OpenMP

AUTHORS: Garbarino Giacomo, Parmiggiani Manuel

INFN CLUSTER SPECIFICS

- CPU: Intel(R) Xeon Phi(TM) CPU 7210 @ 1.30GHz
- Number of cores: 256 (64 cores with 4 threads for each one in hyperthreading)

HOW TO EXECUTE A C PROGRAM USING OpenMP LIBRARY ON THE CLUSTER

1. icc -o DFT_parallelized_version -fopenmp DFT_parallelized_version.c
2. ./DFT_parallelized_version

HOW TO GET A REPORT ANALYSIS ABOUT THE PROGRAM

1. icc -o DFT_serial_version -fopenmp DFT_serial_version.c
2. ./DFT_serial_version
3. gprof DFT_serial_version gmon.out > analysis.txt

DISCUSSION

The analyzed program implements the Discrete Fourier Transform (a.k.a. DFT) algorithm. A finite sequence ***x*** of 10000 elements is represented using two arrays: one for the real part (named ***xr***) and one for the imaginary part (named ***xi***), both with a length equal to 10000. The elements of the real part are all equal to 1, while the elements of the imaginary part are all equal to 0. The serial implementation takes about 10,62 seconds to compute the DFT on both parts of ***x*** and no errors occurred, because the function to check the result about the real part returns exactly 10000. Analyzing the execution through the procedure described above, 85% of the total execution is spent by the function to calculate the DFT. This function represents a hotspot, so it should be vectorised to reduce its execution time, but this is not possible because the nested loops lead to the write-after-write case. In particular, the variable ***Xr_o[k]*** is rewritten with its previous value. There's an output dependency. The same goes for ***Xi_o[k]***. To parallelize the function which calculates DFT using OpenMP, it's necessary to add the instruction ***#pragma omp parallel for num_threads(n)***, where ***n*** is the number of threads, before the first for loop. Doing this, the nested loops are parallelized and collapsed into one loop. Here's a resume about the performances:

- 2 threads, execution time ≃ 5.15 s.
- 8 threads, execution time ≃ 1.21 s.
- 16 threads, execution time ≃ 0.62 s.
- 32 threads, execution time ≃ 0.34 s.
- 64 threads, execution time ≃ 0.25 s.
- 128 threads, execution time ≃ 0.22 s.
- 256 threads, execution time ≃ 0.28 s.

The best result is obtained with 128 threads, but the execution times are such low that sometimes the best result can also be achieved also with 64 and 256 threads.