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: Попов М. Р.
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2023

Условие

1. **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. Использование двухмерной сетки потоков. Исследование производительности программы с помощью утилиты nvprof
2. **Вариант 1:** вычисление детерминанта матрицы.

Программное и аппаратное обеспечение

1. Графический процессор: Nvidia GeForce GT 545
    a. Количество потоковых процессоров: 144
    b. Частота ядра: 720 МГц
    c. Количество транзисторов: 1.170 млн
    d. Тех. процесс: 40 нм
    e. Энергопотребление: 70 Вт
2. ОС: Ubuntu 16.04
3. Текстовый редактор: VS Code
4. Компилятор: nvcc

Метод решения

Вспомним, что детерминант матрицы равен произведению элементов на главной диагонали у матрицы, приведённой к нижнему треугольному виду. Соответственно нам необходимо привести заданную матрицу к такому виду, используя двумерную сетку потоков, а потом найти произведение элементов на главной диагонали. Самое главное использовать преобразования, которые не изменяют определитель матрицы: поменять две строки местами, разделить строку на число и др.

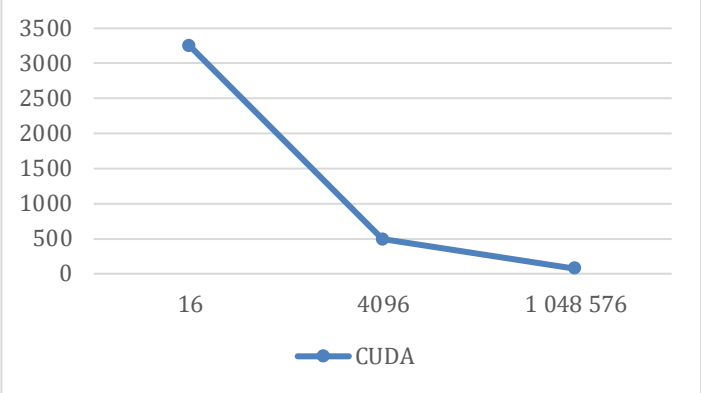
Описание программы

Программа состоит из одного файла, в котором есть функции `swap_rows_kernel`, `divide_row_kernel` и `kernel`. Первые две функции используют одномерную многопоточность для действий с одной или двумя строками матрицы, `kernel` использует двумерную сетку потоков.

Результаты

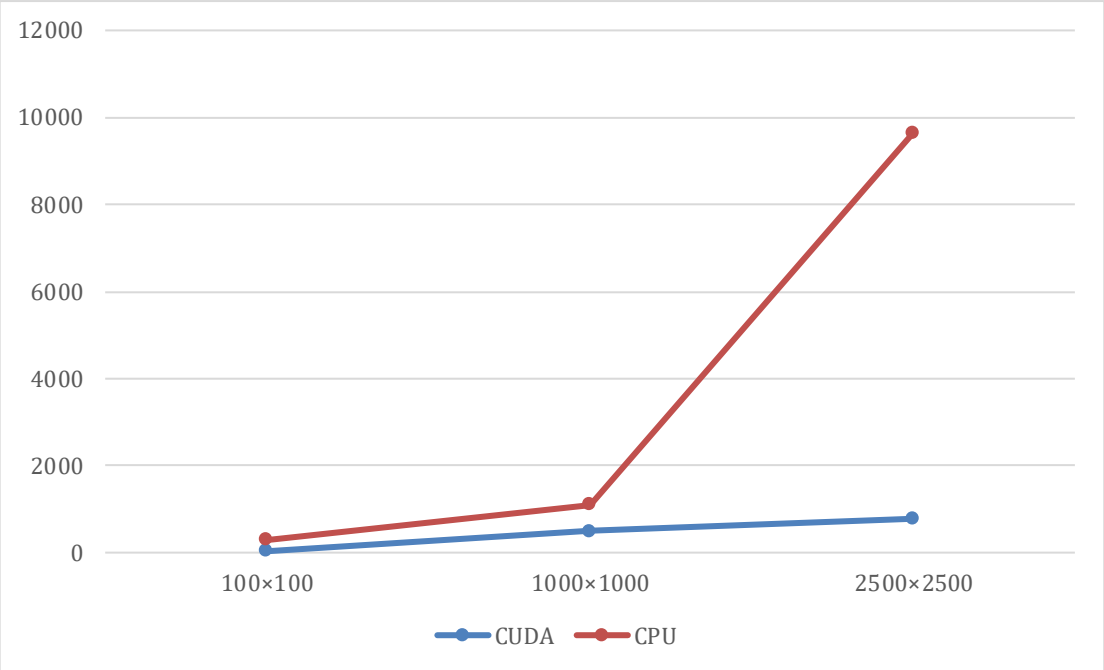
1. Зависимость времени выполнения программы от количества используемых потоков (для тестов использовалась матрица 1000×1000 элементов):

Потоки	Время (в мс)
2×2×2×2	3245
8×8×8×8	494
32×32×32×32	71



2. Сравнение программы на CUDA с 8×8×8×8 потоками и программы на CPU с одним потоком:

Размер матрицы	Время на CUDA (в мс)	Время на CPU (в мс)
100×100	31	288
1000×1000	494	1108
2500×2500	778	9645



3. Результаты исследования производительности с помощью nvprof (матрица 1000×1000, сетка потоков 8×8×8×8)

```
==20814== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 86.11%  258.18ms       999  258.44us  2.0470us  785.51us  kernel(double*, int, int)
  4.53%  13.574ms       999  13.587us  6.9620us  16.776us  void thrust::system::cuda
  3.67%  10.996ms       992  11.084us  6.7380us  13.306us  swap_rows_kernel(double*,
  2.76%  8.2857ms      1999  4.1440us  2.8160us  1.3587ms  [CUDA memcpy DtoH]
  1.05%  3.1423ms       392  8.0160us  6.4170us  8.2000us  void thrust::system::cuda
  0.81%  2.4137ms       999  2.4160us  1.4970us  4.6610us  divide_row_kernel(double*,
  0.57%  1.7043ms       999  1.7050us  1.3840us  1.9000us  void thrust::system::cuda:
  0.51%  1.5388ms         1  1.5388ms  1.5388ms  1.5388ms  [CUDA memcpy HtoD]
```

```
==20814== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 51.33%  299.31ms      1998  149.81us  18.677us  799.51us  cudaMemcpyAsync
 25.45%  148.41ms      1000  148.41us  82.677us  61.691ms  cudaMalloc
  9.49%  55.331ms      1000  55.330us  52.184us  166.39us  cudaFree
  7.10%  41.376ms      5380  7.6900us  4.4430us  429.25us  cudaLaunch
  3.47%  20.236ms      9168  2.2070us  1.8310us  12.175us  cudaFuncGetAttributes
  0.58%  3.4042ms      7170     474ns     254ns  4.2740us  cudaGetDevice
  0.55%  3.2219ms      2390  1.3480us     991ns  9.7870us  cudaEventCreateWithFlags
  0.53%  3.0987ms         2  1.5493ms  1.4986ms  1.6001ms  cudaMemcpy
  0.37%  2.1585ms      2390     903ns     571ns  232.80us  cudaEventDestroy
  0.36%  2.0856ms     12352     168ns     107ns  9.0080us  cudaSetupArgument
  0.35%  2.0522ms      2390     858ns     689ns  4.6260us  cudaEventRecord
  0.32%  1.8571ms      5380     345ns     206ns  231.91us  cudaConfigureCall
  0.06%  375.18us        83  4.5200us     194ns  161.65us  cuDeviceGetAttribute
  0.02%  101.94us         1  101.94us  101.94us  101.94us  cuDeviceTotalMem
  0.01%  40.626us         1  40.626us  40.626us  40.626us  cuDeviceGetName
  0.00%  5.9380us        11     539ns     390ns  1.3980us  cudaDeviceGetAttribute
  0.00%  1.6620us         2     831ns     363ns  1.2990us  cuDeviceGetCount
  0.00%     713ns         2     356ns     226ns     487ns  cuDeviceGet
```

Выводы

Проделав лабораторную работу, я реализовал алгоритм нахождения определителя матрицы с помощью метода Гаусса, для этого я использовал библиотеку Thrust, двумерную сетку потоков и объединение запросов к глобальной памяти.