Parallel Programming HW3
Prateek Roy(111481907), Akshay Satam(111481679)

****

README:

Compile : Put the Makefile attached in the folder to any folder with the source file you want to test and run make.

To run any part of the program:
mpirun -n <no of processors> -ppn <no of process per node> ./run L K
Where L and K are variables declared in Task (1b).

Note: All readings are taken on Comet and not Stampede as stampede was out of funds.
****

Task 1:

1a) The code for the 3 algorithms are in
Figure 1 - MM.cpp
Figure 2 - MM_BCast.cpp
Figure 3 - MM_AllBCast.cpp

(1b)

Figure 1 algorithm:Distributed matrix multiplications using block rotations for both input matrices.

	L = 0	L = 1	L = 2
K = 10	1.399214	0.373948	0.092220
K = 11	11.072402	2.781027	0.760019
K = 12	89.967975	22.602617	5.959565
K = 13	661.32102	180.635603	48.334932
K = 14	NA	1441.93573	388.430059

Figure 2 algorithm:

	L = 0	L = 1	L = 2

K = 10	1.278009	0.349698	0.088692
K = 11	10.117634	2.774956	0.732012
K = 12	82.387464	22.448649	5.278924
K = 13	653.396235	180.41042	48.256339
K = 14	NA	1438.6572	386.948123

Figure 3 algorithm:

	L = 0	L = 1	L = 2
K = 10	1.386733	0.350731	0.089953
K = 11	11.056066	2.778291	0.701962
K = 12	89.872077	22.688645	5.900430
K = 13	656.35121	180.955361	48.24182
K = 14	NA	1439.3452	380.739205

1c) Per processor there are 24 cores. So we run the algorithms for 24 processes per core.
Run command : mpirun -n 16 -ppn 16 2 <n>

Figure 1 algorithm: Distributed matrix multiplications using block rotations for both input matrices.

	L = 0	L = 1	L = 2
K = 10	0.092181	0.098123	0.092220
K = 11	0.759901	0.762130	0.760019
K = 12	5.901921	5.87120	5.959565
K = 13	48.30019	48.31304	48.334932
K = 14	388.10231	388.13091	388.430059

Figure 2 algorithm:

	L = 0	L = 1	L = 2

K = 10	0.088812	0.087021	0.088692
K = 11	0.740123	0.738901	0.732012
K = 12	5.281231	5.133412	5.278924
K = 13	48.28892	48.31029	48.256339
K = 14	386.91002	387.00912	386.948123

Figure 3 algorithm:

	L = 0	L = 1	L = 2
K = 10	0.091204	0.09102	0.090083
K = 11	0.700012	0.71020	0.701883
K = 12	5.97101	5.95182	5.961048
K = 13	47.91022	47.80192	47.913166
K = 14	383.18912	381.01928	381.04681

1d)
The faster algorithm from Part 1a for me is the algorithm given in Figure 3 : Using block broadcast for both input matrices. I added the routines for all the processors sending their segment of the matrix to the master node at the end and the master collaborating all the segments and forming the final C matrix.
The code for above in in 1d folder.
The routines to look out for is void Collaborate(int** C, int n, int processors).

1e) After Distribution/Collaboration from master node:
Repeating Part (1b) for each l and k value for Figure 3 algo(fastest)

Figure 3:

	L = 0	L = 1	L = 2
K = 10	1.3312	0.34512	0.093674
K = 11	11.4213	2.80219	0.716491

K = 12	89.90012	22.10280	6.003706
K = 13	649.41245	182.42122	48.130185
K = 14	NA	1451.1211	381.697091

1f) Running 24 processes per node with Distribution/Collaboration from master node in the (Both block broadcasts for both input matrices algorithm)

	L = 0	L = 1	L = 2
K = 10	0.090971	0.091112	0.091918
K = 11	0.717739	0.713151	0.719123
K = 12	6.038549	6.00912	6.048218
K = 13	48.171754	48.19525	48.081622
K = 14	381.31412	381.12451	381.80941

1g) Time taken in Table1(e) are slightly more than Table1(b) because of

a) The extra time taken by master node to distribute each segment of the matrix to appropriate processor so that the processors can compute their result in parallel.
b) The extra time taken by each processor to send their final segment of the matrix to the master node at the end, so that the master node can accumulate all segments and calculate the whole C matrix.

The total time taken by the above two steps is the reason for increase in time. However this time difference is very small which means the time taken in sending the final matrix by 1, 4, 16 number of processors to master is very small and vice versa.

The difference in time between Table1(c) and Table1(f) is slightly less as compared to the above difference between because of Table1(e) -Table1(b) because here the processes are running on different cores on a single processor as compared to Table1(e) -Table1(b) where the processes were running on different processors.

Task2:

2a) The code is attached as inside 2a folder.

2b) My Fastest from 1a is Both block broadcasts for both input matrices algorithm. The fastest Par Matrix Multiplication from HW1 is the iterative Matrix Multiplication with ikj configuration with parallel i loop. Adding the same Par Matrix Multiplication from HW1 in place of MatMul to it, the results are:

	L = 0	L = 1	L = 2
K = 10	0.42	0.18	0.97
K = 11	3.51	1.4	1.02
K = 12	27.01	11.06	5.03
K = 13	195	92.43	41.023
K = 14	NA	725.81	327.092

2c) As it is evident that part 2b - One with parallel matrix multiplication is taking much less time than the serial matrix multiplication part in 1 (b, c, e, f) . This is because the serial matrix multiplication was replaced with parallel matrix multiplication in the code to utilize all cores available on that processor.
The code is attached in MM_2a.cpp where you can see that multiplyMatrixPar(IC, sA, sB, seg_size, seg_size); is called instead of multiplyMatrix(IC, sA, sB, seg_size, seg_size);
I even tried with #pragma cilk grainsize = 1 before the cilk_for loop.