Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

—

Институт компьютерных наук и кибербезопасности

**Высшая школа кибербезопасности**

**ЛАБОРАТОРНАЯ РАБОТА №3**

РАСЧЁТЫ ПРИ ПОМОЩИ GPU. CUDA.

по дисциплине «Языки программирования»

Выполнили

студенты гр. 5131001/30002 Мишенев Н. С.

Квашенникова В. М.

<*подпись*>

Руководитель

программист Малышев Е. В.

<*подпись*>

Санкт-Петербург

2024г.

**СОДЕРЖАНИЕ**

[ЦЕЛЬ РАБОТЫ 3](#_Toc1406300369)

[ХОД РАБОТЫ 3](#_Toc1108373093)

[1. Разработанный алгоритм на CPU. 4](#_Toc1898444313)

[2. Распараллеленный алгоритм и соответствующее ему разработанное ядро. 4](#_Toc580908121)

[3. Производительности алгоритмов на CPU и GPU. 6](#_Toc1267911245)

[4. Вычисление количества используемых блоков и потоков. 7](#_Toc316715986)

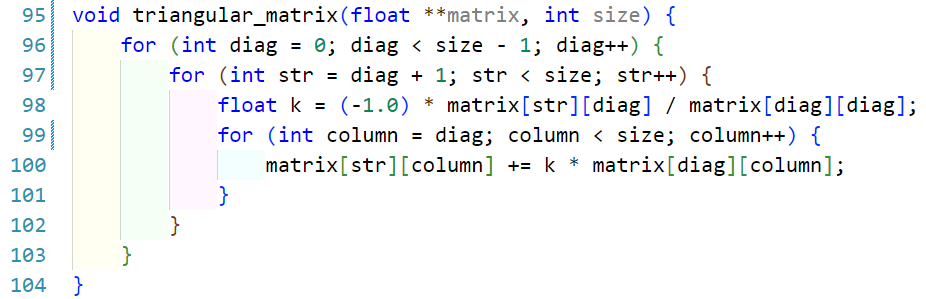
**ЦЕЛЬ РАБОТЫ**

Цель - необходимо научиться выполнять расчеты при помощи GPU (CUDA, OpenCL). Для выполнения необходимо выбрать любую задачу с матрицами из области линейной алгебры (за исключением задач сложения, умножения, вычитания и деления матриц).

# **ХОД РАБОТЫ**

## **1. Разработанный алгоритм на CPU.**

Для выполнения лабораторной работы, метод Гаусса для приведения матрицы к треугольному виду был реализован на ***CPU***:

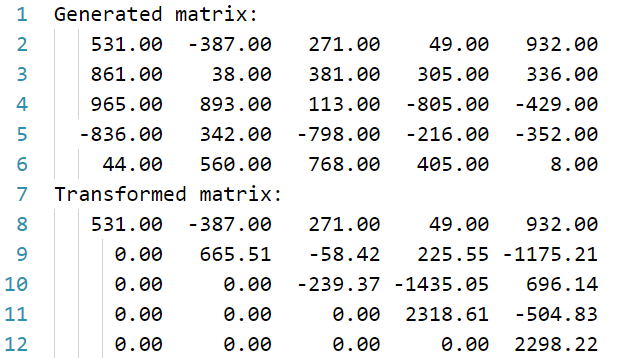


*Рис. 1. Алгоритм приведения матрицы к треугольному виду.*

Алгоритм основан на 2-х основных шагах:

1. Вычисление коэффициента ***k*** (коэффициент обнуления ненулевого элемента под диагональным).
2. Сложение строки с диагональным элементом, умноженной на ***k***, со строкой, содержащей первый ненулевой элемент.

Результат работы алгоритма:

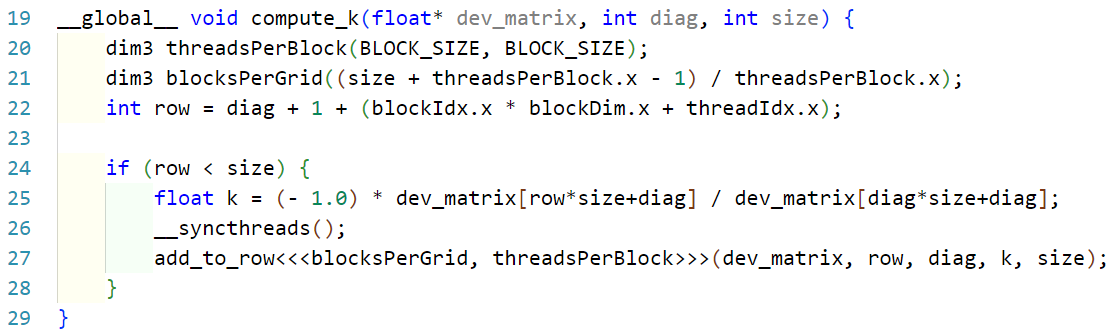


*Рис. 2. Результат работы алгоритма приведения матрицы к треугольному виду.*

## 2. Распараллеленный алгоритм и соответствующее ему разработанное ядро.

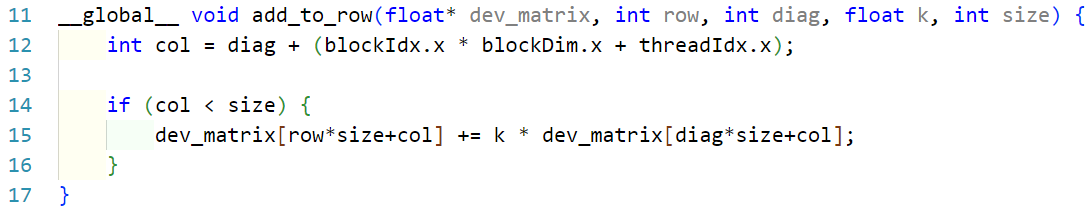
Использование мульти-поточности в данном алгоритме уместно при сложении строк, и вычислении ***k*** для каждого числа под диагональным элементом. Для реализации данной идеи было разработано 2 ядра.

Первое, для вычисления ***k*** для каждой строки. Для каждого элемента под диагональным вычисляется ***k*** и внутри этого ядра вызывается второе ядро, для параллельного сложения строк. В данном ядре реализован первый шаг алгоритма.



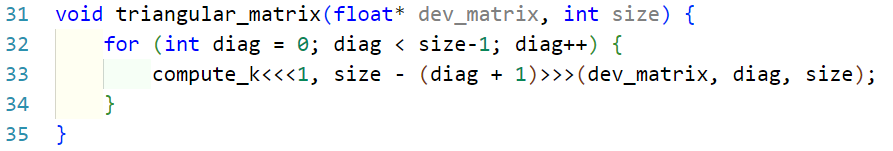
*Рис. 3. Ядро для вычисления k.*

Второе, для сложения строк несколькими процессами. В данном ядре реализован второй шаг алгоритма.



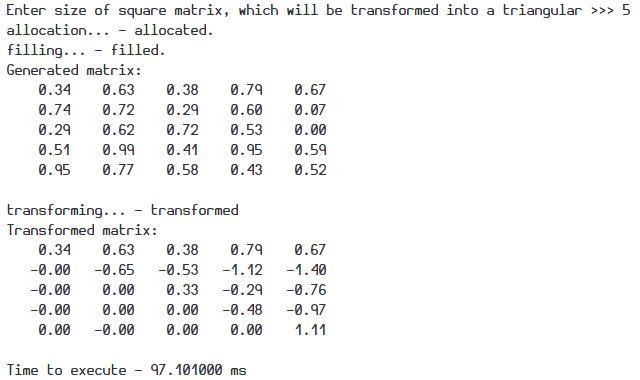
*Рис. 4. Ядро для сложения строк.*

Оба этих ядра вызываются из основной функции ***triangular\_matrix()***, которая и производит приведение матрицы в памяти устройства в треугольную форму.

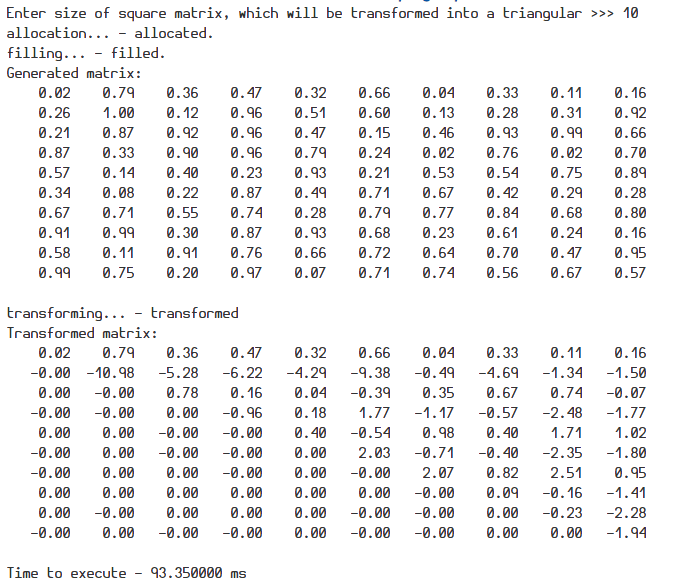


*Рис. 5. Функция triangular\_matrix().*

Результаты работы функции на ***GPU*** с разными размерами матриц:



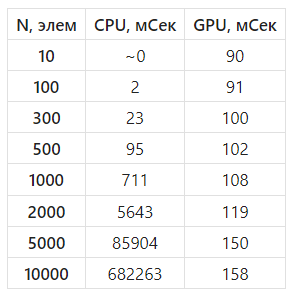
*Рис. 6. Работа функции преобразования для матрицы со стороной 5.*



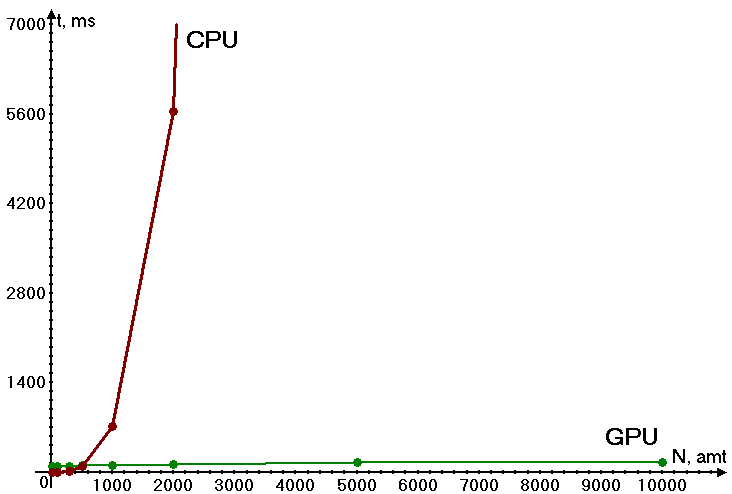
*Рис. 7. Работа функции преобразования для матрицы со стороной 10.*

## Производительности алгоритмов на CPU и GPU.

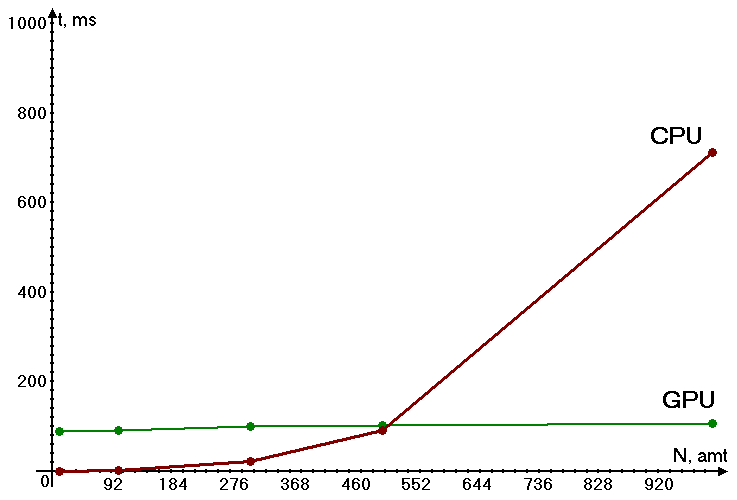
В качестве входных данных возьмем размер входной матрицы, который будем изменять от 10 до 10000 элементов и проверим производительность алгоритмов на CPU и на GPU заполним таблицу и построим графики.



*Рис. 9. Таблица зависимости времени исполнения от количества элементов.*



*Рис. 10. График зависимости времени исполнения от количества элементов.*



*Рис. 11. График вблизи начала координат.*

Из анализа приведенных графиков можно сделать вывод о том, что для маленького количества исходных данных, алгоритм на ***CPU*** показывает себя лучше, однако с ростом размерности входных данных, рост затрачиваемого ***CPU*** времени переходит в экспоненциальную форму, в то время как рост на ***GPU*** чрезвычайно медленен.

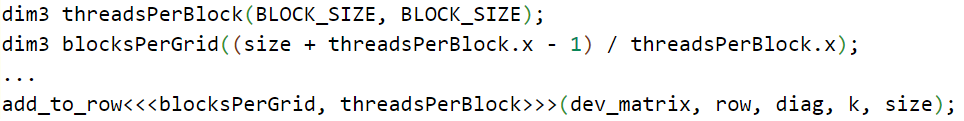
## Вычисление количества используемых блоков и потоков.

Количество используемых блоков и потоков вычисляется следующим образом. Рассмотрим вызов ядра подсчёта коэффициентов ***k***, для которого количество потоков в блоке взято константным. Вызывается 1 блок, количество потоков в нём равно количеству элементов под диагональным элементом.



*Рис. 12. Вызов ядра подсчёта k.*

Рассмотрим вызов ядра для сложения строк. Количество потоков в блоке константно и равно 256 потокам, так как такое количество потоков в блоке (128-256) рекомендовано в документации Nvidia. Количество блоков же вычисляется динамически в зависимости от размерности матрицы так, чтобы количество блоков покрывало полностью необходимое количество вызовов.



*Рис. 13. Вызов ядра сложения строк.*

# **ВЫВОД**

В ходе лабораторной работы, были изучены методы выполнения расчетов при помощи GPU (CUDA, OpenCL). Для выполнения была выбрана задача приведения матрицы к треугольному виду методом Гаусса. Было разработано два алгоритма, для CPU и для GPU с использованием CUDA.

**ПРИЛОЖЕНИЕ А**

Репозиторий с файлами лабораторной работы - [репозиторий на GitHub](https://github.com/clowixdev/pl_lab3)