



# Introducción a OpenCL



Jesús Rioja Bravo



# Índice

1.-Tecnologías de contacto y su funcionamiento

2.-Primeras aproximaciones y antecedentes de las tecnologías de contacto actuales

3.-Principales tecnologías de contacto

3.1.-NFC

3.2.-RFID

3.3.-Relaciones entre las distintas tecnologías de contacto

4.-Dispositivos que hacen uso de NFC

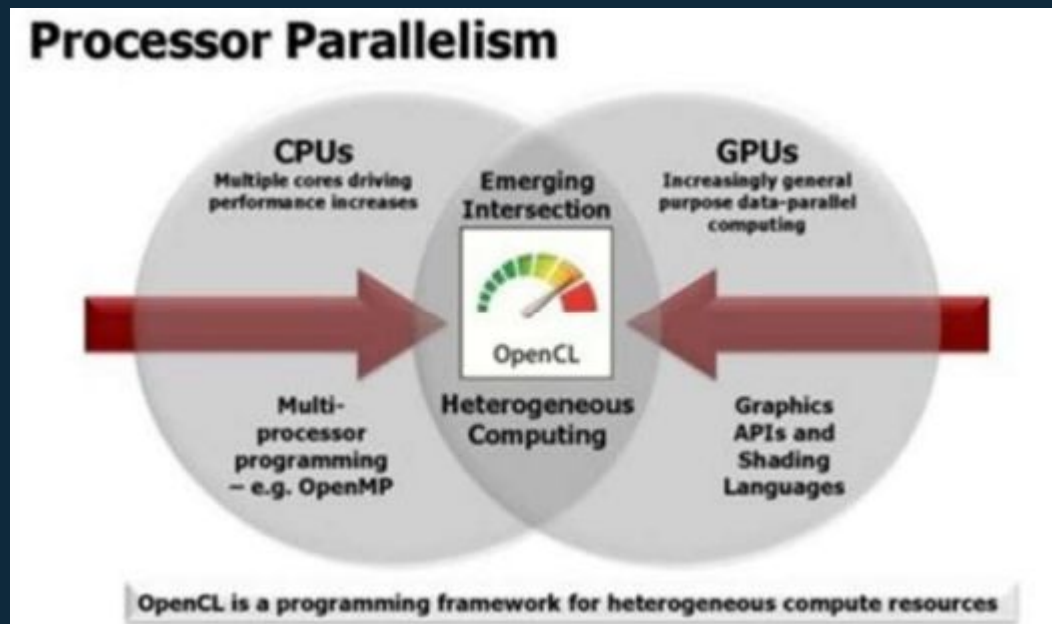
5.-Asociaciones implicadas más destacadas

6.-Bibliografía

7.-Cuestiones relativas al trabajo

# OpenCL – Introducción

Lenguaje que permite a los programas paralelizar tanto los núcleos de la CPU como la GPU





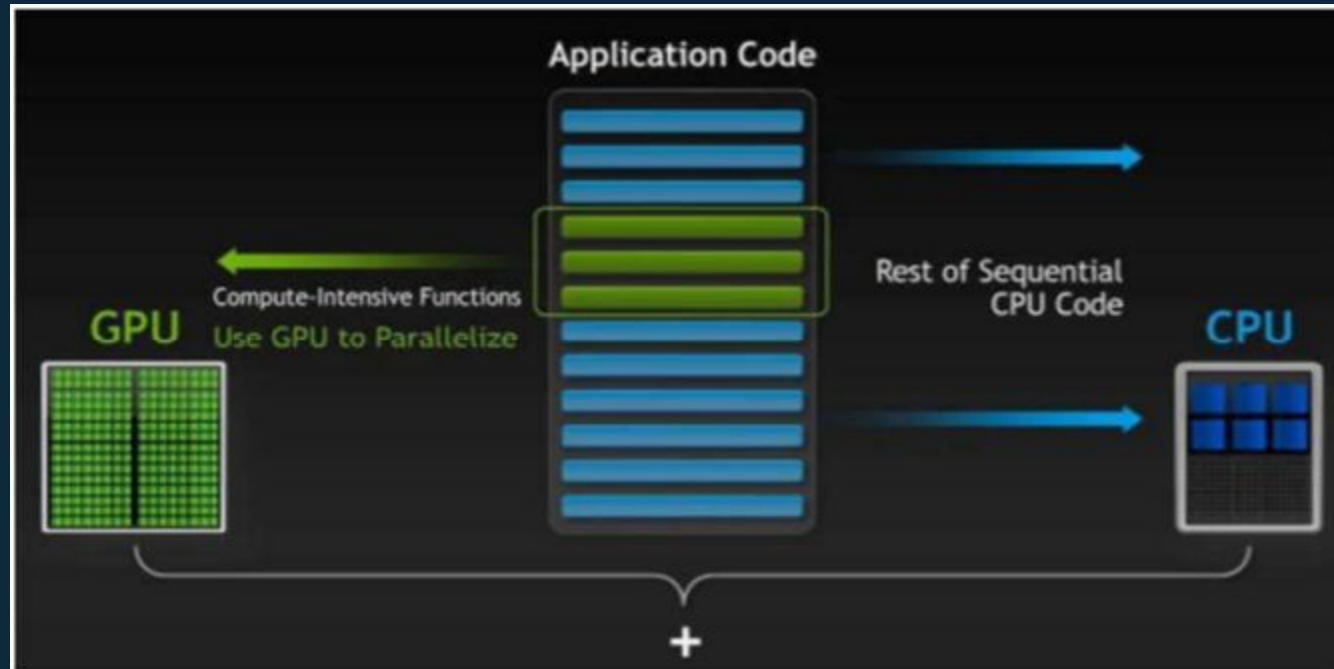
# OpenCL – Conceptos básicos

- Host: en este caso es nuestra CPU.
- Dispositivo o Device: en este caso es nuestra GPU integrada en el procesador.
- Kernel: código que se ejecuta en el dispositivo y que es llamado por el host.

EL host ( la CPU) se conecta con uno o más dispositivos mediante OpenCL.



# OpenCL – Conceptos básicos





# Instalación

- **Descargar SDK:**

- <https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/ocl-sdk/choose-download.html> (intel)
- <https://developer.nvidia.com/cuda-toolkit-40> (nvidia)

- **Descargar OpenCL tools for Visual Studio:**

- <https://dynamicinstaller.intel.com/system-studio/>





# OpenCL – Código Host

- Comprobación de plataformas disponibles

```
cl_int clGetPlatformIDs(cl_uint num_entries,  
                        cl_platform_id *platforms,  
                        cl_uint *num_platforms)
```

- **num\_entries**: número de plataformas que queremos que se metan en la variable platforms.
- **platforms**: variable donde se meterán las plataformas obtenidas.
- **num\_platforms**: número de plataformas que se obtienen al ejecutar la función.



# OpenCL – Código Host

- Selección de la plataforma a utilizar.

```
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**: plataforma de la que queremos obtener nuestro dispositivo.
- **device\_type**: tipo de dispositivo que queremos obtener.
- **num\_entries**: número de dispositivos que queremos que se metan en la variable devices.
- **devices**: aquí se guardarán los dispositivos obtenidos de la función.
- **num\_devices**: aquí se guardará el número de dispositivos disponibles dentro de nuestra plataforma.





# OpenCL – Código Host

- Creación de contexto

```
cl_context clCreateContext(cl_context_properties *properties,  
                           cl_uint num_devices,  
                           const cl_device_id *devices,  
                           void *pfn_notify (  
                           const char *errinfo,  
                           const void *private_info,  
                           size_t cb,  
                           void *user_data  
                           ),  
                           void *user_data,  
                           cl_int *errcode_ret)
```





# OpenCL – Código Host

- **properties:** una serie de propiedades que se deben de mandar al contexto, tienen tres parámetros, el primero debe de ser CL\_CONTEXT\_PLATFORM, el segundo el id de la plataforma que estemos usando y el tercero una descripción( opcional).
- **num\_devices:** el número de dispositivos que haya en la variable devices.
- **devices:** los dispositivos obtenidos anteriormente.
- **pfn\_notify:** una función que se utiliza para reportar errores en el contexto, puede ser NULL.
- **user\_data:** son los datos de usuario que se pasan cuando pfn\_notify no es nula. También puede ser NULL.
- **errcode\_ret:** devuelve un código de error en caso de haberlo, si se pasa como NULL no devolverá nada.





# OpenCL – Código Host

- Definición cola de comandos

```
cl_command_queue clCreateCommandQueue(cl_context context,  
                                       cl_device_id device,  
                                       cl_command_queue_properties properties,  
                                       cl_int *errcode_ret)
```

- **context**: se refiere al contexto que hemos creado anteriormente.
- **device**: dispositivo en el que queremos implementar la cola.
- **properties**: una lista de propiedades que se pasan a la cola, puede tener dos valores, de caso contrario se considerará el parámetro como no válido, estos son `CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE` y `CL_QUEUE_PROFILING_ENABLE`.
- **errcode\_ret**: devuelve un código de error en caso de haberlo, si se pasa como NULL no devolverá nada.



# OpenCL – Código Host

- Creamos objeto de programa

```
cl_program clCreateProgramWithSource (cl_context context,  
                                     cl_uint count,  
                                     const char **strings,  
                                     const size_t *lengths,  
                                     cl_int *errcode_ret)
```

- **context**: contexto creado anteriormente:
- **count**: número de elementos que tendrá la variable strings.
- **strings**: es un array que contiene el código de la función que vamos a cargar.
- **lengths**: es un array con el tamaño de cada string del array strings.
- **errcode\_ret**: devuelve un código de error en caso de haberlo, si se pasa como NULL no devolverá nada.

# OpenCL – Código Host

- Creación del ejecutable

```
cl_int clBuildProgram (cl_program program,  
                      cl_uint num_devices,  
                      const cl_device_id *device_list,  
                      const char *options,  
                      void (*pfn_notify)(cl_program, void *user_data),  
                      void *user_data)
```

- **program**: programa creado en el paso anterior.
- **num\_devices**: número de dispositivos en la variable device\_list.
- **device\_list**: lista de dispositivos que van a ejecutar el programa.
- **options**: es un puntero a un string que describe las opciones de compilación que se van a usar para construir el ejecutable.
- **pfn\_notify**: es una función de notificación que se va a ejecutar cuando el programa sea compilado.
- **user\_data**: datos de usuario que se pasan a pfn\_notify.





# OpenCL – Código Host

- Definición de buffers de memoria

```
cl_mem clCreateBuffer (cl_context context,  
                        cl_mem_flags flags,  
                        size_t size,  
                        void *host_ptr,  
                        cl_int *errcode_ret)
```





# OpenCL – Código Host

- **context:** contexto que hemos creado anteriormente.
- **flags:** es un campo que se utiliza para especificar la localización y el uso que la memoria debe de dar al buffer, puede tener los siguientes campos: CL\_MEM\_READ\_WRITE, CL\_MEM\_WRITE\_ONLY, CL\_MEM\_READ\_ONLY, CL\_MEM\_USE\_HOST\_PTR, CL\_MEM\_ALLOC\_HOST\_PTR y CL\_MEM\_COPY\_HOST\_PTR.
- **size:** tamaño en bytes del buffer.
- **host\_ptr:** un puntero a la localización del buffer una vez esté ubicado en memoria.
- **errcode\_ret:** devuelve un código de error en caso de haberlo, si se pasa como NULL no devolverá nada.





# OpenCL – Código Host

- Asignación de datos a las variables del Kernel

```
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_in_wait_list,  
                             const cl_event *event_wait_list,  
                             cl_event *event)
```







# OpenCL – Código Host

- **command\_queue**: cola de comandos creada anteriormente.
- **buffer**: buffer creado anteriormente.
- **blocking\_write**: indica si las operaciones de escritura son bloqueantes o no bloqueantes.
- **offset**: el offset en bytes del buffer.
- **cb**: el tamaño en bytes de los datos que están siendo escritos.
- **ptr**: un puntero al buffer en la memoria del host donde están ubicados los datos.
- **num\_events\_in\_wait\_list** y **event\_wait\_list**: especifican los eventos que tienen que ocurrir antes de que un comando de la cola pueda ser ejecutado.
- **event**: devuelve un evento que identifica un comando de escritura en particular.





# OpenCL – Código Host

- Creación del kernel

```
cl_kernel clCreateKernel (cl_program program,  
                           const char *kernel_name,  
                           cl_int *errcode_ret)
```

- **program**: programa creado anteriormente.
- **kernel\_name**: nombre de la función kernel que queremos ejecutar.
- **errcode\_ret**: devuelve un código de error en caso de haberlo, si se pasa como NULL no devolverá nada.






# OpenCL – Código Host

- Asignación de argumentos del Kernel.

```
cl_int clSetKernelArg (cl_kernel kernel,  
                       cl_uint arg_index,  
                       size_t arg_size,  
                       const void *arg_value)
```

- **kernel**: kernel creado anteriormente.
  - **arg\_index**: el índice del argumento al que nos estamos refiriendo.
  - **arg\_size**: tamaño del argumento al que nos estamos refiriendo.
  - **arg\_value**: valor del argumento al que nos estamos refiriendo.
- 



# OpenCL – Código Host

- Ejecutamos 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_in_wait_list,  
                                const cl_event *event_wait_list,  
                                cl_event *event)
```





# OpenCL – Código Host

- **command\_queue**: la cola de comandos creada anteriormente.
- **kernel**: el kernel creado anteriormente.
- **work\_dim**: el número de dimensiones que tendrá nuestro rango NDimensional.
- **global\_work\_offset**: en la actual versión de OpenCL esta variable debe de ser nula.
- **global\_work\_size**: se refiere al número de workers que vamos a utilizar para ejecutar nuestra función kernel.
- **local\_work\_size**: se refiere al número de grupos de trabajo que vamos a declarar, este tamaño lo escogeremos en función del número de workers que haya.
- **num\_events\_in\_wait\_list** y **event\_wait\_list**: especifican los eventos que tienen que ocurrir antes de que un comando de la cola pueda ser ejecutado.
- **event**: devuelve un evento que identifica un comando de escritura en particular.





# OpenCL – Código Host

- Por último leemos el resultado del buffer

```
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_in_wait_list,  
                             const cl_event *event_wait_list,  
                             cl_event *event)
```





# OpenCL – Código Host

- **command\_queue**: cola de comandos creada anteriormente.
- **buffer**: buffer creado anteriormente.
- **blocking\_read**: indica si la operación de lectura es bloqueante o no bloqueante.
- **offset**: el offset en bytes del buffer que se está leyendo.
- **cb**: el tamaño en bytes de los datos que se están leyendo.
- **ptr**: puntero al buffer en la posición de memoria del host de donde se están leyendo los datos.
- **num\_events\_in\_wait\_list** y **event\_wait\_list**: especifican los eventos que tienen que ocurrir antes de que un comando de la cola pueda ser ejecutado.
- **event**: devuelve un evento que identifica un comando de escritura en particular.



# OpenCL – Código Kernel

```
__kernel void MatrixMultSimple( __global float* mC,  
                                __global float* mA,  
                                __global float* mB,  
                                const int Mdim,  
                                const int Ndim,  
                                const int Pdim){  
  
    int k;  
    int i = get_global_id(0);  
    int j = get_global_id(1);  
    float sum = 0;  
    for (k = 0; k < Pdim; k++){  
        sum += mA[i * Ndim + k] * mB[k * Pdim + j];  
        mC[i * Ndim + j] = sum;  
    }  
}
```





# OpenCL – Multiplicación de matrices

- Instalamos OpenCL.
- Ejecutamos el código de Multiplicación de matrices con distintos tamaños.
- Registramos los distintos tiempos obtenidos.
- Comparación de tiempos con código secuencial.

