Lab 3 - TFLOP Performance

Advanced Architectures

University of Minho

The Lab 3 focus on the development of efficient code for the Intel Xeon Phi computing unit by covering the programming principles that have a relevant impact on performance, such as vectorisation, parallelisation, and scalability. Use a cluster node with an Intel Xeon Phi (by specifying the keyword `phi` in job submission), use only 4 CPU cores and do not submit interactive jobs (e.g., `qsub -lnodes=1:ppn=4:phi,walltime=...`).

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

The goal of improving the performance of scientific applications is often to process more data per unit of time rather than to process a given data set faster (see Gustafson's Law). One example is the analysis of molecule docking, where faster applications are capable to process more time steps in a simulation. The first two exercises feature an algorithm whose workload increases with the number of threads. Therefore, a throughput metric, such as GFLOP/sec, is best suited for this problem instead of considering the execution time.

To load the compiler in the environment use the one of the following commands:

Intel Compiler: `source /share/apps/intel/compilers_and_libraries_2016.0.109/linux/bin/compilervars_global.sh intel64`.

GNU Compiler (for several system libraries): `module load gnu/4.7.2`.

To copy the libraries and binaries to the Intel Xeon Phi see the commands in the `Makefile`.

3.1 The Xeon Phi Native Mode

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

Consider the code provided in the attached file, with the *SAXPY* algorithm.

HW 3.1 Run the provided code on a 4-core Xeon CPU device. Measure and record the best execution time.

The code is highly vectorisable but the `Makefile` explicitly excludes this compiler option. Remove this restriction, run the code again (use the same measurement methodology) and comment on the impact that the vectorisation has on the performance of this device. How faster did you expect the code would be run and how fast it was? Can you explain this result?

Complement the provided code adding an outer loop that iterates through a preset number of threads and parallelise it using OpenMP. Note that the iteration of the inner loops must be replicated through all threads, so make sure that the iteration counters of those loops are private. Also, each thread will only process a subset of the array, which will require a stride with a step equal to the `LOOP_COUNT`, to be coded inside the first loop.

Lab 3.1 Open two sessions on the compute node (not the frontend this time): use one to compile the code and copy the binary and libraries to the device (see the `Makefile`), and use the other to access the device, using `ssh user@mic0`.

Run the extended code in native mode with 2 threads on the same Xeon Phi device core. Note that you need to force a compact thread affinity, otherwise a 2-thread version will run in 2 separate cores (by default).

Compare the performance in GFLOP/sec between the multicore Xeon and the Xeon Phi versions. Test the Xeon Phi code with 1, 2 and 4 threads/core, with both compact and scatter thread affinity. Justify the obtained results.

OMP_NUM_THREADS: environment variable to set the number of threads.

KMP_AFFINITY: environment variable to set the thread affinity.

3.2 Offloading to the Xeon Phi

Goals: to develop skills in offloading parallel code to the Xeon Phi.

Lab 3.2 Consider the code from the previous exercise and the suggestions given during the class. Adapt the *SAXPY* code to be offloaded to the Xeon Phi.

Note that now the binary must not be copied to the device, as the offloading is similar to the GPU. Run the code from the host and ensure that the Xeon Phi libraries location is properly configured, since they will be accessed from the device runtime system.

```
export MIC_LD_LIBRARY_PATH=/opt/intel/composer_xe_2013.1.117/compiler/lib/mic/
```

Measure the performance with 1, 2 and 4 threads/core, and compare the results with the values from the previous exercise. Compare the performance with/without aligned data structures.

- `__declspec ( target (mic)) float fa[FLOPS_ARRAY_SIZE] __attribute__((align(64)));`: Declare an aligned structure on Xeon Phi
- `#pragma offload target (mic)` followed by `#pragma omp parallel for`: Offload and parallelization of a loop on the Xeon Phi

3.3 Matrix Multiplication on the Xeon Phi

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

Lab 3.3 Adapt the parallel matrix multiplication code from Lab 1 to run in both native and offload modes of the Xeon Phi and set the matrix size to $4096 * 4096$. Measure and compare the performance for the following 3 cases: using **all** cores in the multicore Xeon node 641 (e.g., `qsub -lnodes=1:ppn=32:r641,walltime=...`, since these devices support 2-way SMT) and 4 threads/core on the Xeon Phi (why 4?) for both native and offload modes (e.g., `qsub -lnodes=1:ppn=1:phi:r662,walltime=...`).