CS4234
Project 1

Emil Bengtsson

October 2018

Introduction

This project aims to analyze differences in execution time between a sequential and a parallel implementation of both a matrix-vector multiplication and a matrix-matrix multiplication. Both fixed-workload and fixed-time speedup analyses will be performed.

Implementation

The parallel matrix-vector and matrix-matrix multiplications are implemented in C, using MPI to communicate between processes.

The parallel algorithm for matrix-vector multiplication works like this:
The master process scatters the rows of the matrix among the different processes, with each process receiving close to the same amount of rows. The master process also sends the vector to all the other processes. Then each process does a local matrix multiplication of the matrix rows it received from the master process and the vector to get the part of the result vector that corresponds to the local rows of the matrix. The master process then gathers all the local results from the processes and puts them in the correct places in the final result vector, and the matrix-vector multiplication has been completed.

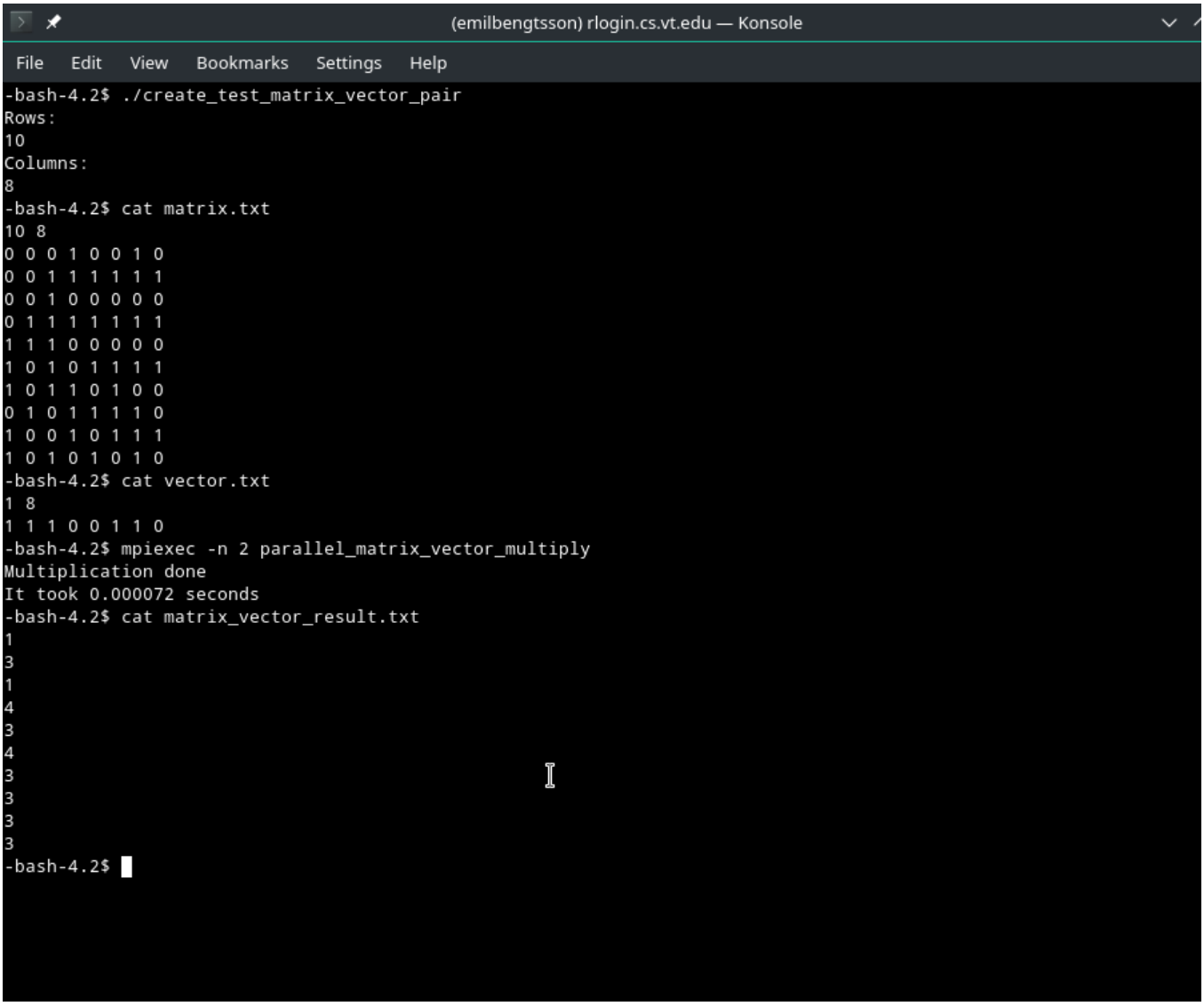
The parallel algorithm for matrix-matrix multiplication was inspired by SUMMA[1], and it works like this:
The master process scatters the columns of the first matrix and the rows of the second matrix among the different processes, with each process receiving close to the same amount of rows and columns respectively. Each process then does a sequential matrix multiply of the local columns of the first matrix with the local rows of the second matrix and ends up with a local result matrix that has the same dimensions as the final result matrix will have. The master process then receives all the local result matrices and adds them all together to form the final result matrix, and the matrix-matrix multiplication has been completed.

The sequential version of both matrix-matrix multiply and matrix-vector multiply can be achieved by running the parallel programs on only one process.

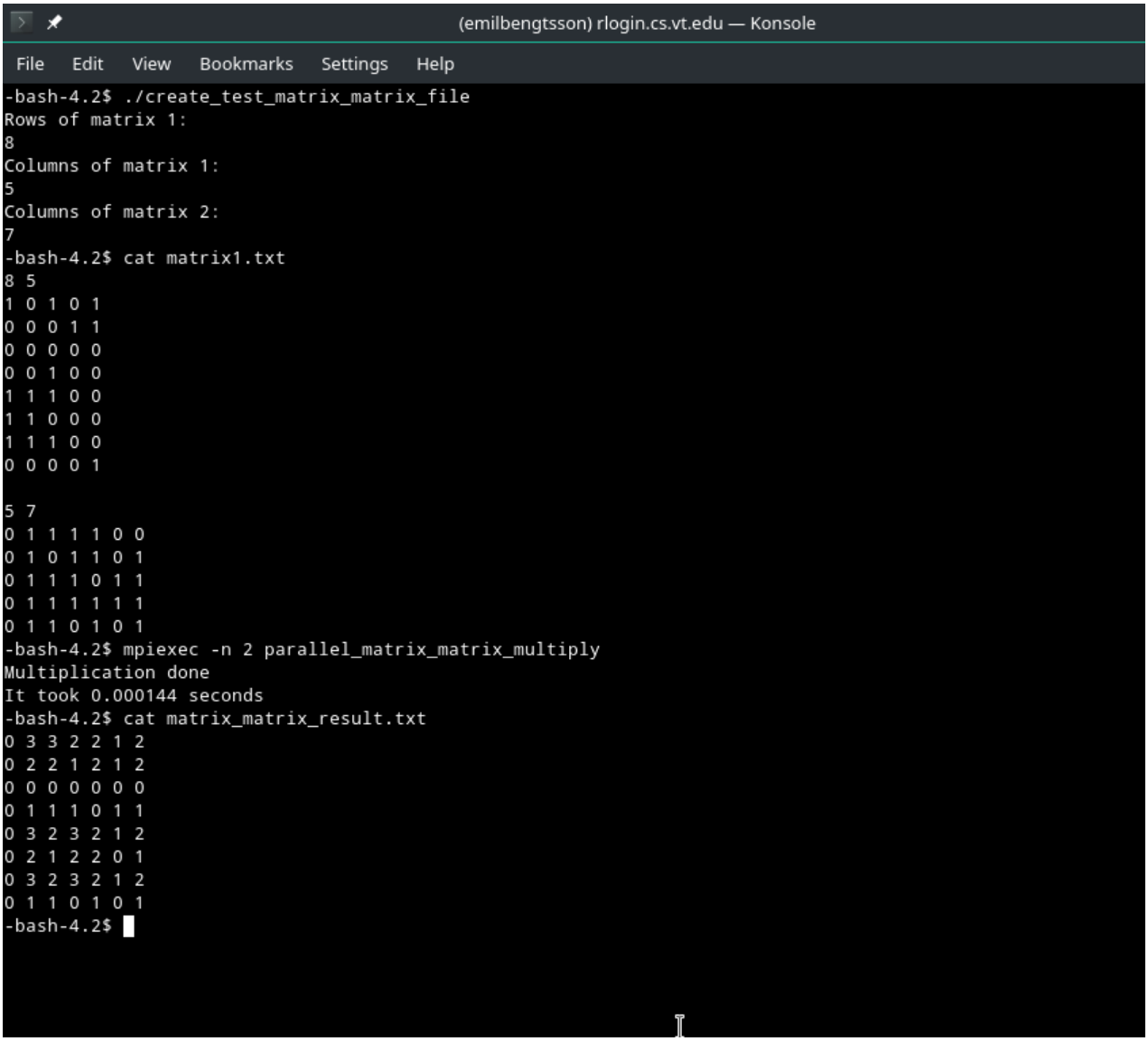
Screen shots of programs

This section contains screen shots of correct execution of the programs.

Matrix-Vector Multiply



Matrix-Matrix Multiply



Results

Matrix-Vector Multiplication

The numbers in the top line of the tables represent the size of the matrix, and the numbers in the leftmost line of the tables represent the number of processes of the program that are run by mpiexec.

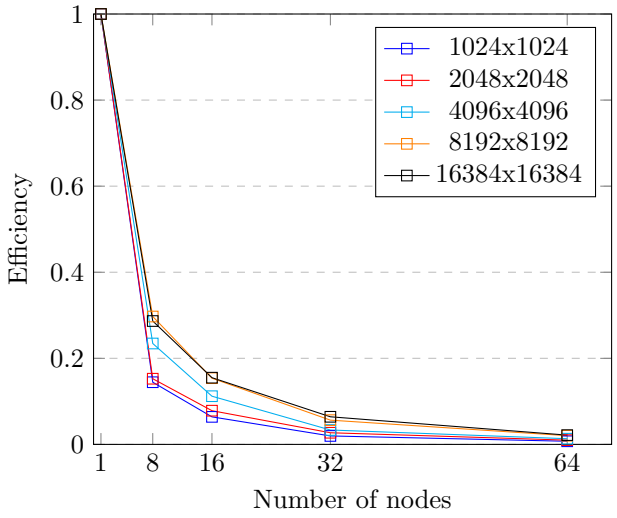
Table of run times

	1024x1024	2048x2048	4096x4096	8192x8192	16384x16384
1 process	0.005169	0.019725	0.077871	0.335553	1.233383
8 processes	0.004472	0.016158	0.041518	0.141172	0.537850
16 processes	0.005050	0.015741	0.043464	0.136304	0.497265
32 processes	0.008225	0.022784	0.072725	0.186018	0.600864
64 processes	0.011097	0.031190	0.092202	0.258943	0.894326

Table of efficiencies

	1024x1024	2048x2048	4096x4096	8192x8192	16384x16384
1 process	1	1	1	1	1
8 processes	0.14448	0.15259	0.234445	0.29711	0.28665
16 processes	0.06397	0.07832	0.11198	0.15386	0.15502
32 processes	0.01964	0.02705	0.03346	0.05637	0.06415
64 processes	0.00728	0.00988	0.01320	0.02025	0.02155

Graph of efficiencies for matrix-vector multiply for different number of processes



Matrix-Matrix Multiplication

The numbers in the top line of the tables represent the size of the matrices, and the numbers in the leftmost line of the tables represent the number of copies of the program that are run by mpiexec.

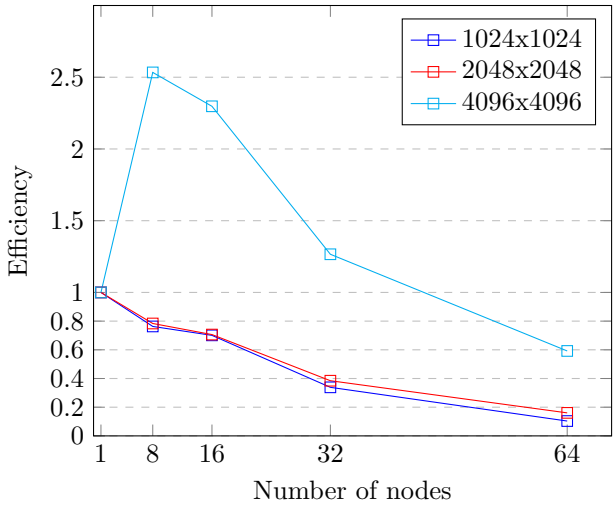
Table of run times

	1024x1024	2048x2048	4096x4096	8192x8192	16384x16384
1 process	5.649425	43.043447	1216.841048	-	-
8 processes	0.925617	6.867018	60.034989	1537.011752	-
16 processes	0.503864	3.808616	33.102984	268.822833	-
32 processes	0.521559	3.497932	30.035726	203.068205	-
64 processes	0.852439	4.192514	32.115164	217.841619	-

Table of efficiencies

	1024x1024	2048x2048	4096x4096	8192x8192	16384x16384
1 process	1	1	1	-	-
8 processes	0.76293	0.78352	2.53361	-	-
16 processes	0.70076	0.70634	2.29745	-	-
32 processes	0.33849	0.38454	1.26603	-	-
64 processes	0.10355	0.16042	0.59203	-	-

Graph of efficiencies for matrix-vector multiply for different number of processes



Conclusion

Matrix-Vector Multiplication

The results from the matrix-vector multiplication program are very interesting. The efficiency for the program is really bad and it does not scale strongly or weakly at all. If you analyze the data in the tables from a fixed-workload perspective it does really poorly, and the efficiency drops whenever you increase the number of processes. If you

analyze the data in the tables from a fixed-time perspective it does quite poorly as well since the run time always increases if you double the amount of processors and the amount of data.

I think the program might be doing so poorly because the it takes such a short time to run, and therefore the extra overhead that comes with all the communication among the processes eliminates most of the time gained from parallelizing the multiplication. The fact that the efficiency rises slightly as the size of the matrix is increased (and the time the program takes to execute is increased) supports my theory. The execution time for when the program is run on 32 and 64 processes is also a lot worse than one would expect. I think that is partially due to the communication overhead that comes with the increase in processes, but the biggest reason is probably because the program is run on only one node (as stated in the assignment). One general purpose compute node in the rlogin cluster only has two processing units with 10 cores each. This means that if you run enough processes, they will not all be able to run at the same time which will lead to a decrease in performance.

Matrix-Matrix Multiplication

The matrix-matrix multiplication program also produces some interesting (and strange) data. Firstly, when I tried to do a matrix-matrix multiplication with 8192x8192 matrices sequentially, it took more than 20 minutes and I had to stop the program to continue trying to run all the other different matrix-matrix multiplications. I also could not do any of the 16384x16384 matrix-matrix multiplications since they took more than 20 minutes for any amount of processes. Because of this I have not calculated the efficiencies for 8192x8192 and 16384x16384 matrix-matrix multiplications and they are not included in the graph either. The matrix-matrix multiplications with the smaller matrices went quite well however, and I mostly got reasonable values. There is one weird thing in the graph though; the run time for the 4096x4096 matrix-matrix multiplication when run sequentially is really high. This messes with the efficiency of the multiplications done on the same matrices but with more processes. For example with 8 processes the efficiency is about 2.5 which is quite strange as it should not be able to achieve a speedup greater than 8 (and an efficiency greater than 1). I have a theory for why this might be though, and that is that the core that has to do all the multiplication in the sequential program might become overheated and have to run at a lower clock speed. This same thing is seen in the run time of the 8192x8192 matrix-matrix multiplication running on 8 processes and it makes the speedup of the program from running on more processes seem almost ridiculous.

If you look at the data in the tables from a fixed-workload perspective you can see that the efficiency drops quite a bit every time you run the program on more processes. There is, however, a minor exception when you increase the processes from 8 to 16, then the efficiency decreases only a little bit. Perhaps if you ran the program on a node with more processing units, or on multiple nodes, the fixed-workload speedup might look a little better when you increase the processing units to 32 and 64 too. If you look at the data in the tables from a fixed-time perspective you can see that it looks pretty bad and that the program does not scale that well, except when you double the processors from 8 to 16 and the matrix size from 1024x1024 to 2048x2048. Then the program scales quite well, with the efficiency only dropping by a little less than 6%.

Optimization

One optimization I did on the matrix-matrix multiplication program was that instead of splitting the largest matrix into as many chunks as the processes that was running the program and sending one chunk and the whole other matrix to each process, I now split both matrices into chunks and calculate the result as is described in the implementation section of the report. This sped up the program a little bit but not as much as I thought it would, I think this is because I still send a lot of data between the processes.

References

[1] van de Geijn, Robert A. and Watts, Jerrell. *SUMMA: Scalable Universal Matrix Multiplication Algorithm.* University of Texas at Austin, 1995.