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. 0304 _____ Шквиря Е.В.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург

2024

Цели работы

Используя OpenCL реализовать параллельное умножение матриц.

Постановка задачи

Реализовать умножение матриц на OpenCL. В отчете: Произвести сравнение производительности с CPU реализацией из лаб.4.

Выполнение работы

Для выполнения работы была использована библиотека OpenCL. Требовалось создать изображение, нарисовать в нём фрактал и сохранить в файл.

Выполнение кода в библиотеке OpenCL выполнялось в два этапа: настроить окружение и работать с конкретным рабочим элементом.

Для настройки окружения сначала необходимо получить устройство выполнения. Для задачи умножение матриц будет достаточно лишь одного устройства - в нашем случае GPU. Далее требуется создать контекст. Он задаёт сколько устройств будет работать, какие это будут устройства и какие у них будут настройки. Контекст в нашей задаче подразумевает использование одного устройства без дополнительных настроек. Также требуется создать очередь исполнения, которая будет определять, в каком порядке должны будут исполняться задачи конкретным устройством.

После создания очереди необходимо создать kernel. Для него нам нужно получить .cl программу и скомпилировать её в real-time. Скомпилированная программа вместе с определённым именем создаёт kernel. Теперь к нему можно привязать переменные. 3 двумерных массива. Это можно сделать, создав одномерные буферы с ручным контролем перебора индексов у них. Для первых двух буферов (для матриц перемножения) передается не только размер, но и начальное значение, когда последний (для результата) остается без последнего. Теперь буферы и переменная размера

привязываются к kernel с помощью команды `clSetKernelArg` и создается задача на вычисление в очереди. Остается дождаться конца вычислений и очистить память.

Рабочие группы играют роль задач в пуле потоков, которые будут ожидать своей очереди в первом освободившемся Compute Unit. А все рабочие элементы внутри одной группы будут выполняться в concurrent режиме. Для начала расчётов требуется выполнить метод `clFlush`, а для ожидания выполнения всех задач - `clFinish`.

Рассмотрим содержимое .cl программы. В ней было решено использовать локальную память, которая общая в пределах рабочей группы. Это позволит переиспользовать память и уменьшить количество запросов в глобальную память. Можно сказать, что программа - это одна функция `multiply_matrices`, которая получает на вход три матрицы и их размер.

Каждый рабочий элемент в группе ответственен за загрузку своего участка локальной памяти (которая имеет форму квадрата). Далее производится операция перемножения в пределах этого блока, после чего загружается следующий блок. Остается добавить, что производимые вычисления в данной программе сравнивались с правильными результатами, рассчитанными с помощью прошлой 4 лабораторной работой.

Производительность

Стоит упомянуть, что вычисления производились на видеокарте с 8 Гб видеопамяти и 2176 универсальными ядрами, а также на процессоре с 6 ядрами и 12 потоками. Рабочая группа выставлялась в значение от 8x8 до 32x32. Для больших матриц нельзя было выставить малые рабочие группы, иначе бы не хватало compute units, в таком случае в таблице будет прочерк.

В качестве результата времени берется среднее арифметическое 10 испытаний. Результат приведён в таблице 1.

Таблица 1 – Замеры времени работы в зависимости от размера изображения и размера рабочей группы.

Матрица	Размер рабочей группы	Время с CPU, мкс		Время с GPU, мкс
		Стандартный алгоритм, 7 потоков	Алгоритм Штрассена	OpenCL
128x128	8x8	6781	8413	1421
	16x16			1623
	32x32			1654
256x256	8x8	36548	44657	4211
	16x16			4264
	32x32			4289
512x512	8x8	237967	278413	-
	16x16			18123
	32x32			18679
1024x1024	8x8	2046784	1874349	-
	16x16			-
	32x32			82416
2048x2048	8x8	16124891	12431895	-
	16x16			-
	32x32			419534

Как можно заметить из таблицы, GPU реализация работает намного более производительней, чем перемножение аналогичных матриц с помощью алгоритмов, запущенных на CPU. В то же время, для tiling-алгоритма нужно осуществлять подбор корректной рабочей группы на размер матриц, иначе во

время работы программы будет ошибка. Для алгоритмов на CPU таких ограничений нет.

Выводы

В результате выполнения лабораторной работы была разработана программа, производящая умножение матриц с использованием GPU на OpenCL. Сравнение данной программы с ранней реализацией алгоритмов на CPU показало, что GPU во много раз быстрее производит умножение.