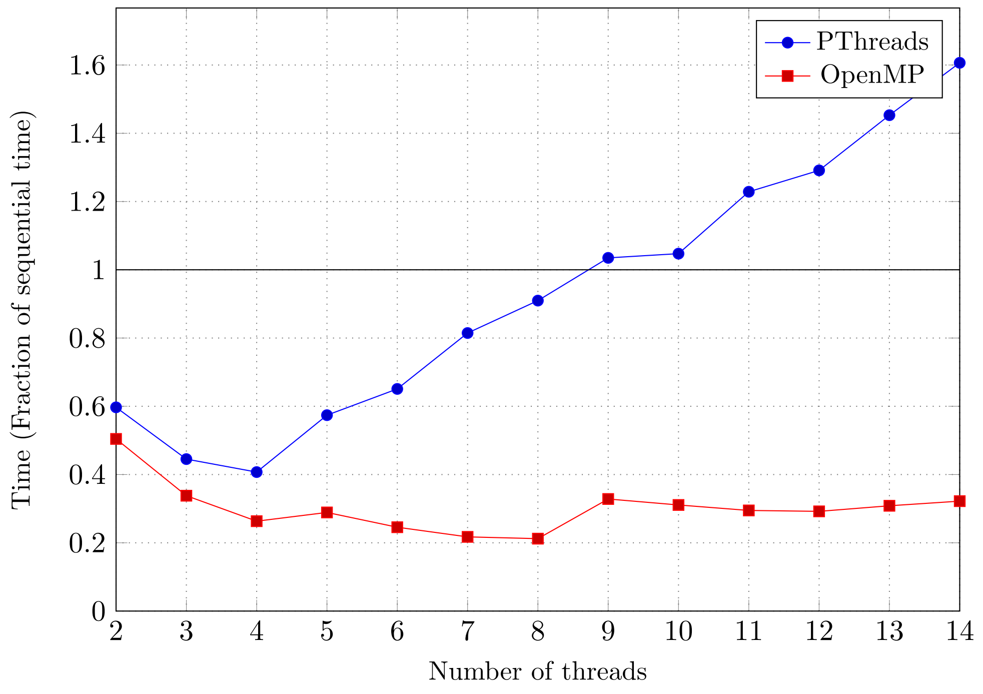
Low-level Parallel Programming (course 1DL550) Uppsala University – Spring 2015 Report for Lab 1 by Team 14

Fredrik Larsson Jimmy Holm Per Bergqwist

5th February 2015

1 Plot



The data used in the plot is the average of 20 runs for each implementation and is presented as fraction of sequential run-time. A value greater than 1 indicates longer run-time than sequential and a value less than 1 indicates a shorter one.

2 System specification

The CPU of the system used for gathering the data presented was an Intel i7 2600k running at a frequency of 4GHz. The system was able to use four cores with Hyper-threading enabled, meaning a possibility of using eight logical cores simultaneously with the drawback that the performance gain from using more than four cores varies depending on the tasks.

3 Questions

A. **What kind of parallelism is exposed in the identified method?**
The methods expose data parallelism as it distributes a shared set of data to be processed by the threads.

B. **How is the workload distributed across the threads?**
For the PThread implementation the workload is distributed uniformly. The agent-vector is divided up among the threads in equal parts and each thread work only on their assigned portion of the data. The OMP implementation makes use of a threadpool, giving threads which finish their assigned data more data to work with until all the data has been processed.

C. **Which number of thread gives you the best results? Why?**
4 or 8 threads gives the best results depending on which implementation that was used. OpenMP gives the best performance at eight threads while the PThreads implementation at four threads. Our conclusion why PThread loses performance with more than four threads is because the overhead from our PThreads implementation counter the performance gain from the Hyper-threading. OpenMP has the best performance at eight threads, however after four threads the gain is minimal. When using more than nine threads, which is more than number of logical cores available to the CPU, the performance for both OpenMP and PThreads get worse and PThread even display worse results than the sequential implementation.

D. **Which version (OpenMP, Pthreads) gives you better results? Why?**
OpenMP produces the best results out of all options, as its use of a thread pool means an asymmetric workload does not hinder performance. The PThread solution displays a naive form of sharing as it is not certain that each task will finish in an equal amount of time and as such threads that finish their part faster, have to wait for the remaining threads to finish up. In OpenMP's implementation, threads that finish can take new tasks and as such they are utilized to a greater extent and slower threads do not hold up the larger task.

4 How to run

The demo runs with a GUI and using the sequential implementation if no flag has overwritten the settings. The following flags can be used to alter the demo and change the implementation for the tick function.

--timing-mode - Without gui.

--pthread - Executes the PThread implementation.

--omp - Executes the OpenMP implementation.

--threads *number* - Tells the demo to use *number* threads when executing.

--silent - Does not print to standard output.

--plot - Stores the timing data to the file testdata.txt