FHPC @ units

Imola Fodor SM3500474

Assignment 1 , Deadline 30.12.2021

Contents

Section 1.

Exercise 1. Bi-directional “ring” implementation

Implementation of streaming messages in a bi-directional manner with OpenMPI using P processors on a ring (i.e., 1D topology). Each processor initiates a turn, with a tag proportional to its rank (P*10), and with the message to be sent equal to its rank, negative if the message is intended to be sent “forward”, and

positive if it is intended to be sent “backwards”. As each message is travelling along the ring, it gets summed with the rank it’s going through. The tags instead stay the same during the process.

The implementation of the program is relying on blocking communication with commands MPI_Send and MPI_Recv. Each processor runs a loop, in order to make in total P moves making sure its initial messages get sent across the ring reaching its source. The code itself can be found under ring.c file.

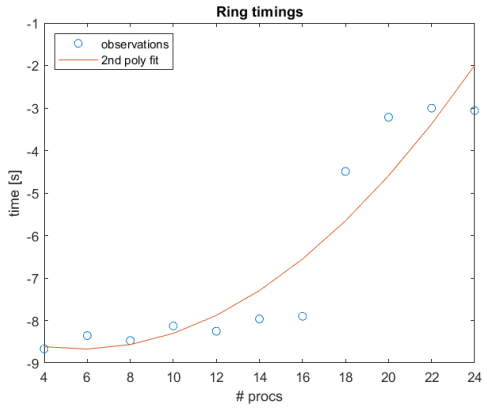
The output when the program runs on 4 processors:

> I am process 0 and I have received 4 messages. My final message has the tag 0 and value msg-backward 6 and msg-forward -6.
>
> I am process 1 and I have received 4 messages. My final message has the tag 10 and value msg-backward 6 and msg-forward -6.
>
> I am process 2 and I have received 4 messages. My final message has the tag 20 and value msg-backward 6 and msg-forward -6.
>
> I am process 3 and I have received 4 messages. My final message has the tag 30 and value msg-backward 6 and msg-forward -6.

Reporting the time that it takes for the processors to make the two full turns, we can take the max of all the times (from each processor) and the overall time of the execution can be reported.

> In the above case it is 0.00017176 s.

Running the program on more processors (every other from 4 to 24), we get the following trend:



The problem in this case is not constant regardless of the number of processors, so naturally, by adding more processors we are able to solve more message passing as we increase the number of processors, hence we have the increase also in time to execution. The increase is slightly quadratic though since for small number of processors we get similar timings.

Exercise 2. 3D matrix addition

Summing of 3D matrices, A and B, and store the result in 3D matrix C.

Combinations of 3D matrix (cubes) sizes: 2400x100x100, 1200x200x100, 800x300x100.

The scope of the exercise is to compare the summing timings of 3 combinations of cube sizes based on different distribution in the 3D topology. To the understanding of the reader (me), the task is to always use a 3D virtual topology (for contiguous memory space simulation), but to distribute (to scatter) the full cubes (A and B) as sub-arrays, sub-matrices and sub-cubes to get the 1D, 2D, 3D distribution of data and define which distribution best fits for each of the 3 cube size combination.

The implementation of the 1D distribution can be found under sum3Dmatrix.c file. The implementation of 2D and 3D distribution is lacking due to the limited skills in C programming. The general idea is to allocate 1D, 2D and 3D arrays respectively, and scatter those. Then, as done for 1D, do a local sum on sub-objects, eventually gathering all the results to the main C object on rank 0.

The implementation of the 1D code is available also without the use of the virtual topology.

Timings of the processors that took the longest time to finish:

2400x100x100 : 0.97694499 s

1200x200x100 : 0.97712265 s

800x300x100 : 0.97221219 s

Section 2. Benchmarking PingPong

For the measurement of latency and effective bandwidth the PingPong benchmark is frequently used. The code sends a message of size N [bytes] once back and forth between two processes running. The processes can be running on two different nodes, two different sockets within one node, two different cores within one socket; furthermore, the latency and bandwidth can be checked on Infiniband 100 Gbit/s 25 Gbit/s network, on Ethernet network, etc. depending on the topology of the cluster being tested.

Herein plotted the observed (called estimated in assignment description) values for the different combinations and the predicted (called computed in the assignment description) values for each, based on least squares fit.

Both axes have been scaled logarithmically in this case because this makes it easier to judge the fit quality on all scales. Also, the sizes have been scaled logarithmically.

The fitted line is a polynomial of order 1.

$$p = a + bx$$

When reporting the predicted values, the transformation has been done to linear scale, by the formula:

$$fit = x^a * exp(b)$$

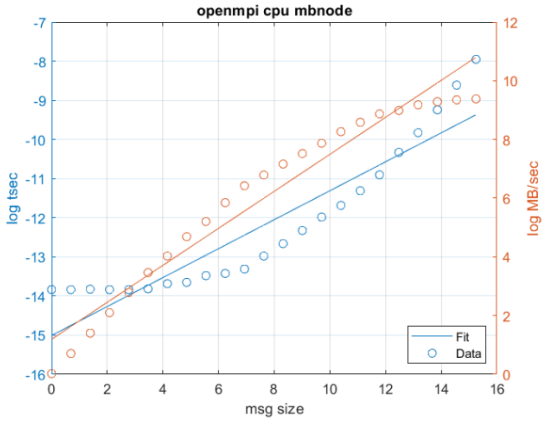
and the same approach for both time and bandwidth fit.

Reports on Infiniband

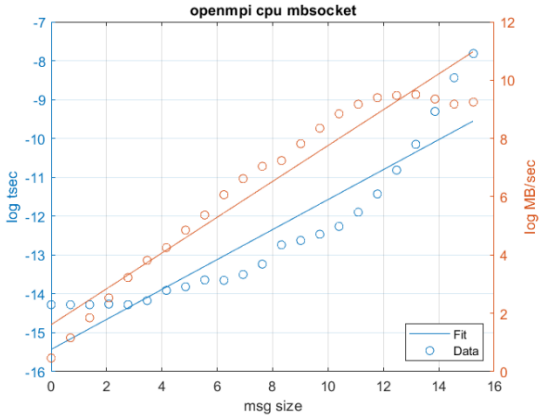
Herein the reports and plots for some of the variants tried IB, namely dividing work among nodes, sockets and cores, thin and gpu nodes, openmpi and intel benchmarks.

Thin nodes

mpirun -np 2 --report-bindings --map-by node ./IMB-MPI1 PingPong					
Eg. thin 008 and 009					
Lambda 0.36, bandwidth 0.63					
size	repetitions	tusec	Mbytessec	tusec_computed	mbsec_computed
0	1000	0.97	0	0.97	
1	1000	0.98	1.02	0.303445	3.294
2	1000	0.98	2.03	0.392109	5.0985
4	1000	0.99	4.06	0.50668	7.8916
8	1000	0.98	8.16	0.654727	12.2146
16	1000	0.98	16.26	0.846031	18.9059
32	1000	1	32.04	1.093234	29.2627
64	1000	1.14	56.21	1.412666	45.2931
128	1000	1.18	108.77	1.825434	70.105
256	1000	1.4	182.26	2.358808	108.5092
512	1000	1.48	346.11	3.048029	167.9515
1024	1000	1.66	616.66	3.938634	259.9567
2048	1000	2.31	888.19	5.089465	402.3633
4096	1000	3.17	1290.55	6.576558	622.7814
8192	1000	4.44	1844.6	8.498165	963.9465
16384	1000	6.25	2621.81	10.98125	1492.005
32768	1000	8.43	3888.83	14.18986	2309.338
65536	640	12.26	5345.97	18.33601	3574.414
131072	320	18.39	7128.93	23.69361	5532.509
262144	160	32.67	8023.72	30.61666	8563.265
524288	80	54	9708.34	39.56255	13254.3
1048576	40	96.91	10820.5	51.12235	20515.12
2097152	20	182.24	11507.75	66.05982	31753.48
4194304	10	352.57	11896.25	85.36186	49148.31

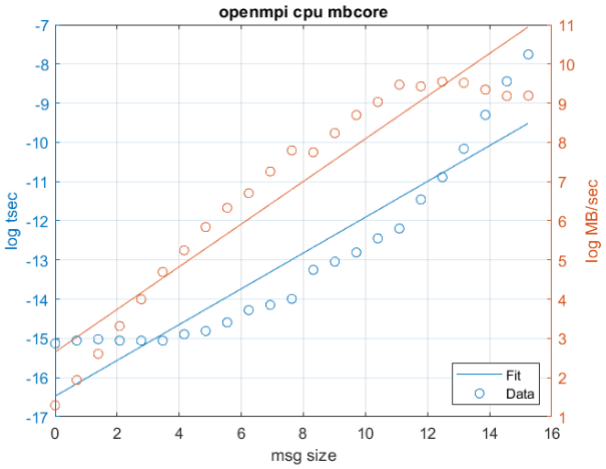


mpirun -np 2 --report-bindings --map-by socket ./IMB-MPI1 PingPong					
Eg thin008					
size	repetitions	tusec	Mbytessec	tusec_computed	mbsec_computed
0	1000	0.56	0	0.56	
1	1000	0.63	1.59	0.199378	5.0142
2	1000	0.63	3.19	0.260456	7.677
4	1000	0.63	6.33	0.340246	11.7538
8	1000	0.64	12.51	0.444479	17.9954
16	1000	0.63	25.25	0.580642	27.5517
32	1000	0.7	45.39	0.758519	42.1827
64	1000	0.91	70.43	0.990888	64.5834
128	1000	1	128.18	1.294441	98.8797
256	1000	1.19	215.53	1.690986	151.3885
512	1000	1.18	432.2	2.209011	231.7817
1024	1000	1.37	750.15	2.88573	354.8667
2048	1000	1.79	1147.12	3.769759	543.3145
4096	1000	2.94	1395.24	4.924605	831.8353
8192	1000	3.29	2491.01	6.433232	1273.572
16384	1000	3.86	4245.11	8.40402	1949.887
32768	1000	4.72	6949.21	10.97855	2985.353
65536	640	6.8	9638.83	14.34177	4570.689
131072	320	10.87	12060.38	18.7353	6997.901
262144	160	20.11	13036.39	24.47476	10714.05
524288	80	39.05	13427.16	31.97248	16403.63
1048576	40	91.03	11518.82	41.76708	25114.59
2097152	20	216.62	9681.34	54.56221	38451.4
4194304	10	403.46	10395.91	71.27706	58870.56



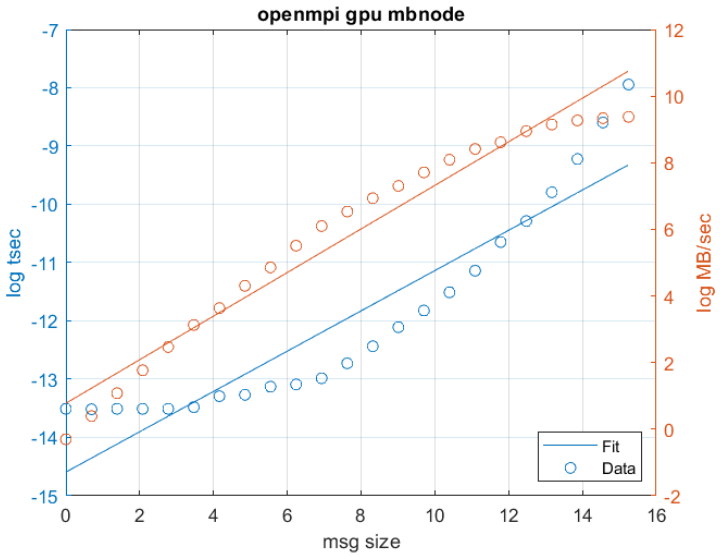
mpirun -np 2 --report-bindings --map-by core ./IMB-MPI1 PingPong

Eg. Thin008					
Lambda 0.45, Bandwidth 0.54					
size	repetitions	tusec	Mbytessec	tusec_computed	mbsec_computed
0	1000	0.24	0	0.24	
1	1000	0.27	3.65	0.070211	14.2305
2	1000	0.29	6.95	0.096335	20.744
4	1000	0.3	13.55	0.13218	30.239
8	1000	0.29	27.51	0.181361	44.08
16	1000	0.29	54.53	0.248843	64.2562
32	1000	0.29	109.19	0.341433	93.6675
64	1000	0.34	189.74	0.468475	136.541
128	1000	0.37	344.26	0.642787	199.0384
256	1000	0.46	561.47	0.881957	290.1422
512	1000	0.63	815.21	1.210119	422.9459
1024	1000	0.72	1420.84	1.660384	616.5364
2048	1000	0.84	2429.65	2.278185	898.7371
4096	1000	1.76	2320.73	3.125859	1310.107
8192	1000	2.17	3778.4	4.288939	1909.768
16384	1000	2.74	5982.87	5.884782	2783.906
32768	1000	3.92	8368.45	8.074411	4058.153
65536	640	5.05	12977.33	11.07876	5915.649
131072	320	10.56	12410.58	15.20099	8623.356
262144	160	18.7	14016.33	20.85702	12570.43
524288	80	38.52	13611.36	28.61757	18324.16
1048576	40	91.49	11460.85	39.26569	26711.49
2097152	20	215.46	9733.4	53.8758	38937.86
4194304	10	427.41	9813.26	73.92209	56760.47



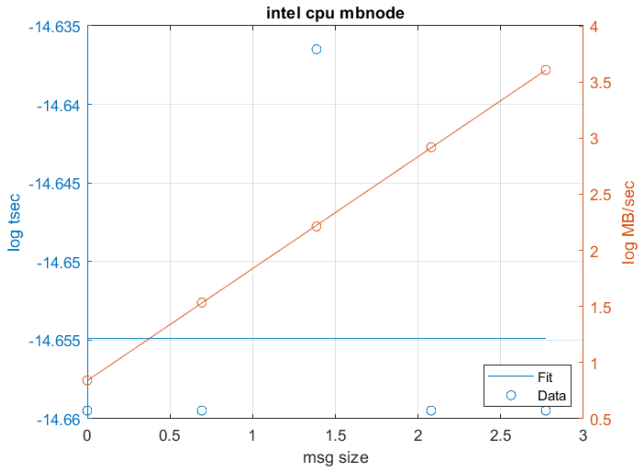
GPU nodes

mpirun -np 2 --report-bindings --map-by node ./IMB-MPI1 PingPong					
Eg. thin 08 and 09					
size	repetitions	tusec	Mbytessec	tusec_computed	mbsec_computed
0	1000	1.36	0	0	0
1	1000	1.36	0.74	0.459918	2.1777
2	1000	1.35	1.48	0.584421	3.4272
4	1000	1.36	2.95	0.742628	5.3936
8	1000	1.36	5.87	0.943663	8.4883
16	1000	1.36	11.77	1.199119	13.3587
32	1000	1.4	22.89	1.523729	21.0236
64	1000	1.69	37.96	1.936214	33.0864
128	1000	1.73	74.19	2.460362	52.0707
256	1000	1.99	128.52	3.1264	81.9476
512	1000	2.07	247.03	3.972739	128.9672
1024	1000	2.3	445.33	5.048188	202.9656
2048	1000	2.97	688.85	6.41477	319.4225
4096	1000	3.98	1029.23	8.151296	502.6997
8192	1000	5.52	1485.28	10.35791	791.1371
16384	1000	7.35	2228.76	13.16187	1245.073
32768	1000	10.04	3264.3	16.72489	1959.467
65536	640	14.55	4505.73	21.25244	3083.764
131072	320	23.76	5516.79	27.00563	4853.156
262144	160	34.07	7693.21	34.31626	7637.783
524288	80	55.87	9384.84	43.60592	12020.17
1048576	40	98.81	10611.72	55.41036	18917.06
2097152	20	184.81	11347.69	70.41035	29771.22
4194304	10	354.55	11829.86	89.47095	46853.26



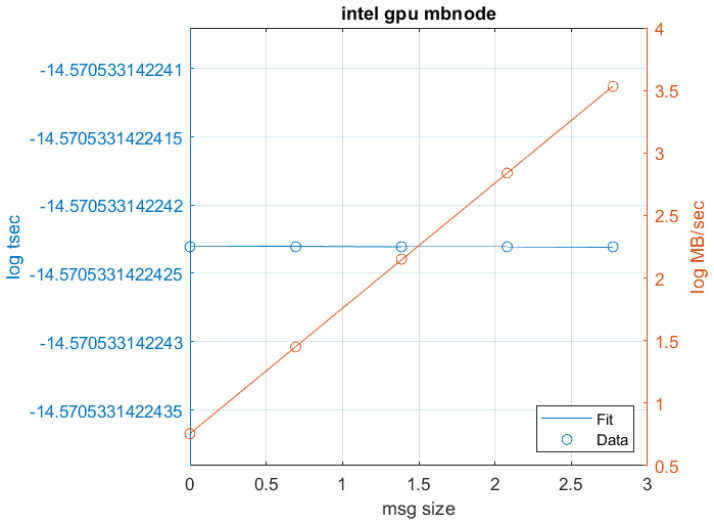
Intel library

mpirun -np 2 -env --map-by node ./IMB-MPI1 PingPong -msglog 4					
Eg. thin 08 and 09					
size	repetitions	tusec	Mbytessec	tusec_computed	mbsec_computed
0	1000	0.43	0	0	0
1	1000	0.43	2.32	0.431982	2.3165
2	1000	0.43	4.64	0.431982	4.6247
4	1000	0.44	9.14	0.431982	9.2328
8	1000	0.43	18.51	0.431982	18.4328
16	1000	0.43	36.84	0.431982	36.7998



mpirun -np 2 --map-by node ./IMB-MPI1 PingPong -msglog 4 NOTE: on dssc_gpu
Eg. thin 08 and 09

size	repetitions	tusec	Mbytessec	tusec_computed	mbsec_computed
0	1000	0.47	0	0	0
1	1000	0.47	2.13	0.47	2.1324
2	1000	0.47	4.26	0.47	4.2685
4	1000	0.47	8.59	0.47	8.5445
8	1000	0.47	17.1	0.47	17.1039
16	1000	0.47	34.17	0.47	34.238



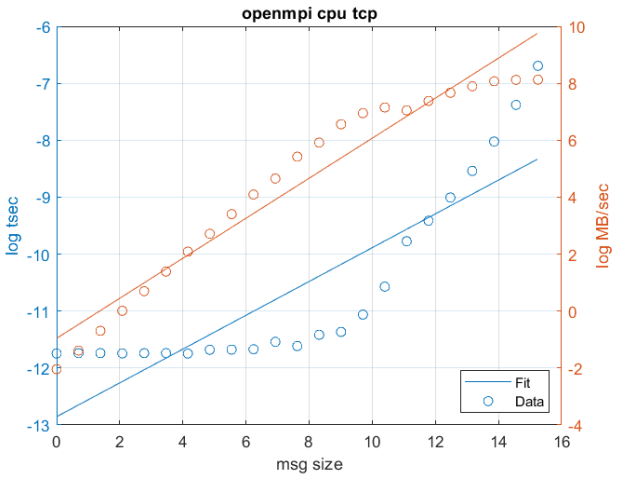
Reports on Ethernet band (TCP/IP)

Herein the reports and plots for some of the variants tried on TCP, namely dividing work among nodes, sockets and cores, thin and gpu nodes, openmpi and intel benchmarks.

Thin nodes

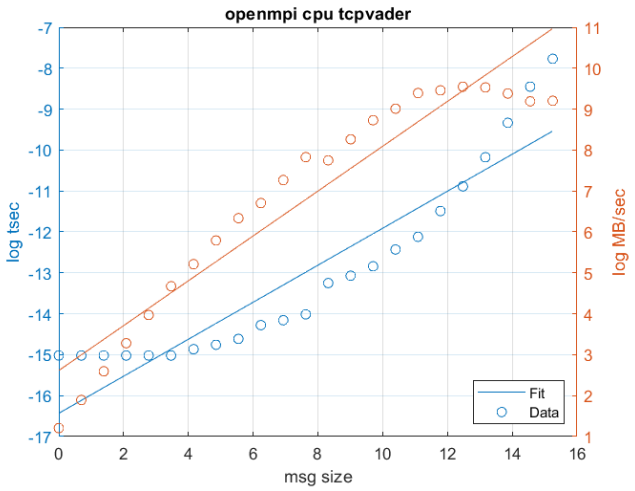
mpirun -np 2 --report-bindings --mca pml ob1 --mca btl tcp,self ./IMB-MPI1 PingPong					
Probably has chosen one node					
size	repetitions	tusec	Mbytessec	tusec_computed	mbsec_computed
0	1000	7.91	0	0	0
1	1000	7.93	0.13	2.61436	0.3843
2	1000	7.97	0.25	3.211471	0.6256
4	1000	7.99	0.5	3.944961	1.0182
8	1000	7.94	1.01	4.845977	1.6572
16	1000	7.98	2.01	5.952782	2.6974
32	1000	7.98	4.01	7.312378	4.3903
64	1000	7.92	8.08	8.982501	7.1458
128	1000	8.47	15.12	11.03407	11.6306

256	1000	8.49	30.14	13.55422	18.9303
512	1000	8.55	59.88	16.64996	30.8114
1024	1000	9.74	105.11	20.45276	50.1493
2048	1000	9.04	226.54	25.1241	81.6242
4096	1000	11.03	371.33	30.86236	132.8534
8192	1000	11.59	707.01	37.91122	216.2353
16384	1000	15.7	1043.79	46.57002	351.9497
32768	1000	25.66	1277.18	57.20646	572.8415
65536	640	56.92	1151.46	70.27223	932.3701
131072	320	81.65	1605.34	86.32218	1517.547
262144	160	122.68	2136.85	106.0379	2469.996
524288	80	195.55	2681.1	130.2566	4020.222
1048576	40	327.86	3198.25	160.0068	6543.407
2097152	20	623.92	3361.22	196.5518	10650.2
4194304	10	1238.46	3386.71	241.4436	17334.52



mpirun -np 2 --report-bindings --mca pml ob1 --mca btl tcp,vader,self ./IMB-MPI1 PingPong					
On one thin node					
size	repetitions	tusec	Mbytessec	tusec_computed	mbsec_computed
0	1000	0.25	0	0	0
1	1000	0.3	3.34	0.073157	13.6585
2	1000	0.3	6.64	0.100083	19.9685
4	1000	0.3	13.4	0.136918	29.1934
8	1000	0.3	26.6	0.187311	42.68
16	1000	0.3	52.97	0.256252	62.3971
32	1000	0.3	107.05	0.350566	91.2231
64	1000	0.35	183.6	0.479592	133.3659
128	1000	0.39	326.98	0.656107	194.9777

256	1000	0.45	564.08	0.897589	285.0526
512	1000	0.63	816.44	1.227949	416.7399
1024	1000	0.71	1433.83	1.679898	609.2635
2048	1000	0.82	2509.06	2.298188	890.7283
4096	1000	1.76	2323.68	3.144042	1302.223
8192	1000	2.11	3881.27	4.301213	1903.818
16384	1000	2.65	6176.49	5.884284	2783.335
32768	1000	4	8192.8	8.050008	4069.167
65536	640	5.46	12007.13	11.01283	5949.023
131072	320	10.22	12822.73	15.06613	8697.325
262144	160	18.71	14007.55	20.61125	12715.28
524288	80	38.07	13770.49	28.19727	18589.42
1048576	40	88.32	11872.46	38.57534	27177.28
2097152	20	213.95	9801.95	52.77308	39732.51
4194304	10	422.29	9932.23	72.19633	58087.96



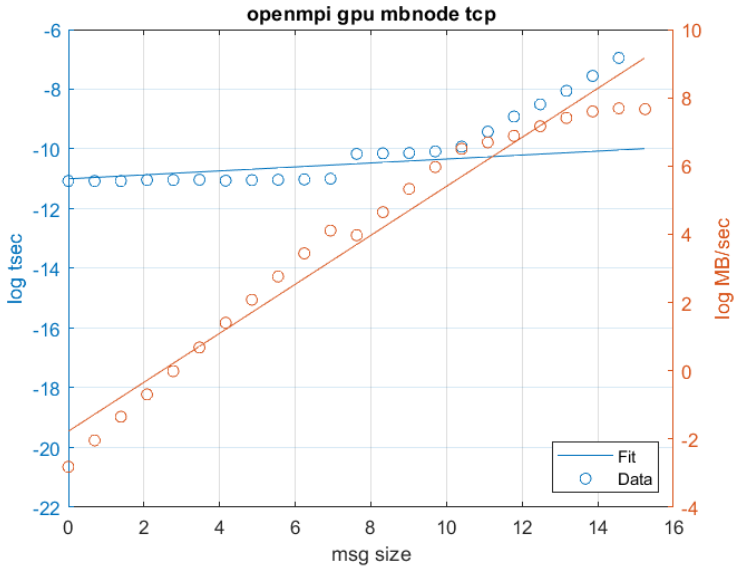
mpirun -np 2 --mca pml ob1 --report-bindings --oversubscribe --map-by node --mca btl tcp,self ./IMB-MPI1 PingPong NOTE: on dssc					
size	repetitions	tusec	Mbytessec	tusec_computed	mbsec_computed
0	1000	16.2	0	0	0
1	1000	16.11	0.06	6.516664	0.1534
2	1000	16	0.13	7.815613	0.2558
4	1000	16.12	0.25	9.373477	0.4266
8	1000	16.16	0.49	11.24187	0.7115
16	1000	16.11	0.99	13.48268	1.1865
32	1000	16.16	1.98	16.17014	1.9787
64	1000	16.29	3.93	19.39329	3.2997
128	1000	17.06	7.5	23.2589	5.5027
256	1000	17.08	14.99	27.89503	9.1764
512	1000	17.11	29.92	33.45527	15.3029

1024	1000	17.89	57.24	40.12382	25.5196
2048	1000	39.46	51.9	48.12159	42.5574
4096	1000	39.74	103.08	57.71354	70.9702
8192	1000	40.86	200.49	69.21742	118.3522
16384	1000	43.45	377.08	83.01433	197.3682
32768	1000	50.43	649.76	99.56135	329.1378
65536	640	63.57	1030.94	119.4066	548.8813
131072	320	117.05	1119.8	143.2076	915.333
262144	160	165.92	1579.98	171.7528	1526.44
524288	80	244.29	2146.17	205.9878	2545.543
1048576	40	394.94	2655.05	247.0468	4245.034
2097152	20	732.04	2864.79	296.29	7079.161
4194304	10	1589.12	2639.39	355.3486	11805.45

GPU nodes

mpirun -np 2 --mca pml ob1 --report-bindings --oversubscribe --map-by node --mca btl tcp,self ./IMB-MPI1 PingPong NOTE: on dssc_gpu					
Eg. thin 08 and 09					
size	repetitions	tusec	Mbytessec	tusec_computed	mbsec_computed
0	1000	15.41	0	15.41	
1	1000	15.57	0.06	16.7008	0.1701
2	1000	15.54	0.13	17.48651	0.2797
4	1000	15.54	0.26	18.30919	0.46
8	1000	16.12	0.5	19.17057	0.7565
16	1000	16.13	0.99	20.07248	1.2441
32	1000	16.19	1.98	21.01682	2.0461
64	1000	15.67	4.08	22.00559	3.365
128	1000	15.96	8.02	23.04088	5.5341
256	1000	16.19	15.81	24.12487	9.1015
512	1000	16.39	31.23	25.25987	14.9684
1024	1000	16.79	61.01	26.44825	24.6172
2048	1000	38.53	53.15	27.69255	40.4859
4096	1000	39.2	104.48	28.99539	66.5837
8192	1000	39.67	206.5	30.35953	109.5044
16384	1000	41.75	392.42	31.78784	180.0926
32768	1000	49.1	667.32	33.28334	296.1828
65536	640	81.05	808.63	34.84921	487.1066
131072	320	134.36	975.52	36.48875	801.1024
262144	160	202.48	1294.69	38.20541	1317.505
524288	80	317.84	1649.52	40.00285	2166.787

1048576	40	524.23	2000.23	41.88484	3563.529
2097152	20	957.16	2191.01	43.85538	5860.631
4194304	10	0.001961	2138.6	45.91862	9638.478



Summary

Infiniband vs Ethernet

Observing the values gotten, it can be concluded that, as expected, running the communication on Infiniband is desirable. Ethernet is performing more slowly, almost for a factor of 10 in case of thin nodes without specifying the topology, and even worse than factor of 10 for both kind of nodes (thin and gpu), when mapped by node as topology.

GPU vs Thin with OpenMPI

Observing the worst option for topology, mapping by node, it is noted that the thin node performs better when it comes to latency, 0.97 usec against 1.36 usec of the GPU nodes.

As for mapping by socket and core (in this document not reported for gpu, see outputs_openmpi_gpu.txt), the results are utmost similar on the other hand.

Intel bench vs OpenMPI

The benchmark for running communication between two nodes, shows rather different results when running on intel and openMPI libraries. ~ 45 usec (mean of gpu and thin on intel) against ~ 120 usec (mean of gpu and thin on openmpi) for latency.

Annex – fits below :

Fit values of coeffs of the first degree for lambda and bandwidth respectively:

-2.1145e-15	0.9974	'intel_cpu_mbnode'
-3.2686e-15	1.0013	'intel_gpu_mbnode'
0.4564	0.5437	'openmpi_cpu_mbcore'
0.3698	0.6302	'openmpi_cpu_mbnode'
0.2622	0.7378	'openmpi_cpu_mbnode_tcp'
0.3855	0.6145	'openmpi_cpu_mbsocket'
0.2968	0.7028	'openmpi_cpu_tcp'
0.4521	0.5479	'openmpi_cpu_tcpvader'
0.4529	0.5473	'openmpi_cpu_vader'
0.3456	0.6542	'openmpi_gpu_mbnode'
0.0663	0.7177	'openmpi_gpu_mbnode_tcp'

Section 3. Jacobi

The Jacobi method is prototypical for many stencil-based iterative methods in numerical analysis and simulation. In its most straightforward form, it can be used for solving the diffusion equation for a scalar function Φ(~r,t),

$$\frac{\partial \Phi}{\partial t} = \Delta \Phi$$

on a rectangular lattice subject to Dirichlet boundary conditions.

In the following benchmark we explore strong scaling, namely the performance of the model with fixed input size (work) as the resources increase to solve the same problem.

To estimate performance the formula below is used.

$$P(L,\sim N) = \frac{L^3 * N}{Ts(L) + Tc(L,\sim N)}$$

L^3 being the problem size per process, N being the total number of processes used (NX*NY*NZ), Tc being the communication time. Tc depends on the domain cuts decided on:

$$Tc(L,\sim N) = c(L,\sim N) * B + k * T\ell$$

Where B and Tl have been extrapolated from the outputs during the PingPong benchmarking, and c(L,~N) can be derived from the Cartesian decomposition:

$$c(L,\sim N) = L^2 * k * 2 * 8$$

k being the number of domain cuts larger than 1.

In contrast to weak scaling, the single-process performance on the subdomain size is hard to predict since it depends on many factors (pipelining effects, prefetching, spatial blocking strategy, copying to intermediate buffers, etc.). To address this, we run the program on single processor on both thin and GPU nodes and take as baseline the prediction for parallel performance in those conditions, namely Ts in the formula of P.

Specific computational resources were required for each of the three sections (same node, across nodes, gpu and by topology) by qsub command:

qsub -l nodes=**1/2**:ppn=24 -q **dssc_gpu/dssc** -l walltime=1:00:00 -I

The program Jacobi supplied from online resources was compiled using the command:

mpif77 -ffized-line-length-none Jacobi_MPI_vectormode.F -o jacobi3D.x

Whereas for the different combinations the following mpirun was executed:

mpirun -np **4/8/12/24/48** --map-by **node/sockets/core** jacobi3D.x 2>/dev/null.

In bold reporting the changes made to the commands according to the different requirements.

Same thin node

--map-by	Tl s	B	Ts	L	N	Nx	Ny	Nz	k	C(L,N)	Tcomm	Tc(L,N)	P(L,N)	P(1)*N/P(L,N)
socket	0.56	12060	15.04	1200	4	2	2	1	4	38400	3.764823	5.42408	337762564.2	1
core	0.24	14016	15.04	1200	4	2	2	1	4	38400	7.658015	3.69973	368842105.3	1.09
socket	0.56	12060	15.04	1200	8	4	2	1	4	38400	1.928692	5.42408	675525128.4	1.09
core	0.24	14016	15.04	1200	8	4	2	1	4	38400	1.921432	3.69973	737684210.5	1.17
socket	0.56	12060	15.04	1200	12	4	3	1	4	38400	1.255236	5.42408	1013287693	1.17
core	0.24	14016	15.04	1200	12	4	3	1	4	38400	1.268727	3.69973	1106526316	1.26

Across thin nodes

--map-by	Tl s	B	Ts	L	N	Nx	Ny	Nz	k	C(L,N)	Tcomm	Tc(L,N)	P(L,N)	P(1)*N/P(L,N)
	0.97	11900	15.04	1200	12	4	3	1	4	38400	1.255	7.106891	936293957.8	1
	0.97	11900	15.04	1200	24	12	2	1	4	38400	0.633	7.106891	1872587916	1.09
	0.97	11900	15.04	1200	48	12	2	2	6	57600	0.633	10.66034	3227350785	1.09

GPU on all 48 threads

--map-by	Tl s	B	Ts	L	N	Nx	Ny	Nz	k	C(L,N)	Tcomm	Tc(L,N)	P(L,N)	P(1)*N/P(L,N)
	0.26	14000	23.26	1200	12	4	3	1	4	38400	1.2549147	3.783	766655513.6	1
	0.26	14000	23.26	1200	24	12	2	1	4	38400	0.6329289	3.783	1533311027	1.09
	0.26	14000	23.26	1200	48	12	2	2	6	57600	0.6334761	5.674	2866188696	1.09

Hyperthreading for GPU nodes are enabled by default, hence taking advantage of one GPU node, with its 2 sockets, each with all 12 cores/processors leads to 48 processes running “in parallel”.

Summary

As a summary for all three benchmarks, it can be concluded that even though a linear scalability for Tcomm (observed time) would have been expected, it was not always strictly such. On the other hand, as noted in a benchmark supplied in class lectures, Tc increases with more computational power, which is consistent also in the above reported benchmarks.