МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ, ЧИСЛЕННЫЕ МЕТОДЫ И КОМПЛЕКСЫ ПРОГРАММ/MATHEMATICAL MODELING, NUMERICAL METHODS AND PROGRAM COMPLEXES

DOI: https://doi.org/10.60797/IRJ.2025.160s.35

РЕАЛИЗАЦИЯ ОБОБЩЕННОЙ БИБЛИОТЕКИ ДЛЯ МАТРИЧНЫХ ВЫЧИСЛЕНИЙ НА ГРАФИЧЕСКОМ ПРОЦЕССОРЕ ДЛЯ МЕТОДОВ РЕШЕНИЯ СЛАУ

Научная статья

Устюжанин Д.А.[1, *], Пицхелаури С.С.[2], Некрасов К.А.[3]

[2] ORCID : 0000-0003-2192-1880;

[3] ORCID : 0000-0002-1863-2597;

[1, 2, 3] Уральский федеральный университет имени первого Президента России Б. Н. Ельцина, Екатеринбург, Российская Федерация

* Корреспондирующий автор (danil.ustiuzhanin[at]urfu.me)

Аннотация

В рамках данной работы была реализована программная библиотека на языках программирования C/C++ и CUDA C. Центральное отличие предложенного в данной работе подхода к организации библиотеки для матричной алгебры на графическом процессоре, от подхода, описанного в статье [1], заключается в использовании системы классов под управлением matrixManagerDevice. Также в работе представлены механизмы обобщенного программирования матричных операций, т. е. механизмы, реализующие плотное взаимодействие матричных объектов, расположенных как в оперативной памяти под управлением центрального процессора (CPU), так и в памяти графического процессора (GPU). Результаты тестирования представленного программного обеспечения показывают, что приведенные изменения в устройстве библиотеки улучшают общую производительность и стабильности расчетов. В целом, данная работа представляет практическую значимость для многих областей, связанных с использованием вычислительной линейной алгебры, таких как машинное обучение, в частности, обучение глубоких нейронных сетей, молекулярная динамика, обработка больших объёмов данных и т. д.

Ключевые слова: распараллеливание алгоритмов, метод градиентного спуска, метод сопряженных градиентов, метод бисопряжённых градиентов.

IMPLEMENTATION OF A GENERALISED LIBRARY FOR MATRIX CALCULATIONS ON A GRAPHICS PROCESSOR FOR METHODS OF SOLVING SLAES

Research article

Ustyuzhanin D.A.[1, *], Pitskhelaury S.S.[2], Nekrasov K.A.[3]

[2] ORCID : 0000-0003-2192-1880;

[3] ORCID : 0000-0002-1863-2597;

[1, 2, 3] Ural Federal University named after the first President of Russia B. N. Yeltsin, Ekaterinburg, Russian Federation

* Corresponding author (danil.ustiuzhanin[at]urfu.me)

Abstract

Within the scope of this work, a software library was implemented in the C/C++ and CUDA C programming languages. The main difference between the approach to organising a library for matrix algebra on a graphics processor suggested in this work and the approach described in article [1] is the use of a class system controlled by matrixManagerDevice. The work also presents mechanisms for generalised programming of matrix operations, i.e. mechanisms that implement close interaction between matrix objects located both in the main memory under the control of the central processing unit (CPU) and in the memory of the graphics processing unit (GPU). The results of testing the presented software show that the changes made to the library improve the overall performance and stability of calculations. Overall, this paper is of practical significance for many areas related to the use of computational linear algebra, such as machine learning, in particular deep neural network training, molecular dynamics, big data processing, etc.

Keywords: parallelisation of algorithms, gradient descent method, conjugate gradient method, biconjugate gradient method.

Введение

Матричная алгебра неотъемлемо связана со многими прикладными областями современного технического мира, в частности, к задачам вычислительной линейной алгебры приводят вопросы, возникающие при компьютерном моделировании физических процессов, машинном обучение, обработке данных. Например, в рамках физических задач часто возникает потребность в численном решении уравнений в частных производных, для которых разработаны различные разностные схемы [2]. Часть разностных схем, которые принято называть неявными, приводят к задаче о решении системы линейных алгебраических уравнений (СЛАУ), которая является традиционной для вычислительной линейной алгебры. В связи со столь широким прикладным спектром задач возникает закономерная потребность в обобщённых математических пакетах для работы с матрицами на ЭВМ.

Центральный процессор (CPU) хорошо справляется с обработкой матриц, но невысокого порядка. Как правило, на персональном компьютере пределом оптимальной работы матричных последовательных алгоритмов являются матрицы размера не более чем 1000 на 1000 элементов, этот предел объясняется тем, что большинство матричных

операций имеют сложность более чем $O\left(n^2\right)$, где n — размерность матрицы, для упрощения квадратной. Этот предел сковывает многих программистов и научных сотрудников, у которых нет доступа к суперкомпьютерам.

Одним из путей решения ограниченности применения матриц высокого порядка являются параллельные вычисления на графическом процессоре (GPU). Графический процессор, изначально спроектированный для работы с графикой, оказался серьезным соперником для центрального процессора в области научных расчетов.

Однако на данный момент не существует полноценных библиотек, не ориентированных на конкретную область, которые бы предоставляли многочисленный функционал для программирования матричных алгоритмов на графическом процессоре. Также нет библиотек, которые предоставляют возможность гибридной обработки матриц как на центральном, так и на графическом процессоре.

В рамках данной работы мы хотим представить подход к организации библиотеки для работы с матрицами без выделенной прикладной области, для GPU и CPU на языках программирования C/C++ и CUDA C [3].

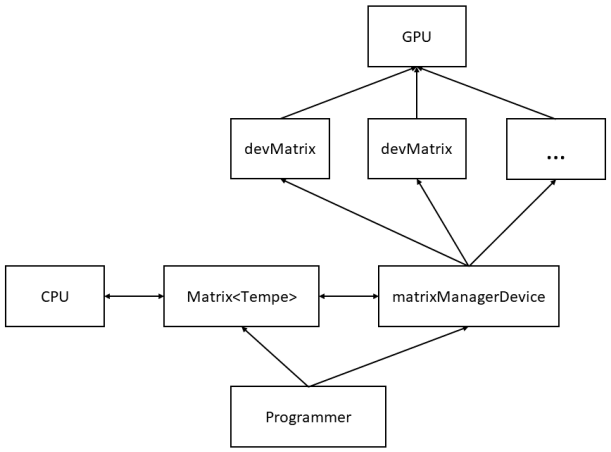

Рисунок 1 - Блок схема библиотеки
DOI: https://doi.org/10.60797/IRJ.2025.160s.35.1

Библиотека формально состоит из двух частей (см. рис. 1). Первая — шаблонный класс Matrix для CPU, вторая — совокупность классов matrixManagerDevice, smartMatrix, devMatrix и devSparseMatrix.

Шаблонный класс Matrix

Matrix<T> — шаблонный класс, представляющий CPU часть библиотеки. В широком смысле предполагается, что данный класс можно использовать не только как двумерный или одномерный контейнер чисел, но и как многомерный контейнер некоторых объектов. Многомерность достигается, как и в случае класса vector из STL C++, путем вложенности шаблонов. Например, шаблон Matrix<Matrix<double>> является представлением объекта 4-го ранга, в качестве базового типа выбран вещественный тип двойной точности. Безопасность гарантируется путем установки базовых требований к классу шаблона Matrix, в частности, сам класс Matrix удовлетворяет поставленным требованиям.

В рамках реализации Matrix, шаблоны также позволяют обрабатывать матрицы с объектами пользовательских типов, например с расширенными числовыми типами с плавающей запятой, комплексными числами и рациональными дробями. Последние в ряде случаев позволяют решать СЛАУ точными методами без потери знаков.

Иллюстрацией применения шаблонного класса Matrix является матричная свертка (конволюция), определяемая как:

$$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) \cdot K(i-m, j-n) \quad (1)$$

где I — исходная матрица (изображение), K - ядро свертки, $s(i, j)$ - элемент результирующей матрицы. Операция свертки широко используется в обработке и анализе изображений, в частности, она составляет основу сверхточных нейронных сетей (СНС) [4]. Рассмотрим процесс прямого распространения данных по СНС в сверточном слое. На вход в слой поступает объект 4-го ранга, будем считать, что первые два быстрых индекса определяют изображение (матрицу), третий — номер входного канала, четвертый — номер объекта в подмножестве тренировочной выборки. В качестве параметров (весов) слоя выступают элементы ядра, также объекта 4-го ранга. Первые два быстрых индекса ядра отвечают его размерам, третий — номеру входного канала, четвертый — номер выходного канала. Тогда свертка в индексах может быть записана следующим образом:

$$O\left(h_{\text{out}}, w_{\text{out}}, c_{\text{out}}, b\right) = \\ = \sum_{k_h=0}^{K_h-1} \sum_{k_w=0}^{K_w-1} \sum_{c_{\text{in}}}^{c_{\text{in}}-1} I\left(h_{\text{out}} + k_h, w_{\text{out}} + k_w, c_{\text{in}}, b\right) \\ \cdot K\left(k_h, k_w, c_{\text{in}}, c_{\text{out}}\right) \tag{2}$$

Операция свертки может быть представлена как умножение двух матриц $I \cdot K$, где элементы первой матрицы — входные изображения, а второй — ядра свертки. Для случая тренировочной выборки из двух изображений двух входных и двух выходных каналов получаем следующею матричную запись:

$$I \cdot K = (I_{00} I_{01} I_{10} I_{11}) \cdot (K_{00} K_{01} K_{10} K_{11}) = \\ = (I_{00} * K_{00} + I_{01} * K_{10} I_{00} * K_{10} + I_{01} * K_{11} I_{10} * K_{00} \\ + I_{11} * K_{10} I_{10} * K_{10} + I_{11} * K_{11}) \tag{3}$$

где * — операция свертки.

Для того чтобы реализовать механизм прямого распространения в CHC, понадобится только определить специализацию оператора * для шаблона Matrix<T>, который выступает в роли изображения, и вызвать оператор * для шаблона Matrix<Matrix<T>>.

Представленный пример демонстрирует способность шаблонов к специализации кода общего назначения. Таким образом, мы хотим акцентировать внимание на том, что класс Matrix и система классов для расчетов на GPU предоставляют обширный функционал вне зависимости от прикладной области разработчика.

DevMatrix под управлением matrixManagerDevice

В ходе тестирования «автономной» реализации [2] выяснилось, что при исполнении итерационных алгоритмов с большими размерностями матриц возникают резкие изменения в зависимости производительности от размерности системы, связанные с особенностями работы сборщика мусора видеокарты. По этой причине была разработана система классов, управляемая matrixManagerDevice (см. рис. 2).

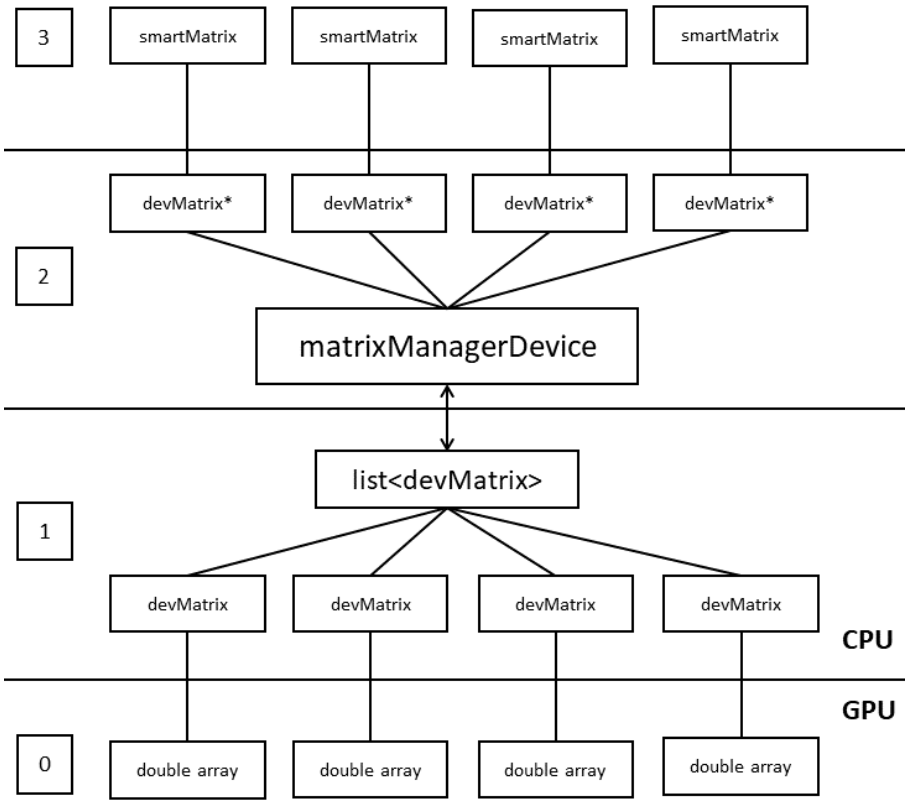

Рисунок 2 - Блок схема GPU части библиотеки

DOI: https://doi.org/10.60797/IRJ.2025.160s.35.2

Класс matrixManagerDevice берет на себя ответственность за управление памятью. В его полях объявлен двухсвязный список объектов типа devMatrix, в свою очередь эти объекты хранят данные о матрицах, расположенных на видеокарте.

Объекты класса devMatrix имеют два состояния: первое отвечает логике активного объекта, который на данный момент участвует в вычислениях, второе состояние, напротив, соответствует «скрытой» матрице, которая готова для взаимодействия, но в данный момент в вычислениях не участвует. В случае инициализации нового объекта верхнего уровня архитектуры «скрытая» матрица проходит процедуру переинициализации и подстраивается под объект верхнего уровня.

Таким образом, в итоговой библиотеке для программиста доступно два уровня архитектуры. Первый уровень требует от программиста, понимания организации внутренней структуры библиотеки и подразумевает ручное управлением состоянием матриц, как следствие, памятью. Он предназначен для реализации высокопроизводительных алгоритмов. На его базе построены различные методы решения СЛАУ.

Второй уровень архитектуры или же высокий уровень абстракции реализован через вспомогательный класс smartMatrix, отвечающий концепции умных указателей. Класс smartMatrix обеспечивает связность библиотеки и простой синтаксис. Данный уровень предназначен для формирования математических моделей и структурных алгоритмов.

Повторное тестирование уже с применением системы классов под управлением matrixManagerDevice, показало, что резкое изменение в производительности больше не возникает, также наблюдается общий прирост скорости вычислений (см. рис. 3).

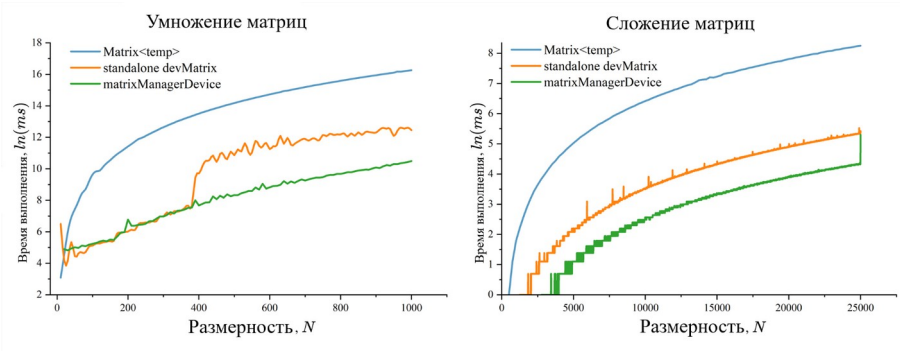

Рисунок 3 - Зависимость между временем работы алгоритма и размерностей матриц операндов
DOI: https://doi.org/10.60797/IRJ.2025.160s.35.3

Стоит отметить, что график зависимости времени вычислений от размерности задачи в области малых размерностей имеет ступенчатый вид, связано это с перераспределением ресурсов на блоках сетки.

Программирование нескольких устройств

Описанная в предыдущем параграфе концепция matrixManagerDevice базируется на монопольном управлении ресурсами видеокарты (устройства) одним классом. Это неявно подразумевает, что класс matrixManagerDevice должен соответствовать паттерну проектирования Singleton. В действительности оказывается, что использование паттерна Singleton — это надежное решение, которое обеспечивает корректный учет памяти и эффективную загрузку GPU [5], однако, это также препятствует развитию идеи вычисления на нескольких устройствах. Чтобы совместить возможность выполнения вычислений на нескольких устройствах и представленную архитектуру, можно воспользоваться шаблонной специализаций.

При компиляции кода на языке программирования C++ компилятор преобразует шаблонный код в отдельные экземпляры шаблона, использующие различные типы. Экземпляры шаблона не взаимосвязаны между собой и воспринимаются как уникальные классы. По этой причине, если изменить реализацию статической функции, которая обеспечивает выдачу ссылки на объект matrixManagerDevice, на шаблонную с одним параметром — беззнаковое целое число, соответствующее номеру устройства, получится механизм разделения классов matrixManagerDevice, отвечающих разным устройствам, на уровне компиляции. В качестве промежуточного звена или транспортера данных выступает шаблонный класс Matrix<T>.

Разреженные матрицы

Рисунок 4 - Формат CSR
DOI: https://doi.org/10.60797/IRJ.2025.160s.35.4

В качестве формата хранения разреженных матриц выбран CSR (сжатая разреженная строка) [6]. Формат CSR предполагает сопоставлять разреженной матрице три одномерных массива — сумм числа элементов строк, индексы столбцов и значения ненулевых элементов (см. рис. 4).

Разреженные матрицы реализованы как подчиненный matrixManagerDevice класс devSparseMatrix. Однако в реализации отсутствует концепция «скрытых» матриц, связано это с тем, что даже в рамках одной задачи размерности массивов Cumulates, Columns и Values для разных матриц могут сильно отличаться.

Применение разреженных матриц несут за собой как однозначно положительные моменты, так и отрицательные. К положительным моментам относятся: сокращение требований к глобальной памяти, уменьшается необходимый объем для хранения матриц (4). Например, матрица размера миллион на миллион с 10 ненулевыми элементами в строке в плотном формате занимает около 7.3 терабайта, тогда как в формате CSR 118.26 мегабайта. Также, что немаловажно, уменьшается число запросов к глобальной памяти устройства, что увеличивает скорость умножения матриц, как следствие скорость решения СЛАУ.

$$Size = \mathrm{N}(int) + \mathrm{V}(int) + \mathrm{V}(double) \tag{4}$$

К недостаткам стоит отнести скорость построения матриц. Формат CSR подразумевает пересчет сумм числа ненулевых элементов для построения строк. Таким образом реализовать алгоритмы построения произвольных разреженных матриц с помощью функциональных объектов можно только в последовательном исполнении. Чтобы использовать GPU, необходимо для каждой математической модели строить уникальные алгоритмы, опираясь на особенности матриц.

Частично компенсирует описанный выше недостаток сохранение разреженных матриц в бинарные файлы. Чтение данных занимает сравнительно меньше времени.

Решение СЛАУ

В качестве инструментария библиотеки реализованы следующие алгоритмы решения систем линейных уравнений:

1) метод Крамера (CPU + GPU);
2) метод на основе LU-разложения (CPU + GPU, SparseMatrix) [7], [8];
3) метод покоординатного спуска (GPU);
4) метод градиентного спуска (CPU + GPU);
5) метод сопряженных градиентов (CPU + GPU, SparseMatrix) [9];
6) стабилизированный метод бисопряженных градиентов (CPU + GPU, SparseMatrix) [10].

В данной работе внимание акцентируется на методах 3-6, т. е. методах основанных на идеи минимизации функционала:

$$\mathrm{F(x)} = (\mathrm{A\ x},\ \mathrm{x}) - 2(\mathrm{b},\ \mathrm{x}) \tag{5}$$

Где A — основная матрица системы, b — вектор-столбец свободных членов. Для методов покоординатного спуска (CDM), градиентного спуска (GDM), сопряженных градиентов (CGM) на матрицу A накладывается условие положительной определенности и симметричности в случае вещественной задачи и положительной определености и самосопряженности в комплексной. Для стабилизиванного метода бисопряженных градиентов (BiCGStab) на основую матрицу системы не накладываются ограничения, помимо невырожденности, что делает этот метод более универсальным.

Решение параболического дифференциального уравнения

Одним из традиционных направлений применения методов решения СЛАУ, основанных на минимизации функционала $\mathrm{F(x)}$, является метод конечных разностей в случае [11], [12], когда производные заменяются неявными разностными схемами.

Рассмотрим однородное двумерное параболическое уравнение:

$$\left\{ \frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial x^2}\right), u^0_{m,k} = \psi_{m,k}, u^n_{0,k} = \varphi^n_k, u^n_{1,k} = \widetilde{\varphi}^n_k, u^n_{m,0} = \xi^n_m, u^n_{m,1} = \widetilde{\xi}^n_m \right. \tag{7}$$

В случае неявной разностной схемы, иттерационная формула будет иметь вид:

$$-\sigma u_{m-1,k}^{n+1} - \sigma u_{m+1,k}^{n+1} + (4\sigma + 1)u_{m,k}^{n+1} - \sigma u_{m,k-1}^{n+1} - \sigma u_{m,k+1}^{n+1} = u_{m,k}^{n} \quad (8)$$

где $\sigma = \frac{D\tau}{h^2}$ — параболический аналог числа Куранта. Если применить формулу (8) ко всем узлам двумерной решетки, получится система линейных уравнений $\mathrm{A\ U{=}F}$:

$$A = (KS : 00SK : 00 \cdot \cdots \cdot$$
$$= (4\sigma + 1 - \sigma 0 - \sigma 4\sigma + 1 - \sigma 0 - \sigma 4\sigma + 1), \quad (9)$$
$$S = (-\sigma 000 - \sigma 000 - \sigma).$$

Основная матрица системы положительно определена и имеет пяти-диагональный вид (трех-диагональный для одномерного случая и семи-диагональный для трехмерного). Размеры подматриц S и K определяются на основании количества узлов в координатной сетке. Столбец свободных членов включает граничные и начальные условия задачи.

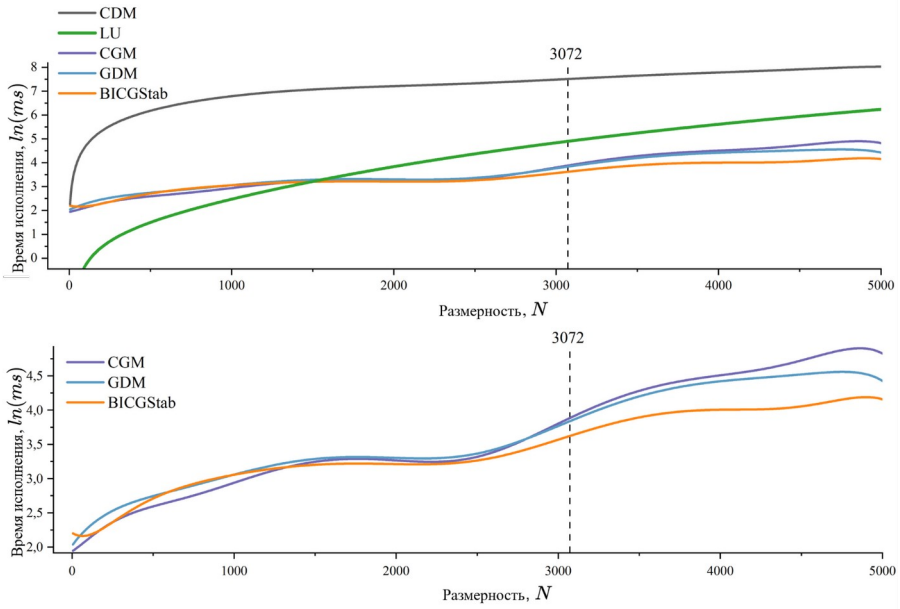

Рисунок 5 - Зависимость между временем работы алгоритмов решения и размерности задачи
DOI: https://doi.org/10.60797/IRJ.2025.160s.35.5

В процессе решения параболического уравнения [13] неявной разностной схемой, проводилось тестирование производительности библиотеки (см. рис. 5).

В рамках задачи о решении параболического уравнения, самым производительным методом решения СЛАУ неявной разностной схемы является стабилизированный метод бисоппряженных градиентов, схожий качественный уровень производительности имеют методы градиентного спуска и сопряженных градиентов. Метод основанный на LU-разложении основной матрицы системы согласуется с оценочной сложностью $O\left(n^3\right)$, на малых размерностях за счет параллелизации графическим процессором одна степень сложности снимается. Метод покоординатного спуска показал наихудший результат и далее рассматриваться не будет. Под временем работы алгоритма решения СЛАУ здесь и далее понимается время, связывающее первую и последнюю итерацию метода, критерием остановки итерационных алгоритмов принята величина модуля вектора невязки (10) [14].

$$||r|| = ||Ax - B|| < \varepsilon \quad (10)$$

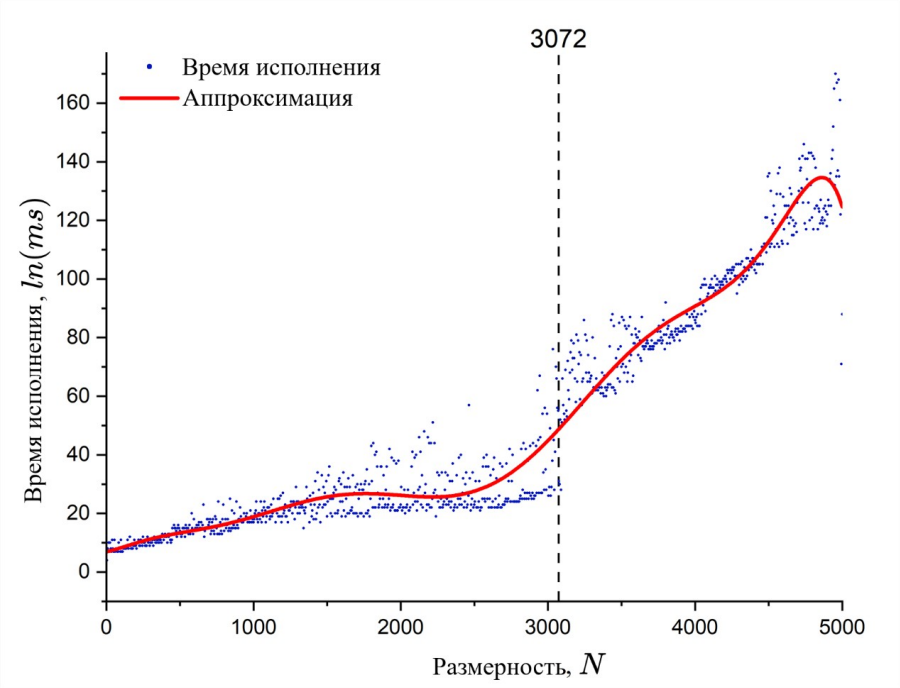

Рисунок 6 - Зависимость между временем работы алгоритма CGM и размерности задачи
DOI: https://doi.org/10.60797/IRJ.2025.160s.35.6

Рассмотрим более подробно результат тестирования метода сопряженных градиентов (см. рис. 6).

При тестировании было замечено, что на размерности 3072 = 3 · 1024 происходит значительный скачок времени решения СЛАУ методами, основанными на минимизации функционала (5). Скачок возникает из-за совокупности факторов, которые относятся как к аппаратной, так и программной части. В первую очередь на размерностях кратных 1024 происходит переопределение числа блоков, запускаемых на устройстве, в таких алгоритмах, как умножение матриц [15]. Причем чем ближе размерность к числу, кратному 1024 справа, тем равномернее используются ресурсы устройства, с другой стороны, при размерностях близких к кратным 1024 слева ресурсы устройства используются неравномерно. Во вторую очередь скачок связан с тем, что матрицы перестают помещаться в кэш L2 графического процессора [16], возникает необходимость в многократном обращении к глобальной памяти, что снижает производительность.

Решение СЛАУ с плотной матрицей

Предлагаем рассмотреть противоположную ситуацию, т.е. решения систем с плотной матрицей (см. рис. 7).

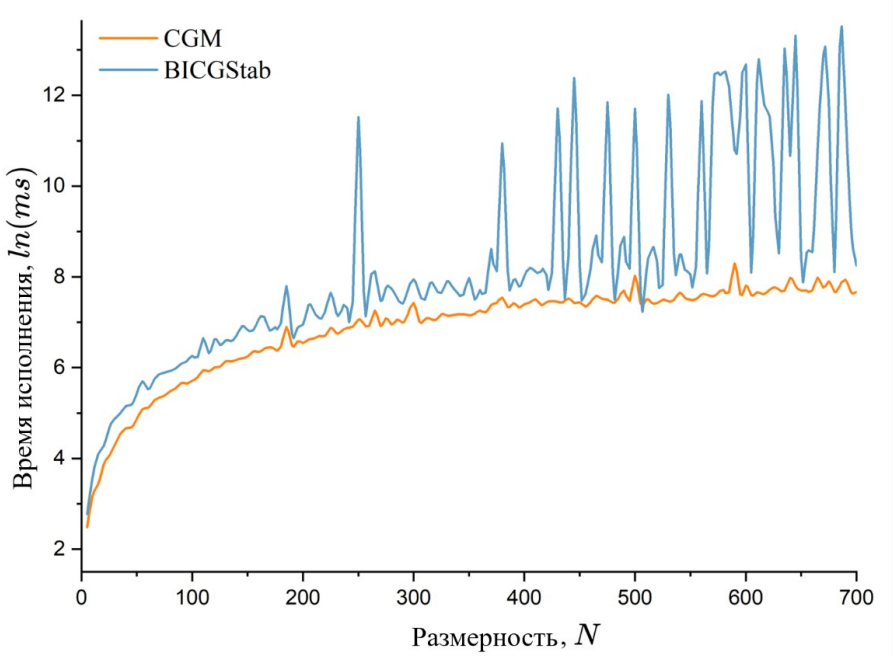

Рисунок 7 - Зависимость между временем работы алгоритмов минимизации и размерностью задачи
DOI: https://doi.org/10.60797/IRJ.2025.160s.35.7

Наблюдается качественно схожая картина со случаем разреженных матриц (см. рис. 5), отличие заключается в порядке времени расчета. Для плотных матриц 500×500 время расчета находится в диапазоне 1.5-2 секунды, в зависимости от метода и критерия остановки, для разреженных матриц диапазон 14–20 мс.

Теоретически метод сопряженных градиентов ассимптотически работает за $O\left(n^3\right)$, но на практике эта оценка не справедлива. Причина плохой сходимости метода сопряженных градиентов, и других методов оптимизаци: GDM, BiCGStab, лежит в неустойчивости шагов минимизации на больших размерностях с плотными матрицами. При вычислении на ЭВМ происходит неизбежная потеря доверительных знаков, соответственно на плотных матрицах достигается наивысшая ошибка, тогда как для разреженных матриц большая часть операций выполняется без ошибки (умножение числа на точный ноль на ЭВМ производится без ошибки, гарантируется стандартом IEEE 754 [17]). Совокупность нестойчивости шагов минимизации и накопительной ошибки при умножении матриц приводит к тому, что методы GDM, CGM, BiCGStab на плотных матриц на практике не применимы.

Заключение

В данной работе описана реализация библиотеки для матричной алгебры, оптимизированной в контексте памяти для решения итерационных задач на графическом процессоре с использованием технологии CUDA. Проведенные тесты показали общее повышение производительности расчетов. Были предложены инструменты для взаимодействия центрального процессора с графическим, а также для программирования матричной алгебры на нескольких GPU.

Кроме того, в работе проведено сравнение различных методов решения систем линейных алгебраических уравнений (СЛАУ), включая методы, основанные на оптимизации. По результатам сравнительных тестов установлено, что методы минимизации применимы только к разреженным матрицам, так как на плотных матрицах из-за вычислительных ошибок они ведут себя неустойчиво. Это открывает перспективы для дальнейших исследований и оптимизаций в области вычислительной линейной алгебры, особенно в контексте использования гибридных систем CPU-GPU и разработки более устойчивых алгоритмов для работы с плотными матрицами.

Финансирование

Работа выполнена на основе гранта Министерства образования РФ № FEUZ-2023-0013.

Конфликт интересов

Не указан.

Рецензия

Все статьи проходят рецензирование. Но рецензент или автор статьи предпочли не публиковать рецензию к этой статье в открытом доступе. Рецензия может быть предоставлена компетентным органам по запросу.

Funding

The work was carried out on the basis of a grant from the Ministry of Education of the Russian Federation No. FEUZ-2023-0013.

Conflict of Interest

None declared.

Review

All articles are peer-reviewed. But the reviewer or the author of the article chose not to publish a review of this article in the public domain. The review can be provided to the competent authorities upon request.

Список литературы / References

1. Устюжанин Д.А. Реализация метода сопряженных градиентов на графическом процессоре с применением матричных методов решения СЛАУ. / Д.А. Устюжанин, С.С. Пицхелаури, К.А. Некрасов // Международный научно-исследовательский журнал. — 2024. — 5 (143) .

2. Крылов В.И. Вычислительные методы в 2 т. / В.И. Крылов, В.В. Бобков, П.И. Монастырный. — Москва: Наука, 1976. — Т. II. — 401 с.

3. CUDA Toolkit Documentation 13.0 // NVIDIA Documentation Hub. — URL: https://docs.nvidia.com/cuda/ (accessed: 05.10.24)

4. Lecun Y. Gradient-Based Learning Applied to Document Recognition. / Y. Lecun, L. Bottou, Y. Bengio et al. // Proceedings of the IEEE. — 1998. — 11. — DOI: 10.1109/5.726791

5. Тумаков Д.Н. Технология программирования CUDA / Д.Н. Тумаков, Д.Е. Чикрин, А.А. Егорчев и др. — Казань: Казанский федеральный ун-т., 2017. — 111 с.

6. Jamalmohammed S.B. Review on Sparse Matrix Storage Formats With Space Complexity Analysis. / S.B. Jamalmohammed, K. Lavanya, I. Sumaiya Thaseen et al. // Applications of Artificial Intelligence for Smart Technology; — Hershey: IGI Global, 2020. — P. 122–145. doi: 10.4018/978-1-7998-3335-2.ch009

7. Lei X. High-Performance Batched LU Decomposition on GPU. / X. Lei, X. Zhang, L. Ma et al. // Applied Mathematics, Modeling and Computer Simulation; — Amsterdam: IOS Press, 2022.

8. Atasoy N.A. Using gauss - Jordan elimination method with CUDA for linear circuit equation systems. / N.A. Atasoy, S. Baha, S. Burhan // Procedia Technology. — 2012. — 1. — P. 31–35. — DOI: 10.1016/j.protcy.2012.02.008

9. Cevahir A. Fast Conjugate Gradients with Multiple GPUs. / A. Cevahir, A. Nukada, S. Matsuoka. // Computational Science — ICCS 2009, 9th International Conference; — New York: Springer, 2009. doi: 10.1007/978-3-642-01970-8_90

10. Lopez G.O. The BiConjugate gradient method on GPUs. / G.O. Lopez, F. Vázquez, E.M. Garzon et al. // The Journal of Supercomputing. — 2013. — 64(1). — P. 49–58. — DOI: 10.1007/s11227-012-0761-2

11. Локтионов И.К. Численные методы / И.К. Локтионов. — Москва: Инфа-Инженерия, 2022. — 308 с.

12. Огородникова О.М. Вычислительные методы в компьютерном инжиниринге / О.М. Огородникова. — Екатеринбург: Изд-во Уральского ун-та, 2013. — 130 с.

13. Амосов А.А. Вычислительные методы для инженеров / А.А. Амосов, Ю.А. Дубинский, Н.В. Копченова. — Москва: Высш. шк, 1994. — 543 с.

14. Пирумов У.Г. Численные методы / У.Г. Пирумов, В.Ю. Гидаспов, И.Э. Иванов. — Москва: Юрайт, 2023. — 421 с.

15. Снытников А.В. Математическое моделирование и программная модель CUDA / А.В. Снытников, А.С. Колганов, Н.Н. Попова. — Москва: МАКС Пресс, 2018. — 171 с.

16. Тоуманен Б. Программирование GPU при помощи Python и CUDA / Б. Тоуманен. — Москва: ДМК Пресс, 2020. — 235 с.

17. IEEE 754 // Wikipedia. — 2003. — URL: https://en.wikipedia.org/wiki/IEEE_754. (дата обращения: 12.10.24)

Список литературы на английском языке / References in English

1. Ustyuzhanin D.A. Realizaciya metoda sopryazhenny'x gradientov na graficheskom processore s primeneniem matrichny'x metodov resheniya SLAU [Implementation of the conjugate gradient method on a GPU using matrix methods for solving SLAE]. / D.A. Ustyuzhanin, S.S. Piczxelauri, K.A. Nekrasov // International Research Journal. — 2024. — 5 (143) . [in Russian]

2. Kry'lov V.I. Vy'chislitel'ny'e metody' v 2 t. [Computational methods in 2 v.] / V.I. Kry'lov, V.V. Bobkov, P.I. Monasty'rny'j. — Moscow: Nauka, 1976. — Vol. II. — 401 p. [in Russian]

3. CUDA Toolkit Documentation 13.0 // NVIDIA Documentation Hub. — URL: https://docs.nvidia.com/cuda/ (accessed: 05.10.24)

4. Lecun Y. Gradient-Based Learning Applied to Document Recognition. / Y. Lecun, L. Bottou, Y. Bengio et al. // Proceedings of the IEEE. — 1998. — 11. — DOI: 10.1109/5.726791

5. Tumakov D.N. Texnologiya programmirovaniya CUDA [CUDA Programming Technology] / D.N. Tumakov, D.E. Chikrin, A.A. Egorchev et al. — Kazan': Kazanskij federal'ny'j un-t., 2017. — 111 p. [in Russian]

6. Jamalmohammed S.B. Review on Sparse Matrix Storage Formats With Space Complexity Analysis. / S.B. Jamalmohammed, K. Lavanya, I. Sumaiya Thaseen et al. // Applications of Artificial Intelligence for Smart Technology; — Hershey: IGI Global, 2020. — P. 122–145. doi: 10.4018/978-1-7998-3335-2.ch009

7. Lei X. High-Performance Batched LU Decomposition on GPU. / X. Lei, X. Zhang, L. Ma et al. // Applied Mathematics, Modeling and Computer Simulation; — Amsterdam: IOS Press, 2022.

8. Atasoy N.A. Using gauss - Jordan elimination method with CUDA for linear circuit equation systems. / N.A. Atasoy, S. Baha, S. Burhan // Procedia Technology. — 2012. — 1. — P. 31–35. — DOI: 10.1016/j.protcy.2012.02.008

9. Cevahir A. Fast Conjugate Gradients with Multiple GPUs. / A. Cevahir, A. Nukada, S. Matsuoka. // Computational Science — ICCS 2009, 9th International Conference; — New York: Springer, 2009. doi: 10.1007/978-3-642-01970-8_90

10. Lopez G.O. The BiConjugate gradient method on GPUs. / G.O. Lopez, F. Vázquez, E.M. Garzon et al. // The Journal of Supercomputing. — 2013. — 64(1). — P. 49–58. — DOI: 10.1007/s11227-012-0761-2

11. Loktionov I.K. Chislenny'e metody' [Numerical methods] / I.K. Loktionov. — Moscow: Infa-Inzheneriya, 2022. — 308 p. [in Russian]

12. Ogorodnikova O.M. Vy'chislitel'ny'e metody' v komp'yuternom inzhiniringe [Computational methods in computer engineering] / O.M. Ogorodnikova. — Ekaterinburg: Izd-vo Ural'skogo un-ta, 2013. — 130 p. [in Russian]

13. Amosov A.A. Vy'chislitel'ny'e metody' dlya inzhenerov [Computational methods for engineers] / A.A. Amosov, Yu.A. Dubinskij, N.V. Kopchenova. — Moscow: Vy'ssh. shk, 1994. — 543 p. [in Russian]

14. Pirumov U.G. Chislenny'e metody' [Numerical methods] / U.G. Pirumov, V.Yu. Gidaspov, I.E'. Ivanov. — Moscow: Yurajt, 2023. — 421 p. [in Russian]

15. Sny'tnikov A.V. Matematicheskoe modelirovanie i programmnaya model' CUDA [Mathematical modeling and the CUDA software model] / A.V. Sny'tnikov, A.S. Kolganov, N.N. Popova. — Moscow: MAKS Press, 2018. — 171 p. [in Russian]

16. Toumanen B. Programmirovanie GPU pri pomoshhi Python i CUDA [GPU programming using Python and CUDA] / B. Toumanen. — Moscow: DMK Press, 2020. — 235 p. [in Russian]

17. IEEE 754 [IEEE 754] // Wikipedia. — 2003. — URL: https://en.wikipedia.org/wiki/IEEE_754. (accessed: 12.10.24) [in Russian]