МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №4**

**по курсу «Программирование графических процессоров»**

**Работа с матрицами. Метод Гаусса.**

Выполнил: А. О. Дубинин

Группа: 8О-407Б

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2020

**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.

**Вариант 6**. **Нахождение ранга матрицы.**

**Программное и аппаратное обеспечение**

**GeForce 940MX**

|  |  |
| --- | --- |
| Compute capability: | 5.0 |
| Dedicated video memory: | 4096 MB |
| shared memory per block: | 49152 bytes |
| constant memory: | 65536 bytes |
| Total number of registers available per block: | 65536 |
| Maximum number of threads per multiprocessor: | 2048 |
| Maximum number of threads per block: | 1024 |
| ( 3) Multiprocessors, (128) CUDA Cores/MP: | 384 CUDA Cores |

**Intel(R) Core (TM) i5-7200U CPU @ 2.50GHz**

|  |  |
| --- | --- |
| Architecture: | x86\_64 |
| Byte Order: | Little Endian |
| CPU(s): | 4 |
| Thread(s) per core: | 2 |
| Core(s) per socket: | 2 |
| CPU MHz:  CPU max MHz:  CPU min MHz: | 713.848  3100,0000  400,0000 |
| L1d cache:  L1i cache:  L2 cache:  L3 cache: | 64 KiB  64 KiB  512 KiB  3 MiB |

|  |  |
| --- | --- |
| RAM | 8GiB SODIMM DDR4 Synchronous Unbuffered (Unregistered) 2400 MHz (0,4 ns) |

|  |  |
| --- | --- |
| SSD(SPCC\_M.2\_SSD) | 223,6G |
| HDD(ST1000LM035-1RK172) | 931,5G |

**OS: Ubuntu 20.04 focal**

**IDE: jetbrains clion**

**compiler: nvcc**

**Метод решения**

Необходимо было реализовать метод Гаусса с выбором главного элемента для моей задачи. Данные мы будем хранить так, чтобы последовательные массивы были столбцы, чтобы можно было искать главный элемент быстро с помощью thrust. На каждой итераций с помощью библиотеки thrust я выбирал максимальный элемент и проверял является ли он нулем, и если нет, то это наш ведущий элемент. Если этот элемент существует, то это ступенька, а значит +1 к рангу матрицы. Далее я менял строки местами на гпу, чтобы можно было продолжать легко искать максимальный элемент в последовательно лежащей памяти с помощью thrust. Потом, я пере вычислил так же на гпу значения строк, которые мы обнулили.

**Описание программы**

Интересная часть программы заключалась в использовании библиотеки thrust.

// get column begin

data\_ptr = device\_pointer\_cast(d\_A + j \* n);

// get max ptr in column  
mx\_ptr = thrust::max\_element(data\_ptr + i, data\_ptr + n, comp);   
  
mx = fabs(\*mx\_ptr);

// get max idx

mx\_idx = mx\_ptr - data\_ptr;

В пересчете значений стоило вычислять нужную формулу, чтобы точно совпадали результаты с чекером.

\_\_global\_\_ void kernel\_change\_lines(double \*A, int row, int col, int n, int m) {  
 int idx = blockDim.x \* blockIdx.x + threadIdx.x; // Абсолютный номер потока  
 int idy = blockDim.y \* blockIdx.y + threadIdx.y; // Абсолютный номер потока  
 int offsetx = blockDim.x \* gridDim.x; // Общее кол-во потоков  
 int offsety = blockDim.y \* gridDim.y; // Общее кол-во потоков  
  
  
 for (int p = col + idx + 1; p < m; p += offsetx) {  
 for (int k = row + idy + 1; k < n; k += offsety) {  
 A[p \* n + k] -=

((A[p \* n + row] / A[col \* n + row]) \* A[col \* n + k]);  
 }  
 }  
}

**Результаты**

Сравним результаты с cpu.

|  |  |  |
| --- | --- | --- |
|  | CPU | GPU |
| 1500x1500 | 5230.2 ms | 2123.9 ms |
| 2000x2000 | 9920.31 ms | 2492.11 ms |

Из результатов видно, что использование GPU совместно с библиотекой thrust существенно ускоряет вычисление на матрицах.

**Вывод**

Данная ЛР была сложна, особенно из индексов матриц, потому что хранение столбцов происходит построчно и запутаться в вычислениях не составляет труда. Так же было главное правильно реализовать формулу, без предварительных вычислений.