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: Семин А. В.
Группа: 8О-406Б
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 3. Решение квадратной СЛАУ.**
   Необходимо решить систему уравнений Ax = b, где A - квадратная матрица n x n, b - вектор-столбец свободных коэффициентов длинной n, x - вектор неизвестных.
   Входные данные. На первой строке задано число n - размер матрицы. В следующих n строках, записано по n вещественных чисел -- элементы матрицы. Далее записываются n элементов вектора свободных коэффициентов. $n \leq 10^4$
   Выходные данные. Необходимо вывести n значений, являющиеся элементами вектора неизвестных x.

Программное и аппаратное обеспечение

Графический процессор (GeForce GTX 1650 Ti)

1. Количество потоковых процессоров: 1024
2. Частота ядра: 1350 МГц
3. Частота в режиме Boost: 1485 МГц
4. Количество транзисторов: 6,600 млн
5. Тип памяти: DDR6
6. Видеопамять: 4096 МБ
7. Частота памяти: 12000 МГц

Процессор AMD Ryzen 7 4800H

1. ядра: 8
2. потоки: 16
3. частота: 2.9 ГГц
4. максимальная частота: 4.2 ГГц
5. кэш 1 уровня: 64 КБ (на ядро)
6. кэш 2 уровня 512 КБ (на ядро)
7. кэш 3 уровня: 8 МБ (общий)

16 ГБ ОЗУ и 512 ГБ SSD.
OS – Windows 11Домашняя, WSL, IDE – VS Code, Compiler - nvcc, g++.

Метод решения

Входные данные считываются из стандартного потока, на их основании заполняется искомая матрица. Матрица хранится в линейном массиве. Доступ к элементу осуществляется в виде matrix[j*n + i], где j – номер столбца, i – номер строки. Такое представление используется для хранения матрицы «по столбцам», что упрощает реализацию метода Гаусса.

После инициализации матрица копируется на GPU, а затем происходит реализация метода Гаусса также на GPU.

Алгоритм заключается в преобразовании матрицы в верхнетреугольную с помощью метода Гаусса. На каждой итерации метода берется максимальный элемент в столбце из позиций (i, i) – (n, i) – От элемента на диагонали до нижней строки. Если максимальный элемент не совпадает с текущей позицей (i, i), то происходит замена строк местами. Далее производим сложение каждой из строк под текущей с текущей строкой по определенной формуле. Таким образом, элемент под текущим диагональным элементом зануляется.

После выполнения метода Гаусса решаем результат копируется на GPU, а с помощью полученной матрицы осуществляется решение квадратичной системы уравнений. Представление матрицы в верхнетреугольном виде позволяет решить уравнение за квадратичную сложность простым выражением каждого из X-ов, начиная с нижней строки. Решение уравнения происходит на CPU.

Описание программы

Программа состоит из одного файла. В нем реализованы следующие функции:

__host__ void readMatrixAndB(double* matrix, int n) – считывание входных данных;

__global__ void swapRows(double* data, int n, int curRow, int rowToSwap) – замена строк местами, реализованная на одномерной сетке потоков;

__global__ void Gauss(double* data, int n, int row) – метод Гаусса, реализованный на двумерной сетке потоков;

__host__ void solveEquatation(double* matrix, double* res, int n) – решение квадратичной системы уравнений.

Использование утилиты nvprof

Исследуем производительность программы утилитой для матрицы 1000 * 1000.

Результат для конфигурации *swapRows<<<64,64>>>(...)* и *Gauss<<<dim3(32,32), dim3(16,16)>>>(...)*:

```
a@WIN-THNQL51M105:/mnt/c/Users/user/Desktop/MAIStudy/PGP/lab4$ nvprof ./lab64 < test1000.txt
==775== NVPROF is profiling process 775, command: ./lab64
==775== Warning: Unified Memory Profiling is not supported on the current configuration because a pair of devices
ppings are not available, system falls back to using zero-copy memory. It can cause kernels, which access unifie
da/cuda-c-programming-guide/index.html#um-managed-memory
==775== Error: Internal profiling error 4075:999.
======== Warning: 502 records have invalid timestamps due to insufficient device buffer space. You can configure
======== Warning: 85 records have invalid timestamps due to insufficient semaphore pool size. You can configure
======== Profiling result:
No kernels were profiled.
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
      API calls:   95.40%  989.22ms       169  5.8534ms  3.8070us  955.70ms  cudaMalloc
                    1.72%  17.842ms       166  107.48us  3.6270us  381.86us  cudaFree
                    1.12%  11.577ms       417  27.762us  7.8140us  89.367us  cudaLaunchKernel
                    0.50%  5.2216ms       251  20.803us  2.9850us  171.54us  cudaDeviceSynchronize
                    0.35%  3.6234ms        84  43.135us  31.920us  106.29us  cudaMemcpyAsync
                    0.30%  3.1393ms         1  3.1393ms  3.1393ms  3.1393ms  cudaEventRecord
                    0.20%  2.0484ms         1  2.0484ms  2.0484ms  2.0484ms  cuDeviceGetPCIBusId
                    0.16%  1.6914ms         1  1.6914ms  1.6914ms  1.6914ms  cudaMemcpy
                    0.15%  1.5311ms       335  4.5700us  2.1740us  264.67us  cudaFuncGetAttributes
                    0.03%  280.90us        83  3.3840us  2.0740us  5.8610us  cudaStreamSynchronize
                    0.03%  267.18us       419     637ns     280ns  2.5040us  cudaGetDevice
                    0.02%  256.34us       419     611ns     270ns  2.1440us  cudaDeviceGetAttribute
                    0.01%  92.411us       502     184ns      90ns     751ns  cudaPeekAtLastError
                    0.00%  41.408us       251     164ns     110ns     401ns  cudaGetLastError
                    0.00%  16.631us        97     171ns     100ns  1.7430us  cuDeviceGetAttribute
                    0.00%  10.019us         2  5.0090us  1.5830us  8.4360us  cudaEventCreate
                    0.00%  3.4160us         3  1.1380us     200ns  2.8450us  cuDeviceGetCount
                    0.00%  2.4240us         2  1.2120us     160ns  2.2640us  cuDeviceGet
                    0.00%     972ns         1     972ns     972ns     972ns  cuDeviceGetName
                    0.00%     300ns         1     300ns     300ns     300ns  cuDeviceTotalMem
                    0.00%     150ns         1     150ns     150ns     150ns  cuDeviceGetUuid
```

Результат для конфигурации *swapRows<<<256,256>>>(...)* и *Gauss<<<dim3(64, 64), dim3(32, 32)>>>(...)*:

```
a@WIN-THNQL51M105:/mnt/c/Users/user/Desktop/MAIStudy/PGP/lab4$ nvprof ./lab256 < test1000.txt
==801== NVPROF is profiling process 801, command: ./lab256
==801== Warning: Unified Memory Profiling is not supported on the current configuration because a pa
ppings are not available, system falls back to using zero-copy memory. It can cause kernels, which a
da/cuda-c-programming-guide/index.html#um-managed-memory
==801== Error: Internal profiling error 4075:999.
======== Warning: 309 records have invalid timestamps due to insufficient device buffer space. You c
======== Warning: 52 records have invalid timestamps due to insufficient semaphore pool size. You ca
======== Profiling result:
No kernels were profiled.
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
      API calls:   96.17%  996.23ms       105  9.4879ms  4.0480us  973.05ms  cudaMalloc
                    1.12%  11.617ms       102  113.89us  3.8070us  341.48us  cudaFree
                    0.82%  8.4745ms       155  54.673us  2.8560us  747.60us  cudaDeviceSynchronize
                    0.66%  6.8501ms       257  26.654us  5.5500us  155.20us  cudaLaunchKernel
                    0.35%  3.6352ms        51  71.278us  33.272us  119.31us  cudaMemcpyAsync
                    0.35%  3.6351ms         1  3.6351ms  3.6351ms  3.6351ms  cudaEventRecord
                    0.19%  2.0124ms         1  2.0124ms  2.0124ms  2.0124ms  cuDeviceGetPCIBusId
                    0.14%  1.4299ms         1  1.4299ms  1.4299ms  1.4299ms  cudaMemcpy
                    0.12%  1.2745ms       207  6.1560us  2.1140us  360.85us  cudaFuncGetAttributes
                    0.02%  228.90us       259     883ns     261ns  39.253us  cudaDeviceGetAttribute
                    0.02%  194.47us        51  3.8130us  1.9740us  7.0230us  cudaStreamSynchronize
                    0.02%  191.49us       259     739ns     280ns  2.5450us  cudaGetDevice
                    0.01%  52.197us       310     168ns      91ns     701ns  cudaPeekAtLastError
                    0.00%  36.684us       155     236ns     110ns     450ns  cudaGetLastError
                    0.00%  15.450us        97     159ns     100ns  1.9130us  cuDeviceGetAttribute
                    0.00%  10.850us         2  5.4250us  1.1020us  9.7480us  cudaEventCreate
                    0.00%  4.2670us         3  1.4220us     170ns  3.7270us  cuDeviceGetCount
                    0.00%  2.4740us         2  1.2370us     130ns  2.3440us  cuDeviceGet
                    0.00%     702ns         1     702ns     702ns     702ns  cuDeviceGetName
                    0.00%     431ns         1     431ns     431ns     431ns  cuDeviceTotalMem
                    0.00%     161ns         1     161ns     161ns     161ns  cuDeviceGetUuid
```

Результаты

Время выведено в ***миллисекундах***.

В каждой ячейке 1-ая конфигурация – функция swapRows(…), вторая конфигурация – Gauss(…).

Конфигурация	Размер тестового файла				
	$2 * 2$	$10 * 10$	$100 * 100$	$1000 * 1000$	$10000 * 10000$
CPU	3	7	4127	$4.04515 * 10^5$	$> 6*10^5$
<<<(16, 16)>>>, <<<(16, 16), (16, 16)>>>	1.374912	**1.319232**	**11.667744**	1015.581482	126497.867188
<<<(64, 64) >>>, <<<(32, 32), (16, 16)>>>	**0.434176**	1.512800	11.868640	1020.329590	**124948.343750**
<<<(128, 128) >>> <<<(32, 32), (32, 32)>>>	0.473920	1.522048	13.062144	1003.864075	125633.125000
<<<(256, 256) >>> <<<(64, 64), (32, 32)>>>	0.536416	2.046048	17.113632	**840.539368**	127811.914062
<<<(512, 512) >>> <<<(64, 64), (32, 32)>>>	0.452736	2.095424	17.721151	1025.087036	128407.757812
<<<(1024,1024) >>> <<<(128, 128), (32, 32)>>>	0.833536	3.933632	36.483391	1052.189697	137199.546875

Выводы

Метод Гаусса – алгоритм, приводящий матрицу к верхнетреугольному виду, очень распространен и имеет множество практических применений. Такой вид представление матрицы упрощает взаимодействие с ней и позволяет быстрее и проще выполнять различные манипуляции, такие как решение СЛАУ, нахождение обратной матрицы, определение ранга матрицы, вычисление определителя, разложение матрицы и т. д.

Реализовав данный алгоритм на CPU и GPU и проведя сравнение на различных тестовых данных, приходим к выводу, что на GPU этот алгоритм на некоторых тестовых данных работает в десятки, а то и в сотни раз быстрее. А хранение матрицы в линейном массиве по столбцам позволяет более удобным образом реализовать этот метод на GPU.