МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Лабораторная работа №2
по курсу «Технологии параллельного программирования**»**

Работа с матрицам. Метод Гаусса.

Выполнил: Филиппов Владимир Михайлович
Группа: М8О-410Б-22
Преподаватели: А.Ю. Морозов,
Е.Е. Заяц

Москва, *2025*

Условие

1. 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).
2. Вычисление детерминанта матрицы.

Программное и аппаратное обеспечение

GPU.

• Compute Capability: 8.9
• Графическая память (Global memory): 8 GB GDDR6 (8188 MiB — видно в nvidia-
smi)
• Shared memory per SM (мультипроцессор): до 100 KB (может конфигурироваться:
64 KB shared + 32 KB L1 или наоборот)
• Shared memory per block: до 48 KB
• Constant memory: 64 KB (аппаратно выделено)
• Регистры на потоковый мультипроцессор (SM): 65,536
• Максимум регистров на блок: 32K (зависит от конфигурации)
• Максимальное число потоков на блок: 1024
• Максимальное число блоков в сетке (grid): 2^31 - 1 по X, 2^16 - 1 по Y и Z
• Количество мультипроцессоров (SM): 24 (у RTX 4060)
• Макс. потоков на SM: 2048
• Итого потоков на GPU: 24 × 2048 = 49,152 одновременно резидентных

CPU.

• Архитектура. x86_64
• Модель. AMD Ryzen 5 3600 6-Core Processor
• Количество физических ядер. 6
• Потоков. 12 (2 потока на ядро)
• Частота (BogoMIPS). 7200.05
• Кэш L1 (данные/инструкции). 192 KiB / 192 KiB
• Кэш L2. 3 MiB
• Кэш L3. 16 MiB

RAM.

• Общий объём. 15 GiB
• Swap. 4 GiB (свободен)

SSD.• Объем. 1 TB
• Использовано. 156 GB

ПО.

• OS. Windows 11 + WSL 2.0.
• IDE. Visual studio code.
• NVCC. 11.5 (Build V11.5.119)

Метод решения

Для решения задачи используется параллельный алгоритм метода Гаусса. В результате получается LU матрица, произведение диагональных элементов с учётом знака является определителем изначальной матрицы. Знак зависит от чётности количества перестановок рядов во время вычисления матрицы.

Описание программы

Всего используется 3 ядра (swapRows, divide, kernel) и функция поиска максимального элемента в столбце при помощи Thrust. Первое ядро по столбцам меняет местами текущую строку и строку с найденным максимальным элементом. Перестановка влияет на знак определителя. Второе ядро вычисляет коэффициенты матрицы L в текущем столбце. Третье занимается основным процессом в методе Гаусса – занулением элементов под диагональным.

Результаты.

У меня возникло множество проблем с профилированием и библиотекой Thrust. Я работаю на компьютере с видеокартой Nvidia 4060 из под WSL 2. Сначала у меня не работал Thrust, поскольку почему-то считал, что я компилирую программу не для архитектуры sm-89. Затем оказалось, что на этой архитектуре не поддерживается nvprof, поэтому пришлось использовать ncu. Однако с ним тоже возникли проблемы, поскольку из под WSL как-то криво собирается статистика по GPU.

Какое-то время у меня тестирующая система не принимала лабораторную с TE. Я это исправил, начав использовать коалесцированный доступ к памяти.

Вывод nsys представлен на рисунке 1.

** CUDA API Summary (cuda_api_sum):

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
35.7	12687448	198	64078.0	56283.0	29715	650145	47390.3	cudaDeviceSynchronize
17.3	6157087	101	60961.3	52519.0	35673	580033	53086.3	cudaMemcpy
17.2	6089601	99	61511.1	57426.0	45022	94648	9047.4	cudaMemcpyAsync
12.5	4442657	297	14958.4	11028.0	8840	523582	30556.1	cudaLaunchKernel
9.9	3511530	198	17735.0	3385.0	835	199750	21763.9	cudaStreamSynchronize
5.2	1862598	100	18626.0	3840.0	3044	1349256	134491.5	cudaMalloc
2.1	752612	100	7526.1	2860.0	2459	368258	36885.9	cudaFree
0.0	1126	1	1126.0	1126.0	1126	1126	0.0	cuModuleGetLoadingMode

Рисунок 1. Замеры времени исполнения утилитой nsys

К сожалению, тут мы не видим скорость выполнения ядер, только функции CUDA API. Думаю, что при выполнении следующей лабораторной я поставлю настоящий Linux.

	CPU (мс)	GPU(мс)
1000x1000	5996.938	1046.395
2000x2000	42595.561	1990.289
3000x3000	146567.751	3217.704

Таблица 1. Сравнительная характеристика времени исполнения на GPU и CPU.

В таблице мы наглядно видим, насколько быстрее работает программа на GPU по сравнении с версией на CPU (реализация которой была благополучно взята из лабораторных работ по численным методам).

Выводы

Данный алгоритм используется, например, в численных методах линейной алгебры для анализа систем линейных уравнений, вычисления обратных матриц или определения объёма многомерных параллелепипедов через определитель матрицы. С его помощью можно эффективно вычислять определитель любой квадратной матрицы, что является важной операцией в инженерных, физических и экономических расчётах.
В реализации алгоритма применяется метод Гаусса с выбором главного элемента: матрица приводится к верхнетреугольной форме посредством последовательного зануления элементов под диагональю. Определитель исходной матрицы вычисляется как произведение диагональных элементов верхнетреугольной матрицы с учётом знака, зависящего от количества перестановок строк. Несмотря на простоту идеи, вычисление требует выполнения операций с каждой строкой и каждым элементом под диагональю, то есть сложность кубического порядка.

Кроме того, я познакомился с утилитами ncu, nvprof и nsys для профилировки программ на GPU. Узнал, что они некорректно работают с WSL + Windows из-за несовместимости драйверов. Кроме того, познакомился с коалесцированным доступом к памяти, который помог мне кратно ускорить выполнение программы.