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.

Выполнила: К.О. Михеева
Группа: 8О-407Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2023

Условие

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

Вариант задания: Поэлементное нахождение минимума векторов.качестве вещественного типа данных необходимо использовать тип данных double. Все результаты выводить с относительной точностью 10^{-10} . Ограничение: $n < 2^{25}$.

Программное и аппаратное обеспечение

Name:	Tesla T4
Compute capability:	7.5
Dedicated video memory:	15835 MB
Shared memory per block:	49152 bytes
Register per block:	65536 bytes
Total constant memory:	65536
Max threads per multiprocessor:	2048
Max threads per block:	1024
Multiprocessors const:	40

AMD RYZEN 5 5500U

Architecture:	Ryzen 5 Zen2(Lucienne)
Processor Technology for CPU Cores:	TSMC 7nm FinFET
CPU Cores:	6
Thread(s):	12
CPU Socket	FP6
CPU MHz:	3600
SMP # CPUs:	1
L2 cache: L3 cache:	3MB 8MB
RAM	16GB

SSD	512GB

OS: Windows 10 - 64-Bit Edition, Ubuntu x86 64-Bit; IDE: VS code, compiler: nvcc

Метод решения

В данной лабораторной работе был написан код на ЯП C++, который представляет собой пример использования CUDA для выполнения параллельных вычислений на GPU. Он применяется для нахождения минимальных значений между двумя векторами, что может быть полезным в различных приложениях, требующих высокой вычислительной мощности. Материалами для написания программы послужили лекции и пример, написанный преподавателем.

Описание программы

В данной лабораторной используется один главный файл, который содержит функцию main, которая является точкой входа в программу. В этой функции выполняется инициализация, выделение памяти на GPU, чтение данных из файла, запуск ядра, копирование данных между хостом и устройством, вывод результатов и освобождение памяти.

findMinKernel - это единственное реализованное ядро в программе, которое выполняется параллельно на множестве нитей GPU. Ядро принимает три аргумента: указатель на массив result для хранения результатов, указатели на массивы vector1 и vector2, а также размер n (длина векторов). Далее оно вычисляет сравнение соответствующих элементов vector1 и vector2 и записывает минимальное значение в соответствующий элемент result.

Код функции:

```
__global__ void findMinKernel(double* result, double* vector1,
double* vector2, int n) {
int idx = blockIdx.x * blockDim.x + threadIdx.x;
int offset = gridDim.x * blockDim.x;
while (idx < n) {
     result[idx] = fmin(vector1[idx], vector2[idx]);
     idx += offset;
   }
}
```

Результаты

Время исполнения программы в мс при различных данных

Размер	10^2	10^4	10^6	10^7
CPU	0.010012	0.360056	14.954000	148.028995
<1, 1>	0.057001	2.830072	258.50341	1437.09345к
<32, 32>	0.023523	0.047556	0.790810	7.211600
<256, 256>	0.032768	0.026040	0.107123	0.999288

<512, 512>	0.028012	0.037036	0.113592	0.980888
<1024, 1024>	0.038760	0.038970	0.107924	0.969632

Выводы

Описать область применения реализованного алгоритма. Указать типовые задачи, решаемые им. Оценить сложность программирования, кратко описать возникшие проблемы при решении задачи. Провести сравнение и объяснение полученных результатов.

Реализованный алгоритм нахождения минимальных значений между двумя векторами с использованием CUDA имеет широкий спектр применения в задачах, где необходимо эффективно обрабатывать большие объемы данных и проводить параллельные вычисления, такие как:

- Обработка изображений и видео;
- Обработка больших данных;
- Научные исследования.

Сложность данного алгоритма можно оценить как среднюю. Разработка программы с использованием CUDA требует знания архитектуры GPU, навыков параллельного программирования и оптимизации для максимальной производительности. Возможные проблемы могут включать в себя управление памятью на устройстве и хосте, правильность параллельных вычислений и оптимизацию работы с данными на GPU.

Сравнение производительности между CUDA и CPU зависит от конкретной задачи. Если есть большое количество данных, то должно быть обработано параллельно то есть GPU будет значительно быстрее. Однако, для небольших объемов данных, CPU может оказаться более эффективным.

Общим выводом является то, что использование CUDA и параллельных вычислений на GPU имеет большой потенциал для ускорения выполнения вычислительных задач, но требует оптимизации и адаптации к конкретным потребностям задачи.