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 матрицам. Метод Гаусса.

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

Москва, 2023

Условие

Цель: Использование объединения запросов к глобальной памяти. Реализация метода Гаусса с выбором главного элемента по столбцу. Ознакомление с библиотекой алгоритмов для параллельных расчетов Thrust. Использование двухмерной сетки потоков. Исследование производительности программы с помощью утилиты nvprof (обязательно отразить в отчете).

Вариант задания: Вариант 2. Вычисление обратной матрицы.

Программное и аппаратное обеспечение

Name:	Tesla T4
Compute capability:	7.5
Dedicated video memory:	15835 MB
Shared memory per block:	49152 bytes
Register per block:	65536 bytes
Total constant memory:	65536
Max threads per multiprocessor:	2048
Max threads per block:	1024
Multiprocessors const:	40

AMD RYZEN 5 5500U

Architecture:	Ryzen 5 Zen2(Lucienne)
Processor Technology for CPU Cores:	TSMC 7nm FinFET
CPU Cores:	6
Thread(s):	12
CPU Socket	FP6
CPU MHz:	3600
SMP # CPUs:	1
L2 cache: L3 cache:	3MB 8MB
RAM	16GB
SSD	512GB

OS: Windows 10 - 64-Bit Edition, Ubuntu x86 64-Bit; IDE: VS code; Compiler: nvcc

Метод решения

Понятие обратной матрицы вводится лишь для квадратных матриц, определитель которых отличен от нуля, то есть для невырожденных квадратных матриц.

Пусть нам задана квадратная матрица $A_{n\times n}$. Допишем справа к матрице A единичную матрицу E n-го порядка. После такого дописывания мы получим матрицу (A|E). Этот метод делят на два этапа, которые называют *прямым ходом* и *обратным.*

В процессе выполнения *прямого хода* мы последовательно используем строки матрицы. На первом шаге работаем с первой строкой, на втором шаге – со второй и так далее. Если в ходе решения в матрице до черты возникла нулевая строка, то прекращаем преобразования, так как обратная матрица A^{-1} не существует.

На этапе *обратным ходом,* мы поднимаемся по матрице "снизу вверх". Сначала используем последнюю строку r_n, затем предпоследнюю r_{n-1} и так далее, пока не дойдем до первой строки. С каждой строкой выполняем однотипные операции.

Описание программы

Данная программа состоит из одного главного файла. В функции main происходит считывание исходных данных для дальнейшей обработки, а после завершения работы с ней выводит новые обработанные данные.

Функция функция Swap в виде ядра CUDA выполняет параллельный обмен строк между row1 и row2 в матрице dev_matrix. Она использует множество потоков для обработки столбцов матрицы, временно сохраняя элементы в t и выполняя обмен значений между row1 и row2 в каждом столбце.

Эта функция GaussianElimination, которая реализует метод Гаусса для приведения матрицы к улучшенному ступенчатому виду. int direction - определяет, выполняется ли прямой ход (1) или обратный ход (-1). В зависимости от direction и с использованием индексов idx и idy для определения текущего потока, происходит выполнение операций приведения матрицы к ступенчатому виду. В прямом ходе (direction = 1), функция выполняет вычитание соответствующих элементов матрицы. В обратном ходе (direction = -1), операции выполняются в обратном порядке для финального приведения матрицы.

Функция Normal, которая выполняет нормализацию матрицы после применения метода Гаусса . Внутри функции лежит двойного цикла for, где выполняется нормализация элементов матрицы, разделяя каждый элемент на значение на главной диагонали (где строка и столбец совпадают). Это позволяет завершить процесс приведения матрицы к ступенчатому виду и получить её в виде, в котором на главной диагонали все элементы равны единице.

Код функции

```
__global__ void Swap(double* dev_matrix, int n, int row1, int
row2) {
      int colIdx = threadIdx.x + blockDim.x * blockIdx.x;
      int colOffset = blockDim.x * gridDim.x;

      for (int col = colIdx; col < 2 * n; col += colOffset) {
      int index_row1 = row1 + n * col;
      int index_row2 = row2 + n * col;

      double t = dev_matrix[index_row1];
      dev_matrix[index_row1] = dev_matrix[index_row2];
      dev_matrix[index_row2] = t;
      }
}
__global__ void GaussianElimination(double* dev_matrix, int n, int
i, int direction) {
      int idx = threadIdx.x + blockDim.x * blockIdx.x;
      int idy = threadIdx.y + blockDim.y * blockIdx.y;
      int offsetx = blockDim.x * gridDim.x;
      int offsety = blockDim.y * gridDim.y;

      if (direction == 1) {
      for (int k = idy + i + 1; k < 2 * n; k += offsety)
            for (int j = idx + i + 1; j < n; j += offsetx)
                  dev_matrix[k * n + j] -= (dev_matrix[k * n + i] *
dev_matrix[i * n + j] / dev_matrix[i + i * n]);
      } else {
      for (int k = idy + i + 1; k < 2 * n; k += offsety)
            for (int j = i - 1 - idx; j >= 0; j -= offsetx)
                  dev_matrix[k * n + j] -= (dev_matrix[k * n + i] *
dev_matrix[i * n + j] / dev_matrix[i + i * n]);
      }
}
__global__ void Normal(double* dev_matrix, int n) {
      int idx = threadIdx.x + blockDim.x * blockIdx.x;
      int idy = threadIdx.y + blockDim.y * blockIdx.y;
      int offsetx = blockDim.x * gridDim.x;
      int offsety = blockDim.y * gridDim.y;

      for (int i = idy; i < n; i += offsety) {
      for (int j = n + idx; j < 2 * n; j += offsetx) {
            int index = i + j * n;
            int diagonal_index = i + i * n;
            dev_matrix[index] /= dev_matrix[diagonal_index];
            }
      }
}
```

Результаты

Время исполнения программы в мс при различных данных

Размер	10^2	10^4	10^6	10^7
<<<1, 32>>>, <<<1, 32>>>	0.127	8.081	165.096	986.791
<<<32, 32>>>, <<<32, 32>>>	0.171	4.304	230.094	1091.861
<<<1, 128>>>, <<<1, 128>>>	0.396	15.685	178.093	1309.101
<<<128, 128>>>, <<<32, 32>>>	0.405	1.108	56.084	1086.982
<<<1, 256>>>, <<<1, 256>>>	0.952	5.259	987.309	1321.594
<<<256, 256>>>, <<<32,32>>>	0.314	12.037	128.564	789.011
<<<1, 512>>>, <<<1, 512>>>	11.748	36.187	58.034	607.935
<<<512, 512>>>, <<<32, 32>>>	17.173	3.058	24.093	300.924
<<<1, 1024>>>, <<<1, 1024>>>	77.282	5.875	11.037	410.104
CPU	0.023	11.365	170.098	1698.981

Выводы

В данной лабораторной работе был представлена реализация метода Гаусса с выбором главного элемента по столбцу, чтобы вычислить обратную матрицу. Данная задача широко применяется в области линейной алгебре, а также в разлиных инженерных расчетов:

- **Решение систем линейных уравнений** для быстрого решения множества связанных систем уравнений.
- **Линейная регрессия** в статистике и машинном обучении может быть полезен при реализации алгоритмов линейной регрессии, которые требуют нахождения коэффициентов регрессии.
- **Обработка изображений** при реализации фильтров могут использоваться для преобразования изображений.
- **Моделирование и симуляции** в области научных вычислений и инженерии, может быть важным инструментом для моделирования и симуляции различных физических и инженерных процессов.
- **Криптографии** обратные матрицы используются в различных алгоритмах, таких как шифрование и дешифрование, а также при создании криптографических ключей.
- **Машинном обучении и искусственном интеллекте** обратные матрицы могут использоваться в методах, которые требуют регуляризации или вычисления весов признаков.
- В **Финансовых моделях** обратные матрицы могут использоваться для анализа и прогнозирования рисков и доходности.

Сложности в данной лабораторной возникли в реализации некоторых функций, которые в итоге неправильно считали функцию. А также были проблемы со считыванием матрицы.