МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №4
по курсу «Параллельные и распределенные вычисления**»**

Работа с матрицам. Метод Гаусса.

Выполнил: *Буловятов А.М*
Группа: *М80-310Б-20*
Преподаватель: А.Ю. Морозов

Москва, 2023

Условие

Вариант 1. Вычисление детерминанта матрицы.

Вычисление определителя матрицы методом Гаусса с выбором главного элемента по столбцу. Использование библиотеки Thrust для поиска максимального элемента в массиве. Использование двухмерной сетки потоков и объединения запросов к глобальной памяти.

Программное и аппаратное обеспечение

GPU:

Графическая карта: NVIDIA Tesla T4
Compute capability : 7.5
Глобальная память : 15835398144
Разделяемая память : 49152
Регистров в блоке : 65536
Макс. размер блока : (1024, 1024, 64)
Макс. размер сетки : (2147483647, 65535, 65535)
Константная память : 65536
Количество мультипроцессоров : 40

CPU: Intel(R) Xeon(R) CPU @ 2.30GHz

Метод решения

Для поиска определителя методом Гаусса необходимо считать матрицу, оптимизация скорости работы программы достигается за счет хранения матрицы в виде одномерного массива по столбцам. Выбор опорного элемента в методе Гаусса нужен для предотвращения деления на ноль и увеличения точности вычислений. Для выбора опроного элемента необходимо найти максимальный элемент массива, для чего применяется библиотека Thrust, которая позволяет производить вычисления на GPU. В случае если индекс максимального элемента в столбце не является диагональным необходимо произвести обмен строк и изменить знак определителя на противоположный. Данный метод также выполняется на GPU, что обеспечивает высокую скорость работы программы.Следующим этапом является заполнение столбца для чего происходит операция вычитания строк, которая не изменяет значение определителя. После приведения матрицы к верхнетреугольному виду определитель будет равен произведению элементов, расположенных на главной диагонали с учетом знака полученной на предыдущих операциях.

Описание программы

Реализация ядра для зануления столбца:

```
__global__
void kernel(double* matrix, int size, int shift) {
	int idx = blockDim.x * blockIdx.x + threadIdx.y + 1 + shift;
	int idy = blockDim.y * blockIdx.y + threadIdx.x + 1 + shift;
	int offsetx = blockDim.x * gridDim.x;
	int offsety = blockDim.y * gridDim.y;
	int x, y;

	int tmp = shift * size;
	double div = matrix[tmp + shift];
	for (y = idy; y < size; y += offsety) {
		double coef = matrix[tmp + y] / div;
		for (x = idx; x < size; x += offsetx) {
			matrix[x * size + y] -= coef * matrix[x * size + shift];
		}
	}
}
```

Реализация ядра для обмена строк местами:

```
__global__
void swapRows(int a, int b, double* arr_dev, int size) {
	int idx = threadIdx.x + blockIdx.x * blockDim.x;
	int offset = gridDim.x * blockDim.x;
	double tmp;
	for (int x = idx; x < size; x += offset) {
		tmp = arr_dev[x*size + a];
		arr_dev[x*size + a] = arr_dev[x*size + b];
		arr_dev[x*size + b] = tmp;
	}
}
```

Реализация функционала поиска максимального элемента в массиве:

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

int getMax(int iter, double * arr_dev, int size, comporator comp) {
	thrust::device_ptr<double> p_arr =
		thrust::device_pointer_cast(arr_dev +size * iter + iter);
	thrust::device_ptr<double> res =
		thrust::max_element(p_arr , p_arr + size - iter, comp);

	return ((int)(res - p_arr) + iter);
}
```

Результаты

Временные замеры на GPU:

матрица\ядро	(4,4) (4,4)	(16,16) (16,16)	(16,16) (32, 32)	(32, 32) (32, 32)
500	136.50 ms	50.50 ms	75.20 ms	116.84 ms
1000	743.93 ms	225.21 ms	275.53 ms	322.49 ms
2000	4280.50 ms	551.00 ms	583.34 ms	729.87 ms
3000	14156.80 ms	1369.45 ms	1528.40 ms	1605.19 ms

Временные замеры на CPU:

100	100.42 ms
200	471.17 ms
500	7481.25 ms
1000	68308.90 ms

Выводы

В результате выполнения лабораторной работы был разработан алгоритм вычисления определителя квадратной матрицы методом Гаусса с выбором опорного элемента. Для ускорения процесса была использована библиотека Thrust, реализующая поиск максимального элемента массива.

Проведенные эксперименты показали значительное ускорение вычислений на GPU по сравнению с последовательным вычислением на CPU. Это демонстрирует высокую эффективность использования CUDA для решения математических задач большой размерности.

Также было выяснено, что выбор опорного элемента влияет на точность и скорость решения системы уравнений методом Гаусса. Правильный выбор опорного элемента может значительно ускорить вычисления и увеличить точность решения.

Таким образом, разработанный алгоритм на основе метода Гаусса с выбором опорного элемента и использованием технологии CUDA является эффективным инструментом для решения задач линейной алгебры с большими объемами данных.