Lab 3 - TFLOP Performance

Advanced Architectures

University of Minho

The Lab 3 focus on the development of efficient code for Intel Xeon multicore and Intel Knights Landing many-core devices, by covering the programming principles that have a relevant impact on performance, such as vectorisation, parallelisation, and scalability. Use a cluster node (e.g., `qsub -qmei -I -lnodes=1:ppn=?,walltime=...`), and the Knights Landing compute server.

This lab tutorial includes one homework assignment (HW 3.1) and two exercises to be solved during the lab class (Lab 3.x).

Improving the performance of scientific applications is often achieved by processing more data per unit of time rather than processing faster a given data set (see Gustafson's Law). One example is the analysis of molecule docking, where faster applications can process more time steps in a simulation.

The goal of Lab 3 is to evaluate the throughput of a matrix dot product algorithm on these Intel-based systems, where the matrix size should be proportional to the number of running threads. The first two exercises feature an implementation of the vector dot product, whose workload increases automatically with the number of threads, to get acquainted with these compute servers. The impact of vectorization and data alignment must be assessed for the best performing configuration of the code in each server.

Measure and document the throughput of every code variation in each exercise in a spreadsheet. Ideally, a K-best measurement heuristic should be used but, due to the short duration of this session, the best of three measurements should be used. A graph comparing the throughput of all implementations must be plotted at the end of the lab session.

To load the compiler in the environment use both commands:

Intel Compiler: `source /share/apps/intel/parallel_studio_xe_2019/compilers_and_libraries_2019/linux/bin/compilervars.sh intel64`

GNU Compiler (for several system libraries): `module load gcc/4.9.0`

3.1 The Xeon Multicore Device

Goals: to develop skills in the design of parallel and vectorisable code for the Xeon multicore device.

Consider the code provided in the attached file, with the vector dot product algorithm.

HW 3.1 Run the provided code on a dual-socket cluster node on the `mei` queue. Measure and record the best throughput using all physical cores and with SMT enabled (HyperThreading).

The code is highly vectorisable, but the `Makefile` explicitly excludes this compiler option. Remove this restriction and measure the code for the best performing thread configuration. Repeat this test with vectorisation and data alignement (check the source code), and plot the results. How faster did you expect the code would be run and how fast it did run? Can you explain this result?

OMP_NUM_THREADS: environment variable to set the number of threads.

3.2 The Knights Landing Compute Server

Goals: to develop skills in developing efficient code for the Knights Landing compute server.

Lab 3.2 Consider the parallel code from the previous exercise.

Access the Knights Landing compute server by executing `ssh compute-002-1` once inside the SeARCH cluster. Note that your home is already mounted on the server. Run the code with 64, 128, and 256 threads. Measure and record the throughput. Repeat the tests for the best thread configuration with vectorisation, vectorisation and data alignement (check source code and Makefile options, respectively), and plot the results. How does it compare with the previous server? Does the vectorization have the expected impact on performance?

Setup the environment by running the following commands:

```
export PATH=/share/apps/gcc/4.9.0/bin:$PATH

export LD_LIBRARY_PATH=/share/apps/gcc/4.9.0/lib:$LD_LIBRARY_PATH

source /opt/intel/compilers_and_libraries/linux/bin/compilervars.sh intel64
```

Note that you do not have exclusive access to this server, so perform multiple measurements to ensure that the results are not affected by other users.

3.3 Matrix Multiplication on Xeon Systems

Goals: to comprehend the concepts of the **HW 3.1** and **Lab 3.2** by implementing an efficient matrix multiplication code for the target servers.

Lab 3.3 Develop a simple parallel implementation of the matrix multiplication algorithm. Study the impact of vectorisation and data alignment on both Xeon servers that were used in the previous exercises. Compare these results with the ones obtained in Lab1.