CS4234
Project 2

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 pthreads.

The parallel algorithm for matrix-vector multiplication works like this:
The "master" thread creates all the other threads, with each thread getting a rank assigned to it. Each thread then calculates which rows in the matrix it is responsible for. Then the thread multiplies its assigned rows with the vector and stores the results in the corresponding places in the result vector. When all threads are done they are joined by the master thread, and the result can be found in the result vector.

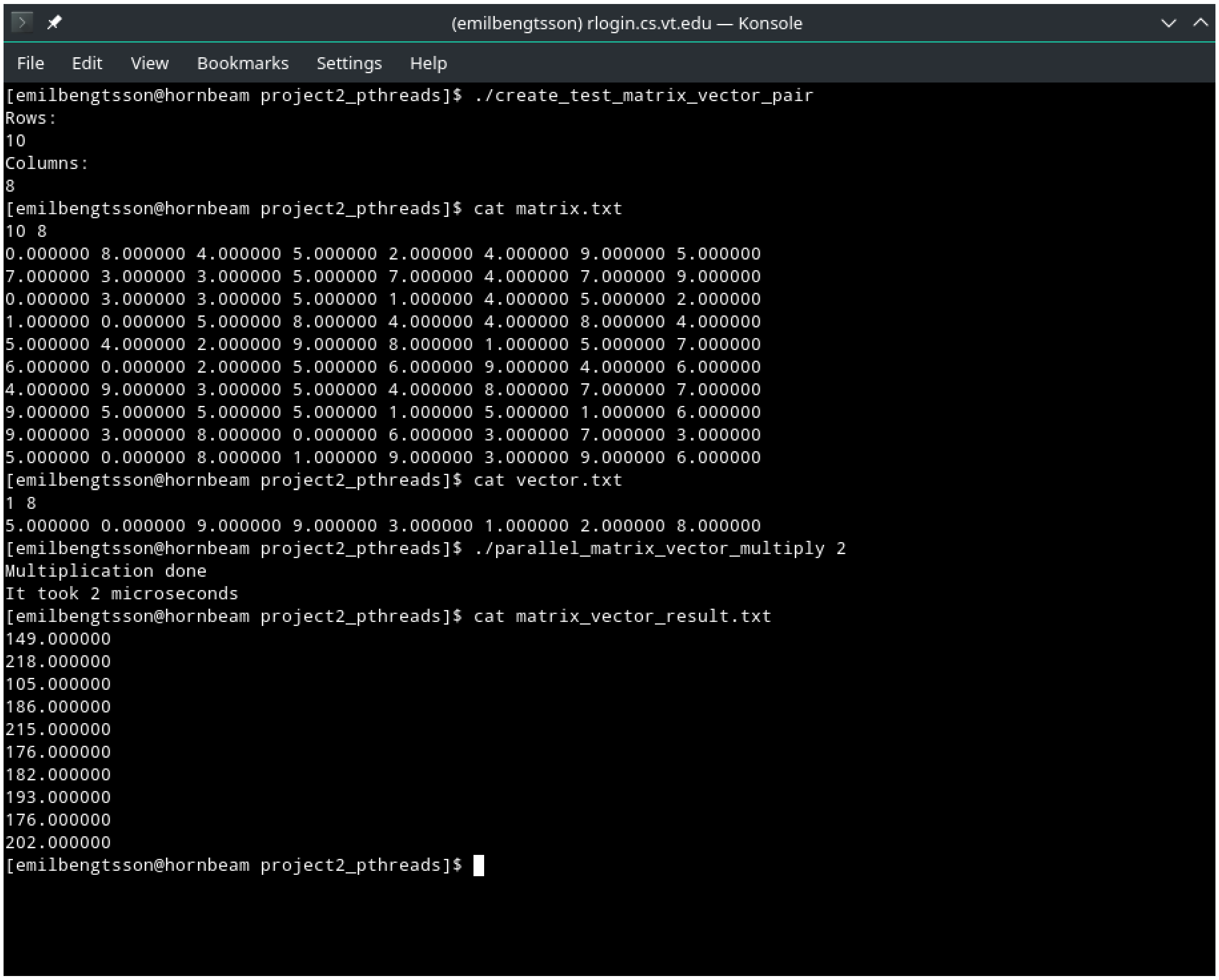
The parallel algorithm for matrix-matrix multiplication works like this:
The "master" thread creates all the other threads, with each thread getting a rank assigned to it. Each thread then calculates which rows in the first matrix it is responsible for. Then the thread multiplies these rows with all the columns in the second matrix, and stores the results in the result matrix. When all threads are done they are joined by the master thread, and the result can be found in the result matrix.

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

Screen shots of programs

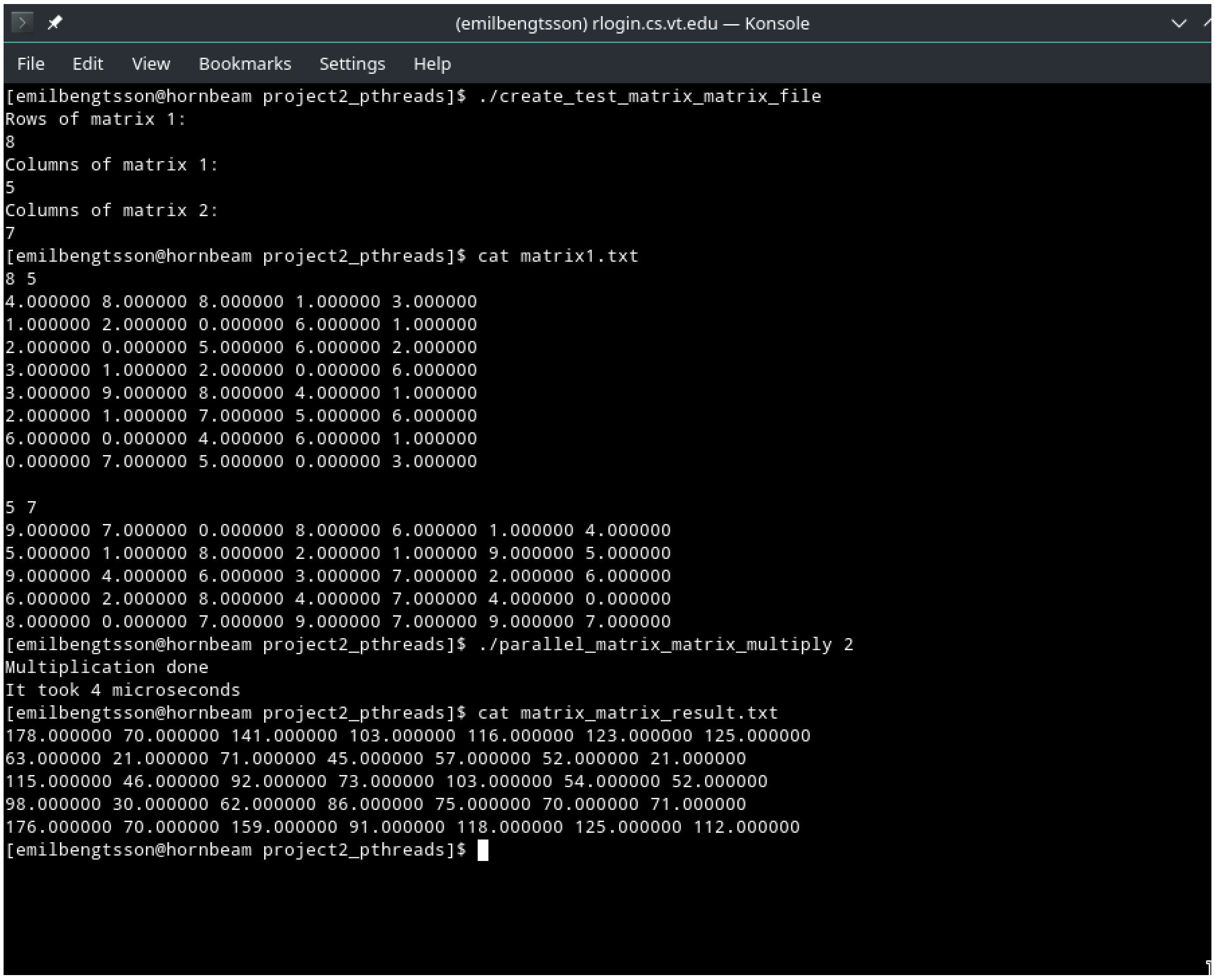
This section contains screen shots of correct execution of the programs.

Matrix-Vector Multiply



```
[emilbengtsson@hornbeam project2_pthreads]$ ./create_test_matrix_vector_pair
Rows:
10
Columns:
8
[emilbengtsson@hornbeam project2_pthreads]$ cat matrix.txt
10 8
0.000000 8.000000 4.000000 5.000000 2.000000 4.000000 9.000000 5.000000
7.000000 3.000000 3.000000 5.000000 7.000000 4.000000 7.000000 9.000000
0.000000 3.000000 3.000000 5.000000 1.000000 4.000000 5.000000 2.000000
1.000000 0.000000 5.000000 8.000000 4.000000 4.000000 8.000000 4.000000
5.000000 4.000000 2.000000 9.000000 8.000000 1.000000 5.000000 7.000000
6.000000 0.000000 2.000000 5.000000 6.000000 9.000000 4.000000 6.000000
4.000000 9.000000 3.000000 5.000000 4.000000 8.000000 7.000000 7.000000
9.000000 5.000000 5.000000 5.000000 1.000000 5.000000 1.000000 6.000000
9.000000 3.000000 8.000000 0.000000 6.000000 3.000000 7.000000 3.000000
5.000000 0.000000 8.000000 1.000000 9.000000 3.000000 9.000000 6.000000
[emilbengtsson@hornbeam project2_pthreads]$ cat vector.txt
1 8
5.000000 0.000000 9.000000 9.000000 3.000000 1.000000 2.000000 8.000000
[emilbengtsson@hornbeam project2_pthreads]$ ./parallel_matrix_vector_multiply 2
Multiplication done
It took 2 microseconds
[emilbengtsson@hornbeam project2_pthreads]$ cat matrix_vector_result.txt
149.000000
218.000000
105.000000
186.000000
215.000000
176.000000
182.000000
193.000000
176.000000
202.000000
[emilbengtsson@hornbeam project2_pthreads]$
```

Matrix-Matrix Multiply


(emilbengtsson) rlogin.cs.vt.edu — Konsole
File Edit View Bookmarks Settings Help

```
[emilbengtsson@hornbeam project2_pthreads]$ ./create_test_matrix_matrix_file
Rows of matrix 1:
8
Columns of matrix 1:
5
Columns of matrix 2:
7
[emilbengtsson@hornbeam project2_pthreads]$ cat matrix1.txt
8 5
4.000000 8.000000 8.000000 1.000000 3.000000
1.000000 2.000000 0.000000 6.000000 1.000000
2.000000 0.000000 5.000000 6.000000 2.000000
3.000000 1.000000 2.000000 0.000000 6.000000
3.000000 9.000000 8.000000 4.000000 1.000000
2.000000 1.000000 7.000000 5.000000 6.000000
6.000000 0.000000 4.000000 6.000000 1.000000
0.000000 7.000000 5.000000 0.000000 3.000000

5 7
9.000000 7.000000 0.000000 8.000000 6.000000 1.000000 4.000000
5.000000 1.000000 8.000000 2.000000 1.000000 9.000000 5.000000
9.000000 4.000000 6.000000 3.000000 7.000000 2.000000 6.000000
6.000000 2.000000 8.000000 4.000000 7.000000 4.000000 0.000000
8.000000 0.000000 7.000000 9.000000 7.000000 9.000000 7.000000
[emilbengtsson@hornbeam project2_pthreads]$ ./parallel_matrix_matrix_multiply 2
Multiplication done
It took 4 microseconds
[emilbengtsson@hornbeam project2_pthreads]$ cat matrix_matrix_result.txt
178.000000 70.000000 141.000000 103.000000 116.000000 123.000000 125.000000
63.000000 21.000000 71.000000 45.000000 57.000000 52.000000 21.000000
115.000000 46.000000 92.000000 73.000000 103.000000 54.000000 52.000000
98.000000 30.000000 62.000000 86.000000 75.000000 70.000000 71.000000
176.000000 70.000000 159.000000 91.000000 118.000000 125.000000 112.000000
[emilbengtsson@hornbeam project2_pthreads]$
```

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 threads that are used.

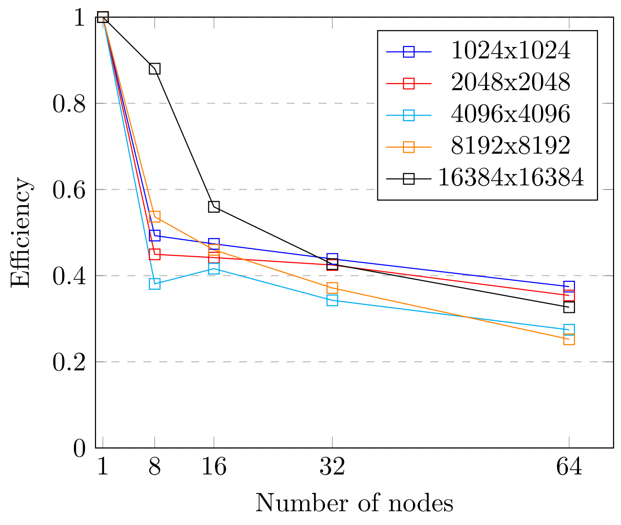
Table of run times (in microseconds)

	1024x1024	2048x2048	4096x4096	8192x8192	16384x16384
1 thread	9899	36041	135801	394492	1493285
8 threads	2510	10023	44570	91853	212022
16 threads	1306	5097	20393	53618	166727
32 threads	705	2652	12384	33193	109356
64 threads	413	1591	7736	24430	71435

Table of efficiencies

	1024x1024	2048x2048	4096x4096	8192x8192	16384x16384
1 thread	1	1	1	1	1
8 threads	0.49298	0.44948	0.38086	0.53685	0.88038
16 threads	0.47373	0.44194	0.41620	0.45984	0.55978
32 threads	0.43879	0.42469	0.34268	0.37140	0.42673
64 threads	0.37451	0.35395	0.27429	0.25231	0.32663

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



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.

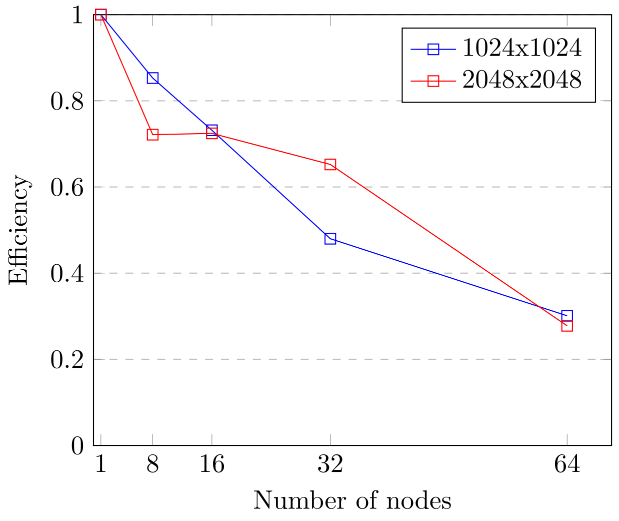
Table of run times (in microseconds)

	1024x1024	2048x2048	4096x4096	8192x8192	16384x16384
1 thread	9558178	263595975	-	-	-
8 threads	1400893	45670520	445046715	-	-
16 threads	816519	22742075	219663284	-	-
32 threads	622381	12628392	111485460	1069132706	-
64 threads	496182	14829012	132574488	1067413674	-

Table of efficiencies

	1024x1024	2048x2048	4096x4096	8192x8192	16384x16384
1 thread	1	1	-	-	-
8 threads	0.85286	0.72146	-	-	-
16 threads	0.73163	0.72442	-	-	-
32 threads	0.47992	0.65229	-	-	-
64 threads	0.30099	0.27775	-	-	-

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



Conclusion

Matrix-Vector Multiplication

The results from the matrix-vector multiplication program quite interesting. The efficiency for the program is not great, as the efficiency at 8 threads is around 0.5 for most of the matrix sizes. If you analyze the data in the tables from a fixed-workload perspective the program seems to generally do poorly when the number of threads are increased from 1 to 8, but after that the efficiency actually stays at a similar level or only drops a little when you increase the number of threads. If you analyze the data in the tables from a fixed-time perspective it does rather good since if you double the amount of threads and the amount of data the run time usually does not increase by that much.

The program actually does not scale that bad, at least with 8 or more threads. The big reduction in efficiency between 1 and 8 threads might be because of the increased overhead that comes with parallelizing the program.

Matrix-Matrix Multiplication

The matrix-matrix multiplication program also produces some interesting data. I could not get the program to run the 4096x4096 matrix-matrix multiplication on only one thread since that took too much time. The even larger matrix sizes took even longer and I could not run the 16384x16384 matrix-matrix multiplication with any number of threads as it took too much time. Because of this I have not calculated the efficiencies for 4096x4096, 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 got some reasonable values.

If you look at the data in the tables from a fixed-workload perspective you can see that the efficiency drops quite a bit almost every time you run the program with more threads, which indicates that there is not much of a strong scaling. It is pretty difficult to look at the data in the tables from a fixed-time perspective, since there is not that much data. But if you take into account that when you double the size of the matrices, you actually increase the problem size by a factor of 16. You can see that it actually doesn't look too bad and that the program scales weakly really well.

Optimization

One optimization I did for both the matrix-matrix and matrix-vector multiplication programs was that I changed how I implemented my barrier from using busy wait to using a mutex and a condition variable. This didn't really improve the performance of the algorithm but it made the barrier a little better.