TDDC78: Lab Report

Lab 3: OpenMP

Name	**PIN**	**Email**
Alexander Yngve	930320-6651	aleyn573@student.liu.se
Pål Kastman	851212-7575	palka285@student.liu.se

Contents

1 Introduction

In this lab we were to solve a heat conducting problem on a shared memory computer, by using Open MP.

2 Heat Conducting Problem

In this lab we had to solve a heat conducting problem using the Jacobi iterative method.

3 Our implementation

When we first read the explanation of this lab, and it said that we were only allowed to use $\mathcal{O}(n)$ additional memory we started with splitting every row into equal size parts depending on how many processes we had been provided. This meant that for every row we had to synchronize so no process started on the next one before copying this area into the temporary memory. This turned out to not work good at all, instead of an decrease in time we got a heavy increase. We think that this is because the program needed to synchronize and wait for each other alot.

But then we realized that the original implementation used $2N$ of memory, which is still inside $\mathcal{O}(n)$. So instead we let every process have their own regions of the array, so say for instance that the array is 1000 rows and we are running 4 processes then every process will have 200 rows each. The problem here is that when the processes reaches the final row they will need values from a row that the next process already has modified, because of this we needed to use another N of memory per process.

Every process will now use $3N$ of memory which will give us a total additional memory of $3N * p$ (where p is the no. processes), even if this sounds alot it will still be inside $\mathcal{O}(n)$ if $p \ll N$, which can be assumed since p is normally equal to the number of cores in the processor.

Every process calculates the error of every index and if the error is greater than the tolerance then we set a flag which will make the program end the initial while-loop. Only one process is allowed to update this flag at time.

4 Execution times

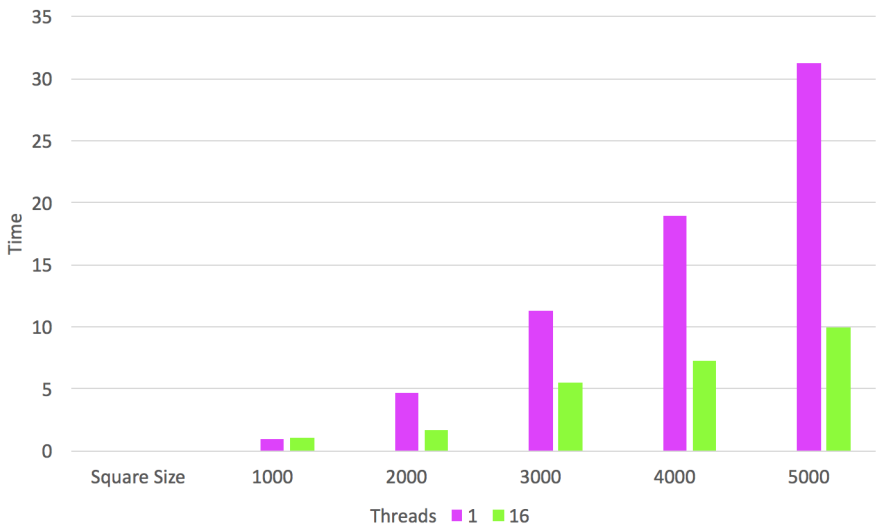

Figure 1 – Results for different sizes of the square running on 1 and 16 threads.