INF-3201 Parallel programming
Assignment 1 report

Fredrik Stenvoll Nylund

September 3rd, 2022

1 Introduction

The purpose of this assignment was to implement a solution for a CrackMe challenge, with both a sequential solution and a parallel solution. A time performance analysis was done to measure the performance increase of the parallel solution, and compare it with a theoretical estimated performance increase.

2 Design and implementation

2.1 Sequential solution

The implementation works by having a character array with size equal to the length of the password. The program iterates through all possible combinations of these characters and compares every combination with the correct password, and finishes when it guesses the correct combination. For each character in the highest index, that character is combined with each character in the next lower index, which is repeated for every index until the lowest index is reached.

2.2 Parallel solution

The parallel solution is built upon the sequential solution. The parallel solution works by splitting up the distribution of characters in the highest index in the character array. This splits up the workload evenly between the number of processes used, and ensures that all character combinations are covered. The range of characters iterated through by each process in the highest index is calculated by the number of processes and the rank of each process. E.g. if there are two processes running, the highest index of the process with rank 0 will range from char values 0-63, and the highest index of the process with rank 1 will range from char values 64-127. The rest of the indexes in the array of each process will still range from 0-127 to ensure that every character combination is checked.

When a process finds the correct password, it aborts itself and all other processes.

3 Time performance analysis

3.1 Testing environment

All time performance testing have been done on nodes in the UV cluster provided by the university. The testing have been performed on several different nodes with the same hardware specifics. The nodes used are Compute-10-0, Compute-10-1, Compute-10-3 and Compute-10-4. The time performance tests have been done 10 times for each combination of password length and number of processes used, and the average of those results is used.

Hardware specifications:
HP SL230s Gen8, CPU 2, cores 10, processors 20, Xeon E5-2680 v2 @ 2.80GHz, RAM 64GB (16 x 4GB), SW-21, Matrox, DISK 500 GB

3.2 Theoretical speedup

The theoretical speedup of the program can be calculated using Amdahl's law. The formula:

$$S = \frac{1}{(1-p)(\frac{p}{s})}$$

can be used to calculate an estimate of the theoretical speedup of the program. Here, S is the theoretical speedup of the program, p is the fraction of the programs run-time which is parallelizable, and s is the speedup of the part of the program which is parallelizable. For our calculations, we will use the number of processes as s, and p will be set at 0.95. Even though pretty much the whole program is parallelizable, some overhead between multiple processes is taken into account, and therefore 0.95 is used as p[1][P.Pacheco, 2011, page 58]. The number of processes used is 1, 2, 4, 8 and 16.

Number of processes	2	4	8	16
Theoretical speedup(S)	1.82	3.08	4.71	6.4

Table 1: Theoretical speedup

The calculated theoretical speedup with s = number of processes and p = 0.95 can be seen in figure 1.

3.3 Results

3.3.1 Password length of 1

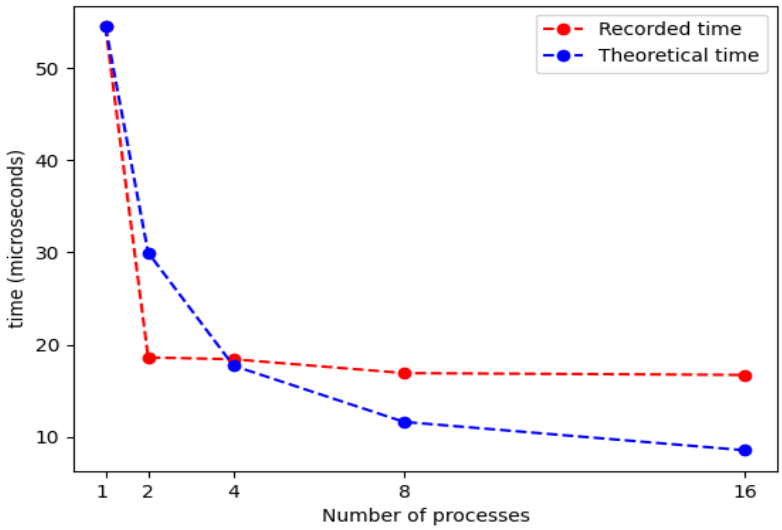

Figure 1: Time measurements on password length: 1

As seen in Figure 1, there is a significant boost in performance when the program is parallelized, when cracking a password of length 1. However, there looks to be almost no difference in performance from running with 2 processes to running with 16 processes. With a password length this short, and therefore a very short time is used to run the parallelized code, the results are likely heavily influenced by overhead and other factors. The recorded speedup values here differs a lot from the calculated theoretical speedup, as seen in Figure 1.

3.3.2 Password length: 2

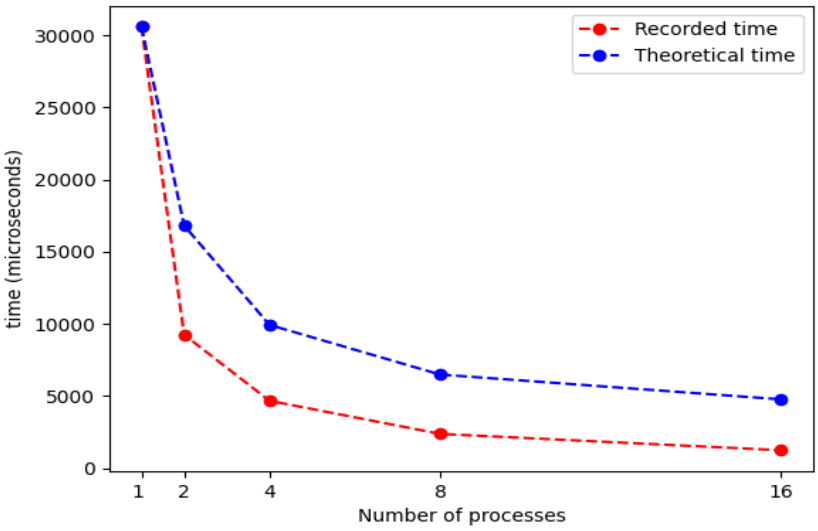

Figure 2: Time measurements on password length: 2

In figure 2, the pattern of recorded speedup is seen to be closer to the calculated theoretical speedup. Here, the recorded speedup between running with 1 and 2 processes is somewhat bigger than the calculated theoretical speedup. However, when running with more processes, the recorded performance increase looks very similar to the theoretical performance increase.

3.3.3 Password length: 3

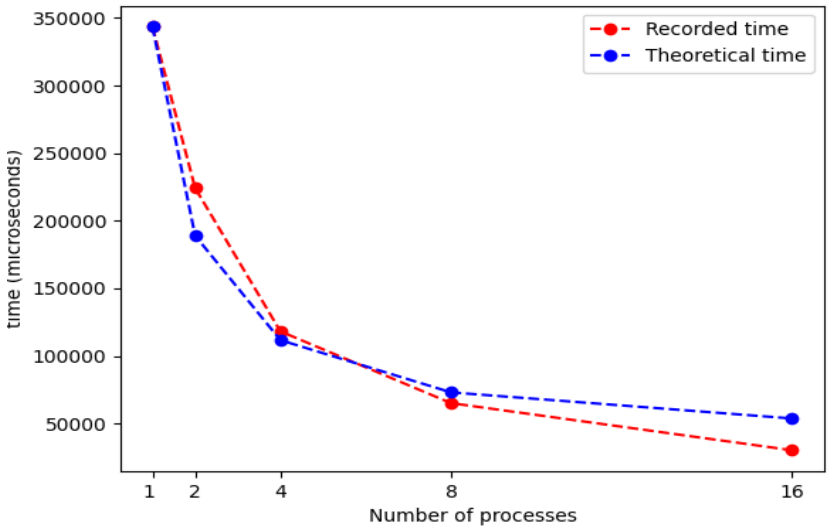

Figure 3: Time measurements on password length: 3

Number of processes	2	4	8	16
Theoretical speedup	1.82	3.08	4.71	6.4
Recorded speedup	1.53	2.91	5.28	11.39

Table 2: Theoretical speedup vs recorded speedup with password length of 3

When running with a password length of 3, there is a lower increase in performance between 1 and 2 processes. However, when using 4, 8 and 16 processes, the recorded increase in performance is greater than the theoretical increase in performance. The recorded results and the theoretical estimates is visualized in Figure 3 and Table 2.

3.3.4 Password length: 4

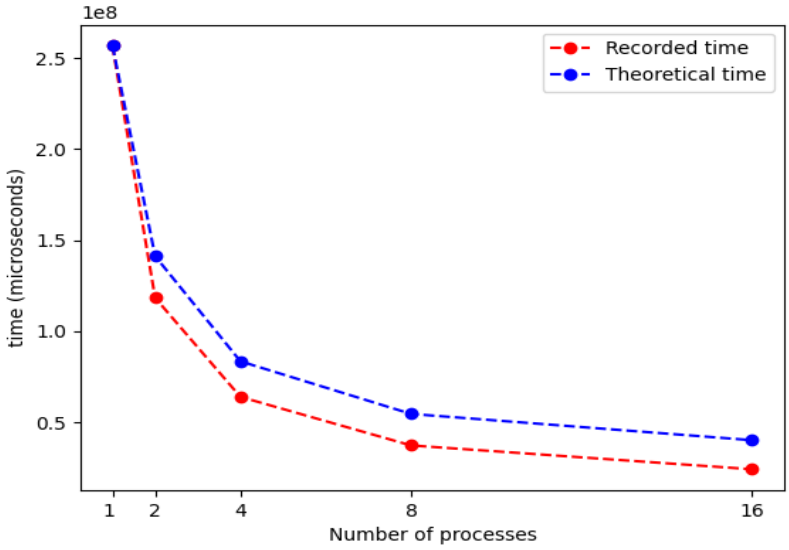

Figure 4: Time measurements on password length: 4

Number of processes	2	4	8	16
Theoretical speedup	1.82	3.08	4.71	6.4
Recorded speedup	2.17	4.02	6.89	10.63

Table 3: Theoretical speedup vs recorded speedup with password length of 4

The recorded speedup with password length of 4 is seen to be slightly greater than the theoretical estimated speedup, as seen in Figure 4 and Table 3.

4 Discussion

4.1 Recorded vs theoretical speedup

As seen in the time performance analysis, the program gets a significant increase in performance when two or more processes executes parallel. The performance increase varies some from the theoretical estimated performance increase calculated by Amdahl's law.

One factor which contributes to the difference from the theoretical estimate

could come from where the correct password combination is relative to where the program splits up the password combinations between processes. If the correct password combination comes early in a process' range of combinations, the performance increase may appear better than it actually is, and vice versa. For a more accurate analysis of performance for related work in the future, this can be taken into account.

Another thing which could be considered is to try different values for p in the theoretical estimate calculations. Since several of the recordings shows a better performance increase than the estimate, it is possible that p = 0.95 could be increased somewhat.

4.2 MPI Abort solution

In the parallel solution in this assignment, the process which guesses the correct password combination uses the MPI Abort call to stop the other processes from executing. This is a simple solution which works for the purpose of this assignment. However, if the program has to use the correct password value in some way after it is correctly guesses, this would not be a possible solution.

4.3 Loaded cluster nodes

Since all the students taking this course has access to the UV cluster, there is a possibility of more than one student is running their program on any given node. There are some outlier results from the time performance testing which could be a result of the used node being used by either more or less other people at a given time. That several students have used the same nodes at the same time is possible, especially since the node used in the performance testing is the nodes with enough cores to test up to 16 processes parallel. However, since the biggest outliers only occurs a couple times, and a average of 10 executions is done for each combination, the given results are still sufficient to analyze the performance increase of parallel programming.

5 Conclusion

In this assignment a solution to a CrackMe challenge has been implemented both sequentially and parallel. The time performance analysis shows a substantial increase in performance when executing parallel.

6 References

[1] - Pacheco, P., 2011. An Introduction to Parallel Programming. 1st ed. Burlington, MA 01803, USA: Elsevier.

7 Appendix A - Time performance test results

All results are represented in microsecond.

7.1 Sequential results

Sequential solution				
	1	**2**	**3**	**4**
1	56	37066	290999	212698519
2	55	37083	425881	249489951
3	56	37092	284915	224859161
4	53	24081	330792	229849347
5	56	18086	353060	275285392
6	54	26088	412107	227910303
7	54	37072	364293	247441323
8	54	32078	363593	243000700
9	54	25078	323338	273978750
10	53	32070	285880	383773858
Average	54.5	30579.4	343485.8	256828730.4

7.2 Parallel results

Parallel solution

2 processes

	1	2	3	4
1	*18*	*9255*	273879	136572406
2	*17*	*9251*	175795	119588752
3	*19*	*9257*	193839	114918382
4	*19*	*9270*	154847	115601776
5	*19*	*9250*	174100	107513633
6	*18*	*9257*	273820	119185986
7	*18*	*9253*	273823	115861295
8	*20*	*9253*	273835	113510312
9	*19*	*9247*	194276	117052232
10	*19*	*9251*	253829	123599936
Average	18.6	9254.4	224204.3	118340471

4 processes

	1	2	3	4
1	18	4681	68607	63494263
2	19	4682	134496	74194420
3	23	4679	135493	60382699
4	19	4691	101508	66163722
5	17	4680	141526	64734522
6	18	4676	117790	59571516
7	17	4679	141512	62896197
8	18	4682	69280	64971184
9	17	4679	135560	62698645
10	18	4688	135460	60195969
Average	18.4	4681.7	118123.2	63930313.7

8 processes

	1	2	3	4
1	16	2386	61348	34907348
2	17	2386	61350	38776007
3	16	2391	60319	38588670
4	17	2387	61349	35880563
5	18	2389	61353	37391463
6	17	2389	56329	37388460
7	17	2388	74339	39108682
8	18	2393	94339	36385333
9	17	2385	54339	37002281
10	16	2390	65340	37254270
Average	16.9	2388.4	65040.5	37268307.7

16 processes

	1	2	3	4
1	16	1243	17260	25580076
2	17	1244	24219	23524033
3	18	1252	44254	24444959
4	17	1244	21257	25543927
5	16	1317	45283	23797611
6	17	1243	17264	24292588
7	16	1246	28250	23813177
8	16	1262	17255	22107762
9	17	1244	46247	23424885
10	17	1246	40254	25075298
Average	16.7	1254.1	30154.3	24160431.6