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 -O1 -fopenmp omp\_homework\_parallel.c
* ./omp\_homework\_parallel

Makefile is provided.

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

-O1 is 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 xxx 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. 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.

For the parallelization of the function which calculates DFT , it is necessary the usage of OpenMP , using 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 analyze the performance of the program ).

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

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

* **Execution time with 2 threads :**
* **Execution time with 8 threads:**
* **Execution time with 16 threads:**
* **Execution time with 32 threads:**
* **Execution time with 64 threads:**
* **Execution time with 128 threads:**
* **Execution time with 256 threads:**

**Conclusions**