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

ОТЧЕТ

по лабораторной работе № 5

по дисциплине «Параллельные алгоритмы»

Тема: Знакомство с программированием гетерогенных систем в стандарте OpenCL

Студент гр. 0304 _____ Шквиря Е.В.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург

2024

Цели работы

На практике освоить методы работы с OpenCL.

Постановка задачи

Реализовать расчёт фрактала Мандельброта на OpenCL. Визуализировать результат.

Выполнение работы

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

В качестве формата для файла-изображения использовался .ppm. Для этого был написан метод `save_image`, попиксельно сохраняющий изображение.

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 создания фрактала 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. В случае используемой программы (о которой будет сказано позже) будет нужно создать переменную

изображения. Для этого в OpenCL используется команда *clCreateImage*, которой на вход подаются контекст и различные конфигурации (в них различные метаданные изображения, в том числе его размер в пискелях). Одной из конфигураций здесь является флаг CL_MEM_WRITE_ONLY, который ограничивает изображение только на запись в рабочих элементах. Далее изображение требуется привязать к ранее созданному kernel с помощью команды *clSetKernelArg*.

Остается добавить kernel в очередь на исполнение. Так как нам нужно распараллелить вычисление фрактала, то следует задать индексное пространство и рабочие группы. Индексное пространство – это массив (1, 2 или 3 мерный), размер которого определяет общее количество рабочий элементов. Рабочая группа – это объединение нескольких рабочих элементов в той же размерности.

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. После выполнения программы нужно получить итоговое изображения. Для этого добавим еще одну задачу в очередь с помощью clEnqueueReadImage. Туда передаем очередь, переменную изображения и наш массив, в который надо записать результат. После этого очередь можно считать сформированной. Для начала расчётов требуется выполнить метод *clFlush*, а для ожидания выполнения всех задач - *clFinish*.

После этого полученное изображение сохраняется в файл .ppm формата.

Рассмотрим содержимое .cl программы. Можно сказать, что это одна функция *draw_mandelbrot*, которая получает на вход переменную out, доступную на запись. С помощью команд get_global_id(0) и get_global_id(1) можно получить индекс текущего элемента в глобальном индексном пространстве. В данном случае – это координаты в пикселях, которые еще

надо преобразовать в относительные координаты. Для преобразования нужно знать общие размеры изображения. Их можно получить через get_image_dim(out). Далее идет расчет цвета пикселя. Подробности расчета здесь не важны, важно лишь то, что проводятся вычисления в теоретически бесконечном цикле. Если в какой-то момент до достижения предельного значения итераций выполняются достаточные условия для завершения расчетов, то мы выбираем цвет такого пикселя по количеству пройденных итераций. Иначе цвет черный. Выбранный цвет теперь записывается в изображение с помощью команды write_imageui, в которой указываются положение пикселя и его цвет.

Результат вычисления:

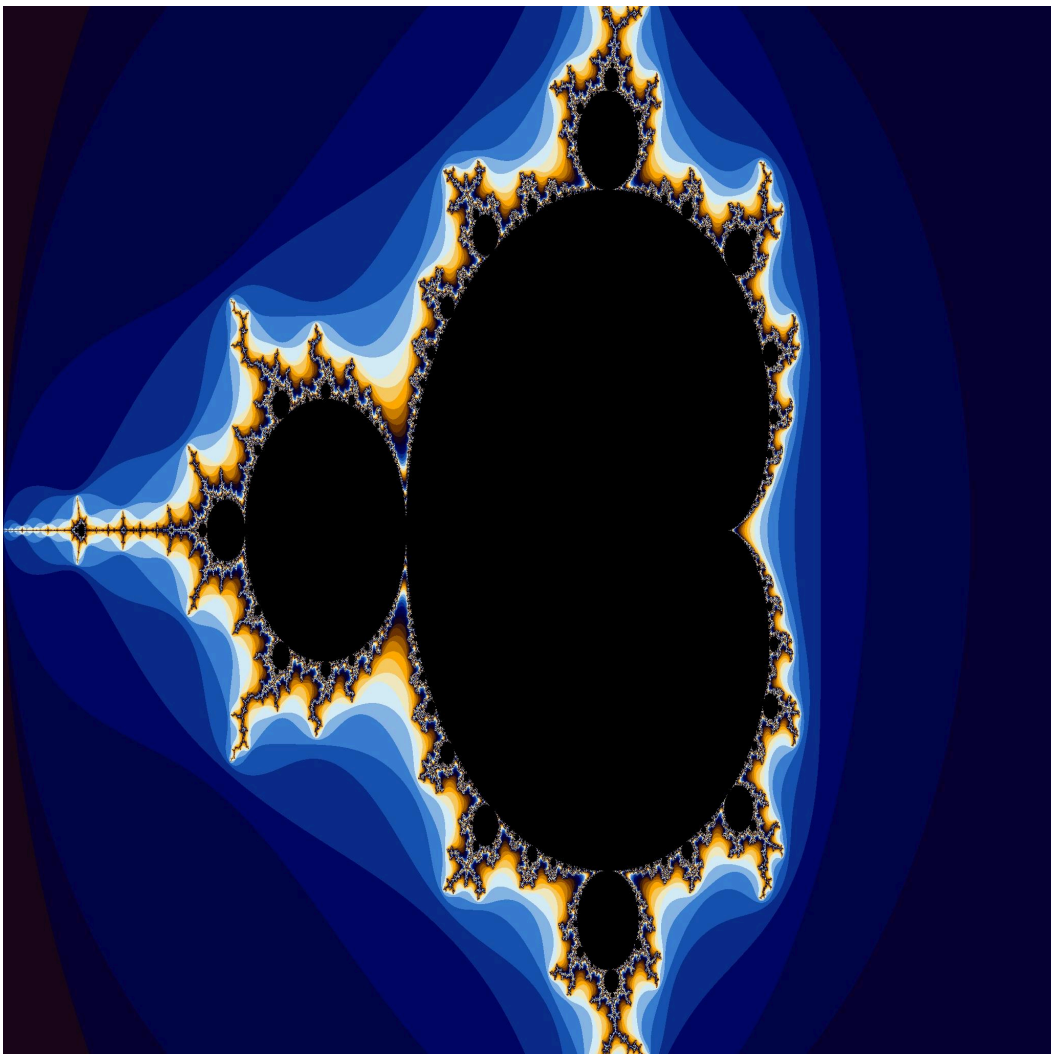

Рисунок 1 — Вычисленный с помощью OpenCL фрактал Мендельброта

Производительность

Проведём замеры, в которых будем рассматривать квадратные изображения с квадратными рабочими группами. Для каждой пары измерим время в миллисекундах. По горизонтали - размер изображения, по вертикали - размер рабочей группы. Результат приведён в таблице 1.

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

	1x1	4x4	8x8	16x16
256x256	825	189	206	208
512x512	2482	914	722	806
1024x1024	9347	2924	2754	2568
2048x2048	36659	11409	10538	10446

Из таблицы 1 видно, что чем больше размер изображения, тем больше времени требуется на вычисления. В то же время если для достаточно большого разрешения уменьшить размер рабочей группы (тем самым увеличив количество этих самых групп), то время исполнения также будет расти вследствие переполнения пула исполнения. Однако слишком малый размер work-группы может привести к слишком большому падению производительности.

Множественные замеры показали, что при увеличении разрешения изображения время выполнения стремительно увеличивается. Можно выделить 2 источника увеличения:

- Часть подготовки OpenCL, различных данных и массивов (все задачи до вызова clEnqueueNDRangeKernel)
- Часть исполнения (clEnqueueNDRangeKernel + clFlush + clFinish)

Самым большим источником увеличения времени будет часть исполнения. В ней берутся три команды, где первая ставит вычисление в очередь, вторая делает flush отправляя задачу на вычисление и третья, которая ожидает окончания исполнения.

Выводы

В данной работы были разобраны базовые принципы работы с OpenCL на примере разработки программы по вычислению фрактала Мандельброта. Было изучено, как работать с адресными пространствами, рабочими группами и элементами. Был изучен C-like язык разработки программ для рабочих элементов.