UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Practica 1: Ejecucion de codigo bajo OpenCL

Rodríguez Martinez, David

OpenCL

Índice general

Capítulo 1

Introducción a la práctica

Esta práctica de Lenguajes y entornos de programación paralela consiste en la explotación de la GPU con el fin de poder obtener unos resultados de ejecución de problemas pesados para poder ser comparados entre diversos dispositivos y modelos de ejecución a fin de saber cuánta mejora ofrece este sistema de procesamiento.

El cálculo acelerado es el uso de una unidad de procesamiento gráfico en combinación con una CPU para acelerar aplicaciones de empresa, ingeniería, análisis y cálculo científico.
Gracias a esto las GPU aceleradoras han pasado a instalarse en centros de datos energéticamente eficientes de laboratorios gubernamentales, universidades y grandes compañías de todo el mundo. Las GPUs aceleran las aplicaciones de plataformas diversas, desde automóviles hasta teléfonos móviles y tablets, drones y robots.

Para ello he realizado la práctica bajo *OpenCL*, una serie de librerías aportadas por *CUDA GPU Programming*.

Con esto implementaremos un código en C que incorpora este modelo de programacion paralela que se ejecutara numerosas veces mientras se le aumenta la talla del problema, además se compararán diversas tecnologías, como la explotación de un solo núcleo (programación secuencial), *CUDA* y *OpenCL* (programación paralela) a fin de obtener una serie de resultados en los que se aplicaran diferentes comparaciones y conclusiones.

Además OpenCL es una especificación desarrollada por Apple, asi que se realizara una prueba en un entorno que incorpora nativamente OpenCL (OS X) con el fin de ver cómo de sencillo resulta trabajar con esta especificación en un entorno preparado.

Capítulo 2

¿Qué es *OpenCL*?

2.1 ¿Qué es y cómo funciona?

Open Computing Language (OpenCL) es un framework para diseñar programas que se ejecutan sobre plataformas heterogéneas que constan de CPUs, GPUs, Procesadores de Señal Digital (DSP o PSD en castellano), FPGAs y otros procesadores y aceleradores. OpenCL especifica un lenguaje para programar dispositivos e interfaces de programación de interfaces (API) para controlar la plataforma y ejecutar programas en los dispositivos de computación.

Se trata de un lenguaje sobre C, Las funciones ejecutadas sobre un dispositivo OpenCL se les llama núcleo. Una unidad de cálculo puede ser denominada como núcleo, pero el termino de núcleo es difícil de definir sobre todos los tipos de dispositivos compatibles con OpenCL. Un solo dispositivo de cálculo normalmente consta de varias unidades de procesamiento, que a su vez comprenden múltiples elementos de proceso (PES). Una sola ejecución del kernel puede ejecutarse en todas o muchas de las unidades de computación en paralelo aunque y el número de unidades de computación puede no corresponder al número de núcleos.

Además de un lenguaje de programación C, OpenCL define una interfaz de programación de aplicaciones (API) que permite a los programas bajo OpenCL que se ejecutan en el host para iniciar núcleos en los dispositivos de computación y administrar la memoria de este dispositivo, que relativamente está separada de la memoria del host. Esta API está especificada en lenguaje C aunque hay librerías de terceros para aplicarlas a otros lenguajes.

Los programas en el lenguaje OpenCL están destinados a ser compilado en tiempo de ejecución, por lo que las aplicaciones OpenCL son portables entre las implementaciones para diferentes dispositivos de computación (GPU,DSP,FPGAs,...)

En la figura 2.1 se puede apreciar el funcionamiento de OpenCL y su gestión del dispositivo de computación. La peculiaridad de estos es su manera de gestionar la memoria ya que cada núcleo la gestiona de manera individual por lo que crea una jerarquía de memoria diferente lo cual implica un modelo de programación diferente.

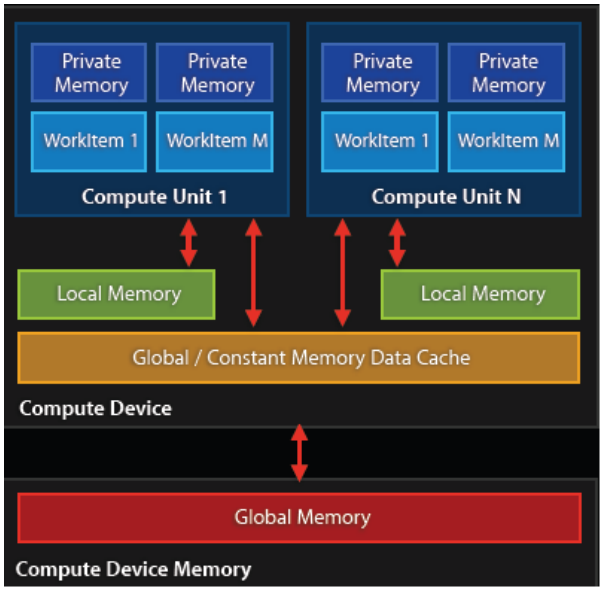

Figura 2.1: Diagrama de OpenCL

2.2 Jerarquia de Memoria

OpenCL define una jerarquía de memoria de cuatro niveles para el dispositivo de computación (una tarjeta gráfica, por ejemplo):

- Memoria global: compartido por todos los elementos de procesamiento, pero tiene un tiempo de acceso alto.
- Memoria Read Only: un tiempo de acceso menor que la global, sin embargo solo puede comunicarse con la CPU y no con los Núcleos del dispositivo de computación.
- Memoria local: compartido por un grupo de Núcleos.
- Memoria privada: por cada núcleo tiene sus registros de memoria.

No todos los dispositivos necesitan implementar cada nivel de esta jerarquía en el hardware. Está ajustado para cumplir reglas de sincronización explícitas, en este caso barreras.

Capítulo 3

Presentacion del Problema

En la práctica se van a presentar los siguientes problemas:

- Instalacion del entorno.
- Multiplicacion de matrices (Linux)
- Procedimiento de desplazamiento Geometrico (OS X)

Estos ejemplos nos permitiran visualizar como funciona en diferentes sistemas operativos (aunque similar kernel) la especificacion OpenCL para ello explicare en que consiste cada uno de los problemas listados anteriormente ademas de explicar como he realizado la instalacion en los diferentes sistemas (Linux y OSx)

3.1 Instalacion del entorno

La instalacion para MacOSx ha sido relativamente sencilla pues las librerias vienen implementadas en el programa implementado por apple Xcode asi que una vez instalado desde la Appstore ya esta listo para su funcionamietno.

Sin embargo cuando nos vamos a otro sistema operativo empiezan los problemas, este ha sido el caso de linux, que aun siendo muy similar a MacOSx, la instalacion en este puede dar problemas por la falta de librerias.

Por suerte Nvidia proporciona una serie de archivos ejecutables (.run) que ayudan a la instalacion de este entorno ademas de incorportar ejemplos

3.2 Multiplicacion de matrices

Aqui presentare el problema pesado para su ejecucion bajo un dispositivo de computacion (en mi caso una tarjeta grafica) este problema aumentara su talla y los resultados se compararan con una ejecucion de secuencial a fin de evaluar estos resultados.

El problema consiste en:

$$A \cdot B = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & a_{i3} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1j} & \cdots & b_{1q} \\ b_{21} & b_{22} & \cdots & b_{2j} & \cdots & b_{2q} \\ b_{31} & b_{32} & \cdots & b_{3j} & \cdots & b_{3q} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nj} & \cdots & b_{nq} \end{pmatrix} =$$

$$= \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1j} & \cdots & p_{1q} \\ p_{21} & p_{22} & \cdots & p_{2j} & \cdots & p_{2q} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{i1} & p_{i2} & \cdots & p_{ij} & \cdots & p_{iq} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mj} & \cdots & p_{mq} \end{pmatrix}$$

Figura 3.1: Explicacion de como se reliza una multiplicacion de matrices

Dos matrices A y B se dicen multiplicables si el número de columnas de A coincide con el número de filas de B.

El elemento C_{ij} de la matriz producto se obtiene multiplicando cada elemento de la fila i de la matriz A por cada elemento de la columna j de la matriz B y sumándolos.

En nuestro codigo tenemos que una matriz A y una matriz B se multiplican para dar a una matriz C, para cada una hay uqe especificar la memoria que consumira, ya que es uno dee los problemas en la jerarquia de memoria de OpenCL.

3.3 Procedimiento de desplazamiento Geometrico

En esta parte de la practica he incorporado un ejemplo propuesto por Apple que a mi opinion me ha parecido curioso que consiste en una aplicacion de OpenCL y OpenGL que cuya implementacion comparte memoria de ambas tecnologias.

En particular, OpenCL y OpenGL pueden compartir datos, lo que reduce los gastos generales. Por ejemplo, los objetos de OpenGL y los objetos OpenCL creados a partir de objetos de OpenGL pueden acceder a la misma memoria. Además, GLSL (OpenGL Shading Language) shaders y almendras de OpenCL pueden acceder a los datos compartidos.

Para asegurarse de OpenCL y OpenGL hay que estabalecer ciertos parametros:

- Establezcer el programa para hacer todo su cálculo y la representación en la GPU.
- Asignar memoria para asegurarse de que los datos se comparten de manera eficiente.

Esto mejorará el rendimiento, ya que evitará tener que transferir datos entre el host y la GPU.

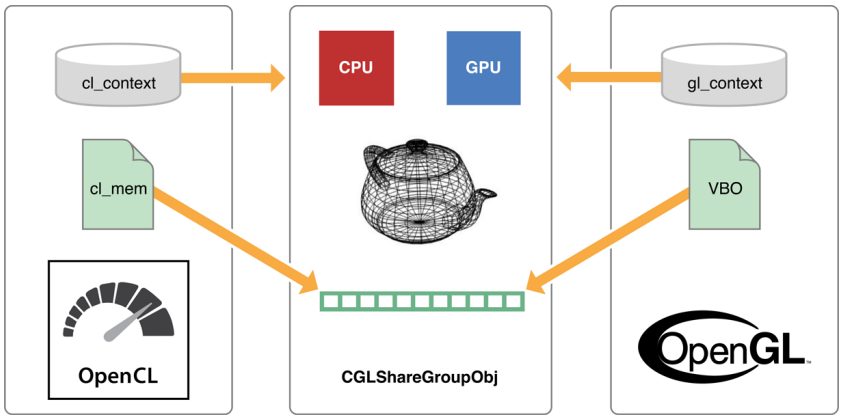

Figura 3.2

En la figura 3.2 se puede apreciar las dos partes del problema y la solucion compartida, OpenGL genera una imagen y este realiza la carga, mientras que OpenCL se encarga de la gestion de la CPU y la memoria, todo esto nos ofrece una solucion en lo cual se comparte un buffer en el que ambas partes se interconectan.

Capítulo 4

Ejecucion y resultados

En este capitulo realizare las pruebas y realizare comparativas con el fin de poder calcular que mejora puede ofrecer OpenCL respecto a la programacion secuencial ya que la programacion paralela siendo mas compleja da muchas ventajas en cuanto a rendimiento y optimizacion.

4.1 Multiplicacion de matrices

En primer lugar procedere a la ejecucion de la Multiplicacion de matrices proporcionada por una serie de ejemplos de Nvidia para la computacion.

Tratara de una serie de ejecuciones en los que la talla del problema ira en aumento hasta que el programa tenga problemas en la ejecucion en terminos de memoria.

Para ello he realizado un script que realizara la ejecucion del codigo secuencialmente de cada problema y como resultado nos dara el tiempo que ha tardado cada problema en resolverse, esto nos dara una grafica de valores en los que se podra obtener el rendimiento de mejora (*speedup*). Ademas he realizado pruebas en diferentes graficas de entornos muy diferentes de procesamiento como por ejemplo una grafica de un portatil el cual esta diseñado para el ahorro de energia y una grafica potente para *Gamming* la cual se le ha realizado *OverClock* y esta diseñada para consumir energia y poder conseguir un aumento de rendimiento.

Para este problema, el cual solo realiza los calculos y no nos devuelve la matriz que alargaria los calculos por culpa del buffer de salida, he implementado un multiplicador de programa que aumentara su talla exponencialmente haciendo una ejecucion muy costosa en valores elevados de la tabla, si OpenCL ofrece una potencia de paralelismo elevada los resultados deberian ser muy notables respecto a la programacion sequencial.

Capítulo 5

Pruebas y resultados

Capítulo 6

Conclusiones

Bibliografía

[1] Llinares R. (2015) *Documentación de Configuración, administración y gestión de redes*,
Departamento de Comunicaciones,
Universitat Politècnica de València.

[2] http://wiki.mikrotik.com/wiki/ *MikroTik Documentation*,
MikroTik maintained documentation pages,
Mikrotikls Ltd.