МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 810 «Информационные технологии в моделировании и управлении»

Лабораторная работа №4
по курсу «Основы Python, Java и Scala, платформы CUDA для анализа данных»

Работа с матрицами. Метод Гаусса.

Выполнил: А.С.Бобряков
Группа: М8О-103М-19
Преподаватель: А.Ю. Морозов

Москва, 2020

Условие

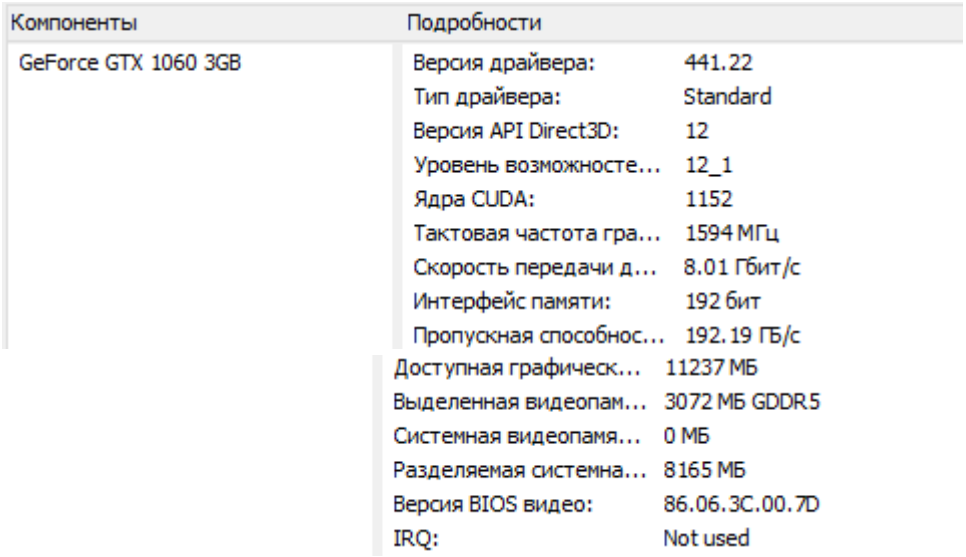
Использование объединения запросов к глобальной памяти. Реализация метода Гаусса с выбором главного элемента по столбцу. Ознакомление с библиотекой алгоритмов для параллельных расчетов Thrust.

В качестве вещественного типа данных необходимо использовать тип данных double. Библиотеку Trust использовать только для поиска максимального элемента на каждой итерации алгоритма. Сравнение с нулем использовать значение 10^{-7}. Результаты выводить с точностью до 10^{-10}.

Вариант 5. Решение произвольной СЛАУ.

Программное и аппаратное обеспечение

Видеокарта: NVIDIA GeForce GTX 1060 3Gb



Компоненты	Подробности	
GeForce GTX 1060 3GB	Версия драйвера:	441.22
	Тип драйвера:	Standard
	Версия API Direct3D:	12
	Уровень возможносте...	12_1
	Ядра CUDA:	1152
	Тактовая частота гра...	1594 МГц
	Скорость передачи д...	8.01 Гбит/с
	Интерфейс памяти:	192 бит
	Пропускная способнос...	192.19 ГБ/с
	Доступная графическ...	11237 МБ
	Выделенная видеопам...	3072 МБ GDDR5
	Системная видеопамя...	0 МБ
	Разделяемая системна...	8165 МБ
	Версия BIOS видео:	86.06.3C.00.7D
	IRQ:	Not used

Процессор: Intel® Core™ i7-8700K CPU @ 3.70GHz

Другое: ОС Windows, IDE – Clion EAP,

Метод решения

Необходимо найти любое решение системы уравнений Ax = b, где A -- матрица n x m, b -- вектор-столбец свободных коэффициентов длиной n, x – вектор неизвестных длиной m.

Необходимо реализовать алгоритм преобразований методом Гаусса для приведения исходного выражения к виду:

$$\begin{cases} \alpha_{1j_1}x_{j_1} + \alpha_{1j_2}x_{j_2} + \ldots + \alpha_{1j_r}x_{j_r} + \ldots + \alpha_{1j_n}x_{j_n} = \beta_1 \\ \alpha_{2j_2}x_{j_2} + \ldots + \alpha_{2j_r}x_{j_r} + \ldots + \alpha_{2j_n}x_{j_n} = \beta_2 \\ \ldots \\ \alpha_{rj_r}x_{j_r} + \ldots + \alpha_{rj_n}x_{j_n} = \beta_r \\ 0 = \beta_{r+1} \\ \ldots \\ 0 = \beta_m \end{cases},$$

где $\alpha_{1j_1}, \ldots, \alpha_{rj_r} \neq 0$.

Из каждого такого выражения находим любое решение СЛАУ. Необходимо учитывать возможность данных, определяющих определенную и неопределенную СЛАУ. Где определенная СЛАУ – СЛАУ, которая совместна и имеет единственное решение; неопределенная – совместна и имеет более одного решения.

Описание программы

В программе использовано ядро для реализации замены местами строк. Код ядра описан на листинге 1.

```
__global__ void swapRows(double* A, int n, int m, int k, int i, int j) {
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    int offset = gridDim.x * blockDim.x;
    while(idx < m + k) {
        int cnt = idx * n;
        int first  = i + cnt;
        int second = j + cnt;
        double tmp = A[first];
        A[first] = A[second];
        A[second] = tmp;
        idx += offset;
    }
}
```

Листинг 1 – Код ядра программы.

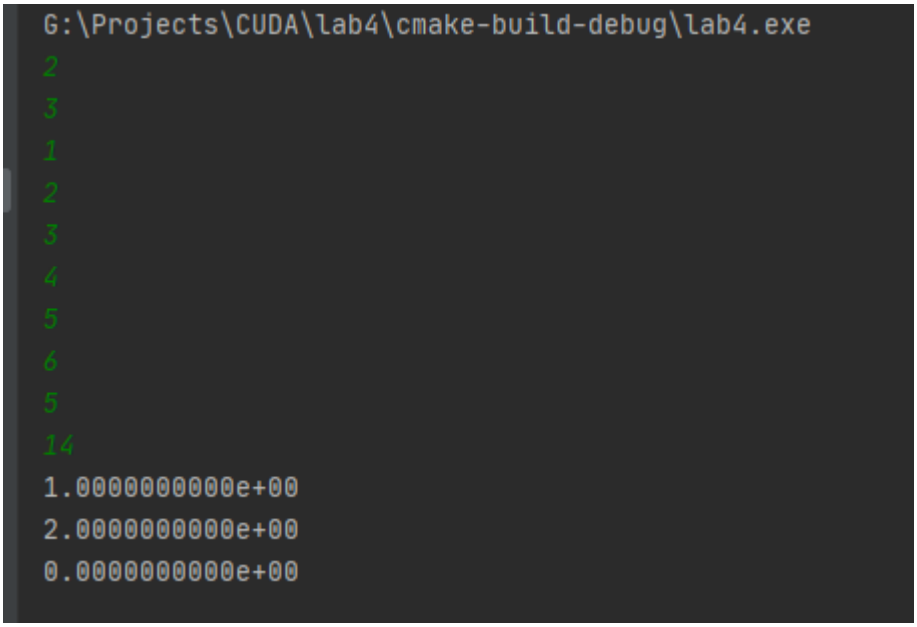
Также реализовано ядра для прямого и обратного прохода по элементам матрицы. Код таких ядер определен в листинге 2.

```cpp
__global__ void forward(double* A, int n, int m, int k, int i, int j) {
    int col = blockIdx.x;
    int row = threadIdx.x;
    int colOffset = gridDim.x;
    int rowOffset = blockDim.x;

    double a1 = A[n*j + i];
    for(col = blockIdx.x + j + 1; col < m + k; col += colOffset) {
        double cur = A[col*n + i];
        for(row = threadIdx.x + i + 1; row < n; row += rowOffset) {
            double l = A[n*j + row] / a1;
            A[col*n + row] -= l * cur;
        }
    }
}

__global__ void back(double* A, int n, int m, int k, int i, int j) {
    int col = blockIdx.x;
    int row = threadIdx.x;
    int colOffset = gridDim.x;
    int rowOffset = blockDim.x;

    double a1 = A[j*n + i];
    int start = m*n;
    for(col = blockIdx.x; col < k; col += colOffset) {
        double cur = A[start + col*n + i];
        for(row = threadIdx.x; row < i; row += rowOffset) {
            double l = A[j*n + row] / a1;
            A[start + col*n + row] -= l * cur;
        }
    }
}
```

Результаты

Пример исходных данных с результатом работы программы:

```
G:\Projects\CUDA\lab4\cmake-build-debug\lab4.exe
3
2
1
2
3
4
5
6
7
8
9
-6.0000000000e+00
6.5000000000e+00
```



Время работы ядер в зависимости от конфигурации представлены в Таблице 1,2.

Таблица 1. Время выполнения ядер программы в зависимости от конфигурации.

ядро	min	max
SwapRows	0.008137	0.010331
Forward	0.083480	0.115103
Back	0.073518	0.135186

Выводы

В ходе выполнения лабораторной работы был исследован метод Гаусса для решения СЛАУ и его реализация на CUDA с учетом типа СЛАУ - определенной/неопределенной.