МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Параллельные алгоритмы»

Тема: Оптимизация доступа к памяти в модели OpenCL

Студент гр. 0303 _____ Морозов А.Ю.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2023

Цель работы.

Реализовать алгоритм умножения матриц с использованием Open CL.

Задание.

Реализовать умножение матриц на Open CL. В отчете произвести сравнение производительности с CPU реализацией из лабораторной работы №4.

Выполнение работы.

Реализованная программа умножения матриц, состоит из 5 основных блоков:

1) Генерация входных матриц.
2) Выбор девайса.
3) Компиляция *kernel*-кода под выбранный девайс.
4) Запуск программы.
5) Сохранение результата в виде картинки.

Генерация входных матриц производится функцией *generate_task*, которая получает на вход размер матрицы и возвращает пару векторов целых чисел, заполненных случайными числами в пределах от -10 до 10.

Выбор девайса производится функцией *create_device*, которая посредством функций *clGetPlatformIDs* и *clGetDeviceIDs* выбирает нужный нам девайс и возвращает его *id*.

Компиляция *kernel*-кода производится функцией *build_program*, которая читает реализованный код из файла *kernel.cl* и посредством функций *clCreateProgramWithSource* и *clBuildProgram* собирает программу, готовую к исполнению, и возвращает ее.

Запуск программы осуществляется функцией *invoke_kernel*, которая перетает в собранный *kernel*-код аргументы с помощью функции *clSetKernelArg,* запускает вычисления с помощью функции *clEnqueueNDRangeKernel* и по завершению вычислений переносит результаты из предоставленного буфера на host с помощью функции *clEnqueueReadBuffer*.

Сохранение результата производится функцией *save_result*, которая создает файл и записывает в него полученную матрицу, разделяя столбцы пробелами, а строки переводами строки.

Исследование.

Для сравнения производительности был выбран вариант умножения матриц с масштабируемым разбиением по потокам. Кроме того, запуск алгоритма из 4 лабораторной работы производился на трех потоках, так как это дало максимальную производительность.

Таблица 1 – Сравнение производительности алгоритмов.

Размер матрицы	Масштабируемый, сек.	GPU-реализация, сек.
32	0.000662	0.0013
64	0.0023	0.0013
128	0.011	0.0015
256	0.059	0.0032
512	0.457	0.014
1024	4.76	0.108
2048	39.27	1.025
4096	347.454	9.556

Таблица 2 – Сравнение производительности GPU-умножения при разном размере work_group и размере матриц 1024 * 1024.

Размер work_group	Время выполнения, сек.
4	0.535
8	0.288
16	0.159
32	0.105
64	0.104
128	0.105
256	0.114

Вывод: алгоритм GPU-умножения матриц выигрывает в десятки раз по скорости у алгоритма умножения матриц с масштабируемым разбиением по потокам.

Выводы.

В ходе выполнения лабораторной работы была реализована программа, производящая умножение матриц и сохраняющая результат в файл. Было проведено исследование зависимости времени выполнения вычислений от размера work_group и сравнение производительности алгоритма на GPU с параллельным алгоритмом масштабируемого разбиения по потокам. Был сделан вывод о значительном выигрыше по времени алгоритма умножения на графическом процессоре.