МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №4**

**по курсу «Программирование графических процессоров»**

**Работа с матрицами. Метод Гаусса.**

Выполнил: Семин А. В.

Группа: 8О-406Б

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2023

**Условие**

1. **Цель работы.** Использование объединения запросов к глобальной памяти. Реализация метода Гаусса с выбором главного элемента по столбцу. Ознакомление с библиотекой алгоритмов для параллельных расчетов Thrust. Использование двухмерной сетки потоков. Исследование производительности программы с помощью утилиты nvprof.
2. **Вариант 3. Решение квадратной СЛАУ.**

Необходимо решить систему уравнений Ax = b, где A - квадратная матрица n x n, b - вектор-столбец свободных коэффициентов длинной n, x - вектор неизвестных.

**Входные данные**. На первой строке задано число n - размер матрицы. В следующих n строках, записано по n вещественных чисел -- элементы матрицы. Далее записываются n элементов вектора свободных коэффициентов. n ≤ 104

**Выходные данные**. Необходимо вывести n значений, являющиеся элементами вектора неизвестных x.

**Программное и аппаратное обеспечение**

**Графический процессор (GeForce GTX 1650 Ti)**

1. Количество потоковых процессоров: 1024
2. Частота ядра: 1350 МГц
3. Частота в режиме Boost: 1485 МГц
4. Количество транзисторов: 6,600 млн
5. Тип памяти: DDR6
6. Видеопамять: 4096 МБ
7. Частота памяти: 12000 МГц

**Процессор AMD Ryzen 7 4800H**

1. ядра: 8
2. потоки: 16
3. частота: 2.9 ГГц
4. максимальная частота: 4.2 ГГц
5. кэш 1 уровня: 64 КБ (на ядро)
6. кэш 2 уровня 512 КБ (на ядро)
7. кэш 3 уровня: 8 МБ (общий)

16 ГБ ОЗУ и 512 ГБSSD.

ОS – Windows 11Домашняя, WSL, IDE – VS Code, Compiler - nvcc, g++.

**Метод решения**

Входные данные считываются из стандартного потока, на их основании заполняется искомая матрица. Матрица хранится в линейном массиве. Доступ к элементу осуществляется в виде matrix[j\*n + i], где j – номер столбца, i – номер строки. Такое представление используется для хранения матрицы «по столбцам», что упрощает реализацию метода Гаусса.

После инициализации матрица копируется на GPU, а затем происходит реализация метода Гаусса также на GPU.

Алгоритм заключается в преобразовании матрицы в верхнетреугольную с помощью метода Гаусса. На каждой итерации метода берется максимальный элемент в столбце из позиций (i, i) – (n, i) – От элемента на диагонали до нижней строки. Если максимальный элемент не совпадает с текущей позицей (i, i), то происходит замена строк местами. Далее производим сложение каждой из строк под текущей с текущей строкой по определенной формуле. Таким образом, элемент под текущим диагональным элементом зануляется.

После выполнения метода Гаусса решаем результат копируется на GPU, а с помощью полученной матрицы осуществляется решение квадратичной системы уравнений. Представление матрицы в верхнетреугольном виде позволяет решить уравнение за квадратичную сложность простым выражением каждого из X-ов, начиная с нижней строки. Решение уравнения происходит на CPU.

**Описание программы**

Программа состоит из одного файла. В нем реализованы следующие функции:

\_\_host\_\_ void readMatrixAndB(double\* matrix, int n) – считывание входных данных;

\_\_global\_\_ void swapRows(double\* data, int n, int curRow, int rowToSwap) – замена строк местами, реализованная на одномерной сетке потоков;

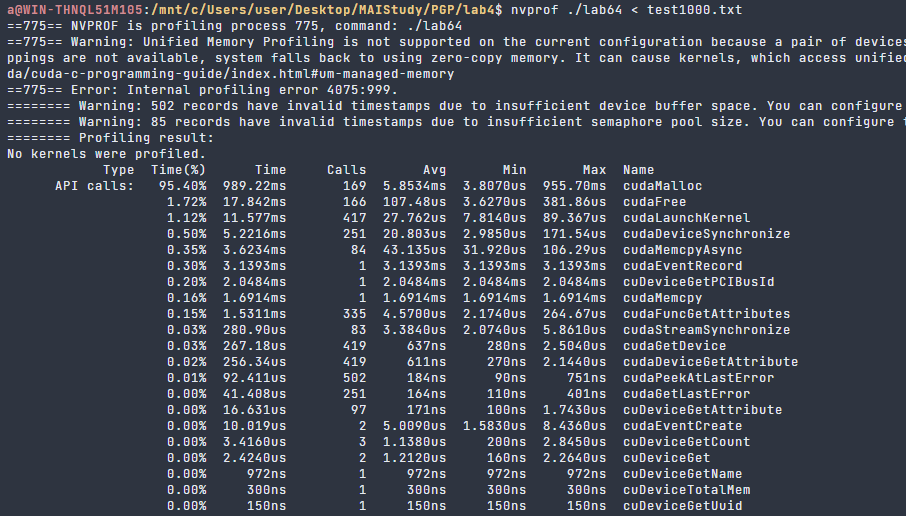
\_\_global\_\_ void Gauss(double\* data, int n, int row) – метод Гаусса, реализованный на двумерной сетке потоков;

\_\_host\_\_ void solveEquatation(double\* matrix, double\* res, int n) – решение квадратичной системы уравнений.

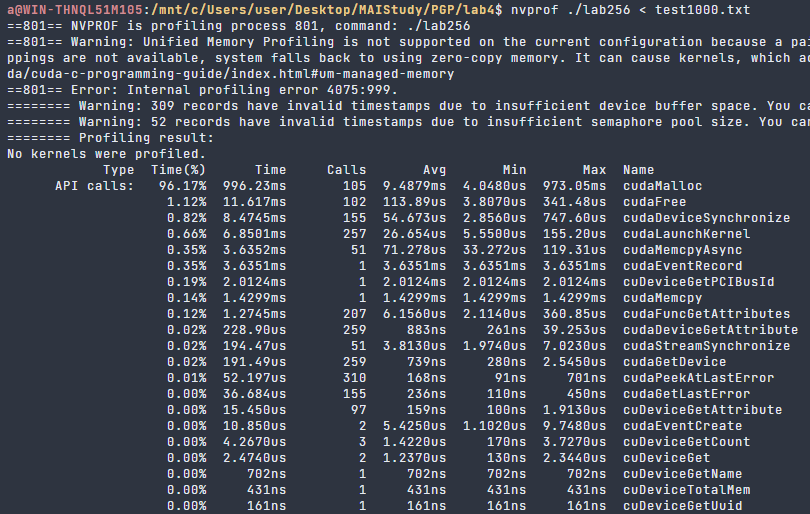
**Использование утилиты nvprof**

Исследуем производительность программы утилитой для матрицы 1000 \* 1000.

Результат для конфигурации *swapRows<<<64,64>>>(…)* и *Gauss<<<dim3(32,32), dim3(16,16)>>>(…)*:



Результат для конфигурации *swapRows<<<256,256>>>(…)* и *Gauss<<<dim3(64, 64), dim3(32, 32)>>>(…)*:



**Результаты**

Время выведено в ***миллисекундах***.

В каждой ячейке 1-ая конфигурация – функция swapRows(…), вторая конфигурация – Gauss(…).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Конфигурация | Размер тестового файла | | | | |
|  |  |  |  |  |
| CPU | 3 | 7 | 4127 | 4.04515 \* 105 | > 6\*105 |
| <<<(16, 16)>>>,  <<<(16, 16), (16, 16)>>> | 1.374912 | **1.319232** | **11.667744** | 1015.581482 | 126497.867188 |
| <<<(64, 64) >>>,  <<<(32, 32), (16, 16)>>> | **0.434176** | 1.512800 | 11.868640 | 1020.329590 | **124948.343750** |
| <<<(128, 128) >>>  <<<(32, 32), (32, 32)>>> | 0.473920 | 1.522048 | 13.062144 | 1003.864075 | 125633.125000 |
| <<<(256, 256) >>>  <<<(64, 64), (32, 32)>>> | 0.536416 | 2.046048 | 17.113632 | **840.539368** | 127811.914062 |
| <<<(512, 512) >>>  <<<(64, 64), (32, 32)>>> | 0.452736 | 2.095424 | 17.721151 | 1025.087036 | 128407.757812 |
| <<<(1024,1024) >>>  <<<(128, 128), (32, 32)>>> | 0.833536 | 3.933632 | 36.483391 | 1052.189697 | 137199.546875 |

**Выводы**

Метод Гаусса – алгоритм, приводящий матрицу к верхнетреугольному виду, очень распространен и имеет множество практических применений. Такой вид представление матрицы упрощает взаимодействие с ней и позволяет быстрее и проще выполнять различные манипуляции, такие как решение СЛАУ, нахождение обратной матрицы, определение ранга матрицы, вычисление определителя, разложение матрицы и т. д.

Реализовав данный алгоритм на CPU и GPU и проведя сравнение на различных тестовых данных, приходим к выводу, что на GPU этот алгоритм на некоторых тестовых данных работает в десятки, а то и в сотни раз быстрее. А хранение матрицы в линейном массиве по столбцам позволяет более удобным образом реализовать этот метод на GPU.