Parallel programmimg exam: Jacobi.c

Nicole Orzan

May 2, 2018

In this exercise I had to parallelize the given code jacobi.c with both OpenMP and CUDA.

In figure 1 you can see two histograms about the execution times obtained running the OpenMP and CUDA codes on a matrix of size 1200 on a node of the Ulysses cluster; for the OpenMP version I ran the code using 1, 5, 10 and 20 cores; each column is the result of a sample of 12 measures. Each measured times does not include the initialization of the matrices nether the allocation of the needed space. In the computation of the time for the CUDA version I included also the time spent by the data transfer on the GPU device and back.
We can see how the CUDA version of the code is slower that the OMP one when we use 5 cores or more; this is due to the fact that this version of the code simply reads the matrix and directly writes it to the device without any optimization, and this operation slows down the performance of the whole code; besides, the matrix size used here is not so big, and this does not allow to take all the advantages of the GPU device.

I decided to check the trend of the timed spent by the OpenMP code changing the number of cores on a Cosilt node (24 cores). In figure figure 2 you can see the result obtained; clearly the code is scaling well increasing the number of nodes.

Just out of curiosity I made a little check for the time implied by the code to terminate using 16 or 128 threads for two different matrix sizes (1280 and 12800); the results are listed in table 1. We can see how for a little size matrix the difference is irrelevant, while for a bigger size matrix, using 128 threads the code is faster.

In figure 3 we can see the trend of the CUDA and OpenMP codes increasing the size of the matrix (fixing the number of iterations). We can clearly see that the CUDA code implies lesser time to compute a matrix of size 1280 rather then a matrix of size 1200 because of the overhead given by the copy on the device.
Above all, we can see that for a matrix of size 12800 the CUDA code becomes finally more efficient than the OpenMP one!

	16 threads	128 threads
1280	0.083	0.0414
12800	8.123	3.812

Table 1: Times implied by the code to run on different matrix sizes for different number of threads.

In figure 4 you can see the final images obtained plotting the results of the computation with the CUDA code for matrices of size 1200, using different numbers of iterations.

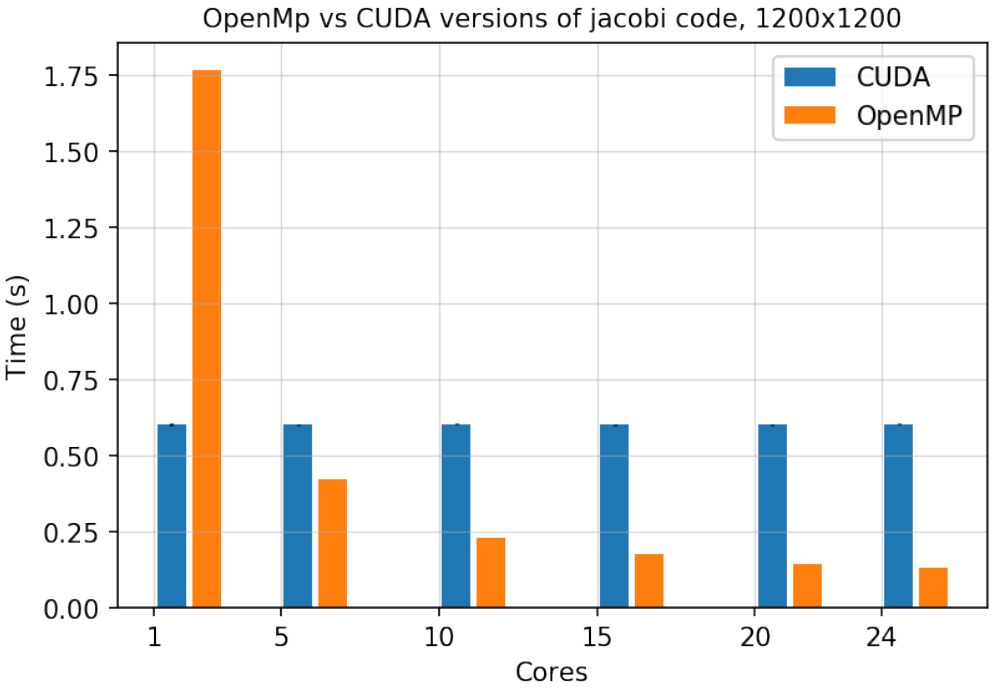

Figure 1:

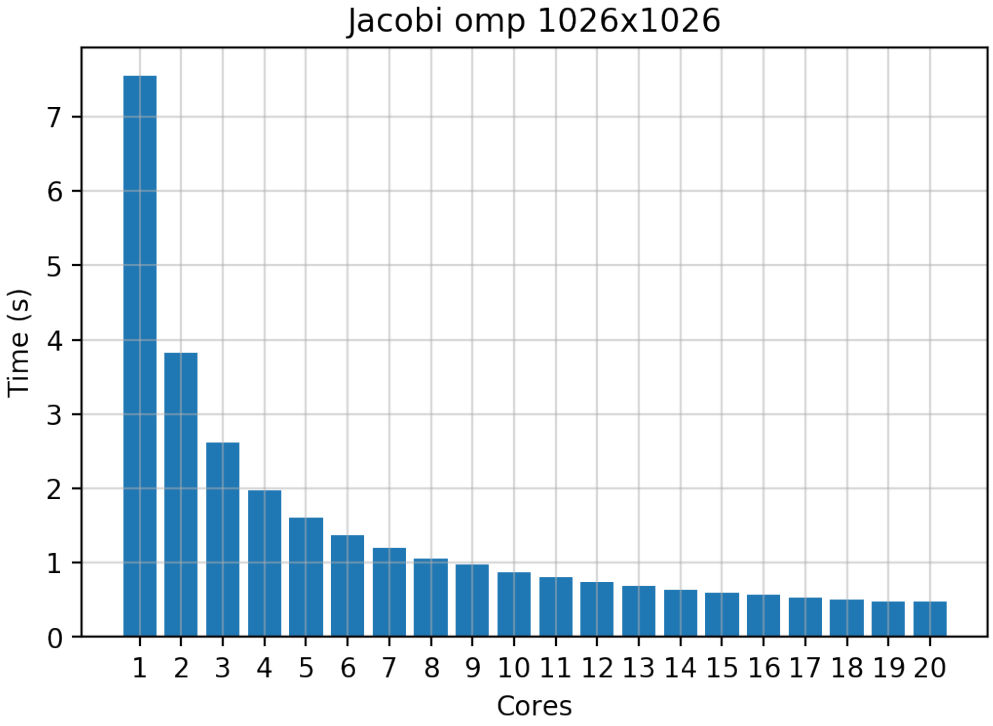

Figure 2: Graph of the timing progress for the OMP code.

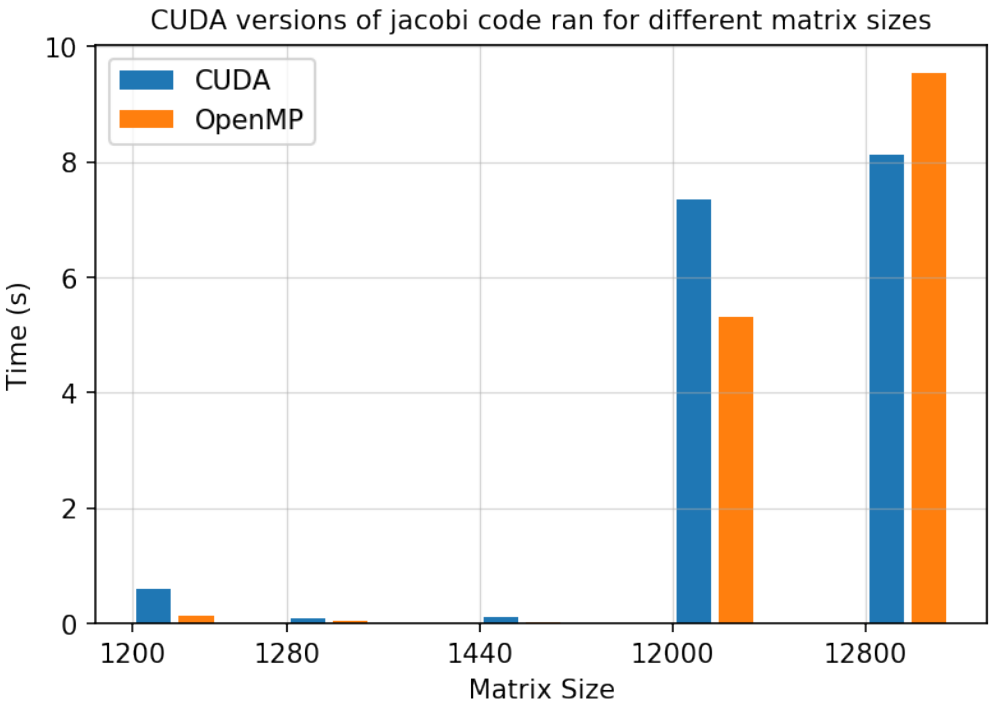

Figure 3: CUDA an OpenMP codes ran for different matrix sizes (100 iterations).

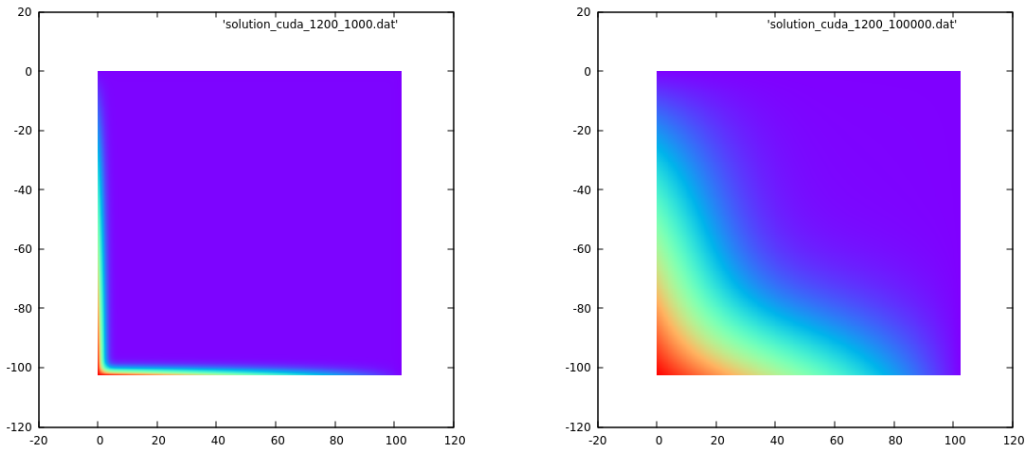

Figure 4: CUDA code, using a matrix of size 1200x1200 and 1000 iterations(left) and 100000 iterations(right).