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: Д. А. Ваньков
Группа: 8О-407Б-17
Преподаватели: А.Ю. Морозов,
К.Г. Крашенинников

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.

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. Все результаты выводить с точностью 10^-7.

Вариант 4. LU-разложение матрицы.

Необходимо вычислить LU-разложение квадратной матрицы: A = LU, где A -- матрица n x n, L -- нижняя треугольная матрица, с единичными элементами на диагонали, U -- верхняя треугольная матрица. Дополнительно нужно получить вектор перестановок строк p, где p[i] содержит номер строки, с которой произошла перестановка на i-ой итерации.

Программное и аппаратное обеспечение

Graphics card: GeForce 940M

Размер глобальной памяти: 4242604032

Размер константной памяти: 65536

Размер разделяемой памяти: 49152

Максимальное количество регистров на блок: 65536

Максимальное количество потоков на блок: 1024

Количество мультипроцессоров: 3

OS: Linux Ubuntu 18.04

Редактор: CLion, Atom

Машины в кластере:

1. Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 16 Gb, GeForce GTX 1050, 2 Gb

2. Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 16 Gb, GeForce GT 545, 3 Gb

3. Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 16 Gb, GeForce GTX 650, 2 Gb

4. Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 12 Gb, GeForce GT 530, 2 Gb

5. Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 8 Gb, GeForce GT 530, 2 Gb

Все машины соединены гигабатным ethernet и находятся в подсети 10.10.1.1/24.

Версии софта: mpirun 1.10.2, g++ 4.8.4, nvcc 7.0

Метод решения

Прежде чем приступить к реализации с использованием библеотек CUDA и thrust я реализовал данный алгоритм на CPU. Идея заключается в двух проходах, каждый из которых можно разбить на части.

1. Нахождение максимального по модулю элемента в i-ом столбце ниже главной диагонали, а также номер строки j в котором находиться этот элемент.
2. Swap i-ой и j-ой строк.
3. Разделение всех элементов i-ого столбца на элемент на главной диагонали, data[i][i].
4. Применение ко всем оставшимся элементам матрицы ниже главной диагонали формулы: $u_{k,m} -= u_{k,i} * u_{i,m}$

В результате мы получаем матрицу6 которая представляет собой объединение матриц L и U.

Описание программы

Для реализации на GPU я использовал 3 ядра, одно для разделения элементов (шаг L), другое для применения формулы $u_{k,m} -= u_{k,i} * u_{i,m}$ (шаг U), и последнее для смены местами строк. Для быстрой перемены мест я использую:

```
thrust::swap(data[i + j * n], data[max_idx + j * n]);
```

Для поиска максимального элемента я также использую thrust. Создается массив, с указателем на начало и ищется максимальный элемент, как только он найден возвращается указатель на этот элемент:

```
data_ptr = thrust::device_pointer_cast(dev_data + i * n);
max_ptr = thrust::max_element(data_ptr + i, data_ptr + n, cmp);
max_idx = max_ptr - data_ptr;
```

Функция max использует свой компаратор:

```
struct abs_comparator{
    __host__ __device__ bool operator()(double x, double y) {
        return fabs(x) < fabs(y);
```

```
        }
};
```

Также я использую функцию cudaThreadSynchronize для синхронизации потоков, во избежание гонки потоков из разных функций.

```
CUDA_ERROR(cudaThreadSynchronize());
```

Результаты

Я сравнил время выполнения двух разных программ: написанных на CPU и GPU.

Размер матрицы	GPU	CPU
1000x1000	2123.8ms	5230.2ms
1500x1500	2492.11ms	9920.31ms
2000x2000	2791.51ms	20135.94ms

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

Выводы

Данный алгоритм я уже реализовал в курсе численных методов, однако было интересно посмотреть на реализацию с использованием графического процессора.

В данной лабораторной работе я узнал про то, как пользоваться библиотекой thrust, которая предоставляет возможность эффективных и безопасных вычислений.

В который раз убедился, что использование графического процессора существенно упрощает работу с матрицами, и простыми вычислениями. Нужно всего лишь аккуратно реализовать ядра, и будет получен существенный прирост по времени по сравнению с CPU.