МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Лабораторная работа №1
по курсу «Программирование графических процессоров»

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

Выполнил: А. Л. Ядров
Группа: 8О-408Б
Преподаватель: А. Ю. Морозов,
К. Г. Крашенинников

Москва, 2023

Условие

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

Вариант: 4. Поэлементное нахождение минимума векторов.

Программное и аппаратное обеспечение

Характеристики графического процессора:

- Наименование: NVIDIA GeForce GTX 1050
- Compute capability: 6.1
- Графическая память: 2047 Mb
- Разделяемая память на блок: 48 Kb
- Количество регистров на блок: 65536
- Максимальное количество потоков на блок: 1024
- Максимальное количество блоков: [1024, 1024, 64]
- Константная память: 64 Kb
- Количество мультипроцессоров: 5

Характеристики системы:

- Процессор: Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
- Память: 8192 Mb

Программное обеспечение:

- ОС: Windows 11 (22H2)
- IDE: Microsoft Visual Studio 2022 (Comminity Edition)
- Компилятор: nvcc V12.2.140, MSVC V19.37.32824

Метод решения

Алгоритм нахождения поэлементного минимума двух векторов принимает три вектора (третий используется для вывода результата). Далее необходимо найти минимум каждого элемента из двух векторов. Очевидно, что если оба вектора принадлежат пространству R^n, то сложность будет $O(n)$. Каждый поток ищет минимум элементов двух векторов с шагом, равным общему количеству потоков, начиная со своего номера потока.

Описание программы

Проект я разделил на следущие модули:

1. $common\ defines$ - содержит множество полезных дефайнов для более приятной работы с $CUDA$ и $C++$.

2. $common\ structures$ - содержит класс-обертку над $CUDA$ событиями и класс-обертку над $std::stringstream$, позволяющим конструировать строки ($sprintf$ с возможностями $C++$).

3. $operation\ system$ - содержит операции, связанные с системными вызовами (например, получение информации о количестве оперативной памяти). Реализован только для ОС $Windows$.

4. $kernel$ - непосредственно программа, выполняющия все операции, связанные с вводом, вычислениями и выводом.

Функция ядра сводится к вычислению номера потока, общего количества потоков и циклу с заданным шагом, внутри которого вычисляется минимум двух элементов вектора.

Также для замера времени была написана программа на языке программирования $Python$, состоящая из слеудющих модулей:

1. $build$ используется для компиляции $Microsoft\ Visual\ Studio$ проекта (файла с расширением $.sln$).

2. $tests\ generator$ используется для генерации тестов и ответов на них (ответы генерируются с помощью программы, написанной под CPU).

3. $cheker$ используется для проверки работоспособности программы. Для этого запускается раннее сгенерированный $.exe$ файл с входными данными из теста, далее вывод программмы проверяется с ответом.

4. $benchmark$ используется для замера скорости работы программы. Для этого программа запускается 5 раз с входными тестовыми данными. Для каждого теста временем работы будет минимальное время работы из всех запусков данного теста.

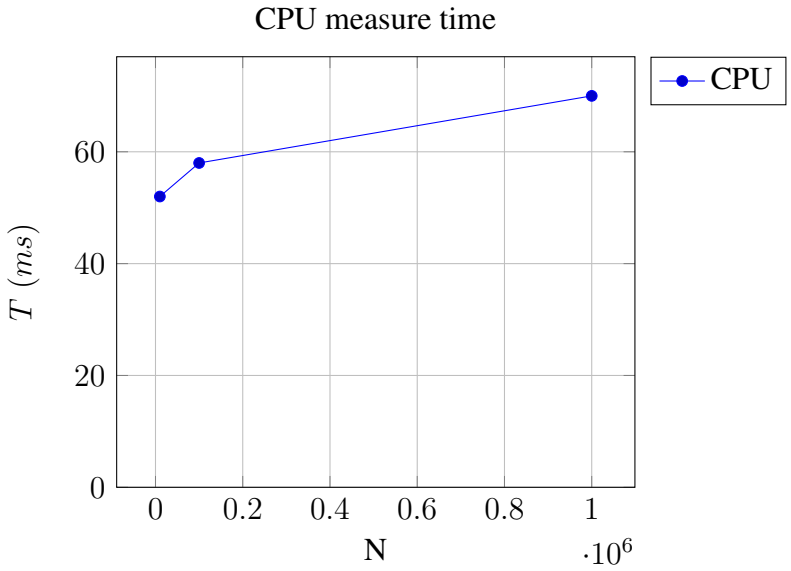
5. $main$ используется в качестве обертке над описанными выше модулями для более комфортного запуска модулей.

Результаты

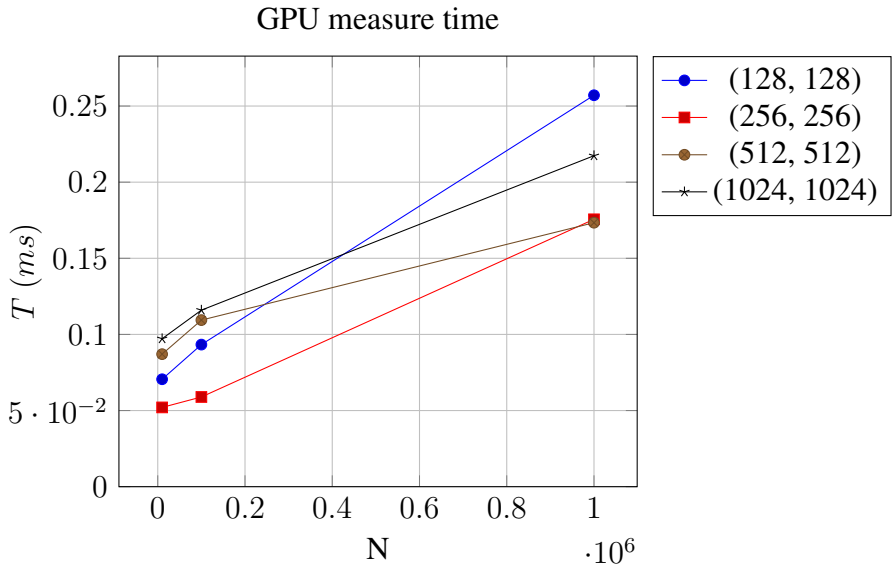
Для замеров скорости работы программы на CPU мною была написана программа, создающая множество потоков, каждый из них обрабатывает входные векторы, начиная с определенного индекса и с шагом, равным количеству векторов. Так как каждый поток только считывает данные из входных векторов, то необязательно их защищать от гонки потоков. Также каждый индекс обрабатывается только одним потоком, поэтому защищать результат от гонки потоков тоже необязательно с целью повышения производительности программы.

В замер времени работы программмы входит промежуток от создания потока до завершения работы всех потоков (метод $join()$ для каждого потока). Общее количество потоков ограничено сверху значением 1000. Также стоит отметить, что время выполнения работы сильно связано с загрузкой системы и с самой системой, так как создание потока - блокирующий системный вызов.

Замеры времени работы обоих программ проходили на одинаковых тестах. В качестве входных чисел были использованы псевдослучайные числа $c : -10^{10} \leq c \leq 10^{10}$



По графику видно, что создание потоков на CPU и ожидание их выполнения происходит достаточно долго.



В то же время исполнение программы на GPU происходит гораздо быстрее. Это связано с легковесностью потоков на GPU, а также с гораздо бóльшим количеством потоков, исполняющихся одновременно. Также из графика можно заключить, что оптимальной конфигурацией для данного оборудования будет $<<< 256, 256 >>>$. Также видно, что на маленьких данных конфигурации с маленькими значениями выигрывают конфигурации с большими значениями, так как сказывается процесс создания потоков.

Выводы

В ходе выполнения лабораторной работы я познакомился с основами технологии CUDA. Алгоритмы из данной лабораторной могут использоваться, например, при анализе больших объемов данных. Программирование для GPU оказалось легче чем я ожидал, в простейших случаях достаточно определить взаимно-однозначное отображение между элементами данных и потоками и реализовать обработку этих элементов.

По результатам выполнения видно, что небольшое число потоков заметно проигрывает однопоточному подходу. Вероятно это связано с тем, что графические ядра по отдельно-

сти слабее ядер CPU. При увеличении числа потоков, их преимущества становятся более заметны, оптимальной конфигурацией для данного графического процессора является $< 256, 256 >$.

Список литературы

[1] Бьярне Страуструп. *Программирование. Принципы и практика с использованием C++, 2-е издание.* — Издательский дом «Вильямс», 2016. Перевод с английского: И. В. Красиков. — 1328 с. (ISBN 978-5-8459-1949-6 (рус.))

[2] *CUDA Runtime API.*
URL: `https://docs.nvidia.com/cuda/cuda-runtime-api/index.html` (дата обращения: 16.12.2013).