Lab 2 - Parallel and Vectorisable Code

Advanced Architectures

University of Minho

The Lab 2 focus on the development of efficient CPU code by covering the programming principles that have a relevant impact on performance, such as vectorisation, parallelisation, and scalability. Use any of the `compute-562` nodes, each with dual Intel Xeon E5-2695 v2 @ 2.40GHz CPU, to perform the execution time measurements. Scripts are provided to execute the time measurements and plot/visualise the results. This lab tutorial includes 3 exercises to be solved during the lab classes.

2.1 Vectorisation

Goals: develop skills in vector report analysis and optimisation.

HW 2.1 Compile the provided code with a matrix-matrix multiplication function. Use either Intel or GNU compilers (Intel is strongly recommended), with the respective vectorisation flags. Complete the provided code with a new version of the matrix multiplication function, containing the necessary modifications to the code, and adding `pragma` directives to instruct the compiler. Analyse the performance to assess the impact of the optimisations.

Intel: `-vec-report3`.

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

2.2 Parallelisation with OpenMP

Goals: develop skills in advance usage of OpenMP parallelisation `pragma`.

Lab 2.2 Consider the code developed in the previous exercise. Perform a simple parallelisation of the new function using OpenMP and use the provided scripts to measure its performance. Explore the usage of the following `pragma` clauses, evaluate their individual (or combined) impact on performance and justify why they improve/worsen the algorithm execution time:

nowait: disables the default synchronisation of threads at the end of each parallel loop iteration.

schedule: describes the work sharing technique among threads; It can be defined as static, dynamic, guided, runtime, and auto, and each heuristic is best suited for specific characteristics of algorithms, such as their irregularity or task granularity.

Extend the functionality of the algorithm to simultaneously perform a sum of the values of the result array with the remaining computation. Implement this feature using (i) a shared variable among all threads and (ii) a thread-private variable with a final reduction. Beware of the thread concurrency when accessing shared resources (see OpenMP `critical` and `atomic` constructs)!

Ext 2.2 Repeat **Lab 2.2** with the original matrix multiplication function. Comment the results (to be solved at home, after the lab session).

2.3 Performance Scalability

Goals: comprehend the concepts restricting performance scalability of parallel algorithms.

Lab 2.3 Consider the provided parallel matrix-matrix multiplication code. Measure the algorithm execution time for the different number of threads supplied in the script. Does the application performance always improve with the increased number of threads? Which is the configuration that provides the highest efficiency (best ratio between performance and number of threads)? Justify your answer and provide possible solutions.

Ext 2.3 Modify the Perl script to plot the performance values so that it generates an efficiency plot instead of the speedup one (to be solved at home, after the lab session).