Parallel Programming Assignment 1

Nils Bäckström, Jonathan Löfgren, Anton Sundin

February 11, 2015

1 Problem description

Matrix multiplication is a common operation which can be quite demanding in both memory usage and number of required operations. Fortunately not all elements and operations are depending on each other which implies that parallelization might be used to speed up the task. The task was to design a parallel implementation which divides both memory and work load between all used processors.

2 Implementation

The algorithm for the parallelisation exploits the fact that the operation $C = AB$, where $C_{i,j} = \sum_{k=1}^{n} A_{i,k}B_{k,j}$, can be divided into partial sums which only requires a smaller subset of both matrices.

Matrices A and B are divided into blocks of smaller matrices a and b among the processors in a Cartesian grid. For each step in the algorithm, one processor from each row broadcasts its block a among the other processors in that row, allowing each processor to perform the required operations between block b and the newly acquired block a, storing the result in a third block matrix c. Following this, each processor passes its block matrix b upwards to its column neighbor, after which a new processor is chosen to broadcast a new a until all the processors of each row have shared their block a this way. The results from each block c is then gathered and put in the correct space in the product matrix by the root processor.

3 Results

The algorithm was tested with several matrix sizes for different number of processors. The speedup is measured as $S = T(1)/T(n)$ where n is number of processors used. The result is presented in figure 1. From this figure we can see that the speedup reaches maximum somewhere between 4 and 9 processors. It is also clear that when the number of processors n is above 9, the performance gets worse for all matrix sizes but in particular for small matrices.

4 Conclusion

The reason for why 4-9 processors are optimal is probably because the overhead from the communication doesn't vary with the size of the system, and at small sizes the yield from utilizing a high number of processors isn't enough to outweigh the time lost to communication overhead. The current implementation of the code stores matrices row-wise in memory. A better solution for this algorithm would be to keep the matrix blocks a row-wise, but change matrix blocks b to be stored column-wise to optimize memory caching.

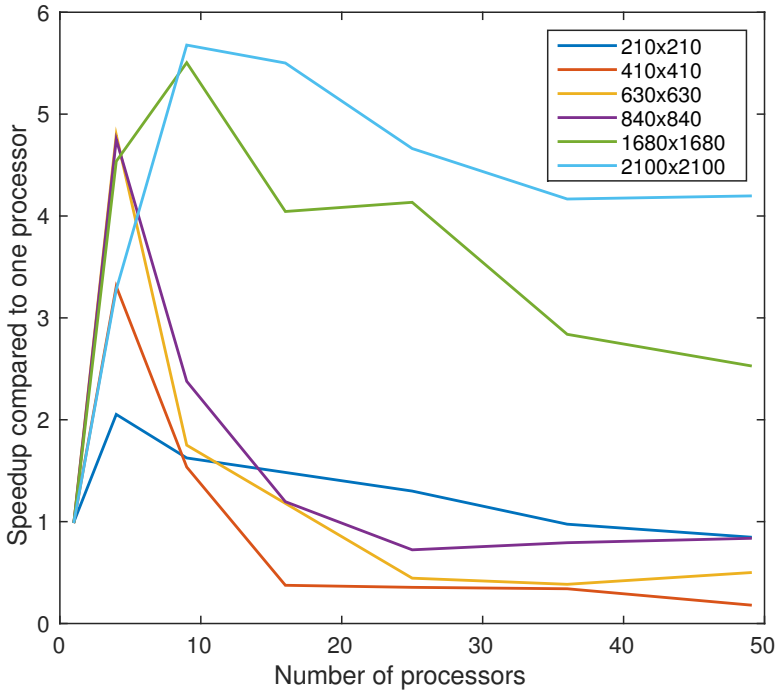

Figure 1: Speedup

5 Appendix

All code is in main.c. In the tar-file you also find a Makefile a matlab file speedupplot.m which contains all data for the figure in the report.

1. main.c
2. Makefile
3. speedupplot.m