МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки / специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль)/специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка эффективных алгоритмов матричного умножения

Обучающийся М.И. Асанова
(Инициалы Фамилия) (личная подпись)

Руководитель канд. физ.-мат. наук О.В. Лелонд
(ученая степень (при наличии), ученое звание (при наличии)), Инициалы Фамилия)

Консультант Е.В. Косс
(ученая степень (при наличии), ученое звание (при наличии)), Инициалы Фамилия)

Тольятти 2022

Аннотация

Разработка эффективных алгоритмов матричного умножения. Бакалаврская работа. Тольятти. Тольяттинский государственный университет, 2022.

В работе представлена разработка эффективных алгоритмов матричного умножения.

Ключевые слова: алгоритмы, матричное умножение, многопоточное программирование, вычислительные ресурсы, компьютерный эксперимент.

Целью выпускной квалификационной работы является разработка эффективных алгоритмов матричного умножения.

Предметом исследования является быстродействие алгоритмов и программ матричного умножения.

Объектом исследования данной работы являются алгоритмы и программы матричного умножения.

В работе проведен анализ существующих решений, исследовано современное состояние вопроса, разработаны алгоритмы и программы повышения эффективности умножения матриц, исследована эффективность разработанных программ путем компьютерного эксперимента.

Выпускная квалификационная работа состоит из 52 страниц, 14 рисунков, 12 таблиц, 23 источников и 4 приложений.

Abstract

Development of efficient algorithms for matrix multiplication. Bachelor's thesis. Togliatti. Togliatti State University, 2022.

This paper presents the development of efficient algorithms for matrix multiplication.

Key words: algorithms, matrix multiplication, multi-threaded programming, computing resources, computer experiment.

The aim of the final qualification work is the development of efficient algorithms for matrix multiplication.

The subject of the research is the performance of algorithms and programs of matrix multiplication.

Algorithms and programs of matrix multiplication are the object of research of this work.

In the work the analysis of existing solutions, studied the current state of the issue, developed algorithms and programs to improve the efficiency of matrix multiplication, examined the effectiveness of developed programs through computer experiment.

Graduate qualification work consists of 47 pages, 14 figures, 12 tables, 23 sources and 4 appendices.

Содержание

Введение

Актуальность исследуемой темы обуславливается тем, что умножение матриц — это один из базовых алгоритмов, который широко применяется в различных численных методах, и в частности в алгоритмах машинного обучения. Многие реализации прямого и обратного распространения сигнала в сверхточных слоях нейронной сети базируются на этой операции. Так порой до 90-95% всего времени, затрачиваемого на машинное обучение, приходится именно на эту операцию.

Матрицы находят много применений в научных областях, а также используются при решении практических проблем реальной жизни, тем самым решая значительную часть практических задач.

Одним из наиболее важных применений матриц в компьютерных приложениях является шифрование кодов сообщений. Сообщение выглядит как последовательность чисел в двоичном формате. Они используются для построения графиков, статистических данных, а также для исследований в самых различных областях.

Во многих организациях для записи данных экспериментов; в робототехнике и автоматизации матрицы являются базовыми элементами движений роботов. Движения роботов программируются с вычислением строк и столбцов матриц. Данные для управляющих роботов приведены на основе вычислений из матриц.

Матричное умножение — один из немногих алгоритмов, которые позволяет эффективно задействовать все вычислительные ресурсы современных процессоров и графических ускорителей. Поэтому многие алгоритмы стараются свести к матричному умножению — дополнительная расходы, связанные с подготовкой данных, как правило с лихвой окупаются общим ускорением алгоритмов.

Объектом исследования данной работы являются алгоритмы и программы матричного умножения.

Предметом исследования является быстродействие алгоритмов и программ матричного умножения.

Цель исследования – разработать эффективные алгоритмы матричного умножения.

Для достижения установленной задачи необходимо решить следующие задачи:

– Исследовать современное состояние вопроса.

– Разработать алгоритмы и программы повышения эффективности умножения матриц.

– Исследовать эффективность разработанных программ путем компьютерного эксперимента.

В работе использованы теоретические методы исследования, методы сравнительного анализа.

Практическая значимость состоит в том, что разработан алгоритм эффективного умножения матриц.

Бакалаврская работа состоит из введения, трех разделов, заключения списка использованной литературы и источников, приложения.

Первый раздел работы посвящен развитию алгоритмов матричного умножения, анализу существующих параллельных технологий, а также увеличению эффективности использования вычислительных ресурсов в одном потоке.

Во втором разделе описаны алгоритмы и программы, реализующих различные подходы к ускорению умножения матриц и их тестирование.

Третий раздел посвящен экспериментальному исследованию влияния размера матрицы на ускорение вычислений. Получены экспериментальные зависимости изменения коэффициента ускорения для различных компьютеров. Проведен анализ результатов и предложены пути совершенствования.

Термины и определения. Перечень сокращений и обозначений

FMA (Fused Multiply-Add, умножение-сложение с однократным округлением) — это набор опциональных 128- и 256-битных SIMD-инструкций для архитектур x86 и x86-64, предназначенный для выполнения операции умножения-сложения над числами в формате с плавающей запятой.

AVX (Advanced Vector Extensions) — расширение системы команд x86 для микропроцессоров Intel и AMD реализующее новые инструкции и новую схему кодирования машинных кодов.

AVX2 (Advanced Vector Extensions 2) - расширение системы команд процессора, разработанное компанией Intel в дополнение к набору инструкций AVX.

OpenMP (Open Multi-Processing) — открытый стандарт для распараллеливания программ на языках Си, Си++ и Фортран.

CUDA (Compute Unified Device Architecture) – программно-аппаратная архитектура параллельных вычислений с использованием графического процессора.

MPI (Message Passing Interface) - библиотека функций, предназначенная для поддержки работы параллельных процессов в терминах передачи сообщений.

1 Теоретическое обоснование задачи

1.1 Развитие алгоритмов матричного умножения

Традиционно считается, что n^3 - это самый быстрый способ умножения матриц, что невозможно умножить матрицы 2*2, используя менее восьми умножений.

С точки зрения необходимого количества шагов «вторая степень» — это идеальная скорость выполнения матричного умножения. то есть к умножению пары матриц n x n всего за n^2 шагов. Если вторая степень достижима, то матричное умножение получится выполнять максимально быстро, насколько это физически возможно. Матрицы представляют собой массивы чисел. Когда две матрицы согласованы (число столбцов в первом сомножителе равно числу строк во втором), их можно перемножить, чтобы получить третью. Например, 2 матрицы 2 x 2, их произведение также будет матрицей 2 x 2, содержащей четыре элемента. В более общем смысле, произведение пары матриц размером n x n представляет собой другую матрицу размером n x n с n^2 элементами. Поэтому наименьшее возможное количество шагов для умножения пар матриц это n^2, то есть количество шагов, необходимое просто для записи ответа. Отсюда и название «вторая степень» [17].

В 1969 году Фолькер Штрассен сосредоточил свои усилия на анализе сложности алгоритмов и разработке быстрых алгоритмов. И нашел способ сделать это с помощью семи умножений. Штрассен придумал сложный набор соотношений, которые позволили заменить одно из этих восьми умножений 14 дополнительными сложениями. Может показаться, что разница совершенно незначительна, но она оправдывает себя, так как умножение вносит больший вклад, чем сложение. Найдя способ избавиться от одного умножения для маленьких матриц 2*2, Штрассен используя рекурсию, предложил быстрый алгоритм Штрассена для умножения больших матриц.

Это первый алгоритм, который позволяет перемножать большие матрицы за время меньше, чем $O(n^3)$ [23].

Путем многократного разбиения больших матриц на более мелкие и с помощью его метода можно сокращать количество шагов на каждом этапе. В целом алгоритм Штрассена увеличил скорость умножения матриц с n^3 до $n^{2.81}$ мультипликативных шагов, что даёт выигрыш на больших плотных матрицах начиная, примерно, от 64×64 [21].

Поставленная Штрассеном проблема быстрого умножения матриц по сей день не решена ни в теоретическом, ни в практическом плане [19].

В конце 1970-х, когда появился принципиально новый подход к решению этой задачи. Он подразумевает перевод матричного умножения в другую вычислительную задачу линейной алгебры с использованием объектов, называемых тензорами. Тензоры, используемые в этой задаче, представляют собой трехмерные массивы чисел, состоящие из множества различных частей, каждая из которых выглядит как небольшая задача на умножение матриц.

Умножение матриц и эта задача, связанная с тензорами, в определенном смысле эквивалентны друг другу, но для решения последней имеются более быстрые процедуры. Таким образом, встала задача определить матрицы какого размера можно перемножить при тех же вычислительных затратах, которые требуются для решения тензорной задачи?

В 1981 году Арнольд Шёнхаге использовал этот подход, чтобы доказать, что умножение матриц возможно выполнить за $n^{2.522}$ шагов. Позднее Штрассен назвал этот подход «лазерным методом» (laser method). Лазерный метод считает для значений N с меньшими ошибками округления, чем при использовании метода Штрассена с использованием какого-либо дифференциального уравнения или численного метода [18].

За последние несколько десятилетий каждое улучшение в процессе умножения матриц происходило за счет усовершенствования лазерного

метода, поскольку исследователи находили все более эффективные способы трансформации задачи.

В 1990 году самый быстрый из известных алгоритмов умножения матриц, созданный Копперсмитом и Виноградом, выполнялся за время $O(n^{2,3755})$.

В 2012 году Вирджиния Василевска Уильямс из Массачусетского технологического института рамках лазерного метода получила ($n^{2.372873}$) [20].

А в 2014 году Франсуа Ле Галлю ($n^{2.3728639}$). Этот результат получен путем анализа более высоких тензорных мощностей определенного тождества Копперсмита и Винограда.

В октябре 2020 года Вирджиния Василевска Уильямс и Джош Алман из Гарвардского университета опубликовали статью «A Refined Laser Method and Faster Matrix Multiplication», где они описали самый быстрый на настоящий момент способ перемножения двух матриц за $n^{2.3728596}$ шагов. Однако этот алгоритм глактического масштаба, то есть только для данных галактического размера, поскольку содержит огромные константы и не может быть реализован на практике. Она также знаменует и конец эпохи для метода, который ученые применяли для исследований на протяжении десятилетий [22].

Развитие алгоритмов матричного умножения представлено в таблице 1.

Таблица 1 – Развитие алгоритмов матричного умножения

Год	Автор	Скорость, m ($O(n^m)$)	Примечание
1	2	3	4
-	Тривиальный метод	3	Скорость матричного умножения при тривиальном подходе
1969	Фолькер Штрассен	2,81	Для матриц 2*2 предложен набор соотношений, которые позволили заменить одно из восьми умножений 14 дополнительными сложениями.

Продолжение таблицы 1

1	2	3	4
1981	Арнольд Шёнхаге	2,522	Применение тензоров при матричном умножении (laser method)
1990	Копперсмит и Виноград	2,3755	Усовершенствованный laser method
2012	Вирджиния Василевска Уильямс	2.372873	Усовершенствованный laser method
2014	Франсуа Ле Галлю	2.3728639	Усовершенствованный laser method
-	-	2	Теоретический предел скорости матричного умножения

В результате анализа современных исследований в области теории алгоритмов матричного умножения. Тривиальный алгоритм имеет сложность $O(n^3)$. Теоретический предел скорости умножения матриц $O(n^2)$. Снижение степени n в большинстве случаев связано с использованием тензоров. По изменению производительности видно, что совершенствование в рамках тензорного матричного умножения не ведет к достижению теоретического предела скорости и ограничено n=2.37. Принципиально новых теоретических подходов к организации матричного умножения в настоящее время не предложено.

Далее рассмотрим практические пути повышения эффективности матричного умножения. Они связаны с использованием многопточности и с совершенствованием вычислительных алгоритмов, направленных на более полное использование ресурсов компьютера в рамках одного потока.

1.2 Многопоточные методы повышения скорости расчета

Параллельные вычисления обладают значительным потенциалом в увеличении скорости умножения матриц [3], [14].

OpenMP – был разработан в 1997 году. Технология применяется для создания параллельных программ на суперкомпьютерах и на домашних персональных компьютерах. В разработку стандартов OpenMP включены много крупных производителей вычислительной техники и программного обеспечения: ASCI Program of the US DOE, Compaq Computer Corporation, EPCC, Hewlett-Packard Company, Intel Corporation, IMP, Sun Microsystems, NEC, Silicon Graphics, cOMPunity.

Этот стандарт может использоваться в программах на C++/C и Fortran.

На сегодняшний день технология OpenMP считается одной из самых популярных и известных технологий.

Технология использует модель распараллеливание – «ветвление-слияние». Её смысл заключается в том, что основной поток порождает несколько других параллельных потоков, какое-то время они выполняются, а потом сливаются в один единый поток. Также существует возможность вложенной параллельности, в то время, когда один из образовавшихся потоков, образовывает еще потоки и становится основным для них. Программист может контролировать число потоков в программе с помощью вызова определенных функций.

OMP имеет следующие директивы для работы с параллельным программированием:

– shared - определяет список переменных, которые обязаны быть общими для всех ветвей параллельной области (директива для работы с данными);

– parallel –с ее помощью создается несколько параллельных нитей (директива создания потоков), количество которых мы можем задать с помощью такой функции, как omp_set_num_threads() или мы можем установить необходимое число в переменной окружения OMP_NUM_THREADS;

– single – с ее помощью есть возможность выделить определенный участок кода в параллельной области, который необходимо выполнить лишь один единственный раз;

– at – с ее помощью указывается область памяти, которая будет обновляться атомарно;

– master – с ее помощью указывается участок кода, который нужно выполнить только главным процессом, а остальными он должен проигнорироваться;

– Critical, Barrier, Atomic – директивы синхронизации потоков

– for – директива для распределения задач между потоками, используется для распараллеливания итераций в циклах, чтобы они выполнялись независимо.

OpenMP можно считать высокоуровневой технологией, рассмотрим основные преимущества, которые дает программисту эта технология:

– хорошо подходит для программ с большим количеством циклов, так как для распараллеливания программы достаточно добавить директивы перед началом цикла;

– дает большие возможности контроля своей программы и возможность быстро менять алгоритм распараллеливания;

– дает возможность параллельной программе работать на однопроцессорном компьютере, директивы просто будут игнорироваться;

– поддержка orphan (оторванных) директив.

– поддерживается большинством компиляторов.

К сожалению, есть и недостатки, OpenMP рассчитан для использования на системах с общей памятью и взаимодействия потоков осуществляется через память, которая является общей, а не посредством обмена сообщениями. Также технология довольно сложна в освоении и в ней не самые удобные средства для поиска ошибок.

С появлением стандарта C++11 стандартизировалась поддержка многопоточного программирования. Была добавлена библиотека для реализации многопоточности, добавлен класс std::thread, а вместе с ним возможность создавать объект, который создаётся отдельным потоком выполнения [8], [16].

Процесс, который создает новый поток, имеет возможность подождать его завершения с помощью вызова thread.join(), либо же не ждать окончания и вызвать thread.detach().

Во избежание состояния гонки (конкуренции) существует класс mutex, у которого есть такие методы как lock() и unlock().

Данные методы разрешают создавать критические области в ситуациях, два или же более потоков в одно время пытаются редактировать и считывать одни и те же данные.

Также существует средство создания отложенных задач - future. Когда существует потребность в получении значения, которое вычисляется в отдельном потоке, вызов future.get(), позволит потоку дождаться выполнения задачи и возвращения итогового результата.

В случае необходимости осуществить синхронизацию работы процессов, применяется condition_variable, который позволяет процессу приостановиться и ожидать сигнал от другого процесса, благодаря методу wait(). Если же нам необходимо разблокировать и оповестить процесс, то нужно выполнить contition_variable.notify_one().

У данной технологии имеются ряд возможностей для разработки программ:

– работа с атомарными типами, блокировками, мьютексами;

– работа с условными переменными для приостановки потоков до достижения какой-либо задачи;

– работа с асинхронными задачами и внутренние элементы потоков. [10].

В качестве преимущества технологии можно считать возможность создания потоков только там, где это необходимо, что очень сильно увеличивает скорость и производительность программы.

Минусом же можно выделить возможность возникновения ошибок из-за состояния гонки (англ. race condition). Такие ошибки очень сложно выявить или исправить, что может создавать трудности и привести к потере времени во время разработки.

Технология Message Passing Interface (MPI) – это стандарт для обеспечения связи между ветвями параллельно приложения. MPI (message passing interface) переводится как взаимодействие через передачу сообщений. До появления стандарта MPI каждый разработчик параллельных вычислительных машин разрабатывал свою собственную библиотеку для создания параллельных программ. Основная цель стандарта MPI добиться независимости от архитектуры вычислительной машины.

MPI дает возможность программисту создавать параллельные программы независимо от архитектуры. Программы на MPI легко приносятся с одной платформы на другую. В MPI входит библиотека и загрузчик исполняемых файлов. При запуске программы все процессы переходят в общую для всех область. Также новые области могут создаваться на основе существующей области.

Важное преимущество MPI – это высока эффективность. При использовании параллельной программы пользователь может не догадываться, что используется параллельная программы. Высокая скорость вычислений параллельных программ с использованием MPI достигается за счет невыполнения пересылки большого количества информации вместе с сообщением. В MPI эффективно организован процесс вычисления и коммуникация.

Основными положительными особенностями являются:

– помогает создавать программы, которые способны работать на различных компьютерных системах;

– существует множество реализаций стандарта MPI для различных вычислительных систем, позволяющих полностью использовать преимущества оборудования для повышения эффективности параллельных вычислений;

– создано немало библиотек, упрощающих написание параллельных программ с использованием этого стандарта;

– позволяет создавать программы, которые в будущем можно легко адаптировать под новые условия.

Недостатки MPI:

– на обмен данными и синхронизации необходимы относительно высокие затраты;

– возможностей размещения по процессорам процессов не предоставляет;

– невозможно распараллеливание кода по участкам.

Основными элементами в MPI являются процессы, которые параллельно друг другу выполняют вычисления. У каждого процесса есть свой идентификационный номер (ID). В MPI, помимо процессов, используются следующие сущности: виртуальные топологии, коммуникаторы, группы процессов, кэширование атрибутов.

Из достоинств MPI – решается проблема переноса параллельных программ между различными системами (компьютерами). Так же данная система упрощает процесс написания параллельных программ при помощи библиотек, написанных с использованием самого MPI. [9]

В современном мире многопоточному программированию уделяется большое внимание в силу распространения систем с все большим количеством потоков у процессора. Однако, нельзя забывать и про развитие графических процессоров, поскольку, в отличие от центральных процессоров, количество потоков в графическом процессоре выросло в десятки (иногда сотни) раз, впоследствии чего стало рентабельным использовать мощность графических процессоров для вычислений сложных математических и программных задач,

освобождая тем самым центральный процессор, тем более что скорость выполнения у графического процессора превышает скорость выполнения у центрального процессора. Соответственно указанным выше тенденциям, разработка технологий для реализации многопоточного программирования с использованием графического процессора является все более перспективной, поэтому большинство современных технологий старается реализовать свою технологию с применением графического процессора, помимо центрального.

Сравнительные характеристики приведены в таблице 2.

Таблица 2 - Сравнительные характеристики многопоточных технологий

Название	Год разработки	Достоинства	Недостатки	Характеристика
1	2	3	4	**5**
OpenMP (Open Multi-Processing)	1997	1. Хорошо подходит для программ с большим количеством циклов, так как для распараллеливания программы достаточно добавить директивы перед началом цикла; 2. Дает большие возможности контроля своей программы и возможность быстро менять алгоритм распараллеливания; 3. Дает возможность параллельной программе работать на однопроцес-сорном компьютере, директивы будут игнорироваться; 4. Так же одним из достоинств является поддержка orphan директив; 5. Поддерживается большинством компиляторов.	OpenMP рассчитан для использования на системах с общей памятью и взаимодействия потоков осуществляется через память, которая является общей, а не посредством обмена сообщениями. Также технология довольно сложна в освоении и в ней не самые удобные средства для поиска ошибок.	Технология использует модель распараллеливание – «ветвление-слияние». Её смысл заключается в том, что основной поток порождает несколько других параллельных потоков, какое-то время они выполняются, а потом сливаются в один единый поток.

Продолжение таблицы 2

1	2	3	4	5
C++11 Parallel Technologies	2011	Возможность создания потоков только там, где это необходимо, что очень сильно увеличивает скорость и производительность программы.	Возможность возникновения ошибок из-за состояния гонки (англ. race condition). Эти ошибки очень сложно выявить или исправить, что может создавать трудности и привести к потере времени во время разработки.	У данной технологии имеются ряд возможностей для разработки программ: 1. работа с атомарными типами, блокировками, мьютексами; 2. работа с условными переменными для приостановки потоков до достижения какой-либо задачи; 3. работа с асинхронными задачами и внутренние элементы потоков.
Message Passing Interface (MPI).	1994	1. Помогает создавать программы, которые способны работать на различных компьютерных системах; 2. Существует множество реализаций стандарта MPI для различных вычислительных систем, позволяющих полностью использовать преиму-щества оборудования для повышения эффектив-ности параллельных вычислений; 3. Создано немало библиотек, упрощающих написание параллельных программ с использова-нием этого стандарта; позволяет создавать программы, которые в будущем можно легко адаптировать под новые условия.	1. На обмен данными и синхронизации необходимы относительно высокие затраты; 2. Возможностей размещения по процессорам процессов не предоставляет; 3. Невозможно распараллеливание кода по участкам.	MPI – это стандарт для обеспечения связи между ветвями параллельно приложения. MPI (message passing interface) переводится как взаимодействие через передачу сообщений.

Продолжение таблицы 2

1	2	3	4	5
CUDA Compute Unified Device Architecture	2007	1. Интерфейс программирования основан на стандартном языке программирования Си, что упрощает процесс изучения и внедрения архитектуры CUDA; 2. Разделяемая между потоками память размером в 16 Кб, которая может быть использована для организации кэша с широкой полосой пропускания, по сравнению с текстурными выборками; 3. Более эффективная передача данных между системной и видеопамятью; 4. Отсутствие необходимости в графических API с избыточностью и накладными расходами; 5. Линейная адресация памяти, и gather, и scatter, возможность записи по произвольным адресам; Аппаратная поддержка целочисленных и битовых операций.	1. Отсутствие поддержки рекурсии для выполняемых функций; 2. Минимальная ширина блока в 32 потока; 3. Закрытая архитектура CUDA, принадлежащая NVIDIA.	Платформа параллельных вычислений и интерфейс прикладного программирования (API), позволяющая использовать графические процессоры фирмы Nvidia.

На текущий момент есть огромное множество технологий для реализации многопоточного программирования, например, CUDA (Compute Unified Device Architecture) позволяет разработчикам ПО и инженерам задействовать в своих параллельных программах графический процессор с поддержкой CUDA. Платформа CUDA предоставляет прямой доступ к набору инструкция графического процессора. CUDA предназначен для таких языков программирования как C, C++ и Fortran.

1.3 Эффективное использование вычислительных ресурсов

Матричное умножение — один из немногих алгоритмов, которые позволяет эффективно задействовать все вычислительные ресурсы современных процессоров и графических ускорителей. Поэтому не удивительно, что многие алгоритмы стараются свести к матричному умножению — дополнительная расходы, связанные с подготовкой данных, как правило с лихвой окупаются общим ускорением алгоритмов [1], [2].

Тривиальное умножение матриц описывается так:

$$C[i,j] = a*C[i,j] + b*Sum(A[i,k]*B[k,j]), \quad (1)$$

где матрица A имеет размер M x K, матрица B — K x N, и матрица C — M x N.

Учитываем, что a = 0 и b = 1.

Тривиальное умножение низко производительно, далеко от теоретического предела однопоточной производительности для процессора.

Для повышения скорости: вычисление адресов элементов массивов можно упростить — вынести постоянную часть из внутреннего цикла.

В оригинальной версии доступ к элементам массива B производится не последовательно. Его можно упорядочить, если поменять порядок вычисления таким образом, чтобы внутренним циклом был последовательный обход по строчкам для всех трех матриц [7].

Дальнейшее повышение скорости можно обеспечить за счет вычисления блоками (векторами) [9]. Практически все современные процессоры позволяют проводить вычисления над такими векторами. В частности, набор инструкций AVX оперирует с векторами размерностью 256 бит. Что позволяет выполнить 8 операций для вещественных чисел с одинарной точностью за такт. AVX2/FMA делает еще один шаг вперед — он позволяет выполнить слитную операцию умножения и сложения (d = a*b + c) над вектором.

Настольные процессоры Интел начиная с 4-го поколения имеют 2 256-bit FMA модуля, что позволяет им теоретически выполнять 2*2*8 = 32 операции (float-32) за такт.

Тут нужно учитывать, два фактора:

- современные компиляторы хорошо справляются с задачей автовекторизации простых циклов, поэтому ручная оптимизация может не дать практически никаких преимуществ.

- скорость расчетов в данном случае упирается не в вычислительные возможности процессора, а в скорость загрузки и выгрузки данных. В данном случае процессору для задействования 2 256-bit FMA блоков требуется загрузить 4 и выгрузить 2 256-bit вектора за такт. Это в два раза превышает даже пропускную способность L1 кеша процессора (512/256 bit), не говоря уже о пропускной способности памяти, которая еще на порядок меньше (64-bit на канал)).

Таким образом, основная проблема в ограниченной пропускной способности памяти в современных процессорах. Процессор фактически простаивает 90% времени, ожидая, когда данные загрузятся и сохранятся в памяти.

Дальнейшие шаги по оптимизации алгоритма направлены на минимизацию доступа в память.

Больше всего загрузок и выгрузок происходит с результирующей матрицей C: данные из нее нужно загрузить, прибавить к ним произведение C[i][j] += A[i][k]*B[k][j], а потом сохранить. И так много раз. Наиболее быстрая память, с которой может работать процессор — это его собственные регистры. Если хранить результирующее значение матрицы C в регистре процессора, то в процессе расчета нужно будет подгружать только значение матриц A и B. Теперь на 1 FMA операцию приходится только 2 загрузки, до этого на 1 FMA операцию приходилось 2 загрузки и 1 выгрузка.

Если хранить в регистрах значения двух соседних столбцов матрицы C[i][j] и C[i][j+1], то можно повторно использовать загруженное значение

матрицы A[i][k]. И тогда на 1 FMA операцию потребуется только 1.5 загрузки. Кроме того, сохраняя результат в 2 независимых регистрах, можно позволить процессору выполнять 2 FMA операции за такт. Аналогично можно хранить в регистрах значения двух соседних строк — тогда будет осуществляться экономия на загрузке значений матрицы B.

Настольные процессоры Интел начиная с 2-го поколения имеют 16 256-bit векторных регистров (справедливо для 64-bit режима процессора). 12 из них можно использовать для хранения кусочка результирующей матрицы C размером 6x16. В итоге можно выполнить 12*8 = 96 FMA операций загрузив из памяти только 16 + 6 = 22 значений.

Функция, которая осуществляет вычисление такого маленького кусочка матрицы C, обычно называется микроядром.

Микроядро за каждую итерацию загружает два 256-bit вектора из матрицы B. Причем каждый раз из новой строчки. Это делает невозможным для процессора эффективное кеширование этих данных. Для исправления этой ситуации можно сделать два изменения: скопировать данные матрицы B во временный буфер так, чтобы данные, необходимые одному микроядру лежали рядом и изменить порядок обхода матрицы C: сначала по столбцам и только потом по строкам. Это позволит эффективнее использовать переупорядоченные значения матрицы B.

Дальнейшее совершенствование алгоритма следует направить на независимость от размера матриц.

При больших значениях K величина буфера превосходит размером кэш процессора. Решением проблемы будет ограничение его величины до размера кэша данных L1. Для процессоров Интел размер кэша данных L1 составляет 32 kb. С ограничением размера буфера, микроядро будет пробегать не по всем значениям K, а только по диапазону, который помещается в L1 кэш. Результаты промежуточных расчетов матрицы C будут храниться в основной памяти. Размер буфера для переупорядоченной матрицы A ограничивается

размером L2 кэша процессора (он обычно составляет от 256 до 1024 kb для разных типов процессоров).

В процессорах помимо кэша L1 и L2 еще часто бывает кэш L3 (обычно его размер составляет 1-2 MB на ядро). Его тоже можно задействовать, например, для хранения переупорядоченных значений матриц B.

По результатам анализа способов ускорения матричного умножения установлено, что существующие способы предусматривают непосредственный учет особенностей процессорных архитектур с соответствующим делением умножаемых матриц на более мелкие части, размещаемые в кэше ядра соответствующего уровня. Это выполняется с использованием программных микроядер. Другой способ, предусматривающий вынесение постоянных элементов из внутреннего цикла, является более универсальным и может быть реализован на различных конфигурациях вычислительных устройств [13].

Выводы по разделу:

– установлено, что умножение матриц является актуальной задачей, развитие и исследование, которой идет и в настоящее время.

– многопоточное программирование является современным, востребованным и эффективным средством повышения скорости вычислений. По результатам анализа существующих параллельных технологий определено, что в работе наиболее целесообразно применение параллельной технологии OMP.

– повышение скорости умножения матриц возможно за счет увеличения эффективности использования вычислительных ресурсов в одном потоке. Приемы повышения скорости расчета часто связаны с формированием программного микроядра, которое дробит умножаемые матрицы в соответствии с размерами кэшей процессора. Наряду с этим, существуют приемы, позволяющие повысить эффективность внутри потока и не зависящие от архитектуры процессора, одним из которых является вынесение постоянных действий из внутреннего цикла.

2 Разработка алгоритмов и программ эффективного умножения матриц

2.1 Базовый вариант матричного умножения

В качестве базового варианта (вариант 0) использовано тривиальное умножение матриц, схема которого показана на рисунке 1.

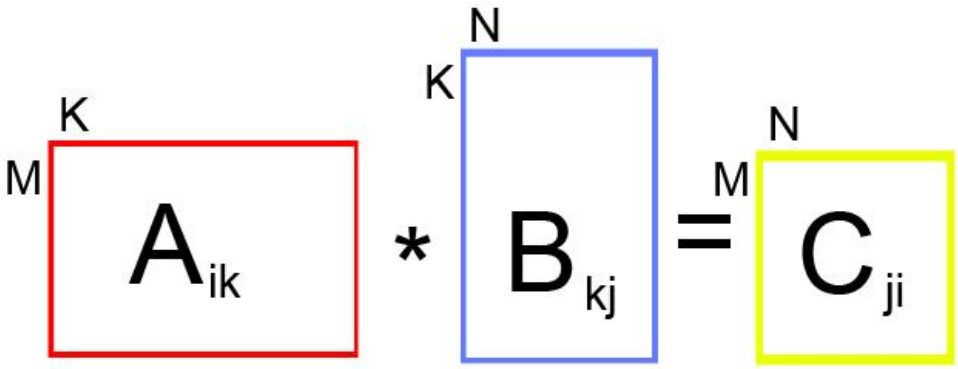


Рисунок 1 - Матричное умножение

Умножение матрицы A размера m × k и матрицы B размера k×n приводит к получению матрицы C размера m×n, каждый элемент которой определяется в соответствии с выражением:

$$C_j^i = \sum_{k=0}^{n-1} A_{ik} B_{kj,}\ 0 \le i < m, 0 \le j < n \qquad (2)$$

Как следует из (2), каждый элемент результирующей матрицы C есть скалярное произведение соответствующих строки матрицы A и столбца матрицы B [5]. [10].

Этот алгоритм предполагает выполнение m×k×n операций умножения и столько же операций сложения элементов исходных матриц [11], [12]. При умножении квадратных матриц размера n×n количество выполненных

операций имеет порядок $O(n^3)$. Будем предполагать далее, что все матрицы являются квадратными и имеют размер n× n [4], [6], [15].

Разработка программ проводилась в среде IDE Code::Blocks : 20.03-r11983 на языке C++, компилятор GNU GCC Compiler с флагами -static -lgomp –fopenmp, операционная система Windows 10 64bit.

Блок-схема тривиального умножения матрицы показана на рисунке 2 содержит 3 вложенных цикла. В двух внешних циклах выполняется перебор матриц по срокам и столбцам.

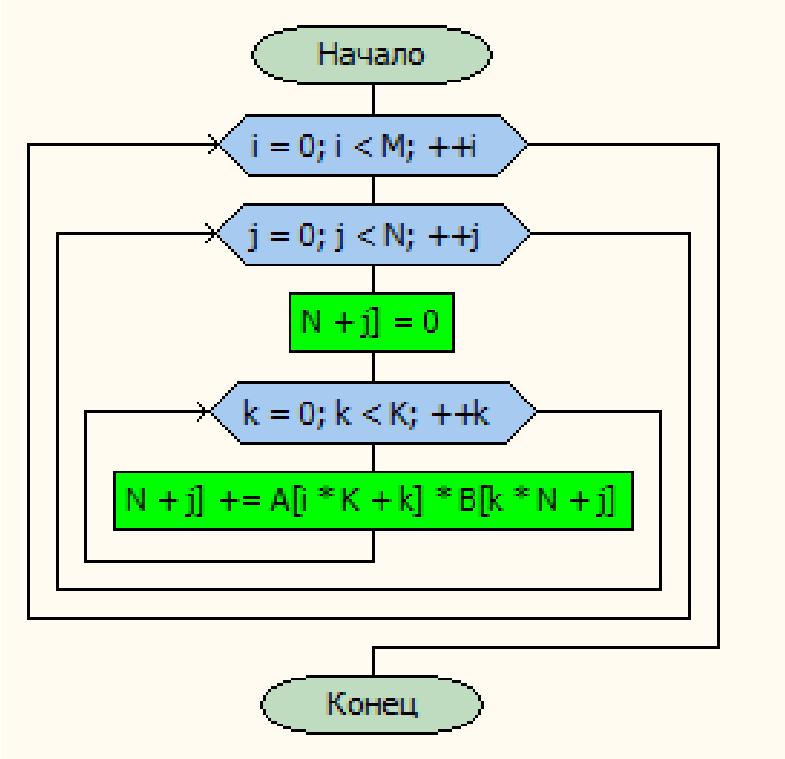


Рисунок 2 - Блок-схема тривиального умножения матрицы

Тривиальное умножение матрицы реализовано с помощью листинга:

```
void m_v0(int M, int N, int K, const float * A, const float
* B, float * C)
{
    for (int i = 0; i < M; ++i)
    {
        for (int j = 0; j < N; ++j)
        {
            C[i*N + j] = 0;
```

```
                for (int k = 0; k < K; ++k)
                    C[i*N + j] += A[i*K + k] * B[k*N + j];
            }
        }
    }
```

Для тестирования программы выбраны квадратные матрицы размером 3 из целых чисел от 0 до 9. Результаты тестирования (рисунок 3) показывают, что умножение матриц проведено правильно.

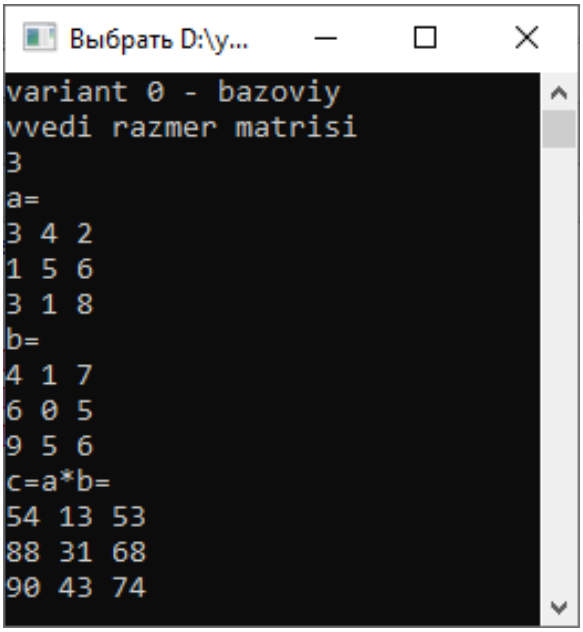


Рисунок 3 – Результат тестирования тривиального умножения матриц

2.2 Разработка программ эффективного использования вычислительных ресурсов при умножении матриц

С точки зрения эффективной реализации на различных компьютерах для дальнейшей работы выбрано совершенствование алгоритма путем разгрузки внутреннего цикла (вариант 1). Блок-схема умножения матриц с разгрузкой внутреннего цикла представлена на рисунке 4.

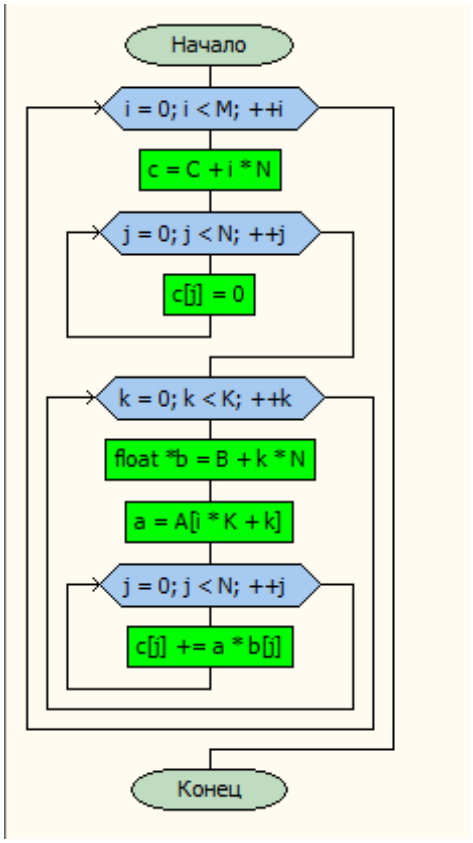


Рисунок 4 - Блок-схема умножения матриц с разгрузкой внутреннего цикла

Умножение матриц с разгрузкой внутреннего цикла реализовано с помощью следующего листинга:

```
    void m_v1(int M, int N, int K, const float * A, const
float * B, float * C)
    {
        for (int i = 0; i < M; ++i)
        {
            float * c = C + i * N;
            for (int j = 0; j < N; ++j)
                c[j] = 0;
            for (int k = 0; k < K; ++k)
            {
                const float * b = B + k * N;
```

```
                float a = A[i*K + k];
                for (int j = 0; j < N; ++j)
                    c[j] += a * b[j];
            }
        }
    }
```

Для тестирования программы выбраны квадратные матрицы размером 3 из целых чисел от 0 до 9. Результаты тестирования (рисунок 5) показывают, что умножение матриц проведено правильно.

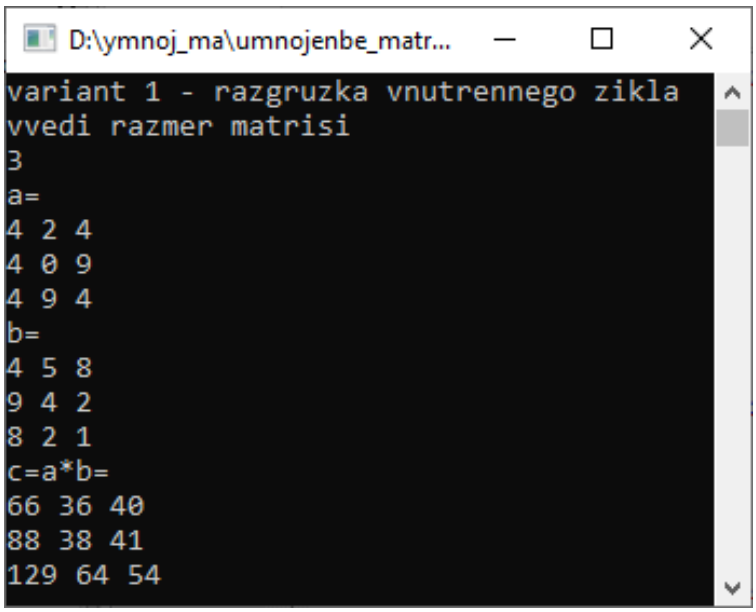


Рисунок 5 – Результат тестирования матрицы с разгрузкой внутреннего цикла

2.3 Разработка программ многопоточного умножении матриц

Затем применим технологию многопоточного умножения матриц (вариант 2). С точки зрения простоты программной реализации и универсальности для выбрана технология параллельных вычислений openMP. В программный код варианта 0 добавлена директива #pragma omp parallel for. Блок-схема многопоточного умножения матриц показана на рисунке 6.

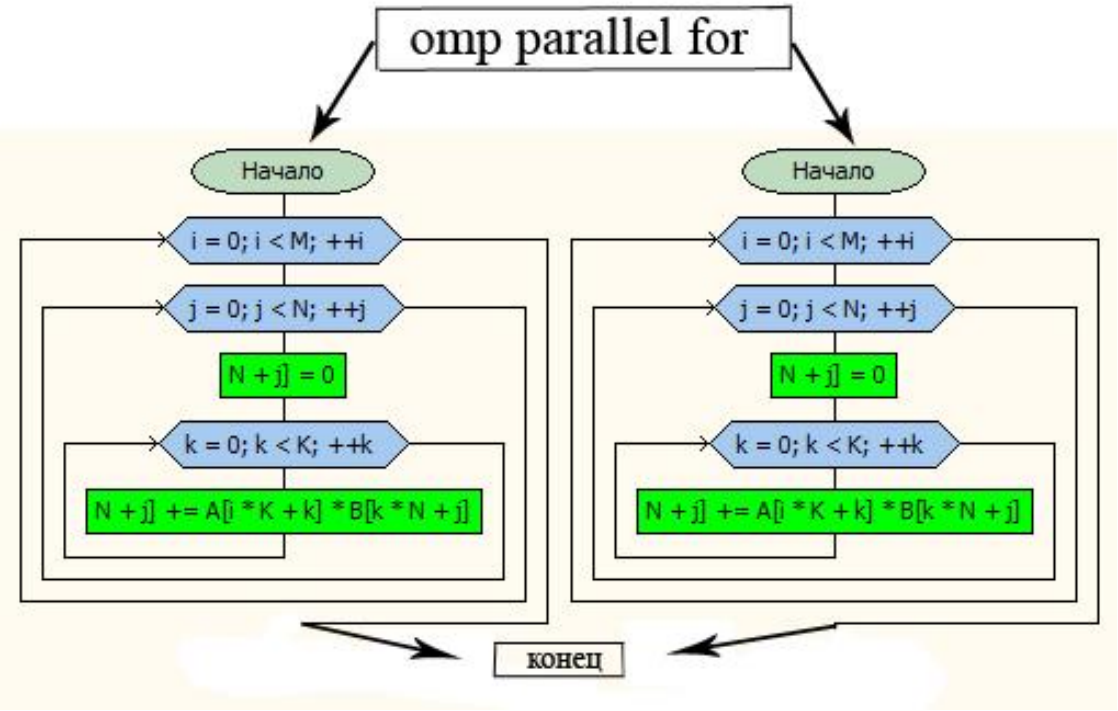


Рисунок 6 - Блок-схема многопоточного умножения матриц

Технология многопоточного умножения матриц реализована с помощью следующего листинга:

```
    void m_v2(int M, int N, int K, const float * A,
const float * B, float * C)
    {
         #pragma omp parallel for
    for (int i = 0; i < M; ++i)
    {
    …
        }
    }
```

Данная функция производит умножение строк матрицы A на столбцы матрицы B с использованием нескольких параллельных потоков. Каждый поток выполняет вычисления над несколькими соседними строками матрицы A и, таким образом, получает несколько соседних строк результирующей матрицы C.

Для тестирования программы выбраны квадратные матрицы размером 3 из целых чисел от 0 до 9. Результаты тестирования (рисунок 7) показывают, что умножение матриц проведено правильно.

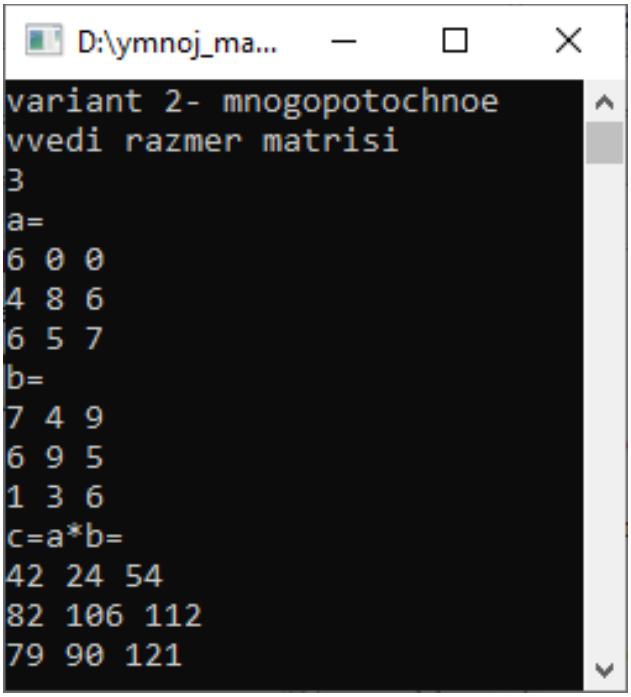


Рисунок 7 - Результат тестирования многопоточного умножения матриц

Разработаем комплексный вариант, который сочетает подходы, основанные на многопоточности и повышении эффективности за счет разгрузки внутреннего цикла (вариант 3).

Для тестирования программы выбраны квадратные матрицы размером 3 из целых чисел от 0 до 9. Результаты тестирования (рисунок 8) показывают, что умножение матриц проведено правильно.

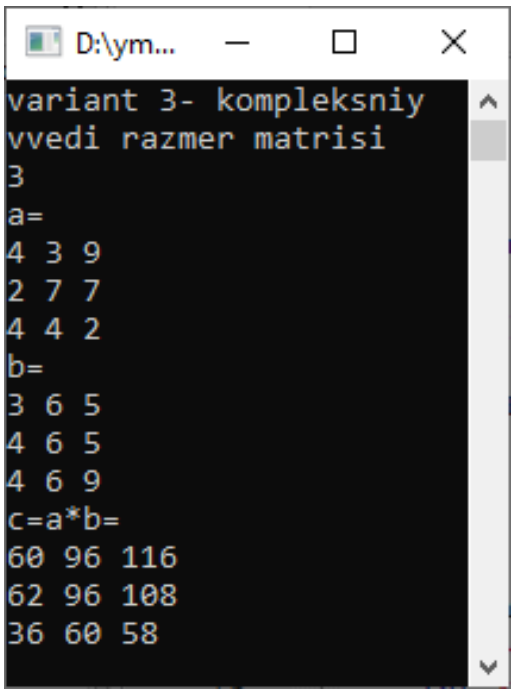


Рисунок 8 - Результат тестирования комплексного варианта умножения матриц

Выводы по разделу:

– разработаны 4 варианта программ, реализующих различные подходы к ускорению умножения матриц - за счет разгрузки внутреннего цикла – вариант 1, за счет многопоточности - вариант 2 и разработана программа, комбинирующая оба варианта – вариант 3.

– проведено тестирование разработанных программ на правильность вычислений.

3 Экспериментальное исследование эффективности разработанных программ

3.1 Планирование эксперимента

Цель – исследовать эффективность способов ускорения умножения матриц на различных компьютерах.

Экспериментальная часть работы имеет целью проверить эффективность ускорения матричного умножения в зависимости от размера матриц на различных компьютерах.

Оценивалось время выполнения умножения, и рассчитывался коэффициент ускорения относительно варианта 0.

$$K_y = \frac{T_i}{T_0} \quad (3)$$

где T_i – время исследуемых вариантов, T_0 – время варианта 0.

Каждый опыт повторялся по 3 раза. Ряд размеров матриц от 8 до 2048 по степеням двойки (8, 16, 32, 64, 128, 256, 1024, 2048), а также две дополнительные точки 768 и 1536 взятые для более точного описания кривых в области перегибов.

Таблица 3 – Варианты программ для исследования

Обозначение	Описание
Вариант 0	Тривиальное умножение матриц
Вариант 1	Алгоритм с вынесением повторяющихся действий из внутреннего цикла
Вариант 2	Многопоточное умножение матриц openMP
Вариант 3	Многопоточное умножение матриц openMP по алгоритму с вынесением повторяющихся действий из внутреннего цикла

В качестве аппаратуры для проведения эксперимента выбраны 4 компьютера, которые могут быть условно названы: сильный ноутбук, слабый ноутбук, сильный ПК, слабый ПК. Технические характеристики компьютеров приведены в таблице 4.

Таблица 4 - Экспериментальное оборудование

Характеристика	Компьютер А	Компьютер Б	Компьютер В	Компьютер Г
Имя (обозначение)	Слабый ноутбук (Н-)	Слабый ПК (ПК-)	Сильный ПК (ПК+)	Сильный ноутбук (Н+)
Модель	Lenovo G580	–	–	Lenovo 720
Частота процессора	Intel Core i5-3230M CPU 2,60GHz	Intel Pentium CPU G4600 @ 3.60GHz	AMD Ryzen 3 3100 4-Core Processor 3.60GHz	Intel Core i5-8250U CPU @ 1.60GHz
Число ядер процессора	2	2	4	4
Число потоков процессоров	4	4	8	8
Оперативная память	8ГБ	8ГБ	16ГБ	12ГБ
Операционная система	Windows 10	Windows 10	Windows 10	Windows 10

В программе вывод был организован в текстовый файл. Пример вывода представлен на рисунке 9.

```
0.018    0.011    0.040    0.035  16
17082.296    8670.037    4245.434    2317.521  1536
1.162    0.633    0.449    0.272  64
74.652    40.002    21.494    12.051  256
2092.022    1080.648    534.595    280.772  768
0.003    0.002    0.031    0.031  8
144355.827    19103.023    28280.839    5277.708  2048
0.133    0.075    0.070    0.052  32
8.489    4.776    2.633    1.511  128
648.170    297.269    153.946    83.436  512
5693.334    2360.234    1148.258    660.149  1024
15957.002    7979.794    4439.504    2180.096  1536
660.648    295.316    158.896    84.377  512
5759.332    2363.532    1247.279    655.147  1024
140270.753    18857.202    31610.606    5263.182  2048
71.802    37.856    20.044    11.017  256
1906.648    1012.227    506.613    277.063  768
8.414    4.737    2.613    1.456  128
0.016    0.010    0.035    0.032  16
1.141    0.602    0.359    0.220  64
0.002    0.001    0.031    0.030  8
0.128    0.074    0.070    0.051  32
144283.185    18691.389    33918.844    5152.157  2048
0.002    0.001    0.033    0.030  8
15445.047    7972.793    3995.909    2290.914  1536
0.017    0.010    0.032    0.031  16
5890.866    2350.528    1209.273    652.146  1024
0.130    0.074    0.068    0.050  32
1866.178    993.223    466.104    265.059  768
1.042    0.580    0.349    0.206  64
649.646    297.067    147.150    83.033  512
8.527    4.683    2.506    1.461  128
67.748    37.490    19.358    11.121  256
```

Рисунок 9 - Пример вывода результатов в текстовый файл

В соответствии с разработанным планом, проведены эксперименты, предусматривающие трехкратное повторение каждого опыта в случайном порядке. Размер умножаемых матриц варьировали по степеням двойки.

3.2 Результаты эксперимента

Результаты экспериментов на слабом ноутбуке приведены в приложении А. Среднее время расчета приведено в таблице 5. Коэффициент ускорения – в таблице 6.

Таблица 5 – Среднее время расчета на слабом ноутбуке, мс.

Размер матриц	Обозначение программы			
	v0	v1	v2	v3
8	0,003	0,002	0,055	0,05367
16	0,025	0,01667	0,066	0,05933
32	0,20267	0,12633	0,16633	0,11367
64	1,66767	0,99467	0,937	0,56267
128	14,0393	7,853	7,51167	3,9
256	122,443	61,1483	61,675	30,0633
512	1305,19	492,143	578,475	250,978
768	5241,11	1666,71	2310,57	853,471
1024	32213	3885,11	13115,7	2072,72
1536	110709	13236,3	39751,2	6543,28
2048	287626	32622,8	121317	16902,9

Таблица 6 – Коэффициент ускорения на слабом ноутуке

Размер матриц	Обозначение программы		
	v1	v2	v3
8	1,50	0,05	0,06
16	1,50	0,38	0,42
32	1,60	1,22	1,78
64	1,68	1,78	2,96
128	1,79	1,87	3,60
256	2,00	1,99	4,07
512	2,65	2,26	5,20
768	3,14	2,27	6,14
1024	8,29	2,46	15,54
1536	8,36	2,79	16,92
2048	8,82	2,37	17,02

Результаты экспериментов на сильном ноутбуке приведены в приложении Б. Среднее время расчета приведено в таблице 7. Коэффициент ускорения – в таблице 8.

Таблица 7 – Среднее время расчета на сильном ноутбуке, мс.

Размер матриц	Обозначение программы			
	v0	v1	v2	v3
1	2	3	4	5
8	0,002	0,002	0,04067	0,041

Продолжение таблицы 7

1	2	3	4	5
16	0,01767	0,01133	0,045	0,04333
32	0,13567	0,08933	0,08567	0,063
64	1,07133	0,71033	0,42133	0,265
128	9,55833	5,82367	3,08867	1,723
256	84,068	45,9193	25,0593	12,7613
512	1017,78	370,025	288,749	98,7277
768	3445,04	1231,12	984,014	331,902
1024	16677,4	2906,56	3701,27	790,238
1536	45295,4	9863,63	13238,2	3420,73
2048	244562	23144,9	90020	8220,93

Таблица 8 – Коэффициент ускорения на сильном ноутуке.

Размер матриц	Обозначение программы		
	v1	v2	v3
8	1,00	0,05	0,05
16	1,56	0,39	0,41
32	1,52	1,58	2,15
64	1,51	2,54	4,04
128	1,64	3,09	5,55
256	1,83	3,35	6,59
512	2,75	3,52	10,31
768	2,80	3,50	10,38
1024	5,74	4,51	21,10
1536	4,59	3,42	13,24
2048	10,57	2,72	29,75

Результаты экспериментов на слабом ПК приведены в приложении В. Среднее время расчета приведено в таблице 9. Коэффициент ускорения – в таблице 10.

Таблица 9 – Среднее время расчета на слабом ПК, мс.

Размер матриц	Обозначение программы			
	v0	v1	v2	v3
1	2	3	4	5
8	0,002	0,00133	0,04033	0,03933
16	0,01767	0,01133	0,04833	0,046
32	0,13433	0,08733	0,111	0,08133
64	1,092	0,70933	0,59667	0,361

Продолжение таблицы 9

1	2	3	4	5
128	9,46933	5,731	4,961	2,68867
256	86,4083	45,831	42,8333	21,8587
512	1035,89	363,593	482,044	155,766
768	3180,12	1141,46	1485,73	559,753
1024	26214,6	2958,44	12229,2	1274,92
1536	70471,9	9079,95	30010,7	3880,63
2048	234489	22793	110765	10005

Таблица 10 – Коэффициент ускорения на слабом ПК.

Размер матриц	Обозначение программы		
	v1	v2	v3
8	1,50	0,05	0,05
16	1,56	0,37	0,38
32	1,54	1,21	1,65
64	1,54	1,83	3,02
128	1,65	1,91	3,52
256	1,89	2,02	3,95
512	2,85	2,15	6,65
768	2,79	2,14	5,68
1024	8,86	2,14	20,56
1536	7,76	2,35	18,16
2048	10,29	2,12	23,44

Результаты экспериментов на сильном ПК приведены в приложении Г. Среднее время расчета приведено в таблице 11. Коэффициент ускорения – в таблице 12.

Таблица 11 – Среднее время расчета на сильном ПК, мс.

Размер матриц	Обозначение программы			
	v0	v1	v2	v3
1	2	3	4	5
8	0,00233	0,00133	0,03167	0,03033
16	0,017	0,01033	0,03567	0,03267
32	0,13033	0,07433	0,06933	0,051
64	1,115	0,605	0,38567	0,23267
128	8,47667	4,732	2,584	1,476
256	71,4007	38,4493	20,2987	11,3963

Продолжение таблицы 11

1	2	3	4	5
512	652,821	296,551	153,331	83,6153
768	1954,95	1028,7	502,437	274,298
1024	5781,18	2358,1	1201,6	655,814
1536	16161,4	8207,54	4226,95	2262,84
2048	142970	18883,9	31270,1	5231,02

Таблица 12 – Коэффициент ускорения на сильном ПК.

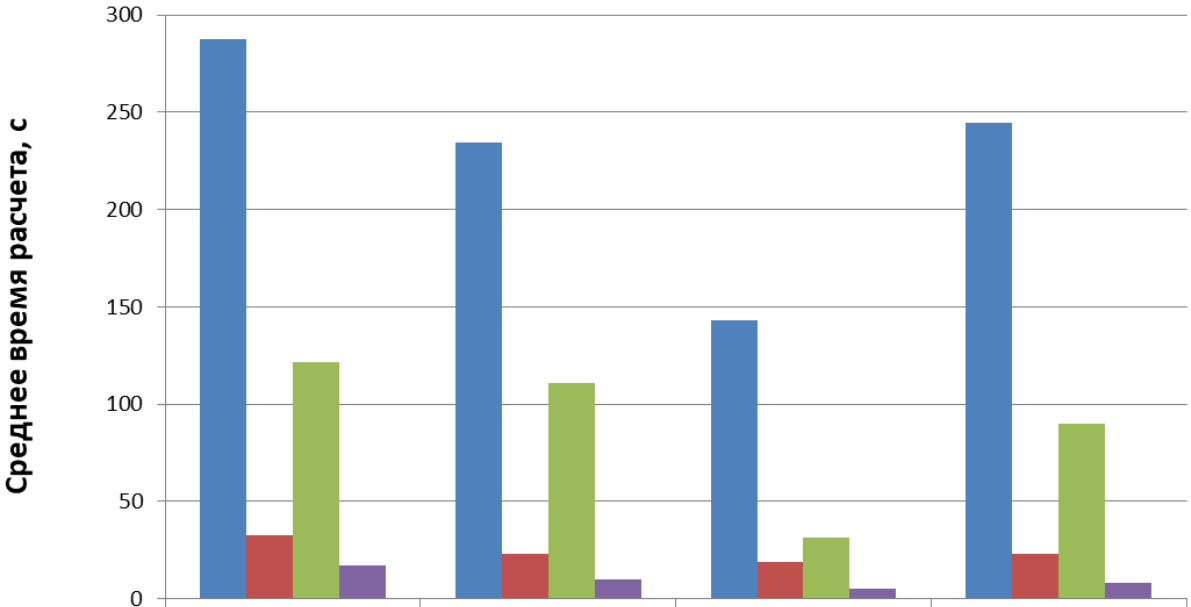
Размер матриц	Обозначение программы		
	v1	v2	v3
8	1,75	0,07	0,08
16	1,65	0,48	0,52
32	1,75	1,88	2,56
64	1,84	2,89	4,79
128	1,79	3,28	5,74
256	1,86	3,52	6,27
512	2,20	4,26	7,81
768	1,90	3,89	7,13
1024	2,45	4,81	8,82
1536	1,97	3,82	7,14
2048	7,57	4,57	27,33

Результаты, представленные, в таблицах 5-12 и в приложениях А-Г, показывают что разброс между повторами существенно ниже, чем отличие в средних значениях между экспериментами. Следовательно, полученные результаты – достоверны.

3.3 Анализ результатов эксперимента

На рисунке 10 показаны результаты экспериментов в абсолютном выражении для размера матриц 2048x2048.

На диаграмме видно, что среднее время перемножения гораздо меньше на разработанном комплексном алгоритме (вариант 3), чем при тривиальном умножении матрицы (вариант 0).



	Н-	ПК-	ПК+	Н+
вариант 0	288	234	143	245
вариант 1	33	23	19	23
вариант 2	121	111	31	90
вариант 3	17	10	5	8

Рисунок 10 - Результаты экспериментов в абсолютном выражении (размер матриц 2048x2048)

На рисунке 11 показано влияние размера матрицы на ускорение от разгрузки внутреннего цикла (вариант 1) на различных компьтерах.

Пунктиром обозначены ноутбуки, сплошной линией – ПК. Слабые обозначены черными точками, сильные – белыми квадратами.

Результаты показывают, что на слабых имеет место скачкообразный рост ускорения в диапазоне размера матриц от 768 до 1024. При дальнейшем росте размера матриц сильные ноутбук и компьютер приближаются по величине ускорения к слабым.

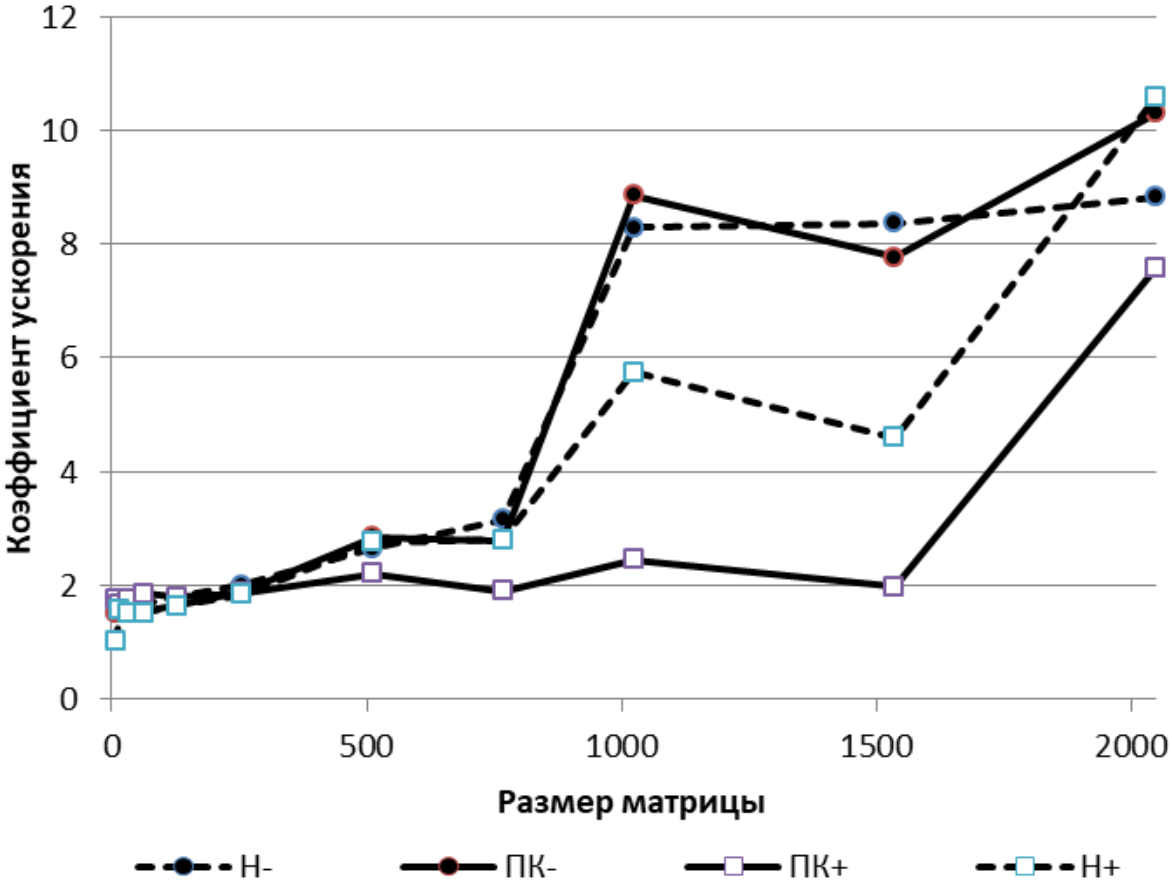


Рисунок 11 - Влияние размера матрицы на ускорение от разгрузки внутреннего цикла (вариант 1)

На рисунке 12 показано влияние размера матрицы на ускорение от распараллеливания вычислений (вариант 2)

Ускорение от параллельности вычислений, как показал эксперимент, в целом соответствует числу вычислительных потоков ядер процессоров. Слабые компьютер и нотбук, оснащенные 2х ядерными процессорами с 4ми потоками, сильные – 4х ядерные процессоры и 8 потоков.

С ростом размера матрицы коэффициент ускорения достигнув максимума начинает снижаться. В наибольшей степени данный эффект проявляется на сильном ноутбуке. Вероятно, это связано с тем, что с увеличение объемов, вычисления частично переносятся в более медленные участки памяти, с которыми технология OMP работает менее эффективно. На

больших матрицах более эффективными будут технологии, использующие графические процессоры, например, CUDA.

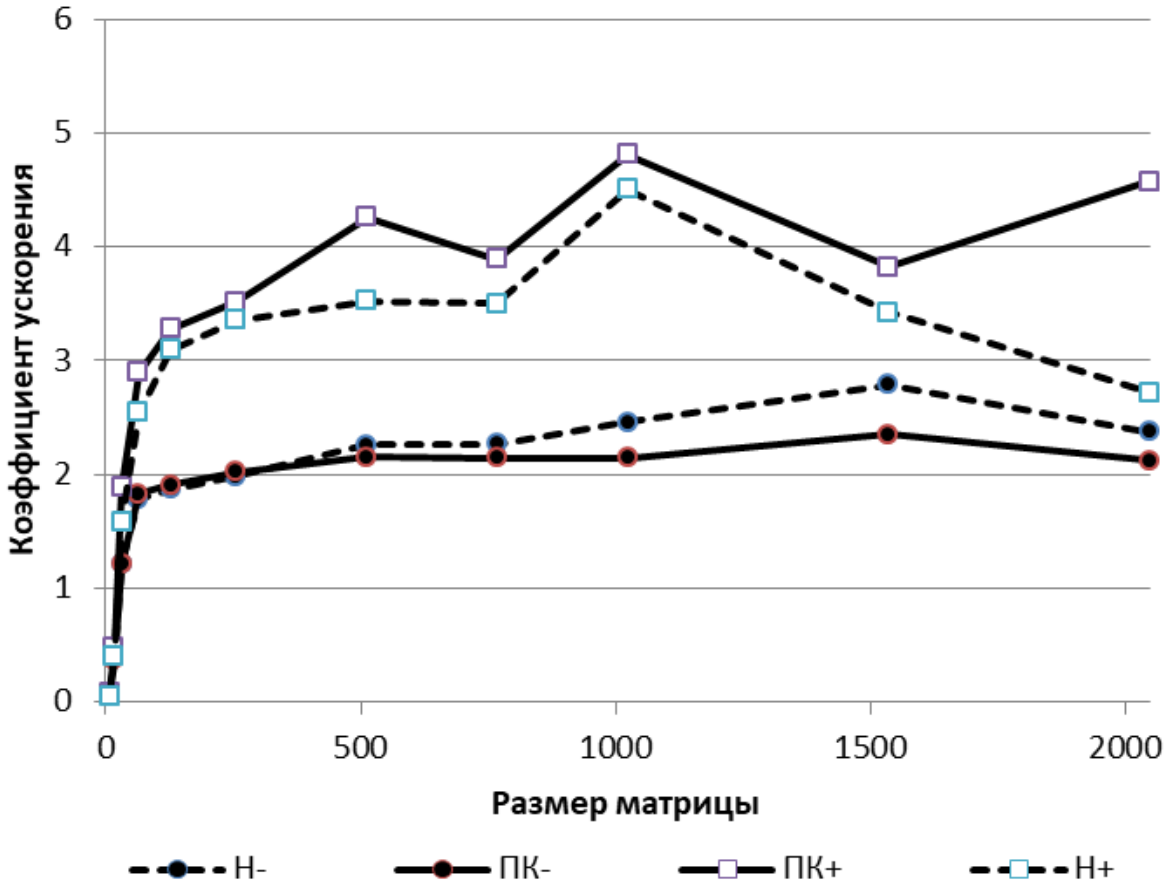


Рисунок 12 - Влияние размера матрицы на ускорение от распараллеливания вычислений (вариант 2)

На рисунке 13 показано ускорение от распараллеливания вычислений на малых размерах матрицы.

На малых размерах матриц четко виден случай, когда величина вычислительных расходов на управление потоками превышает расходы на вычисление в одном потоке. При этом ускорение меньше единицы.

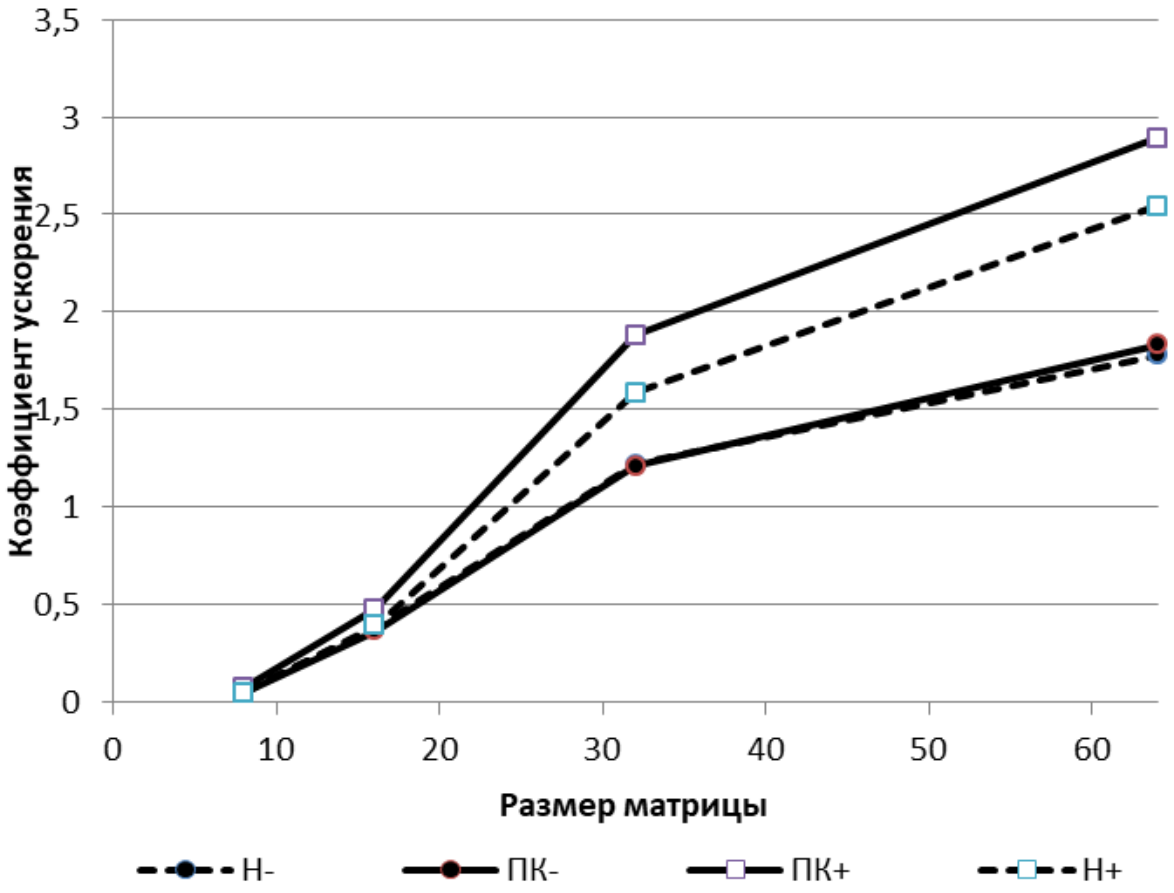


Рисунок 13 - Ускорение от распараллеливания вычислений на малых размерах матрицы

На рисунке 14 показано влияние размера матрицы на ускорение (вариант 3 - комплексный).

В случае комплексного применения двух рассматриваемых подходов очевидно имеет место большее ускорение. При этом существенных различий между ноутбуками и ПК не наблюдается – при максимальном размере матрицы максимальное ускорение (около 30ти) у сильного ноутбука, минимальное – у слабого – в 17 раз.

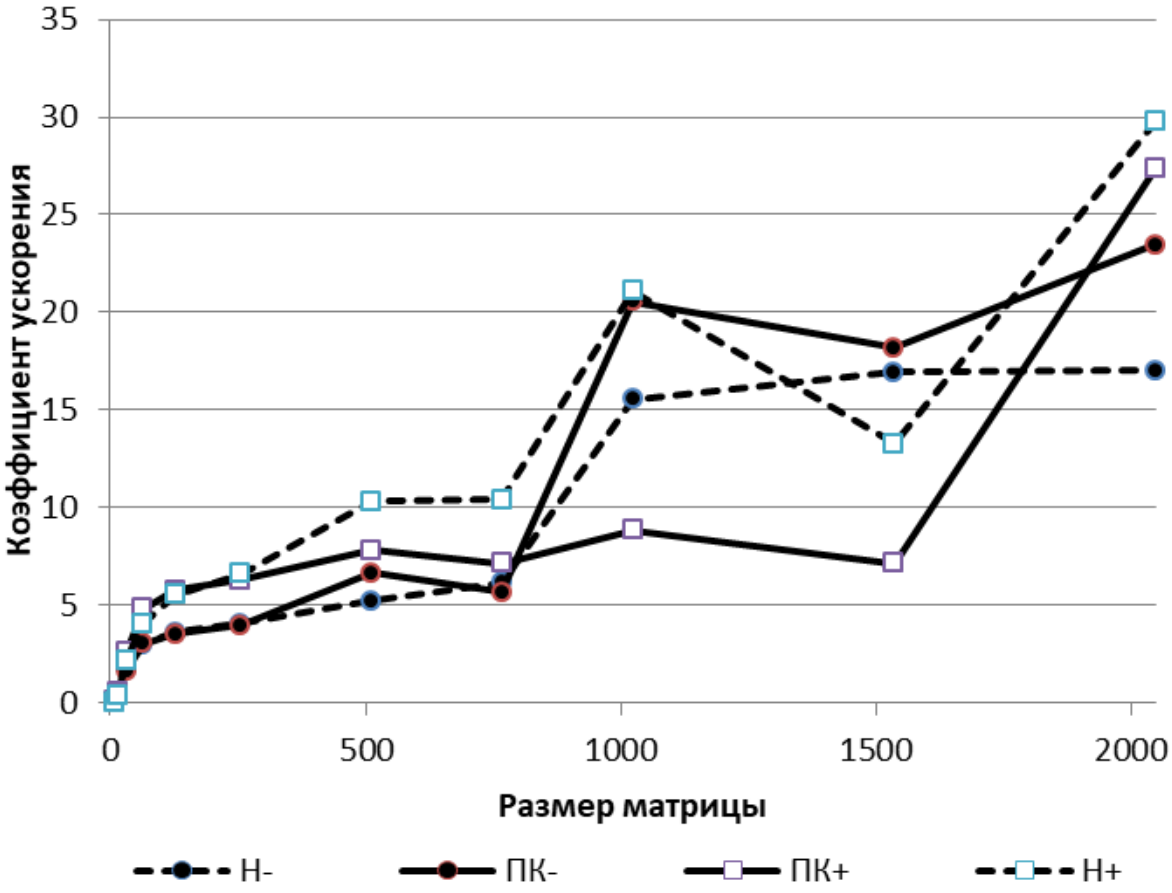


Рисунок 14 - Влияние размера матрицы на ускорение (вариант 3 - комплексный)

Выводы по разделу:

– получены экспериментальные зависимости изменения коэффициента ускорения для различных компьютеров.

– проведен анализ результатов и предложены пути совершенствования. Наибольшее ускорение матричного умножения происходит при выполнении разработанного комплексного алгоритма (вариант 3).

При этом существенных различий между ноутбуками и ПК не наблюдается.

Заключение

В процессе выполнения бакалаврской работы разработаны эффективные алгоритмы матричного умножения. В ходе данной работы были поставлены и выполнены следующие задачи.

Исследовано современное состояние в области теории и практики матричного умножения. Установлены основные направления практических подходов к ускорению умножения матриц, области применения, достоинства и недостатки используемых приемов. Определены приемы для практической реализации матричного умножения.

Практически реализованы программы, позволяющие ускорить умножение матриц за счет изменения вычислительного алгоритма (за счет вынесение постоянных действий из внутреннего цикла) и за счет параллельных вычислений. Параллельным вычислениям уделяется большое внимание в связи с распространением систем с большим количеством потоков у процессора. По результатам анализа существующих параллельных технологий определено, что в работе наиболее целесообразно применение параллельной технологии OMP.

С применением разработанных программ экспериментально исследовано влияние размера матрицы на ускорение вычислений. Получены экспериментальные зависимости изменения коэффициента ускорения для различных компьютеров. Проведен анализ результатов и предложены пути совершенствования.

В результате проведенного сравнительного анализа, были сделаны следующие выводы из проделанной работы:

Наиболее эффективным алгоритмом матричного умножения является разработанный комплексный алгоритм (вариант 3), включающий в себя подходы, основанные на многопоточности и повышении эффективности за счет разгрузки внутреннего цикла.

Список используемой литературы и используемых источников

1. Алексеев, В.Б. Умножения матриц размеров 5×2 И 2×2 / В.Б. Алексеев // Ученые записки Казанского университета. Серия Физико-математические науки. – 2014. – № 3. – С. 19-29. – Текст: электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/journal/issue/296338 (дата обращения: 13.05.2022).
2. Белоусов, И.В. Матрицы и Определители: учеб. Пособие. Кишинев, 2006. 101 с.
3. Биллиг, В. А. Параллельные вычисления и многопоточное программирование : учебное пособие / В. А. Биллиг. — 2-е изд. — Москва : ИНТУИТ, 2016. — 310 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/100361 (дата обращения: 07.05.2022).
4. Ефимов, Н. В. Квадратичные формы и матрицы : учебное пособие / Н. В. Ефимов. — Москва : ФИЗМАТЛИТ, 2012. — 168 с. – Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/59543 (дата обращения: 03.05.2022.
5. Игонина, Е. В. Основы алгебры матриц и векторов. Линейное программирование : учебное пособие / Е. В. Игонина, О. Н. Прокуратова. — Елец : ЕГУ им. И.А. Бунина, 2008. – 75 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/195891 (дата обращения: 13.05.2022).
6. Корчевская, Е.А. Автоматизированная система распределённых вычислений для численного решения алгебраических задач / Е.А. Корчевская, Л.В. Маркова , А.Н. Красоткина // Математические структуры и моделирование 2014. – №4. – С. 69 – 72
7. Матричное умножение. Медленное достижение мифической цели – URL: https://habr.com/ru/company/timeweb/blog/549360/ (дата обращения: 04.05.2022).

8. Немцова, Т.И. Программирование на языке высокого уровня. Программирование на языке С++ : учебное пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев ; под ред. Л.Г. Гагариной. — Москва : ФОРУМ : ИНФРА-М, 2021. — 512 с. + Доп. материалы [Электронный ресурс]. — (Среднее профессиональное образование). Текст : электронный. - URL: https://znanium.com/catalog/product/1172261 (дата обращения: 23.05.2022).

9. Пан, В.Я. Быстрое умножение матриц и смежные вопросы алгебры // Математический сборник. – 2017. – №11.

10. Панкратов, Е. Л. Операции над матрицами. решение систем линейных алгебраических уравнений : учебно-методическое пособие / Е. Л. Панкратов. — Нижний Новгород : ННГУ им. Н. И. Лобачевского, 2020. — 17 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/191782 (дата обращения: 03.05.2022).

11. Тыртышников, Е. Е. Матричный анализ и линейная алгебра : учебное пособие / Е. Е. Тыртышников. — Москва : ФИЗМАТЛИТ, 2007. — 480 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/2352 (дата обращения: 21.05.2022).

12. Умнов А.Е. Аналитическая геометрия и линейная алгебра: Учеб. Пособие/ Умнов А.Е./ – 3е. изд., испр. и доп. – М.: МФТИ, 2011. – 544 с.

13. Умножение матриц: эффективная реализация шаг за шагом. – Режим доступа: https://habr.com/ru/post/359272/ (дата обращения: 4.05.2022).

14. Федотов, И. Е. Параллельное программирование. Модели и приемы : практическое пособие / И. Е. Федотов. - Москва : СОЛОН-Пресс, 2020. - 390 с. - (Серия «Библиотека профессионала»). Текст : электронный. - URL: https://znanium.com/catalog/product/1858781 (дата обращения: 13.05.2022).

15. Чубариков В. Н., Добровольский Н. Н., Реброва И. Ю., Добровольский Н. М. - Информатика, компьютер, сложность вычислений // Чебышевский сборник. — 2021. — №1 (77).

16. Энтони, У. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ : учебное пособие / У. Энтони ; перевод с английского А. А. Слинкин. — Москва : ДМК Пресс, 2012. — 672 с. – Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/4813 (дата обращения: 07.05.2022).

17. Alman, Josh & Williams, Virginia. (2021). A Refined Laser Method and Faster Matrix Multiplication. 10.1137/1.9781611976465.32. - https://arxiv.org/abs/2010.05846 (дата обращения: 07.05.2022).

18. Coppersmith D. and Winograd S.. Matrix multiplication via arithmetic progressions. J. Symbolic Computation, 9:251–280, 1990.

19. Divide&Conquer над алгоритмом Штрассена https://habr.com/ru/post/313258/(дата обращения: 04.05.2022).

20. Henry Cohn and Christopher Umans. Fast matrix multiplication using coherent configurations. pages 1074–1086, 2013.

21. Joseph M. Landsberg and Giorgio Ottaviani. New lower bounds for the border rank of matrix multiplication. Theory of Computing, 11(1):285–298, 2015.

22. Matrix Multiplication Inches Closer to Mythic Goal [Электронный ресурс] URL: https://science-education.ru/ru/article/view?id=25956 (дата обращения: 27.05.2021).

23. Strassen V. Gaussian Elimination is not Optimal (англ.) // Numer. Math / F. Brezzi — Springer Science+Business Media, 1969. — Vol. 13, Iss. 4. — P. 354—356.

Приложение А

Результаты экспериментов

Таблица А1 - Результаты экспериментов на слабом ноутбуке

Размер матриц	Обозначение программы			
	v0	v1	v2	v3
8	0,003	0,002	0,054	0,053
8	0,003	0,002	0,054	0,053
8	0,003	0,002	0,057	0,055
16	0,025	0,016	0,064	0,058
16	0,025	0,017	0,065	0,059
16	0,025	0,017	0,069	0,061
32	0,204	0,125	0,172	0,119
32	0,201	0,128	0,157	0,111
32	0,203	0,126	0,17	0,111
64	1,662	0,978	0,916	0,527
64	1,676	0,996	0,934	0,537
64	1,665	1,01	0,961	0,624
128	14,556	7,718	7,447	4,018
128	13,654	7,749	6,996	3,786
128	13,908	8,092	8,092	3,896
256	121,036	59,786	59,728	29,658
256	122,591	60,198	60,962	30,187
256	123,701	63,461	64,335	30,345
512	1292,199	485,045	568,646	239,252
512	1368,151	483,367	551,66	239,851
512	1255,223	508,018	615,119	273,83
768	5207,775	1641,2	2405,51	821,491
768	5246,814	1716,937	2332,557	911,435
768	5268,735	1641,981	2193,64	827,487
1024	33494	4022,066	13915,73	1890,829
1024	31696,36	3814,637	13006,94	1912,815
1024	31448,55	3818,634	12424,3	2414,504
1536	110225,9	13227,32	40293,49	6589,918
1536	110423,7	13319,75	39925,27	6379,048
1536	111476,9	13161,96	39034,82	6660,872
2048	277863,2	31561,45	115170,5	15545,37
2048	306990,9	31400,55	119895	16357,87
2048	278024,8	34906,38	128884,2	18805,35

Приложение Б

Результаты экспериментов

Таблица Б1 - Результаты экспериментов на сильном ноутбуке

Размер матриц	Обозначение программы			
	v0	v1	v2	v3
8	0,002	0,002	0,04	0,042
8	0,002	0,002	0,041	0,041
8	0,002	0,002	0,041	0,04
16	0,018	0,012	0,046	0,044
16	0,017	0,011	0,044	0,043
16	0,018	0,011	0,045	0,043
32	0,138	0,089	0,086	0,063
32	0,135	0,089	0,084	0,062
32	0,134	0,09	0,087	0,064
64	1,063	0,708	0,425	0,258
64	1,069	0,709	0,415	0,265
64	1,082	0,714	0,424	0,272
128	9,578	5,86	3,022	1,731
128	9,49	5,829	3,119	1,71
128	9,607	5,782	3,125	1,728
256	85,785	45,995	25,088	13,253
256	83,687	46,204	25,35	12,568
256	82,732	45,559	24,74	12,463
512	1017,989	370,823	285,39	102,182
512	1001,866	369,688	296,376	95,752
512	1033,488	369,564	284,482	98,249
768	3523,82	1236,464	1023,015	333,777
768	3394,883	1216,292	957,884	326,584
768	3416,422	1240,606	971,142	335,345
1024	17138,15	2964,64	3724,391	794,031
1024	18317,61	2948,645	4140,527	832,187
1024	14576,53	2806,398	3238,888	744,496
1536	46025,47	9882,695	12707,16	3363,418
1536	44235,78	9927,166	13675,54	3418,041
1536	45625,06	9781,016	13331,98	3480,73
2048	245174	23609,02	93160,51	8481,55
2048	238979,9	23331,44	86261,75	8040,887
2048	249532,3	22494,13	90637,64	8140,357

Приложение В

Результаты экспериментов

Таблица В1 - Результаты экспериментов на слабом ПК

Размер матриц	Обозначение программы			
	v0	v1	v2	v3
8	0,002	0,002	0,041	0,04
8	0,002	0,001	0,04	0,039
8	0,002	0,001	0,04	0,039
16	0,019	0,012	0,049	0,05
16	0,017	0,011	0,048	0,044
16	0,017	0,011	0,048	0,044
32	0,141	0,09	0,118	0,085
32	0,131	0,085	0,107	0,079
32	0,131	0,087	0,108	0,08
64	1,142	0,726	0,626	0,375
64	1,062	0,698	0,585	0,364
64	1,072	0,704	0,579	0,344
128	9,731	5,925	5,702	2,995
128	9,396	5,611	4,645	2,561
128	9,281	5,657	4,536	2,51
256	88,596	46,287	43,492	21,687
256	86,604	46,151	44,317	23,31
256	84,025	45,055	40,691	20,579
512	1076,113	388,969	545,541	170,378
512	1015,284	347,415	442,151	146,465
512	1016,281	354,396	458,441	150,455
768	26831,27	3001,973	12885,54	1268,608
768	26438,29	3073,807	12127,28	1293,538
768	25374,14	2799,543	11674,81	1262,625
1024	236548,4	23207,69	109735,2	9406,842
1024	228502,8	22684,33	112765,7	9692,078
1024	238416,6	22487,04	109794,6	10916,11
1536	0,002	0,002	0,041	0,04
1536	0,002	0,001	0,04	0,039
1536	0,002	0,001	0,04	0,039
2048	0,019	0,012	0,049	0,05
2048	0,017	0,011	0,048	0,044
2048	0,017	0,011	0,048	0,044

Приложение Г

Результаты экспериментов

Таблица Г1 - Результаты экспериментов на сильном ПК

Размер матриц	Обозначение программы			
	v0	v1	v2	v3
8	0,003	0,002	0,031	0,031
8	0,002	0,001	0,031	0,03
8	0,002	0,001	0,033	0,03
16	0,018	0,011	0,04	0,035
16	0,016	0,01	0,035	0,032
16	0,017	0,01	0,032	0,031
32	0,133	0,075	0,07	0,052
32	0,128	0,074	0,07	0,051
32	0,13	0,074	0,068	0,05
64	1,162	0,633	0,449	0,272
64	1,141	0,602	0,359	0,22
64	1,042	0,58	0,349	0,206
128	8,489	4,776	2,633	1,511
128	8,414	4,737	2,613	1,456
128	8,527	4,683	2,506	1,461
256	74,652	40,002	21,494	12,051
256	71,802	37,856	20,044	11,017
256	67,748	37,49	19,358	11,121
512	648,17	297,269	153,946	83,436
512	660,648	295,316	158,896	84,377
512	649,646	297,067	147,15	83,033
768	2092,022	1080,648	534,595	280,772
768	1906,648	1012,227	506,613	277,063
768	1866,178	993,223	466,104	265,059
1024	5693,334	2360,234	1148,258	660,149
1024	5759,332	2363,532	1247,279	655,147
1024	5890,866	2350,528	1209,273	652,146
1536	17082,3	8670,037	4245,434	2317,521
1536	15957	7979,794	4439,504	2180,096
1536	15445,05	7972,793	3995,909	2290,914
2048	144355,8	19103,02	28280,84	5277,708
2048	140270,8	18857,2	31610,61	5263,182
2048	144283,2	18691,39	33918,84	5152,157