CPSC 524: Parallel Programming Techniques

Assignment 3

Information on building and running

1. Modules

a) Loaded Module List

1) StdEnv (S) 2) Langs/Intel/2015_update2 3) MPI/OpenMPI/2.1.1-intel15

b) Intel C/C++ Compiler (ICC Version 15.0.2 20150121)
/apps/hpc.rhel7/MPI/OpenMPI/2.1.1-intel15/bin/mpicc

2. Commands

On the Grace cluster, use ***sbatch*** to run the scripts in scripts in ***/scripts*** folder to execute each task separately.

3. Outputs

The output of each script is in ***/out*** folder, with the same name of corresponding script.

Task 1: Serial Program

1. Command

> *sbatch /scripts/serial.sh*

2. Result

By execute above command 5 times, we got the following average results:

	N = 1000	**N = 2000**	**N = 4000**	**N = 8000**
1#	0.1513	1.1693	13.935	120.1668
2#	0.1535	1.3936	14.188	124.1491
3#	0.1519	1.2088	14.8568	122.3154
4#	0.1538	1.169	14.977	123.6939
5#	0.1521	1.1943	14.62	120.3718
Avg.	0.15252	1.227	14.51536	122.1394

Task 2A: Blocking MPI Parallel Program (Part A)

1. Command

> *sbatch /scripts/task2a-1.sh && sbatch /scripts/task2a-2.sh && sbatch /scripts/task2a-4.sh && sbatch /scripts/task2a-8.sh*

2. Detailed Results

P = 1, N = 1000		**P = 1, N = 2000**	
Rank	Time	Rank	Time
0	0.1551	0	1.4126
F-Norm Error	0.0000000000	F-Norm Error	0.0000000000
Total Runtime	0.1551	Total Runtime	1.4126
P = 1, N = 4000		**P = 1, N = 8000**	
Rank	Time	Rank	Time
0	15.5824	0	122.8605
F-Norm Error	0.0000000000	F-Norm Error	0.0000000000
Total Runtime	15.5824	Total Runtime	122.8605

P = 2, N = 1000		**P = 2, N = 2000**	
Rank	Time	Rank	Time
0	0.1103	0	0.8258
1	0.1103	1	0.8258
F-Norm Error	0.0000000000	F-Norm Error	0.0000000000
Total Runtime	0.1103	Total Runtime	0.8258
P = 2, N = 4000		**P = 2, N = 8000**	
Rank	Time	Rank	Time
0	10.1511	0	87.7051
1	10.1511	1	87.7051
F-Norm Error	0.0000000000	F-Norm Error	0.0000000000
Total Runtime	10.1511	Total Runtime	87.7051

P = 4, N = 1000		**P = 4, N = 2000**	
Rank	Time	Rank	Time
0	0.0757	0	0.5600
1	0.0624	1	0.4557
2	0.0746	2	0.5571
3	0.0757	3	0.5600
F-Norm Error	0.0000000000	F-Norm Error	0.0000000000
Total Runtime	0.0757	Total Runtime	0.5600
P = 4, N = 4000		**P = 4, N = 8000**	
Rank	Time	Rank	Time
0	6.3028	0	61.8996
1	5.2114	1	50.1221
2	6.2909	2	61.8480
3	6.3028	3	61.8996
F-Norm Error	0.0000000000	F-Norm Error	0.0000000000
Total Runtime	6.3028	Total Runtime	61.8996

P = 8, N = 1000		**P = 8, N = 2000**	
Rank	Time	Rank	Time
0	0.0338	0	0.2251
1	0.0301	1	0.1968
2	0.0306	2	0.1985
3	0.0309	3	0.2040
4	0.0323	4	0.2133
5	0.0327	5	0.2191
6	0.0332	6	0.2236
7	0.0338	7	0.2251
F-Norm Error	0.0000000000	F-Norm Error	0.0000000001
Total Runtime	0.0338	Total Runtime	0.2251
P = 8, N = 4000		**P = 8, N = 8000**	
Rank	Time	Rank	Time
0	2.0520	0	30.8198
1	1.7786	1	26.2124
2	1.7850	2	26.2380
3	1.8268	3	27.0624
4	1.9435	4	30.3688
5	2.0010	5	30.3963
6	2.0457	6	30.7952
7	2.0520	7	30.8198
F-Norm Error	0.0000000003	F-Norm Error	0.0000000014
Total Runtime	2.0520	Total Runtime	30.8198

3. Summary

	N = 1000	**N = 2000**	**N = 4000**	**N = 8000**
P = 1	0.1551	1.4126	15.5824	122.8605
P = 2	0.1103	0.8258	10.1511	87.7051
P = 4	0.0757	0.5600	6.3028	61.8996
P = 8	0.0338	0.2251	2.0520	30.8198

4. Analyze

For P = 1, we see the performance is very close to the serial program. It is slightly worse than serial program, which might be some overhead of MPI initialization operation, etc.

For the scalability, as P increasing the scalability will get closer to linear, such for N = 8000, P = 8 cost just half of runtime of P = 4. However, the scalability of initial stage (P is small), the scalability is not quite close to linear, but still it can achieve reasonable speedup.

As for the load balance, it's not that bad, but we do can see node with larger rank generally took more time to finish, since the triangle matrix have more elements with larger row index. Thus, high rank node will have to do much more calculation than those low rank nodes. And such runtime gap between different node increases as N increasing.

Task 2B: Blocking MPI Parallel Program (Part B)

1. Command

> *sbatch /scripts/task2b-p4-n2.sh && sbatch /scripts/task2b-p4-n4.sh && sbatch /scripts/task2b-p8-n2.sh && sbatch /scripts/task2b-p8-n4.sh*

2. Detailed Results

P = 4, Node = 1, N = 8000			
Rank	Comp Time	Comm Time	Total
0	11.9916	47.5601	59.5518
1	29.8898	18.7373	48.6271
2	41.3327	18.1727	59.5053
3	47.4922	12.0596	59.5518
F-Norm Error	0.0000000000		
Total Runtime	59.5518		

P = 4, Node = 2, N = 8000			
Rank	Comp Time	Comm Time	Total
0	8.4763	50.6959	59.1722
1	25.9976	22.1357	48.1332
2	37.1569	21.9627	59.1196
3	46.2399	12.9464	59.1862
F-Norm Error	0.0000000000		
Total Runtime	59.1722		

P = 4, Node = 4, N = 8000			
Rank	Comp Time	Comm Time	Total
0	8.0687	47.0753	55.1440
1	25.8368	18.1827	44.0196
2	37.1669	17.9257	55.0926
3	42.3209	12.8349	55.1558
F-Norm Error	0.0000000000		
Total Runtime	55.1440		

P = 8, Node = 1, N = 8000			
Rank	Comp Time	Comm Time	Total
0	1.2541	29.5657	30.8198
1	3.4103	22.8021	26.2124
2	10.8652	15.3727	26.2380
3	11.5449	15.5175	27.0624
4	18.8919	11.4769	30.3688
5	16.2301	14.1663	30.3963

6	22.3010	8.4942	30.7952
7	22.8070	8.0128	30.8198
F-Norm Error	0.0000000014		
Total Runtime	30.8198		

P = 8, Node = 2, N = 8000			
Rank	Comp Time	Comm Time	Total
0	1.7755	30.3798	32.1553
1	6.3072	21.0586	27.3658
2	9.5915	17.7958	27.3873
3	15.2136	13.7894	29.0030
4	19.0850	11.5763	30.6613
5	21.7554	9.3781	31.1335
6	22.4526	9.6888	32.1414
7	23.3918	8.7774	32.1692
F-Norm Error	0.0000000000		
Total Runtime	32.1553		

P = 8, Node = 4, N = 8000			
Rank	Comp Time	Comm Time	Total
0	1.2585	29.3916	30.6501
1	3.2300	22.4993	25.7293
2	10.4446	15.3055	25.7501
3	14.5771	12.2376	26.8147
4	20.9660	8.4720	29.4380
5	25.1059	5.5226	30.6285
6	20.2531	10.3831	30.6361
7	18.2438	12.4363	30.6801
F-Norm Error	0.0000000000		
Total Runtime	30.6501		

3. Summary

	P = 4	**P = 8**
Node = 1	59.5518	30.8198
Node = 2	59.1722	32.1553
Node = 4	55.1440	30.6501

4. Analyze

From the computation time and communication time listed above, we can see that, node with higher rank generally have longer computation time. This is because of the imbalance split of matrix A. In this part, we split A evenly by row number, but since A is a lower triangular matrix, thus number of

elements grows as row number increases. Higher rank nodes were assigned with higher row number part of matrix, thus cost more computation time.

As for communication time, master node (rank = 0) takes most communication time since it is responsible for assign row and collect final answer from workers.

Generally, as the number of node increasing, we will get better performance. But the difference is quite small. The reason behind such improvement is probably for Part A, communication is handled by only one node, which means that single daemon need to handle all those processes. While for part B, we have more node to handle those processes. From the memory aspect, we also can reach same conclusion that more nodes can leads to less memory required on each node. However, we do see some exception, my guess is that there is still some messaging cost between different node. So, it is possible that the performance actually going down as we have more nodes.

A few suggestions to improve the program will be: (i) using non-blocking operations, so that nodes can do the calculation while sending and receiving column data. (ii) Instead of split A based on row number, we can split it based on element number. That means more rows for lower rank nodes, since those rows generally holds less elements. By doing this, we should be able to reduce the gap of computation time among different nodes.

Task 3A: Non-Blocking MPI Parallel Program (Part A)

1. Command

> *sbatch /scripts/task3a-1.sh && sbatch /scripts/task3a-2.sh && sbatch /scripts/task3a-4.sh && sbatch /scripts/task3a-8.sh*

2. Detailed Results

P = 1, N = 1000		**P = 1, N = 2000**	
Rank	Time	Rank	Time
0	0.1537	0	1.2289
F-Norm Error	0.0000000000	F-Norm Error	0.0000000000
Total Runtime	0.1537	Total Runtime	1.2289
P = 1, N = 4000		**P = 1, N = 8000**	
Rank	Time	Rank	Time
0	14.7848	0	120.1992
F-Norm Error	0.0000000000	F-Norm Error	0.0000000000
Total Runtime	14.7848	Total Runtime	120.1992

P = 2, N = 1000		**P = 2, N = 2000**	
Rank	Time	Rank	Time
0	0.1102	0	0.8276
1	0.1102	1	0.8276
F-Norm Error	0.0000000000	F-Norm Error	0.0000000000
Total Runtime	0.1102	Total Runtime	0.8276
P = 2, N = 4000		**P = 2, N = 8000**	
Rank	Time	Rank	Time

0	10.2163	0	85.3569
1	10.2163	1	85.3569
F-Norm Error	0.0000000000	F-Norm Error	0.0000000000
Total Runtime	10.2163	Total Runtime	85.3569

P = 4, N = 1000		**P = 4, N = 2000**	
Rank	Time	Rank	Time
0	0.0721	0	0.4683
1	0.0556	1	0.3324
2	0.0721	2	0.4683
3	0.0583	3	0.4417
F-Norm Error	0.0000000000	F-Norm Error	0.0000000000
Total Runtime	0.0721	Total Runtime	0.4683
P = 4, N = 4000		**P = 4, N = 8000**	
Rank	Time	Rank	Time
0	6.4082	0	51.3995
1	5.0815	1	43.0789
2	6.4083	2	51.3995
3	5.1839	3	47.7932
F-Norm Error	0.0000000000	F-Norm Error	0.0000000000
Total Runtime	6.4082	Total Runtime	51.3995

P = 8, N = 1000		**P = 8, N = 2000**	
Rank	Time	Rank	Time
0	0.0236	0	0.1864
1	0.0197	1	0.1452
2	0.0198	2	0.1468
3	0.0205	3	0.1485
4	0.0217	4	0.1634
5	0.0235	5	0.1864
6	0.0236	6	0.1819
7	0.0215	7	0.1538
F-Norm Error	0.0000000000	F-Norm Error	0.0000000001
Total Runtime	0.0236	Total Runtime	0.1864
P = 8, N = 4000		**P = 8, N = 8000**	
Rank	Time	Rank	Time
0	1.8613	0	28.2292
1	1.4568	1	22.3856
2	1.4628	2	22.4127
3	1.4699	3	22.4414
4	1.6438	4	25.7801
5	1.8613	5	28.2291

6	1.7993	6	27.5423
7	1.5270	7	22.9500
F-Norm Error	0.0000000003	F-Norm Error	0.0000000014
Total Runtime	1.8613	Total Runtime	28.2292

3. Summary

	N = 1000	**N = 2000**	**N = 4000**	**N = 8000**
P = 1	0.1537	1.2289	14.7848	120.1992
P = 2	0.1102	0.8276	10.2163	85.3569
P = 4	0.0721	0.4683	6.4082	51.3995
P = 8	0.0236	0.1864	1.8613	28.2292

4. Analyze

For this part, we achieve non-blocking MPI program by sending and receiving column data while doing calculation. From the summary table above, we can see that the program achieves significant improvement for large N and P, more than 10%.
The conclusion of the scalability of Task2 still holds true. We can even observe some over linear speed up when N and P is large (6.4082s to 1.8613s for N = 4000). And the load balance is still not good, since changing non-blocking won't help that unfair split of A.
In conclusion, using non-blocking MPI operations does help a lot to the overall performance.

Task 3B: Non-Blocking MPI Parallel Program (Part B)

1. Command

> *sbatch /scripts/task3b-p4-n2.sh && sbatch /scripts/task3b-p4-n4.sh && sbatch /scripts/task3b-p8-n2.sh && sbatch /scripts/task3b-p8-n4.sh*

2. Detailed Results

P = 4, Node = 1, N = 8000			
Rank	Comp Time	Comm Time	Total
0	11.6594	39.7401	51.3995
1	30.6377	12.4412	43.0789
2	42.1020	9.2975	51.3995
3	47.6064	0.1868	47.7932
F-Norm Error	0.0000000000		
Total Runtime	51.3995		

P = 4, Node = 2, N = 8000			
Rank	Comp Time	Comm Time	Total

0	9.2218	41.4786	50.7004
1	27.5003	14.5536	42.0538
2	39.2583	11.4421	50.7004
3	46.2836	0.2629	46.5465
F-Norm Error	0.0000000000		
Total Runtime	50.7004		

P = 4, Node = 4, N = 8000			
Rank	Comp Time	Comm Time	Total
0	8.7496	47.1947	55.9443
1	25.7966	18.5902	44.3869
2	37.5733	18.3709	55.9442
3	43.7433	0.2337	43.9770
F-Norm Error	0.0000000000		
Total Runtime	55.9443		

P = 8, Node = 1, N = 8000			
Rank	Comp Time	Comm Time	Total
0	1.2746	26.9546	28.2292
1	3.4198	18.9657	22.3856
2	12.7303	9.6824	22.4127
3	10.2587	12.1827	22.4414
4	19.6628	6.1173	25.7801
5	16.2091	12.0200	28.2291
6	23.0689	4.4735	27.5423
7	22.8032	0.1469	22.9500
F-Norm Error	0.0000000014		
Total Runtime	28.2292		

P = 8, Node = 2, N = 8000			
Rank	Comp Time	Comm Time	Total
0	1.8825	27.3350	29.2175
1	5.6114	18.4899	24.1014
2	11.6611	12.4622	24.1233
3	14.1381	10.8823	25.0204
4	21.3511	4.9576	26.3087
5	22.9532	5.6015	28.5547
6	26.0164	3.2129	29.2293
7	24.8116	0.2349	25.0465
F-Norm Error	0.0000000000		
Total Runtime	29.2175		

P = 8, Node = 4, N = 8000			
Rank	Comp Time	Comm Time	Total
0	1.3103	25.9016	27.2120
1	3.2310	16.5520	19.7830
2	12.8985	8.6203	21.5188
3	15.5608	9.7486	25.3094
4	14.2588	11.4483	25.7071
5	17.3967	9.7923	27.1890
6	24.9794	2.2444	27.2238
7	20.0448	0.3049	20.3497
F-Norm Error	0.0000000000		
Total Runtime	27.2120		

3. Summary

	P = 4	**P = 8**
Node = 1	51.3995	28.2292
Node = 2	50.7004	29.2175
Node = 4	55.9443	27.2120

4. Analyze

From the communication and computation time listed above, we can clearly see how non-blocking improve the performance. For those high rank node, we can observe that the communication time is almost close to 0. That's because those nodes are assigned with larger rows, thus they require too much time to do the calculation, so that all those non-blocking sending and receiving operations are actually done during they doing the calculation. That why they seem almost have no communication time. The higher rank one node is, more computation time is required and less communication time it cost (actually more part of communication time will hide under its computation time). This also clearly demonstrate the poor load balance of our current program.

Task 4: Load Balance

1. Command

> sbatch /scripts/task4b-p4-n1.sh && sbatch /scripts/task4b-p4-n2.sh && sbatch /scripts/task4b-p4-n4.sh && sbatch /scripts/task4b-p8-n1.sh && sbatch /scripts/task4b-p8-n2.sh && sbatch /scripts/task4b-p8-n4.sh

2. Detailed Results

P = 4, Node = 1, N = 8000			
Rank	Comp Time	Comm Time	Total
0	41.7374	2.9908	44.7282
1	36.2997	8.4284	44.7282
2	31.7077	3.1547	34.8624
3	27.9382	16.7521	44.6903
F-Norm Error	0.0000000000		
Total Runtime	44.7282		

P = 4, Node = 2, N = 8000			
Rank	Comp Time	Comm Time	Total
0	37.4317	4.2038	41.6355
1	30.7952	10.8259	41.6211
2	28.0463	13.5783	41.6246
3	25.2460	16.4037	41.6497
F-Norm Error	0.0000000000		
Total Runtime	41.6355		

P = 4, Node = 4, N = 8000			
Rank	Comp Time	Comm Time	Total
0	38.6608	4.1558	42.8166
1	30.2379	12.5905	42.8284
2	26.1968	16.6029	42.7997
3	23.0809	19.7306	42.8115
F-Norm Error	0.0000000000		
Total Runtime	42.8166		

P = 8, Node = 1, N = 8000			
Rank	Comp Time	Comm Time	Total
0	22.7343	1.3693	24.1036
1	23.0700	1.0335	24.1035
2	21.5686	1.5351	23.1037
3	20.1986	2.9246	23.1232
4	19.0373	4.1035	23.1408
5	17.7763	5.3798	23.1562
6	16.4195	7.6552	24.0747
7	15.8706	8.1904	24.0610
F-Norm Error	0.0000000014		
Total Runtime	24.1036		

P = 8, Node = 2, N = 8000			
Rank	Comp Time	Comm Time	Total
0	23.0210	0.4731	23.4940
1	18.4578	3.5405	21.9982
2	20.3005	3.1917	23.4922
3	16.1925	5.8258	22.0183
4	16.3997	7.0890	23.4886
5	14.3814	9.1213	23.5027
6	14.8266	8.6738	23.5005
7	12.7674	10.7407	23.5081
F-Norm Error	0.0000000000		
Total Runtime	23.4940		

P = 8, Node = 4, N = 8000			
Rank	Comp Time	Comm Time	Total
0	10.9195	10.2747	21.1941
1	11.5961	8.0242	19.6203
2	18.8660	2.3640	21.2301
3	14.8797	3.4026	18.2823
4	15.1271	3.9196	19.0466
5	12.7071	3.4014	16.1085
6	12.4450	3.6148	16.0598
7	10.5215	5.5566	16.0781
F-Norm Error	0.0000000014		
Total Runtime	21.1941		

3. Summary

	P = 4	**P = 8**
Node = 1	44.7282	24.1036
Node = 2	41.6355	23.4940
Node = 4	42.8166	21.1941

4. Analyze

In this task, we split A based on the number of elements. We will try to split the original A so that each node will receive generally same number of elements.

From the computation and communication time listed above, we can observe that the load balance of our program has been improved a lot. The gap between different nodes is much less than the previous two version. By doing this, we can see the performance is improved more than 20%.

We also implement generalization in this part. For the row splitting part, due to the split-by-element implementation, we can automatically support any number of rows. As for the column splitting, still

split it evenly by column number. Thus, we have to add additional checking to see if it reaches the last part of B, if so the last part will take all the rest of B. By doing this, out program can support different size of input.

Task 5: Generalization

1. Command

> *sbatch /scripts/task5.sh*

2. Detailed Results

P = 7, Node = 4, N = 7633			
Rank	Comp Time	Comm Time	Total
0	10.8057	8.6646	19.4703
1	11.7493	5.8966	17.6459
2	17.145	2.3108	19.4559
3	15.761	2.5203	18.2813
4	15.3998	2.4251	17.8249
5	13.9899	0.6045	14.5944
6	10.563	3.1948	13.7577
F-Norm Error	0.0000000014		
Total Runtime	19.4703		

From the table above, we can see that our program can correctly calculating N = 7633 matrix correctly using 7 processes on 4 nodes. The implementation method is described above.