МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №*1***

**по курсу «Параллельная обработка данных»**

***Работа с матрицам. Метод Гаусса.***

Выполнил: *А. Ю. Голов*

Группа: *М8О-401*

Преподаватель: А.Ю. Морозов

Москва, 2025

**Условие**

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.

Вариант 2. Вычисление обратной матрицы.

**Программное и аппаратное обеспечение**

Дать характеристики графического процессора (compute capability, графическая память, разделяемая память, константная память, количество регистров на блок, максимальное количество блоков и нитей, количество мультипроцессоров), процессора, оперативной памяти и жесткого диска. Описать программное обеспечение (ОS, IDE, compiler и тд.).

Видеокарта: NVIDIA GeForce RTX 2060 Mobile

* **Compute Capability**: 7.5
* **Графическая память**: 6 ГБ GDDR6, с 192-битной шиной и пропускной способностью 336 ГБ/с
* **Разделяемая память**: до 64 КБ на мультипроцессор
* **Константная память**: 64 КБ
* **Количество регистров на блок**: 65 536
* **Максимальное количество блоков на мультипроцессор**: 16
* **Максимальное количество нитей на мультипроцессор**: 1 024
* **Количество мультипроцессоров (SM)**: 30

Процессор: **Intel Core i7-9750H** имеет следующие характеристики:

* **Количество ядер**: 6
* **Количество потоков**: 12
* **Техпроцесс**: 14 нм

Оперативная память:

* Объём: 16 ГБ
* **Тактовая частота:** 3500 MHz
* **Поколение**: DDR4

Жёсткий диск:

* **Объём: 512 ГБ**
* Формат: SSD M2

Программное обеспечение:

* **Операционная система: Ubuntu 24.04 LTS**
* IDE: Lunar Vim

**Метод решения**

Этот метод нахождения обратной матрицы основан на **методе Гаусса-Жордана**. Он включает несколько основных этапов:

1. **Формирование расширенной матрицы**: к исходной квадратной матрице AA размером n×nn \times n дописывается единичная матрица такого же размера, образуя матрицу размером n×2nn \times 2n.
2. **Прямой ход (приведение к треугольному виду)**:
   * Для каждой строки выбирается ведущий элемент (обычно наибольший по модулю в текущем столбце).
   * Если необходимо, строки меняются местами, чтобы ведущий элемент оказался на диагонали.
   * Строка нормируется, то есть ведущий элемент становится равным 1, а остальные элементы строки делятся на него.
   * Из всех строк ниже ведущего элемента вычитается соответствующая линейная комбинация, чтобы обнулить элементы в текущем столбце.
3. **Обратный ход (приведение к единичной матрице)**:
   * Аналогично, из строк выше ведущего элемента вычитается линейная комбинация строк, чтобы получить единичную матрицу слева.
4. **Выделение обратной матрицы**: после выполнения всех преобразований вторая половина расширенной матрицы становится обратной матрицей A−1A^{-1}.

Этот метод хорошо параллелизуется на GPU за счет независимых операций с элементами строк.

**Описание программы**

SwapLines  
Меняет местами столбцы i и j как в исходной матрице matrix, так и в расширенной матрице unitedMatrix. Параллельное исполнение позволяет ускорить обмен элементов в разных строках.

\_\_global\_\_ void SwapLines(double \*matrix, double \*unitedMatrix, int n, int i, int j)

{

int posX = blockIdx.x \* blockDim.x + threadIdx.x;

int shift = gridDim.x \* blockDim.x;

double tmp;

for (int k = posX; k < n; k += shift)

{

tmp = matrix[n \* k + i];

matrix[n \* k + i] = matrix[n \* k + j];

matrix[n \* k + j] = tmp;

tmp = unitedMatrix[n \* k + i];

unitedMatrix[n \* k + i] = unitedMatrix[n \* k + j];

unitedMatrix[n \* k + j] = tmp;

}

}

Divide  
Делит все элементы столбца i расширенной матрицы unitedMatrix на ведущий элемент matrix[i \* n + i], нормируя строку так, чтобы ведущий элемент стал равен 1.

\_\_global\_\_ void Divide(double\* matrix, double\* unitedMatrix, int n)

{

int posX = blockIdx.x \* blockDim.x + threadIdx.x;

int posY = blockIdx.y \* blockDim.y + threadIdx.y;

int shiftX = gridDim.x \* blockDim.x;

int shiftY = gridDim.y \* blockDim.y;

for (int i = posX; i < n; i += shiftX){

for (int j = posY; j < n; j += shiftY){

unitedMatrix[j \* n + i] /= matrix[i \* n + i];

}

}

}

DelLower  
Обнуляет элементы ниже главной диагонали в текущем столбце sep, выполняя элементарные преобразования строк. Корректирует не только исходную, но и расширенную матрицу.

\_\_global\_\_ void DelLower (double\* matrix, double\* unitedMatrix, int n, int sep)

{

int posX = blockIdx.x \* blockDim.x + threadIdx.x;

int posY = blockIdx.y \* blockDim.y + threadIdx.y;

int shiftX = gridDim.x \* blockDim.x;

int shiftY = gridDim.y \* blockDim.y;

for (int i = sep + 1 + posX; i < n; i += shiftX)

{

double div = -matrix[sep \* n + i] / matrix[sep \* n + sep];

for (int j = sep + 1 + posY; j < n; j += shiftY) {

matrix[j \* n + i] += div \* matrix[j \* n + sep];

}

for (int j = posY; j < n; j += shiftY) {

unitedMatrix[j \* n + i] += div \* unitedMatrix[j \* n + sep];

}

}

}

DelUpper  
Аналогично DelLower, но обнуляет элементы выше главной диагонали в столбце sep, завершая процесс приведения матрицы к диагональной форме.

\_\_global\_\_ void DelUpper (double\* matrix, double\* unitedMatrix, int n, int sep)

{

int posX = threadIdx.x + blockIdx.x \* blockDim.x;

int posY = threadIdx.y + blockIdx.y \* blockDim.y;

int shiftX = gridDim.x \* blockDim.x;

int shiftY = gridDim.y \* blockDim.y;

for (int i = sep - posX - 1; i >= 0; i -= shiftX)

{

double div = -matrix[sep \* n + i] / matrix[sep \* n + sep];

for (int j = posY; j < n; j += shiftY) {

unitedMatrix[j \* n + i] += div \* unitedMatrix[j \* n + sep];

}

}

}

**Результаты**

Были проведены замеры производительности вычислений обратной матрицы для матриц размерами 100х100, 500х500 и 1000х1000. Все данные приведены в миллисекундах.

|  |  |  |
| --- | --- | --- |
| Размер матрицы | GPU | CPU |
| 100x100 | 5.913536 | 33.2102 |
| 500x500 | 37.259071 | 1450.73 |
| 1000x1000 | 210.014435 | 12055.4 |

Исследование производительности программы при помощи утилиты nvprof при вычислении обратной матрицы для матрицы размером 3х3:

➜ lab4 git:(main) ✗ nvprof --print-gpu-trace ./a.out

==35828== NVPROF is profiling process 35828, command: ./a.out

Size: 3 x 3, Time: 1.876576 ms

==35828== Profiling application: ./a.out

==35828== Profiling result:

Start Duration Grid Size Block Size Regs\* SSMem\* DSMem\* Size Throughput SrcMemType DstMemType Device Context Stream Name

3.19476s 928ns - - - - - 72B 73.992MB/s Pageable Device NVIDIA GeForce 1 7 [CUDA memcpy HtoD]

3.19476s 544ns - - - - - 72B 126.22MB/s Pageable Device NVIDIA GeForce 1 7 [CUDA memcpy HtoD]

3.19652s 3.5840us (1 1 1) (256 1 1) 46 0B 160B - - - - NVIDIA GeForce 1 7 \_ZN6thrust20THRUST\_200700\_520\_NS8cuda\_cub4core13\_kernel\_agentINS1\_8\_\_reduce11ReduceAgentINS0\_12zip\_iteratorINS0\_5tupleIJNS0\_10device\_ptrIdEENS1\_19counting\_iterator\_tIlEEEEEEEPNS7\_IJdlEEESE\_iNS1\_9\_\_extrema9arg\_max\_fIdl11TComparatorEEEEJSD\_SF\_iSJ\_EEEvDpT0\_ [159]

3.19655s 1.5680us - - - - - 16B 9.7314MB/s Device Pageable NVIDIA GeForce 1 7 [CUDA memcpy DtoH]

3.19657s 3.1040us (256 1 1) (256 1 1) 36 0B 0B - - - - NVIDIA GeForce 1 7 SwapLines(double\*, double\*, int, int, int) [173]

3.19657s 9.7600us (32 16 1) (32 16 1) 54 0B 0B - - - - NVIDIA GeForce 1 7 DelLower(double\*, double\*, int, int) [174]

3.19658s 2.7520us (1 1 1) (256 1 1) 46 0B 160B - - - - NVIDIA GeForce 1 7 \_ZN6thrust20THRUST\_200700\_520\_NS8cuda\_cub4core13\_kernel\_agentINS1\_8\_\_reduce11ReduceAgentINS0\_12zip\_iteratorINS0\_5tupleIJNS0\_10device\_ptrIdEENS1\_19counting\_iterator\_tIlEEEEEEEPNS7\_IJdlEEESE\_iNS1\_9\_\_extrema9arg\_max\_fIdl11TComparatorEEEEJSD\_SF\_iSJ\_EEEvDpT0\_ [197]

3.19659s 1.0560us - - - - - 16B 14.450MB/s Device Pageable NVIDIA GeForce 1 7 [CUDA memcpy DtoH]

3.19661s 2.5280us (256 1 1) (256 1 1) 36 0B 0B - - - - NVIDIA GeForce 1 7 SwapLines(double\*, double\*, int, int, int) [211]

3.19661s 9.5680us (32 16 1) (32 16 1) 54 0B 0B - - - - NVIDIA GeForce 1 7 DelLower(double\*, double\*, int, int) [212]

3.19662s 9.8240us (32 16 1) (32 16 1) 42 0B 0B - - - - NVIDIA GeForce 1 7 DelUpper(double\*, double\*, int, int) [213]

3.19663s 9.4080us (32 16 1) (32 16 1) 42 0B 0B - - - - NVIDIA GeForce 1 7 DelUpper(double\*, double\*, int, int) [214]

3.19664s 6.0800us (32 16 1) (32 16 1) 40 0B 0B - - - - NVIDIA GeForce 1 7 Divide(double\*, double\*, int) [215]

Regs: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or tools and can be more than what the compiler shows.

SSMem: Static shared memory allocated per CUDA block.

DSMem: Dynamic shared memory allocated per CUDA block.

SrcMemType: The type of source memory accessed by memory operation/copy

DstMemType: The type of destination memory accessed by memory operation/copy

И, соответственно, профилирование для того же исполняемого файла:  
 lab4 git:(main) ✗ nvprof ./a.out

==36067== NVPROF is profiling process 36067, command: ./a.out

Size: 3 x 3, Time: 1.715936 ms

==36067== Profiling application: ./a.out

==36067== Profiling result:

Type Time(%) Time Calls Avg Min Max Name

GPU activities: 32.99% 20.256us 2 10.128us 9.6960us 10.560us DelLower(double\*, double\*, int, int)

30.64% 18.816us 2 9.4080us 9.2800us 9.5360us DelUpper(double\*, double\*, int, int)

10.27% 6.3050us 2 3.1520us 2.7200us 3.5850us \_ZN6thrust20THRUST\_200700\_520\_NS8cuda\_cub4core13\_kernel\_agentINS1\_8\_\_reduce11ReduceAgentINS0\_12zip\_iteratorINS0\_5tupleIJNS0\_10device\_ptrIdEENS1\_19counting\_iterator\_tIlEEEEEEEPNS7\_IJdlEEESE\_iNS1\_9\_\_extrema9arg\_max\_fIdl11TComparatorEEEEJSD\_SF\_iSJ\_EEEvDpT0\_

9.95% 6.1120us 1 6.1120us 6.1120us 6.1120us Divide(double\*, double\*, int)

9.07% 5.5680us 2 2.7840us 2.4960us 3.0720us SwapLines(double\*, double\*, int, int, int)

4.74% 2.9120us 2 1.4560us 1.0250us 1.8870us [CUDA memcpy DtoH]

2.34% 1.4380us 2 719ns 511ns 927ns [CUDA memcpy HtoD]

API calls: 96.72% 166.38ms 4 41.596ms 3.4670us 166.37ms cudaMalloc

2.21% 3.7964ms 114 33.301us 337ns 2.4195ms cuDeviceGetAttribute

0.90% 1.5464ms 1 1.5464ms 1.5464ms 1.5464ms cudaFuncGetAttributes

0.04% 72.675us 4 18.168us 2.9890us 61.792us cudaFree

0.03% 53.900us 9 5.9880us 2.4160us 22.545us cudaLaunchKernel

0.03% 44.859us 1 44.859us 44.859us 44.859us cuDeviceGetName

0.02% 29.495us 2 14.747us 9.0360us 20.459us cudaMemcpyAsync

0.02% 27.677us 1 27.677us 27.677us 27.677us cudaEventSynchronize

0.01% 22.411us 2 11.205us 3.4330us 18.978us cudaMemcpy

0.01% 12.623us 2 6.3110us 3.5690us 9.0540us cudaEventRecord

0.00% 8.5070us 4 2.1260us 681ns 3.8790us cudaStreamSynchronize

0.00% 5.5490us 1 5.5490us 5.5490us 5.5490us cuDeviceGetPCIBusId

0.00% 5.1770us 2 2.5880us 374ns 4.8030us cudaEventCreate

0.00% 4.1750us 3 1.3910us 352ns 3.0230us cuDeviceGetCount

0.00% 2.6560us 2 1.3280us 388ns 2.2680us cuDeviceGet

0.00% 2.6220us 42 62ns 48ns 317ns cudaGetLastError

0.00% 2.4280us 11 220ns 138ns 455ns cudaGetDevice

0.00% 1.6740us 6 279ns 138ns 736ns cudaDeviceGetAttribute

0.00% 1.3410us 1 1.3410us 1.3410us 1.3410us cuModuleGetLoadingMode

0.00% 1.1640us 1 1.1640us 1.1640us 1.1640us cuDeviceTotalMem

0.00% 1.0930us 1 1.0930us 1.0930us 1.0930us cudaEventElapsedTime

0.00% 776ns 1 776ns 776ns 776ns cuDeviceGetUuid

0.00% 317ns 4 79ns 50ns 120ns cudaPeekAtLastError

0.00% 145ns 1 145ns 145ns 145ns cudaGetDeviceCount

**Выводы**

Данный метод нахождения обратной матрицы с использованием **параллельных вычислений на GPU** показал значительное ускорение по сравнению с последовательным вычислением на CPU. Благодаря параллельному выполнению операций над строками и столбцами, алгоритм эффективно использует ресурсы графического процессора, минимизируя время ожидания.

При увеличении размера матрицы разница в скорости выполнения становится **особенно заметной**, так как GPU обрабатывает сразу **тысячи элементов одновременно**, тогда как CPU вынужден выполнять операции последовательно или с ограниченной параллельностью.

Таким образом, для задач, связанных с линейной алгеброй и матричными преобразованиями, **использование GPU дает колоссальное преимущество** в скорости по сравнению с традиционными вычислениями на процессоре.