Guillermo Martín-Coello Juárez

Grupo 149

Computación de Altas Prestaciones

**Exercise 3: GPU**

**DATA**

N=8192

**Arq**

16 SM

**SM**

8 blocks

24 Warps

**Block**

512 threads

**Warp**

32 threads

**Part A**

N = 8192, for this implementation we need the same number of threads as N, 8192 threads.

# Warps = 8192 / 32 = 256 Warps.

We have 16 SM, so the amount of warps per SM is 256 /16 = 16 warps/SM. This means that the utilization of warps per SM is not optimal (24 warps).

**Part B**

To achieve this, we decrease the amount of SM to the minimum necessary, so they all run at full capacity. This is 256 / 24 = 10.7, so we need 11 SM.

**Part C**

\_\_shared\_\_ float partialSum[]

unsigned int t = threadIdx.x;

for (unsigned int stride = 1; stride < blockDim.x; stride \*= 2)

{

\_\_syncthreads();

if (t % (2\*stride) == 0)

partialSum[t] += partialSum[t+stride];

}

As we can observe in the code, the only used threads are the even ones. These add the vector numbers and save them in shared memory. This is not very efficient because half of the threads are not being used.

**Part D**

An alternate schema to solve the divergence problem of odd and even threads is to use only N/2 threads.

We could achieve this via the following code:

\_\_shared\_\_ float partialSum[]

unsigned int t = threadIdx.x;

for (int stride = blockDim.x; stride > 1; stride >> 1){

\_\_syncthreads();

if (t < stride)

partialSum[t] += partialSum[t + stride];

}