Spring 2018 CSE613 HW3

Abiyaz Chowdhury
StonyBrook ID: 111580554

June 21, 2018

Task 1: Distributed memory matrix multplication

B

(Note: All time taken is shown in seconds)
(RR: Rotate-rotate)
(RB: Rotate-broadcast)
(BB: Broadcast-broadcast)

Size	1N(RR)	1N(RB)	1N(BB)	4N(RR)	4N(RB)	4N(BB)	16N(RR)	16N(RB)	16N(BB)
1024	1.3874	1.40955	1.40867	0.384284	0.356074	0.384391	0.0964031	0.0910032	0.0966599
2048	11.05	11.1386	11.1415	2.82802	2.79729	2.82034	0.758112	0.706797	0.712641
4096	89.7809	90.1164	90.3651	22.3308	22.6732	22.3115	5.61082	5.9005	5.60557
8192	716.083	717.846	718.606	180.803	180.787	180.577	44.499	47.8353	44.4859
16384	N/A	N/A	N/A	N/A	N/A	N/A	360.71	380.38	360.605

C

(Note: All time taken is shown in seconds. 16 processors were used per node out of Comet's available 24 (my implementation only works when total processes across all nodes is a power of 4).
(RR: Rotate-rotate)
(RB: Rotate-broadcast)
(BB: Broadcast-broadcast)

Size	1N(RR)	1N(RB)	1N(BB)	4N(RR)	4N(RB)	4N(BB)	16N(RR)	16N(RB)	16N(BB)
1024	0.0939569	0.0910401	0.0914869	0.027194	0.025635	0.0270898	0.024323	0.0335021	0.068414
2048	0.750892	0.706511	0.706968	0.189452	0.183339	0.18651	0.0552521	0.0579751	0.05951
4096	6.00466	6.00613	5.9566	1.51167	1.43585	1.43955	0.388127	0.37929	0.39294
8192	50.0144	48.0448	48.2424	12.0959	12.1864	12.0638	3.11714	2.939	2.93591
16384	393.305	384.896	385.388	98.372	98.1421	96.9726	24.5318	24.5294	24.5052

Adding more cores per node speeds up the program by a factor roughly equal to the number of cores added (disregarding the overhead). This makes sense, since the work is being evenly parallized across all processes and the overhead is not too significant compared to the gains in performance for large workloads.

E

(Note: All time taken is shown in seconds. For this part, Rotate-rotate was used, and process 0 created a matrix which was then transmitted in blocks to the processors and then the final result was collected from each processor and combined into a full matrix.)

Size	1N(RR)	4N(RR)	16N(RR)
1024	1.43475	0.435187	0.151635
2048	11.2429	3.02088	0.961879
4096	90.5513	23.1213	6.39162
8192	719.735	183.896	47.6928
16384	N/A	N/A	373.583

F

(Note: All time taken is shown in seconds. 16 processors were used per node out of Comet's available 24 (my implementation only works when total processes across all nodes is a power of 4).

Size	1N(RR)	4N(RR)	16N(RR)
1024	0.126077	0.089628	0.320218
2048	0.869418	0.402798	0.305423
4096	6.52313	2.32942	1.24021
8192	50.7024	15.4679	6.31437
16384	395.774	111.063	37.0692

G

The time it takes is greater when a master node has to distribute blocks to each processor and then gather up the final result into a whole matrix. This is simply because the process of distributing and collecting back the result often takes some time.

Task 2: Distributed-shared-memory matrix multiplication

A

The nested-for-loop parallelization was used, where the outermost loop is over i and the innermost loop is over j. A parallel-for loop is applied on loop i. This modification is done to the rotate-rotate implementations in 1a and 1d. All 24 of Comet's cores are used in each process.

B

Size	1N(1a)	1N(1d)	4N(1a)	4N(1d)	16N(1a)	16N(1d)
1024	0.082207	0.139714	0.0839932	0.10528	0.0950401	0.147921
2048	0.533099	0.957865	0.231338	0.414408	0.0842721	0.265243
4096	3.59374	5.10574	1.49464	2.09078	0.427595	1.20862
8192	25.8794	29.8426	8.57292	10.194	2.67678	5.88358
16384	207.719	247.828	55.1993	71.798	16.4588	29.8926

C

We actually do expect to see better performance here than in task 1 because task 1 used only 16 cores in each node. Even if all 24 cores were used, I surmise this implementation would still perform better since shared-memory involves less overhead than distributed memory computations, although it may also depend on various other factors such as CPU affinities of the processors involved, and the system architecture.