MANEJO DE MODULOS EN CUDA

Netz Romero

Declarando funciones

- `__global__` - función llamada desde el host, y es ejecutada en el device.
- `__device__` - función llamada desde el device y es ejecutada en el device.
- `__host__` - función llamada desde el host y es ejecutada en el host.

	Executed on the:	Only callable from the:
`__device__ float DeviceFunc()`	device	device
`__global__ void KernelFunc()`	device	host
`__host__ float HostFunc()`	host	host

Llamando al device

```
#include <stdio.h>

__device__ int get_global_index(void)
{
  return blockIdx.x * blockDim.x + threadIdx.x;
}

__global__ void kernel(int *array)
{
  int index = get_global_index();
  array[index] = get_global_index();
}
```

```
int main(void) {
  int num_elements = 256;
  int block_size = 128;
  int grid_size = 2;
  int size = num_elements * sizeof(int);
  int *device_array;
  int *host_array = (int*)malloc(size);
  cudaMalloc((void**)&device_array, size);
  kernel<<<grid_size,block_size>>>(device_array);
  cudaMemcpy(host_array, device_array, size,
                                 cudaMemcpyDeviceToHost);
  printf("kernel results:\n");
  for(int i = 0; i < num_elements; ++i) {
    printf("%d ", host_array[i]);
  }
  printf("\n\n");
  free(host_array);
  cudaFree(device_array);
  return 0;
}
```

Llamando dos veces al kernel

```
#include <stdio.h>

__device__ int get_constant(void) {
  return 1;
}
__global__ void kernel1(int *array) {
  int index = blockIdx.x * blockDim.x + threadIdx.x;
  array[index] = get_constant();
}
__global__ void kernel2(int *array) {
  int index = blockIdx.x * blockDim.x + threadIdx.x;
  array[index] = get_constant() * 2;
}
```

```
int main(void) {
 int num_elements = 8;
 int size = num_elements * sizeof(int);
 int host_array[8];
 int *device_array;
 cudaMalloc((void**)&device_array, size);
 kernel1<<<2, 4>>>(device_array);
 cudaMemcpy(host_array, device_array, size,
                              cudaMemcpyDeviceToHost);
 for(int i = 0; i < num_elements; ++i)
     printf("k1 %d   ", host_array[i]);
 printf("\n\n");
 kernel2<<<2, 4>>>(device_array);
 cudaMemcpy(host_array, device_array, size,
                              cudaMemcpyDeviceToHost);
 for(int i = 0; i < num_elements; ++i)
    printf("k2 %d  ", host_array[i]);
 printf("\n\n");
 cudaFree(device_array);
 return 0;
}
```

Recursividad

```
#include <stdio.h>

__device__ int recursiva(int n)
{
 if(n == 0)
   return 0;
 else
   return recursiva(n -1);
}

__global__ void kernel(int *array)
{
  int index = blockIdx.x * blockDim.x + threadIdx.x;;
  array[index] = recursiva(index);
}
```

- Compilando el programa de recursividad

```
$ nvcc deviceRecursivo.cu -o deviceRecursivo4
./deviceRecursivo.cu(8): Error: Recursive function
call is not supported yet: recursiva(int)
```

Referencias

- Sito de NVIDIA, https://developer.nvidia.com/
- CUDA by Examples, NVIDIA
- Memoria compartida, http://devblogs.nvidia.com/parallelforall/using-shared-memory-cuda-cc/