Results and Analysis

Parallelization of a BruteForce Algorithm

Student ID 10027049

University of the West of England

Parallel Computing UFCFFL-15-M

December 2018

Results, analysis and conclusions from parallelization of a BruteForce algorithm, written in C, using OpenMP and MPI.

1 Contents

2 Results

All of the runtime results are displayed in the attached spreadsheet "Results.xlsx" There were a total of 334 program runs in various modes eg OMP 3 threads and mean execution times calculated. The total runtime for all the tests was 23540 s (approx. 6.5 hrs).

As the key was known, it was possible to position the first character of the key at different positions within the search alphabet and note execution times accordingly. This was purely for testing purposes, and if user input is taken for the ciphertext, plaintext and IV values, the program would find the relevant key.

The spreadsheet of raw data displays all of the results of the various positions of the first character. Positions 1,2,3,4,8,16, and 36 (the end of the search alphabet) were tested, as well as an exhaustive search where the search alphabet was adjusted to not include the first character.

Due to time constraints for testing, some of the longer searches were only run 2 or 3 times, rather than the planned 5. All program runs executed on UWE cluster 164.11.39.11 from a Lenovo Laptop (4 cores, i7 processor)

Table 1 shows a summary of the results and the associated speed-up times and efficiencies. The speed up times have been calclulated by dividing the benchmarked serial time by the mean execution time of the relevant program, and the efficiency figure is derived by dividing the speedup by the number of threads/processors.

Table 1

		Position of first character of key within search alphabet of length 36			
Version of program		**8**	**16**	**36**	**Exhaustive**
Serial	Mean Time	92.31	197.69	461.21	474.45
OMP 1 thread	Mean Time	104.92	226.22	522.80	542.41
	Speed-up	0.88	0.87	0.88	0.87
	Efficiency	***0.88***	***0.87***	***0.88***	***0.87***
MPI 1 proc	Mean Time	91.65	195.86	468.45	515.89
	Speed-up	1.01	1.01	0.98	0.92
	Efficiency	1.01	1.01	0.98	0.92
OMP 2 threads	Mean Time	68.56	168.34	384.64	411.14
	Speed-up	1.35	1.17	1.20	1.15
	Efficiency	***0.67***	***0.59***	***0.60***	***0.58***
MPI 2 procs	Mean Time	40.56	94.23	228.46	245.01
	Speed-up	2.28	2.10	2.02	1.94
	Efficiency	1.14	1.05	1.01	0.97
OMP 3 threads	Mean Time	59.40	146.09	335.75	364.01
	Speed-up	1.55	1.35	1.37	1.30
	Efficiency	***0.52***	***0.45***	***0.46***	***0.43***
MPI 3 procs	Mean Time	27.68	68.21	149.70	162.55
	Speed-up	3.33	2.90	3.08	2.92
	Efficiency	1.11	0.97	1.03	0.97
OMP 4 threads	Mean Time	35.81	106.11	283.72	304.87
	Speed-up	2.58	1.86	1.63	1.56
	Efficiency	***0.64***	***0.47***	***0.41***	***0.39***
MPI 4 procs	Mean Time	14.91	42.28	111.13	124.82
	Speed-up	6.19	4.68	4.15	3.80
	Efficiency	***1.55***	***1.17***	***1.04***	***0.95***
MPI 5 procs	Mean Time	17.42	52.50	121.30	134.10
	Speed-up	5.30	3.77	3.80	3.54
	Efficiency	***1.06***	***0.75***	***0.76***	***0.71***
MPI 6 procs	Mean Time	21.08	40.57	100.90	122.35
	Speed-up	4.38	4.87	4.57	3.88
	Efficiency	***0.73***	***0.81***	***0.76***	***0.65***
MPI 7 procs	Mean Time	24.13	49.26	119.65	133.89
	Speed-up	3.83	4.01	3.85	3.54
	Efficiency	***0.55***	***0.57***	***0.55***	***0.51***
MPI 8 procs	Mean Time	0.72	28.49	111.32	124.82
	Speed-up	127.42	6.94	4.14	3.80
	Efficiency	***15.93***	***0.87***	***0.52***	***0.48***

Figure 1 show the results of the various program runs for different positions of the first key character:

Exhaustive Search (character not in the alphabet, so key not found).

End of alphabet (character is at the end of the search alphabet, in this case position 36).

Posn 16 (normal position of the first character of the key within the search alphabet).

Posn 8 (first character positioned towards the beginning of the search alphabet

Figure 1

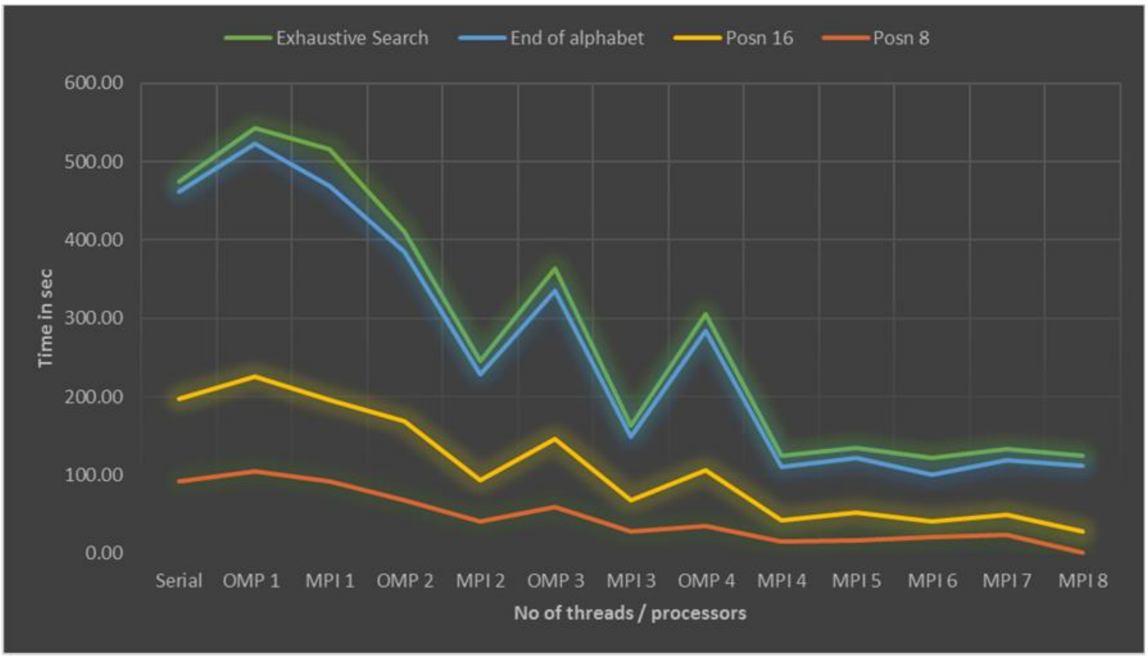

Figure 2 shows a graph of the efficiencies for when the key was at the end of the search space. i.e. "p" was at position 36 in a search alphabet of length 36.

Figure 2

3.1 Amdahl’s Law & Gustafson’s Law

It would have been beneficial to analyse the results in terms of Amdahl’s law and Gustafson’s Law, which both determine the speedup achievable by a parallel program in terms of the proportion of time spent on the serial execution versus the time spent on the execution of the parallel sections of the program. However, in this case, all of the program execution time was within the parallel section, bar that time used for taking user input.

3.2 General analysis

3.2.1 Mean time and Speedup Factor

There is a general reduction in time for all 3 tests as the number of threads/processors increase. However, there are spikes in the OMP 3 thread and OMP 4 thread results, indicating slower times. This is probably due how the search alphabet is distributed among the threads. It is also clear from Figure 1 that there is no significant decrease in time when the number of processors is greater than 4. The exception to this is when the number of processors in the MPI run for the End of Alphabet search was 6. This produced the best overall time for that particular test. MPI parallelization using 8 processes when the first character was positioned at position 8 produced a very quick result (0.72s), which was a speed up of a factor of 127.42 over the serial version. Again, this was because of the way the search alphabet was distributed between the processes. Position 8 would be among the first batch to be processed and hence a very quick result.

3.2.2 Efficiency

The efficiencies were calculated for all of the mean speedup factors, and charted for when the key was at the end of the search space. i.e. when $p = 36$. The theoretical maximum efficiency is 1 (100%). However, some of the efficiencies exceeded this. This was probably due to how the data was distributed in both the MPI and OMP versions.

For example, in the OMP version executed using 4 threads, then all keys beginning with a, b, c, or d would all be generated at the same time. One thread would use “a” as the key starting character, another would use “b”, and so on. Thus, batches of 4 are processed together.

If the search space (size of alphabet) is divisible by the no. of threads or processors then each batch will be the same size. If this is not the case, particularly in the MPI version, this may slow the execution time.

4 Conclusions

Data distribution was a major factor in the performance of the parallel versions of the BruteForce program. The parallel versions produced the expected results. Overheads, in terms of communication time between the processes in the MPI version also effected the results obtained.

Further work is recommended on the MPI version of the program. Although it compiles, runs successfully uses multiple processors, and produces results that are an improvement on the serial version, it is still inefficient, particularly when the maximum number of processors available, 8, did not produce the expected high performance. Running on 8 processors was little better than running on 4.

Lack of experience in using MPI and lack of time to gain that experience was another major factor which effected the results.

5 Raw data

Table 2 and Table 3 display all the obtained timings.

Table 2 (Serial and OMP results)

Program Version	No of processors or threads	Posn of p	Run1	Run2	Run3	Run4	Run5	Mean Time (s)	Mean Count	Count 1	Count 2	Count3
Serial		1	0.6966	0.6948	0.7017	0.6985	0.6973	**0.70**	**3,166,925**	3166925	3166925	3166925
		2	13.6198	16.2965	13.4889	13.6164	13.4662	**14.10**	**61,953,485**	61953485	61953485	61953485
		3	27.2633	26.5803	26.8435	27.5186		**27.05**	**122,419,661**	122419661	122419661	122419661
		4	39.6554	39.7008	39.7608			**39.71**	**182,885,837**	182885837	182885837	182885837
		8	92.2113	92.1902	92.5307			**92.31**	**424,750,540**	424750540	424750540	424750540
		16	197.7609	197.7441	197.5756			**197.69**	**908,479,948**	908479948	908479948	908479948
		36	461.4178	461.0027				**461.21**	**2,117,755,480**	2117755480	2117755480	
		exhaustive	474.9618	473.9433				**474.45**	**2,176,782,336**	2176782336	2176782336	
OMP	**1 thread**	1	0.79522	0.78819	0.79262	0.7902	0.79206	**0.79**	**3,166,925**	3,166,925	3,166,925	3,166,925
		2	15.296	15.343	15.324	15.321	15.734	**15.40**	**61,953,485**	61,953,485	61,953,485	61,953,485
		3	30.372	30.722	30.224			**30.44**	**122,419,661**	122,419,661	122,419,661	122,419,661
		4	45.733	46.396	45.144			**45.76**	**182,885,837**	182,885,837	182,885,837	182,885,837
		8	105.16	104.82	104.79			**104.92**	**424,750,540**	424,750,540	424,750,540	424,750,540
		16	224.32	224.32	230.03			**226.22**	**908,479,948**	908,479,948	908,479,948	908,479,948
		36	522.77	522.82				**522.80**	**2,117,755,480**	2,117,755,480	2,117,755,480	
		exhaustive	538.8	546.02				**542.41**	**2,176,782,336**	2,176,782,336	2,176,782,336	
	2 threads	1	1.1859	1.1929	1.1855	1.2123	1.1961	**1.19**	**6,296,407**	6,333,103	6,269,916	6,286,202
		2	0.56475	0.55843	0.59266	0.55906	0.5552	**0.57**	**2,969,535**	2,968,780	2,980,297	2,959,527
		3	22.676	22.858	22.626	22.676	23.716	**22.91**	**123,515,810**	123,809,635	122,918,548	123,819,248
		4	23.06	24.635	24.01	26.987	23.304	**24.40**	**124,190,264**	123,991,578	123,673,718	124,905,496
		8	68.46	68.336	67.265	70.55	68.167	**68.56**	**366,349,425**	367,150,731	365,776,491	366,121,053
		16	172.61	160.29	172.12			**168.34**	**845,393,634**	837,478,190	852,613,188	846,089,525
		36	386.03	385.53	382.35			**384.64**	**2,066,682,808**	2,067,118,832	2,071,628,102	2,061,301,489
		exhaustive	408.9	418.15	406.38			**411.14**	**2,176,782,336**	2,176,782,336	2,176,782,336	2,176,782,336
	3 threads	1	1.5527	1.514	1.5503	1.4988	1.5377	**1.53**	**8,379,501**	8,305,977	8,110,673	8,721,852
		2	0.75696	0.75205	0.71733	0.72767	0.78491	**0.75**	**4,164,265**	4,234,331	4,014,126	4,244,337
		3	0.71526	0.73994	0.73239	0.7148	0.73638	**0.73**	**4,090,392**	4,138,107	4,305,843	3,827,227
		4	29.388	30.274	30.391	29.404	29.859	**29.86**	**170,531,796**	173,134,164	165,031,163	173,430,060
		8	60.275	58.782	58.471	59.523	59.947	**59.40**	**332,592,350**	331,538,072	341,588,782	324,650,197
		16	145.42	147.46	146.9	145.57	145.11	**146.09**	**846,357,382**	845,633,622	863,397,444	830,041,079
		36	337.23	330.68	339.35			**335.75**	**1,924,460,726**	1,921,138,307	1,905,044,314	1,947,199,556
		exhaustive	360.55	363.54	367.93			**364.01**	**2,176,782,336**	2,176,782,336	2,176,782,336	2,176,782,336
	4 threads	1	1.7315	1.7199	1.7302	1.7335	1.7287	**1.73**	**10,791,724**			
		2	0.81446	0.99138	0.8014	0.80708	0.81322	**0.85**	**5,244,124**			
		3	0.81854	0.79409	0.82157	0.79433	0.77678	**0.80**	**4,887,542**			
		4	0.88126	0.80702	0.80699	0.87651	0.87086	**0.85**	**5,174,164**			
		8	35.885	34.205	36.926	37.346	34.665	**35.81**	**221,207,260**			
		16	106.26	109.42	106.5	101.57	106.8	**106.11**	**670,777,631**			
		36	284.26	286.43	281.06	285.65	281.19	**283.72**	**1,883,212,272**			
		exhaustive	308.84	301.52	304.26			**304.87**	**2,176,782,336**			

Table 3 (MPI Results)

Program Version	No of processors or threads	Posn of p	Run1	Run2	Run3	Run4	Run5	Mean Time (s)	Mean Count	Count 1	Count 2	Count3
MPI	1 proc	1	0.768398	0.689645	0.68732			**0.72**	**3,166,925**			
		2	13.36349	14.17904	13.36216			**13.63**	**61,953,485**			
		3	26.3393	26.38305	26.40797			**26.38**	**122,419,661**			
		4	39.35706	39.40234	39.56521			**39.44**	**182,885,837**			
		8	91.48913	91.98718	91.47026			**91.65**	**424,750,540**			
		16	196.2155	195.6466	195.7246			**195.86**	**908,479,948**			
		36	460.4248	476.4738				**468.45**	**2,117,755,480**			
		exhaustive	515.8911					**515.89**	**2,176,782,336**			
	2 procs	1	0.706138	0.746487	0.704952			**0.72**	**3,166,925**			
		2	0.332666	0.333461	0.333499			**0.33**	**1,487,309**			
		3	13.75085	14.80422	13.74726			**14.10**	**61,953,485**			
		4	13.74297	13.91009	13.75808			**13.80**	**61,953,485**			
		8	40.59607	40.57155	40.51024			**40.56**	**182,885,836**			
		16	94.21589	94.44507	94.02402			**94.23**	**424,750,540**			
		36	228.638	228.2808				**228.46**	**1,029,364,312**			
		exhaustive	245.0074					**245.01**				
	3 procs	1	0.71032	0.711189	0.714607			**0.71**	**3,166,925**			
		2	0.334228	0.333145	0.33418			**0.33**	**1,487,309**			
		3	0.333831	0.334199	0.334147			**0.33**	**1,487,309**			
		4	13.94175	13.87485	13.90437			**13.91**	**61,953,485**			
		8	27.51158	28.06831	27.4609			**27.68**	**122,419,660**			
		16	68.31525	68.16699	68.15397			**68.21**	**303,818,188**			
		36	149.8516	149.5476				**149.70**	**666,567,256**			
		exhaustive	162.5546					**162.55**	**725,594,112**			
	4 procs	1	0.727761	0.750773	0.738594			**0.74**	**3,166,925**			
		2	0.337682	0.34439	0.340914			**0.34**	**1,487,309**			
		3	0.337645	0.337991	0.340757			**0.34**	**1,487,309**			
		4	0.338109	0.35383	0.340749			**0.34**	**1,487,309**			
		8	14.08414	14.57588	16.07193			**14.91**	**61,953,484**			
		16	41.5167	42.69525	42.63943			**42.28**	**182,885,836**			
		36	112.1576	110.1056				**111.13**	**485,168,728**			
		exhaustive	124.8209					**124.82**	**544,195,584**			
	5 procs	1	1.204825	0.962179	1.435989			**1.20**	**3,166,925**			
		2	0.345536	0.541452	0.417733			**0.43**	**1,487,309**			
		3	0.457666	0.517274	0.344432			**0.44**	**1,487,309**			
		4	0.425101	0.33711	0.655874			**0.47**	**1,487,309**			
		8	16.15433	17.75586	18.35608			**17.42**	**61,953,484**			
		16	56.79571	51.22479	49.47918			**52.50**	**182,885,836**			
		36	120.3621	122.2373				**121.30**	**424,702,552**			
		exhaustive	134.0972					**134.10**	**483,729,408**			
	6 procs	1	1.16992	0.995297	1.105558			**1.09**	**3,166,925**			
		2	0.422131	0.61605	0.66695			**0.57**	**1,487,309**			
		3	0.402147	0.668302	0.423609			**0.50**	**1,487,309**			
		4	0.415148	0.66035	0.4909			**0.52**	**1,487,309**			
		8	21.58406	20.86207	20.79847			**21.08**	**61,953,484**			
		16	40.55182	40.45869	40.71363			**40.57**	**122,419,660**			
		36	100.9222	100.8817				**100.90**	**303,770,200**			
		exhaustive	122.3512					**122.35**	**362,797,056**			
	7 procs	1	1.058003	0.987399	1.499253			**1.18**	**3,166,925**			
		2	0.68131	0.697367	0.68555			**0.69**	**1,487,309**			
		3	0.71069	0.689936	0.444216			**0.61**	**1,487,309**			
		4	0.384869	0.696285	0.692383			**0.59**	**1,487,309**			
		8	23.50498	24.24896	24.63445			**24.13**	**61,953,484**			
		16	50.01321	49.61092	48.15976			**49.26**	**122,419,660**			
		36	119.1086	120.187				**119.65**	**303,770,200**			
		exhaustive	133.8863					**133.89**	**362,797,056**			
	8 procs	1	1.497889	1.52364	1.446222			**1.49**	**3,166,925**			
		2	0.728921	0.701961	0.695412			**0.71**	**1,487,309**			
		3	0.700729	0.688372	0.759351			**0.72**	**1,487,309**			
		4	0.601441	0.713404	0.708346			**0.67**	**1,487,309**			
		8	0.697114	0.696355	0.77999			**0.72**	**1,487,308**			
		16	28.22435	28.54018	28.70439			**28.49**	**61,953,484**			
		36	111.3138	111.3317				**111.32**	**243,304,024**			
		exhaustive	124.8243					**124.82**	**302,330,880**			