графа, каждый из которых распределен между P / 2 процессами. Этот прием позволяет найти лучший разделитель за счет выполнения вычислений из различных начальных условий для алгоритмов, использующих случайные числа. В результате огрубления со «сложением и дублированием» на каждом процессе хранится свой наименьший огрубленный граф $G_m$.

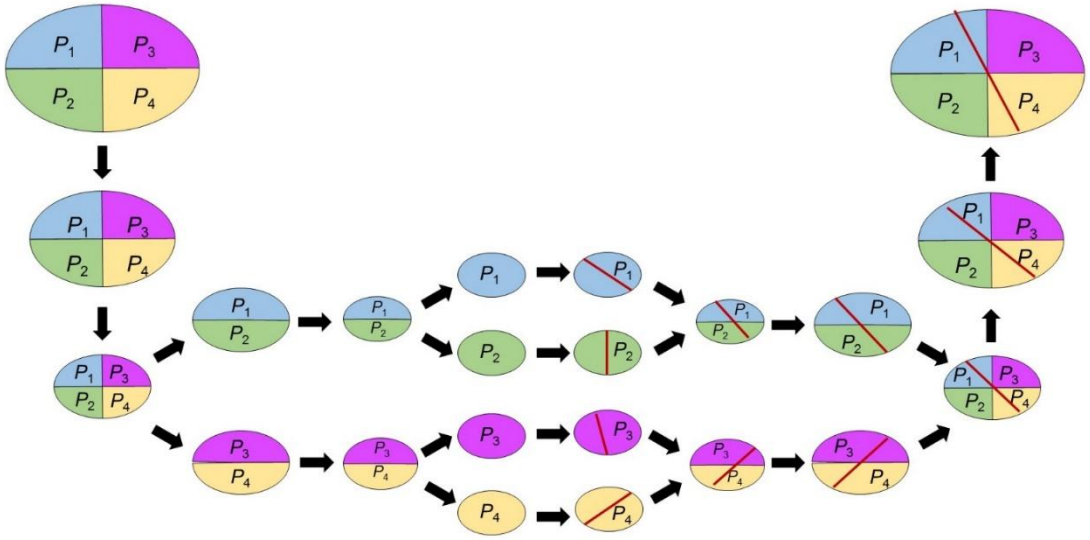


**Рис. 2**. Поиск разделителя графа, хранящегося на 4 процессах, с процедурой «сложение и дублирование».

2. *Разделение.* Этот этап выполняется последовательно, локально на каждом процессе. Если в результате «сложения и дублирования» граф $G_m$ распределен между несколькими процессами, то перед разделением выполняется сборка графа на каждом процессе. Для вычисления вершинного разделителя графа используется обычный или жадный поиск в ширину из нескольких случайных вершин. В результате этого этапа на каждом процессе хранится разделение графа $G_m$.
3. *Развертывание*. Для каждой пары графов $G_{i+1}$ и $G_i$, в которой $G_{i+1}$ был построен как огрубление графа $G_i$, выполняется:
   3.1. Каждый процесс выполняет проекцию разделения графа $G_{i+1}$ на граф $G_i$, а также вспомогательной локальной информации о разделении.
   3.2. Все процессы строят вспомогательный ленточный граф $B$, содержащий в себе разделитель и прилегающие к нему вершины графа $G_i$. Для этого выполняется параллельный поиск в ширину из вершин, принадлежащих разделителю. Количество уровней поиска в ширину определяется параметрами алгоритма. Вершины последнего уровня дерева поиска связываются с «мнимыми» вершинами, которые заменяют множество отброшенных вершин из каждой части разделения. В результате этой процедуры на каждом процессе хранится локальная копия графа $B$ и его разделение.
   3.3. Каждый процесс выполняет улучшение разделения графа $B$, начиная со случайной перестановки его разделителей. Улучшение разделения выполняется алгоритмом Эшкрафта–Лю.
   3.4. Выбирается лучшее разделение графа $B$ и рассылается по всем процессам. После этого каждый процесс проецирует разделение графа $B$ на исходный граф $G_i$ (рис. 3).
4. *Построение новых несвязных графов*. На этом этапе выполняется формирование новых подграфов, образующихся после удаления разделителя, и распределение их по процессам. Считается, что после удаления разделителя образуется ровно два подграфа, состоящих из вершин первой и второй части разделения. Если исходный граф хранился на P процессах, то новые подграфы будут храниться каждый на P / 2 процессах. Для этого:
   4.1. Для каждого процесса определяется, какой из новых подграфов он будет хранить, и сколько вершин будут для процесса локальными. Распределение вершин выполняется так, чтобы у процессов оказалось близкое число локальных вершин. При этом, в большинстве случаев, процесс будет хранить ту часть графа, вершин которой у него было больше. Вершины из другой части графа и избыток вершин из своей части графа будут отправлены другим процессам.

4.2. Каждый процесс отправляет другим процессам информацию о вершинах, которые станут для них локальными.
4.3. На каждом процессе выполняется построение локальной части нового графа.

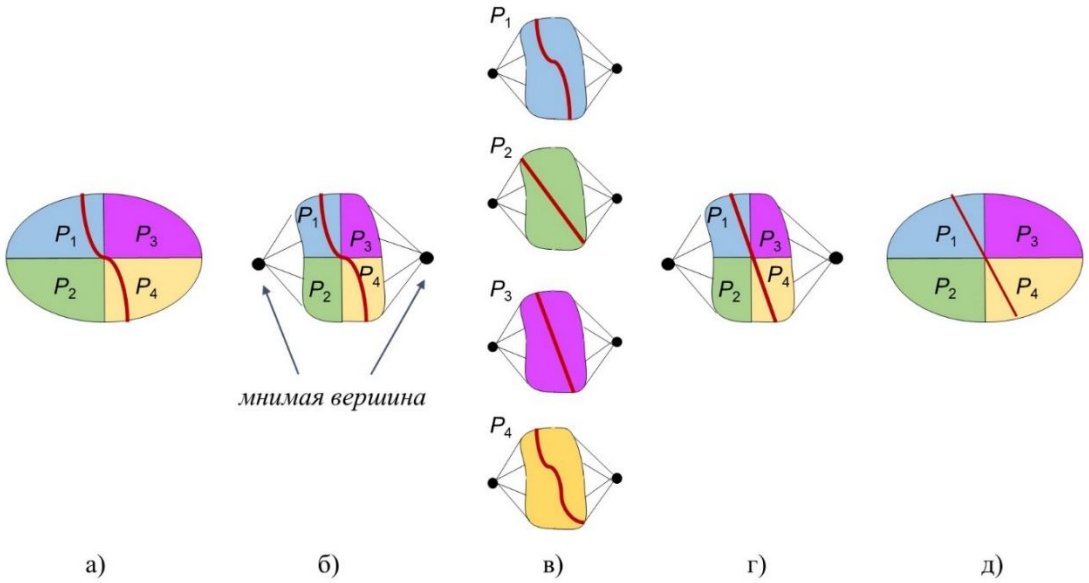


**Рис. 3**. Улучшение разделения графа, хранящегося на 4 процессах.
а) начальный разделитель в исходном графе; б) начальный разделитель в ленточном графе;
в) улучшенный разделитель, найденный на каждом процессе;
г) лучший разделитель, найденный для ленточного графа;
д) улучшенный разделитель, спроецированный на исходный граф.

# 3. Результаты вычислительных экспериментов

## 3.1 Методика проведения экспериментов

Эксперименты проводились на кластере ННГУ «Лобачевский» с использованием двух узлов следующей конфигурации: 2 x 8 ядер Intel Xeon CPU E5-2660 2,20 GHz, 128 GB RAM. Программные пакеты ParMETIS v.4.0.3, PT-Scotch v.6.0.4 и DMORSy собраны из исходных кодов компилятором из пакета Intel Parallel Studio XE, запуски производились с параметрами по умолчанию. Результаты экспериментов представлены на симметричных матрицах из коллекции Suite Sparse. Таблица 1 содержит параметры тестовых матриц.

**Таблица 1. Параметры тестовых матриц**

| Матрица | Размер (N) | Число ненулевых элементов (NZ) |
|---|---|---|
| ecology1 | 1 000 000 | 2 998 000 |
| dielFilterV3real | 1 102 824 | 45 204 422 |
| dielFilterV2real | 1 157 456 | 24 848 204 |
| thermal2 | 1 228 045 | 4 904 179 |
| Serena | 1 391 349 | 32 961 525 |
| Geo_1438 | 1 437 960 | 32 297 325 |
| StocF-1465 | 1 465 137 | 11 235 263 |
| Hook_1498 | 1 498 023 | 31 207 734 |
| af_shell10 | 1 508 065 | 27 090 195 |
| Flan_1565 | 1 564 794 | 59 485 419 |

## 3.2 Анализ производительности

Для анализа масштабируемости разработанного прототипа параллельного переупорядочивателя DMORSy были проведены следующие запуски. Число используемых процессоров варьировалось от 1 до 4, число процессов, выполняющихся на одном процессоре, изменялось от 1 до 8 по количеству доступных вычислительных ядер. Результаты экспериментов представлены в таблице 2.

**Таблица 2. Время работы пакета DMORSy. Время дано в секундах**

| Матрица | Время работы DMORSy на 1–32 ядрах | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 16 | 32 |
| ecology1 | 10,85 | 5,12 | 2,76 | 1,63 | 1,14 | 0,99 |
| dielFilterV3real | 46,09 | 24,73 | 14,76 | 9,55 | 8,50 | 7,07 |
| dielFilterV2real | 32,01 | 17,05 | 11,19 | 6,79 | 4,68 | 4,35 |
| thermal2 | 15,46 | 7,76 | 4,12 | 2,47 | 1,78 | 1,41 |
| Serena | 33,09 | 18,07 | 10,48 | 7,78 | 9,82 | 11,74 |
| Geo_1438 | 40,77 | 21,27 | 11,89 | 9,96 | 5,86 | 5,24 |
| StocF-1465 | 20,76 | 10,11 | 5,38 | 3,35 | 2,42 | 2,14 |
| Hook_1498 | 65,47 | 57,28 | 53,13 | 25,85 | 24,72 | 23,84 |
| af_shell10 | 44,77 | 22,26 | 11,59 | 6,30 | 5,14 | 2,91 |
| Flan_1565 | 62,54 | 31,13 | 16,39 | 8,71 | 6,53 | 5,62 |

Результаты показывают, что на всех рассмотренных матрицах, кроме одной, время работы сокращается вплоть до 32 ядер, при этом эффективность сильной масштабируемости постепенно падает и на большинстве матриц составляет 25-30%. Далее в таблице 3 представлено время работы пакета ParMETIS на тех же входных данных. Результаты показывают, что при переходе через границу одного узла (16 ядер) время работы практически перестает сокращаться, эффективность масштабируемости на 16 ядрах составляет в среднем 17%.

**Таблица 3. Время работы пакета ParMETIS. Время дано в секундах**

| Матрица | Время работы ParMETIS на 1–32 ядрах | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 16 | 32 |
| ecology1 | 6,76 | 4,23 | 2,73 | 2,06 | 1,83 | 1,77 |
| dielFilterV3real | 13,38 | 11,16 | 9,74 | 9,25 | 8,98 | 10,13 |
| dielFilterV2real | 20,72 | 13,08 | 9,45 | 7,43 | 6,52 | 7,03 |
| thermal2 | 10,81 | 6,55 | 4,36 | 3,11 | 2,61 | 2,43 |
| Serena | 17,45 | 13,89 | 10,71 | 9,16 | 8,15 | 8,34 |
| Geo_1438 | 16,55 | 13,00 | 10,06 | 8,58 | 7,70 | 7,96 |
| StocF-1465 | 21,79 | 12,83 | 8,20 | 6,25 | 5,21 | 4,96 |
| Hook_1498 | 17,50 | 13,66 | 10,39 | 8,66 | 7,47 | 7,60 |
| af_shell10 | 7,38 | 6,84 | 5,80 | 5,14 | 4,67 | 4,63 |
| Flan_1565 | 23,08 | 17,69 | 13,62 | 11,90 | 10,73 | 11,14 |

Представленные в таблице 4 результаты запусков переупорядочивателя PT-Scotch показывают, что использование 32 ядер для перестановки рассматриваемых матриц в основном не является оправданным. Время работы при переходе с 16 на 32 ядра сокращается незначительно, эффективность масштабируемости на 16 ядрах составляет ~25-30%, а на 32 ядрах – около 15%.

**Таблица 4. Время работы пакета PT-Scotch. Время дано в секундах**

| Матрица | Время работы PT-Scotch на 1–32 ядрах | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 8 | 16 | 32 |
| ecology1 | 4,52 | 2,62 | 1,71 | 1,34 | 1,22 | 1,25 |
| dielFilterV3real | 45,90 | 24,97 | 16,74 | 12,83 | 11,04 | 10,22 |
| dielFilterV2real | 31,31 | 16,93 | 10,40 | 8,06 | 6,51 | 6,41 |
| thermal2 | 7,17 | 4,09 | 2,55 | 1,95 | 1,70 | 1,59 |
| Serena | 34,45 | 19,07 | 11,85 | 8,97 | Ошибка | 7,57 |
| Geo_1438 | 31,77 | 17,96 | 11,10 | 8,29 | 7,44 | 6,97 |
| StocF-1465 | 18,31 | 10,08 | 5,85 | 3,99 | 3,09 | 2,91 |
| Hook_1498 | 32,33 | 18,33 | 11,40 | 8,52 | 7,13 | 6,79 |
| af_shell10 | 19,98 | 11,23 | 7,01 | 5,29 | 4,55 | 4,40 |
| Flan_1565 | 50,00 | 27,81 | 16,80 | 12,07 | 9,96 | 9,92 |

Значительный интерес для практического использования представляет сравнение времени работы указанных пакетов (рис. 4). Для каждой матрицы указано наименьшее время, за которое удалось решить задачу для каждого из рассматриваемых пакетов. Результаты показывают, что для большинства рассмотренных матриц DMORSy позволяет построить перестановку за меньшее время.

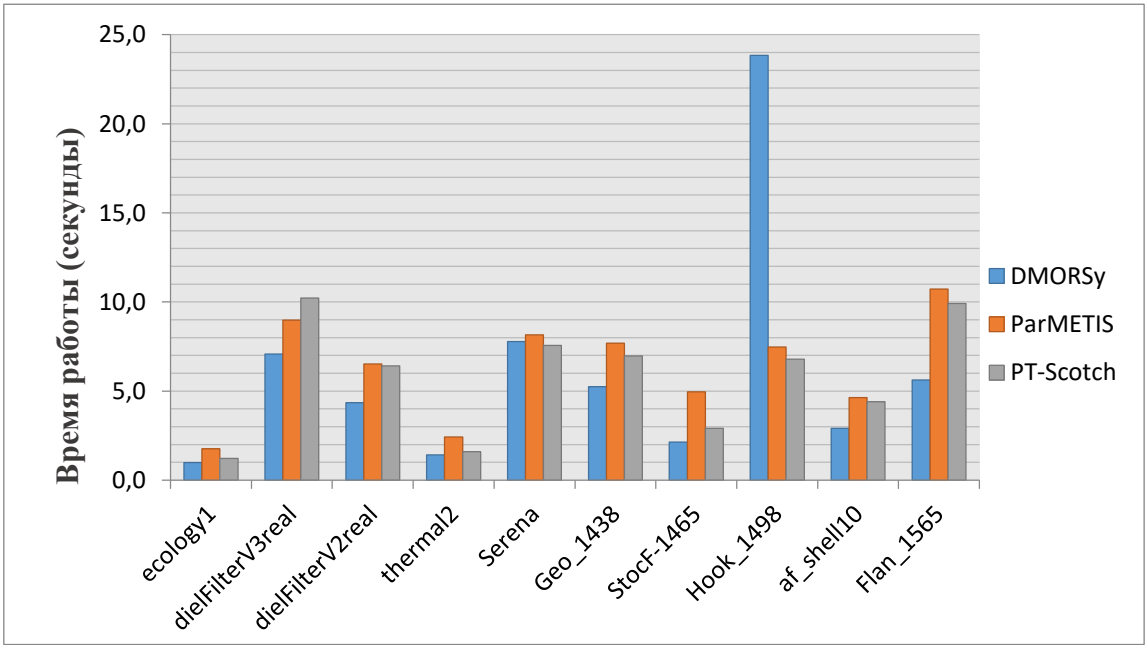


**Рис. 4**. Лучшее время решения задачи в пакетах DMORSy, ParMETIS и PT-Scotch

## 3.3 Анализ качества переупорядочения

Перейдем к сравнению качества перестановок, построенных рассматриваемыми пакетами. Для этого сравним заполнение фактора для каждой из рассмотренных матриц.

Сопоставим минимальное число ненулевых элементов в факторах матриц, достигнутое в результате перестановок с использованием рассматриваемых пакетов (рис. 5). Из диаграммы видно, что в целом результаты экспериментов сравнимы. На матрицах ecology1, af_shell10 и Flan_1565 пакет DMORSy опережает аналоги, в других случаях ParMETIS показывает лучшие результаты. Сравним заполнение фактора матриц, полученное в каждом пакете за наименьшее время работы (рис. 6). В этом случае DMORSy позволяет найти лучшие перестановки для 5 матриц из тестового набора, выигрывая до 9% заполнения фактора в сравнении с ParMETIS и до 36% в сравнении с PT-Scotch.

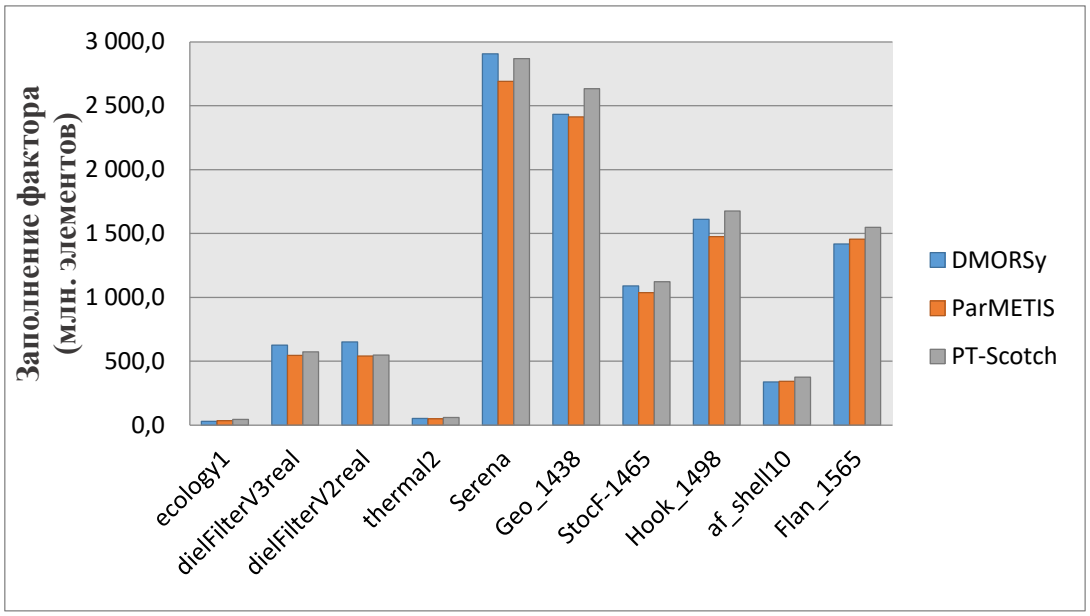


**Рис. 5**. Лучшее качество перестановок в пакетах DMORSy, ParMETIS и PT-Scotch

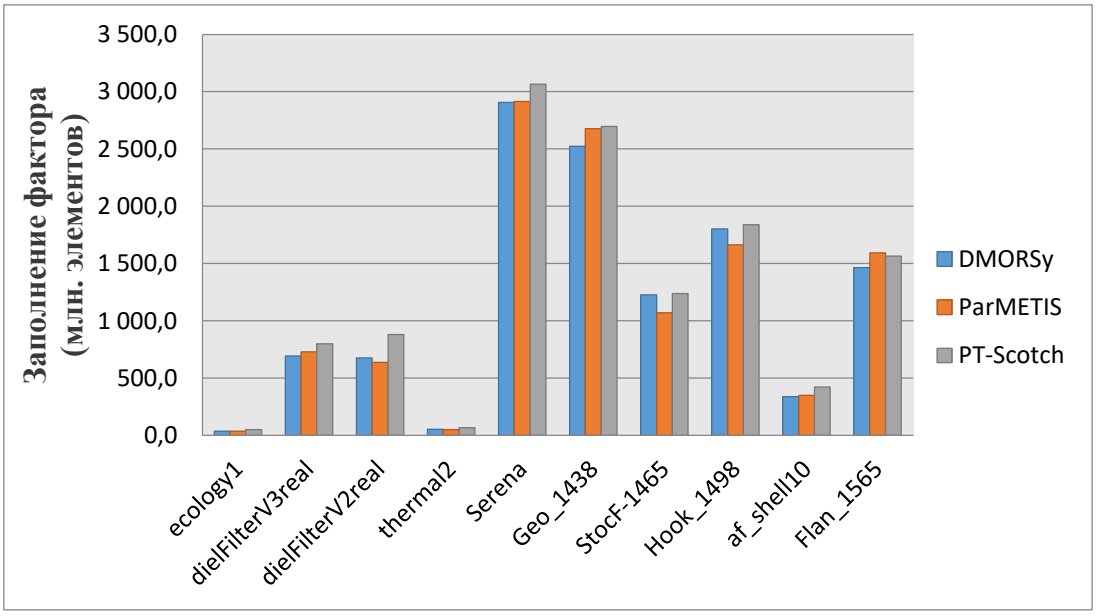


**Рис. 6**. Качество перестановок, полученное за наименьшее время переупорядочения в пакетах DMORSy, ParMETIS и PT-Scotch

Анализ полных экспериментальных данных дает представление о характере изменения заполнения фактора в зависимости от числа используемых ядер и конфигурации запуска. Так, в большинстве случаев увеличение числа MPI-процессов ведет к ухудшению получаемых перестановок с точки зрения заполнения фактора рассматриваемых матриц. Данный факт объясняется тем обстоятельством, что во всех алгоритмах переупорядочения, ориентированных на системы с распределенной памятью, ведется поиск компромисса между временем работы (и, соответственно, временем коммуникаций) и качеством перестановок. При этом некоторые обмены данными, потенциально полезные для улучшения перестановок, не производятся.

## Заключение

В статье описано продолжение работ по разработке переупорядочивателя симметричных разреженных матриц. Ранее авторами был представлен пакет PMORSy для систем с общей па-

мятью [8]. В данной работе описан разработанный на его основе прототип параллельного переупорядочивателя DMORSy для систем с распределенной памятью. Результаты вычислительных экспериментов на матрицах из коллекции Suit Sparse показали, что DMORSy позволяет получить перестановки, сравнимые с аналогами по времени и качеству (числу ненулевых элементов фактора матрицы). На ряде тестовых матриц DMORSy опережает аналогичные пакеты в смысле указанных критериев.

В ходе дальнейших исследований предполагается совместить лучшие стороны PMORSy и DMORSy в рамках одного алгоритма, использующего комбинацию технологий MPI + OpenMP для более рационального использования вычислительных ресурсов. Предполагается, что возможность использования систем с распределенной памятью в DMORSy позволит разделять работу по узлам кластера с использованием технологии MPI, а многопоточные алгоритмы из PMORSy обладают потенциалом получения хороших перестановок за лучшее время при работе в пределах одного узла. Данное направление является основным для дальнейшей работы.

## Литература


1. 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
2. Karypis G. and Kumar V. ParMetis: Parallel graph partitioning and sparse matrix ordering library. Tech. Rep. TR 97-060, University Of Minnesota, Department Of Computer Science. 1997.
3. Karypis G., Kumar V. A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering // J. Parallel and Distributed Computing. 1998. Vol. 48, No. 1. P. 71–95. DOI:10.1006/jpdc.1997.1403
4. 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
5. LaSalle D. and Karypis G. Efficient Nested Dissection for Multicore Architectures // Euro-Par 2015: Parallel Processing. 2015, Springer Berlin Heidelberg. P. 467–478. DOI:10.1007/978-3-662-48096-0_36
6. Intel MKL PARDISO – Parallel Direct Sparse Solver Interface. URL: https://software.intel.com/en-us/mkl-developer-reference-c-intel-mkl-pardiso-parallel-direct-sparse-solver-interface (дата обращения: 20.02.2018)
7. Parallel Direct Sparse Solver for Clusters Interface. URL: https://software.intel.com/en-us/mkl-developer-reference-c-parallel-direct-sparse-solver-for-clusters-interface (дата обращения: 20.02.2018)
8. 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.
9. Davis T. A., Hu Y. The University of Florida sparse matrix collection //ACM Transactions on Mathematical Software (TOMS). 2011. Vol. 38, No. 1. P. 1–25. DOI: 10.1145/2049662.2049663
10. 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.