CUDA

September 10, 2023

Ramasamy Kandasamy

DeviceManagement

- `cudaDeviceSynchronize()`:
  Tells CPU to wait for GPU code execution.
- `cudaGetDeviceProperty(cudaDeviceProp* prop, int Device)`
  `Device`: Device number, eg: `0`. Store device property in the structure pointed by `prop`.

Memory Management

- `cudaMalloc(void** (&dp), size_t sz)`:
  Allocates a memory of size `sz` and store the pointer value at `dp`, where dp is a pointer.
- `cudaMemset(void* da, char v, size_t sz)`:
  This function copies the byte-sized value `v`, starting from the memory location `da` **byte-by-byte** upto `sz` bytes.
  textcolorredIMPORTANT: The following will give unexpected results:
  `cudaMemset(da, 0, sizeof(int))`
- `cudaMemcpy(void* da, void* ha, size_t sz, cudaMemcpyHostToDevice)`
- `cudaMemcpy(void* ha, void* da, size_t sz, cudaMemcpyDeviceToHost)`

Kernels

Types of functions

- `__global__`: Can be called from CPU/GPU(?), runs on GPU.
- `__device__`: Can be called only from GPU, runs on GPU.
- `__host__`: Can be called only from CPU, runs on CPU.

If a function is decorated by both `__host__` and `__device__`, then two instances of the function are created, one for the device and another for host.
However if a function in device is to be called by host it should be decorated by `__global__`.
NOTE: Functions decorated by `__global__` should only return `void`.

Grids and Blocks

```
dkernel<<<1, 1>>>(); // One block with one thread.
dkernel<<<1, M>>>(); // One block with M threads.
dkernel<<<N, 1>>>(); // N blocks with one thread each.
dkernel<<<N, M>>>(); // N blocks with M threads each.
```

Grids contain **blocks** which contain **threads.**.

Advanced

```
dim3 block(x, y); // Contains x * y threads.
dim3 block(x, y, z); // Contains x * y * z threads.
dim3 grid(a, b);  // Contains x * y blocks.
dim3 grid(a, b, c);  // Contains x * y * z blocks.
dkernel<<<grid, block>>>(da);
```

Here, the above kernel will run a * b * c * x * y * z threads.

Attributes

- `threadIdx.x`, `threadIdx.y` and `threadIdx.z`
- `blockIdx.x`, `blockIdx.y` and `blockIdx.z`
- `blockDim.x`, `blockDim.y` and `blockDim.z`.
- `gridDim.x`, `gridDim.y` and `gridDim.z`.

Errors

Detecting error in CUDA

Ref:
https://stackoverflow.com/a/14038590/5607735

```
#define gpuErrchk(ans){gpuAssert((ans), __FILE__, __LINE__);}
inline void gpuAssert(cudaError_t code, char *file,
            int line, bool abort=true)
{
   if (code != cudaSuccess)
   {
      fprintf(stderr,"GPUassert: %s %s %d\n",
       cudaGetErrorString(code), file, line);
      if (abort) exit(code);
   }
}
```

Notable errors in CUDA

1. Incomplete output

```
__global__ void per_row_AB_kernel(long int m, long int n){
    long int K = blockIdx.x * 1024 + threadIdx.x;
    if(K >= (m * n)) return;
    printf("%ld \n", K);
}

int main(){
    long int m, n;
    scanf("%ld", &m);
    scanf("%ld", &n);

    dim3 grid_1((m * n)/1024 + 1, 1, 1);
    dim3 block_1(1024, 1, 1);

    per_row_AB_kernel<<<grid_1, block_1>>>(m, n);
    cudaDeviceSynchronize();
}
```

In the above code for large values of `m` and `n` not all values of `K` will be printed. This is because there is limited space for in output buffer. It is intended for small scale debug style output not large scale output.
Source: https://stackoverflow.com/a/15421935/5607735