BOLETÍN OPENCL

Alumnos: Javier Meliá Sevilla y Marco Moroni

Ejercicio 1

a) La salida del programa DeviceInfo es:

```
[tpp_jamese@nowherman Ej01]$ ./DeviceInfo

Number of OpenCL platforms: 2

-------------------------
Platform: NVIDIA CUDA
Vendor: NVIDIA Corporation
Version: OpenCL 3.0 CUDA 11.7.101
Number of devices: 4
        -------------------------
                Name: Tesla P100-SXM2-16GB
                Version: OpenCL C 1.2
                Max. Compute Units: 56
                Local Memory Size: 48 KB
                Global Memory Size: 16280 MB
                Max Alloc Size: 4070 MB
                Max Work-group Total Size: 1024
                Max Work-group Dims: ( 1024 1024 64 )
        -------------------------
        -------------------------
                Name: Tesla P100-SXM2-16GB
                Version: OpenCL C 1.2
                Max. Compute Units: 56
                Local Memory Size: 48 KB
                Global Memory Size: 16280 MB
                Max Alloc Size: 4070 MB
                Max Work-group Total Size: 1024
                Max Work-group Dims: ( 1024 1024 64 )
        -------------------------
        -------------------------
                Name: Tesla P100-SXM2-16GB
                Version: OpenCL C 1.2
                Max. Compute Units: 56
                Local Memory Size: 48 KB
                Global Memory Size: 16280 MB
                Max Alloc Size: 4070 MB
                Max Work-group Total Size: 1024
                Max Work-group Dims: ( 1024 1024 64 )
        -------------------------
        -------------------------
                Name: Tesla P100-SXM2-16GB
                Version: OpenCL C 1.2
                Max. Compute Units: 56
                Local Memory Size: 48 KB
                Global Memory Size: 16280 MB
                Max Alloc Size: 4070 MB
                Max Work-group Total Size: 1024
```

```
-------------------------
Platform: Intel(R) OpenCL
Vendor: Intel(R) Corporation
Version: OpenCL 3.0 LINUX
Number of devices: 1
        -------------------------
                Name: Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
                Version: OpenCL C 3.0
                Max. Compute Units: 80
                Local Memory Size: 32 KB
                Global Memory Size: 515879 MB
                Max Alloc Size: 257939 MB
                Max Work-group Total Size: 8192
                Max Work-group Dims: ( 8192 8192 8192 )
        -------------------------

-------------------------
[tpp_jamese@nowherman Ej01]$
```

b) Para obtener la lista de plataformas es la función:
clGetPlatformIDs(num_entries, platforms, numplatforms)

Y para obtener la lista de dispositivos es:
clGetDeviceIDs(platform, device_type,num_entries,devices,num_devices)

Se llaman dos veces a ambas funciones porque la primera llamada sirve para contar el numero de plataformas y el de dispositivos en la plataforma y la segunda llamada es para crear una lista por ids de las plataformas y de los dispositivos.

c) Para obtener información de las plataformas la función utilizada es:
clGetPlatformInfo(platform, param_name, param_value_size, value_parameter, param_value_size_ret) donde:

platform:
El ID de plataforma devuelto por clGetPlatformIDs o puede ser NULL. Si platform es NULL, el comportamiento está definido por la implementación.
param_name:
Una constante de enumeración que identifica la información de la plataforma que se consulta.
param_value_size:
Especifica el tamaño en bytes de la memoria a la que apunta param_value. Este tamaño en bytes debe ser mayor o igual que el tamaño del tipo de valor devuelto especificado en la siguiente tabla.
param_value:
Un puntero a la ubicación de la memoria donde se devolverán los valores apropiados para un determinado . Los valores aceptables param_value se enumeran en la siguiente tabla. Si param_value es NULL, se ignora.
param_value_size_ret:
Devuelve el tamaño real en bytes de los datos consultados por param_value. Si param_value_size_ret es NULL, se ignora.

Para obtener información de los dispositivos la función utilizada es:
clGetDeviceInfo(device, param_name, param_value_size, param_value, param_value_size_ret) y los parametrons significan:

device:
Hace referencia al dispositivo devuelto por clGetDeviceIDs .
param_name:
Una constante de enumeración que identifica la información del dispositivo que se consulta. Puede ser uno de los valores especificados en la siguiente tabla.
param_value:
Un puntero a la ubicación de la memoria donde se devolverán los valores apropiados para un determinado param_name. Si param_value es NULL, se ignora.
param_value_size:

Especifica el tamaño en bytes de la memoria a la que apunta param_value. Este tamaño en bytes debe ser mayor o igual que el tamaño del tipo de valor devuelto especificado en la siguiente tabla.

param_value_size_ret:

Devuelve el tamaño real en bytes de los datos consultados por param_value. Si param_value_size_ret es NULL, se ignora.

d) Para mostrar la maxima frecuencia de reloj de cada dispositivo seria de la siguiente manera:

```
 // Get maximum clock frequency
err = clGetDeviceInfo(device[j], CL_DEVICE_MAX_CLOCK_FREQUENCY,
sizeof(cl_uint), &num, NULL);
checkError(err, "Getting device max clock frequency");
printf("\t\tMax Clock Frequency: %lu MB\n", num);
```

Ejercicio 2

a) Las principales etapas del programa vadd.c con sus llamadas a la API serian:

1. Definir el entorno o la plataforma y las llamadas:

```
err = clGetPlatformIDs(1, &platform, &numPlatforms);

err = clGetDeviceIDs(platform, CL_DEVICE_TYPE_CPU, 1, &device, NULL);

context = clCreateContext(0, 1, &device, NULL, NULL, &err);

commands = clCreateCommandQueueWithProperties(context, device, 0, &err);
```

2. Construir el programa y las llamadas:

```
program = clCreateProgramWithSource(context, 1, (const char **) &programSource,
NULL, &err);

err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
```

3. Crear los objetos de Memoria y las llamadas:

```
d_a = clCreateBuffer(context, CL_MEM_READ_ONLY, LENGTH * sizeof(float), NULL, &err);
d_b = clCreateBuffer(context, CL_MEM_READ_ONLY, LENGTH * sizeof(float), NULL, &err);
d_c = clCreateBuffer(context, CL_MEM_WRITE_ONLY, LENGTH * sizeof(float), NULL, &err);
```

4. Crear y parametrizar el Kernel y las llamadas:

```
kernel = clCreateKernel(program, "vadd", &err);
err = clSetKernelArg(kernel, 0, sizeof(cl_mem), &d_a);
err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &d_b);
err |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &d_c);
err |= clSetKernelArg(kernel, 3, sizeof(unsigned int), &LENGTH);
```

5. Lanza órdenes… transfiere objetos de memoria, ejecuta kernels y recoge resultados.

```
err = clEnqueueWriteBuffer(commands, d_a, CL_FALSE, 0, LENGTH * sizeof(float), h_a, 0, NULL, NULL);
err = clEnqueueWriteBuffer(commands, d_b, CL_FALSE, 0, LENGTH * sizeof(float), h_b, 0, NULL, NULL);
err = clEnqueueNDRangeKernel(commands, kernel, 1, NULL, &LENGTH, NULL, 0, NULL, NULL);
err = clEnqueueReadBuffer( commands, d_c, CL_TRUE, 0, LENGTH * sizeof(float), h_c, 0, NULL, NULL );
```

b) Elegimos el tipo de dispositivo con la variable:
#define DEVICE_TYPE CL_DEVICE_TYPE_ALL, que indica que puede ser cualquier tipo de dispositivo ya sea CPU O GPU el que la maquina elija.

c) Con el método checkError que se ha construido en el archivo "cl_utils.h" para simplificar el código y eso nos devuelve si hay algún error en las llamadas a la API.

d) El tiempo se mide con el método wtime() midiendo desde antes de ejecutar el kernel hasta que esperamos que acaben todas las tareas de este mismo. Y la forma de comprobar que ha acabado es con el método clFinish(command_queue) que espera a que finalizan todas las tareas.

e) El kernel a ejecutar esta en otro archivo llamado "vadd.cl" y lo que hacemos es crear una variable char de la siguiente forma: "char * filename="vadd.cl";"
Y luego construimos el programa pasándoselo como una cadena de caracteres constantes para que lo lea de la siguiente manera:

```
 char *kernelSource = getKernelSource(filename);
 program = clCreateProgramWithSource(context, 1, (const char **)
&kernelSource, NULL, &err);
```

Ejercicio 3

Tanto el apartado a y b están adjuntados como ficheros para revisar el código.

Ejercicio 4

a) **parseArguments**: sirve para indicar cuando ejecutas el ejecutable para decir si quieres utilizar la CPU o GPU con el argumento –device o devolver la lista dispositivos con –list.
getDeviceList: Se utiliza para conseguir la lista de los dispositivos de la plataforma disponibles.
getDeviceName: Devuelve el nombre del dispositivo que se está utilizando.

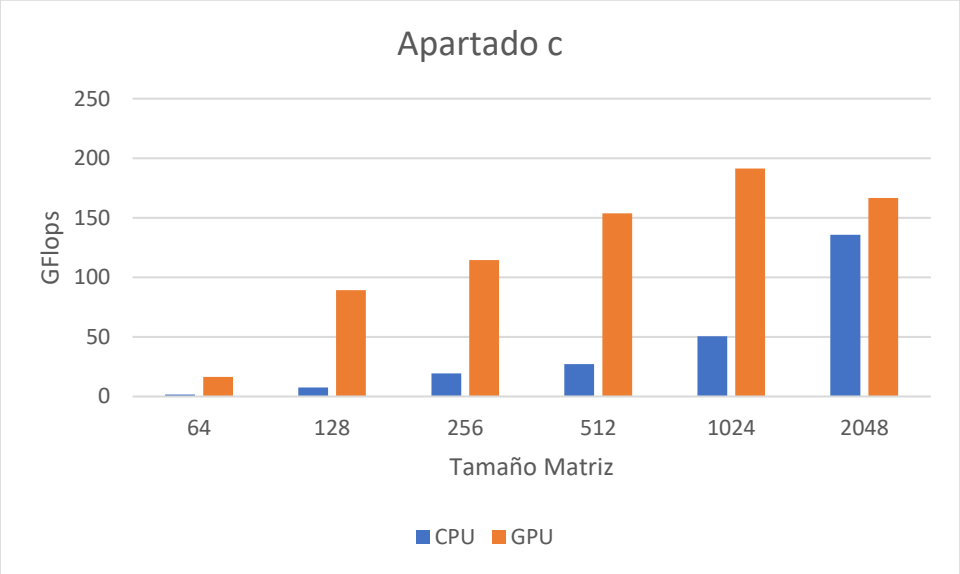
b) Las matrices se transfieren de la entrada al dispositivo cuando se crea el buffer añadiéndole la opción directamente CL_MEM_COPY_HOST_PTR, de la siguiente manera:

```
d_a = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
sizeof(float) * size, h_A, &err);
```

La modificación se puede observar en el código adjunto con los códigos correspondientes, pero se usa clEnqueueWriteBuffer

c) Los resultados son:

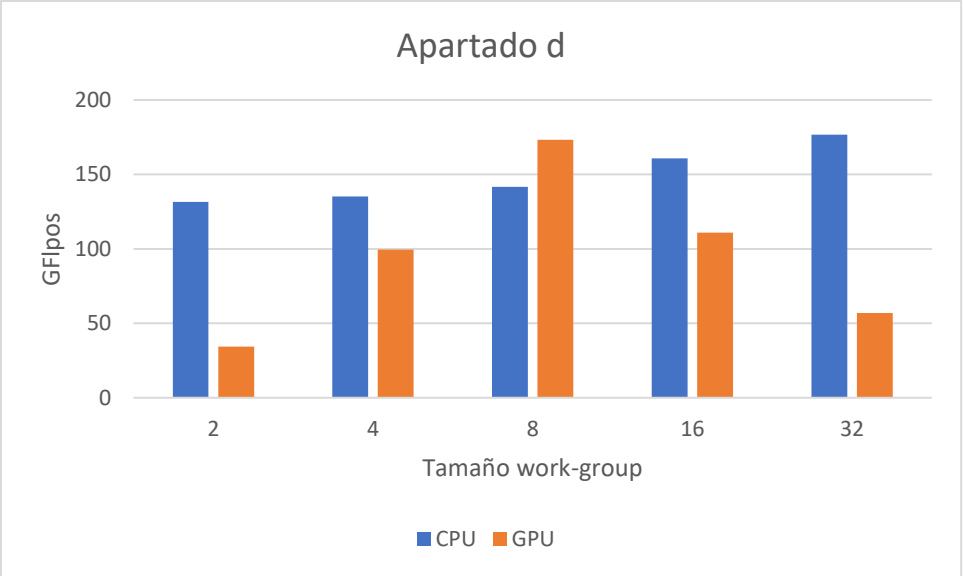
Tamaño Matriz	CPU	GPU
64	1,72	16,38
128	7,65	89,24
256	19,27	114,52
512	27,11	153,83
1024	50,42	191,24
2048	135,63	166,57



Podemos observar que con la GPU da muchos mejor resultados que con la CPU y conforme se aumenta el tamaño de la matriz también mejoras las prestaciones en ambos casos.

d) Los resultados son:

work-group	CPU	GPU
2	131,45	34,43
4	135,28	99,46
8	141,73	173,34
16	160,68	110,84
32	176,6	57,02



Como podemos observar para un tamaño de matriz constante, dependiendo del tamaño del work-group en la CPU no modifica mucho las prestaciones totales mientras que en la GPU si que es determinante elegir un buen tamaño para sacar el máximo de las prestaciones posibles.

e) Tanto si intentas usar un tamaño de work-group 5 o 64 da un error de “CL_INVALID_WORK_GROUP_SIZE”. En el caso del work-group de 5 es porque no es divisible por el tamaño de la matriz. Y en el caso de 64 porque el “CL_DEVICE_MAX_WORK_GROUP_SIZE” es menor que 64.