Homework #3

(Due: 7 May)

Group Members		
Name	SBU ID	% Contribution
Varun	111491232	Equal
Aditya	111491409	Equal

1 Task 1. [120 Points] Distributed-Memory Matrix Multiplication.

Question 1a. (a) [50 Points] Implement the three distributed-memory algorithms for multiplying two square matrices shown in Figures 13. Assume the initial distribution of input matrices and the final distribution of the output matrix from lecture 13.
Ans.
All three algorithms developed, code attached. Location of code at stampede2 machine at location $/work/05567/varun31/stampede2/Group19\_VarunAgarwal\_111491232\#\_AdityaTomer\_111491409\#/code/distributed\_matrix\_mul\_mpi$

WE have implemented the code using MPI_SEND and MPI_RECV for initial distribution and final collection of matrices.

Question 1b. (b) Use your implementations from part 1(a) to multiply two 2^k x 2^k matrices (initialized with random integers in [100, 100]) on 2^l x 2^l compute nodes with 1 process/node for $10 \leq$ k ≤ 14 and $0 \leq$ l ≤ 2. Report the running times and explain your findings..
Ans.

MM - rotate A - rotate B —— processor per node =1		
Array Size (N x N)	**Compute Nodes**	**Running Time (ms)**
1024	1	3260
1024	4	850
1024	16	289
2048	1	23269
2048	4	7051
2048	16	1895
4096	1	176891
4096	4	50713
4096	16	14743
8192	1	>400000
8192	4	368257
8192	16	105481
16384	1	>400000
16384	4	>400000
16384	16	>400000

MM - rotate A - broadcast B—— processor per node =1		
Array Size (N x N)	**Compute Nodes**	**Running Time (ms)**
1024	1	1634
1024	4	507
1024	16	209
2048	1	12107
2048	4	3582
2048	16	1208
4096	1	94855
4096	4	25736
4096	16	7878
8192	1	>400000
8192	4	197760
8192	16	54646
16384	1	>400000
16384	4	>400000
16384	16	>400000

MM - broadcast A - broadcast B—— processor per node =1		
Array Size (N x N)	**Compute Nodes**	**Running Time (ms)**
1024	1	1651
1024	4	512
1024	16	208
2048	1	12164
2048	4	3585
2048	16	1216
4096	1	94503
4096	4	25713
4096	16	7836
8192	1	>400000
8192	4	257917
8192	16	54399
16384	1	>400000
16384	4	>400000
16384	16	>400000

It is pretty evident from above 3 tables that $broadcast\_A\_broadcast_B$ **algorithm is significantly faster over** $rotate\_A\_rotate\_B$ **but is slightly faster than** $rotate\_A\_broadcast\_B$ **for higher values of matrix size. For smaller values the difference is not that evident**

Question 1c. (c) [10 Points] Repeat part 1(b) with t processes/node, where t is the num-

ber of cores available on a compute node. Report the running times. Compare with part 1(b) and explain.
Ans.

MM - rotate A - rotate B—— processor per node =68		
Array Size (N x N)	**Compute Nodes**	**Running Time (ms)**
1024	1	3286
1024	4	843
1024	16	290
2048	1	23844
2048	4	7070
2048	16	1914
4096	1	176060
4096	4	48561
4096	16	14736
8192	1	>400000
8192	4	367739
8192	16	103371
16384	1	>400000
16384	4	>400000
16384	16	>400000

MM - rotate A - broadcast B—— processor per node =68		
Array Size (N x N)	**Compute Nodes**	**Running Time (ms)**
1024	1	1661
1024	4	505
1024	16	208
2048	1	12477
2048	4	3601
2048	16	1206
4096	1	95263
4096	4	25737
4096	16	7871
8192	1	>400000
8192	4	196871
8192	16	54922
16384	1	>400000
16384	4	>400000
16384	16	>400000

MM - broadcast A - broadcast B—— processor per node =68		
Array Size (N x N)	**Compute Nodes**	**Running Time (ms)**
1024	1	1672
1024	4	511
1024	16	208
2048	1	12304
2048	4	3579
2048	16	1203
4096	1	94882
4096	4	25716
4096	16	7876
8192	1	>400000
8192	4	196245
8192	16	54105
16384	1	>400000
16384	4	>400000
16384	16	>400000

Comparing the values from from above 3 tables conclusion is that there is a slight difference when the processes per node is set as 68. Timing is not significantly lower but slightly lower.Distribution of processes to cores and processors is handled by the operating system and the MPI implementation. Running on a desktop, the operating system will generally put each process on a different core, potentially redistributing processes during run-time. In larger systems such a s a supercomputer or a cluster, the distribution is handled by resource managers such as SLURM. However this happens, one or multiple processes will be assigned to each core. Regarding hardware, a core can run only a single process at a time. Technologies such as hyper-threading allows multiple processes to share the resources of a single core. There are cases where two or more processes per core is optimal. For instance, if a processes is doing a large amount of file I/O another may take its place and do computation while the first is hung on a read or write.

Question 1d. [20 Points] Suppose a master node initially holds the input matrices and will hold the final output matrix. Augment your fastest implementation from part 1(a) with efficient routines for initial distribution and final collection of matrices. When you measure the running time of this algorithm please include the time needed for these additional distribution/collection steps

Ans. Fastest implementation among the 3 given algorithms , broadcast-A-broadcast-B was the fastest.

Question 1e. Repeat part 1(b) with the algorithm from part 1(d).

Ans.

MM - broadcast A - broadcast B —— processor per node = 1		
Array Size (N x N)	**Nodes**	**Running Time (ms)**
1024	1	1681
1024	4	478
1024	16	181
2048	1	12228
2048	4	3516
2048	16	1101
4096	1	94511
4096	4	25659
4096	16	7642
8192	1	>400000
8192	4	195444
8192	16	54423
16384	1	>400000
16384	4	>400000
16384	16	>400000

Using Scatter and Gather techniques for initial distribution and final gathering, scatter technique comes out to be slightly faster, because of parallel implementation of distribution and gathering of matrices internally by MPI library api's which are optimised.

Question 1f. Repeat part 1(c) with the algorithm from part 1(d).
Ans.

MM - broadcast A - broadcast B —— processor per node = 1		
Array Size (N x N)	**Nodes**	**Running Time (ms)**
1024	1	1642
1024	4	486
1024	16	179
2048	1	12184
2048	4	3520
2048	16	1097
4096	1	94389
4096	4	25547
4096	16	7660
8192	1	>400000
8192	4	196871
8192	16	54727
16384	1	>400000
16384	4	>400000
16384	16	>400000

Comparing the values from from table conclusion is that there is a slight difference when the processes per node is set as 68. Timing is not significantly lower but slightly lower.

2 Task 2. [80 Points] Distributed-Shared-Memory Matrix Multiplication

Question 2a Modify your fastest implementation from part 1(a) and and its modified version from part 1(d) to use a shared-memory parallel matrix multiplication algorithm inside each process. Use your fastest shared-memory parallel matrix multiplication routine from HW1.

Ans 2a.

recursive matrix multiplication is implemented for shared-memory parallel matrix multiplication. Implementing recursive multiplication has a significant speed up over other algorithms

Question 2b [25 Points] Repeat part 1(b) with the two implementations from part 2(a). Use 1 process/node, but inside each process use all cores available on that node.

Ans.

MM - broadcast A - broadcast B —— processor per node = 1—— Scatter/Gather		
Array Size (N x N)	**Nodes**	**Running Time (ms)**
1024	1	196
1024	4	198
1024	16	346
2048	1	1009
2048	4	618
2048	16	844
4096	1	5669
4096	4	3610
4096	16	3728
8192	1	47525
8192	4	28577
8192	16	26951
16384	1	>400000
16384	4	264325
16384	16	231793

MM - broadcast A - broadcast B —— processor per node = 1—— Scatter/Gather		
Array Size (N x N)	**Nodes**	**Running Time (ms)**
1024	1	196
1024	4	340
1024	16	598
2048	1	1009
2048	4	942
2048	16	1186
4096	1	5669
4096	4	4667
4096	16	4997
8192	1	47525
8192	4	33247
8192	16	31808
16384	1	>400000
16384	4	264325
16384	16	251681

Question 2c [25 Points] Compare your results from parts 1(b, c, e, f) with those from part

2(b). Explain your findings.

Comparing the results from b,c,e,f form 1 question it is pretty evident that introducing cilk parallel multiplication has drastically reduced the running time of algorithms for all compute nodes. cilk reduces the time significantly due to parallel processing as all cores are used for running the algorithms. Also in recursive multiplication grain size of 32 is used as breakpoint of recursion from parallel to sequential.