Lab 1 - Parallel and Vectorisable Code

Advanced Architectures

University of Minho

The Lab 1 focus on the development of efficient CPU code by covering the programming principles that have a relevant impact on performance, such as cache usage, vectorisation and scalability of multithreaded algorithms. Submit jobs to the `mei` queue in SeARCH to measure code execution times, but be careful to always use the same node architecture (i.e., compute-6xx ou compute-7xx).

This lab tutorial includes one homework assignment (HW 1.1), three exercises to be solved during the lab class (Lab 1.x) and suggested additional exercises (Ext 1.x).

A separate compacted folder (lab1.zip) contains the template for an example code (a squared integer matrix-matrix multiplication, and a derived sample irregular workload) and a simple script to measure the code execution times.

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

GNU Compiler: `module load gnu/4.9.3`.

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

If you want to switch compilers execute `module purge` before loading the compiler to use. Remember that this must be done inside a script if you are not using the compute node interactively.

All performance measurements for the entire session must be documented and plotted in a spreadsheet, otherwise the exercises will be considered incomplete.

1.1 Efficient Cache Usage

Goals: to develop skills in common optimization techniques and efficient cache usage.

HW 1.1 Study the efficiency of the sequential matrix multiplication code `regularMatrixMult` by assessing the impact of the optimizations studied in previous classes, which are available on the code file. Measure and document (on a spreadsheet) the performance of each optimization for a matrix size that fits in the L1, L2 and L3 caches, and in RAM (by modifying the `#define SIZE` clause in the code).

1.2 Vectorisation

Goals: to develop skills in vector report analysis and optimisation.

Lab 1.2 Compile the provided code with the supplied matrix-matrix multiplication function. Use either Intel or GNU compilers (Intel is strongly recommended), with the respective vectorisation flags. Do not forget to add a flag to request a full report on the vectorisation results.

Complete the provided code with a new version of the matrix multiplication function, containing the necessary modifications to the code and adding `pragma` clauses to aid the compiler to generate vector code. Analyse the performance to assess the impact of the optimisations.

GNU Compiler: `-O2 -ftree-vectorize -fopt-info-vec-all`.

Intel Compiler: `-O2 -vec-report3`.

1.3 Performance Scalability

Goals: to comprehend the concepts restricting performance scalability of multithreaded algorithms.

Lab 1.3 Consider two similar synthetic parallel algorithms, one with regular and the other with irregular workloads. It is not necessary to analyze the algorithms or the code. Assess the scalability of these algorithms when using static and dynamic workload distributions (functions `(ir)regularWorkload(Static)Dynamic`) for several number of threads and problem sizes. Which scheduler is best fit for each type of workload? Plot the results using a column chart for 1, 2, 4, 8, max_#cores, 1.5x max_#cores, 2x max_#cores, 3x max_#cores and 4x max_#cores.

If the environment variable `DYNAMIC` is set to `yes` (by `export DYNAMIC=yes`) and the code recompiled the algorithms will use a dynamic scheduling strategy, which otherwise will be static. To test the algorithm with the irregular workload set the environment variable `IRREGULAR` to `yes` and recompile the code, which otherwise the workload will be regular. For instance, if you perform `export DYNAMIC=yes` and `export IRREGULAR=no` the application will be compiled to run the algorithm with a regular workload and a dynamic scheduling strategy.

Ext 1.3 Experiment with different chunks of data that are assigned to each thread in the dynamic scheduler. How does it affect the performance for various matrix sizes? (to be solved at home, after the lab session).