Marcus Blaisdell
CptS 411
Programming Project 2

Cpt S 411 Assignment Cover Sheet

(To be turned in along with each homework and program project submission)

Assignment # 2

For individual assignments:

Student name (Last, First): Blaisdell, Marcus

For team projects:

List of all students (Last, First):

List of collaborative personnel (excluding team participants):

I[1] certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received or given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment. I also certify that I have not referred to online solutions that may be available on the web or sought the help of other students outside the class, in preparing my solution. I attest that the solution is my own and if evidence is found to the contrary, I understand that I will be subject to the academic dishonesty policy as outlined in the course syllabus.

Please print your names.
Marcus Blaisdell

Assignment Project Participant(s):
Marcus Blaisdell

Today's Date: October 4, 2019

1 If you worked as a team, then the word "I" includes yourself and your team members.

Marcus Blaisdell
CptS 411
Programming Project 2

Summary:
Due to an error, I am missing some numbers for the NaiveReduce, and MyReduce functions for 2 and 4 processes.

These results do not match my expectations. I expected that as the data sizes increased, the parallel processing time would decrease, and the speedup would improve for larger numbers of processes but this was not what I observed. I consider this data to be suspect and in need of review.
The results are documented below.

One observation that was consistent with expectation is that as the data size increases, the overall time required to process the data increases. This was clearly defined for smaller numbers of processes but as the number of processes increased, the trend became less defined and did not show a clear increase.

A direct comparison of runtimes for all three algorithms shows that NaïveReduce has the best performance with 8 processes. MyReduce function has the best performance for 16, 32, and 64 processes. I had expected that the built-in MPI_Allreduce function would outperform both the Naïve and the MyReduce functions but this did not happen.

Since the results did not match my expectations, I need to review my logic and consult with an expert to determine where my errors are.

Running the code:
The code uses multiple programs and scripts to evaluate the functions.
The functions are defined in two.c and two.h. the main.c program contains the main function and calls the individual functions: NaiveReduce, MyReduce, and MPILibraryReduce.
Due to the way that MPI runs, I had to comment out two of the three functions and focus on running only one for each instance.
The timing is determined from when the function is called to when the function terminates. This does include the time to generate the arrays and may be contributing to the odd results I am seeing.
Since each process is returning its own time value, I maintain a running sum to capture the total time elapsed and since it is mixed in with all of the process times, I wrote a python program, "parser.py", to parse the times and capture the max values and report them to the command line where I copy them into MS Excel for evaluation.
I experimented with trying to get mpirun to run each set of processes sequentially but was unable to get this to work and was forced to initiate each run for p processes individually.
I was able to create a unix script to sequentially run the main program for each data size.
"loopRun.sh" contains the script that automated this part of the process.
It cleans out old slurm files and initiates an individual run of the mpi c code for each data size.
"runit" and "runit2" are helper scripts.

“runit” was used for development and cleans the slurm files, compiles the latest c code, and initiates an mpirun session.
“runit2” is used for testing and initiates the mpirun session for the current data size.

Parallel runtime:

Parallel Runtime (n/p) - NaiveReduce						
1	2	4	8	16	32	64
9.2			90.45	52.425	48.13125	1521.325
14.2			85.6	63.3375	25.29375	1462.70938
24.2			88.025	37.625	41.61875	1423.41875
47.4			72.475	35.5	39.56875	575.94375
90.2			319.8	35.4375	20.4125	1340.15625
178.6			142.1	33.575	20.5375	700.144531
378.2			95.125	41.4375	45.36875	566.140625
777.8			94.975	43.825	22.6625	1418.25
1453.4			75.8	22.825	56.0375	903.03125
2899.4			99.975	17.025	20.83125	856.175
5844			242.825	57.55	78.25	1357.425

Figure 1: Parallel runtime for the NaiveReduce function

Parallel Runtime (n/p) - MyReduce						
1	2	4	8	16	32	64
0.2			140.75	28.7875	16.3125	1070.8375
0.4			117.3	29.0125	11.46875	773.703125
0.2			55.1	20	16.8625	560.415625
0.6			153.15	31.5375	23.10625	526.934375
0.2			135.475	30.4625	38.6125	639.275
0.4			273.675	32.4125	54.30625	541.428125
0.2			86.725	39.625	41.30625	630.978125
0.2			117.3	30.45	16.13125	859.046875
0			191.675	32.2	12.95	917.775
0.6			613.675	46.025	28.3125	1104.71563

Figure 2: Parallel runtime for the MyReduce function

Parallel Runtime (n/p) - MPILibraryReduce						
1	2	4	8	16	32	64
12.8	46.2	263.65	207.45	21.9625	17.00625	507.05625
18.4	52.3	250.9	142.725	25.1875	17.98125	1129.73125
27.8	75.8	273.45	177.825	12.825	72.7875	628.315625
48.8	72.9	295.75	128.225	43.875	11.8375	797.45
93.8	97.6	306.45	144.9	44.0125	47.75	864.3875
180.6	152.6	308.45	99.025	32.9125	25.28125	838.9
369.8	270.4	430.3	195.075	37.325	61.04375	941.20625
723.2	445.2	365.2	275.1	45.5	15.83125	1229.6
4100.4	897.6	1124.55	369.825	54.6625	27.00625	1190.38438
2909.4	6632.9	1802.9	837.95	56.8375	50.65	891.334375
5822.2	2257.1	2626.85	1353.975	46.6125	145.05625	1405.57813

Figure 3: Parallel runtime for the MPILibraryReduce function

Speedup:

Speedup (serial/parallel) - NaiveReduce					
2	4	8	16	32	64
		0.10171365	0.17548879	0.19114401	0.00604736
		0.16588785	0.22419578	0.56140351	0.00970801
		0.2749219	0.64318937	0.58146869	0.01700132
		0.65401863	1.33521127	1.19791502	0.0822997
		0.28205128	2.54532628	4.41886099	0.06730558
		1.25686137	5.3194341	8.69628728	0.25509019
		3.97582129	9.12699849	8.33613445	0.6680319
		8.18952356	17.7478608	34.3210149	0.54842235
		19.1741425	63.6757941	25.9362034	1.60946811
		29.0012503	170.302496	139.185119	3.38645721
		24.0667147	101.546481	74.6837061	4.30521023

Figure 4: Speedup for NaïveReduce function

Speedup (serial/parallel) - MyReduce					
2	4	8	16	32	64
		0.00142096	0.00694746	0.01226054	0.00018677
		0.00341006	0.01378716	0.03487738	0.00051699
		0.00362976	0.01	0.01186064	0.00035688
		0.00391773	0.01902497	0.025967	0.00113866
		0.00147629	0.00656545	0.00517967	0.00031285
		0.00146159	0.01234092	0.00736563	0.00073879
		0.00230614	0.00504732	0.00484188	0.00031697
		0.00170503	0.00656814	0.0123983	0.00023282
		0	0	0	0
		0.00097772	0.01303639	0.02119205	0.00054313

Figure 5: Speedup for MyReduce function

Speedup (serial/parallel) - MPILibraryReduce					
2	4	8	16	32	64
0.27705628	0.27705628	0.06170161	0.58281161	0.75266446	0.02524375
0.35181644	0.35181644	0.12891925	0.73052109	1.02328815	0.01628706
0.36675462	0.36675462	0.15633347	2.16764133	0.38193371	0.04424528
0.66941015	0.66941015	0.38058101	1.11225071	4.12249208	0.06119506
0.96106557	0.96106557	0.647343	2.13121272	1.96439791	0.10851615
1.18348624	1.18348624	1.82378187	5.48727687	7.14363412	0.21528192
1.36760355	1.36760355	1.89568115	9.90756865	6.05795024	0.39290007
1.62443845	1.62443845	2.62886223	15.8945055	45.6818002	0.58815875
4.56818182	4.56818182	11.0874062	75.0130345	151.83152	3.44460166
0.43863167	0.43863167	3.47204487	51.1880361	57.4412636	3.26409491
2.57950467	2.57950467	4.3000794	124.906409	40.1375329	4.14221017

Figure 6: Speedup for MPILibraryReduce function

Efficiency:

Efficiency (speedup/processes) - NaiveReduce					
2	4	8	16	32	64
		0.01271421	0.01096805	0.00597325	9.449E-05
		0.02073598	0.01401224	0.01754386	0.00015169
		0.03436524	0.04019934	0.0181709	0.00026565
		0.08175233	0.0834507	0.03743484	0.00128593
		0.03525641	0.15908289	0.13808941	0.00105165
		0.15710767	0.33246463	0.27175898	0.00398578
		0.49697766	0.57043741	0.2605042	0.010438
		1.02369044	1.1092413	1.07253172	0.0085691
		2.39676781	3.97973713	0.81050636	0.02514794
		3.62515629	10.643906	4.34953495	0.05291339
		3.00833934	6.34665508	2.33386581	0.06726891

Figure 7: Efficiency for NaïveReduce function

Efficiency (speedup/processes) - MyReduce					
2	4	8	16	32	64
		0.00017762	0.00043422	0.00038314	2.9183E-06
		0.00042626	0.0008617	0.00108992	8.078E-06
		0.00045372	0.000625	0.00037064	5.5762E-06
		0.00048972	0.00118906	0.00081147	1.7792E-05
		0.00018454	0.00041034	0.00016186	4.8884E-06
		0.0001827	0.00077131	0.00023018	1.1544E-05
		0.00028827	0.00031546	0.00015131	4.9526E-06
		0.00021313	0.00041051	0.00038745	3.6378E-06
		0	0	0	0
		0.00012221	0.00081477	0.00066225	8.4863E-06

Figure 8: Efficiency for MyReduce function

Efficiency (speedup/processes) - MPILibraryReduce					
2	4	8	16	32	64
0.13852814	0.06926407	0.0077127	0.03642573	0.02352076	0.00039443
0.17590822	0.08795411	0.01611491	0.04565757	0.03197775	0.00025449
0.18337731	0.09168865	0.01954168	0.13547758	0.01193543	0.00069133
0.33470508	0.16735254	0.04757263	0.06951567	0.12882788	0.00095617
0.48053279	0.24026639	0.08091787	0.1332008	0.06138743	0.00169556
0.59174312	0.29587156	0.22797273	0.3429548	0.22323857	0.00336378
0.68380178	0.34190089	0.23696014	0.61922304	0.18931095	0.00613906
0.81221923	0.40610961	0.32860778	0.99340659	1.42755626	0.00918998
2.28409091	1.14204545	1.38592578	4.68831466	4.74473502	0.0538219
0.21931583	0.10965792	0.43400561	3.19925225	1.79503949	0.05100148
1.28975234	0.64487617	0.53750992	7.80665058	1.2542979	0.06472203

Figure 9: Efficiency for MPILibraryReduce function

Run speed by process count for multiple data sizes:

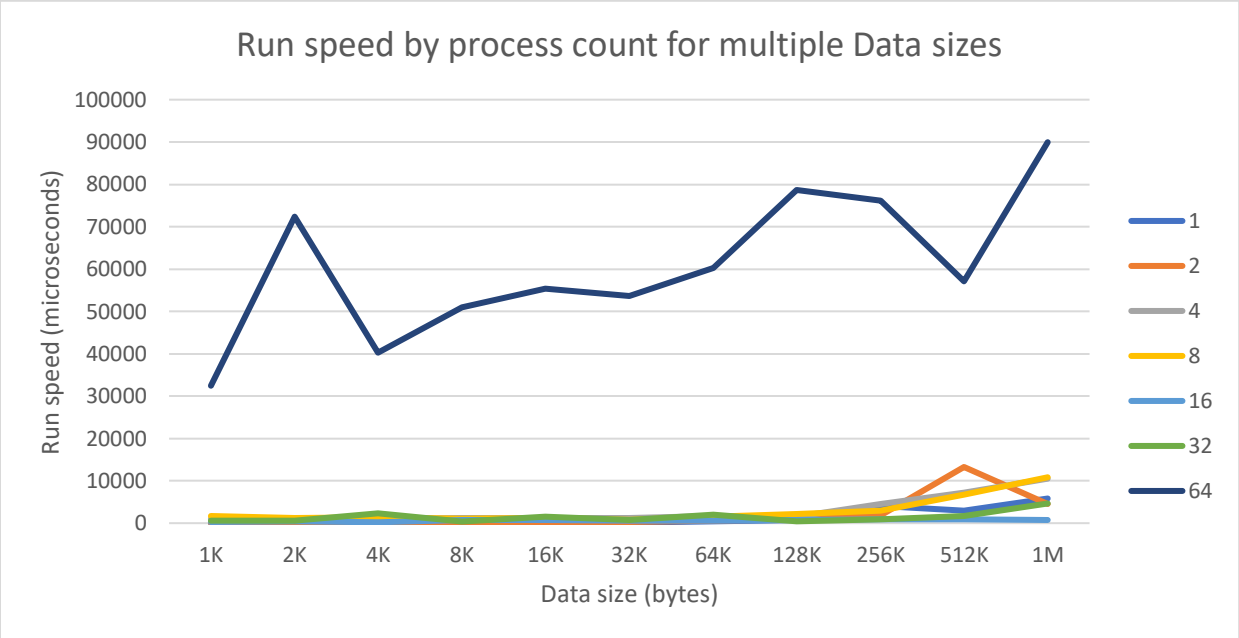
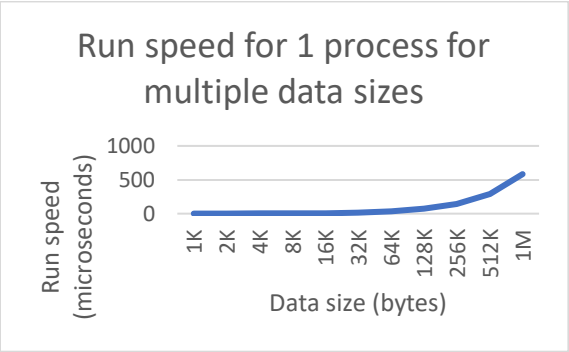


Chart 1: The collective run speed by process count for all data sizes.



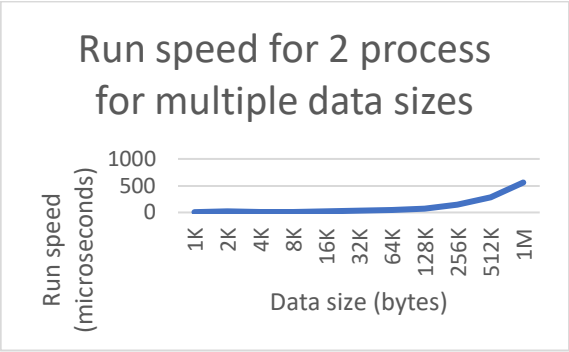
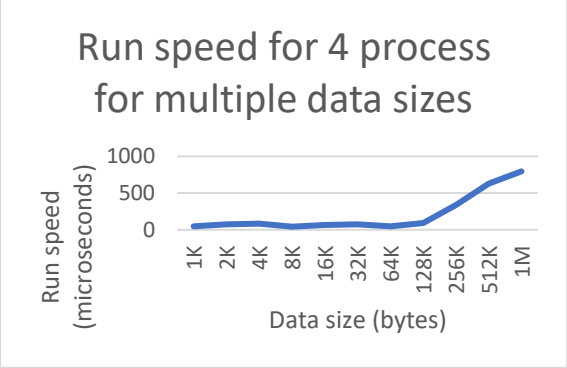


Chart 2, 3: Run speed chart for 1 and 2 processes



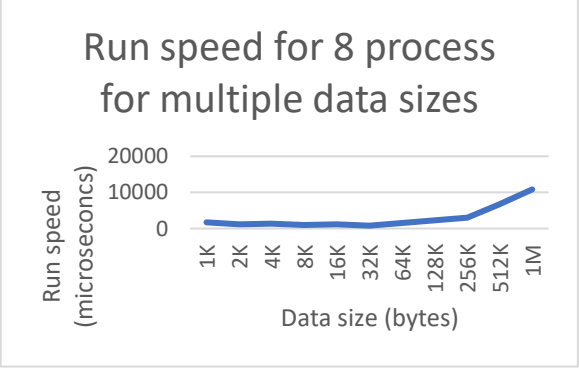
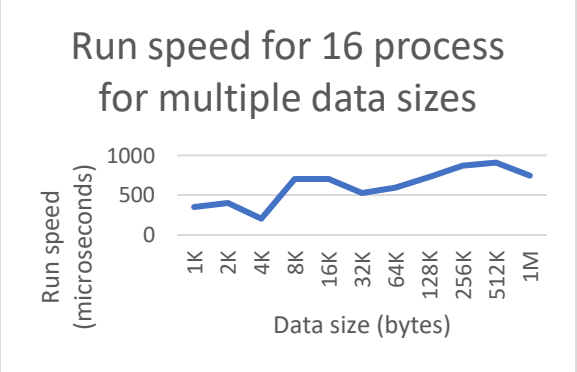


Chart 4, 5: Run speed chart for 4 and 8 processes



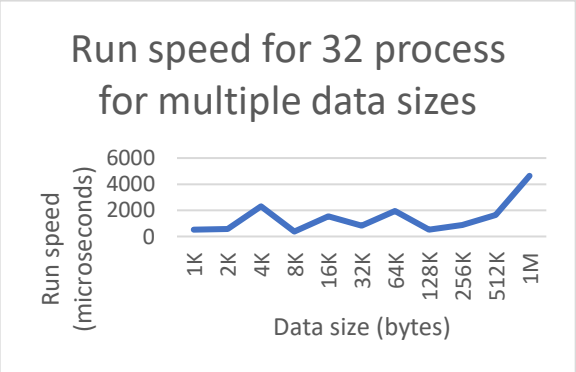


Chart 6, 7: Run speed chart for 16 and 32 processes

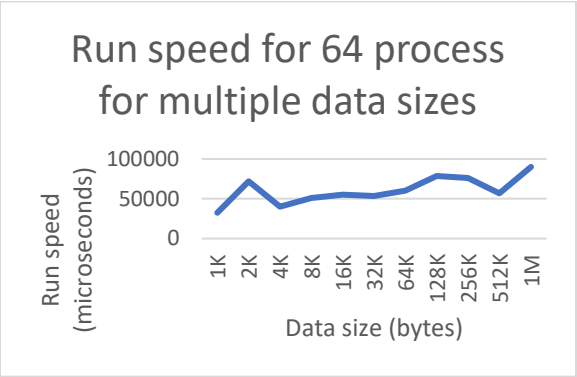


Chart 8: Run speed chart for 64 processes

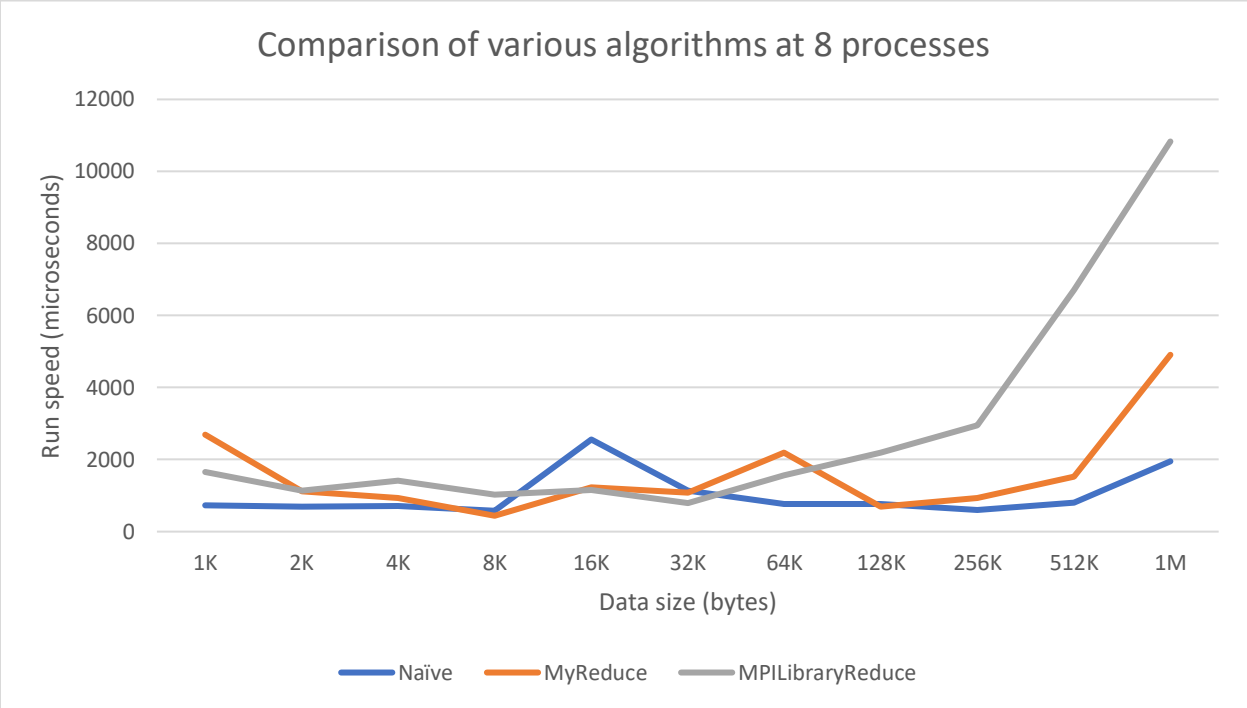


Chart 9: Comparison of the runspeed times for all algorithms at 8 processes. Notice that NaïveReduce performs better than the other algorithms.

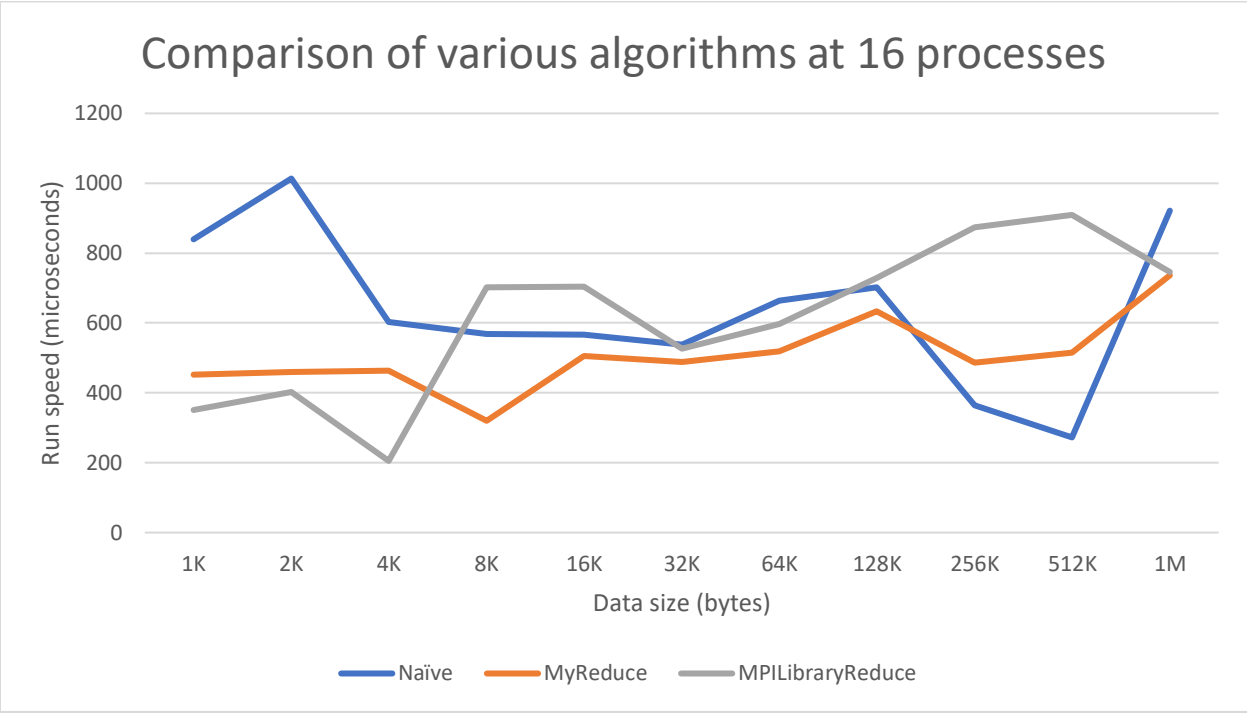


Chart 10: Comparison of the runspeed times for all algorithms at 16 processes. Notice that MyReduce performs better than the other algorithms.

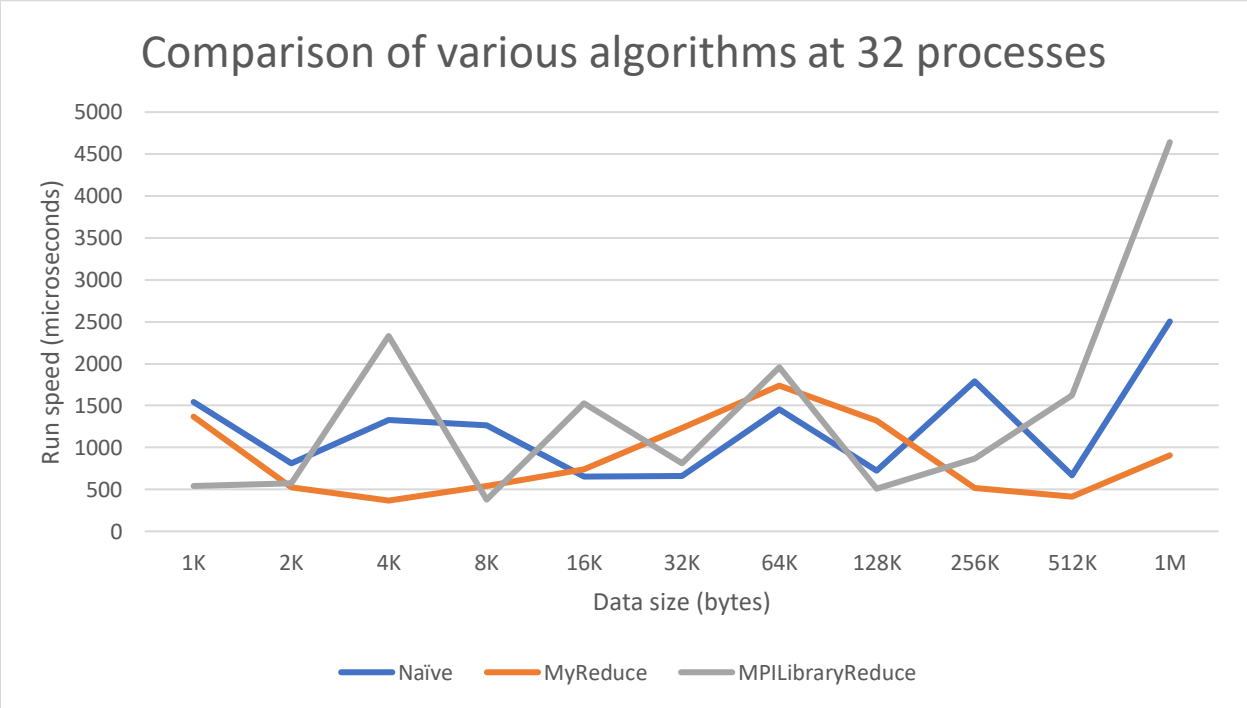


Chart 11: Comparison of the runspeed times for all algorithms at 32 processes. Notice that MyReduce performs better than the other algorithms.

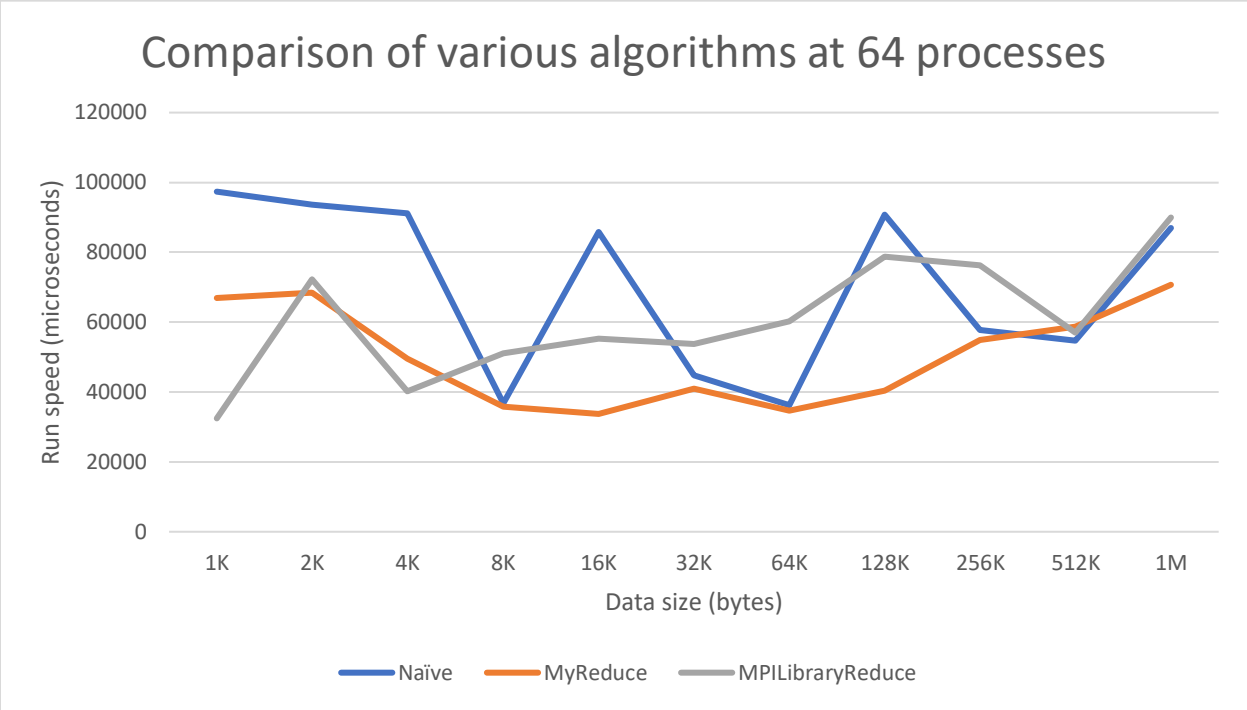


Chart 12: Comparison of the runspeed times for all algorithms at 64 processes. Notice that MyReduce performs better than the other algorithms.