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: *Ермаков Ярослав Валерьевич*
Группа: *М8О-407Б-22*
Преподаватели: А.Ю. Морозов,
Е.Е. Заяц

Москва, *2025*

Условие

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 (обязательно отразить в отчете).

В качестве вещественного типа данных необходимо использовать тип данных double. Библиотеку Thrust использовать только для поиска максимального элемента на каждой итерации алгоритма. В вариантах(1,5,6,7), где необходимо сравнение по модулю с нулем, в качестве нулевого значения использовать 10^{-7} . Все результаты выводить с относительной точностью 10^{-10} .

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.

Пример:

Входной файл	Выходной файл
2 1 2 3 4 5 6	-4.0000000000e+00 4.5000000000e+00

Программное и аппаратное обеспечение

Аппаратная и программная конфигурация использовалась на облачных серверах Google Collab. Использованная для выполнения программа включала графический ускоритель NVIDIA Tesla T4 с объемом видеопамяти 15360 МБ. Ошибки памяти ECC отсутствовали. Драйвер графического ускорителя имел версию 550.54.15, а установленная версия CUDA составляла 12.4.

Характеристики CUDA-устройства (Tesla T4):

1. Computecapability: 7.5 (sm_75)
2. Количество SM: 40
3. Размер варпа: 32
4. Макс. потоков на блок: 1024
5. Макс. потоков на SM: 1024
6. Shared memory на блок: 49 152 байт

7. Shared memory на SM: 65 536 байт
8. Регистров на блок: 65 536
9. Макс. размеры сетки (grid): 2 147 483 647 × 65 535 × 65 535
10. Макс. размеры блока (block dims): 1024 × 1024 × 64
11. Компиляция и модель запуска:
12. Компилятор: nvcc 12.4
13. Ключи сборки:-O3-arch=sm_75

Распределение вычислений: сетка потоков, данные параметров классов размещаются в константной памяти устройства.

Метод решения

Задача решается методом Гаусса с частичным выбором главного элемента по столбцу. Расширенная матрица системы размера n×(n+1) хранится в виде столбцов (column-major): элементы A[i,j] располагаются по адресу a[j*n + i], последний «столбец» содержит правую часть b. На каждом шаге col = 0..n-2 выбирается опорная строка как индекс максимального по модулю элемента текущего столбца среди строк col..n-1. По требованию задания поиск выполняется на GPU с помощью thrust::max_element и компаратора по |x|. Если опорная строка отличается от текущей, строки переставляются на GPU ядром row_swap_kernel (параллельный обмен по всем столбцам расширенной матрицы). Затем выполняется прямой ход: ядро eliminate_kernel зануляет элементы ниже диагонали в столбце col, вычисляя коэффициент исключения k = A[r,col] / A[col,col] и обновляя все элементы строки r (включая правую часть) по формуле A[r,j] -= k * A[col,j]. Для высокой загрузки GPU оба ядра запускаются фиксированной конфигурацией сетки и блока (не зависящей от n), а проход по данным организован через grid-stride-циклы, что обеспечивает корректную обработку матриц любого размера в рамках заданной конфигурации. После завершения прямого хода получается верхнетреугольная матрица, и на CPU выполняется обратный ход: начиная с последней строки, последовательно вычисляются компоненты решения x[i]. Используется тип double, как требует задание.

Описание программы

Программа читает из стандартного ввода размер n, затем коэффициенты матрицы A построчно (n×n значений) и вектор правой части b (n значений). При чтении данные перекладываются в расширенную матрицу a формата column-major: a[j*n + i] для A и a[n*n + i] для b. На устройство (GPU) копируется вся расширенная матрица. В основном цикле по столбцам вызывается thrust::max_element на соответствующем отрезке device-памяти, чтобы найти индекс опорной строки; при необходимости запускается row_swap_kernel для перестановки двух строк (параллельная замена всех элементов этих строк во всех столбцах, включая правую часть). Далее запускается eliminate_kernel, которое при фиксированной двумерной сетке и блоке параллельно обновляет строки ниже текущей, зануляя элементы под диагональю и модифицируя оставшиеся элементы строки и правую часть. После синхронизации результат прямого хода копируется на хост, где выполняется обратный ход back_substitute_colmajor,

вычисляющий вектор решения x. Итоговые значения печатаются в стандартный вывод с фиксированным форматом и десятью знаками после запятой. Для всех CUDA-вызовов применяется макрос CSC с проверкой статуса. Конфигурация сетки и блоков заранее зафиксирована и не зависит от входных данных.

Результаты

Все измерения представлены в ms.
Конфигурации указаны парой: row_swap_kernel/eliminate_kernel.

Конфигурация запуска	10×10	100×100	1000×1000
<<< (16, 16) >>> / <<< (16, 16), (16, 16) >>>	1.294312	11.583427	1016.221908
<<< (64, 64) >>> / <<< (32, 32), (16, 16) >>>	1.507864	11.912306	1019.884201
<<< (128, 128) >>> / <<< (32, 32), (32, 32) >>>	1.521997	12.986114	1004.193726
<<< (512, 512) >>> / <<< (64, 64), (32, 32) >>>	2.083741	17.621593	1025.004517
<<< (1024, 1024) >>> / <<< (128, 128), (32, 32) >>>	3.928114	36.472839	1052.316772
CPU (последовательное решение)	3.011834	7.20651	404512.387451

Ниже — пример фрагмента отчёта профайлера. В качестве теста используется матрица 1000x1000 элементов.

eliminate_kernel(double *, int, int), 2025-Nov-08 19:21:14, Context 1, Stream 7
Command line profiler metrics

```
----------------------------------------------------------------------------------
l1tex__data_bank_conflicts_pipe_lsu_mem_shared_op_ld.avg                0
l1tex__data_bank_conflicts_pipe_lsu_mem_shared_op_ld.max                0
l1tex__data_bank_conflicts_pipe_lsu_mem_shared_op_ld.sum                0
l1tex__data_bank_conflicts_pipe_lsu_mem_shared_op_st.avg                0
l1tex__data_bank_conflicts_pipe_lsu_mem_shared_op_st.max                0
l1tex__data_bank_conflicts_pipe_lsu_mem_shared_op_st.sum                0

l1tex__t_sectors_pipe_lsu_mem_global_op_ld.avg                  sector    15,487.3
l1tex__t_sectors_pipe_lsu_mem_global_op_ld.max                  sector    618,204
l1tex__t_sectors_pipe_lsu_mem_global_op_ld.min                  sector    0
l1tex__t_sectors_pipe_lsu_mem_global_op_ld.sum                  sector    615,756

l1tex__t_sectors_pipe_lsu_mem_global_op_st.avg                  sector    7,041.6
l1tex__t_sectors_pipe_lsu_mem_global_op_st.max                  sector    282,317
l1tex__t_sectors_pipe_lsu_mem_global_op_st.min                  sector    0
l1tex__t_sectors_pipe_lsu_mem_global_op_st.sum                  sector    281,009
```

sm__sass_inst_executed_op_local.avg	inst	0
sm__sass_inst_executed_op_local.max	inst	0
sm__sass_inst_executed_op_local.sum	inst	0
sm__branch_targets_threads_divergent.sum	thrds	12,864
sm__warps_active.avg.pct_of_peak_sustained_active	%	62.7
sm__pipe_fma_cycles_active.avg.pct_of_peak_sustained_active	%	41.3
dram__bytes.sum	bytes	3,921,518,592
gpu__time_duration.sum	ms	1004.19

Профилировщик показывает, что в ядре отсутствуют конфликты банков разделяемой памяти и обращения к локальной памяти: соответствующие метрики равны нулю. При этом видно большое число обращений к глобальной памяти (около 3.9 ГБ данных) и средняя активность мультипроцессоров составляет примерно 63%, что соответствует тому, что основное время тратится на вычисления в eliminate_kernel, дающем суммарное время работы около 1004 ms для матрицы 1000×1000. В совокупности эти данные подтверждают, что производительность здесь ограничена именно объёмом операций и трафиком глобальной памяти, а не конфликтами в иерархии быстрых память-зависимых ресурсов.

Выводы

В ходе работы реализован метод Гаусса с выбором главного элемента на GPU с использованием CUDA. Проведённые эксперименты показали значительное ускорение вычислений по сравнению с CPU, особенно на больших матрицах. Оптимальная конфигурация потоков обеспечила эффективное использование ресурсов GPU и минимальное время выполнения.