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: П.А. Гамов
Группа: 8О-407Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

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 4. LU-разложение матрицы.

Программное и аппаратное обеспечение

nvcc 7.0

Ubuntu 14.04 LTS

Compute capability	6.1
Name	GeForce GTX 1050
Total Global Memory	2096103424
Shared Mem per block	49152
Registers per block	65534
Max thread per block	(1024,1024,64)
Max block	(2147483647, 65535, 65535)
Total constant memory	65536
Multiprocessor’s count	5

Метод решения

Будем хранить матрицы L и U в одной матрице A. Каждую итерацию будем искать максимальный элемент в столбце с помощью библиотеки Trust. Потом проводить операции на матрице, далее в конце выводить получившееся разложение.

Описание программы

Поиск максимального значения в столбце будем производить с помощью библиотеки Thrust.

```
thrust::device_ptr<double> d_ptr = thrust::device_pointer_cast(A_DEV) + (i * n + i);
```

Для начала создадим указатель на память в девайсе, откуда начинается наш столбец. Так как мы храним столбцы рядом, нам требуется найти смещение, в диапазоне которого мы будем искать максимальный элемент.

```
thrust::device_ptr<double> max = thrust::max_element(d_ptr, d_ptr + (n - i), comp);
```

Максимальный элемент ищем в диапазоне от главной диагонали до самого конца.

```
newidxarr[i] = max - d_ptr + i;
```

Добавляем индекс нового максимального элемента в массив пермутаций.

```
struct comparator {
	__host__ __device__ bool operator()(double a, double b) {
		return abs(a) < abs(b);
	}
};
```

Так же для Thust требуется компаратор, который в данном случае проверяет модуль числа.

```
__global__ void LUP_swap(double * A, int i, int n, int newidx) {
	int idx = blockDim.x * blockIdx.x + threadIdx.x;
	double piv;
	for (int var = idx; var < n; var += blockDim.x * gridDim.x) {
		piv = A[newidx + n * var];
		A[newidx + n * var] = A[i + n * var];
		A[i + n * var] = piv;
	}
}
```

Первое ядро просто меняет местами текущую и новую главную сроку.

```
__global__ void LUP_N(double * A, int i, int n) {
	int idx = blockDim.x * blockIdx.x + threadIdx.x;
	int shift = blockDim.x * gridDim.x;
	for (int var = idx + i + 1; var < n; var += shift)
		A[var + n * i] /= A[i + n * i];
}
```

Второе ядро делит элементы текущего столбца ниже главной диагонали на элемент главной диагонали, тем самым формирует матрицу L.

```
__global__ void LUP(double * A, int i, int n) {
	int idx = blockDim.x * blockIdx.x + threadIdx.x;
	int idy = blockDim.y * blockIdx.y + threadIdx.y;
	int shiftx = blockDim.x * gridDim.x;
	int shifty = blockDim.y * gridDim.y;
	for (int var = idx + i + 1; var < n; var += shiftx)
		for (int k = idy + i + 1; k < n; k += shifty)
			A[var + n * k] -= A[var + n * i] * A[i + n * k];
}
```

И финальное двумерное ядро бегает по всем строкам ниже текущей, делит и вычитает из матрицы элементы, просто делает метод Гаусса. Тем самым формируется матрица U.

Результаты

	500x500	1000x1000	1500x1500
(32,32), (32,32)	0.232 sec	1.326 sec	3.248 sec
(64,64), (32,32)	0.206 sec	1.206 sec	3.127 sec

	500x500	1000x1000	1500x1500
C++	0.978 sec	3.087 sec	8.012 sec

Выводы

Алгоритм LUP разложения применяется в многих областях линейной алгебры, численных методах, он нужен для решения систем линейных уравнений, нахождения определителя или обратной матрицы. Библиотека Thrust позволяет писать меньше кода и опереться на проверенный быстрый метод решения конкретных проблем.