МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №2
по курсу «Параллельная обработка данных»

Работа с матрицами. Метод Гаусса.

Выполнил: И. П. Моисеенков
Группа: М8О-408Б-19
Преподаватель: А.Ю. Морозов

Москва, 2022

Условие

Цель работы: исследование объединения запросов к глобальной памяти. Реализация метода Гаусса с выбором главного элемента по столбцу. Ознакомление с библиотекой алгоритмов для параллельных расчетов Thrust. Использование двумерной сетки потоков. Исследование производительности программы с помощью утилиты nvprof.

Вариант 6. Нахождение ранга матрицы:

Входные данные. На первой строке заданы числа n и m - размеры матрицы. В следующих n строках записано по m вещественных чисел - элементы матрицы.

Выходные данные. Необходимо вывести одно число - ранг матрицы.

Программное и аппаратное обеспечение

В качестве графического процессора использую видеокарту Nvidia GeForce GT 545, установленную на сервере преподавателя.

```
Compute capability      : 2.1
Name                    : GeForce GT 545
Total Global Memory     : 3150381056
Shared memory per block : 49152
Registers per block     : 32768
Warp size               : 32
Max threads per block   : (1024, 1024, 64)
Max block   : (65535, 65535, 65535)
Total constant memory   : 65536
Multiprocessors count   : 3
```

В качестве редактора кода использовался Visual Studio Code.

Метод решения

Для нахождения ранга матрицу необходимо привести к треугольному виду. Количество ненулевых строк и будет равняться рангу.

Для приведения к треугольному виду будем использовать метод Гаусса с выбором главного элемента по столбцу (максимального по модулю). Если максимальный по модулю элемент в столбце равен нулю, то пропускаем его. Иначе выполняем итерацию метода Гаусса. Ранг матрицы равен количеству итераций, которые потребовалось совершить (т.е. равен количеству «ступенек» в полученной треугольной матрице).

Параллельно будем выполнять две операции:

1. Поменять две строки местами - тут все тривиально;
2. Выполнить шаг метода Гаусса - для заданной подматрицы из каждой строки вычитаем главную строку, домноженную на соответствующий коэффициент.

Описание программы

В программе реализовано два ядра. Ядро kernel_swap берет в себя две строки, которые нужно поменять местами. Для выполнения этой операции используется одномерная сетка потоков.

Ядро kernel_gauss_step выполняет итерацию метода Гаусса. Ему передается начало подматрицы, в которой нужно применить метод Гаусса. Так мы будем фактически изменять только те элементы матрицы, которые нам понадобятся в дальнейшем. Для выполнения этой операции используется двумерная сетка потоков.

Для нахождения главного элемента по столбцу будем использовать библиотеку Thrust. Для использования нужно указать промежуток, на котором искать максимум. Так как мы ищем максимум по столбцам, то и хранить матрицу будем по столбцам (но в одномерном массиве для удобства).

Результаты

Рассмотрим время работы программы на различных тестах при различных размерах сетки (и без использования графического процессора вообще). Размер сетки будем изменять у ядра, выполняющего шаг метода Гаусса. Будем замерять непосредственно время работы алгоритма. В тестах будем искать ранг матриц разных размеров. Результаты приведены в таблице ниже.

Размер сетки ядра \ Размер матрицы	100 x 100, мс	500 x 500, мс	1000 x 1000, мс
CPU	4.38	270.41	1732.28
<<<(1, 32), (1, 32)>>>	45.55	444.38	2377.53
<<<(32, 32), (1, 32)>>>	37.55	300.77	1590.96
<<<(1, 128), (1, 128)>>>	49.21	510.92	2326.24
<<<(1, 256), (1, 32)>>>	45.12	441.35	2485.03

На маленьких матрицах алгоритм на CPU работает быстрее. Но на матрице 1000x1000 алгоритм на графическом процессоре становится более оптимальным по времени. Я уверен, что на еще больших матрицах выигрыш по времени от использования GPU будет еще выше.

Применим утилиту nvprof для исследования производительности программы. Применять будем на тесте 1000x1000. Приложу результат только для своих ядер (ядра от thrust рассматривать не буду).

```
==32077== Profiling application: ./a.out
==32077== Profiling result:
==32077== Event result:
Invocations                Event Name     Min         Max         Avg
Device "GeForce GT 545 (0)"
        Kernel: kernel_gauss_step(double*, int, int, int, int)
        999          divergent_branch       0       30994        1038
        999  global_store_transaction       3      125268       40544
        999   l1_shared_bank_conflict       0           0           0
        999         l1_local_load_hit       0           0           0

        Kernel: kernel_swap(double*, int, int, int, int)
        990          divergent_branch       1           1           1
        990  global_store_transaction    2304        2304        2304
        990   l1_shared_bank_conflict       0           0           0
        990         l1_local_load_hit       0           0           0
```

Параметр, отвечающий за обращение к глобальной памяти - global_store_transaction. Заметим, что в ядре для метода Гаусса количество обращений к памяти внутри варпа ниже общего количества элементов матрицы. Следовательно,

производится объединение запросов к глобальной памяти для оптимизации программы.

Выводы

В данной лабораторной работе я реализовал метод Гаусса на графическом процессоре и с его помощью научился рассчитывать ранг произвольной матрицы. Эта работа вызвала некоторые затруднения, так как пришлось потратить довольно много времени на тестирование и поиск ошибок.

В ходе выполнения этой работы я познакомился с библиотекой thrust, в которой реализованы многие параллельные алгоритмы на cuda. Также я познакомился с профилировщиком nvprof и попробовал применить его для анализа своей программы.