# OpenMP Assignment Report

High Performance Computing Course – Riccardo Caprile (43707074)

## Execution of the DFT program using OpenMP on INFN cluster

- icc -o omp_homework_parallel  -O2 -fopenmp omp_homework_parallel.c
- ./omp_homework_parallel

Makefile is provided.

The cluster specifics are : CPU  Intel(R) Xeon Phi(TM) CPU 7210 @ 1.30GHz , with 256 number of cores.

The command “make clean” is used for removing all the .o files.

-O2 is used for the local optimisations, which is a compromise between compilation speed, optimisation , code accuracy and executable size.

As we have seen during lessons , it is possible to get a report about the program adding in the command line during the phase of compilation the string -qopt=2 for showing which loops are not vectorized and why.

## Code Analysis

This program implements the Discrete Fourier Transform. Basically we have a sequence x of 1000 elements (N). The sequence is represented with 2 arrays : xr for the real part and xi for the imaginary part. Value of the real part is set to 1 , whereas the imaginary part is set to 0.

The first thing to is to try the serial implementation of the algorithm , this kind of implementation takes about 10.42 seconds with no errors because the check about the real parts returns exactly 10000.

The next part of the analysis , before talking about the parallelization , is the hotspot identification which is an important step in the parallelism process. We can use a simple approach to identify the hotspot based on the identification of the loops inside the program that spend a considerable amount of time on the completion of the task.

In the DFT function we have two nested loops , where the imaginary and real part are computed and where the program spend most of the time inside. So we have to focus on the parallelization of this loop. Unfortunately the hotspot cannot be vectorised because the nested loops are included in the case of write-after-write. Indeed , the variable Xr_o[k] is rewritten with its previous value , so there is an output dependency. There is the same situation for Xi_o[k] variable.

The latest loop , where we got the normalization in the case of the IDFT , is vectorized.

For the parallelization of the function which calculates DFT , it is necessary the usage of OpenMP , with the instruction #pragma omp parallel for num_threads(n) , n is the number of threads we want to test our program with (important when we have to analyse the performance of the program ).

Now we have a summary about the performances (execution time) of the program using a different number of threads :

- **Execution time with 2 threads** : 5.038926 seconds
- **Execution time with 8 threads:** 1.210994 seconds
- **Execution time with 16 threads:** 0.623548 seconds
- **Execution time with 32 threads**: 0.364656 seconds
- **Execution time with 64 threads:** 0.216998 seconds
- **Execution time with 128 threads**: 0.246904 seconds
- **Execution time with 256 threads**: 0.255944 seconds

## Conclusions

In the end , the objective of this assignment was to optimize the performance of a program in C using parallelization and vectorization , through OpenMP and Icc compiler. To evaluate the performance of the parallelized program , I've used the execution time and we can clearly see that with 2 threads the execution time is halved with respect to the serial version , with 8 threads is lower , with 16 is halved again and even with 32. Starting from 64 threads the execution is a little bit lower but not halved.

I got the best result with 64 threads , but the execution time are so low that sometimes the best performance could be reached with 128 and 256 threads. Using a a larger number of threads does not always bring better performance. This situation tells us that we need to find the optimal balance between the number of threads and the workload.