Giulia Franco
matricola SM3500370
Year 2018/2019
Exercise 5, High Performance Computing course.

MPI intranode and internode

Modern CPU architecture is quite complex: different socket with several core inside are connected by a high speed network (Infiniband) as Fig. 1 shows. The comunication between these components is generally characterized by the concepts of *latency*, the time it takes in order to open a communication channel, and *bandwidth*, the number of bytes per second that travels through the channel.
In this exercise we will use Intel MPI benchmarks, *PingPong*, in order to explore these two concepts, measuring values for messages passing between processors.

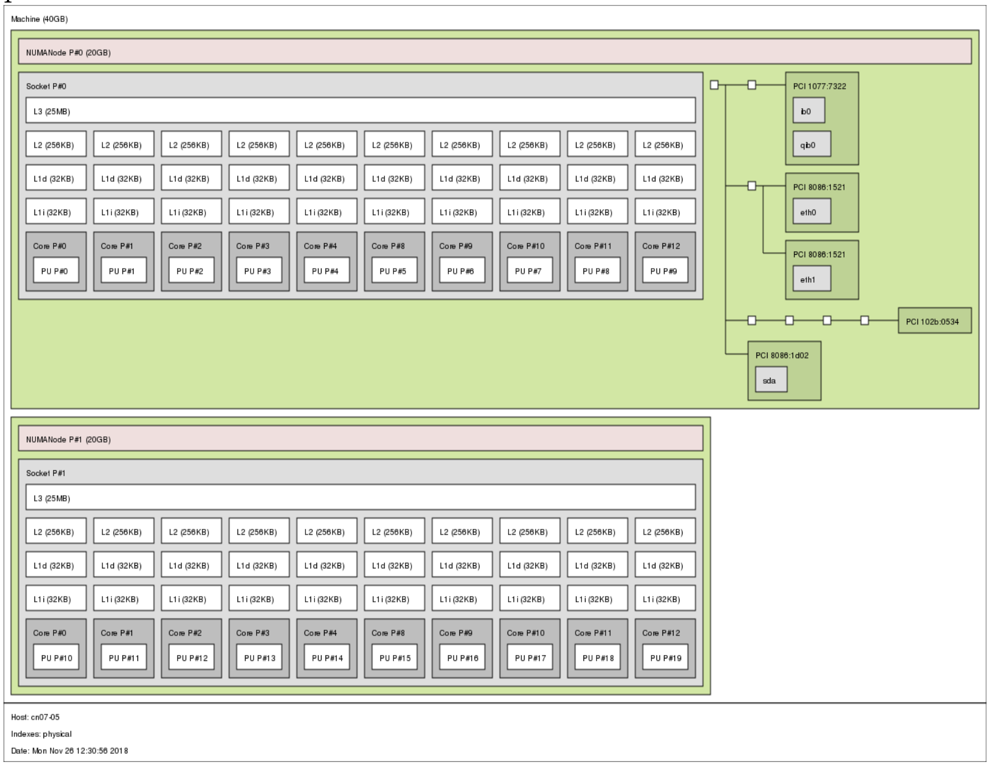

Figura 1: Representation of an Ulysses node.

Latency

Running the benchmark using 10000 iterations, it's possible to notice an initial plateux on small values of sent bytes.

Error bar for Latency

Bytes	**Run 1 (μ s)**	**Run 2 (μ s)**	**Run 3 (μ s)**
0	**0.60**	**0.60**	**0.48**
1	**0.64**	**0.64**	**0.52**
2	**0.63**	**0.64**	**0.52**
4	**0.64**	**0.65**	**0.52**
8	**0.64**	**0.64**	**0.52**

Communication Performance

We can now use the benchmark in order to measure the bandwidth between different cores in the same socket and on different sockets. From the Fig.1 it's clear that the cores are numbered from 0 to 9 for the first socket (marked 0) and from 10 to 19 for the second socket (marked 1) in the node. The results are represented in the graphs below.

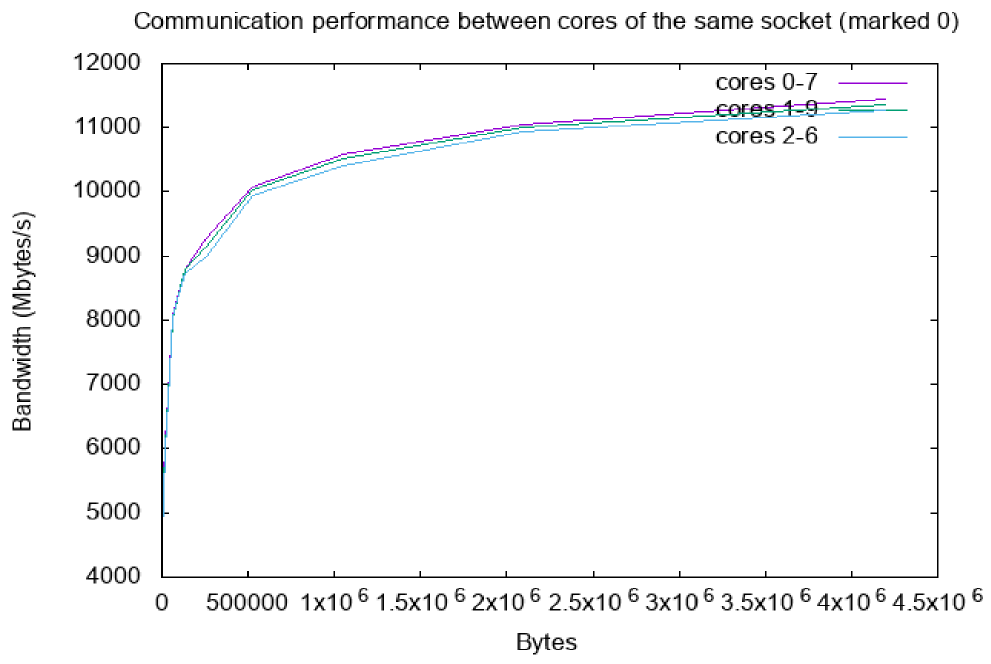

Figura 2: Bandwidth measured between cores on the first socket.

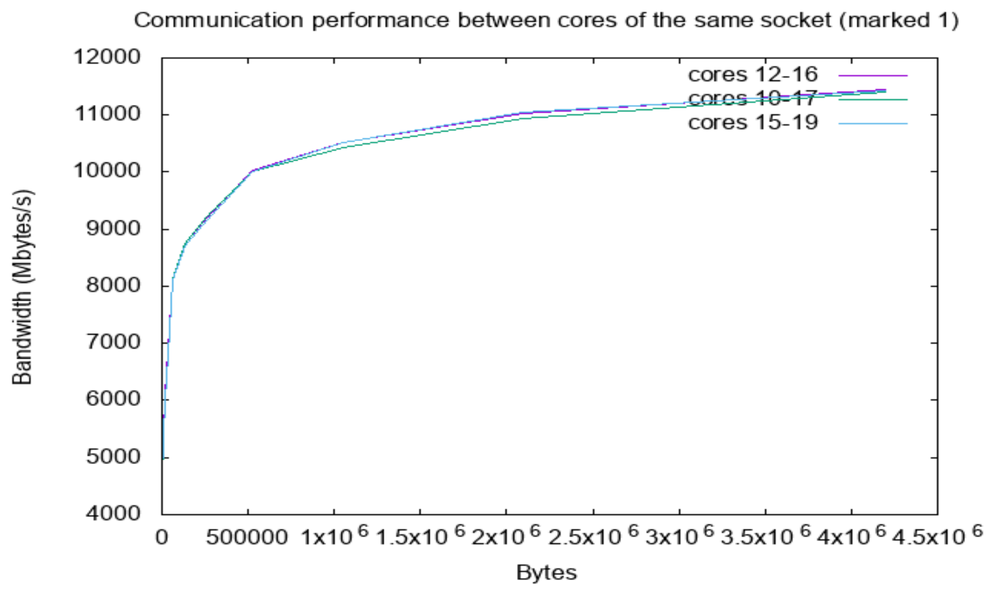

Figura 3: Bandwidth measured between cores on the second socket.

So it's possible to conclude that generally we can reach a bandwidth of 11400 Mbytes/s in 350 μs for cores in the same socket.
Moreover we can compare communication performance between cores in different sockets.

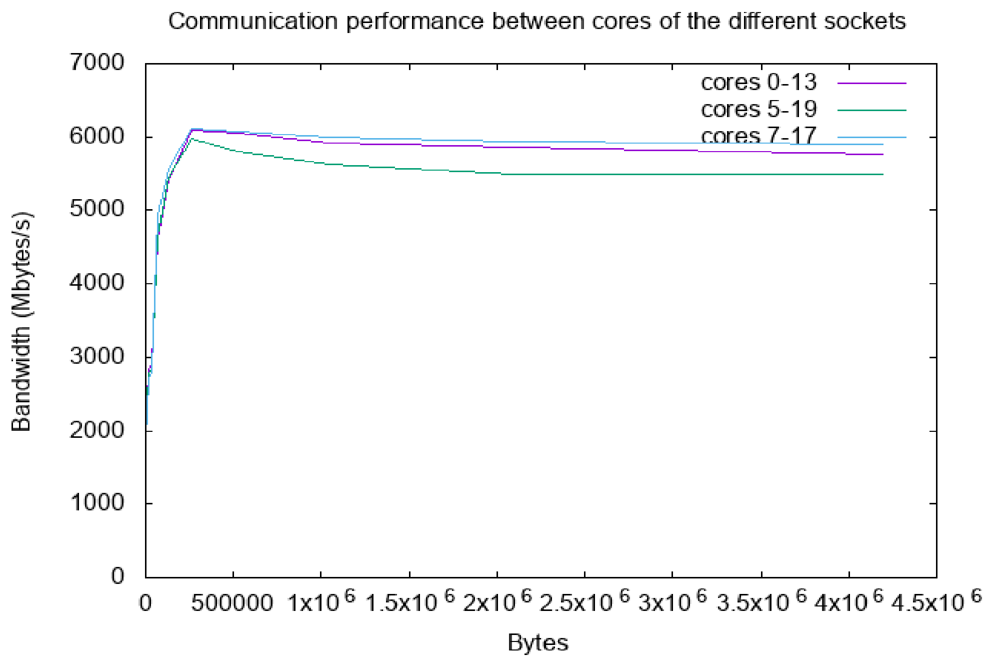

Figura 4: Bandwidth measured between cores on different sockets.

As expected, bandwidth values between cores on different sockets are smaller than the ones computed considering intranodes cores. As well the communication time is bigger considering two different sockets than a single one.
Finally it's possible to use the benchmark in order to compute bandwidth between cores on different nodes. Since the path between nodes is biggest considered in this experiment, it's reasonable to expect the lowest bandwidth and the highest amount of time needed. The results are reportent in Fig.5.

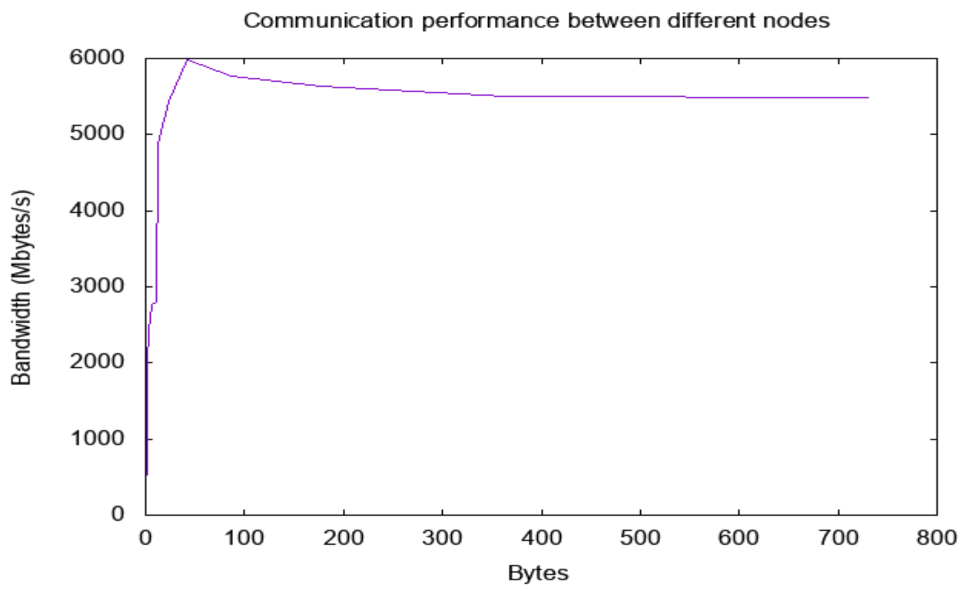

Figura 5: Bandwidth measured between cores on different nodes.

Comparison Bandwidth values for 4194304 Bytes

Mode	time (μs)	Bandwidth (GBytes/s)
Intranode (Cores 7-17)	**677.45**	**5904.46**
Intranode (Cores 0-13)	**694.60**	**5758.74**
Intranode (Cores 2-19)	**728.61**	**5489.93**
Internode	**729.95**	**5479.80**

From the result obtained, since the values of latency and bandiwth are actually close to the ones reported for intranode computation, we can conclude that maybe there're some issues on the node reservation for mpi to run.

Stream

Stream is a benchmark that measure the sustainable memory bandwidth. We perform the measuraments in two different ways: first binding the process to socket 0 and NUMA memory 0 (*close memory* values), secondly binding it to socket 0 and NUMA memory 1 (*distant memory* values).

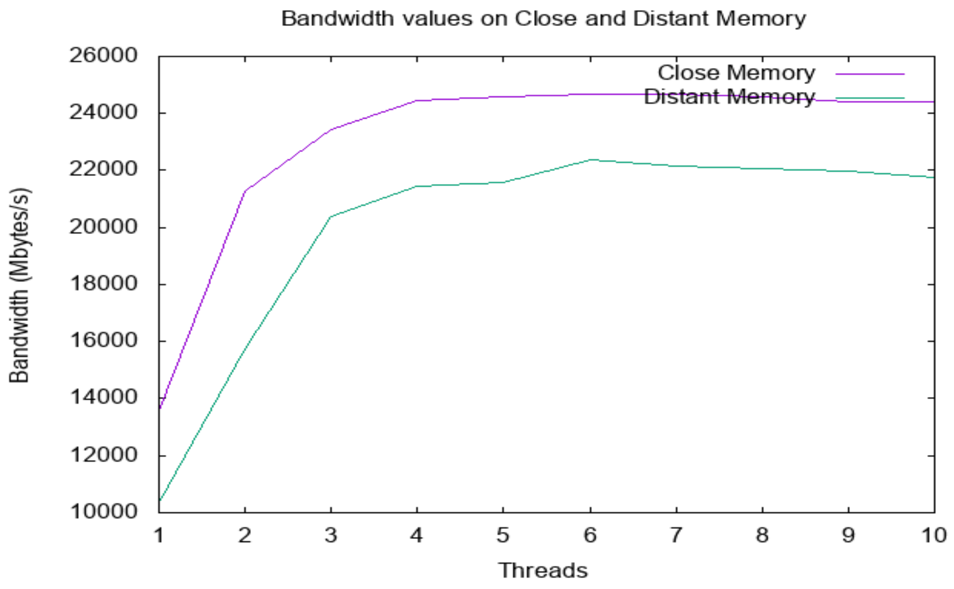

Figura 6: Bandwidth measured on close and distant memory.

As Fig.6 shows, the avarage bandwidth values on close memory is higher than the ones on distant memory. This is expected since the message path is smaller in the formal case.

Nodeperf

Nodeperf program computes matrix-matrix multiplication in order to measure the peak performance of an Ulysses' node. We compute this quantity using two different compilation, the Intel and gnu one, specifying the dimension of the matrix (-lda). The results are reported in the following table.

Comparison Peak Performance values for different compilations

theoretical PP (GFlops)	**Intel PP (GFlops)**	**Gnu PP(GFlops)**
448	**440**	**26.35**

It's clear that the closest performance is the one obtained using the Intel compiler, probably because of the frequent use of data that has been recently referred (findable in caches or even in the register).