Foundations of High Performance Computing

Assignment I

Lorenzo Trubian [mat. SM3500519]

8/12/2021

Section 1

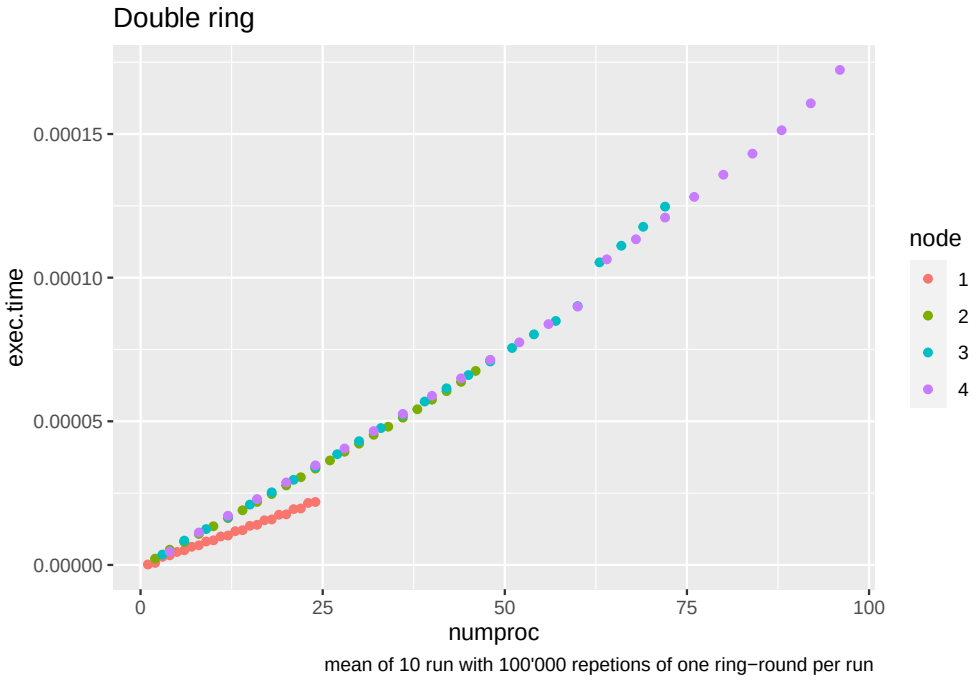
Double ring

Aside from the efficiency of the program, which was not compared with others, we can note two main facts regarding to the collected data:

- first of all the relation between the execution time and the number of processor is linear;
- and in the second place there are only two clearly different slopes.

We can explain the first note remembering that the only difference between execution with two number of processors is how many iteration are required for each processors to receive its original message. Increasing the *number of process* by one means increasing the *number of process through which the message has to pass* by one, and therefore means increasing the *number of iteration* to complete the loop by one.

The second observation is trivially explainable if we note that the main gap in the considered communication is the latency of the network: if the nodes are two, three or four the latency is always given by the Infiniband network, therefore the slopes of time execution of these three measurements are perfectly equal. We observe that in one-node scenario the latency is lower and so is the slope.



Matrix

In this subsection we can observe some obvious facts about a program that implements a domain decomposition but does not make any communication between the processes during the computation: indeed the matrix addition do not require communication aside the ones used to decompose the domain.

We start explaining how such decomposition is implemented. Given a matrix of $N_x \times N_y \times N_z$ dimensions, we considered that it is stored in memory such that the distance in memory between two elements (x_0, y_0, z_0) and (x_1, y_1, z_1) is $|x_0 - x_1| + N_x * |y_0 - y_1| + N_x * N_y * |z_0 - z_1|$. Due to this idea the best one dimensional split is on the z dimension and the two dimensional split is the one on the z and y dimensions. In case the dimension is not a multiple of the processes assigned to that direction, a round robin distribution is executed. The figure is quite explanatory. Indeed, these types of decomposition allow to send neighboring elements in memory to the same process, reducing the number of jump in memory during the communication.

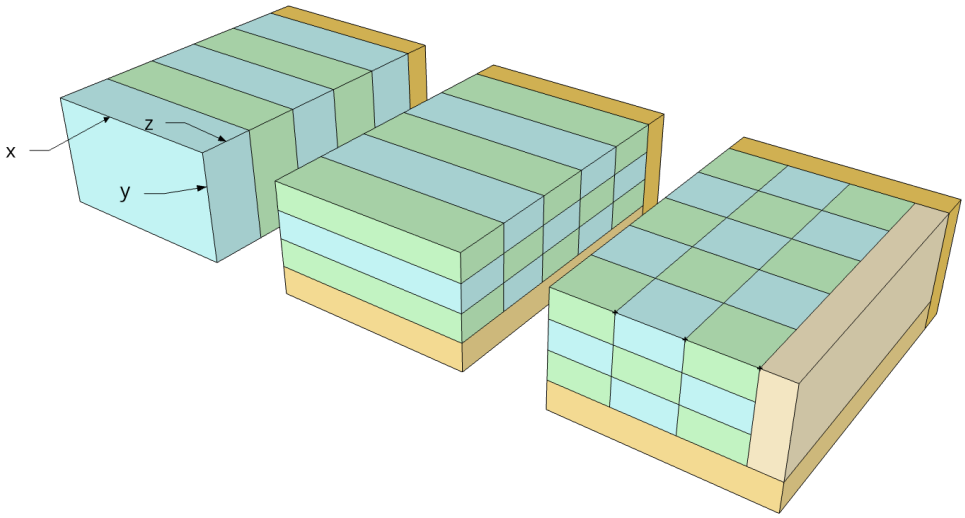


Figure 1: Scheme of 1,2 and 3 D decompositions. The orange parts are the remaining ones of the matrix when the dimension in not a multiple of the process assigned to that direction.

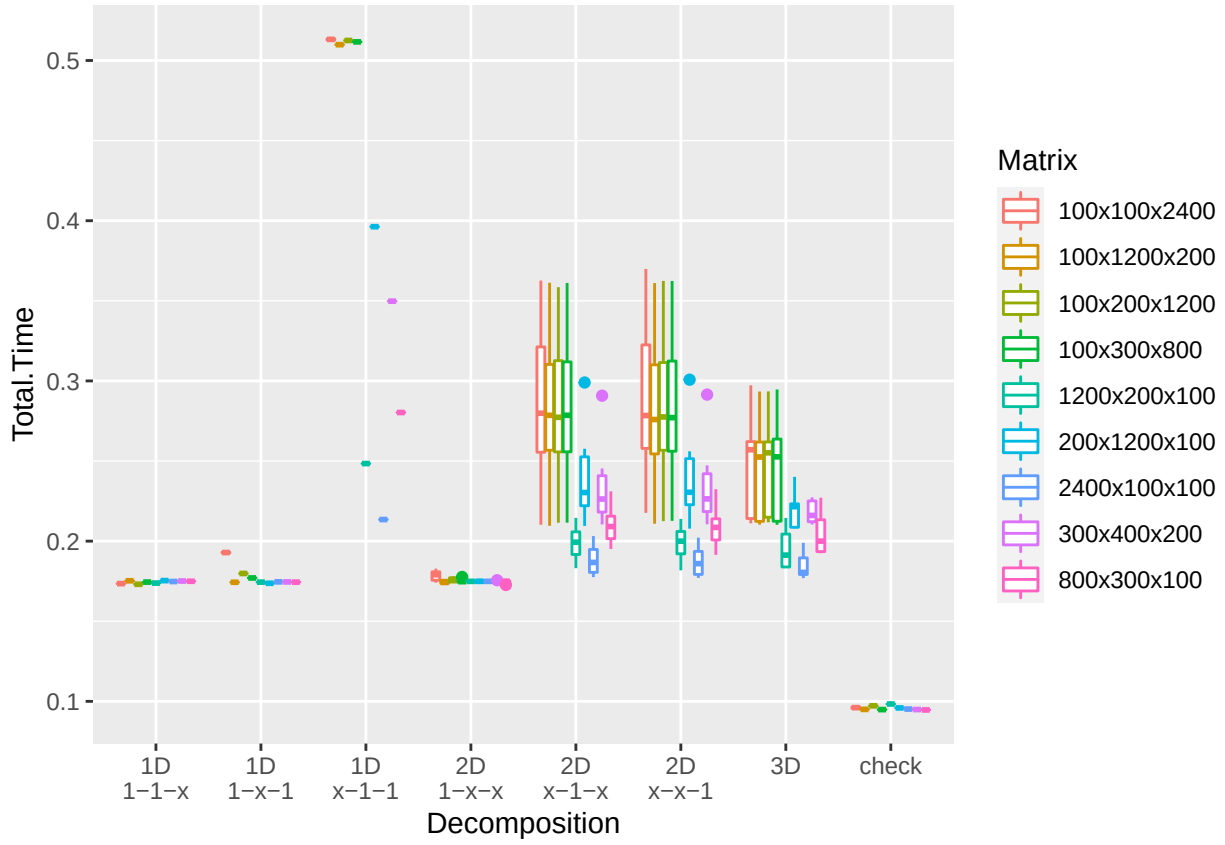
The collected data confirm this claim regarding the efficient splits. The grid of processes will be represented using the numbers of assigned process to the considered dimension: $3-4-2$ is a grid of processes where 3 is the number of split the matrix is divided into along x dimension. We write an x in any place to indicate any possible number, for example: $x-1-x$ represents all the splits in which the y dimension is not subdivided. It is important to remember that in this case the number of total processes is kept constant and so is the product of the three grid numbers. This particular notation is used cause to the fact that every possible split was considered. The main consequence is that for $2D$ and $3D$ dimension decomposition the graph will present a box with mean (and variance) over all possible splits.

The main things to be noticed is the low performance of any splits that involves a decomposition along the x dimension cause to the larger jumps in memory that this type of decomposition implies. In this case the size of the jump to be made to send a block of the matrix is (more or less) $N_x \times N_y$ and the worst choices are the $x-1-1$, $x-1-x$ and $x-x-1$. The decompositions over the y dimension are more efficient than the previous because the size of the jump is circa N_x but less robust than the ones over z (compare $1-x-1$ and $1-1-x$).

Furthermore, for each split different matrix dimensions were tested. It can be observed, for example, that the first four matrices (the ones with the larger N_z) are the ones with the worst performance due to the biggest jumps in memory done during the communication.

Finally, a program that use a simple "scatter-gather" dynamic over the same number of processes is used

as a baseline of the results (the **check** part of the graph). It is not a surprise that is far more efficient not only than the $2D$ and $3D$ decomposition (due to the absence of any jump) but also than the best $1D$ decomposition: indeed the **check** case has not the overhead given by the specific decomposition and the round-robin on the remaining part of the matrix.



Section 2: PingPong

We focus our discussion on the communication between and within thin nodes using MPI libraries. The subdivision of the observation is based on the position (different nodes, sockets or cores) of the two used processors. It is useful to recall that the model discuss during the course for the communication time is

$$T_{comm} = \lambda + \frac{(\text{size of message})}{b}$$

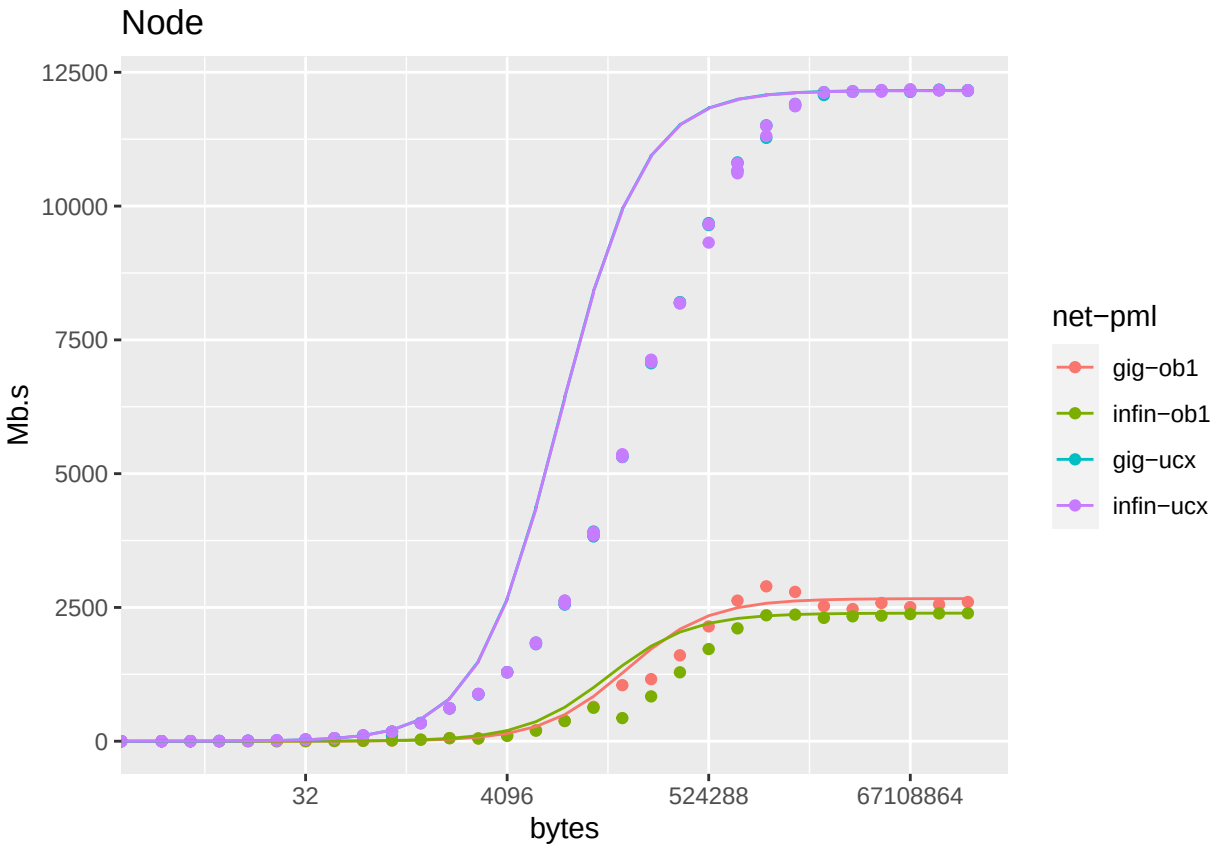
where λ and b are respectively the latency and the bandwidth of the used network.

The graph represents the performed and computed 'Mb per second' of the communication between two different thin nodes and three observations are clear:

- the UCX protocol forces the communication on the Infiniband network;
- there is a plain difference between Infiniband and the Ethernet network if the OB1 protocol is used;
- and finally the given model is not perfectly fitted in any of the protocol-network combination.

The first point is clearly observable by the graph looking at the quite good overlap of the measurement with the UCX protocol. Regarding the second point, it is interesting to observe the low performance of the OB1 protocol with the Infiniband network with respect to both the UCX protocol on the same network and the

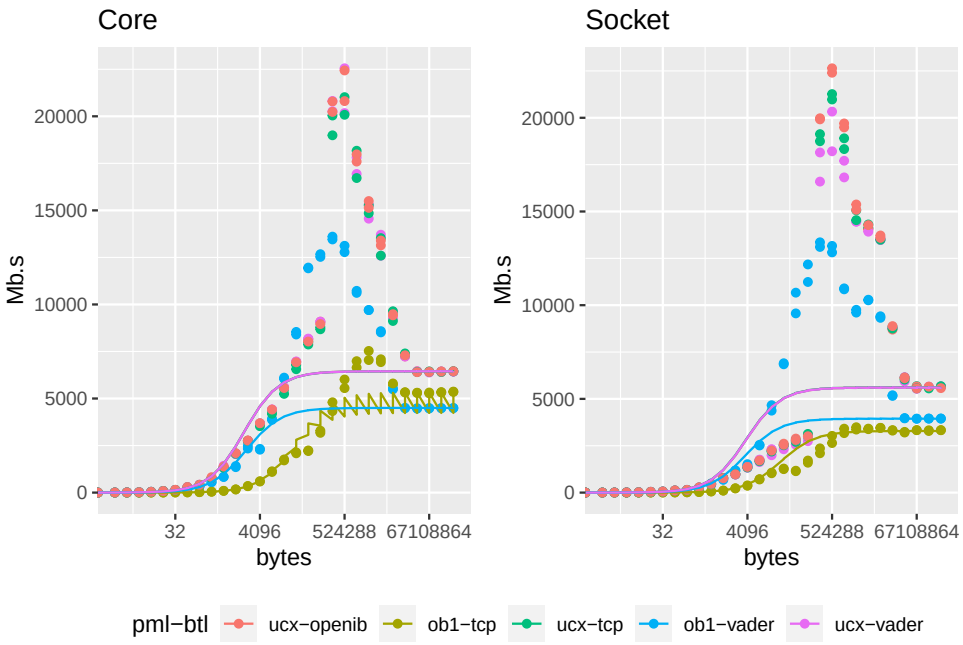
OB1 protocol with the Ethernet network. In the end, even though the model seen during the lessons works well to explain the shape of the performances, it does not fit the results in the central part of the graph with a evident overestimation.

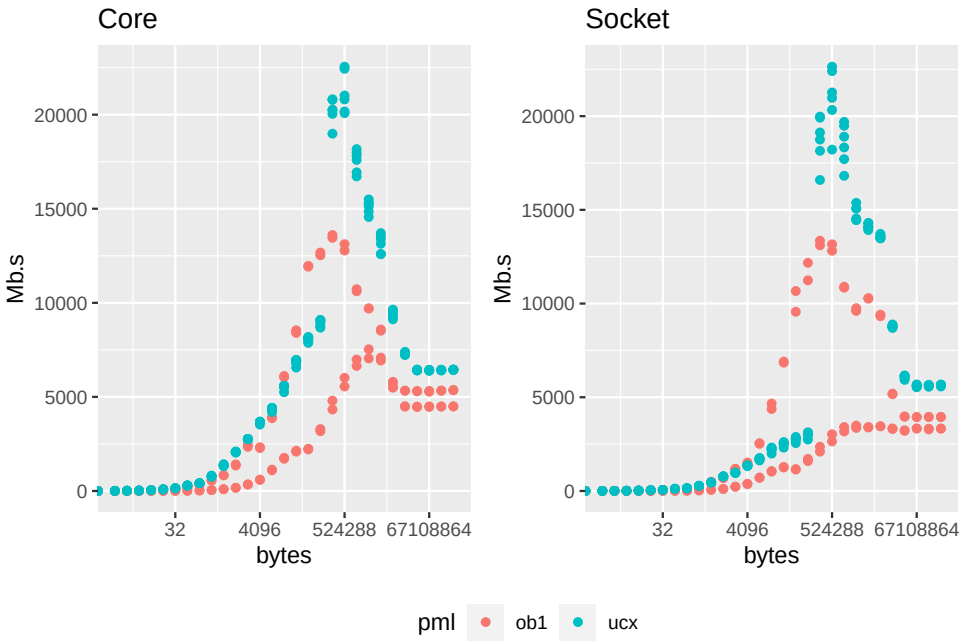


Mapping by core and by socket brings out something not perfectly explainable with the model discussed during the lessons. Indeed, it is clear that the peak in performance at 524288 bytes (= 0.5 Mb) was not expected. The only hypothesis is that for that size of the message the cache misses are perfectly optimized, even though the cache size is not large enough. Nevertheless, there are some curios facts that emerge from the graph:

- the pml-OB1 protocol is the only one that behaves different depending on the btl and on the topology;
- the communication performance within the same socket (processes mapped by **core**) are higher than the one between two sockets (processes mapped by **socket**);
- the pml-UCX protocol is the most performing.

The first fact is noticed based on the first of the two graph that compare the performance between mapping by core and by socket and focusing on the pml-OB1 results. The second is obvious comparing the scales of the two graphs. The last observation is clear by looking the second graph that stress the different results using OB1 and UCX pml-protocols.





Section 3: Jacoby solver

In this section, the aim is to compare the real scalability with the expected one by the model discussed during the lessons. According to the theory the MLUPs/sec are

$$P(L,N) = \frac{L^3 N}{T_s(L) + T_c(L,N)}$$

where L^3N is the size of the problem, $T_s(L)$ is the serial time estimated on a single processor and the communication time is

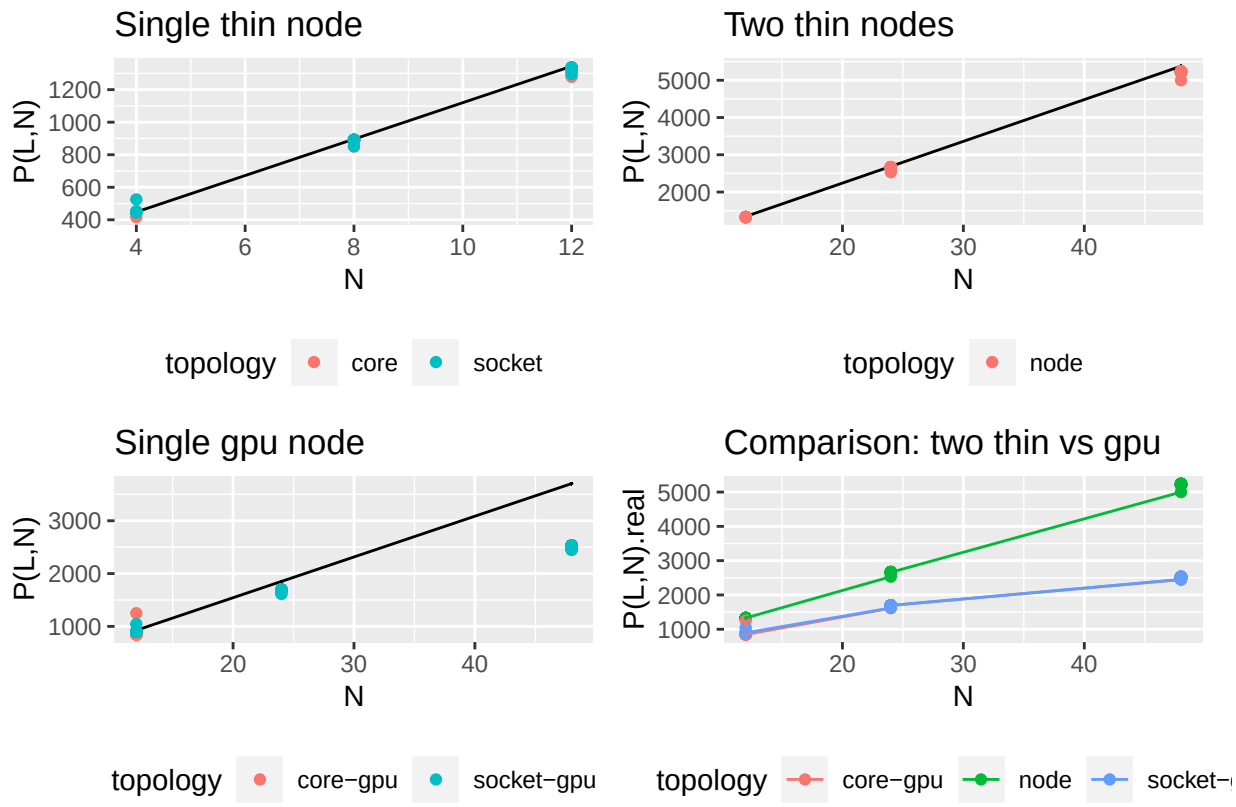
$$T_c(L,N) = \frac{c(L,N)}{B} + kT_l \ ,$$

where B and T_l are respectively the latency and the bandwidth of the network, k is the number of the coordinate directions in which the number of processes is greater than one and finally $c(L,N)$ is the maximum bidirectional data volume transferred over a node's network link. $c(L,N)$ can be derived as

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

For this exercise L was fixed to 600 and the problem was construct such that in each direction the size was a multiple of both L and the number of processes assigned to that coordinate direction. The main consequence is that there are no differences between executions which differ only in the grid of processes. Hence, the grid is not reported in the table: each results is the mean taken over every possible grid for a fixed number of processors.

Before continuing, the following graphs are a useful overview and comparison between the three cases. First of all, the weak scalability is checked both within a node (mapping by core or by socket seems not to make significant difference) and between two nodes. In second place, there is a clear gap between real and theoretical scalability on the gpu node where hyper-threading is enabled. Finallly, the comparison between a gpu node and a couple of thin nodes brings out an important note: 24 cores on a gpu node are less effective than 24 cores on thin nodes, so hyper-threading is not useful at all for this problem. The fact that 48 cores on gpu are less performing than 48 core on thin nodes is an obvious consequence of the previuos observation.



NOTE: The columns $P(L,N)$ reports the MLUPs/sec, the one $NP1/P$ reports the ratio between the performance of a single process times N and the performance on N parallel processes. The columns with *.real* show the collected measurements, while the one without *.real* are showing the values computed by the model.

The following table collects the data of the scalability within a single thin node considering mapping by core and by socket. It is worth to be noticed that there is not a large difference between mapping by core and by node when number of processors and k are fixed. A curios fact is that the performance are higher (both $P(L,N)$ and $(NP1/P).real$ columns) if it was mapped by socket. The perfect scalability is not reached but the model fits the data really well.

Table 1: 4,8 and 12 processes on a single thin node mapped by core and socket

N	k	topology	P(L,N)	P(L,N).real	NP1/P	(NP1/P).real
4	1	core	448.04	446.62	1.0002	1.0034
4	1	socket	448.02	447.02	1.0003	1.0025
4	2	core	447.94	446.45	1.0005	1.0038
4	2	socket	447.90	446.21	1.0005	1.0043
8	1	core	896.08	891.78	1.0002	1.0051
8	1	socket	896.05	891.86	1.0003	1.0050
8	2	core	895.87	891.82	1.0005	1.0050
8	2	socket	895.81	892.45	1.0005	1.0043
8	3	core	895.67	891.08	1.0007	1.0058
8	3	socket	895.57	891.49	1.0008	1.0054
12	1	core	1344.12	1320.48	1.0002	1.0181
12	1	socket	1344.07	1335.72	1.0003	1.0065
12	2	core	1343.81	1320.90	1.0005	1.0178
12	2	socket	1343.71	1332.73	1.0005	1.0088
12	3	core	1343.50	1321.82	1.0007	1.0171
12	3	socket	1343.36	1335.70	1.0008	1.0065

The table below shows the data of the scalability running the program on two different thin nodes (it was mapped by node to obtain comparable results). Even in this case there is not the perfect scalability, however the model fits the collected data. Furthermore. the performance with 12 cores mapped by node, core or socket (look at previous table for the last cases) are more or less equal.

Table 2: 12,24 and 48 processes mapped on two thin nodes

N	k	topology	P(L,N)	P(L,N).real	NP1/P	(NP1/P).real
12	1	node	1344.26	1329.08	1.0001	1.0115
12	2	node	1344.10	1329.94	1.0002	1.0109
12	3	node	1343.93	1327.36	1.0004	1.0129
24	1	node	2688.52	2667.93	1.0001	1.0078
24	2	node	2688.19	2648.26	1.0002	1.0153
24	3	node	2687.86	2646.73	1.0004	1.0159
48	1	node	5377.05	5240.83	1.0001	1.0261
48	2	node	5376.38	5229.33	1.0002	1.0284
48	3	node	5375.72	5227.62	1.0004	1.0287

This last table compares the scalability within a single gpu node mapping by core and by socket. As for the thin node, there is no difference between mapping by core or socket. It has to be observed that in this case the scalability is really poor and the usual model is insufficient to explain the data. Nevertheless, it is clear that the low performance is caused by the hyperthreading that is enabled on the node. It is important to be noticed that both with 12 and 24 cores (before that all the available threads are busy) the gpu node is less performing than a couple of thin nodes (look at $P(L,N)$ column both in the following and previous table).

Table 3: 12,24 and 48 processes on gpu node mapped by core and socket

N	k	topology	P(L,N)	P(L,N).real	NP1/P	(NP1/P).real
12	1	core-gpu	925.86	846.89	1.0002	1.0934
12	1	socket-gpu	925.86	896.40	1.0002	1.0330
12	2	core-gpu	925.72	853.50	1.0003	1.0850
12	2	socket-gpu	925.72	897.53	1.0003	1.0317
12	3	core-gpu	925.57	847.33	1.0005	1.0929
12	3	socket-gpu	925.57	897.07	1.0005	1.0323
24	1	core-gpu	1851.73	1693.02	1.0002	1.0939
24	1	socket-gpu	1851.73	1693.51	1.0002	1.0936
24	2	core-gpu	1851.44	1693.48	1.0003	1.0936
24	2	socket-gpu	1851.44	1686.46	1.0003	1.0982
24	3	core-gpu	1851.14	1692.97	1.0005	1.0939
24	3	socket-gpu	1851.14	1692.64	1.0005	1.0942
48	1	core-gpu	3703.46	2529.00	1.0002	1.4646
48	1	socket-gpu	3703.46	2527.21	1.0002	1.4657
48	2	core-gpu	3702.87	2519.82	1.0003	1.4700
48	2	socket-gpu	3702.87	2533.73	1.0003	1.4619
48	3	core-gpu	3702.28	2520.20	1.0005	1.4697
48	3	socket-gpu	3702.28	2521.13	1.0005	1.4692