OpenCL

Rubén Akagi
maestrokame@gmail.com

Universidad Católica - Nuestra Señora de la Asunción

Abstract. Una breve introducción al OpenCL framework. Analiza la estructura interna de la plataforma, las memorias, los procesos. El lenguaje OpenCL junto con algunos aspectos interesantes. Los ambientes de ejecución y las herramientas de desarrollo. También compara las competencias actuales en el área de GPGPU con sus ventajas y desventajas.

Key words: OpenCL, GPGPU

1 Introducción

En los últimos tiempos, las unidades de procesamiento gráfico han evolucionado de forma sorprendente, ya sea por su capacidad de procesamiento o las variadas funcionalidades que ofrecen al programador con sus sets de instrucciones, así como el aumento en la presición numérica. Este avance ha encaminado necesariamente a la idea de aprovechar su potencial computacional para realizar cálculos no necesariamente gráficos. Pero los APIs tradicionales para manejar los gráficos no eran aptos para programar computación de propósito general.

1.1 GPGPU

Así surge la técnica llamada GPGPU, o *General-Purpose computation on Graphics Processing Units* por su sigla en en inglés. Consiste en utilizar la capacidad de las Unidades de Procesamiento Gráfico para asignar tareas que normalmente son procesadas por la CPU. [2]

Las grandes empresas como Nvidia, AMD, ATI, y últimamente Microsoft, han echo el esfuerzo de explotar esta capacidad desaprovechada lanzando al mercado sus propias versiones que se analizaran a continuación.

1.2 CUDA

Sigla en inglés de Compute Unified Device Architecture, fué desarrollado por Nvidia. Utiliza un lenguaje parecido a C llamado *C for Cuda*. Tiene soporte total para punteros, funcionalidades de C++ como el templating, muchas librerías que tienen aceleración por excelencia sobre GPU, buenas herramientas de desarrollo, bien documentadas llenos de ejemplos disponiles. Son signos de la madurez que ha alcanzado estando un tiempo considerable en el mercado, aunque tiene la gran desventaja de que se ejecuta solamente sobre GPU de Nvidia.

1.3 DirectCompute

Es la respuesta de Microsoft frente al surgimiento de las GPGPU. Están incluidas en los últimos dos DirectX, la10 y la 11. Mantiene la sintaxis del lenguaje HLSL (High Level Shading Language), así que los programadores de DirectX pueden aprender con mayor facilidad. Lo malo es que DirectX está disponible solamente para windows 7 y Vista. También es muy notable la falta de ejemplos y documentaciones, así como una muy pequeña cantidad de bindings disponibles.

2 ¿Qué es OpenCL?

Los sistemas actuales tienen una amplia variedad de dispositivos, desde los tradicionales CPU hasta los GPU, procesadores Cell, DSP, FPGA y otros. Cada uno tienen diferentes capacidades de procesamientos y niveles de paralelismo. Desarrollar softwares eficientes capaces de aprovechar estos recursos siempre ha sido un reto para los programadores. [4]

OpenCL, un framework multiplataforma libre y abierto mantenido por Khronos Group un consorcio sin fines de lucro, ha tratado de resolver este problema proporcionando una interfaz estándar de programación.

Las unidades de procesamiento actuales vienen en varios núcleos, ya que la velocidad del clock individual ha sentido el límite teórico. En especial las GPUs, que están diseñadas para realizar calculos vectoriales de espacios dimensionales, tienen capacidades de procesamiento paralelo enorme para manejar estas operaciones de forma efectiva.

OpenCL aprovecha de estas capacidades de procesamiento en paralelo que ofrecen cada dispositivo, con un lenguaje de programación propia basado en C99 que permite realizar paralelismo de taeas o de datos, aumentando drásticamente la capacidad total de procesamiento, especialmente de las aplicaciones computacionalmente intensas. [11]

3 ¿Cómo funciona el OpenCL?

3.1 Modelo de pataforma

El ambiente de ejecución o la platafora sobre la cual corren los programas en OpenCL, como vemos en la figura 1, consta de varios dispositivos. Cada dispositivo, por ejemplo las CPUs o las GPUs, pueden tener varias unidades de cómputo o núcleos. [11][3]

Cada una de las unidades de pocesamiento pueden tener varias unidades de cómputo que son ejecutadas como SIMD o SPMD.

El host, normalmente la CPU, es el que administra el comportameinto global de la plataforma.

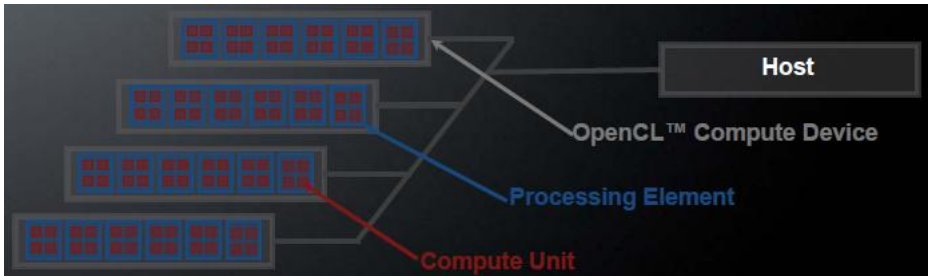

Fig. 1. Modelo de plataforma.

3.2 Modelo de ejecución

Los programas en OpenCL están formados por kernels, que son unidades básicas de código paralelamente ejecutable.

Cuando se ejecuta un programa, el host va instanciando los kernels necesarios y agregada en las colas de espera de distintos dispositivos. [11]

3.3 Los work-items y los workgroups

En OpenCL se puede definir un dominio computacional de 1, 2 o 3 dimensiones. Cada elemento del dominio se llama work-item, y representa a una unidad de procesamiento capaz de ejecutar un kernel a la vez. Dicho de otra forma, las dimensiones del dominio define la cantidad de work-tems que se va a ejecutar en paralelo.[11][3]

Por ejemplo, sea un programa que necesite sumar dos vectores a y b de tamaño n. La forma tradicional de programación escalar sería:

```
void
scalar_sum(int n,
           const float *a,
           const float *b,
           float *result){
    int i;
    for (i=0; i<n; i++) result[i] = a[i] + b[i];
}
```

Pero utilizando OpenCL, se define un dominio de una sola dimensión de longitud n, que contiene n work-items. Luego se ejecuta el siguiente kernel en cada uno de los work-items en forma paralela.

```
kernel void
dp_sum(globalconst float *a,
       globalconst float *b,
       globalfloat *result){
    int id = get_global_id(0);
```

```
    result[id] = a[id] + b[id];
}
```

Los work-items se pueden agrupar para formar un workgroup. Todos los work-items que conforman un workgroup se ejecutan en un mismo dispositivo. Así pueden compartir la memoria local o realizar sincronizaciones entre los work-items. (o entre los kernels que se ejecutan en cada work-item)[4]

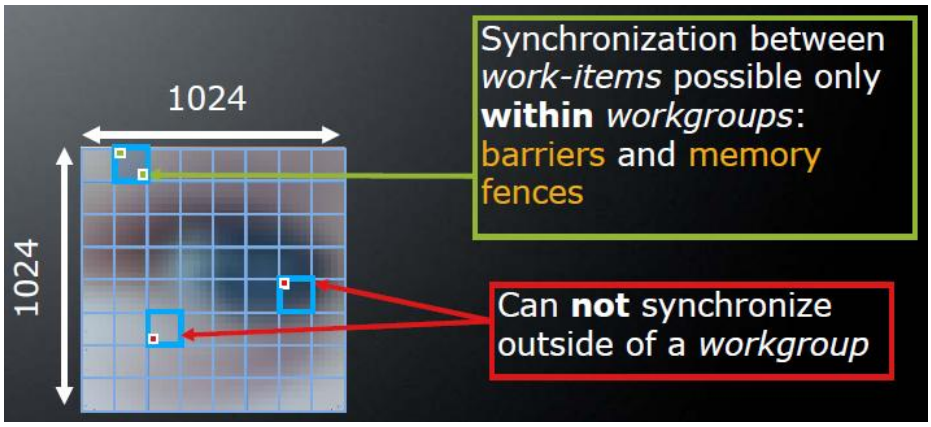

Fig. 2. Los work-items y los workgroups.

La figura 2 muestra en un espacio global computacional de dos dimensiones de 1024x1024 que contiene a 64 workspaces también de dos dimensiones de 128x128, o sea 64x128x128 work-items en total. Los work-items rojos no pueden sincronizar entre sí por estar en workgroups distintos pero sí los work-items verdes.[11]

3.4 Modelo de memoria

Como vemos en la figura 3, las memorias dentro de la plataforma están organizadas en forma jerárquica. Esta jerarquía es una abstracción que ofrece el framework, y no necesariamente está físicamente organizado de esta forma.[4]

La memoria privada es exlusivo para cada work-item y sirve para almacenar informaciones necesarias para ejecutar un kernel.

La memoria local es compartida por los work-items dentro del workgroup así como la memoria global es compartida entre los workgroups. [11]

3.5 Uniendo todos

La ejecución del programa se inicia con la compilación de los kernels que los componen. Se compilan de diferente manera para cada dispositivo, ya que tienen

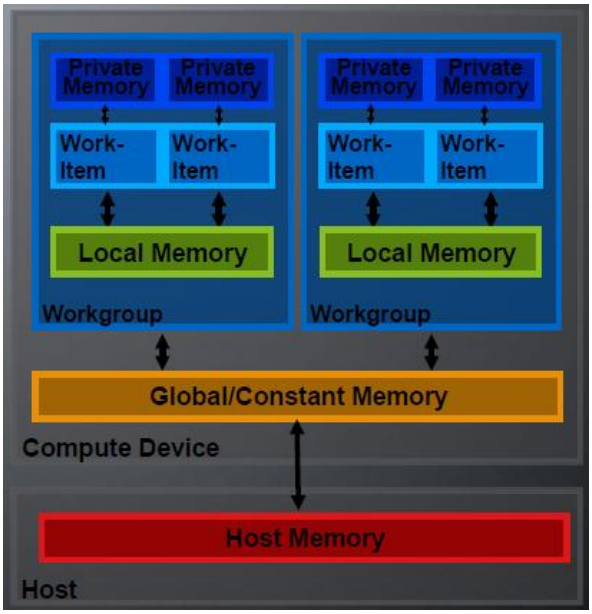

Fig. 3. Modelo de memoria.

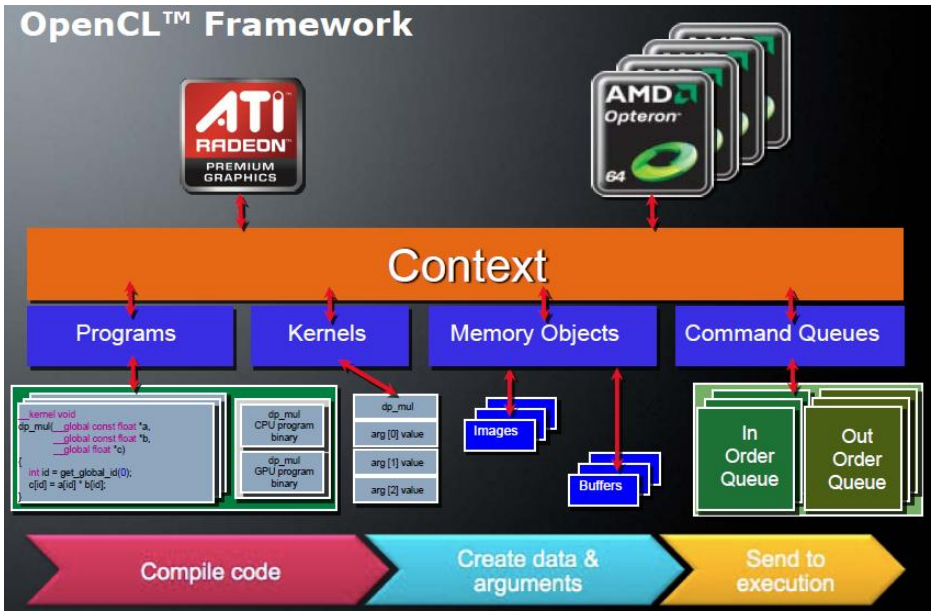

Fig. 4. Uniendo todos.

diferentes sets de instrucciones, pudiendo especificar cuál kernel ejecutar sobre cual dispositivo. Cuando se terima la compilación, OpenCL realiza el chequeo de la compilación para ver si no hubo algún error en el proceso.

Después viene la fase de creación de las estructuras de memorias que se van a consumir, y luego la asignación de los parametros a las instancias de los kernels.

Finalmente, cuando los kernels están listos para ser ejecutados, son enviados a las colas de espera de los dispositivos para su posterior ejecución. Es posible especificar la manera en que se va a atender las colas de los dispositivos. Los In-Order son como los CPU clásicos en donde otras istrucciones esperan hasta terminar por completo una instrucción. Los Out-Order en cambio, son cuando las instrucciones no se ejecutan en realidad en el orden real, por ejemplo para realizar pipeline cuando una instrucción tarda más de un clock.

4 CPU vs GPGPU

Con la generalización de las GPU, se han creado una rivalidad hasta ahora inexistente entre las empresas de CPU y las de GPGPU. Cada una realiza y publica métricas con resultados diferentes.

Pero dependiendo de los problemas a resolver, se han observado en una cantidad de universidades e institutos de renombre, resultados realmente asombroso a favor de GPGPU como se puede observar en la figura 5.

5 ¿Por qué utilizar OpenCL?

OpenCL, como los otros frameworks de GPGPU, permite aprovechar el enorme poder computacional paralelo, mejorando drásticamente el rendimiento de los programas computacionalmente intensas.

Pero a diferencia del resto, es el primero en ser abierto y gratuito. Permite escribir códigos que pueden correr en multiples plataformas despreocupandonos de muchos aspectos del hardware. [9] La compilación se realiza en tiempo de ejecución mediante el API, permitiendo la oportunidad de optimizar para ciertos tipos de dispositivos. [4]

6 Desventajas de OpenCL

OpenCL tiene dos grandes desventajas.

Primero, como trata de crear una interfaz uniforme para distintos dispositivos con sus respectivas ventajas y desventajas, inevitablemente es simpre menos eficiente que los frameworks diseñados específicamente para ciertos dispositivos.

Segundo, es un framework joven. No tiene la maduración suficiente comparado con otros frameworks, que han estado ya un buen tiempo en el mercado. Necesita acumular experiencias, funtes de información y una buena cantidad de ejemplos documentados, desarrollar buenas herramientas de desarrollo, y como todos los frameworks, una buena calidad y cantidad de librerías.

Developer	Speed Up	Reference
Massachusetts General Hospital	300x	http://www.opticsinfobase.org/oe/abstract.cfm?uri=oe-17-22-20178
University of Rochester	160x	http://cyberaide.googlecode.com/svn/trunk/papers/08-cuda-biostat/vonLaszewski-08-cuda-biostat.pdf
University of Amsterdam	150x	http://arxiv.org/PS_cache/arxiv/pdf/0709/0709.3225v1.pdf
Harvard University	130x	http://www.springerlink.com/content/u1704254764133t5/?p=c5eead9af73340e58a313d95581cfd40&pi=49
University of Pennsylvania	130x	http://ic.ese.upenn.edu/abstracts/spice_fpl2009.html
Nanyang Tech, Singapore	130x	http://www.opticsinfobase.org/abstract.cfm?URI=oe-17-25-23147
University of Illinois	125x	http://www.nvidia.com/object/cuda_apps_flash_new.html#state=detailsOpen;aid=c24dcc0f-c60c-45f9-8d57-588e9460a58f
Boise State	100x	http://coen.boisestate.edu/senocak/files/BSU_CUDA_Res_v5.pdf
Florida Atlantic University	100x	http://portal.acm.org/citation.cfm?id=1730836.1730839&coll=GUIDE&dl=ACM&CFID=88441459&CFTOKEN=90295264
Cambridge University	100x	http://www.wbic.cam.ac.uk/~rea1/research/AIRWC.pdf

Fig. 5. GPGPU vs CPU.

7 OpenCL™ C Language

7.1 Características

Es un lenguaje que trae consigo el framework. Está basado en el lenguaje C99 con ligeras modificaciones, con ciertas restricciones y agregando nuevos elementos.

Se han quitado las cabeceras de las librarías estandar, puntero a funciones, recursiones, arreglos de longitud variable, y los campos de bits.

Agregaron, para soportar la programación paralela, los work-items y los work-groups, sincronización, y vectores como nuevos tipos de variables. Viene además con una variedad de funciones incorporadas en su API.

Veremos a continuación algunas características interesantes del lenguaje.

7.2 Vectores

Aparte de los tipos de variables escalares tradicionales y otras específicas de la GPU (como image2d_t, image3d_t, sampler_t), OpenCL incorpora el tipo vector con sus variantes que trae con sigo un interesante y variado grupo de operadores.

Para determinar el tipo del vector primero se toma el tipo escalar del elemento, seguida de la longitud que es un número entre 2, 4, 8 y 16. Así, int8 es un vector de 8 enteros. Lo interesante de los vectores en OpenCL es que se puede obtener y asignar de forma variada a los elementos de manera sencilla.

Por ejemplo, la siguiente línea asigna -7 a todos los elementos del vector.

```
int4 v = (int4) -7;
```

Esta asigna directamente los valores a los elementos del vector.

```
int4 v = (int4) (0, 1, 2, 3);
```

También permite acceder a los low elements o hi elements, que retorna en cada caso, un vector de la mitad del tamaño original.

```
int4 v = (int4) (0, 1, 2, 3);
```

Es posible acceder elemento por elemento en este caso el elemento de la posición 5,

```
v.s5
```

un conjunto arbitrario de elementos, sea los elementos en las posiciones 0, 1, 4 y 7,

```
v.s0147
```

los elementos de posiciones pares o impares,

```
v.odd
```

o las composiciones o uniones de varios vectores, en este caso los elementos de las posiciones 0, 1, 4 y 7 del vector 1, seguido de los elementos de las posiciones pares del vector v2.

```
(v1.s0147, v2.odd)
```

Los vectores vienen también con operaciones especializadas muy eficientes gracias a la capacidad de procesamiento vectorial de las GPU. Abajo mestra un ejemplo de la suma y el valor absolute de un vector. [11] [8]

```
vi0 += vi1;
vi0 = abs(vi0);
```

7.3 Tipos de conversiones

Las conversiones de los escalares y de los punteros obedecen a las reglas del estándar C99. Pero los vectores no pueden realizar conversiones de tipos.

Los casts o las conversiones de tipos siempre han tenido problemas en cuanto a las impresiciones de los decimales, o la regla de conversión entre un entero y un decimal. OpenCL ofrece una forma más específica de hacer las conversiones. Necesita especificar además de los tipos, la saturación o redondeo hacia arriba, y el modo de redondeo.

Aparte de los casts, están los reinterpret, que es tomar como válido un tipo de variable aunque en realidad es otro tipo de variable. Esto sí se puede utilizar con los vectores para, digamos, convertir los tipos de variables. [11] [8]

7.4 Espacio de direcciones

Las variables a declarar en OpenCL pueden tener prefijos que especifican cual espacio de direcciones tomar. Puede ver la figura 3 para recordar la jerarquía y el alcance de cada uno. [8]

- __global
- __constant: es como el global pero de solo lectura
- __local
- __private
- __read_only: solamente para imagenes
- __write_only: solamente para imagenes

7.5 Funciones de Sincronización

Al programar aplicaciones o algoritmos que se va a ejecutar en paralelo, surge la necesidad de realizar sincronizaciónes. OpenCL trae consigo varias formas de sincronización.

```
barrier()
```

Es un mtodo de sincronización tradicional en donde todos los work-items del mismo workgroup que lo implementan, al alcanzar esta función esperan a que lleguen los demás work-items para luego continuar con la ejecución. Es para sincronizar el flujo de control del programa. [10]

`mem_fence(), read_mem_fence(), write_mem_fence()`

Controla la carga o almacenamiento en memoria dentro del work-item. Es un método de sincronizacion de acceso a datos. [13]

8 Ambiente y herramientas de desarrollo

Para empezar a programar en OpenCL se necesitan instalar las siguientes herramientas: [1]

- OpenCL Compiler
- OpenCL Runtime Library
- OpenCL drivers (para cada device)

El programa principal (main) se puede programar en C, C++, JAVA, Python, y más lenguajes están en desarrollo de sus bindings al OpenCL.

Se puede programar en Windows, Linux, o en Mac OS X. OS X es su última versión ya viene completamente integrada, y el próximo Ubuntu 10.10 está previsto incorporar de forma nativa a OpenCL.

Existen varios IDEs con depuradores para OpenCL incorporados como el OpenCL Studio que necesita comprar la versión profesional para liberar todas las funcionalidades. También está el OpenCL SDK 1.5 de Intel que viene para Windows Vista, Windows 7 y Linux. Otra opción es utilizar un plugin para Visual Sudio, y (para los usuarios de Visual Sudio) programar en un ambiente bien familiar.

9 Conclusión

Es una tecnología que no surgió de golpe, sino fue cobrando fuerza de a poco, comenzando con incrementar las capacidades de procesamiento. Luego fué incorporando lenguajes de programación de propósito específico para los shaders que fueron los primeros pasos para programar sobre un ambiente GPU.

La gran ventaja está por supuesto en lo libre y abierto el la multiplataforma, pero m'as en el aprovechamiento de la capacidad de procesamiento imnineteme nte paralelo de los GPU.

OpenCL tiene un futuro comprometedor, ya que participan en el proceso todas las empresas líder en una variedad de ambiente informático. Está creciendo de forma sorprendente, ya que desde su propuesta hasta la implementación de la versión 1.0 ha tardado solo 6 meses.

References

1. The opencl programming book.
2. General-purpose computation on graphics hardware. 2009.
3. Opencl, introduction and overview. pages 1–20. Khronos Group, 2010.
4. *Heterogeneous Computing with OpenCL*. Morgan Kaufmann, 2011.
5. AMD. Breve historia de la gpu para propsito general (gpgpu).
6. AMD Architect Benedict R. Gaster. Introductory tutorial to opencl.
7. Ph.D. David W. Gohara. Episode 2 - opencl fundamentals. pages 1–29, Center for Computational Biology, 2009. MacResearch.
8. Khronos Group. Opencl 1.0 reference pages.
9. Andre Heidekruger. What is opencl?, stream computing workshop. pages 1–59, Stockholm, KTH, 2009. AMD.
10. Paul E. McKenney. Memory barriers: a hardware view for software hackers. 2010.
11. Aaftab Munshi. Opencl parallel computing on the gpu and cpu. pages 1–38. SIGGRAPH, 2008.
12. Aaftab Munshi. The opencl specification. pages 1–308. Khronos OpenCL Working Group, 2009.
13. Anshul Gupta George Karpis Vipin Kumar, Ananth Grama. *Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms*. Benjamin-Cummings Pub Co, 1994.