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. Use either `compute-652-1` or `compute-652-2` nodes, each with dual Intel Xeon E5-2670v2 @ 2.50GHz CPU, to measure code execution times.

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 scripts to adequately measure the code execution times and to plot/visualize the results.

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

GNU Compiler: `module load gnu/4.9.0`.

Intel Compiler: `module load gnu/4.7.0 && module load intel/2013.1.117`.

If you want to switch compilers execute `module purge` before loading the compiler to use.

1.1 Efficient Cache Usage

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

HW 1.1 Improve the efficiency of the sequential matrix multiplication code `regularMatrixMult` using the optimizations studied in previous classes. Measure and document 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).

Replicate the optimizations implemented for the remaining matrix multiplication functions `irregularMatrixMult`, `(ir)regularWorkloadStatic` and `(ir)regularWorkloadDynamic`. Does the performance scale as expected with the increase in the amount of threads for each version?

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 algorithms with regular and irregular workloads. 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.

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).