Tema 2. OpenCL

Grado en Ingeniería Informática

Ingeniería de Computadores

José María Granado Criado

Índice

- Introducción.
- Arquitectura OpenCL.
- Programando con OpenCL.

Introducción

- OpenCL.
  - Open Computing Language es una API de computación paralela diseñada para hacer que GPUs y coprocesadores trabajen en colaboración con la CPU.
    - En otras palabras, es una API para programación heterogénea.
  - Aparece como estándar OpenCL 1.0 el 8 de diciembre de 2008 por el grupo Khronos, un consorcio de estándares independiente.
  - https://www.khronos.org/opencl/
  - La principal ventaja es que es un lenguaje abierto y multiplataforma.
    - Da la posibilidad al programador de no tener que usar lenguajes propietarios que limitan su uso a un hardware concreto, como NVIDIA CUDA, que está limitado a las GPUs NVIDIA.

Introducción

- OpenCL.
  - Implementadores de OpenCL
    - QUALCOMM
    - Texas Instruments
    - Intel
    - AMD
    - Altera Corporation
    - Vivante Corporation
    - Xilinx, Inc.
    - MediaTek Inc
    - ARM Limited
    - Imagination Technologies
    - Apple, Inc.
    - STMicroelectronics International NV
    - ARM
    - IBM Corporation
    - Creative Labs
    - NVIDIA
    - Samsung Electronics
  - Productos que permiten OpenCL
    - https://www.khronos.org/conformance/adopters/conformant-products/opencl

Introducción

- Computación heterogénea.
  - En el pasado, las aplicaciones podían escalar con las ventajas proporcionadas por los avances en la tecnología de CPUs.
  - En los ordenadores modernos, las aplicaciones requieren interacciones con varios sistemas (audio/video, redes, etc.), por lo que los avances en la tecnología de CPUs no es suficiente.
  - Para conseguir grandes ganancias de rendimiento, se necesita hardware especializado, convirtiéndolo en un sistema heterogéneo.
    - Esto permite al programador utilizar el hardware que más se adapte a las necesidades en cada instante.
  - Computación heterogénea es aquella que envuelve varios tipos de unidades computacionales, como CPUs, GPUs, aceleradores hardware (FPGAs, Xeon Phi), DSPs, …

Introducción

- Marco de trabajo de OpenCL.
  - El marco de trabajo de OpenCL se especifica en tres partes principales: La especificación del lenguaje, la API de plataforma y la API de ejecución.
  - Especificación del lenguaje.
    - Describe la sintaxis y el interface de programación para escribir los programas o kernels que se ejecutarán en los dispositivos.
      - Los kernels pueden ser precompilados o dejar a OpenCL que los compile en tiempo de ejecución.
    - El lenguaje OpenCL está basado en la especificación ISO C99, donde se especifican las extensiones y restricciones.
      - Extensiones: tipos y operaciones vectoriales, acceso optimizado a imágenes y calificadores de espacios de direcciones.
      - Restricciones: punteros a funciones, bit-fields y recursividad.

Introducción

- Marco de trabajo de OpenCL.
  - API de plataforma.
    - Dan al programador la posibilidad de preguntar al sistema por la existencia de dispositivos OpenCL.
    - También permiten usar los conceptos de contexto de dispositivo y colas de trabajo para seleccionar e inicializar dispositivos OpenCL, enviar trabajos a esos dispositivos y habilitar la transferencia de datos entre el host y los dispositivos.
  - API de ejecución.
    - La API de ejecución usa los contextos para manejar objetos como colas de comandos, objetos memoria y objetos kernels así como para ejecutar los kernels en uno o más de los dispositivos especificados en el contexto.

Introducción

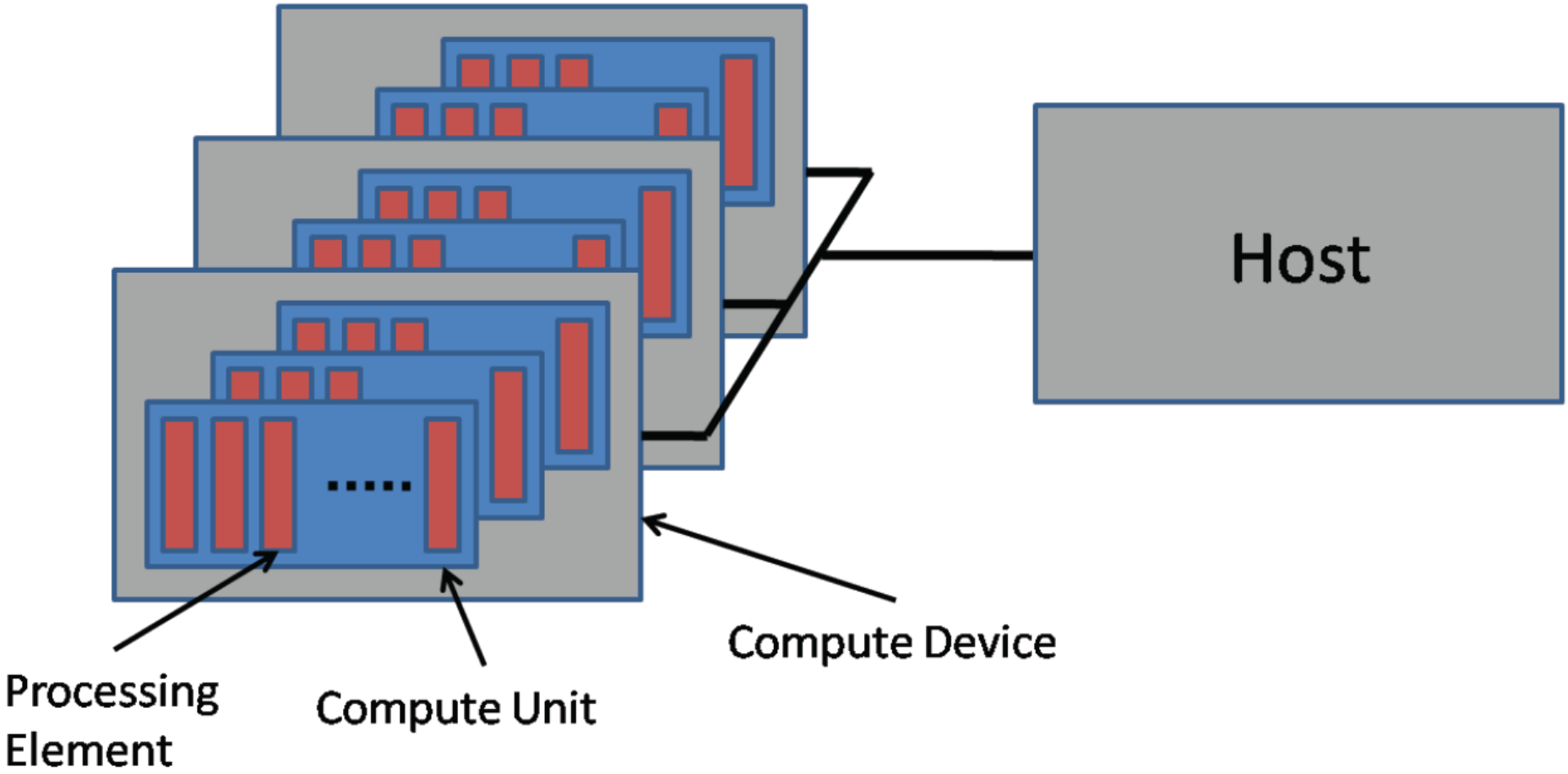
- Marco de trabajo de OpenCL.
  - Definiciones.
    - Plataforma → Host más una colección de dispositivos manejados por el marco de trabajo de OpenCL que permite a una aplicación compartir recursos y ejecutar kernels en los dispositivos de la plataforma.
    - Contexto → Entorno en el que se ejecutan los kernels y en cuyo dominio están definidas la sincronización y el manejo de memoria.
    - Kernel → Función declarada en un programa y ejecutada en un dispositivo OpenCL.
    - Colas de comandos → Objeto que contiene comandos que serán ejecutados en un dispositivo concreto.
    - Objeto memoria → Manejador de una región concreta de memoria global.
    - Objeto kernel → Encapsula un kernel específico así como los valores de los parámetros que serán usados cuando se ejecute ese kernel.

Arquitectura OpenCL

- La plataforma.
  - La plataforma OpenCL se define como un Host conectado a una serie de dispositivos OpenCL.
  - Host → Cualquier computador con una CPU en la que se ejecuta un sistema operativo estándar.
  - Dispositivo OpenCL:
    - GPU, DSP, aceleradores, CPUs multicore…
    - Consisten en una colección de una o más unidades de cómputo.
    - Una unidad de cómputo está compuesta por uno o más elementos de procesamiento.
    - Estos elementos de procesamiento ejecutan instrucciones en modo
      - SIMD (Single Instruction, Multiple Data) en dispositivos con procesadores vectoriales (GPUs) o unidades vectoriales (CPUs).
      - SPMD (Single Program, Multiple Data) en dispositivos de propósito general como CPUs.

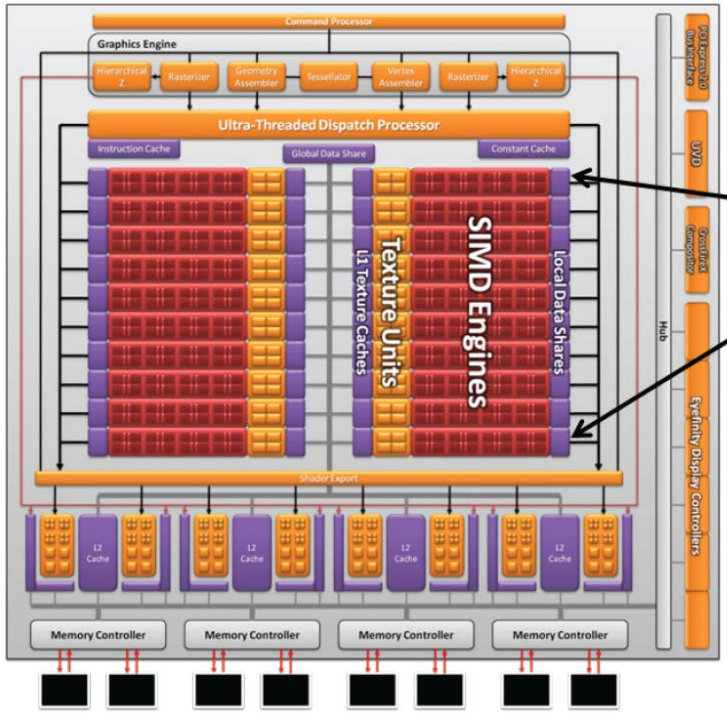
Arquitectura OpenCL

- La plataforma.

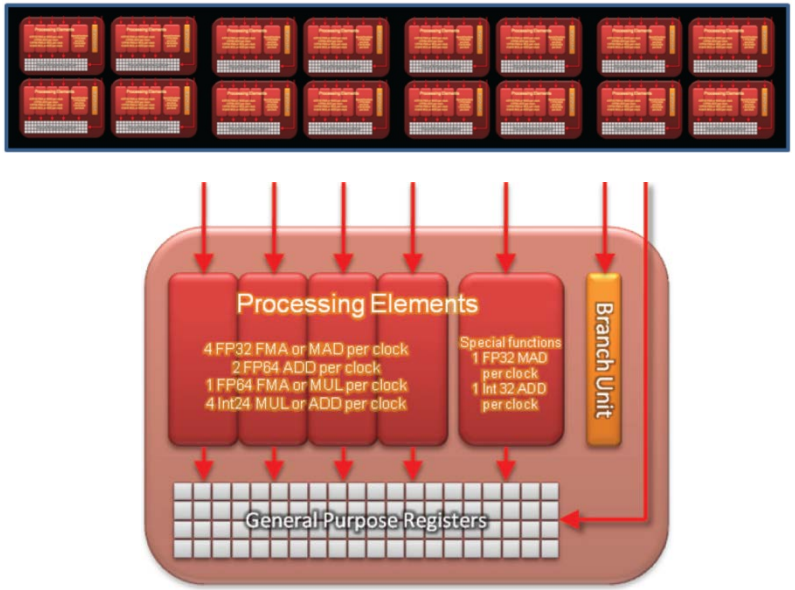


Arquitectura OpenCL

- La plataforma.
  - Ejemplo: ATI Radeon HD 5870



Unidad de cómputo



- 20 Unidades de cómputo (unidades SIMD).
- Unidad de cómputo → 16 cores con 5 elementos de procesamiento → 80 (16x5) elementos de procesamiento.
- GPU → 1600 (80x20) elementos de procesamiento.

Arquitectura OpenCL

- El modelo de ejecución.
  - Comprende dos componentes:
    - Kernel.
      - Unidad básica ejecutable en uno o más dispositivos OpenCL.
      - Similar a una función C que pueda ser paralelizada a nivel de datos o tareas.
    - Programa Host.
      - Se ejecuta en el Host.
      - Define los contextos de dispositivos y encola las instancias de ejecución de los kernels usando colas de comandos.
      - Los kernels son encolados en orden pero pueden ejecutarse en orden o fuera de orden.

Arquitectura OpenCL

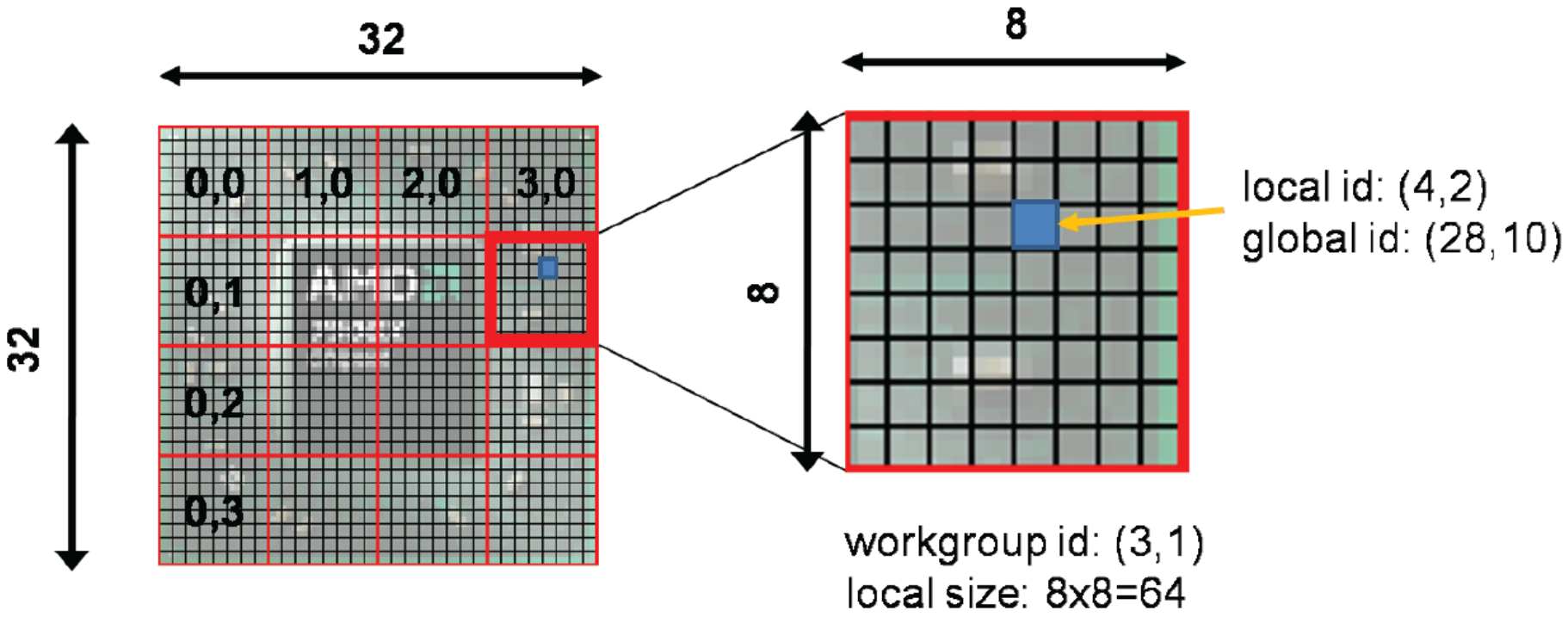
- El modelo de ejecución.
  - Los kernels.
    - OpenCL explota la computación paralela en dispositivos de cómputo definiendo el problema en un espacio de índices de N dimensiones.
      - N=1 → Arrays.
      - N=2 → imágenes.
      - N=3 → Imágenes 3D.
    - Cuando un kernel es encolado para su ejecución por el programa Host, se define un espacio de índices.
    - Cada elemento de ejecución independiente en este espacio de índices se denomina elemento de trabajo o work-item.
    - Cada work-item ejecuta la misma función kernel pero con datos diferentes.
    - Cuando un comando kernel es encolado en la cola de comandos, el espacio de índices debe estar definido para permitir que el dispositivo sepa el número total de work-items que requieren ser ejecutados.

Arquitectura OpenCL

- El modelo de ejecución.
  - Los kernels.
    - Ejemplo: El procesamiento de una imagen de 1024x1024 podría manejarse de la siguiente manera:
      - El espacio de índices comprende un espacio global de 2 dimensiones de 32x32 consistente en un kernel o work-item por cada 1024 pixeles.
      - Total → 1024 ejecuciones.
      - Con este espacio de índices, cada work-item tiene asignado un único identificador global.
    - OpenCL permite agrupar los work-items en grupos de trabajo o work-groups.
      - El tamaño de los work-group es definido por su espacio de índices local.
      - Todos los work-items del mismo work-group son ejecutados juntos en el mismo dispositivo.
      - Esto les permite compartir la memoria local y sincronizarse → los work-items globales son independientes y no pueden sincronizarse. Tampoco los de diferentes work-groups.

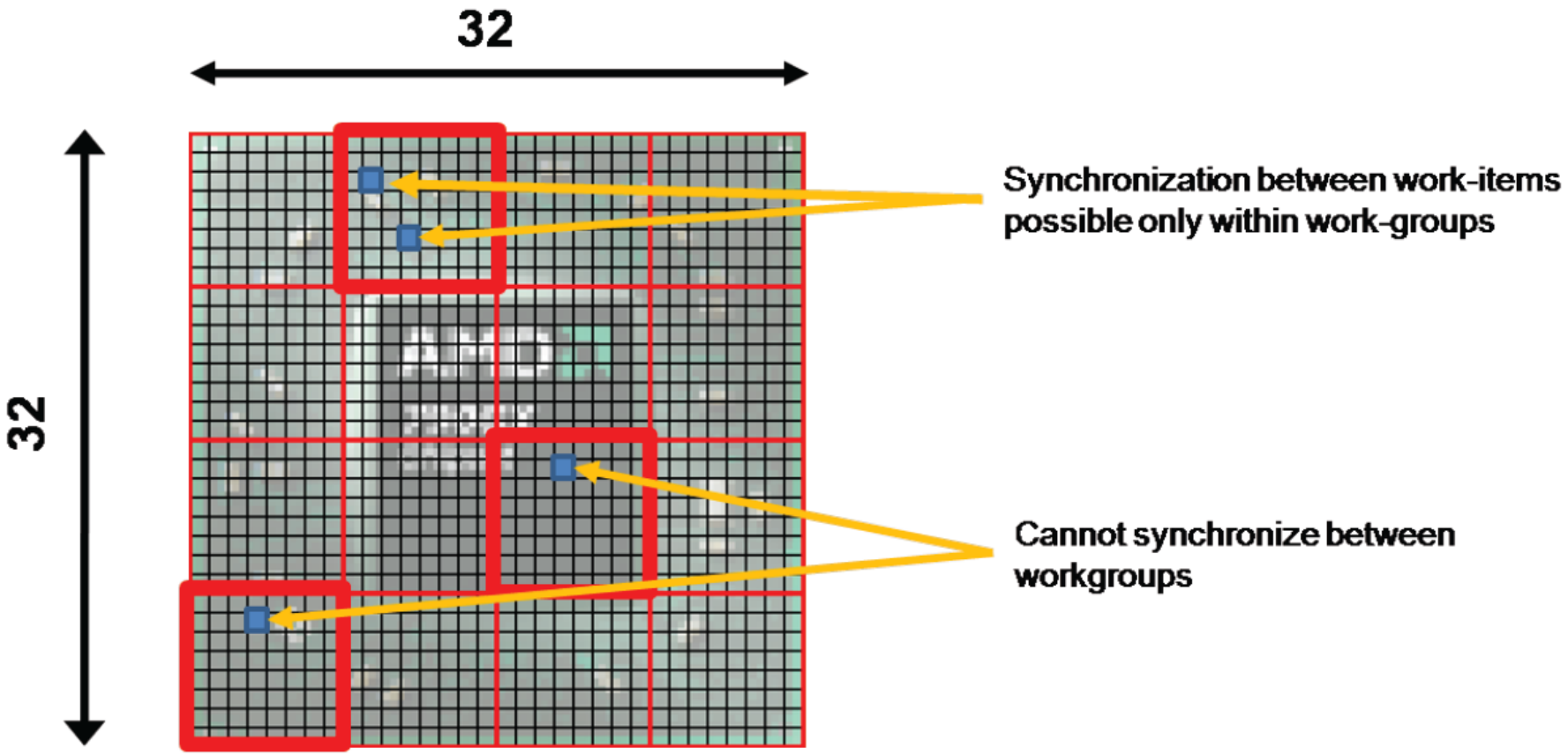
Arquitectura OpenCL

- El modelo de ejecución.
  - Los kernels.



Arquitectura OpenCL

- El modelo de ejecución.
  - Los kernels.



Arquitectura OpenCL

- El modelo de ejecución.
  - Los kernels.
    - Existen dos tipos de kernels:
      - Kernels OpenCL.
        - » Están escritos en el lenguage OpenCL y compilados con el compilador Open CL.
        - » Todos los dispositivos que soportan OpenCL pueden ejecutarlos.
      - Kernels nativos.
        - » Extensiones específicas para aceleradores concretos.
    - Ejemplo: Cuadrado de un vector lineal de n elementos.

Versión escalar

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

Versión OpenCL

```
kernel void dp_square
(global const float *a, global float *result)
{
    int id = get_global_id(0);
    result[id] = a[id]*a[id];
}
// dp_square se ejecuta en "n" work-items
```

Arquitectura OpenCL

- El modelo de ejecución.
  - El programa Host.
    - Es el responsable de configurar y manejar la ejecución de kernels en los dispositivos OpenCL a través de contextos.
    - Usando la API OpenCL, el host puede crear y manipular los contextos a través de los siguientes recursos:
      - Dispositivos: Conjunto de dispositivos OpenCL utilizados por el host para ejecutar los kernels.
      - Objetos de programa: Código fuente o archivo compilado que implementa un kernel o una colección de ellos.
      - Kernels.
      - Objetos de memoria: Conjunto de buffers o mapas de memoria comunes al host y a los dispositivos OpenCL

Arquitectura OpenCL

- El modelo de ejecución.
  - El programa Host.
    - Tras crear el contexto, se crean colas de comandos para manejar la ejecución de kernels en los dispositivos OpenCL asociados al contexto.
    - Las colas de comandos aceptan tres tipos de comandos:
      - Comandos de ejecución de kernel → Ordenan la ejecución de kernels en dispositivos OpenCL (no implica que comience inmediatamente).
      - Comandos de memoria → Transfieren objetos memoria entre el espacio de memoria del host y de los dispositivos OpenCL
      - Comandos de sincronización → Definen el orden en que los comandos son ejecutados.
    - Los comandos se colocan en la cola en orden y se ejecutan:
      - En orden → En el mismo orden de inserción en la cola.
      - Fuera de orden → El orden de los comandos se basa en las restricciones de sincronización indicadas en los comandos.

Arquitectura OpenCL

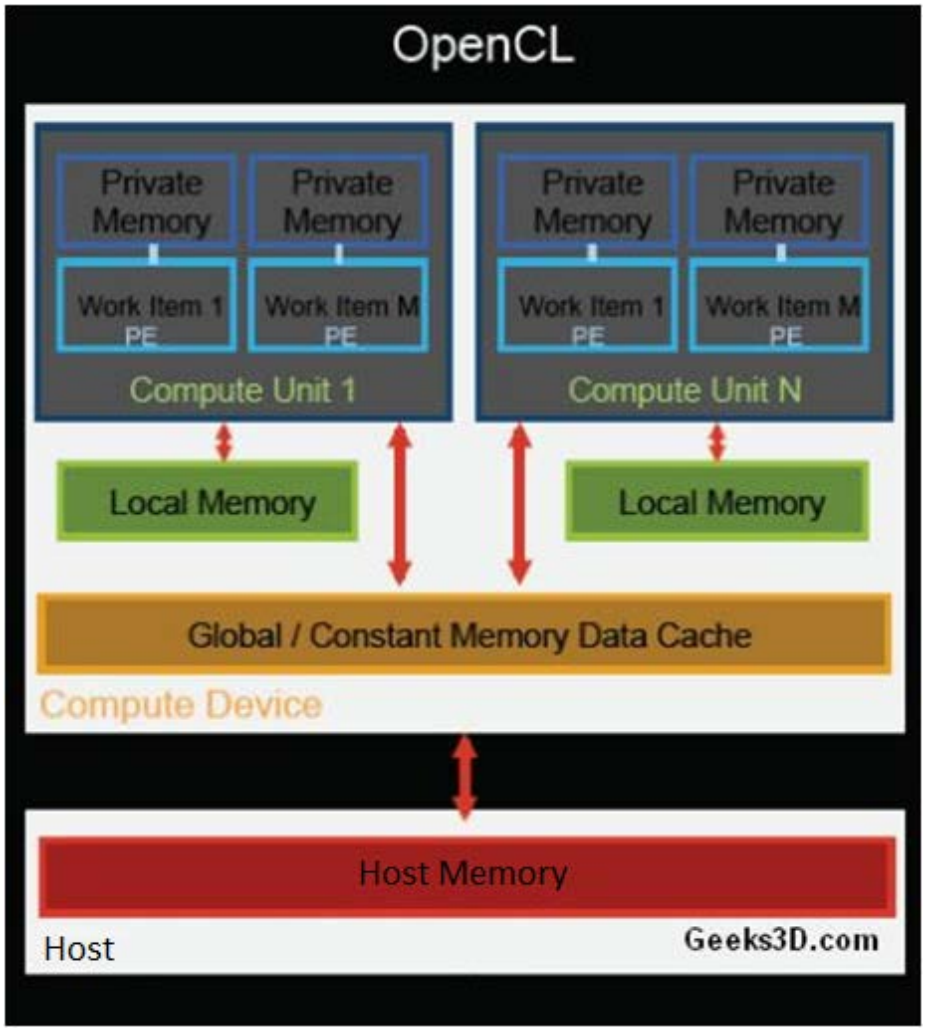
- El modelo de memoria.
  - OpenCL define cuatro regiones de memoria:
    - Memoria Global.
      - Región de memoria en la que todos los work-items y work-groups tienen acceso de lectura/escritura tanto desde el host como desde los elementos de cómputo.
    - Memoria Constante.
      - Región de Memoria Global que permanece constante durante la ejecución de los kernels.
      - El host tiene acceso de lectura/escritura y los work-items de lectura.
    - Memoria Local.
      - Región de memoria usada para compartir datos entre los work-items de un mismo work-group.
      - El host y los work-items tienen acceso de lectura/escritura.
    - Memoria Privada
      - Región de memoria accesible únicamente por un work-item.

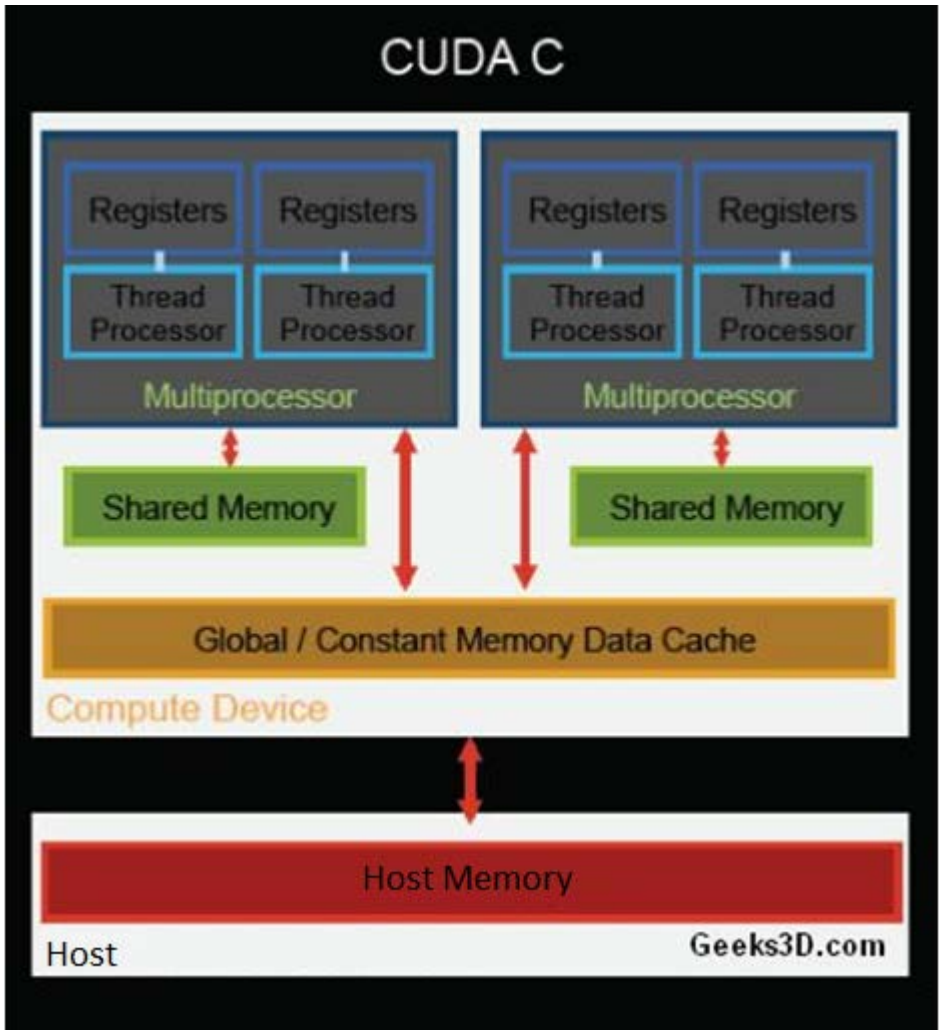
Arquitectura OpenCL

- El modelo de memoria.
  - En muchos casos, la memoria del host y la del dispositivo de cómputo son independientes la una de la otra.
    - Deben definirse manejadores de memoria (objetos de memoria) para permitir la compartición de datos entre ellos.
    - Los datos deben ser explícitamente movidos desde la memoria del host a las memorias del dispositivo y viceversa.
    - Este proceso se realiza mediante la inclusión de comandos de lectura y escritura en la cola de comandos.
    - Estos comandos pueden ser bloqueantes o no bloqueantes.
      - Bloqueantes → El host espera hasta que la transacción de memoria haya concluido.
      - No bloqueantes → El host simplemente pone el comando en la cola y continúa, sin esperar hasta que la transacción de memoria haya terminado.

Arquitectura OpenCL

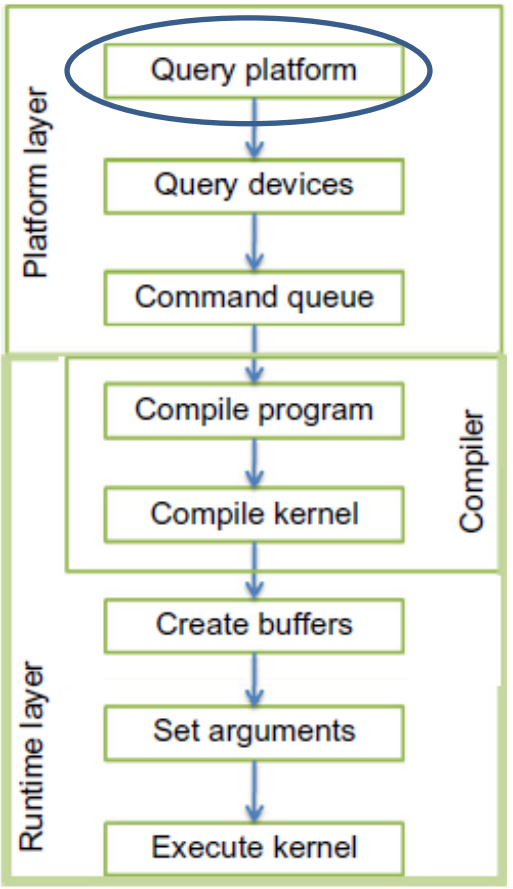
- El modelo de memoria.
  - Comparación con NVIDIA CUDA.





Programando con OpenCL

- Estructura de un programa host en OpenCL.
  - El programa host es el responsable de preparar las tareas paralelas a ejecutar en el dispositivo, estableciendo un contexto para la transmisión de datos, órdenes de ejecución de kernels y recuperación de resultados.
  - Pasos:
    - Consultar la plataforma y los dispositivos OpenCL .
    - Establecer un contexto con el dispositivo de cómputo, definiendo una cola de comandos.
    - Compilar el programa a ejecutar en el dispositivo (kernels).
    - Inicializar los buffers de entrada/salida y establecer los argumentos del kernel.
    - Ejecutar el kernel.



Programando con OpenCL

- Obtener las plataformas.
  - Primer paso en cualquier aplicación OpenCL.
  - Cada implementación de OpenCL define su propia plataforma.
  - Las plataformas permiten interactuar con los dispositivos OpenCL existentes en la máquina.
  - *clGetPlatformIDs*.
    - Obtiene las plataformas OpenCL disponibles en el sistema.
    - Suele invocarse dos veces, una para obtener el número de plataformas y otra para obtener las plataformas.
  - *clGetPlatformInfo*
    - Obtiene información para el usuario de una plataforma OpenCL.

Programando con OpenCL

- Obtener las plataformas.
  - *cl_int clGetPlatformIDs (cl_uint num_entries, cl_platform_id *platforms, cl_uint *num_platforms)*.
    - *num_entries* → Número de plataformas a obtener. Si *platforms* no es NULL, debe ser mayor que 0.
    - *platforms* → Array con las plataformas obtenidas (mínimo entre las solicitadas y las disponibles). Si es NULL, se ignora.
    - *num_platforms* → Número de plataformas obtenidas. Si es NULL, se ignora.
    - Códigos de error (*CL_SUCCESS* si éxito):
      - *CL_INVALID_VALUE* → *num_entries* es cero y *platforms* no es NULL o *platforms* y *num_platforms* son NULL.
      - *CL_OUT_OF_HOST_MEMORY* → problema de asignación de memoria en el host.
    - Para saber el número de plataformas: *num_entries=0* y *platforms=NULL* → *num_platforms* nos indicará cuántas hay disponibles en el sistema.

Programando con OpenCL

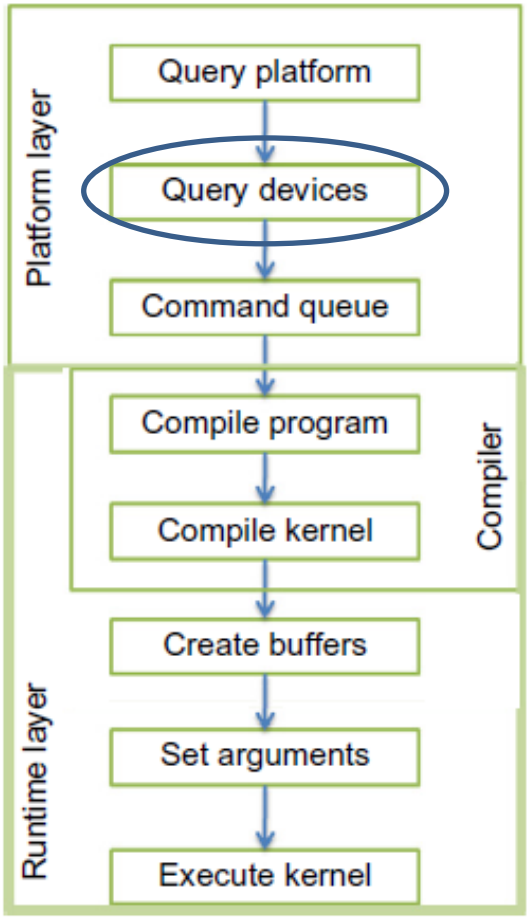
- Obtener las plataformas.
  - *cl_int clGetPlatformInfo (cl_platform_id platform, cl_platform_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)*.
    - *platform* → Plataforma de la que deseamos obtener información.
    - *param_name* → Dato que queremos obtener.
    - *param_value_size* → Tamaño máximo en bytes de *param_value*.
    - *param_value* → Información solicitada.
    - *param_value_size_ret* → devuelve el tamaño en bytes de los datos almacenados en *param_value*.
    - Códigos de error (*CL_SUCCESS* si éxito):
      - *CL_INVALID_PLATFORM* → plataforma no válida.
      - *CL_INVALID_VALUE* → *param_name* tiene un valor inválido o *param_value_size* es menor que el dato que se va a retornar.
      - *CL_OUT_OF_HOST_MEMORY* → problema de asignación de memoria en el host.

Programando con OpenCL

- Obtener las plataformas.
  - *cl_int clGetPlatformInfo* – valores de param_name (char []).
    - *CL_PLATFORM_PROFILE* → tipo de especificación soportada.
      - *FULL_PROFILE* → Especificación completa.
      - *EMBEDDED_PROFILE* → Especificación reducida.
    - *CL_PLATFORM_VERSION* → Versión de OpenCL soportada.
      - OpenCL<space><major_version.minor_version><space><platform-specific information>
      - Cluster de docencia → OpenCL 1.2 LINUX
    - *CL_PLATFORM_NAME* → Nombre de la plataforma.
      - Cluster de docencia → Intel(R) OpenCL
    - *CL_PLATFORM_VENDOR* → Fabricante de la plataforma.
      - Cluster de docencia → Intel(R) Corporation
    - *CL_PLATFORM_EXTENSIONS* → Extensiones soportadas por todos los dispositivos de la plataforma separadas por espacios.

Programando con OpenCL

- Obtener los dispositivos OpenCL.
  - Una vez obtenidas las plataformas, debemos obtener los dispositivos de cada una de ellas.
  - *clGetDeviceIDs*.
    - Obtiene los dispositivos de una plataforma.
    - Se suele invocar dos veces, una para saber cuántos hay y otra para obtenerlos (análogo al mismo proceso en las plataformas).
    - Tipos de dispositivos que se pueden obtener: CPUs, GPUs, Aceleradores, dispositivos por defecto y todos.
  - *clGetDeviceInfo*.
    - Obtiene información para el usuario de un dispositivo OpenCL.



Programando con OpenCL

- Obtener los dispositivos OpenCL.
  - *cl_int clGetDeviceIDs (cl_platform_id platform, cl_device_type device_type, cl_uint num_entries, cl_device_id *devices, cl_uint *num_devices).*
    - *platform* → identificador de la plataforma seleccionada.
    - *device_type* → tipo de dispositivo OpenCL.
      - *CL_DEVICE_TYPE_CPU*
      - *CL_DEVICE_TYPE_GPU*
      - *CL_DEVICE_TYPE_ACCELERATOR*
      - *CL_DEVICE_TYPE_CUSTOM* → Aceleradores dedicados que no soportan programas en OpenCL.
      - *CL_DEVICE_TYPE_DEFAULT* → Dispositivo por defecto del sistema (no puede ser un dispositivo *CL_DEVICE_TYPE_CUSTOM*).
      - *CL_DEVICE_TYPE_ALL* → Todos los dispositivos excepto los dispositivos *CL_DEVICE_TYPE_CUSTOM*.
    - *num_entries* → número de dispositivos a obtener. Si *devices* no es NULL, debe ser mayor que 0.

Programando con OpenCL

- Obtener los dispositivos OpenCL.
  - *cl_int clGetDeviceIDs (cl_platform_id platform, cl_device_type device_type, cl_uint num_entries, cl_device_id *devices, cl_uint *num_devices))*.
    - *devices* → Array con los dispositivos obtenidos (mínimo entre los solicitados y los disponibles). Si es NULL, se ignora.
    - *num_devices* → Número de dispositivos obtenidos.
    - Códigos de error (*CL_SUCCESS* si éxito):
      - *CL_INVALID_PLATFORM* → Plataforma indicada no válida
      - *CL_INVALID_DEVICE_TYPE* → Tipo de dispositivo indicado no válido.
      - *CL_INVALID_VALUE* → *num_entries* es 0 y *devices* no es NULL o tanto *num_devices* como *devices* son NULL.
      - *CL_DEVICE_NOT_FOUND* → No hay dispositivos del tipo indicado.
      - *CL_OUT_OF_RESOURCES* → Problema de asignación de memoria en el dispositivo.
      - *CL_OUT_OF_HOST_MEMORY* → Problema de asignación de memoria en el host.

Programando con OpenCL

- Obtener los dispositivos OpenCL.
  - *cl_int clGetDeviceInfo (cl_device_id device, cl_device_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)*.
    - *device* → dispositivo del que se pretende obtener información.
    - *param_name* → Dato que queremos obtener.
    - *param_value_size* → Tamaño máximo en bytes de *param_value*.
    - *param_value* → Información solicitada. El tipo del puntero depende del tipo de información consultado.
    - *param_value_size_ret* → devuelve el tamaño en bytes de los datos almacenados en *param_value*.
    - Códigos de error (CL_SUCCESS si éxito):
      - CL_INVALID_DEVICE → Dispositivo indicado no válido
      - CL_INVALID_VALUE → *param_name* tiene un valor inválido o *param_value_size* es menor que el dato que se va a retornar.
      - *CL_OUT_OF_RESOURCES* y *CL_OUT_OF_HOST_MEMORY*

Programando con OpenCL

- Obtener los dispositivos OpenCL.
  - *cl_int clGetDeviceInfo* – algunos valores de *param_name*

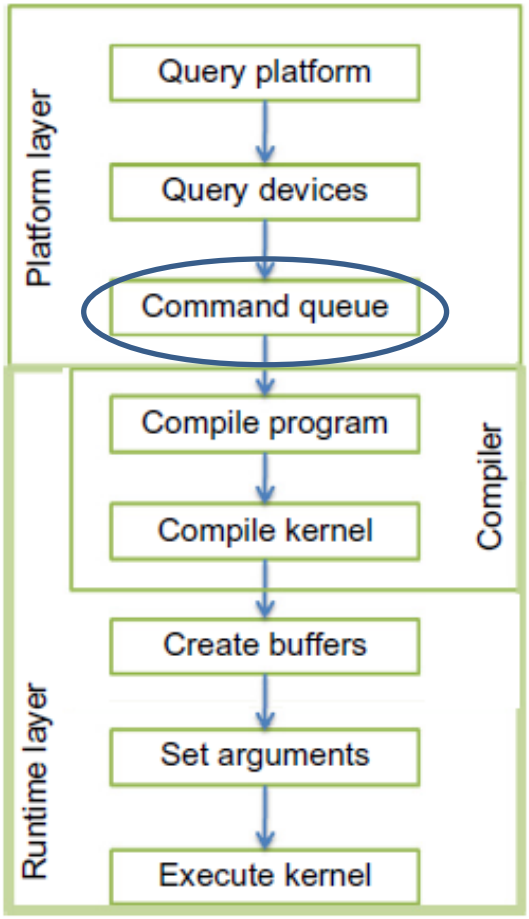
Tipo de consulta	Tipo de dato devuelto	Descripción
CL_DEVICE_AVAILABLE	*cl_bool*	Devuelve CL_TRUE si el dispositivo está disponible y CL_FALSE en caso contario
CL_DEVICE_GLOBAL_MEM_SIZE	*cl_ulong*	Devuelve el tamaño en bytes de la memoria global del dispositivo
CL_DEVICE_LOCAL_MEM_SIZE	*cl_ulong*	Devuelve el tamaño en bytes de la memoria local del dispositivo
CL_DEVICE_MAX_CLOCK_FREQUENCY	*cl_uint*	Devuelve la frecuencia de reloj del dispositivo en MHz
CL_DEVICE_MAX_COMPUTE_UNITS	*cl_uint*	Devuelve el número de unidades de cómputo del dispositivo
CL_DEVICE_MAX_WORK_GROUP_SIZE	*size_t*	Devuelve el número máximo de work-items que pueden ser contenidos en un work-group
CL_DEVICE_NAME	*char[]*	Devuelve el nombre del dispositivo
CL_DEVICE_TYPE	*cl_device_type*	Devuelve el tipo del dispositivo

Programando con OpenCL

- Ejercicio.
  - Crear una función que obtenga las plataformas disponibles del cluster de docencia mostrando la información de cada plataforma.
    - *cl_int ObtenerPlataformas(cl_platform_id *&platforms, cl_uint &num_platforms)*.
  - Crear una función que obtenga los dispositivos de una plataforma dada mostrando la información de cada uno.
    - *cl_int ObtenerDispositivos(cl_platform_id platform, cl_device_type device_type, cl_device_id *&devices, cl_uint &num_devices)*
  - Includes:
    - Debe incluirse <CL/cl.h>
  - Compilación:
    - gcc codigo.c -o ejecutable -lOpenCL
    - g++ codigo.cpp -o ejecutable -lOpenCL

Programando con OpenCL

- Creación de contextos.
  - Una vez obtenidos los dispositivos OpenCL, éstos deben ser asociados a contextos.
  - El contexto es utilizado para manejar las colas de comandos, los objetos de programas, los objetos kernel y para compartir los objetos de memoria de los dispositivos asociados con dicho contexto.
  - Para crear un contexto se utiliza la función *clCreateContext*.



Programando con OpenCL

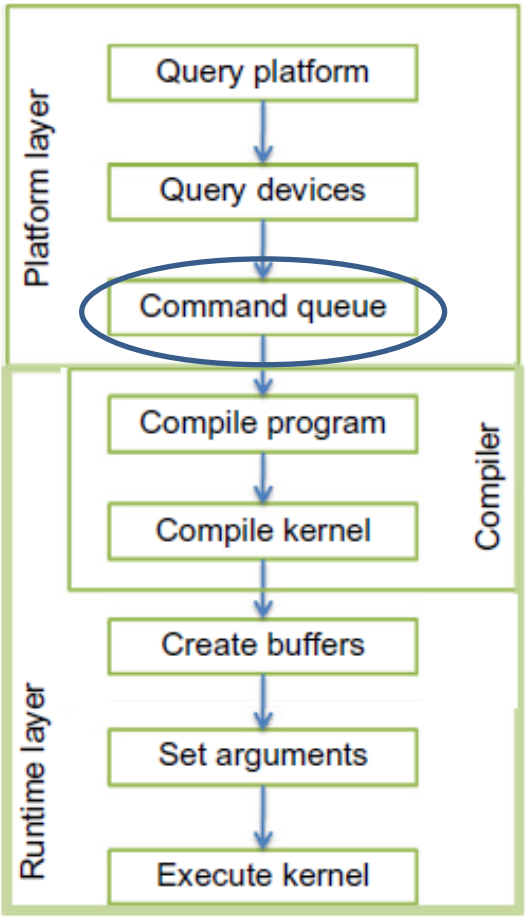
- Creación de contextos.
  - *cl_context clCreateContext (cl_context_properties *prop, cl_uint num_devices, const cl_device_id *devices, void *pfn_notify, void *user_data, cl_int *errcode_ret)*.
    - *prop* → Array que especifica una lista de propiedades de contexto.
      - Las propiedades se definen por parejas de datos:
        - » *prop[i]* → Propiedad (actualmente sólo puede ser *CL_CONTEXT_PLATFORM*).
        - » *Prop[i+1]* → Valor de la propiedad (plataforma del contexto).
      - El último valor del array debe ser 0.
    - *num_devices* → Número de dispositivos de *devices* que se asociarán
    - *devices* → Array de dispositivos.
    - *pfn_notify* → Puntero a función de callback usado para registrar información de errores en el contexto. Puede ser NULL.
    - *user_data* → Argumentos de entrada de pfn_notify. Puede ser NULL.
    - *errcode_ret* → Código de error. Puede ser NULL.

Programando con OpenCL

- Creación de contextos.
  - *clCreateContext* – Códigos de error (*CL_SUCCESS* si éxito):
    - *CL_INVALID_PLATFORM* → La lista de propiedades es NULL o la plataforma indicada no es válida.
    - *CL_INVALID_VALUE* → Puede ser por las siguientes causas:
      - El nombre de una propiedad no es válido.
      - El número de dispositivos es 0.
      - *devices* es NULL.
      - Algún dispositivo en *devices* no es válido o no está asociado con la plataforma.
    - *CL_DEVICE_NOT_AVAILABLE* → Algún dispositivo de *devices* no está actualmente disponible.
    - *CL_OUT_OF_HOST_MEMORY*.

Programando con OpenCL

- Creación de colas de comandos.
  - Una vez establecido el contexto, se crean las colas de comandos.
  - Estas colas permiten enviar comandos a los dispositivos de cómputo asociados con el contexto.
  - Los comandos se encolan en la cola de comandos en orden.
    - No tienen porqué ejecutarse en orden.
  - Para crear una cola de comandos se utilizan las funciones *clCreateCommandQueue* (OpenCL 1.2) *clCreateCommandQueueWithProperties* (OpenCL 2.0)



Programando con OpenCL

- Creación de colas de comandos.
  - *cl_command_queue clCreateCommandQueueWithProperties (cl_context context, cl_device_id device, cl_command_queue_properties *prop, cl_int *errcode_ret)*.
    - *context* → Contexto al que se asociará la cola.
    - *device* → Dispositivo con el que crear la cola de comandos.
    - *prop* → Propiedades que debe cumplir la cola de comandos.
    - *errcode_ret* → código de error. Puede ser NULL.
      - *CL_SUCCESS*.
      - *CL_INVALID_CONTEXT* → Contexto no válido.
      - *CL_INVALID_DEVICE* → Dispositivo no válido o no asociado al contexto.
      - *CL_INVALID_VALUE* → Valor especificado en *prop* no válido.
      - *CL_INVALID_QUEUE_PROPERTIES* → propiedad válida pero no soportada.
      - *CL_OUT_OF_RESOURCES*
      - *CL_OUT_OF_HOST_MEMORY*

Programando con OpenCL

- Creación de colas de comandos.
  - *clCreateCommandQueueWithProperties* – Propiedades
    - *CL_QUEUE_PROPERTIES* (de tipo *cl_command_queue_properties*).
      - Conjunto de propiedades → Cada propiedad es un bit. Para activar varias propiedades se usa la operación *bitwise OR (|)*.
      - Si no se especifican propiedades, se crea una cola en orden.
      - *CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE* → Si se activa, los comandos se ejecutan fuera de orden, sino, en orden.
      - *CL_QUEUE_PROFILING_ENABLE* → Si se activa, los comandos de profiling están disponibles.
      - *CL_QUEUE_ON_DEVICE* → Si se activa, la cola se crea en el dispositivo (requiere *CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE*).
    - *CL_QUEUE_SIZE* (de tipo *cl_uint*).
      - Establece el tamaño de la cola en bytes.
      - Debe estar activada la propiedad *CL_QUEUE_ON_DEVICE*.
      - Debe ser menor que *CL_DEVICE_QUEUE_ON_DEVICE_MAX_SIZE*.
      - Tamaño por defecto *CL_DEVICE_QUEUE_ON_DEVICE_PREFERRED_SIZE*.

Programando con OpenCL

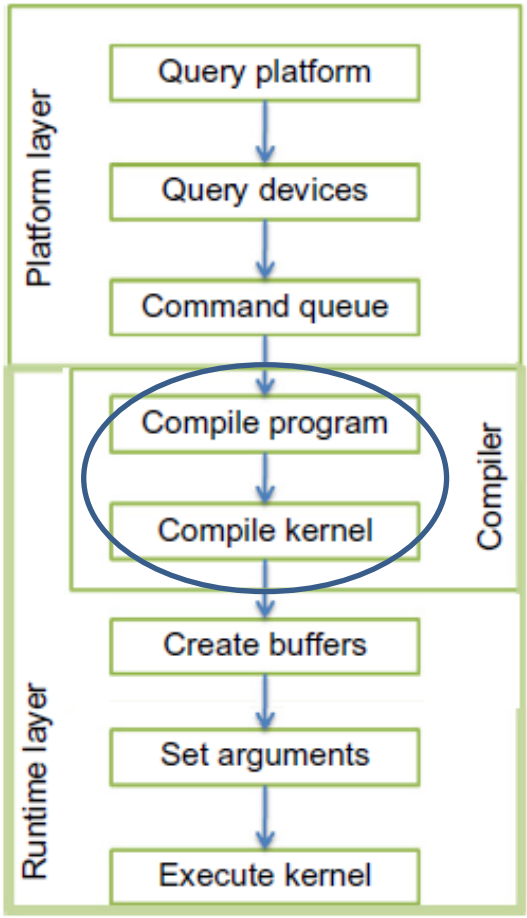
- Creación de colas de comandos.
  - *cl_command_queue clCreateCommandQueue (cl_context context, cl_device_id device, cl_command_queue_properties properties, cl_int *errcode_ret)*.
    - Obsoleta en OpenCL 2.0, vigente en OpenCL 1.2.
      - Necesaria en dispositivos que no tengan la implementación 2.0 (NVIDIA).
    - *context*, *device* y *errcode_ret* → Igual que *clCreateCommandQueueWithProperties*
    - *prop* → Propiedades que debe cumplir la cola de comandos. Cada propiedad es un bit. Para activar varias propiedades se usa la operación *bitwise OR (|)*.
      - *CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE* → Si se activa, los comandos se ejecutan fuera de orden, sino, en orden.
      - *CL_QUEUE_PROFILING_ENABLE* → Si se activa, los comandos de profiling están disponibles.

Programando con OpenCL

- Ejercicio.
  - Crear una función que cree un contexto dados una plataforma y su lista de dispositivos.
    - *cl_int CrearContexto(cl_platform_id platform, cl_device_id *devices, cl_uint num_devices, cl_context &contexto).*
  - Crear una función que cree una cola de comandos dado un contexto, un dispositivo y las propiedades de la cola (usar la versión para OpenCL 1.2).
    - *cl_int CrearCola(cl_context contexto, cl_device_id device_id, cl_command_queue_properties prop, cl_command_queue &cola).*

Programando con OpenCL

- Creación de programas y kernels.
  - Un programa OpenCL (.cl) es una colección de funciones (kernels) escritas en lenguaje OpenCL.
  - Lo compila el host indicando el contexto que se usará para ejecutarlo.
  - Pasos:
    1. Leer el código fuente y almacenarlo en una cadena de caracteres.
    2. Crear el objeto programa a partir del código fuente.
    3. Compilar el objeto programa.
    4. Capturar los posibles fallos de compilación.
    5. Crear el objeto kernel con la función del programa a ejecutar por los dispositivos del contexto.



Programando con OpenCL

- Creación de programas y kernels – Paso 2
  - *cl_program clCreateProgramWithSource (cl_context context, cl_uint count, const char **strings, const size_t *lengths, cl_int *errcode_ret).*
    - *context* → Contexto de los dispositivos asociados al programa.
    - *count* → Número de cadenas en el parámetro *strings*.
    - *strings* → Cadenas con el código fuente del programa.
    - *lengths* → Longitudes de cada cadena de *strings*. Puede ser NULL si las cadenas de *strings* terminan con el valor 0.
    - *errcode_ret* → Código de error. Puede ser:
      - *CL_SUCCESS*.
      - *CL_INVALID_CONTEXT*.
      - *CL_INVALID_VALUE* → Si *count* es 0 o *strings* o cualquiera de sus entradas es NULL.
      - *CL_OUT_OF_RESOURCES*.
      - *CL_OUT_OF_HOST_MEMORY*.

Programando con OpenCL

- Creación de programas y kernels – Paso 3
  - *cl_int clBuildProgram (cl_program program, cl_uint num_devices, const cl_device_id *device_list, const char *options, void *pfn_notify, void *user_data)*.
    - *program* → Objeto programa a compilar.
    - *num_devices* → Número de dispositivos en *device_list*.
    - *device_list* → Lista de dispositivos asociados con el programa. Si es NULL, el ejecutable se genera para todos los dispositivos.
    - *options* → Opciones de compilación. Debe terminar en 0.
    - *pfn_notify* → Función de retorno. Sus parámetros son *program* y *user_data*. Puede ser NULL.
    - *user_data* → Parámetro de *pfn_notify*. Puede ser NULL.

Programando con OpenCL

- Creación de programas y kernels – Paso 3
  - *clBuildProgram* – Códigos de error (*CL_SUCCESS* si éxito):
    - *CL_INVALID_PROGRAM*
    - *CL_INVALID_VALUE* → Si *device_list* es NULL y *num_devices>0* o viceversa, o si *pfn_notify* es NULL pero *user_data* no.
    - *CL_INVALID_DEVICE* → Si algún dispositivo de *device_list* no está asociado a *program*.
    - *CL_INVALID_BUILD_OPTIONS*.
    - *CL_INVALID_OPERATION* → Si se está intentando compilar el código para algún dispositivo y no ha terminado de generarse en alguna llamada previa a *clBuildProgram*, hay objetos kernel asociados a *program* o si *program* no se creó correctamente.
    - *CL_COMPILER_NOT_AVAILABLE*.
    - *CL_BUILD_PROGRAM_FAILURE* → Error de compilación del kernel.
    - *CL_OUT_OF_RESOURCES*.
    - *CL_OUT_OF_HOST_MEMORY* .

Programando con OpenCL

- Creación de programas y kernels – Paso 4
  - *cl_int clGetProgramBuildInfo (cl_program program, cl_device_id device, cl_program_build_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)*.
    - *program* → Objeto programa.
    - *device* → Dispositivo para el que se compiló el programa.
    - *param_name* → Información que se solicita.
    - *param_value_size* → Tamaño del parámetro *param_value*.
    - *param_value* → Información devuelta.
    - *param_value_size_ret* → Tamaño del valor devuelto en *param_value*
    - Códigos de error (*CL_SUCCESS* si éxito):
      - *CL_INVALID_DEVICE*, *CL_INVALID_PROGRAM*.
      - *CL_INVALID_VALUE* → *param_name* no válido, *param_value_size* es menor que el tamaño devuelto en *param_value* y éste no es NULL.

Programando con OpenCL

- Creación de programas y kernels – Paso 4
  - *clGetProgramBuildInfo* – param_name.
    - *CL_PROGRAM_BUILD_STATUS* (*cl_build_status*) :
      - *CL_BUILD_NONE* → No se ha realizado ninguna generación, compilación o linkado sobre el programa y dispositivo especificados.
      - *CL_BUILD_ERROR*.
      - *CL_BUILD_SUCCESS*.
      - *CL_BUILD_IN_PROGRESS*.
    - *CL_PROGRAM_BUILD_OPTIONS (char[]).*
    - *CL_PROGRAM_BUILD_LOG (char[])* → Errores de compilación.

Programando con OpenCL

- Creación de programas y kernels – Paso 5
  - *cl_kernel clCreateKernel (cl_program program, const char *kernel_name, cl_int *errcode_ret)*.
    - *program* → Objeto programa del que se cogerá el kernel.
    - *kernel_name* → Función dentro de *program*. Bebe ser declarada con el calificador *__kernel*.
    - *errcode_ret* → Código de error. Puede ser NULL:
      - *CL_SUCCESS* si éxito
      - *CL_INVALID_PROGRAM*.
      - *CL_INVALID_PROGRAM_EXECUTABLE* → No hay un ejecutable válido para *program*.
      - *CL_INVALID_KERNEL_NAME*.
      - *CL_INVALID_KERNEL_DEFINITION* → Errores en la definición del kernel.
      - *CL_INVALID_VALUE* → *kernel_name* es NULL.
      - *CL_OUT_OF_RESOURCES*.
      - *CL_OUT_OF_HOST_MEMORY*.

Programando con OpenCL

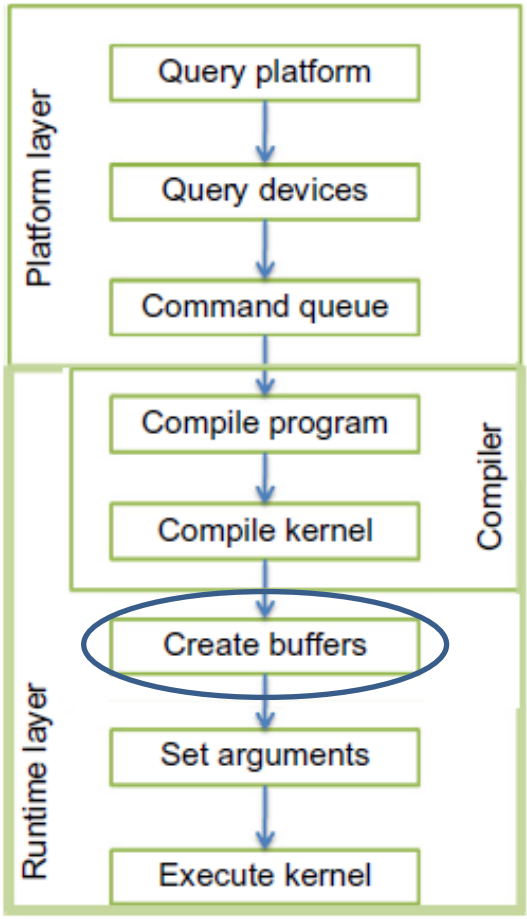
- Ejercicio.
  - Crear un fichero programa.cl con el siguiente kernel:

```
__kernel void mult_vec(__global float *input, __global float *output) {
    size_t id = get_global_id(0);
    output[id] = input[id]*input[id];
}
```

  - Crear una función que cree un objeto programa con el código anterior.
    - *cl_int CrearPrograma(cl_program &program, cl_context context, cl_uint num_devices, const cl_device_id *device_list, const char *options, const char *fichero)*.
    - Introducir un error en el código del programa para ver qué ocurre.
    - Usar la función *leerFuentes* (Campus Virtual) para leer el programa.
  - Crear una función que cree un objeto kernel.
    - *cl_int CrearKernel(cl_kernel &kernel, cl_program program, const char *kernel_name)*.

Programando con OpenCL

- Creación de buffers.
  - Son las estructuras de entrada/salida del kernel.
  - Los hemos denominado objetos memoria.
    - Manejarán los vectores y las imágenes 2D y 3D.
  - Para valores constantes no es necesario crear objetos memoria.
  - La creación o inicialización de un objeto memoria no implica la transferencia de datos al dispositivo.



Programando con OpenCL

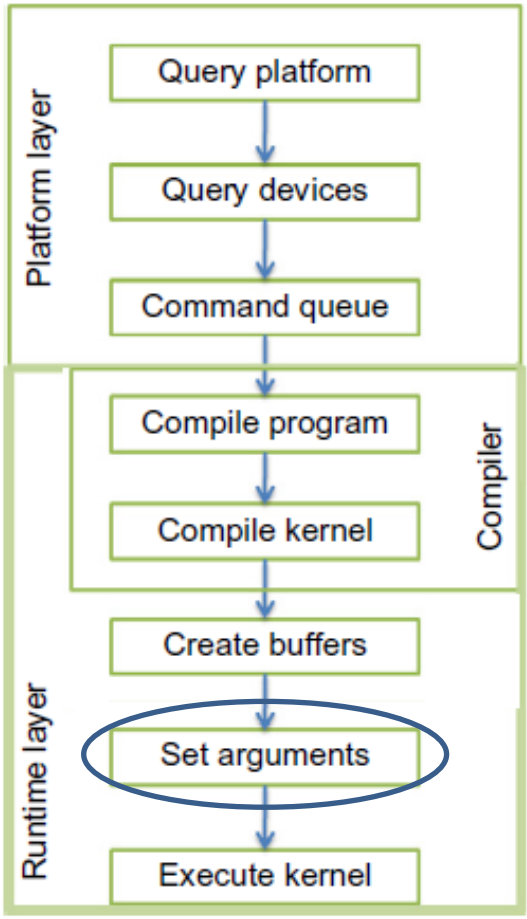
- Creación de buffers – arrays unidimensionales
  - *cl_mem clCreateBuffer (cl_context context, cl_mem_flags flags, size_t size, void *host_ptr, cl_int *errcode_ret)*.
    - *context* → Contexto al que se asociará el buffer.
    - *flags* → Flags que van a indicar qué tipo de operaciones pueden hacerse con el objeto creado.
    - *size* → Tamaño en bytes del buffer de memoria utilizado.
    - *host_post* → Buffer de datos que será enlazado por el objeto.
    - *errcode_ret* → Códigos de error (CL_SUCCESS si éxito). Puede ser NULL:
      - *CL_INVALID_CONTEXT*.
      - *CL_INVALID_VALUE* → Valores de *flags* no válidos.
      - *CL_INVALID_BUFFER_SIZE* → size es igual a 0.
      - *CL_INVALID_HOST_PTR* → *host_ptr* es NULL y se han establecido el flag *CL_MEM_USE_HOST_PTR* o el *CL_MEM_COPY_HOST_PTR*, o viceversa.
      - *CL_MEM_OBJECT_ALLOCATION_FAILURE* → Fallo de reserva de memoria del buffer.

Programando con OpenCL

- Creación de buffers.
  - *cl_mem_flags*
    - *CL_MEM_READ_WRITE* → El kernel tiene acceso de lectura/escritura.
    - *CL_MEM_WRITE_ONLY* → El kernel tiene acceso sólo de escritura.
    - *CL_MEM_READ_ONLY* → El kernel tiene acceso sólo de lectura.
    - *CL_MEM_HOST_WRITE_ONLY* → El host tiene acceso sólo de escritura.
    - *CL_MEM_HOST_READ_ONLY* → El host tiene acceso sólo de lectura.
    - *CL_MEM_HOST_NO_ACCESS* → El host no tiene acceso.
    - *CL_MEM_USE_HOST_PTR* → Se usa la dirección de memoria apuntada por *host_ptr* (el dispositivo puede cachear).
    - *CL_MEM_COPY_HOST_PTR* → Se copia el contenido de *host_ptr* en la memoria del dispositivo.

Programando con OpenCL

- Asignación de parámetros.
  - Una vez que tenemos los objetos de memoria, debemos enlazarlos con los diferentes parámetros del kernel.
  - Usaremos la función *cl_int clSetKernelArg*.
  - Habrá que hacer una llamada para cada parámetro del kernel.



Programando con OpenCL

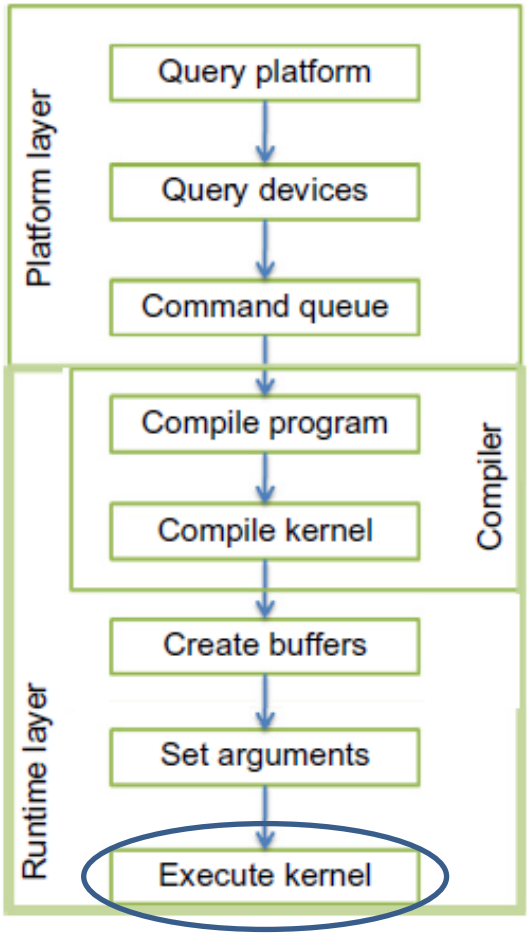
- Asignación de parámetros.
  - *cl_int clSetKernelArg (cl_kernel kernel, cl_uint arg_index, size_t arg_size, const void *arg_value)*.
    - *kernel* → Objeto kernel.
    - *arg_index* → Orden del parámetro al que enlazar *arg_value*. Valor entre 0 y n-1, donde n es el número de parámetros.
    - *arg_size* → Tamaño de *arg_value*.
    - *arg_value* → Variable u objeto de memoria que se usará como argumento.
      - Se usan variables para elementos simples y objetos memoria para arrays e imágenes.

Programando con OpenCL

- Asignación de parámetros.
  - *clSetKernelArg* – Códigos de error (*CL_SUCCESS* si éxito):
    - *CL_INVALID_KERNEL*.
    - *CL_INVALID_ARG_INDEX* → *arg_index<0* o *arg_index>n-1*.
    - *CL_INVALID_ARG_VALUE*.
    - *CL_INVALID_MEM_OBJECT* → Parámetro declarado como objeto memoria pero *arg_value* no lo es.
    - *CL_INVALID_ARG_SIZE* → *arg_size* no coincide con el tamaño de *arg_value*.
    - *CL_OUT_OF_RESOURCES*.
    - *CL_OUT_OF_HOST_MEMORY*.

Programando con OpenCL

- Ejecución del kernel.
  - En este punto, el host y el dispositivo pueden comunicarse → Siguiente paso: ejecutar el kernel en el dispositivo.
  - La ejecución del kernel requiere los siguientes pasos (los pasos entre [] son opcionales y dependerán de la aplicación):
    - [Transferencia de los datos de los objetos de memoria del host al dispositivo].
    - Ejecución del kernel.
    - [Espera por la finalización del kernel].
    - [Transferencia de los datos de los objetos de memoria del dispositivo al host].
    - [Obtención de tiempos de ejecución].



Programando con OpenCL

- Ejecución del kernel – Transferencia Host-Dispositivo.
  - Para buffers sin el flag *CL_MEM_USE_HOST_PTR* (se usa la propia memoria del Host, por lo que no hay transferencia).
  - *cl_int clEnqueueWriteBuffer (cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_write, size_t offset, size_t cb, const void *ptr, cl_uint num_events, const cl_event *event_wait_list, cl_event *event)*.
    - *command_queue* → Cola de comandos asociada al dispositivo.
    - *buffer* → Objeto de memoria usado para transferir los datos.
    - *blocking_write* → *CL_TRUE* (transferencia síncrona) o *CL_FALSE* (transferencia asíncrona).
    - *offset* → Desplazamiento en bytes de los datos a transferir.
    - *cb* → Tamaño en bytes de los datos a transferir.
    - *ptr* → Dirección de memoria del host que contiene los datos a transferir a través del objeto de memoria *buffer*.

Programando con OpenCL

- Ejecución del kernel – Transferencia Host-Dispositivo.
  - *cl_int clEnqueueWriteBuffer (cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_write, size_t offset, size_t cb, const void *ptr, cl_uint num_events, const cl_event *event_wait_list, cl_event *event)*.
    - *num_events* → Número de eventos contenidos en event_wait_list.
    - *event_wait_list*:
      - Lista de eventos que deben darse antes de que se ejecute la orden.
      - Los eventos funcionan como elementos de sincronización.
      - Puede ser NULL → En ese caso, no se espera por ningún evento.
      - El contexto asociado a los eventos de *event_wait_list* y de *command_queue* debe ser el mismo.
    - *event* → Evento asociado a esta orden que indica su terminación. Puede ser NULL.

Programando con OpenCL

- Ejecución del kernel – Transferencia Host-Dispositivo.
  - *clEnqueueWriteBuffe*r – Códigos de error (CL_SUCCESS si éxito):
    - *CL_INVALID_COMMAND_QUEUE* → *command_queue* no es válida.
    - *CL_INVALID_CONTEXT* → El contexto asociado a *command_queue* no es el mismo que los asociados a buffer y a los eventos.
    - *CL_INVALID_MEM_OBJECT* → buffer no es un objeto de memoria válido.
    - *CL_INVALID_EVENT_WAIT_LIST* → *event_wait_list* es NULL y *num_events>0* o viceversa, o si los objetos de *event_wait_list* no son eventos válidos.
    - *CL_INVALID_OPERATION* → buffer se creó con *CL_MEM_HOST_WRITE_ONLY* y se usó para leer del dispositivo, se creó con *CL_MEM_HOST_READ_ONLY* y se usó para escribir en el dispositivo, o se creó con *CL_MEM_HOST_NO_ACCESS*.
    - *CL_OUT_OF_RESOURCES* y *CL_OUT_OF_HOST_MEMORY*.

Programando con OpenCL

- Ejecución del kernel – Ejecución del kernel.
  - *cl_int clEnqueueNDRangeKernel (cl_command_queue command_queue, cl_kernel kernel, cl_uint work_dim, const size_t *global_work_offset, const size_t *global_work_size, const size_t *local_work_size, cl_uint num_events, const cl_event *event_wait_list, cl_event *event)*.
    - *command_queue* → Cola de comandos asociada al dispositivo.
    - *kernel* → Objeto kernel que identifica la función a ejecutar.
    - *work_dim* → Número de dimensiones del problema [1-3].
    - *global_work_offset* → Desplazamiento para calcular el identificador global de los work-items. Puede ser NULL.
    - *global_work_size* → variable/array de *work_dim* elementos que indican el número de work-items a lanzar en cada dimensión.
      - work_items = global_work_size[0]*…*global_work_size[work_dim-1].

Programando con OpenCL

- Ejecución del kernel – Ejecución del kernel.
  - *cl_int clEnqueueNDRangeKernel (cl_command_queue command_queue, cl_kernel kernel, cl_uint work_dim, const size_t *global_work_offset, const size_t *global_work_size, const size_t *local_work_size, cl_uint num_events, const cl_event *event_wait_list, cl_event *event)*.
    - *local_work_size* → variable/array de *work_dim* elementos que describen el número de work-items que formarán un work-group para cada dimensión.
      - Work_items por work_group = local_work_size[0] *…* local_work_size[work_dim-1] .
      - *local_work_size[x]* debe ser menor que *CL_DEVICE_MAX_WORK_ITEM_SIZES[x]* (x = [0.. work_dim-1]).
      - El número total de work_items por work_group debe ser menor que *CL_KERNEL_WORK_GROUP_SIZE*.
    - *num_events*, *event_wait_list* y *event* → Igual que en *clEnqueueWriteBuffer*.

Programando con OpenCL

- Ejecución del kernel – Ejecución del kernel.
  - *clEnqueueNDRangeKernel* – Códigos de error (CL_SUCCESS si éxito):
    - *CL_INVALID_PROGRAM_EXECUTABLE*.
    - *CL_INVALID_COMMAND_QUEUE*.
    - *CL_INVALID_KERNEL*.
    - *CL_INVALID_CONTEXT*.
    - *CL_INVALID_KERNEL_ARGS* → No se han definido los parámetros del kernel o no coinciden los tipos.
    - *CL_INVALID_WORK_DIMENSION* → *work_dim* no es un valor válido.
    - *CL_INVALID_GLOBAL_WORK_SIZE* → *global_work_size* es NULL o alguno de sus valores es 0 o demasiado grande.
    - *CL_INVALID_WORK_GROUP_SIZE* → Tamaño de grupo no válido.
    - *CL_INVALID_WORK_ITEM_SIZE* → Número de work_items no válido (excede el total global o el total de alguna dimensión).
    - *CL_OUT_OF_RESOURCES* y *CL_OUT_OF_HOST_MEMORY*.

Programando con OpenCL

- Ejecución del kernel – Espera por la finalización.
  - *cl_int clFinish (cl_command_queue command_queue)*
    - *command_queue* → Cola a la que queremos consultar.
    - Bloquea al Host hasta que *command_queue* haya terminado.
    - Códigos de error (*CL_SUCCESS* si éxito): *CL_OUT_OF_RESOURCES, CL_INVALID_COMMAND_QUEUE y CL_OUT_OF_HOST_MEMORY*.
- Ejecución del kernel – Transferencia Dispositivo-Host.
  - Para buffers sin el flag *CL_MEM_USE_HOST_PTR*.
  - *cl_int clEnqueueReadBuffer (cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_read, size_t offset, size_t cb, void *ptr, cl_uint num_events, const cl_event *event_wait_list, cl_event *event)*.
  - Mismos parámetros (referidos en este caso a lectura) y códigos de error que *clEnqueueWriteBuffer*.

Programando con OpenCL

- Ejecución del kernel – Obtención de tiempos.
  - *cl_int clGetEventProfilingInfo (cl_event event, cl_event_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)*.
    - *event* → Evento cuya información se quiere consultar.
    - *param_name* → Información a consultar.
      - *CL_PROFILING_COMMAND_QUEUED (cl_ulong)* → Momento en ns en que el comando fue insertado en la cola.
      - *CL_PROFILING_COMMAND_SUBMIT (cl_ulong)* → Momento en ns en que el comando fue enviado del host al dispositivo.
      - *CL_PROFILING_COMMAND_START (cl_ulong)* → Momento en ns en que el comando comenzó su ejecución.
      - *CL_PROFILING_COMMAND_END (cl_ulong)* → Momento en ns en que el comando terminó su ejecución.
    - *param_value_size* → Tamaño en bytes de *param_value*.
    - *param_value* → Resultado de la consulta.
    - *param_value_size_ret* → Tamaño de la información devuelta.

Programando con OpenCL

- Ejecución del kernel – Obtención de tiempos.
  - *clGetEventProfilingInfo* – Códigos de error (CL_SUCCESS si éxito).
    - *CL_PROFILING_INFO_NOT_AVAILABLE* → No se activó el flag *CL_QUEUE_PROFILING_ENABLE* al crear la cola de comandos.
    - *CL_INVALID_VALUE* → *param_name* no es válido o el tamaño especificado por *param_value_size* es menor que el tamaño retornado.
    - *CL_INVALID_EVENT.*
    - *CL_OUT_OF_RESOURCES.*
    - *CL_OUT_OF_HOST_MEMORY.*

Programando con OpenCL

- Liberación de recursos.
  - *cl_int clReleaseContext (cl_context context).*
  - *cl_int clReleaseCommandQueue (cl_command_queue command_queue).*
  - *cl_int clReleaseMemObject (cl_mem memobj).*
  - *cl_int clReleaseProgram (cl_program program).*
  - *cl_int clReleaseKernel (cl_kernel kernel).*
  - *cl_int clReleaseDevice (cl_device_id device).*
  - *cl_int clReleaseEvent (cl_event event).*
  - Códigos de error (*CL_SUCCESS* si éxito):
    - *CL_OUT_OF_RESOURCES.*
    - *CL_OUT_OF_HOST_MEMORY.*
    - *CL_INVALID_XXXX* → Donde *XXXX* es el tipo de objeto erróneo (depende de la función).

Programando con OpenCL

- Ejercicio.
  - Crear las siguientes funciones:
    - *cl_int CrearBuffer(cl_context context, cl_mem_flags flags, size_t size, void *host_ptr,cl_mem &buffer)*
    - *cl_int AsignarParametro(cl_kernel kernel, cl_uint arg_index, size_t arg_size,const void *arg_value)*
    - *cl_int EnviarBuffer(cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_write,size_t offset, size_t cb, const void *ptr, cl_uint num_events,const cl_event *event_wait_list, cl_event &event)*
    - *cl_int RecibirBuffer(cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_read,size_t offset, size_t cb, void *ptr, cl_uint num_events,const cl_event *event_wait_list, cl_event &event)*

Programando con OpenCL

- Ejercicio.
  - Crear las siguientes funciones:
    - *cl_int EjecutarKernel (cl_command_queue command_queue, cl_kernel kernel, cl_uint work_dim,const size_t *global_work_offset, const size_t *global_work_size,size_t *local_work_size, cl_uint num_events,const cl_event *event_wait_list, cl_event &event)*
    - *cl_int ObtenerTiempoEjecucionEvento(cl_event event, cl_ulong &tiempo)*
    - *cl_int ObtenerTiempoEjecucionEntreEventos(cl_event event_ini, cl_event event_fin, cl_ulong &tiempo)*

Programando con OpenCL

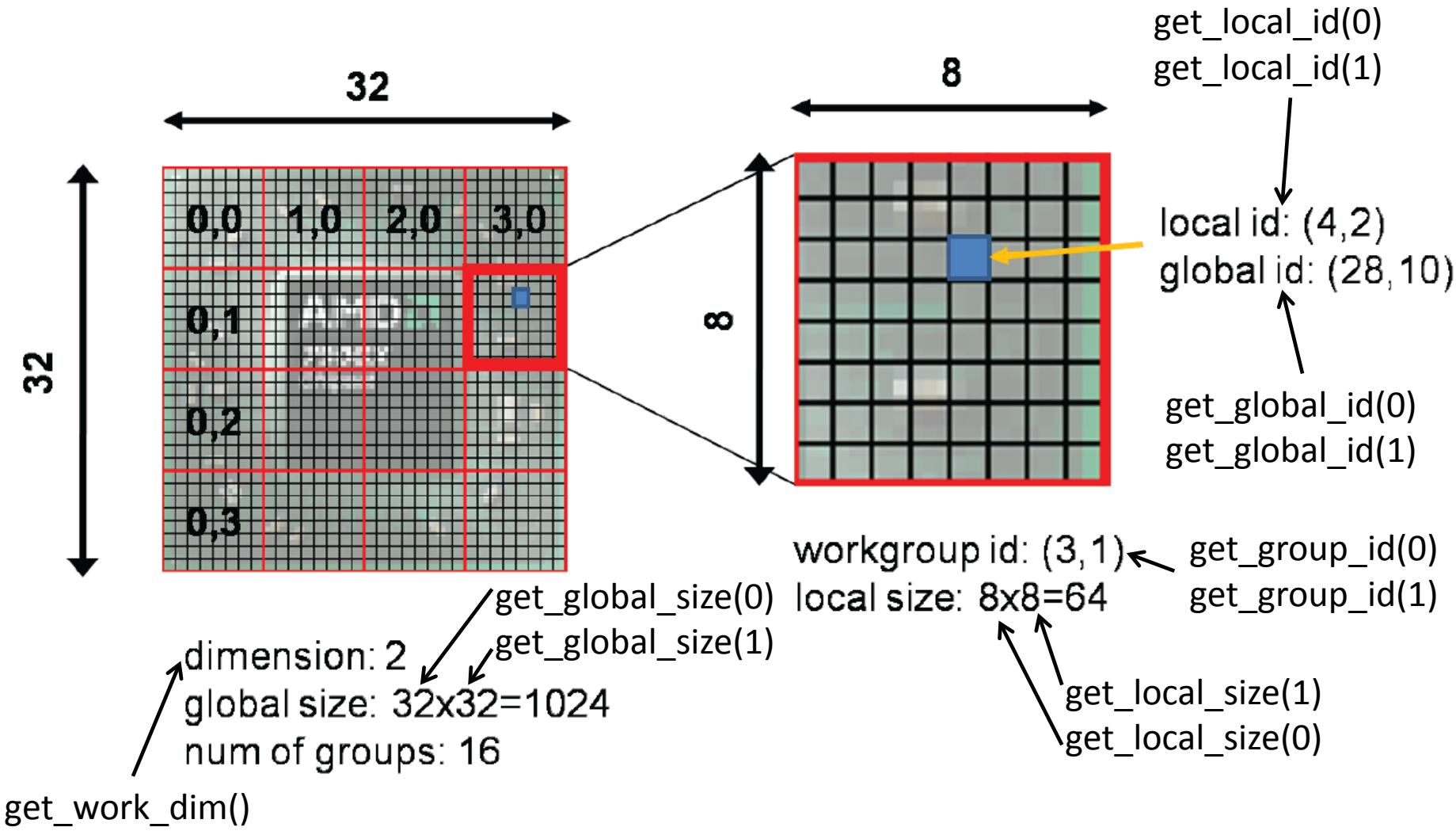
- Funciones útiles (en el kernel).
  - *uint get_work_dim()*
    - Devuelve el número de dimensiones del espacio del problema.
  - *size_t get_global_size(dimidx)*
    - Devuelve el número total de work-items para la dimensión *dimidx*.
  - *size_t get_local_size(dimidx)*
    - Devuelve el número de work-items locales dentro de un work-group para la dimensión *dimidx*.
  - *size_t get_global_id(dimidx)*
    - Devuelve el identificador global del work-item para la dimensión *dimidx.*
  - *size_t get_local_id(dimidx)*
    - Devuelve el identificador local del work-item (dentro de un work-group) para la dimensión *dimidx.*

Programando con OpenCL

- Funciones útiles (en el kernel).
  - *size_t get_num_groups(dimidx)*
    - Devuelve el número de work-groups para la dimensión *dimidx.*
  - *size_t get_group_id(dimidx)*
    - Devuelve el identificador del work-group para la dimensión *dimidx.*
  - Donde *dimidx*:
    - Identifica la dimensión sobre la que estamos pidiendo información.
    - 0..N-1, siendo N el número de dimensiones del problema.
    - Ejemplos:
      - N==1 → *dimidx* = 0.
      - N==2 → *dimidx* = 0 (columnas) y *dimidx* = 1 (filas).

Arquitectura OpenCL

- Funciones útiles.



Programando con OpenCL

- Gestión de memoria.
  - OpenCL especifica cuatro regiones de memoria accesibles por los work-items durante la ejecución del kernel:
    - Memoria global: más grande pero más lenta.
    - Memoria constante: solo lectura.
    - Memoria local: más rápida pero más pequeña que la global.
    - Memoria privada: la más rápida y pequeña; solo para variables locales del work-item.
  - Para alojar una variable en una región específica de memoria, se usan calificadores en su declaración:
    - __*global*: memoria global.
    - __*constant*: memoria constante.
    - __*local*: memoria local.
    - __*private*: memoria privada, es la opción por defecto en la declaración de variables si no se especifica calificador. Si una variable no cabe en la memoria privada, se declarará en la global.

Programando con OpenCL

- Ejercicio.
  - Ejecutar el kernel creado en el ejercicio anterior con N=100000000 con las siguientes opciones:
    - Creando los buffers con *CL_MEM_USE_HOST_PTR* → Mostrar el tiempo de ejecución del kernel.
    - Creando los buffers con *CL_MEM_COPY_HOST_PTR* (hay que hacer transferencia de datos Host-Dispositivo-Host) → Mostrar el tiempo de ejecución del kernel y los tiempos de transferencia de datos (mediremos por separado los tiempos de transferencia y cómputo).
  - Usando la mejor opción del caso anterior, crear un nuevo kernel para que cada work-item procese *N/num_elem_comp* datos del vector.
    - *Num_elem_comp* lo obtenemos mediante la función get_global_size(0).
    - *N* se lo pasaremos al kernel como un parámetro
    - Comparar el tiempo con el caso anterior.

Programando con OpenCL

- Ejercicio.
  - Probar los kernels del ejercicio anterior compilando el programa con la opción “-g” para desactivar la autovectorización automática.

Bibliografía

- Lee Howes and Aaftab Munshi, “The OpenCL Specification. Version: 2.0. Document Revision: 26”, Khronos OpenCL Working Group, 2014.
- AMD, “Introduction to OpenCL Programing. Training Guide”, 2010
- Benedict R. Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry, Dana Schaa, “Heterogeneous Computing with OpenCL. Revised OpenCL 1.2 Edition”, Morgan Kaufmann, 2013