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: К.М. Воронов
Группа: 8О-407Б
Преподаватель: А.Ю. Морозов

Москва, 2022

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).
Вариант 4. LU-разложение матрицы.

Программное и аппаратное обеспечение

GPU:

- Название NVIDIA GeForce GTX 1050
- Compute capability: 6.1
- Графическая память: 4236378112
- Разделяемая память: 49152
- Константная память: 65536
- Количество регистров на блок: 65536
- Максимальное количество нитей: (1024, 1024, 64)
- Максимальное количество блоков: (2147483647, 65535, 65535)
- Количество мультипроцессоров: 5

Сведения о системе:

- Процессор: Intel Core i5-8300H 2.30GHz
- ОЗУ: 32 ГБ
- SSD 1ТБ
- HDD 1ТБ

Программное обеспечение:

- OS: Kubuntu 20.04
- Текстовый редактор: Sublime text
- Компилятор: nvcc

Метод решения

Разложение состоит из нескольких этапов. Для каждого столбца я сначала выбираю строку с максимальным по модулю элементом в этом столбце, далее я ее двигаю наверх. Потом я считаю коэффициенты преобразования (чтобы занулить все элементы в столбце под максимальным) — они и будут элементами матрицы L, и преобразую остальные на их основе путем сложения строк, умноженных на эти коэффициенты с отрицательным знаком, тем самым получая матрицу U.

Описание программы

Для хранения значений элементов матриц я использую тип и double. В программе присутствует три ядра. Первое(swap_str) меняет местами строки. Вторая (kernel_1) считает коэффициенты матрицы L заданного столбца и записывает их в исходную

матрицу. Третья (kernel_2) выполняет преобразование строк. Матрица хранится по стобцам для объединения запросов обращения в память, это очень ускоряет программу. Для нахождения максимума используется библиотека thrust.

nvprof

Тестирование происходило на матрице 1000x1000 с параметрами:
swap_str<<<512, 512>>>
kernel_1<<<512, 512>>>
kernel_2<<<dim3(128, 128), dim3(32, 32)>>>

```
==11839== Profiling application: ./solution
==11839== Profiling result:
==11839== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1050 (0)"
    Kernel: swap_str(double*, int, int, int)
       1000                          gst_transactions                  Global Store Transactions        2000        2000        2000
    Kernel: kernel_1(double*, int, int)
       1000                          gst_transactions                  Global Store Transactions           0         281         136
    Kernel: kernel_2(double*, int, int)
       1000                          gst_transactions                  Global Store Transactions           0      280719       91021
```

Во время выполнения неправильно шел по индексам в ядре kernel_2, что не приводило к объединению запросов и , соответственно, к увеличению времени работы.
Необъединенные запросы в память в ядре kernel_2

```
==11659== Profiling application: ./solution
==11659== Profiling result:
==11659== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1050 (0)"
    Kernel: kernel_2(double*, int, int)
       1000                          gst_transactions                  Global Store Transactions           0      998001      332833
```

Как видно, обращений в память было намного больше.

CPU:

n	Время, мс
10	0.099 ms
100	8.741 ms
1000	8285.27 ms
5000	1318820 ms

GPU:
<<<1, 32>>>
<<<1, 32>>>
<<< dim3(16, 16), dim3(32, 32) >>>

n	Время, мс
10	0.477184 ms

100	5.076096 ms
1000	197.048157 ms
5000	9771.003906 ms

<<<256, 256>>>
<<<256, 256>>>
<<< dim3(128, 128), dim3(8, 8) >>>

n	Время, мс
10	1.214240 ms
100	13.130016 ms
1000	246.655106 ms
5000	14576.129883 ms

<<< 512, 512 >>>
<<< 512, 512 >>>
<<< dim3(128, 128), dim3(32, 32) >>>

n	Время, мс
10	5.865472 ms
100	57.846306 ms
1000	660.027405 ms
5000	9933.025391 ms

<<<1024, 1024>>>
<<<1024, 1024>>>
<<<dim3(32, 32), dim3(32, 32)>>>

n	Время, мс
10	1.292288 ms
100	12.812320 ms
1000	222.368698 ms
5000	8991.470703 ms

Выводы

Выполнив данную лабораторную, я научился работать с матрицами с использованием технологии CUDA. Во время выполнения столкнулся с ошибкой взятия максимума в

строке, а именно брал не по модулю. Из-за этого, на некоторых тестах, бралось очень маленькое число и при делении на него получалось очень большое, что приводило к переполнению. Также в kernel_2 неправильно шел по индексам, что не приводило к объединению запросов в память.