Report Verardo Thomas

Verardo Thomas

SECTION 2

In this section, the main purpose was to estimate the latency and the bandwidth of all available combinations of topologies and networks on ORFEO computational nodes. for finding the latency and the bandwidth with the OpenMPI library, I writed a bash script that first submitted a job for running the code in the CPU or in the GPU, and than runned it for 10 times for each topologies. In this way, I can take the maximum of the result of each topologies, so to have better data to study.

Inserire MPI_barrier()

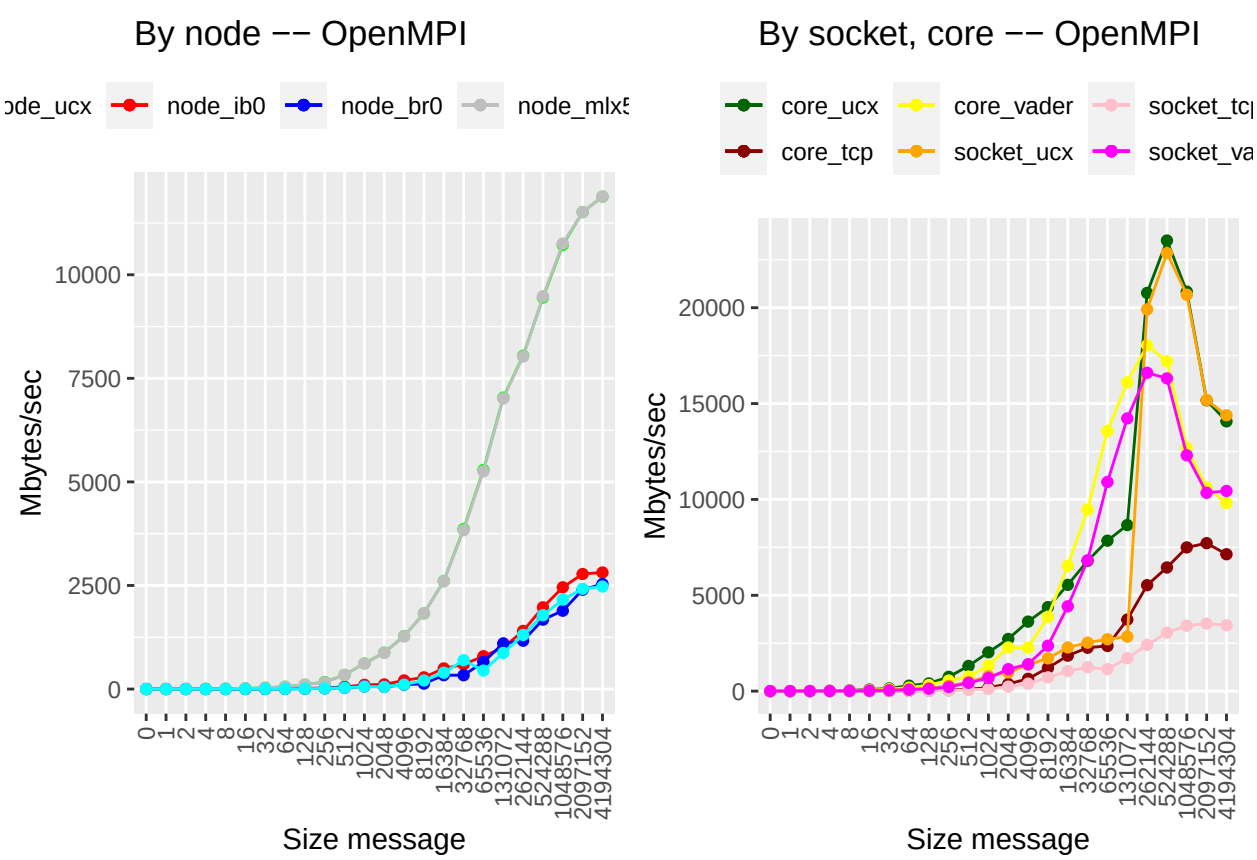
OpenMPI

First of all, I did a benchmark with OpenMPI of some topologies that I found. The topologies tested in PingPong for across 2 nodes that I found were these:

- *node_ucx*: Use the UCX communication library over InfiniBand using the default configuration
- *node_ib0*: Use the InfiniBand adabter over the TCP/IP protocol
- *node_br0*: Use UCX with Ethernet (and so also the protocol TCP/IP)
- *node_mlx5_0_1*: Use UCX but with the configuration 0 of the interface number (InfiniBand, that is the default one)
- *node_tcp*: Use ob1 communication library, that is a multi-device and multi-rail engine

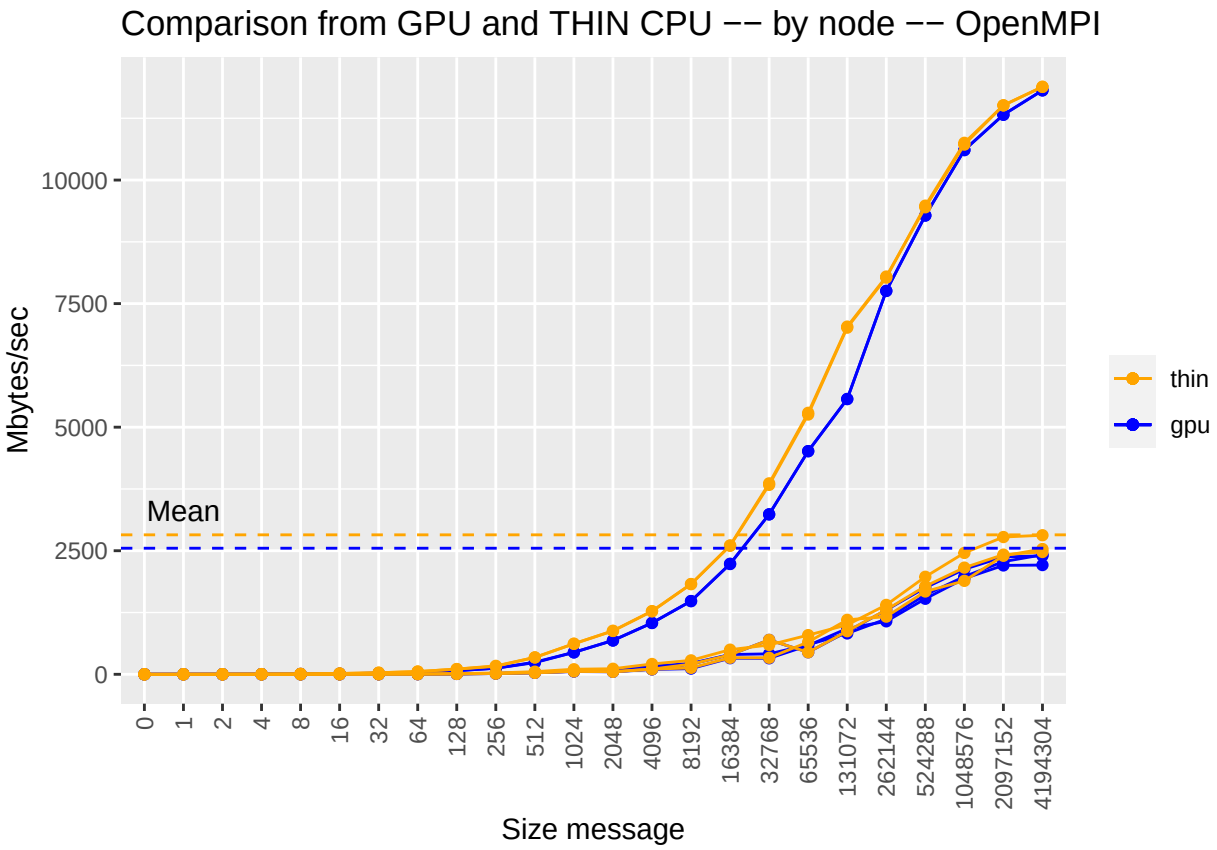
In the first graph we can see the difference between the communication protocol and their interfaces. This plots are made with Thin nodes using the OpenMPI library. In particolar, *node_ucx* and *node_mlx5_0*, also with different package size, have equal bandwidth; this because *node_mlx5_0* describes the default configuration, that is *node_ucx*. In the graph, with message size very high, you can see the difference between the default configuration and the new configuration with *ib0*, *br0* and the last one that uses *ob1* as point-to-point messaging layer and *TCP* as byte transfer layer.
Then I did the same benchmark across 2 socket and across 2 nodes. The results are in the second graph. In both, I first used the default configuration and then I tried to use a different PML. In fact, I used *ob1* as PML and first I used *TCP* as BTL and then I used *Vader* (that used shared memory). In this graph you can see that send messages across 2 core is faster than sending messages across 2 socket. This because cores are within the same socket and are physically closer.

The main difference between the chart of the nodes and the chart of the cores and sockets is that in the first one, the line continue to rise without never discending and so the bandwith is always getting bigger as the size memory. In my opinion, in the core and socket chart, there is this descending curve because the L2 cache fills up if the message size is grater than 1024 KB (1048576 B) and the message goes in the L3 cache, that can save up to 14080 KB.



Intead, in the nodes chart, I think that is different from the sockets and cores chart because Infiniband provides native support for RDAM (Remote Direct Memory Access). In fact, integral to RDMA is the concept of zero-copy networking, which makes it possible to read data directly from the main memory of one computer and write that data directly to the main memory of another computer. RDMA data transfers bypass the kernel networking stack in both computers, so it doesn't have to pack and unpack the message, improving network performance. As a result, the conversation between the two systems will complete much quicker than comparable non-RDMA networked systems.

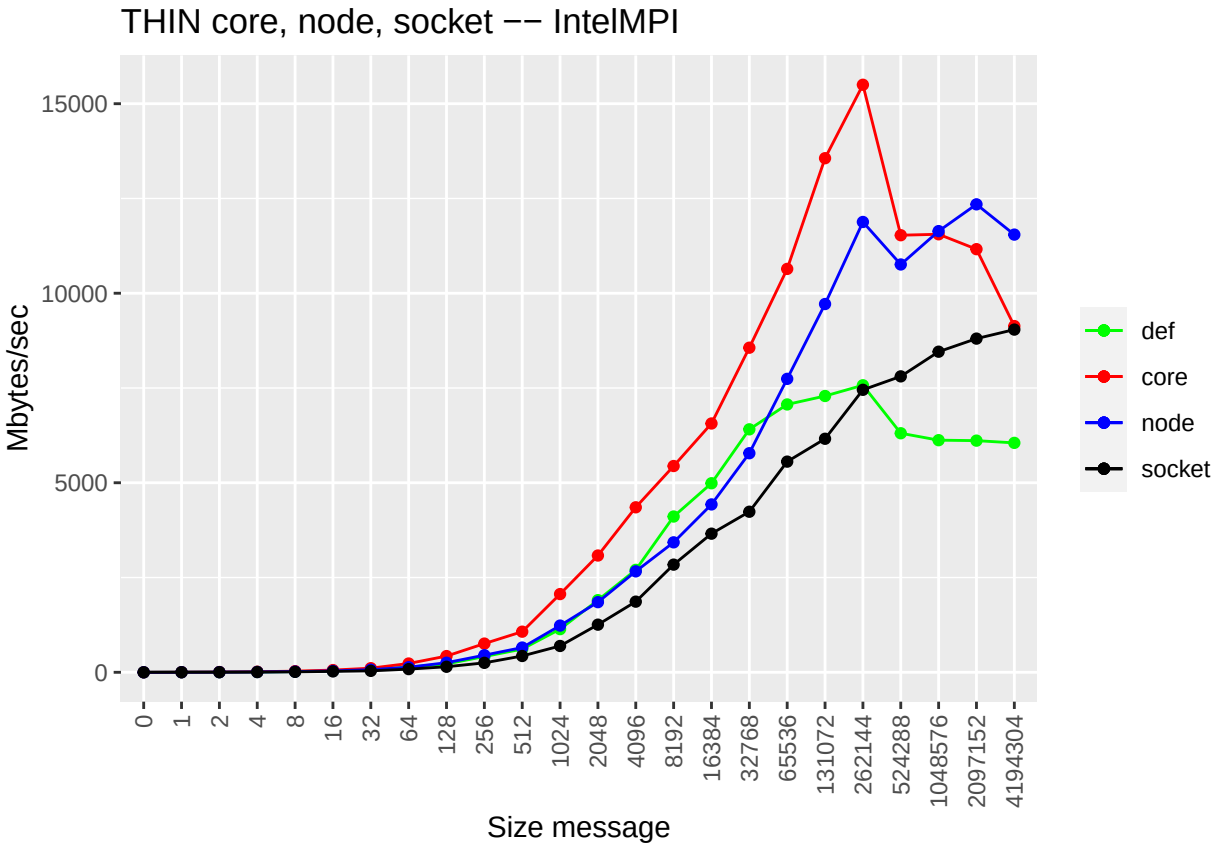
Then, I did the charts also in the GPU nodes (see the github repository).



Here there is the comparison between the benchmark in the Thin CPU nodes or in the GPU nodes. You can see from this chart and from the mean, that the Thin node is faster then the GPU node.

IntelMPI

After doing all charts of the PingPong benchmark with OpenMPI for the topologies that I mentioned before across two nodes, two sockets and two core in the Thin CPU nodes and in GPU nodes of ORFEO, I taken new data with the IntelMPI library. Therefore, I ran the PingPong code first with the defaul configuration, and than across two nodes, across 2 sockets and in the end across two core.



Also with the IntelMPI library, it's easy to see that the line that represent the PingPong across 2 core is the only line that at one moment no longer grows, without considering the default line.

Model fitting

Finally, I compered the resulting estimated latency and bandwidth parameters against the one provided by a least-square fitting model. I used the model provied in class that described the total transfer time of a message:

$$t_{comm} = \lambda + \frac{(Size\ of\ message)}{b_{network}}$$

To fit the model, first I divided my data in two parts and I calculated the linear model for the first half and another linear model for the second half. Then I take the slope from the first model, that is similar to the latency and then I take the angular coefficient from the second model, that is similar to the $1/bandwidth$. After that, I used the formula above to estimate my latency and that I divided the size message with this latency estimation.