МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №*1*
по курсу «*Параллельные и распределенные вычисления*»

Освоение программного обеспечения для работы с технологией CUDA. Примитивные операции над векторами.

Выполнил: *Новиков Сергей Сергеевич*
Группа: М8О-311Б-22
Преподаватель: А.Ю. Морозов
Е.Е. Заяц

Москва, 2025

Условие

Ознакомление и установка программного обеспечения для
работы с программно-аппаратной архитектурой параллельных вычислений(CUDA).
Реализация одной из примитивных операций над векторами.

Вариант 4.

Входные данные. На первой строке задано число n -- размер векторов. В следующих 2-х строках, записано по n вещественных чисел -- элементы векторов.

Выходные данные. Необходимо вывести n чисел -- результат поэлементного нахождения минимума исходных векторов.

Программное и аппаратное обеспечение

GPU Name: AMD Radeon Graphics
Compute Capability: 9.0
Total Global Memory: 9679 MB
Shared Memory Per Block: 64 KB
Constant Memory: 2097151 KB
Registers Per Block: 65536
Max Threads Per Block: 1024
Multiprocessors: 6
CPU Cores: 6
Total RAM: 19358 MB
Total Disk Space: 233 GB
Operating System: Linux
Compiler: GCC 4.2

Метод решения

Решение данной задачи можно разбить на 3 пункта. Сначала необходимо считать ввод в буфер на хост-устройстве. Затем на устройстве с помощью CUDA-ядра вычисляется покомпонентный минимум двух векторов. В завершение результат копируется обратно на хост и выводится на экран.

Описание программы

Ядро `VectorPerElemMinKernel` вычисляет покомпонентный минимум двух входных массивов элементов типа `double`. Оно рассчитано на фиксированное количество нитей (до 1024) и обрабатывает необходимые элементы в цикле, используя стратегию разбиения по блокам. Управление вычислениями организовано с помощью `CudaGraph`, который позволяет захватывать и повторно запускать граф вычислений для повышения производительности

Результаты

Размер входных данных	<<<1, 32>>>	<<<32, 32>>>	<<<256, 256>>>	<<<1024, 1024>>>	CPU (1 thread)
1000	1.50635	0.049002	0.041508	0.092824	0.000908
10000	0.414698	0.085129	0.078667	0.108714	0.000978
100000	3.57339	0.592632	0.636334	0.760447	0.000908
1000000	45.2393	4.15666	2.97783	2.88373	0.001467
3000000	133.847	8.74622	8.35334	7.79671	0.001397

Показано время в наносекундах(нс)

Выводы

Данный алгоритм представляет интерес для тренировки навыков гетерогенного программирования с использованием CUDA. Однако для небольших входных данных выполнение на CPU оказывается быстрее, поскольку исключаются затраты на копирование данных между хостом и устройством, а также ожидание запуска ядра на GPU.

- Для малых векторов процессор превосходит видеокарту, так как накладные расходы на передачу данных и запуск вычислений на GPU превышают саму вычислительную нагрузку.
- По мере увеличения размера входных данных видеокарта начинает демонстрировать свою эффективность, особенно при большом количестве потоков и блоков.
- Тем не менее, даже на больших входных данных выигрыш не становится значительным, так как алгоритм выполняет простую по компонентную операцию, основное время уходит на доступ к глобальной памяти и передачу буферов между CPU и GPU.

Использование CudaGraph позволяет уменьшить накладные расходы за счет повторного использования графа вычислений, однако его эффективность зависит от конкретного железа и параметров запуска.