МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

Гомельский государственный технический университет имени П.О.Сухого Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

Специальность 1-40 05 01 01 Информационные системы и технологии (в проектировании и производстве)

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Распределенные информационные системы»

на тему: «Распределённое приложение для параллельного решения СЛАУ методом *LLT* разложения в кластере *MPI*»

Исполнитель: студент гр. ИТП-41

Бурлаков А.О. Руководитель: ст.преподаватель

Стефановский И.Л.

Дата проверки:

Дата допуска к защите:

Дата защиты:

Оценка работы:

Подписи членов комиссии

по защите курсового проекта:

2025

**СОДЕРЖАНИЕ**

[Введение 4](#_Toc211097964)

[1 Обзор литературы и доступных программно-технических решений 5](#_Toc211097965)

[1.1 Понятия и основные виды уравнений 5](#_Toc211097966)

[1.2 Обзор методов решения систем линейных алгебраических уравнений 8](#_Toc211097967)

[1.3 Обзор программных средств для решения задачи 10](#_Toc211097968)

# ВВЕДЕНИЕ

Решение систем линейных алгебраических уравнений (СЛАУ) является одной из фундаментальных задач вычислительной математики и находит широкое применение в различных областях науки и техники. С ростом размерности задач, возникающих в современных приложениях, традиционные последовательные методы решения СЛАУ становятся недостаточно эффективными из-за высоких временных и ресурсных затрат. Это делает актуальной разработку параллельных алгоритмов, способных эффективно использовать вычислительные ресурсы многопроцессорных систем и кластеров.

Особый интерес представляет **метод** *LLT* **разложения (разложение Холецкого)**, который позволяет эффективно решать СЛАУ с симметричными положительно определёнными матрицами коэффициентов – классом матриц, часто встречающимся в реальных задачах, например, в методе конечных элементов или в задачах оптимизации. Однако последовательная реализация этого метода плохо масштабируется на больших объёмах данных, что обуславливает необходимость его адаптации к распределённой вычислительной среде.

В условиях стремительного развития высокопроизводительных вычислений всё большее значение приобретают технологии, позволяющие эффективно распределять вычислительную нагрузку между узлами кластера. Одной из наиболее распространённых и проверенных платформ для построения таких систем является *MPI* (*Message Passing Interface*) – стандарт обмена сообщениями, обеспечивающий переносимость и масштабируемость параллельных приложений.

Актуальность данного курсового проекта обусловлена необходимостью повышения производительности решения СЛАУ за счёт использования параллельных вычислений, а также практической ценностью разработки программного инструмента, который может быть применён в учебных, научных и инженерных задачах. Разработка распределённого приложения на основе *MPI* позволит не только ускорить вычисления, но и получить опыт проектирования и отладки параллельных систем, что крайне важно для будущего специалиста в области прикладной математики и информационных технологий.

Выбор *MPI* обусловлен его широкой поддержкой, зрелостью экосистемы и возможностью запуска приложения на реальных кластерных системах или эмуляции распределённой среды на локальной машине. Полученные в ходе работы результаты позволят оценить масштабируемость предложенного подхода и его применимость к решению практических задач большой размерности.

# ОБЗОР ЛИТЕРАТУРЫ И ДОСТУПНЫХ ПРОГРАММНО-ТЕХНИЧЕСКИХ РЕШЕНИЙ

## Понятия и основные виды уравнений

В математике уравнение – это равенство, содержащее одну или несколько неизвестных величин (переменных), значения которых требуется найти. Формально уравнение записывается в виде (1.1):

|  |  |
| --- | --- |
| *F*(*x*1,*x*2,…,*xn*) = *g*(*x*1*,x*2,…,*xn*), | (1.1) |

где *f* и *g* – некоторые функции от переменных *x*1,*x*2,…,*xn*.

Решением (или корнем) уравнения называется такой набор значений переменных, при подстановке которого равенство становится тождественно верным.

На первый взгляд, это определение кажется простым и даже тривиальным. Однако за ним скрывается мощнейший инструмент описания реальных процессов. Почти любая задача, с которой сталкивается инженер, физик, экономист или программист, в конечном счёте сводится к поиску таких значений, при которых выполняется определённое условие – то есть к решению уравнения. Если же таких условий несколько и они взаимосвязаны, возникает система уравнений. И именно системы уравнений, а не отдельные уравнения, составляют основу большинства вычислительных моделей.

В рамках численных методов и вычислительной математики особое внимание уделяется системам линейных алгебраических уравнений (СЛАУ). Причина этого – их универсальность и частота появления в самых разных областях. СЛАУ возникают при численном решении дифференциальных уравнений в частных производных (например, при моделировании теплопроводности, течения жидкости или напряжённо-деформированного состояния конструкций), в задачах компьютерной томографии, при обработке сигналов, в машинном обучении (линейная регрессия), в экономике (модели межотраслевого баланса), в компьютерной графике (преобразования координат, рендеринг), а также во множестве других прикладных дисциплин. Даже если исходная задача нелинейна, часто её линеаризуют на каждом шаге итерационного процесса – и снова получают СЛАУ.

В зависимости от соотношения между числом уравнений *m* и числом неизвестных *n*, СЛАУ делятся на три основных типа:

– квадратные системы (*m*=*n*) – наиболее «симметричный» и часто встречающийся случай. Здесь количество уравнений совпадает с количеством неизвестных, что создаёт предпосылки для единственного решения. Однако наличие такого решения не гарантировано. Система может быть несовместной (противоречивой), если, например, два уравнения описывают параллельные, но не совпадающие прямые. Или, наоборот, избыточной, если одно уравнение является линейной комбинацией других – тогда решений бесконечно много. Критерий совместности и единственности даётся теоремой Кронекера–Капелли: система совместна тогда и только тогда, когда ранг матрицы коэффициентов равен рангу расширенной матрицы [*A∣b*]. А если этот ранг равен числу неизвестных, решение единственно. Квадратные системы — основная цель многих прямых методов, таких как метод Гаусса или *LU-*разложение;

– недоопределённые системы (*m*<*n)* – уравнений меньше, чем неизвестных. Это означает, что система содержит недостаточно информации для однозначного определения всех переменных. Геометрически это можно представить как пересечение нескольких плоскостей в пространстве высокой размерности — результатом будет не точка, а целая линия, плоскость или подпространство. Такие системы часто возникают в задачах восстановления данных, где известны лишь некоторые проекции или измерения, а полная картина должна быть реконструирована. В машинном обучении недоопределённые системы появляются, например, при обучении моделей с большим числом признаков по малому количеству примеров. В таких случаях говорят о «недообучении» или необходимости регуляризации. Решение, как правило, не единственно, и ставится дополнительная задача – выбрать из множества решений «наилучшее», например, с минимальной евклидовой нормой или с минимальным числом ненулевых компонент (разреженное решение);

– переопределённые системы (*m*>*n*) – уравнений больше, чем неизвестных. Это, пожалуй, самый распространённый тип в прикладных задачах. Представьте, что вы проводите серию измерений (например, координаты точки с помощью *GPS*) и хотите определить её истинное положение. Каждое измерение даёт уравнение, но из-за погрешностей ни одно из них не будет выполнено точно. Точного решения не существует, но можно найти такое значение неизвестных, при котором невязка, разница между левой и правой частями, будет минимальной в каком-то смысле.

Наиболее распространённый подход – метод наименьших квадратов (МНК), который минимизирует сумму квадратов невязок. Однако у этого подхода есть существенный недостаток: матрица *A* имеет число обусловленности, равное квадрату числа обусловленности исходной матрицы *A*, что может привести к сильной потере точности. Поэтому на практике предпочтение отдаётся методам, которые решают задачу наименьших квадратов напрямую, без формирования нормальных уравнений – например, через *QR*-разложение или сингулярное разложение (*SVD*). Именно такие методы обеспечивают высокую численную устойчивость и широко используются в инженерных и научных расчётах.

Кроме классификации по размерности, СЛАУ также различают по свойствам матрицы коэффициентов, что напрямую влияет на выбор алгоритма решения.

Одно из важнейших разделений – на полные и разреженные системы. В полных матрицах большинство элементов ненулевые. Такие системы типичны для задач, где каждая переменная влияет на почти все уравнения – например, в задачах глобальной оптимизации или при работе с плотными ядрами в интегральных уравнениях. Разреженные матрицы, напротив, содержат подавляющее большинство нулевых элементов. Это характерно для задач, построенных на локальных взаимодействиях: в методе конечных элементов каждая точка сетки связана только с соседями, в электрических цепях каждый узел соединён лишь с несколькими другими. Для разреженных систем разработаны специальные форматы хранения (*CSR*, *CSC* и др.), которые экономят память и ускоряют вычисления, а также специализированные итерационные методы, которые не «трогают» нулевые элементы.

Ещё один критически важный аспект – обусловленность системы. Даже если система имеет единственное решение, оно может быть крайне чувствительно к малейшим изменениям в исходных данных. Такие системы называются плохо обусловленными. В машинных вычислениях, где все операции выполняются с ограниченной точностью (обычно двойной — около 16 десятичных знаков), ошибки округления могут привести к тому, что полученное «решение» окажется совершенно неадекватным. Хорошо обусловленные системы, напротив, устойчивы к таким погрешностям. Число обусловленности *κ*(*A*) (для невырожденной квадратной матрицы) количественно характеризует эту чувствительность: чем оно больше, тем менее надёжно решение.

Диагонально преобладающие матрицы – те, у которых в каждой строке модуль диагонального элемента больше суммы модулей остальных элементов. Такие матрицы гарантированно невырождены, и для них хорошо работают простые итерационные методы (Якоби, Гаусс–Зейдель).

Ленточные матрицы – ненулевые элементы сосредоточены вблизи главной диагонали. Типичны для одномерных задач или сеток с упорядоченными узлами. Для них существуют модификации метода Гаусса, которые учитывают структуру и экономят ресурсы.

Важно отметить, что аналитическое решение СЛАУ возможно лишь для небольших систем. Например, формулы Крамера позволяют выразить решение через определители, но их вычислительная сложность растёт как факториал от размера — уже при *n*=10 это становится нереалистичным.

Поэтому на практике применяются численные методы, которые делятся на две большие группы. Прямые методы дают решение за конечное число арифметических операций (в рамках машинной арифметики). Они надёжны, предсказуемы и хорошо подходят для систем умеренного размера (до нескольких десятков тысяч неизвестных). К ним относятся:

– метод Гаусса с выбором главного элемента (классика численного анализа);

– *LU*-разложение (факторизация матрицы на нижнюю и верхнюю треугольные);

– *QR*-разложение (факторизация на ортогональную и верхнетреугольную матрицы);

– метод Холецкого (для симметричных положительно определённых матриц).

В данной работе основное внимание уделяется**методу *LLT*-разложения (разложению Холецкого)**, который заключается в факторизации симметричной положительно определённой матрицы *A* в произведение нижней треугольной матрицы *L* и её транспонирования: *A* = *LL*^*T*. Суть метода решения СЛАУ сводится к последовательному решению двух треугольных систем: сначала системы *L* *y* = *b* прямой подстановкой, а затем системы *L*^*T* *x* = *y* обратной подстановкой. Наиболее известные варианты включают классический алгоритм Холецкого и его устойчивую модификацию — разложение Холецкого с диагональным преобладанием (разложение *LDL*^*T*), которое исключает операцию извлечения квадратного корня. Этот подход является одним из наиболее эффективных прямых методов для данного класса матриц и широко реализован в профессиональных библиотеках (например, *LAPACK*).

Этот подход особенно полезен для систем с симметричными положительно определёнными матрицами, так как он требует в два раза меньше операций и памяти по сравнению с методами для произвольных матриц (такими как *LU* или *QR*-разложение) и гарантирует устойчивость без необходимости в выборе ведущего элемента.

Таким образом, понимание типов уравнений, их свойств и особенностей не просто академическое упражнение. Оно лежит в основе грамотного выбора алгоритма, определяет требования к памяти и времени выполнения, влияет на устойчивость вычислений и, в конечном счёте, на достоверность результата. Особенно это актуально при переходе к параллельным вычислениям: например, разреженные и плотные матрицы требуют совершенно разных стратегий распределения данных по узлам кластера, а итерационные и прямые методы по-разному используют межпроцессное взаимодействие.

## Обзор методов решения систем линейных алгебраических уравнений

Выбор подходящего метода для решения системы линейных алгебраических уравнений – это не просто технический вопрос, а важнейший этап проектирования вычислительного алгоритма. От этого выбора напрямую зависят точность результата, время выполнения, объём требуемой памяти и, что особенно актуально в современных условиях, возможность эффективного распараллеливания. В отличие от школьных задач, где система состоит из двух-трёх уравнений и решается вручную, реальные инженерные и научные задачи часто приводят к системам с миллионами неизвестных. Для таких масштабов даже небольшая разница в эффективности метода может означать разницу между решением за час и решением за неделю.

Все существующие численные методы решения СЛАУ можно условно разделить на две большие группы: прямые и итерационные. Это разделение отражает не только математическую суть подходов, но и их поведение в реальных вычислениях, особенно при работе на многопроцессорных системах.

***1.2.1*** Прямые методы основаны на последовательном преобразовании исходной системы к виду, из которого решение можно получить простым способом – например, с помощью обратной подстановки. Классическим примером является метод Гаусса, известный ещё со школьного курса, но в реальных вычислениях он почти всегда используется с дополнительными улучшениями: выбором ведущего элемента (чтобы избежать деления на малые числа и повысить устойчивость), блочной организацией данных и т.д. Более продвинутые прямые методы, такие как *LU*- или *QR*-разложение, представляют матрицу системы в виде произведения более простых матриц — треугольных, ортогональных и т.п. Это позволяет не только решать одну систему, но и эффективно обрабатывать несколько систем с одинаковой матрицей, но разными правыми частями, что часто встречается в динамических задачах.

Главное достоинство прямых методов — предсказуемость. Они дают решение за конечное, заранее известное число операций (в рамках машинной арифметики), и при корректной реализации результат будет максимально точным для данной задачи. Однако у них есть и серьёзный недостаток: вычислительная сложность, плохая масштабируемость на кластерах. Хотя существуют параллельные версии *LU*- или *QR*-разложения (например, в библиотеках *ScaLAPACK*), они требуют интенсивного обмена данными между процессами, особенно на этапах, связанных с выбором ведущего элемента или обновлением блоков. Это приводит к тому, что при увеличении числа узлов кластера выигрыш в скорости быстро насыщается, и дальнейшее добавление процессоров почти не даёт эффекта. Поэтому прямые методы редко используются для очень больших задач в распределённых системах, если только матрица не имеет специальной структуры.

***1.2.2*** Итерационные методы, напротив, строят решение постепенно. Они начинают с некоторого начального приближения и на каждом шаге уточняют его, приближаясь к истинному решению. Процесс продолжается до тех пор, пока невязка (разница между левой и правой частями) не станет меньше заданного порога. Классические итерационные методы – такие как методы Якоби или Гаусса-Зейделя – просты в реализации, но сходятся медленно и только для узкого класса матриц (например, диагонально преобладающих). Поэтому в современных приложениях они практически не используются.

***1.2.3*** На смену им пришли методы подпространств Крылова – целое семейство мощных итерационных алгоритмов, включающее метод сопряжённых градиентов (*CG),* *GMRES*, *BiCGSTAB* и другие. Эти методы гораздо быстрее, устойчивее и применимы к более широкому кругу задач. Особенно важно то, что на каждой итерации они требуют в основном умножения матрицы на вектор — операции, которая идеально подходит для распараллеливания. Если матрица разреженная (а в большинстве реальных задач она именно такая), то умножение можно выполнить очень быстро, задействовав только ненулевые элементы.

Именно поэтому итерационные методы стали стандартом де-факто для решения больших СЛАУ в высокопроизводительных вычислениях. Они экономно используют память (часто достаточно хранить только ненулевые элементы в одном из разреженных форматов), легко масштабируются на десятки и сотни узлов кластера, и позволяют гибко управлять точностью результата. Однако у них есть и свои сложности. Во-первых, сходимость не гарантирована — для некоторых матриц итерации могут зациклиться или расходиться. Во-вторых, даже при сходимости она может быть очень медленной, если система плохо обусловлена. Чтобы справиться с этой проблемой, широко применяется предобуславливание (*preconditioning*). Идея проста: вместо исходной системы *Ax*=*b* решается эквивалентная система. Хороший предобуславливатель «сглаживает» спектр матрицы, ускоряя сходимость в десятки и даже сотни раз. На практике используются такие предобуславливатели, как неполное *LU*-разложение (*ILU*), многосеточные схемы (*multigrid*), или даже упрощённые версии самой задачи. Подбор эффективного предобуславливателя – это часто больше искусство, чем наука, и требует глубокого понимания физики моделируемой задачи.

Теперь стоит отдельно остановиться на **методах, основанных на разложении Холецкого (*LLT*)**, поскольку именно они лежат в основе данного проекта. Эти методы относятся к прямым и обладают рядом свойств, которые делают их особенно ценными для определённого класса задач — в первую очередь, для систем с симметричными положительно определёнными матрицами, повсеместно возникающими в задачах оптимизации, методе конечных элементов и регрессионном анализе.

Существует несколько способов реализации разложения Холецкого: классический алгоритм (требующий извлечения квадратных корней) и его устойчивая модификация — ***LDLT*-разложение**, которое исключает эту операцию и является численно предпочтительным. Классический алгоритм проще для понимания, но менее универсален из-за требования положительной определённости. *LDLT*-разложение, хотя и вычислительно чуть дороже, обеспечивает лучшую устойчивость для более широкого круга симметричных матриц и поэтому часто используется в профессиональных библиотеках (например, в *LAPACK*). Важно отметить, что все эти подходы допускают блочную и параллельную реализацию. Например, разложение можно проводить блоками, что позволяет эффективно использовать кэш-память процессора и даёт значительный выигрыш при работе на многоядерных системах или *GPU*. В распределённой среде *MPI* матрица обычно разбивается блоками 2*D*-схемой, и каждый процессор отвечает за вычисления в своём блоке, периодически обмениваясь данными с соседями.

Подводя итог, можно сказать, что не существует «лучшего» метода решения СЛАУ вообще – есть метод, наилучший для конкретной задачи. Выбор зависит от размера системы, структуры матрицы, требуемой точности, доступных вычислительных ресурсов и, что особенно важно в контексте данной работы, от возможности эффективной параллельной реализации. Для небольших плотных систем – прямые методы. Для больших разреженных – итерационные с предобуславливанием. А для **симметричных положительно определённых систем, где критически важны вычислительная эффективность и устойчивость, – методы на основе разложения Холецкого (*LLT*/*LDLT*).**

## Обзор программных средств для решения задачи

Для реализации распределённого приложения, решающего СЛАУ методом *LLT* в кластерной среде, необходимо выбрать программные средства, которые поддерживают работу с *MPI*, линейной алгеброй и позволяют контролировать распределение данных. Рассмотрим основные доступные варианты.

***1.3.1*** *MPI* (*Message Passing Interface*) – не библиотека для решения СЛАУ, а стандарт обмена сообщениями между процессами. Он лежит в основе почти всех кластерных вычислений. Реализации *MPI* (*Open* *MPI*, *MPICH*, *Intel* *MPI)* предоставляют функции для отправки/приёма данных, коллективных операций (например, редукция, рассылка) и управления процессами. Для нашей задачи *MPI* необходим как основа – без него нельзя организовать распределённые вычисления. Однако сам по себе он не содержит алгоритмов линейной алгебры, поэтому требует дополнения.

***1.3.2*** *ScaLAPACK* – одна из немногих библиотек, предоставляющих распараллеленные прямые методы, включая *QR*-разложение. Это важно, потому что метод ортогонализации, используемый в работе, напрямую связан с *QR*. *ScaLAPACK* работает с плотными матрицами, распределяет их по процессам блочно-циклически и построена поверх *BLACS*, который использует *MPI*. Библиотека стабильна, оптимизирована и входит в состав многих *HPC*-дистрибутивов.

***1.3.3*** *Elemental* – более современная альтернатива *ScaLAPACK*. Также реализует распределённые прямые методы, включая *QR*, но использует более гибкую модель распределения данных и написана на *C++* с поддержкой шаблонов. *Elemental* лучше масштабируется и проще в использовании, чем *ScaLAPACK*, но менее распространена и может отсутствовать в стандартных пакетах кластеров. Для учебного проекта её подключение может потребовать дополнительных усилий по сборке и настройке.

***1.3.4*** *PETSc* и *Trilinos* – мощные фреймворки для научных вычислений, но они ориентированы на итерационные методы. *PETSc* предоставляет отличную поддержку разреженных матриц, предобуславливателей и масштабируется до тысяч процессов. Однако *QR*-разложение в *PETSc* отсутствует – оно не входит в набор базовых решателей. Аналогично, *Trilinos* содержит модуль *Anasazi* для задач на собственные значения и *Belos* для итерационных решателей, но прямые методы ортогонализации реализованы слабо или требуют подключения внешних библиотек (например, *SuiteSparseQR*). Поэтому эти инструменты не подходят для задачи, где ключевой метод — ортогонализация.

***1.3.5*** *LAPACK* и *Eigen* – популярные библиотеки линейной алгебры, но они не поддерживают распределённые вычисления. *LAPACK* работает только на одном узле и не знает о *MPI*. *Eigen* – шаблонная *C++* библиотека, удобная для прототипирования, но тоже однопоточная по умолчанию (есть поддержка *OpenMP*, но не *MPI*). Их можно использовать для последовательной версии алгоритма или для тестирования, но не для распределённой реализации.

***1.3.6*** *Intel MKL (Math Kernel Library*) – коммерческая библиотека, содержащая высокопроизводительные реализации *BLAS*, *LAPACK*, а также *Cluster* *Sparse* *Solver* и *ScaLAPACK*-совместимый интерфейс. Если кластер оснащён процессорами *Intel* и лицензией *MKL*, можно использовать её версию *ScaLAPACK* с ускорением через оптимизированные ядра. Однако в учебной среде доступ к *MKL* часто ограничен, а сама библиотека привязана к экосистеме *Intel*.

***1.3.7*** *Open*-*source* альтернативы без *MPI* – такие как *Armadillo*, *Blaze* или *GSL* – также не подходят, так как не поддерживают распределённую память. Они полезны для локальных расчётов, но не для кластера.

***1.3.8*** *C* – исторически основной язык высокопроизводительных вычислений. Все реализации *MPI* (*Open* *MPI*, *MPICH*, *Intel* *MPI*) изначально разрабатывались под *C*, и их *C-*интерфейс остаётся самым стабильным и быстрым. *C* предоставляет полный контроль над памятью, позволяет эффективно работать с массивами и напрямую вызывать функции *BLAS*/*LAPACK* без накладных расходов. Для задач, где критичны скорость и предсказуемость (например, ручная реализация ортогонализации с точным управлением распределением данных), *C* – оптимальный выбор. Недостаток – отсутствие встроенных средств защиты от ошибок (выход за границы массива, утечки памяти), что повышает требования к аккуратности программиста.

***1.3.9*** *C++* – логичное развитие *C*, сочетающее низкоуровневый контроль с возможностями объектно-ориентированного и обобщённого программирования. Современные версии *C*++ (17/20) позволяют писать более безопасный и читаемый код, не теряя в производительности. *MPI* поддерживает *C*++. *C*++ удобен при использовании библиотек вроде *Eigen* (для локальных операций) или *Trilinos*, но для чистой *MPI*-реализации преимущества перед *C* невелики. Тем не менее, если планируется расширение проекта (например, добавление абстракций для матриц или решателей), *C*++ может оказаться удобнее.

***1.3.10*** Несмотря на наличие обёрток *MPI* (*mpi4py*) и библиотек линейной алгебры (*NumPy*, *SciPy*), *Python* слишком медленен для реализации ядра численного метода, особенно при ручной ортогонализации с частыми обменами данными. Он может использоваться для прототипирования или визуализации, но не для производительного распределённого вычисления.

Таким образом, для реализации метода *LLT* в *MPI*-кластере *C* остаётся наиболее рациональным выбором: он обеспечивает максимальную производительность, совместимость и контроль, что критично при работе с низкоуровневыми операциями линейной алгебры и распределённой памятью.