МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет «Информационные технологии и прикладная математика»

Кафедра «Вычислительная математика и программирование»

**Лабораторная работа №4**

**по курсу «Программирование графических процессоров»**

**Изучение технологии CUDA**

Выполнил: И.И. Тишин

Группа: 8О-406Б

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2019

**Условие**

Использование объединения запросов к глобальной памяти.

Реализация метода Гаусса с выбором главного элемента по столбцу. Ознакомление с библиотекой алгоритмов для параллельных расчетов Thrust.

В качестве вещественного типа данных необходимо использовать тип данных double. Библиотеку Thrust использовать только для поиска максимального элемента на

каждой итерации алгоритма. 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 ≤ 104.

Выходные данные:

​Необходимо вывести на n строках, по n чисел -- элементы

обратной матрицы.

**Программное и аппаратное обеспечение**

GPU:

Device 0: "GeForce GTX 1060 with Max-Q Design"

CUDA Driver Version / Runtime Version 10.1 / 10.1

CUDA Capability Major/Minor version number: 6.1

Total amount of global memory: 6078 MBytes (6373572608 bytes)

(10) Multiprocessors, (128) CUDA Cores/MP: 1280 CUDA Cores

GPU Max Clock rate: 1480 MHz (1.48 GHz)

Memory Clock rate: 4004 Mhz

Memory Bus Width: 192-bit

L2 Cache Size: 1572864 bytes

Maximum Texture Dimension Size (x,y,z) 1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)

Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers

Maximum Layered 2D Texture Size, (num) layers 2D=(32768, 32768), 2048 layers

Total amount of constant memory: 65536 bytes

Total amount of shared memory per block: 49152 bytes

Total number of registers available per block: 65536

Warp size: 32

Maximum number of threads per multiprocessor: 2048

Maximum number of threads per block: 1024

Max dimension size of a thread block (x,y,z): (1024, 1024, 64)

Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)

Maximum memory pitch: 2147483647 bytes

Texture alignment: 512 bytes

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 10.1, CUDA Runtime Version = 10.1, NumDevs = 1

CPU:

Архитектура: x86\_64

CPU op-mode(s): 32-bit, 64-bit

CPU(s): 12

Потоков на ядро: 2

Ядер на сокет: 6

Имя модели: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz

CPU max MHz: 4100,0000

CPU min MHz: 800,0000

BogoMIPS: 4416.00

Виртуализация: VT-x

L1d cache: 32K

L1i cache: 32K

L2 cache: 256K

L3 cache: 9216K

Ram:

32 GiB, 2666 MHz

HDD:

500 GB

Программное обеспечение:

Ubuntu 18.04.2 LTS

g++ (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0

nvcc: release 10.1, V10.1.105

**Метод решения**

Записать матрицу (A|E), где А – исходная матрица, Е – единичная. С помощью метода Гаусса привести матрицу А к верхнетреугольному виду (все преобразования над строками матрицы А также применяются к матрице Е). Используя метод Гаусса в обратную сторону матрицы А, привести её к диагональному виду. Разделить каждую строку матрицы Е на диагональный элемент матрицы А. В итоге получим матрицу (Е|B), где Е – единичная, а В – обратная к А.

**Описание программы**

Программа состоит из функций: main, my\_swap, normalization,kernel . Также используется компаратор для сравнения чисел модулем thrust.

В main считываются данные со стандартного ввода, заполняется массив данными об исходной матрице. Матрица хранится в памяти в транспонированном виде, для удобного использования функции thrust::max\_element.. В этой функции данные копируются в gpu память. Далее запускается цикл псевдо обнуления поддиагональных и наддиагональных элементов матрицы: находим максимальный элемент в столбце (строке, поскольку храним в транспонированном виде) и, если нужно, меняем местами столбцы (вызывается функция my\_swap). После вызывается функция normalization, для нормализации строки.в kernel происходит зануления очередного столбца методом Гаусса. После всех операций данные копируются на cpu и выводятся на стандартный поток вывода.

\_\_global\_\_ void my\_swap(double\* data,double\* E, int n,int x,int y) {

int idx = threadIdx.x + blockIdx.x \* blockDim.x; // Индекс нити

int offset = gridDim.x \* blockDim.x; // кол-во блоков \* размер блока

int i;

double tmp;

for(i=idx;i<n;i+=offset){

tmp=data[i\*n+x];

data[i\*n+x]=data[i\*n+y];

data[i\*n+y]=tmp;

tmp=E[i\*n+x];

E[i\*n+x]=E[i\*n+y];

E[i\*n+y]=tmp;

}

}

\_\_global\_\_ void normalization(double\* data,double\* E, int n,int i){

int idx = threadIdx.x + blockIdx.x \* blockDim.x; // Индекс нити

int offset = gridDim.x \* blockDim.x; // кол-во блоков \* размер блока

int j;

double tmp=data[i\*n+i];

for(j=idx;j<n;j+=offset){

if(j!=i)

data[j\*n+i]/=tmp;

E[j\*n+i]/=tmp;

}

}

\_\_global\_\_ void kernel(double\* data,double\* E, int n,int x) {

int idx = threadIdx.x + blockDim.x \* blockIdx.x;

int idy = threadIdx.y + blockDim.y \* blockIdx.y;

int offsetx = blockDim.x \* gridDim.x;

int offsety = blockDim.y \* gridDim.y;

int i, j;

for(i = idx; i < n; i += offsetx){

for(j = idy; j < n; j += offsety){

if(i!=x){

//a\*n b

E[j\*n+i]-=data[x\*n+i]\*E[j\*n+x];

if(j!=x)

data[j\*n+i]-=data[x\*n+i]\*data[j\*n+x];

}

}

}

}

**Результаты**

Тест на 100 элементах

|  |  |
| --- | --- |
| Время выполнения | Операция |
| 5.6792ms | kernel |
| 2.4824ms | my\_swap |
| 187.39us | [CUDA memcpy DtoH] |
| 1.497ms | normalization |
| 13.824us | [CUDA memcpy HtoD] |

Тест на 1000 элементах

|  |  |
| --- | --- |
| Время выполнения | Операция |
| 252.46ms | kernel |
| 23.569ms | my\_swap |
| 5.3793ms | [CUDA memcpy DtoH] |
| 3.1721ms | [CUDA memcpy HtoD] |
| 17.03ms | normalization |

Тест на 4000 элементах

|  |  |
| --- | --- |
| Время выполнения | Операция |
| 10.88309s | kernel |
| 123.25ms | my\_swap |
| 73.496ms | [CUDA memcpy DtoH |
| 52.406ms | [CUDA memcpy HtoD |
| 93.0555ms | normalization |

**Выводы**

Реализованный алгоритм правильно считает матрицу в 100 из 100 случаев (сравнение производилось с результатами, полученными с помощью библиотеки numpy языка python). Расчет элементов матрицы на GPU дает огромный выигрыш во времени. Реализация трудностей не вызвала, за исключением мелких неточностей в вычислениях индексов элементов.

Также я пользовался во время тестирования компьютером соседа.