ASSIGNMENT - 3(Group Num : 12)

(Hiren Mayani, Krishna Sharma)

All these programs are computed on Stampede compute nodes.
Instruction to compile and run:
We provided —ntasks-per-node and —nodes while submitting the batch.

Compilation: mpiicc --std=c++11 DM_MatMul.cpp -o task1
Run: mpirun task1 $algo $k $isPar “out.csv”

Here, $algo can be 1,2,3,4

1	: Rotate-A-Rotate-B
2	: Bcast-A-Bcast-B
3	: Bcast-A-Bcast-B
4	: Scatter-gather

$k - size of matrix : 10, 11, 12, 13, 14
$isPar = 0, if sequential matrix multiplication, 1 otherwise

Task - 1

1(a)

We implemented all three versions of the Matrix multiplication algorithm. You can check it in DM_MatMul.cpp file. There are separate methods for each of these algorithms. We tested results of all these implementations with different data and It is correct.

1(b)

We used all these three implementations to multiply two matrix of size 2^K X 2^K 2 .

Size of Matrix = 2^k * 2k
Computer Nodes = 2^l * 2^l
Number of processes per node = 1

Here’s a table for all running times for different combinations of above variables.

Number of processes/Node = 1

algo	k	l(log2(total nodes))	Number of process/node	Running time(ms)
1	10	0	1	28.9891
1	11	0	1	227.691
1	12	0	1	1782.82
1	10	1	1	5.1727
1	11	1	1	58.5909
1	12	1	1	453.464
1	13	1	1	9113.466
1	10	2	1	0.448
1	11	2	1	11.9777
1	12	2	1	117.3550
1	13	2	1	913.466
2	10	0	1	22.3196
2	11	0	1	176.566
2	12	0	1	1376.7
2	10	1	1	4.1946
2	11	1	1	46.2450
2	12	1	1	350.311
2	13	1	1	9477.591
2	10	2	1	0.3055
2	11	2	1	9.5311
2	12	2	1	92.233
2	13	2	1	707.591
3	10	0	1	22.713
3	11	0	1	177.937
3	12	0	1	1407.66
3	10	1	1	4.5399
3	11	1	1	45.7719
3	12	1	1	358.392
3	13	1	1	9810.264
3	10	2	1	0.305
3	11	2	1	10.4115
3	12	2	1	94.028
3	13	2	1	710.264

Here are some of the observations we made from this data.

- Running time exponentially increase with respect to k (size of matrix)
- Running time exponentially decrease with respect to l (numbers of nodes)
- Running time for all algorithms are relatively similar. This may because array multiplication cost is quite high and It dominates other costs. However we can see 3rd algorithms is slightly faster that other two.

1(C)

Now we will repeat same part with total number of processes per node increased to 64. Because, Stampede2 has 64 cores available per node.

Notation:
Running Algorithms :
1 : Rotate-A-Rotate-B
2 : Bcast-A-Bcast-B
3 : Bcast-A-Bcast-B

Size of Matrix = 2^k * 2k
Computer Nodes = 2^l * 2^l
Number of processes per node = 64

Here's a table for all running times for different combinations of above variables.

Number of process/node = 64

algo	k	l(log2(total nodes))	Number of process/node	Running time(ms)
1	10	0	64	0.1077
1	11	0	64	0.992
1	12	0	64	27.27
1	13	0	64	269.694
1	14	0	64	2124.39
1	10	2	64	0.01
1	11	2	64	0.05
1	12	2	64	0.43
1	13	2	64	4.2945
1	14	2	64	109.631
1	10	1	64	0.02898
1	11	1	64	0.216388
1	12	1	64	2.06594
1	13	1	64	54.6383
1	14	1	64	550.896

2	10	0	64	0.0850
2	11	0	64	0.656
2	12	0	64	20.8
2	13	0	64	210.646
2	10	2	64	0.04
2	11	2	64	0.06
2	12	2	64	0.37
2	13	2	64	3.14988
2	14	2	64	96.436
2	10	1	64	0.0362449
2	11	1	64	0.173578
2	12	1	64	1.36792
2	13	1	64	41.9494
2	14	1	64	435.174
3	10	0	64	0.084
3	11	0	64	0.706
3	12	0	64	21.65
3	13	0	64	218.062
3	10	2	64	0.03
3	11	2	64	0.07
3	12	2	64	0.38
3	13	2	64	2.96555
3	14	2	64	89.0027
3	10	1	64	0.0332689
3	11	1	64	0.168483
3	12	1	64	1.36584
3	13	1	64	43.8219
3	14	1	64	435.611

Here are some of the observations from this data.
- Running time decreased **substantially** compared to 1 process/node.
- Running time exponentially increases with respect to k (size of matrix)
- Running time exponentially decreases with respect to l (numbers of nodes)

1(d)

Implementation of all three were taking quite close time. However broadcastAbroadcastB was taking relatively less time. This is because of it's efficient implementation in MPI library.

We used scatterv and gatherv to share data of A and B and gather data of C.

1(e)

We used fastest implementation(BcastA-BcastB) to multiply two matrix of size 2^K X 2^K 2 .

Size of Matrix = 2^k * 2k
Computer Nodes = 2^l * 2^l
Number of processes per node = 1

Table 1

k	l(log2(total nodes))	Number of process/node	Running time(ms)
10	0	1	22.4905
11	0	1	178.893
12	0	1	1393.11
10	2	1	0.4269
11	2	1	9.5215
10	1	1	4.5702
11	1	1	44.641
12	2	1	92.087
12	1	1	352.563
13	2	1	712.435

Here are some of the observations from this data.

- These times are not much different than times in 1(b), because matrix multiplication cost dominates.
- Running time exponentially increases with respect to k (size of matrix)
- Running time exponentially decreases with respect to l (numbers of nodes)

1(f)

We used fastest implementation(BcastA-BcastB) to multiply two matrix of size 2^K X 2^K 2 .

Size of Matrix = 2^k * 2k
Computer Nodes = 2^l * 2^l
Number of processes per node = 64

Table 1-1

k	l(log2(total nodes))	Number of process/node	Running time(ms)
10	0	64	0.1481
11	0	64	0.873
12	0	64	22.39
13	0	64	222.279
10	2	64	0.50
11	2	64	0.49
12	2	64	1.81324
13	2	64	7.4602
10	1	64	0.112718
11	1	64	0.436841
12	1	64	2.38945
13	1	64	47.0437

Here are some of the observations from this data.

- This times are not much different than times in 1(c), because matrix multiplication dominates. However, it's slightly higher that running time from 1(e).
- Running time exponentially increase with respect to k (size of matrix)
- Running time exponentially decrease with respect to l (numbers of nodes)

1(g)
As we can see,
time in 1(d) are not much different than times in 1(b). This is because scatter and gather are not taking substantially high time compare to rest of the work(like matrix multiplication). So these timings are very close to timing without scatter and gather.
Matrix multination is sequential here, So, total time won't change by minor change in matrix distribution and collection.
For higher number of processes, Times remained almost unchanged.

Task - 2

2(a)

So far we were using distributed memory for matrix multiplication. However, It’s good idea to do matrix multiplication in parallel and shared memory. This can decrease running time substantially.

We implemented fastest version from 1(b) with parallel matrix multiplication algorithm. Although, we can run any algorithms from 1(a), 1(d) in shared memory by specifying option while running. This option is mentioned at the start of this document.

2(b)

Now, we are using Parallel algorithm to multiply matrix.

Notation:
Running Algorithms :
1 : without scatter and gather
4 : with scatter and gather

Size of Matrix = $2^k * 2k$
Computer Nodes = $2^l * 2^l$
Number of processes per node = 64,1

As we can note, these times are relatively less that times in 1(b). This is because cilk_for will multiply matrix quite faster than normal for loop. Running time decreased by more than half for several examples.

It’s also worth noting that algo with scatter and gather is taking quite more time than local matrices.

Table 1-2

algo	k	l	Number of process/node	time
1	13	0	64	35.5581

algo	k	l	Number of process/node	time
1	10	1	64	0.020
1	11	1	64	0.144
1	12	1	64	1.097
1	13	1	64	9.37751
1	14	1	64	74.2192
1	10	2	64	0.008
1	11	2	64	0.043
1	12	2	64	0.287
1	13	2	64	2.195
1	14	2	64	18.737
4	10	0	64	0.119586
4	11	0	64	0.54621
4	12	0	64	3.98936
4	13	0	64	29.144600
4	10	1	64	0.104593
4	11	1	64	0.403612
4	12	1	64	1.76096
4	13	1	64	10.1283
4	10	2	64	0.132
4	11	2	64	0.4658
4	12	2	64	1.57029
4	13	2	64	5.76264
1	10	0	1	28.9891
1	10	0	1	4.19078
1	11	0	1	33.3443
1	12	0	1	266.07
1	10	1	1	1.04253
1	11	1	1	8.40089
1	12	1	1	66.6425
1	10	2	1	0.254228

algo	k	l	Number of process/node	time
1	11	2	1	2.10312
1	12	2	1	17.0361
1	13	2	1	133.934
4	10	1	1	0.796731
4	11	1	1	5.970360
4	12	1	1	46.1837
4	10	2	1	0.27151
4	11	2	1	1.71182
4	12	2	1	12.4581
4	13	2	1	97.417700

2(C)

Here are some of the insights that we got from these running data.

- Running time decreases if we use shared memory algorithms to multiply matrix.
- For normal distributed memory algorithm, running time for Scatter gather algorithms were not very different.
- However, for shared memory distributed algorithm, total cost is not dominated by matrix multiplication cost. So time sent in scatter and gather will substantially affect total running time.
- Running time still increases with respect to k (size of matrix).
- Running time decreases with respect to l (numbers of nodes).