Parallel Computing Project
Heatdiffusion openMP and MPI

Jordy Aaldering, s1004292
Elianne Heuer, s4320514

July 15, 2020

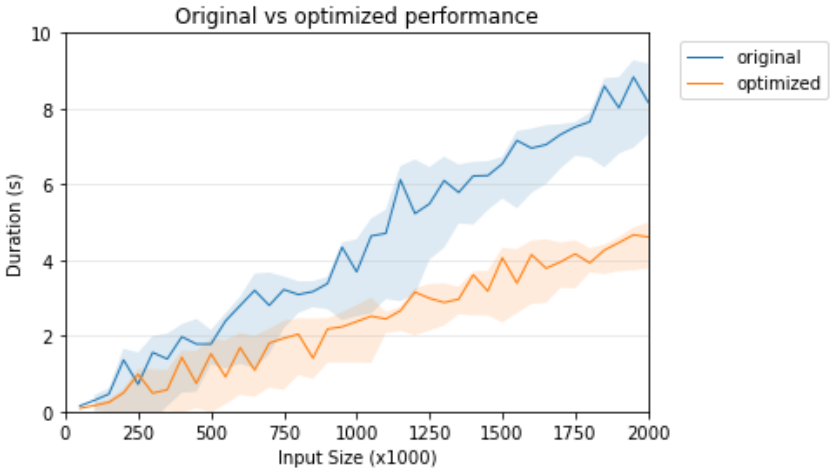
1 Introduction

All tests were done with a heat of 100 and an epsilon of 0.05. We found that increasing the heat or decreasing the epsilon increased the number of iterations with the same factor; thus increasing the total runtime. All tests were run on a system with an i7-7700HQ, which has 4 cores and 8 threads. It has a base frequency of 2.80 GHz and a max turbo frequency of 3.80 GHz, and 8 GB of RAM. The latest compiler version with the compiler flag `-O3` was used for all tests. All tests were repeated 10 times to increase test accuracy.

2 Evaluation

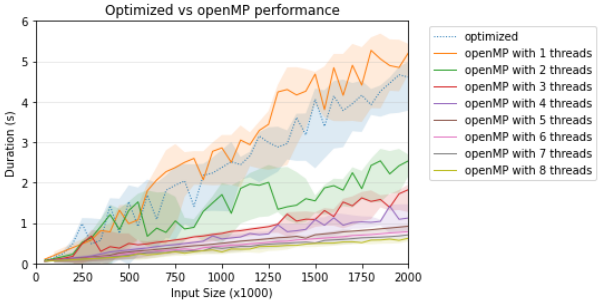
2.1 Sequential

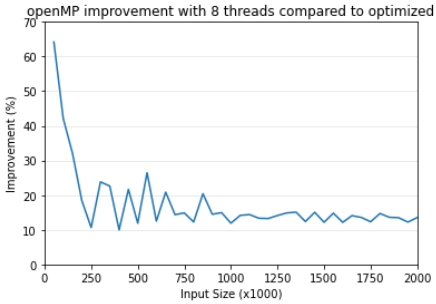
We took a few steps to optimise the original algorithm, since it was not very efficient. The two loops were combined into one single loop which already cut down the runtime by a lot. Here the check for stability was only done if the array was still stable. Array initialisation was replaced by a `memset` which was another minor improvement. In the figure below that these improvements almost doubled the efficiency of our algorithm.



2.2 openMP

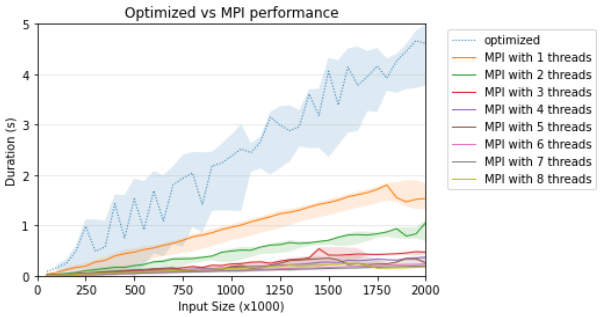
In the graphs below we see that openMP only starts being faster than the optimised algorithm when it is run with at least two threads. This is due to the extra steps required to set up openMP, and the overhead caused by the communication between threads. In the improvement graph we see that there are a lot of fluctuations in improvement when the size of the input array is smaller, this stabilises as the input size grows. The openMP implementation increased efficiency of the optimised algorithm by between 10 and 15 percent.

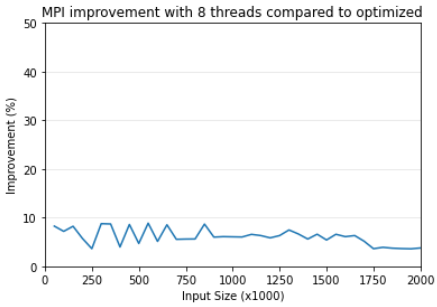




2.3 MPI

Even with just a single thread the MPI version is faster than the optimised algorithm. This is likely due to a bug with the single threaded version as this is not an expected result due to the added overhead that comes with parallelisation. When adding more threads we do get expected results with a similar improvement as the openMP implementation. In the improvement graph we see that there are a lot of fluctuations in improvement when the size of the input array is smaller, this stabilises as the input size grows, similar to the improvement of openMP. The MPI implementation increases efficiency of the optimised algorithm by 5 to 10 percent. With smaller increased efficiency with bigger input.





3 Performance

Comparing the performances of the different programs, we see that the runtime drastically improves with both openMP and MPI as the number of threads increases. MPI seems to run a lot faster when run with 1 thread, but we already discussed that this could be due to a bug in the MPI program.

But we can clearly see the advantage of using openMP and MPI over the improved linear program. If we just look at openMP and MPI, it seems that MPI is faster than openMP. If we compare the improvement over input size compared to the optimised program for both methods, openMP improves less with the array's input size and it seems to stabilise after a while, while the improvement in MPI is more linear with the increasing input size, the improvement does decrease a little bit as the size gets bigger.

4 Team

We worked together on optimising the sequential version and on writing the report. Elianne did the MPI implementation while Jordy did the openMP implementation and evaluation metrics. We did not get to do everything that we wanted to do. Cooperation went very smoothly.