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 матрицами. Метод Гаусса.

Выполнил: Лобанов О. А.
Группа: 8О-408Б-20
Преподаватель: А.Ю. Морозов

Москва, 2023

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 в Thrust.

Вариант 2. Вычисление обратной матрицы.

Программное и аппаратное обеспечение

Device: Tesla T4
Compute capability: 7.5
Total constant memory: 65536
Registers per block: 65536
Max threads per block: 1024
Multiprocessors count: 40
OS: Ubuntu 22.04.2 LTS
Redactor: colab google

Метод решения

Хранение матрицы осуществляется по столбцам. С помощью метода Гаусса приводим изначальную и единичную матрицу к верхнему треугольному виду, находя максимальный элемент через созданный компаратор, меняя строки.

Описание программы

Компратор

```
class Comparator{
public:
    __host__ __device__ bool operator()(const double a, const double b) const{
        return fabs(a) < fabs(b);
    }
}
```

Основные функции

```
__global__ void kernel(double * A, double* E, int n, int i, int maximum) {
    int xdx = blockIdx.x * blockDim.x + threadIdx.x;
    int offsetx = gridDim.x * blockDim.x;
    for (int l = xdx; l < n; l += offsetx) {
        thrust::swap(A[l * n + i], A[l * n + maximum]);
        thrust::swap(E[l * n + i], E[l * n + maximum]);
    }
}
```

```
__global__ void Triangle_1(double *A, double *E, int n, int number) {
    int xdx = blockIdx.x * blockDim.x + threadIdx.x;
    int ydy = blockIdx.y * blockDim.y + threadIdx.y;
    int offsetx = gridDim.x * blockDim.x;
    int offsety = gridDim.y * blockDim.y;
    double check;
    for (int i = number + 1 + xdx; i < n; i += offsetx) {
        check = -A[number * n + i]/A[number * n + number];
        for (int j = number + 1 + ydy; j < n; j += offsety) {
            A[j * n + i] = check * A[j * n + number] +A[j * n + i];
        }
        for (int j = ydy; j < n; j += offsety) {
            E[j * n + i] = check * E[j * n + number] + E[j * n + i];
        }
    }
}
```

```
__global__ void Triangle_2(double *A, double *E, int n, int number) {
    int xdx = threadIdx.x + blockIdx.x * blockDim.x;
    int ydy = threadIdx.y + blockIdx.y * blockDim.y;
    int offsetx = gridDim.x * blockDim.x;
    int offsety = gridDim.y * blockDim.y;
    double check;
    for (int i = number - 1 - xdx; i >= 0; i -= offsetx) {
        check = -A[number * n + i] /A[number * n + number];
        for (int j = ydy; j < n; j += offsety) {
            E[j * n + i] = check * E[j * n + number] + E[j * n + i];
        }
    }
}
```

```
__global__ void BasedKernel(double *A, double *E, int n) {
    int xdx = threadIdx.x + blockIdx.x * blockDim.x;
    int ydy = threadIdx.y + blockIdx.y * blockDim.y;
    int offsetx = gridDim.x * blockDim.x;
    int offsety = gridDim.y * blockDim.y;
    for (int i = xdx; i < n; i += offsetx) {
        for (int j = ydy; j < n; j += offsety) {
            E[j * n + i] = E[j * n + i] / A[i * n + i];
        }
    }
}
```

Результаты

	Размер	Время
CPU	10^2	0.011
GPU <<<(16, 16), (16, 16) >>>	10^2	0.001
GPU <<<(32, 32), (32, 32) >>>	10^2	1.213
CPU	4 * 10^3	323.390
GPU <<<(16, 16), (16, 16) >>>	4 * 10^3	58.251
GPU <<<(32, 32), (32, 32) >>>	4 * 10^3	60.394

Выводы

В результате выполненной работы изучил сделал метод Гаусса оптимизированный под GPU. По результатам можно заметить, что на GPU алгоритм работает быстрее, чем на CPU. Стоит отметить, что работа с глобальной памятью становится медленнее по сравнению с другими типами памяти на GPU.