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 матрицам. Метод Гаусса.

Выполнил: *И. П. Попов*
Группа: *8О-406Б*
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2023

Условие

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 (обязательно отразить в отчете).
В качестве вещественного типа данных необходимо использовать тип данных double. Библиотеку Thrust использовать только для поиска максимального элемента на каждой итерации алгоритма. В вариантах(1,5,6,7), где необходимо сравнение по модулю с нулем, в качестве нулевого значения использовать 10^{-7} . Все результаты выводить с относительной точностью 10^{-10}.

Вариант 2. Вычисление обратной матрицы.
Входные данные. На первой строке задано число n -- размер матрицы. В следующих n строках, записано по n вещественных чисел -- элементы матрицы. $n \leq 10^{-4}$.
Выходные данные. Необходимо вывести на n строках, по n чисел -- элементы обратной матрицы.

Программное и аппаратное обеспечение

NVIDIA GeForce GTX 1660 Super:
Compute capability: 7.5
Dedicated video memory: Typically 6 GB (may vary by manufacturer)
Shared memory per block: 49152 bytes
Register per block: 65536 bytes
Total constant memory: 65536 bytes
Max threads per multiprocessor: 2048
Max threads per block: 1024

CPU AMD Ryzen 3600X
Physical cores: 6
Threads: 12
Base clock speed: 3.8 GHz
Boost clock speed: 4.4 GHz
L1 cache: 384KB (per core)
L2 cache: 512KB (per core)
L3 cache: 32 MB (shared)
Chip lithography: 7 nm

16 Гб RAM
1 ТБ HDD
OS – Windows 11 ProWSL, IDE – VS Code, compiler - nvcc

Метод решения

Программа, использует метод Гаусса с частичным выбором (partial pivoting) для решения системы линейных уравнений и нахождения обратной матрицы.

Описание программы

1. Ввод данных
2. Преобразования методом Гаусса:
    a. На каждом шаге выбирается максимальный по модулю элемент в текущем столбце и производится обмен строк для того, чтобы этот элемент стал диагональным (если он не находится уже на диагонали). Это обеспечивает численную стабильность метода Гаусса, предотвращая деление на очень маленькие или большие числа, что может привести к ошибкам округления
    b. Далее происходит обнуление нижних элементов в текущем столбце матрицы путем вычитания из строк ниже диагонали дополненных к первой строке, умноженных на подходящий коэффициент
    c. Далее обнуляются верхние элементы в текущем столбце путем вычитания из строк выше диагонали дополненных к последней строке, умноженных на подходящий коэффициент
    d. В конце матрица единичной единицы делится на диагональные элементы матрицы, чтобы получить обратную матрицу
3. Вывод результата
4. Освобождение памяти

Единичная матрица используется в этом процессе для хранения обратной матрицы. На каждом этапе преобразования методом Гаусса, матрица unity подвергается тем же операциям, что и входная матрица matrix. Поэтому после завершения преобразований матрица unity содержит обратную матрицу к введенной матрице.

Результаты

Все измерения представлены в ms

	10^2	10^3	2 * 10^3
<<<(32, 16), (32, 16)>>>	1110.930543	6828.510352	56278.400143
<<<(32, 32), (32, 32)>>>	2364.440599	10105.200456	59552.900254
<<< (16, 16), (32, 16)>>>	1073.070237	6994.200799	57650.100695
<<< (32, 16), (16, 16)>>>	1005.000162	6882.090289	65580.200521

<<<(64, 16), (32, 16)>>>	1195.500764	6798.310492	54787.100125
CPU	10.015972	15219.200488	159141.837495

Выводы

Применение параллельных вычислений на GPU с использованием CUDA технологии позволяет эффективно распараллеливать операции над матрицами и векторами.

Из проведенных тестов видно, что вычисления на GPU проходят в 2-3 раза быстрее, чем на CPU. Это связано с параллельной обработкой данных на множестве ядер GPU, что позволяет эффективно использовать ресурсы устройства. Благодаря использованию библиотеки Thrust и оптимизированным алгоритмам параллельных вычислений, удалось уменьшить количество запросов к глобальной памяти, что также положительно сказалось на производительности программы.