Università degli studi di trieste

Assignment 1

Silvio Baratto

30 dicembre 2021

Indice

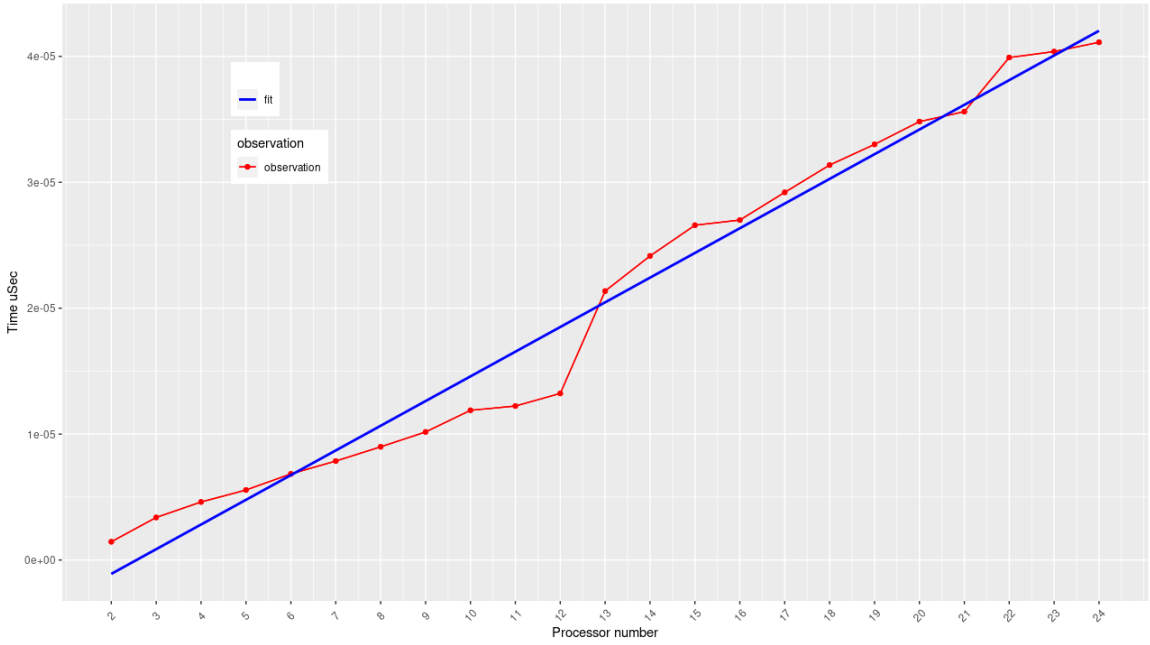
Capitolo 1

section 1

All the files used for this report are in: https://github.com/SilvioBaratto/HPC

1.1 Ring

This program builds a 1D ring and makes each process sends a message to his right and left neighbour with a tag proportional to its rank. In this way each process add or substracts its rank value depending on which side the message is received. Then each process will resend the messages along the ring until the original message has come back to the original sender.
To obtain significant data, I have taken the mean over 10000 repetitions. The figure below shows the time as a function of the number of processors used, as expected the time grows linearly.



1.2 Matrix

The problem consists in writing a program which performs the sum between 2 3D matrices in parallel, with different virtual topologies.

1.2.1 Solution

A possible solution is to initialize A and B on the root process, then to scatter portions of the 2 matrices to each processes Scatter(), performing the sum between each corresponding block and gathering the results on root Gather().
By setting the number of process to 24, three different domains are analyzed for each of the 3 different topologies (1D, 2D and 3D) and the computational time is recorded. The tables do not show any significant differences among the combinations and the topologies.

Distribution	Topology	Time
(2400 x 100 x 100)	(24, 1, 1)	0.75044058
(2400 x 100 x 100)	(12, 2, 1)	0.69981940
(2400 x 100 x 100)	(6, 2, 2)	0.75580620
(2400 x 100 x 100)	(4, 3, 2)	0.70167095
(1200 x 200 x 100)	(24, 1, 1)	0.74583788
(1200 x 200 x 100)	(12, 2, 1)	0.69193223
(1200 x 200 x 100)	(6, 2, 2)	0.75146050
(1200 x 200 x 100)	(4, 3, 2)	0.69529179
(800 x 300 x 100)	(24, 1, 1)	0.75275431
(800 x 300 x 100)	(12, 2, 1)	0.76012270
(800 x 300 x 100)	(6, 2, 2)	0.69036465
(800 x 300 x 100)	(4, 3, 2)	0.68957827

Capitolo 2

section 2

2.1 MPI PingPong performance

The PingPong benchmark consists of a message of N bytes sent back and forth between two processes. The output reports the size of the message in number of bytes, the number of repetitions, the communication time in μs and the effective bandwidth in $Mbytes/s$. The analyses are performed on GPU and Thin nodes. For each type of node, the communication between cores inside the same socket is modeled, across 2 different sockets, and across two different nodes. The assumed model for the communication time is:

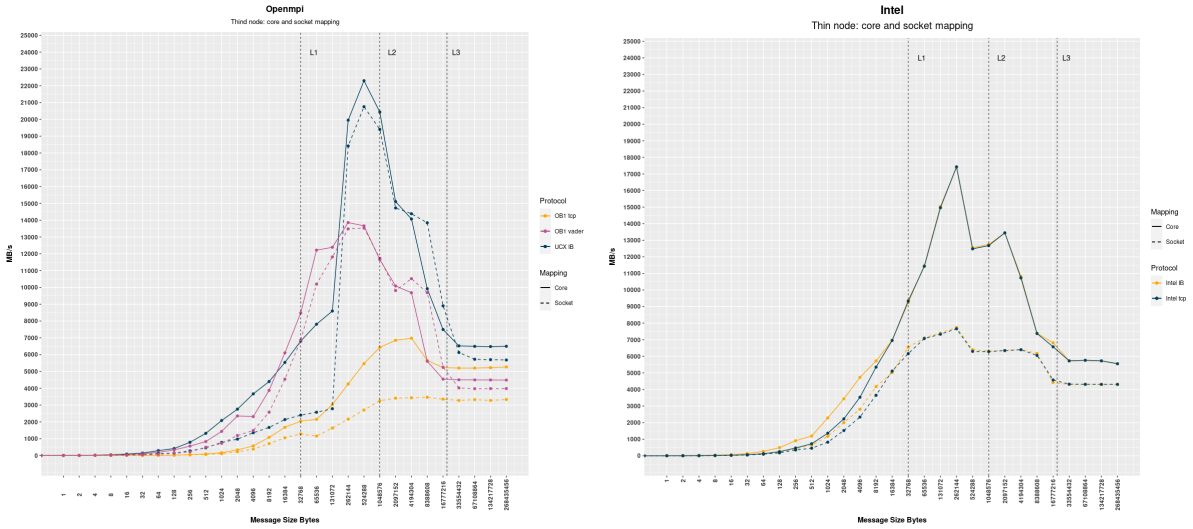
$$T_{comm} = \lambda + \frac{m_{size}}{b_{network}}$$

where λ is the latency, m_{size} is the size of message, and $b_{network}$ the bandwidth.

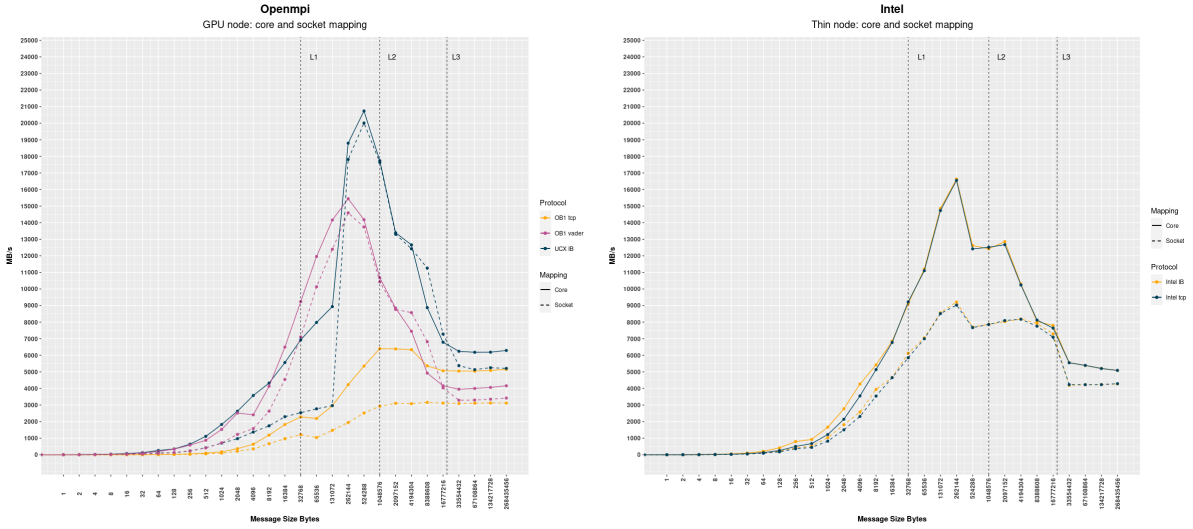
Thin node Openmpi and Intel between sockets and cores

Inside socket.
The plot shows a peak at a message size between 10^5 and 10^7 bytes. To explain this behavior we can consider the caches and the architecture. A single core of a node has an L1-cache of size $32KB$ and L2-cache of $1024KB$. The L2-cache is shared among the cores of one socket. To take measurements the PingPong benchmark, given a small message size repeats the sending of the message several times until it fits in the L1-cache. When the L1-cache is completely full part of the message will be stored inside the L2-cache. The same reasoning can be applied between L2-cache and L3-cache. Up to $1MB$ the message is entirely stored inside the L2-cache, but as the size increases the data retrieved process from L3-cache slows down the process, untill the region inside which the asymptotic bandwidth can be measured is reached.

Across socket.
Analogous results are obtained when pinning processes across 2 different sockets of the same thin node.

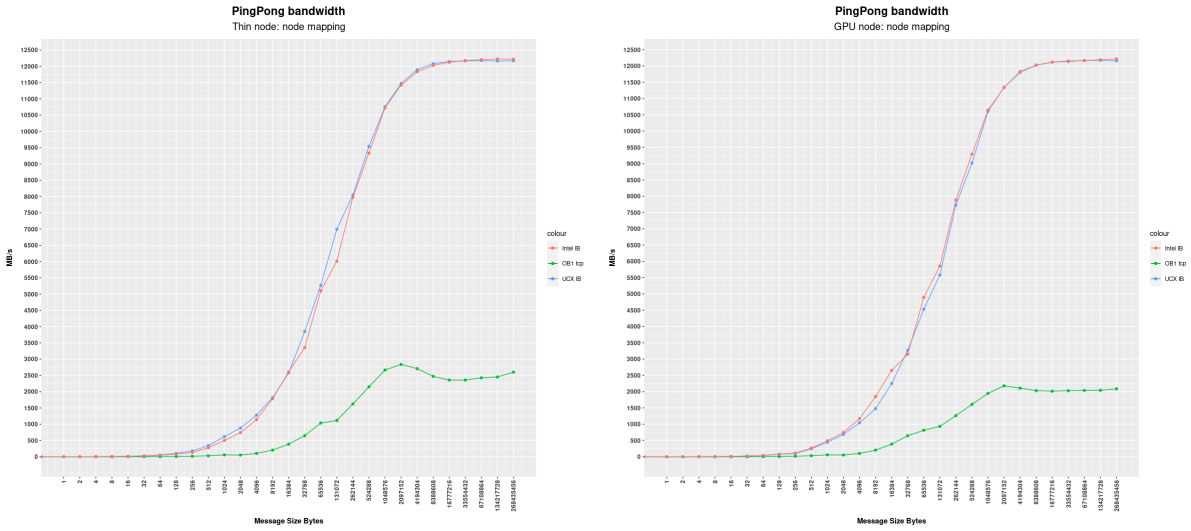


Thin node Openmpi and Intel between sockets and cores



Across nodes.
Have observed that without specifying any btl parameter, the messages are exchanged through the 100 Gbit HDR Infiniband which connect the computational nodes in the ORFEO cluster, in this case the values are comparable with the theoretical ones.

Thin node and Gpu node between nodes



2.1.1 Thin node latency and bandwidth

Thin node	Protocol	Latency	Bandwidth
Core mapping	UCX ib	0.22	6450.36
	Intel ib	0.24	5702.59
	OB1 vader	0.31	4472.75
	OB1 tcp	5.5	5123.82
Socket mapping	UCX ib	0.54	5679.78
	Intel ib	0.48	4289.15
	OB1 vader	0.72	3954.68
	OB1 tcp	8.19	3284.49
Node mapping	Openmpi ib	1.07	12176.39
	Intel ib	1.34	12235.85
	OB1 vader	1.06	12187.99
	OB1 tcp	15.76	2402.39

2.1.2 Gpu node latency and bandwidth

Gpu node	Protocol	Latency	Bandwidth
Core mapping	UCX ib	0.26	6157.24
	Intel ib	0.32	5216.97
	OB1 vader	0.30	4030.76
	OB1 tcp	5.03	5072.48
Socket mapping	UCX ib	0.60	5201.56
	Intel ib	0.53	4209.24
	OB1 vader	0.71	3318.39
	OB1 tcp	8.73	3121.66

Capitolo 3

section 3

3.1 Jacobi solver

3.1.1 Performance

To predict the performance of the Jacobi model we use the following formula:

$$P(L,N) = \frac{L^3 N}{T_s + T_c}[MLUP/s]$$

L is the size of cubic sub-domains, T_s is the time for the lattice updates of a domain with size L and T_c is the communication time. The quantity T_s can be modeled estimating the latency λ, bandwidth B and messages size C.

$$T_c = \frac{C(L,N)}{B} + 4k\lambda[s]$$

$C(L,N)$ is the maximum bidirectional bandwidth data volume transferred over a node's network link, k is the largest number of coordinate directions in which the number of processes is greater than one and λ is the latency. The formula for C is:

$$C = L^2 \times k \times 2 \times 8[byte]$$

3.1.2 Results

The following tables compare the results obtained by running Jacobi with the performance model. In thin nodes, the performance $P(N,L)$ predicted from theoretical models reflect the real performance obtained from computation well. In Gpu nodes, instead there is an overestimation from the model, this could happen because hyper-threading enabled is on this node.

Thin node, core mapping

$\lambda = 0.22$, Bandwidth $= 6450.36$

N	k	C[Mb]	Tc/s	MLUP/s est	MLUP/s real	MLUP/s diff	NP(1)P(N)
1	2	46.08	0.00714423	112.741	112.738	−0.00312003	1
4	4	92.16	0.0142885	450.755	451.917	1.16281	0.997864
8	6	138.24	0.0214327	901.089	890.34	−10.749	1.01299
12	6	138.24	0.0214327	1351.63	1325.43	−26.2079	1.0207

Thin node, socket mapping

$\lambda = 0.54$, Bandwidth = 5679.78

N	k	C[Mb]	Tc/s	MLUP/s est	MLUP/s real	MLUP/s diff	NP(1)P(N)
1	2	46.08	0.00811407	112.734	112.738	0.00401334	1
4	4	92.16	0.0162281	450.698	447.417	−3.28006	1.0079
8	6	138.24	0.0243422	900.918	883.429	−17.4894	1.02091
12	6	138.24	0.0243422	1351.38	1336.19	−15.1886	2.09478

Thin node, node mapping

$\lambda = 1.07$, Bandwidth = 12176.39

N	k	C[Mb]	Tc/s	MLUP/s est	MLUP/s real	MLUP/s diff	NP(1)P(N)
1	2	46.08	0.00378651	112.766	112.738	−0.0278236	1
12	6	138.24	0.0113595	1352.52	1326.58	−25.942	1.01981
24	6	138.24	0.0113595	2705.04	2627.13	−77.913	1.02991
48	6	138.24	0.0113595	5410.09	5131.27	−278.814	1.0546

Gpu node, core mapping

$\lambda = 0.26$, Bandwidth = 6157.24

N	k	C[Mb]	Tc/s	MLUP/s est	MLUP/s real	MLUP/s diff	NP(1)P(N)
1	2	46.08	0.00748439	78.1282	78.1421	0.0138761	1
4	4	92.16	0.0149688	312.407	309.684	−2.72303	1.00931
8	6	138.24	0.0224532	624.603	586.338	−38.2651	1.06617
12	6	138.24	0.0224532	936.905	850.567	−86.3376	1.10245

Gpu node, socket mapping

$\lambda = 0.60$, Bandwidth = 5201.56

N	k	C[Mb]	Tc/s	MLUP/s est	MLUP/s real	MLUP/s diff	NP(1)P(N)
1	2	46.08	0.00886008	78.1234	78.1421	0.0187353	1
4	4	92.16	0.0177202	312.368	311.43	−0.938776	1.00366
8	6	138.24	0.0265802	624.487	611.262	−13.2245	1.0227
12	6	138.24	0.0265802	936.73	899.664	−37.066	1.04228

Gpu node, node mapping

$\lambda = 1.58$, Bandwidth = 12185.46

N	k	C[Mb]	Tc/s	MLUP/s est	MLUP/s real	MLUP/s diff	NP(1)P(N)
1	2	46.08	0.00378472	78.1413	78.1421	0.000805104	1
12	6	138.24	0.0113541	937.375	897.053	−40.3223	1.04532
24	6	138.24	0.0113541	1874.75	1686.53	−188.224	1.112
48	6	138.24	0.0113541	3749.5	2488.66	−1260.83	1.50716

The biggest difference between model and data is the communication time. In fact, the time T_c predicted from the model is hundred time smaller then the observed one. However, the simple model used is good enough to predict the performance of the program.