2025, **26** (2), 140–149. doi 10.26089/NumMet.v26r210

doi 10.26089/NumMet.v26r210 УДК 519.612.2

Асинхронная реализация метода Холецкого для разреженных матриц на компьютерах с NUMA-архитектурой

А. С. Маслов
ООО "ТС Интеграция", Москва, Российская Федерация
ORCID: 0009-0001-1098-829X, e-mail: maslov.andrei@phystech.edu

М. М. Макаров
ООО "ТС Интеграция", Москва, Российская Федерация
ORCID: 0009-0000-5397-7139, e-mail: MMMakarov@t1.ru

Н. Н. Потравкин
ООО "ТС Интеграция", Москва, Российская Федерация
ORCID: 0000-0001-9677-5158, e-mail: potravkin@physics.msu.ru

С. О. Проскурня
ООО "ТС Интеграция", Москва, Российская Федерация
e-mail: SProskurnia@t1.ru

Аннотация: Реализован параллельный алгоритм разложения Холецкого для разреженных матриц, основанный на парадигме асинхронного выполнения и учитывающий особенности NUMA-архитектуры. Выполнение стадий численного разложения и прямой/обратной подстановки представляется в виде ориентированного ациклического графа, что позволяет обходиться без барьеров синхронизации, а также увеличить локальность доступа к данным с целью более эффективного использования иерархии подсистемы памяти вычислительного устройства. Оценка производительности показывает хорошую масштабируемость в сравнении с высоко оптимизированным коммерческим пакетом Intel MKL PARDISO, подтверждая эффективность предлагаемого подхода.

Ключевые слова: разложение Холецкого, NUMA-архитектура, парадигма асинхронного выполнения, ориентированный ациклический граф, библиотека hwloc.

Благодарности: При получении результатов использовалось оборудование ООО "ТС Интеграция".

Для цитирования: Маслов А.С., Макаров М.М., Потравкин Н.Н., Проскурня С.О. Асинхронная реализация метода Холецкого для разреженных матриц на компьютерах с NUMA-архитектурой // Вычислительные методы и программирование. 2025. **26**, № 2. 140–149. doi 10.26089/NumMet.v26r210.

2025, **26** (2), 140–149. doi 10.26089/NumMet.v26r210

An asynchronous sparse Cholesky solver for workstations with NUMA architecture

Andrei S. Maslov
LLC "TS INTEGRATION", Moscow, Russia
ORCID: 0009-0001-1098-829X, e-mail: maslov.andrei@phystech.edu

Michail M. Makarov
LLC "TS INTEGRATION", Moscow, Russia
ORCID: 0009-0000-5397-7139, e-mail: MMMakarov@t1.ru

Nikolay N. Potravkin
LLC "TS INTEGRATION", Moscow, Russia
ORCID: 0000-0001-9677-5158, e-mail: potravkin@physics.msu.ru

Svyatoslav O. Proskurnia
LLC "TS INTEGRATION", Moscow, Russia
e-mail: SProskurnia@t1.ru

Abstract: A parallel Cholesky factorization algorithm for sparse matrices has been implemented, based on the asynchronous task paradigm and accounting for the specifics of NUMA architecture. The execution of the numerical factorization stage and forward/backward substitution is represented as a directed acyclic graph (DAG), which removes synchronization barriers and enhances data access locality to improve the utilization efficiency of the computational device's memory hierarchy. Performance evaluation demonstrates good scalability compared to the highly optimized commercial package Intel MKL PARDISO, confirming the effectiveness of the proposed approach.

Keywords: Cholesky factorization, NUMA architecture, asynchronous task paradigm, directed acyclic graph, hwloc library.

Acknowledgements: The equipment provided by LLC "TC Integration" was utilized in the acquisition of the results.

For citation: A. S. Maslov, M. M. Makarov, N. N. Potravkin and S. O. Proskurnia, "An asynchronous sparse Cholesky solver for workstations with NUMA architecture," Numerical Methods and Programming. **26** (2), 140–149 (2025). doi 10.26089/NumMet.v26r210.

1. Введение. Решение разреженных симметричных положительно определенных систем линейных уравнений большой размерности лежит в основе многих прикладных задач в инженерии и имеет ключевое значение для высокопроизводительных научных расчетов. Эффективность решения таких систем зачастую определяет общую производительность конечного приложения. В данной работе рассматривается прямой метод решения систем, основанный на разложении Холецкого, который сводится к представлению симметричной положительно определенной матрицы коэффициентов A в виде произведения нижнетреугольной матрицы L на ее транспонированную $A = LL^T$. Получение такого представления для матриц большой размерности может быть достаточно дорогим, с точки зрения требований к оперативной памяти и времени выполнения, по сравнению с решением системы итерационными методами. При этом в ряде приложений возникают плохо обусловленные системы, для которых усилия на построение хорошего предобуславливателя могут превысить стоимость прямого решения [1]. Кроме того, прямые методы обеспечивают эффективные средства для решения нескольких систем с одинаковой матрицей коэффициентов и разными векторами правой части, поскольку разложение должно быть выполнено только один раз. Также прямой метод часто используется на самом грубом уровне при построении многосеточных методов [2].

Достижения в области комбинаторных методов в сочетании с современными компьютерными архитектурами значительно повлияли на разработку передовых прямых решателей, которые позволяют эффективно решать системы большей размерности в вычислительных средах с быстро увеличивающимися объемами памяти и количеством вычислительных ядер. В литературе было предложено несколь-

2025, **26** (2), 140–149. doi 10.26089/NumMet.v26r210

ко параллельных алгоритмов для разложения Холецкого, наиболее известные из которых PARDISO [1], MUMPS [3], PASTIX [4], SymPACK [5]. Существующие прямые методы для решения систем с разреженными матрицами агрегируют близкие с точки зрения структуры заполнения столбцы в т.н. суперноды, что позволяет использовать высокооптимизированные базовые подпрограммы линейной алгебры (BLAS) для вычислений с плотными блоками. Традиционной метрикой эффективности численных алгоритмов было количество выполняемых арифметических операций. Технологический прогресс в области микроэлектроники обеспечивает сокращение доли времени выполнения арифметических операций в общем времени выполнения алгоритма, поэтому они уже не являются узким местом для многих алгоритмов. Вместо этого узким местом становится перемещение данных как между уровнями иерархии памяти в рамках одного процесса, так и между параллельными процессами по сети. Это побуждает к модификации существующих и разработке новых алгоритмов, минимизирующих перемещение данных [6]. Несмотря на то, что для процесса и его потоков операционная система создает иллюзию плоского адресного пространства, время доступа потока к странице памяти может зависеть от того, где расположены сам поток и эта страница. Системы, в которых эти эффекты существенны, часто называют вычислительными узлами с неоднородным временем доступа к памяти (Non-Uniform Memory Access, NUMA). В NUMA-системах несколько процессоров могут иметь свои локальные блоки памяти и доступ к этой локальной памяти быстрее, чем доступ к другим блокам памяти. Детали построения алгоритмов разложения Холецкого как для систем с общей, так и систем с распределенной памятью достаточно подробно освещены в литературе [7]. При этом тема оптимизации таких подходов для систем с NUMA-архитектурой практически не освещена.

В этой работе мы предлагаем модификацию алгоритма разложения Холецкого для разреженных матриц, основанную на парадигме асинхронного выполнения. Аналогично [5] стадии численного разложения и прямой/обратной подстановки представляются в виде ориентированного ациклического графа, вершины которого ассоциированы с порциями работы, а ребра определяют зависимости при выполнении этой работы. Для организации параллелизма внутри приложения мы используем кроссплатформенную библиотеку шаблонов C++ TBB [8]. В отличие от [5] мы не ассоциируем с отдельными вершинами графа выполнения суперноды небольшого размера, находящиеся вдали от корня дерева исключения, а аналогично [9] объединяем их в группы, что позволяет повысить локальность доступа к данным и уменьшить накладные расходы, связанные с необходимостью порождать в планировщике работы большое количество мелких задач. В контексте организации параллельных вычислений такие задачи, представляющие отдельные единицы работы, которые могут быть выполнены параллельно, часто называют подзадачами (tasks). Измерения производительности показывают хорошую масштабируемость по сравнению с высоко оптимизированным коммерческим пакетом Intel MKL. С использованием инструментов, предоставляемых библиотекой TBB, все данные, ассоциированные с некоторой группой суперноd небольшого размера вдали от корня дерева исключения, размещаются в одном блоке памяти и обрабатываются вычислительными ядрами, для которых доступ к этой памяти является наиболее быстрым (локальным). Благодаря этому достигается значительное повышение производительности стадии прямой/обратной подстановки.

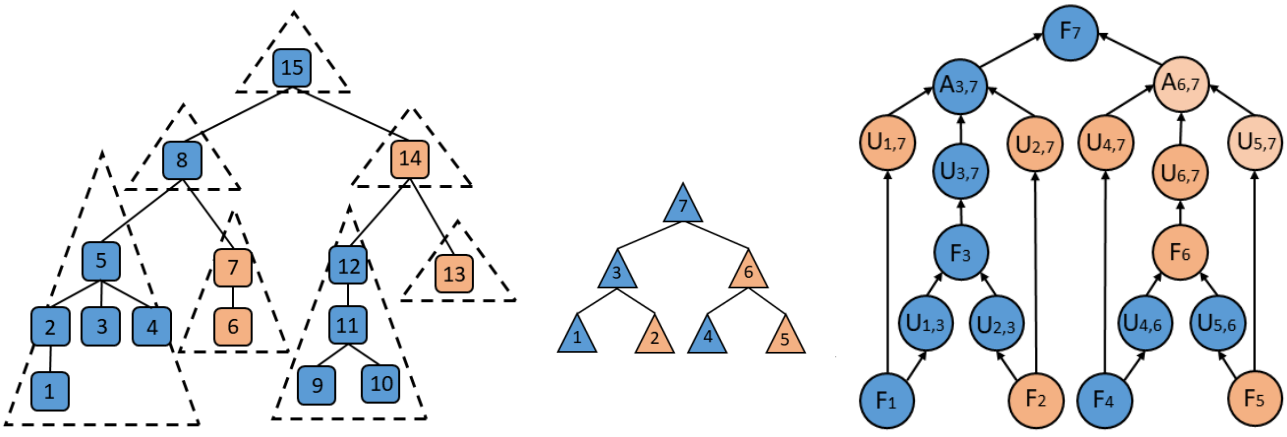
2. Базовые блоки алгоритма разложения Холецкого. Мы рассматриваем прямой метод решения системы уравнений $A\boldsymbol{x} = \boldsymbol{b}$. Получение решения такой системы прямым методом условно делится на три стадии. Разложение Холецкого симметричной положительно определенной разреженной матрицы A, сводящееся к представлению этой матрицы в виде произведения нижнетреугольной матрицы L на ее транспонированную $A = LL^T$, называется стадией численного разложения. Нахождение вектора решения $\boldsymbol{x}$ с использованием вектора правой части $\boldsymbol{b}$ и матрицы L, полученной на стадии численного разложения, называется стадией прямой/обратной подстановки. На месте нулевых элементов матрицы A могут появиться ненулевые в полученной в результате разложения матрице L. Эти дополнительные ненулевые элементы называются заполнением (fill-in). Для уменьшения заполнения и, как следствие, времени вычислений разложение Холецкого применяют к матрице PAP^T, полученной из исходной матрицы A симметричным переупорядочиванием строк и столбцов. Получение переупорядочивания, минимизирующего заполнение, называется стадией символьного разложения.

Исследования методов разбиения графов привели к появлению высококачественных программных пакетов, например Metis [10] и Scotch [11]. Реализованные в них методы вложенных сечений позволяют получать переупорядочивания P, которые, с одной стороны, приводят к небольшому заполнению для методов разложения, а с другой стороны, обеспечивают высокий уровень параллелизма. Одним из ключевых объектов при построении параллельных алгоритмов разложения Холецкого является дерево исключения [12], определяющее зависимости между столбцами на стадиях численного разложения

2025, **26** (2), 140–149. doi 10.26089/NumMet.v26r210

и прямой/обратной подстановки. Использование методов вложенных сечений позволяет получать хорошо сбалансированное дерево исключения и параллельно выполнять работу, ассоциированную с ветвями этого дерева. Для увеличения производительности разреженные прямые методы агрегируют близкие с точки зрения структуры заполнения столбцы в суперноды, что позволяет использовать высокооптимизированные базовые подпрограммы линейной алгебры и оперировать деревом исключения для супернод. Существует значительная свобода в стратегиях организации вычислений, которые могут использовать алгоритмы разложения. Единственные ограничения, которые должны соблюдаться — это зависимости между супернодами, описываемые деревом исключения, но порядок, в котором происходят обновления, математически не имеет значения.

3. Алгоритм разложения Холецкого, учитывающий особенности NUMA-архитектуры. В [5] приведена достаточно подробная классификация алгоритмов разложения Холецкого с точки зрения организации вычислений. На рис. 1 a приведен пример дерева исключения суперноd. Аналогично вычислительным платформам с распределенной памятью рабочие станции с NUMA-архитектурой к альтернативе о порядке вычислений добавляют вопрос о том, где эти вычисления будут выполняться. Для увеличения локальности доступа к данным и минимизации негативного NUMA-эффекта группу суперноd небольшого размера, образующих поддерево в исходном дереве исключения, мы ассоциируем с одним процессором. Разными цветами показано отображение данных, связанных с супернодами, на NUMA-узлы рабочей станции с двумя процессорами. Описание стратегий привязки данных, возникающих в процессе разложения Холецкого, к вычислительным узлам можно найти в [13]. На рис. 1 b изображено дерево, вершины 3, 6, 7 которого соответствуют сепараторам верхнего уровня, а вершины 1, 2, 4, 5 соответствуют поддеревьям исходного дерева исключения суперноd на рис. 1 a. Для управления привязкой данных к NUMA-узлам мы пользуемся функциональностью библиотек TBB и hwloc [14]. При выполнении алгоритма на рабочей станции NUMA создается массив объектов $tbb :: task\_arena$, размер которого равен количеству вычислительных узлов. Каждый объект $tbb :: task\_arena$ из этого массива при своей инициализации привязывается к соответствующему вычислительному узлу, что позволяет выбирать вычислительный узел для выполнения подзадач и размещать объекты в локальном блоке памяти для этого узла. В нашей работе мы используем подход, в котором выполнение стадий численного разложения и прямой/обратной подстановки представляется в виде ориентированного ациклического графа, что позволяет обойтись без барьеров синхронизации и иметь достаточную гибкость при загрузке работой доступных для выполнения задач процессоров рабочей станции NUMA. На рис. 1 c изображен граф выполнения стадии численного разло-



a) b) c)

Рис. 1. Формирование ациклического ориентированного графа выполнения алгоритма: a) дерево исключения суперноd; b) дерево, полученное объединением некоторых вершин исходного дерева исключения; c) граф выполнения стадии численного разложения. Цветом обозначено отображение суперноd и вершин графа выполнения на два процессора рабочей станции NUMA

Fig. 1. Formation of an acyclic directed graph of algorithm execution: a) supernode elimination tree; b) tree obtained by merging some nodes of the original elimination tree; c) execution graph of the numerical decomposition stage. The color indicates the mapping of supernodes and execution graph vertices onto two processors of the NUMA workstation

2025, **26** (2), 140–149. doi 10.26089/NumMet.v26r210

жения, соответствующий дереву исключения на рис. 1 а. Аналогично [5] мы используем три типа подзадач для построения графа выполнения стадий численного разложения. Разложение i-й группы супернод в дереве исключения обозначено как F_i. Вычисление обновлений j-й группы от i-й обозначено как $U_{i,j}$. Стадия агрегации обновлений обозначена как $A_{i,j}$. Каждый из трех типов подзадач, выполняющийся на одном из процессоров рабочей станции NUMA, в свою очередь может породить дочерние подзадачи, попадающие в планировщик работы TBB для выполнения на этом же процессоре, что повышает качество динамической балансировки работы между ядрами процессоров, не ухудшая при этом время доступа к данным. Стрелочками на рис. 1 с изображены зависимости между подзадачами. В нашей программе учет зависимостей реализован через наличие атомарного счетчика $std :: atomic$ у каждой из подзадач в графе выполнения. На момент начала выполнения графа, изображенного на рис. 1 с, подзадачи F_1, F_2, F_4, F_5 не имеют зависимостей от других подзадач, поэтому их атомарные счетчики равны нулю. Подзадачи F_1, F_4 сразу могут быть отправлены на выполнение на первый вычислительный узел, а подзадачи F_2, F_5 — на второй. Подзадача F_3 зависит от $U_{1,3}$ и $U_{2,3}$, поэтому ее атомарный счетчик зависимостей инициализируется значением 2. После выполнения произвольной подзадачи атомарный счетчик зависящей от нее подзадачи уменьшается на единицу. Когда счетчик становится равным нулю, подзадача отправляется на выполнение. Для улучшения балансировки вычислительной работы между процессорами мы допускаем объединение не более 1024 столбцов в одну суперноду. При выполнении стадии прямой/обратной подстановки граф выполнения является деревом. Ребра этого дерева, определяющие зависимости между подзадачами, ориентированы от листьев к корню в случае прямой и в противоположную сторону в случае обратной подстановки.

4. Измерение производительности. Мы оценивали производительность нашего параллельного алгоритма разложения Холецкого для разреженных матриц, основанного на парадигме асинхронного выполнения, на двух рабочих станциях NUMA. Первая рабочая станция имеет два процессора Intel(R) Xeon(R) Gold 6354 CPU, каждый из которых имеет 18 вычислительных ядер. Вторая рабочая станция имеет четыре процессора Intel(R) Xeon(R) Gold 6348H CPU, каждый из которых имеет 24 вычислительных ядра. В качестве входных данных мы использовали набор матриц из коллекции разреженных матриц, находящихся в открытом доступе [15]. Описание каждой из используемых нами матриц можно найти в табл. 1. Для получения переупорядочивания, уменьшающего заполнение, мы используем библиотеку Metis [10]. В качестве базовых подпрограмм линейной алгебры мы используем оптимизированную для процессоров Intel реализацию BLAS Intel Math Kernel Library версии 2024.2.

Сначала рассмотрим влияние на время выполнения стадий численного разложения и прямой/обратной подстановки оптимизаций, связанных с размещением данных и задач, учитывающих знание о NUMA-архитектуре. Наши эксперименты показали, что при использовании ядер в количестве, не превышающем их числа на одном процессоре, привязка данных и потоков выполнения задач средствами библиотеки hwloc практически не влияет на время выполнения стадий численного разложения и прямой/обратной подстановки, т.е. это влияние находится в пределах погрешности наших измерений. На рис. 2 представлены зависимости времени выполнения этих стадий, усредненных по трем запускам, на двухпроцессорной рабочей станции NUMA для матрицы Flan_1565 от количества вычислительных ядер. Синим цветом изображены зависимости для прямого решателя Intel MKL PARDISO версии 2024.2. Зеленым и красным цветами изображены зависимости для нашей реализации без использования и с использованием библиотеки hwloc. Такой вид зависимостей является типичным и для остальных матриц из табл. 1. При использовании всех ядер, доступных на двухпроцессорной рабочей станции NUMA, на матрице Flan_1565 использование hwloc дает ускорение при выполнении стадии численного разложения порядка 20%, а для стадии прямой/обратной подстановки порядка 50%. Большее влияние NUMA-оптимизаций на стадию прямой/обратной подстановки объясняется тем, что при выполнении этой стадии над единицей данных, полученных из оперативной памяти, в среднем проделывается гораздо меньше вычислений. На стадии прямой/обратной подстановки использование больше одного процессора не дает прироста производительности в случае отсутствия NUMA-оптимизаций. Это становится еще более критичным для рабочей станции NUMA, имеющей четыре процессора. Для прямого решателя Intel MKL PARDISO на матрице Flan_1565 использование всех доступных ядер не дает прироста производительности относительно однопроцессорного запуска для обеих стадий (см. синие зависимости на рис. 2).

В табл. 2 приведено время выполнения стадий численного разложения и прямой/обратной подстановки на двух рабочих станциях NUMA при использовании всех доступных ядер для матриц из табл. 1. Колонки со значениями времени сгруппированы по три: в первой приведено время для Intel MKL PARDISO,

2025, **26** (2), 140–149. doi 10.26089/NumMet.v26r210

Таблица 1. Матрицы, используемые в измерениях производительности. В третьей и четвертой колонках указаны количество столбцов и количество ненулевых элементов в матрице A. В пятой колонке указано количество ненулевых элементов в матрице L

Table 1. Matrices used in the experiments. The third and fourth columns indicate the number of columns and the number of nonzero elements in matrix A. The fifth column specifies the number of nonzero elements in the matrix L

Имя Name	Тип Type	n	$nnz(A)$	$nnz(L)$
af_shell7	Структурный анализ Structural analysis	504 855	17 579 155	90 982 890
audikw_1	Структурный анализ Structural analysis	943 695	77 651 847	1 249 991 211
bone010	Задача редукции модели Model reduction problem	986 703	47 851 783	1 088 646 300
boneS10	Задача редукции модели Model reduction problem	914 898	40 878 708	267 318 504
Flan_1565	Структурный анализ Structural analysis	1 564 794	114 165 372	1 453 451 592
G3_circuit	Моделирование электрических цепей Modeling of electrical circuits	1 585 478	7 660 826	90 397 858
Hook_1498	Структурный анализ Structural analysis	1 498 023	59 374 451	1 519 326 639
inline_1	Структурный анализ Structural analysis	503 712	36 816 170	167 706 291
StocF-1465	Вычислительная гидродинамика Computational Fluid Dynamics	1 465 137	21 005 389	1 039 392 181
thermal2	Стационарный теплообмен Stationary heat exchange	1 228 045	8 580 313	50 293 930

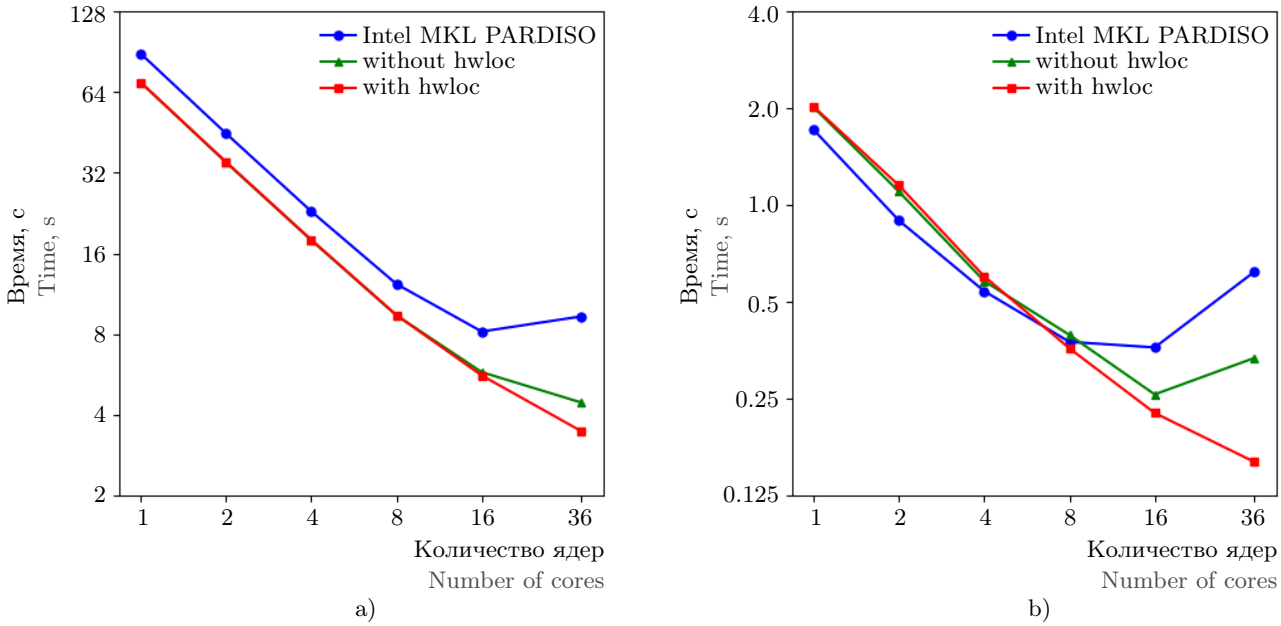


Рис. 2. Время обработки матрицы Flan_1565 на двухпроцессорной рабочей станции NUMA в зависимости от количества вычислительных ядер: a) на стадии численного разложения; b) на стадии прямой/обратной подстановки

Fig. 2. Flan_1565 matrix processing time on a dual-processor NUMA workstation depending on the number of computing cores: a) at the numerical factorization stage; b) at the forward/backward substitution stage

2025, **26** (2), 140–149. doi 10.26089/NumMet.v26r210

Таблица 2. Значения времени выполнения стадий численного разложения и прямой/обратной подстановки, полученные в результате усреднения по трем запускам на двух рабочих станциях NUMA. Колонки со значениями времени сгруппированы по три: в первой из трех колонок приведено время выполнения для прямого решателя Intel MKL PARDISO, во второй и третьей колонках приведено время выполнения для нашей реализации без и с использованием NUMA-оптимизаций

Table 2. The execution times for the numerical factorization and forward/backward substitution stages, averaged over three runs on two NUMA workstations. The columns with the times are grouped into threes: the first of the three columns shows the execution time for the Intel MKL PARDISO direct solver, the second and third columns show the execution time for our implementation without and with NUMA optimizations

Имя Name	Время выполнения численного разложения, с Execution time of numerical factorization, s						Время выполнения прямой/обратной подстановки, с Execution time of forward/backward substitution, s					
	NUMA_2 × 18			NUMA_4 × 24			NUMA_2 × 18			NUMA_4 × 24		
af_shell7	0.302	0.188	0.141	0.754	0.375	0.298	0.073	0.027	0.015	0.090	0.086	0.025
audikw_1	20.05	7.194	5.915	18.75	10.58	9.629	0.499	0.277	0.137	0.483	1.510	0.263
bone010	11.77	5.253	4.493	10.82	8.377	7.564	0.454	0.269	0.132	0.454	1.23	0.260
boneS10	1.097	0.632	0.455	1.519	1.153	0.825	0.175	0.079	0.042	0.209	0.225	0.070
Flan_1565	9.312	4.440	3.472	10.63	6.797	5.847	0.620	0.335	0.159	0.496	1.581	0.307
G3_circuit	0.667	0.201	0.154	1.550	0.284	0.247	0.156	0.041	0.032	0.244	0.102	0.044
Hook_1498	34.03	10.25	8.292	27.03	13.37	12.05	0.711	0.336	0.194	0.768	1.886	0.320
inline_1	0.606	0.376	0.263	0.889	0.756	0.537	0.099	0.048	0.024	0.113	0.149	0.039
StocF-1465	13.99	4.630	3.644	14.70	6.982	6.179	0.437	0.238	0.126	0.434	1.195	0.243
thermal2	0.347	0.088	0.068	1.082	0.128	0.116	0.104	0.026	0.019	0.179	0.051	0.029

Таблица 3. Значения ускорения вычислений на стадиях численного разложения и прямой/обратной подстановки. Колонки со значениями ускорения сгруппированы по парам: первая колонка в паре содержит ускорения для Intel MKL PARDISO на всех доступных ядрах в системе, вторая колонка — ускорения для нашей реализации с использованием библиотеки hwloc

Table 3. Speedup values for computations at the stages of numerical decomposition and forward/backward substitution. Columns with speedup values are grouped into pairs: the first column in the pair contains speedups for Intel MKL PARDISO on all available cores in the system, the second column contains speedups for our implementation using the hwloc library

Имя Name	Ускорение стадии численного разложения Speedup of numerical factorization stage				Ускорение стадии прямой/обратной подстановки Speedup of forward/backward substitution stage			
	NUMA_2 × 18		NUMA_4 × 24		NUMA_2 × 18		NUMA_4 × 24	
af_shell7	6.9	10.5	4.0	7.5	2.2	9.3	3.0	9.3
audikw_1	7.2	18.1	10.9	16.1	3.0	11.2	4.5	10.8
bone010	8.8	13.5	11.9	14.8	2.4	9.5	4.2	9.5
boneS10	7.8	17.3	9.2	11.0	2.7	11.5	3.3	9.5
Flan_1565	9.6	20.0	11.9	16.9	2.8	12.7	5.4	10.7
G3_circuit	6.4	15.6	4.2	17.5	1.9	9.5	2.0	11.3
Hook_1498	6.4	19.7	11.4	20.0	2.5	11.3	3.5	10.9
inline_1	9.1	13.4	8.9	9.8	2.6	10.0	3.8	10.5
StocF-1465	7.7	19.2	10.5	16.9	3.0	11.3	4.7	10.5
thermal2	4.7	16.6	2.5	18.5	1.9	10.4	1.9	11.0

2025, **26** (2), 140–149. doi 10.26089/NumMet.v26r210

во второй и третьей — время нашей реализации без и с использованием библиотеки hwloc. Для всех матриц на обеих рабочих станциях NUMA, наша реализация, использующая hwloc, оказывается быстрее Intel MKL PARDISO. Без использования библиотеки hwloc выполнение стадии прямой/обратной подстановки на 7 матрицах из 10 происходит медленнее на четырехпроцессорной рабочей станции, чем у Intel MKL PARDISO.

В табл. 3 приведены результаты измерения ускорения при разных NUMA-конфигурациях для Intel MKL PARDISO (первые колонки в парах) и предлагаемой реализации (вторые колонки в парах). Отношение времени выполнения в однопоточном режиме к времени выполнения в многопоточном режиме у нашей реализации выше, чем у Intel MKL PARDISO на всех матрицах из табл. 1 на обеих рабочих станциях NUMA. По нашему мнению, лучшее масштабирование стадий численного разложения и прямой/обратной подстановки у предложенной реализации объясняется отсутствием барьеров синхронизации и меньшими накладными расходами, связанными с доступом к данным, за счет размещения объектов в локальных для исполняющих ядер блоках памяти. Также объединение супернод небольшого размера в группы позволяет уменьшить накладные расходы, связанные с необходимостью порождать в планировщике работы большое количество мелких задач. Вопрос, почему не удается получить большее ускорение на четырехпроцессорной рабочей станции по сравнению с двухпроцессорной, притом что у нее существенно больше вычислительных ядер, требует дополнительного исследования.

5. Заключение. В работе предложена модификация алгоритма разложения Холецкого для разреженных симметричных положительно определенных матриц, основанная на парадигме асинхронного выполнения. Так как выполнение стадий численного разложения и прямой/обратной подстановки представляется в виде ориентированного ациклического графа, что позволяет обходиться без барьеров синхронизации, предлагаемый алгоритм демонстрирует лучшую масштабируемость, чем коммерческая высокооптимизированная реализация Intel MKL PARDISO, подтверждая эффективность нашего подхода. Использование кроссплатформенной библиотеки шаблонов C++ TBB существенно облегчает реализацию алгоритма разложения Холецкого, основанного на подзадачах, и обеспечивает динамическую балансировку вычислений между ядрами процессоров. Предложенные оптимизации связаны с управлением размещением данных в памяти с использованием функциональности библиотек TBB и hwloc. Эти оптимизации позволяют улучшить масштабирование алгоритма для систем с NUMA-архитектурой, что особенно критично влияет на стадию прямой/обратной подстановки.

Список литературы

1. *Schenk O., Gärtner K., Fichtner W., Stricker A.* PARDISO: a high-performance serial and parallel sparse linear solver in semiconductor device simulation // Future Generation Computer Systems. 2001. **18**, N 1. 69–78. doi 10.1016/S0167-739X(00)00076-5.
2. *Trottenberg U., Oosterlee C.W., Schüller A.* Multigrid. London: Academic Press, 2001.
3. *Amestoy P.R., Duff I.S., L'Excellent J.-Y., Koster J.* A fully asynchronous multifrontal solver using distributed dynamic scheduling // SIAM Journal on Matrix Analysis and Applications. 2001. **23**, N 1. 15–41. doi 10.1137/S0895479899358194.
4. *Hénon P., Ramet P., Roman J.* PaStiX: a high-performance parallel direct solver for sparse symmetric positive definite systems // Parallel Comput. 2002. **28**, N 2. 301–321. doi 10.1016/S0167-8191(01)00141-7.
5. *Jacquelin M., Zheng Y., Ng E., Yelick K.* An asynchronous task-based fan-both sparse Cholesky solver // arXiv:1608.00044v2. 2016. doi 10.48550/arXiv.1608.00044.
6. *Ballard G., Carson E., Demmel J., et al.* Communication lower bounds and optimal algorithms for numerical linear algebra // Acta Numerica. 2014. **23**. 1–155. doi 10.1017/S0962492914000038.
7. *Bollhöfer M., Schenk O., Janalik R., et al.* State-of-the-art sparse direct solvers // Parallel Algorithms in Computational Science and Engineering. Cham: Birkhäuser, 2020. 3–33. doi 10.1007/978-3-030-43736-7_1.
8. oneAPI Threading Building Blocks (oneTBB). https://www.threadingbuildingblocks.org/.
9. *Schenk O., Gärtner K.* Two-level dynamic scheduling in PARDISO: improved scalability on shared memory multiprocessing systems // Parallel Computing. 2002. **28**, N 2. 187–197. doi 10.1016/S0167-8191(01)00135-1.
10. *Karypis G., Kumar V.* A fast and high quality multilevel scheme for partitioning irregular graphs // SIAM Journal on Scientific Computing. 1998. **20**, N 1. 359–392. doi 10.1137/S1064827595287997.

2025, **26** (2), 140–149. doi 10.26089/NumMet.v26r210

11. *Chevalier C., Pellegrini F.* PT-scotch: a tool for efficient parallel graph ordering // Parallel Computing. 2008. **34**, N 6–8. 318–331. doi 10.1016/j.parco.2007.12.001.
12. *Liu J.W.H.* The role of elimination trees in sparse factorization // SIAM Journal on Matrix Analysis and Applications. 1990. **11**, N 1. 134–172. doi 10.1137/0611010.
13. *Ashcraft C.C.* A taxonomy of column-based Cholesky factorizations. PhD Thesis in Mathematics. New Haven: Yale University, 1996. https://dl.acm.org/doi/book/10.5555/241365.
14. *Broquedis F., Clet-Ortega J., Moreaud S., et al.* hwloc: a generic framework for managing hardware affinities in HPC applications // 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing. 2010. doi 10.1109/PDP.2010.67.
15. *Davis T.A., Hu Y.* The University of Florida sparse matrix collection // ACM Transactions on Mathematical Software (TOMS). 2011. **38**, N 1, Article No 1. 1–25. doi 10.1145/2049662.2049663.

Поступила в редакцию
12 февраля 2025 г.

Принята к публикации
17 марта 2025 г.

Информация об авторах

Андрей Сергеевич Маслов — разработчик; ООО “ТС Интеграция”, Ленинградский пр-кт, д. 36, стр. 41, 125167, Москва, Российская Федерация.

Михаил Михайлович Макаров — к.ф.-м.н., руководитель отдела физико-математического моделирования; ООО “ТС Интеграция”, Ленинградский пр-кт, д. 36, стр. 41, 125167, Москва, Российская Федерация.

Николай Николаевич Потравкин — к.ф.-м.н., ведущий разработчик; ООО “ТС Интеграция”, Ленинградский пр-кт, д. 36, стр. 41, 125167, Москва, Российская Федерация.

Святослав Олегович Проскурня — ведущий разработчик; ООО “ТС Интеграция”, Ленинградский пр-кт, д. 36, стр. 41, 125167, Москва, Российская Федерация.

References

1. O. Schenk, K. Gärtner, W. Fichtner, and A. Stricker, “PARDISO: A High-Performance Serial and Parallel Sparse Linear Solver in Semiconductor Device Simulation,” Future Gener. Comput. Syst. **18** (1), 69–78 (2001). doi 10.1016/S0167-739X(00)00076-5.
2. U. Trottenberg, C. W. Oosterlee, and A. Schüller, *Multigrid* (Academic Press, London, 2001).
3. P. R. Amestoy, I. S. Duff, J.-Y. L’Excellent, and J. Koster, “A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling,” SIAM J. Matrix Anal. Appl. **23** (1), 15–41 (2001). doi 10.1137/S0895479899358194.
4. P. Hénon, P. Ramet, and J. Roman, “PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Positive Definite Systems,” Parallel Comput. **28** (2), 301–321 (2002). doi 10.1016/S0167-8191(01)00141-7.
5. M. Jacquelin, Y. Zheng, E. Ng, and K. Yelick, “An Asynchronous Task-based Fan-Both Sparse Cholesky Solver,” arXiv:1608.00044v2 (2016). doi 10.48550/arXiv.1608.00044.
6. G. Ballard, E. Carson, J. Demmel, et al., “Communication Lower Bounds and Optimal Algorithms for Numerical Linear Algebra,” Acta Numer. **23**, 1–155 (2014). doi 10.1017/S0962492914000038.
7. M. Bollhöfer, O. Schenk, R. Janalik, et al., “State-of-the-Art Sparse Direct Solvers,” in *Parallel Algorithms in Computational Science and Engineering* (Birkhäuser, Cham, 2020), pp. 3–33. doi 10.1007/978-3-030-43736-7_1.
8. oneAPI Threading Building Blocks (oneTBB). https://www.threadingbuildingblocks.org/.
9. O. Schenk and K. Gärtner, “Two-Level Dynamic Scheduling in PARDISO: Improved Scalability on Shared Memory Multiprocessing Systems,” Parallel Comput. **28** (2), 187–197 (2002). doi 10.1016/S0167-8191(01)00135-1.
10. 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). doi 10.1137/S1064827595287997.
11. C. Chevalier and F. Pellegrini, “PT-Scotch: A Tool for Efficient Parallel Graph Ordering,” Parallel Comput. **34** (6–8), 318–331 (2008). doi 10.1016/j.parco.2007.12.001.

2025, **26** (2), 140–149. doi 10.26089/NumMet.v26r210

12. J. W. H. Liu, "The Role of Elimination Trees in Sparse Factorization," SIAM J. Matrix Anal. Appl. **11** (1), 134–172 (1990). doi 10.1137/0611010.
13. C. C. Ashcraft, *A Taxonomy of Column-Based Cholesky Factorizations*, PhD Thesis in Mathematics (Yale University, New Haven, US, 1996). https://dl.acm.org/doi/book/10.5555/241365.
14. F. Broquedis, J. Clet-Ortega, S. Moreaud, et al., "hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications," *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing* (2010). doi 10.1109/PDP.2010.67.
15. T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," ACM Trans. Math. Softw. **38** (1), Article No 1, 1–25 (2011). doi 10.1145/2049662.2049663.

Received
February 12, 2025

Accepted for publication
March 17, 2025

Information about the authors

Andrei S. Maslov — Programmer; LLC "TS INTEGRATION", Leningradsky prospekt, 36, building 41, 125167, Moscow, Russia.

Michail M. Makarov — Ph. D., Head of the department of physical and mathematical modeling; LLC "TS INTEGRATION", Leningradsky prospekt, 36, building 41, 125167, Moscow, Russia.

Nikolay N. Potravkin — Ph. D., Lead Programmer; LLC "TS INTEGRATION", Leningradsky prospekt, 36, building 41, 125167, Moscow, Russia.

Svyatoslav O. Proskurnia — Lead Programmer; LLC "TS INTEGRATION", Leningradsky prospekt, 36, building 41, 125167, Moscow, Russia.