Assignment 4 : GPU programming

Simon Wicky
Samuel Chassot
Group 37

December 21, 2018

1 Implementation and optimization

The first thing to notice is that all threads have to synchronize after each iterations to update the heat value of the grid. Since CUDA has no way to achieve synchronization between threads blocks, we have to issue a kernel for each iteration, to ensure no data race.

During an iteration's execution no synchronization is required. Each thread is assigned to a single cell of the grid and use the neighbouring cells to compute its value. Since the grid is symmetric we compute only a fourth of the grid and the values are then copied to the three other quarters of the grid. The maximum number of threads per block being 1024, we can easily create one block per line for approximately 500 blocks and 500 threads per block. In this configuration, the coordinates of a cell is simply (*blockId.x*, *threadId.x*). This means that each line of the grid corresponds to a threads block and each cell (column) corresponds to a thread. This let us do computation for grid with a maximum size of 2048×2048 cells.

The rest of the implementation is pretty simple. It just consists of allocating memory on the GPU, copying the input array, running the algorithm and copy back the array. The pointer swap is made by the CPU after each kernel.

2 Effect of optimization

The optimization of cutting the grid in four was pretty effective. The improvement in execution time is displayed in the table 1

(side, iterations)	(100,10)	(100,1000)	(1000, 100)	(1000,10000)
GPU Non optimized	0.0001668	0.007733	0.02362	2.356
GPU Optimized	0.0001658	0.007027	0.008389	0.8255

Table 1: Computation time is seconds

(side, iterations)	(100,10)	(100,1000)	(1000, 100)	(1000,10000)
CPU	0.000144	0.02107	0.1486	21.68
CPU optimized 16 threads	0.000432	0.01417	0.01019	0.7692
GPU Host to Device	$6.899 * 10^{-5}$	$5.891 * 10^{-}5$	0.002272	0.00279
GPU Computation	0.0001658	0.007027	0.008389	0.8255
GPU Device to Host	$5.069 * 10^{-5}$	$5.078 * 10^{-5}$	0.002224	0.002226
GPU Total	0.09326	0.1005	0.1054	0.9155

Table 2: Time for operations in seconds

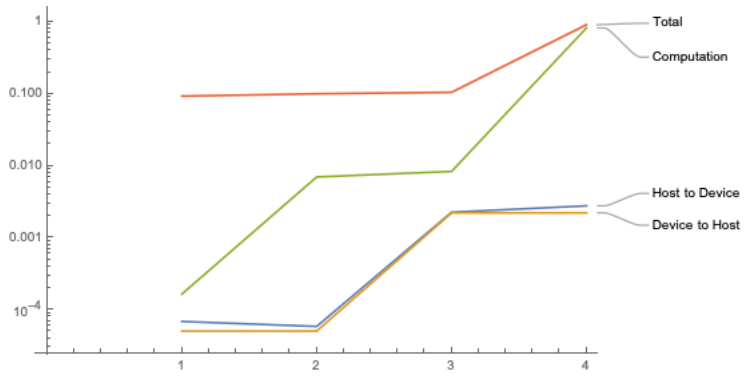

Figure 1: Time graph

3 Measures

The time taken by each operation is indicated in table 2 including the CPU Baseline.
As expected the copy time increase more or less linearly with the size of the grid and is independent of the number of iterations.

4 Analysis

The optimization we made is substantially faster mainly because the GPU has less data to fetch for each kernel, making the threads ready faster.
When the grid is small the CPU is faster than the GPU because some time is lost for copying the data. This changes when the grid and the number of iterations gets bigger because the copy takes a less significant part of the total time.