High Performance Computing - Assignment 1

Estimating latency of OpenMPI algorithms with core allocations

Gabriele Pintus
gabrielegavino.pintus@studenti.units.it

ABSTRACT

In this report we estimate the latency of the default OpenMPI implementation of two collective blocking operations: broadcast and scatter. We vary the number of processes, the message size, the core allocation strategy and the distribution algorithm.

KEYWORDS

HPC, OpenMPI, Latency, Core Allocation, Collective Operations

1 INTRODUCTION

OpenMPI is a widely used library for message passing in parallel and distributed computing. It provides a set of collective operations that allow processes to communicate with each other in a coordinated way. In this report we focus on two of these operations: broadcast and scatter. Both operations are blocking, meaning that the root process will wait until all the other processes have received the message before continuing.

1.1 Distribution algorithms

OpenMPI provides several algorithms for distributing the message among the processes. The default algorithm is chosen based on the message size and the number of processes. Herafter we list the three algorithms we used in our experiments:

(1) basic linear
(2) chain
(3) binomial tree
(4) binary tree

1.1.1 Basic linear. The basic linear algorithm is the simplest of the four. It sends the message in a linear way: the root process starts by sending the message to the first process, then it sends the message to the second process, and so on, until all the processes have received the message. The total amount of data transmitted is $(n-1)\cdot m$, where n is the number of processes and m is the message size. With this algorithm, only one process is in charge of sending the message to all the other processes.

1.1.2 Chain. The chain algorithm is similar to the basic linear algorithm, but instead of sending the message from the first process to the last process, it sends the message from the root process to the next process, and then from that process to the next one, and so on. This way every node is sending data to another node in a given instant of time. This implicate that the root process can start working on its own data right after sending the first message to the first process.
In the case of the scatter operation, we will see later, The size of the message decrease at each step, since every process is taking from itself a part of the message and sending only the remaining part to the next one. The total number of messages is the same, however, because of the decreasing in size of the message, the distribution becomes faster at each step.

1.1.3 Binomial tree. With the binomial tree algorithm, each process communicates only with a subset of the other processes. In total, the root process will send the message to $\log_2(n)$ processes, where n is the number of processes.

$t_0: P_0 \to P_1,$
$t_1: P_0 \to P_1, P_2 \to P_3,$
$t_2: P_0 \to P_4, P_2 \to P_5, P_3 \to P_6,$
$t_3: \dots$

The total amount of data transmitted is $(n-1)\cdot m$, where m is the message size.
In the case of the scatter operation, the number of steps remain the same, but the amount of data transmitted is reduced at each step. Assume P_0 holds an array divided into 8 chunks $[c_0, \dots, c_7]$ and $P_1, \dots, p_7$ are the other processes.

$t_0: P_0 \xrightarrow{c_4,\dots,c_7} P_1,$
$t_1: P_0 \xrightarrow{c_2,c_3} P_2, P_1 \xrightarrow{c_6,c_7} P_3,$
$t_2: P_0 \xrightarrow{c_1} P_4, P_1 \xrightarrow{c_5} P_6, P_2 \xrightarrow{c_4} P_5, P_3 \xrightarrow{c_7} P_7,$

1.1.4 Binary tree. The total data to be sent is divided in multiple pieces, one for each node. The root process sends the first two chunks to two different nodes and these two to their children, and so on, until all the nodes have received the data.This way the total number of messages sent is $\log_2(n)$, where n is the number of processes. The total amount of data transmitted is $log_2(n)\cdot n\cdot m/2$, where m is the message size. The drawback of this algorithm is that clearly there is a redudndancy in the data transferred due to the overlapping paths of communication that are inherent to the binary tree structure. This imply there could be issues with the network bandwidth, especially when the number of processes is high.
A possible variation consists in changing the number of children each node has, hence changing the base of the logarithm, thus reducing the contribution of the logarithm in the formula.
In the case of the broadcast operation, we will see later, every process needs the same data, therefore the redundancy is minimized.

2 OPERATIONS

In this section we perform three benchmarks, one for the pure latency, one for the broadcast operation and one for the scatter operation. Since the last two operations build on top of the first one, this will act as a baseline for the other two.

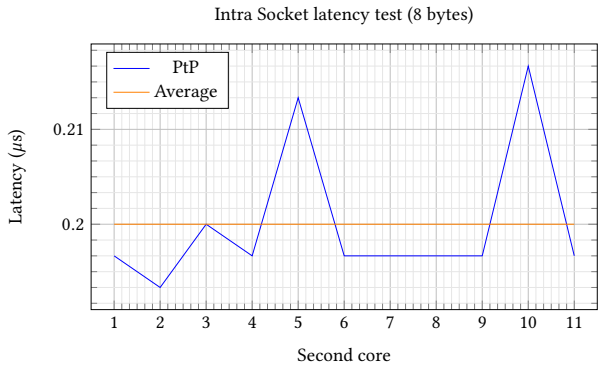
2.1 Latency

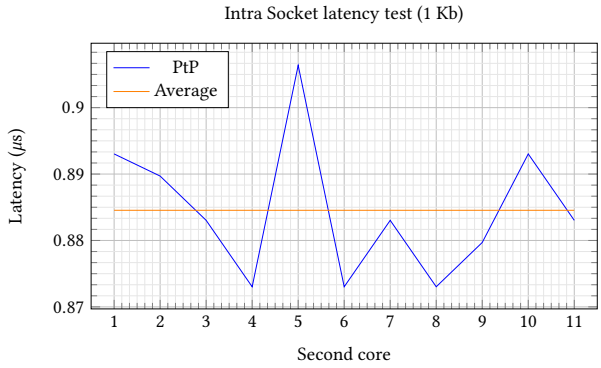
In this test we measure the point-to-point latency between two MPI processes. By strategically selecting the allocation of these two, we can get a deeper insight on the latencies of our architecture. This approach is valuable also for understanding the impact of different

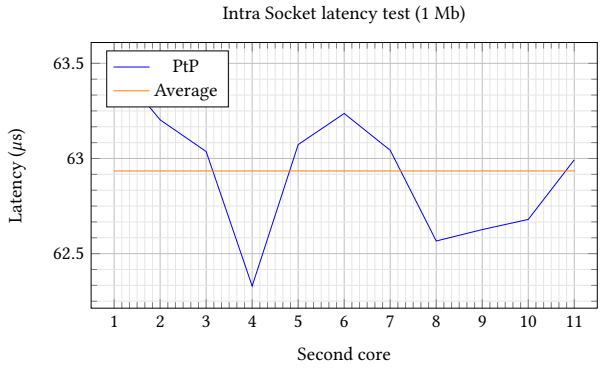
topologies across each layer of the system. For each test we used the following parameters:

- iterations: 100
- warmup: 10
- message size: 1 byte to 1 Mb

2.1.1 Intra-Socket. In this test we bind the two processes to the same socket fixing the first to core 0 and varying the second from core 0 to core 11. Additionally we consider three different message sizes: 8 bytes, 1 Kb, and 1 Mb. This way we obtain twelve measurements for each message size and we can analyze the impact of the message size on the latency.



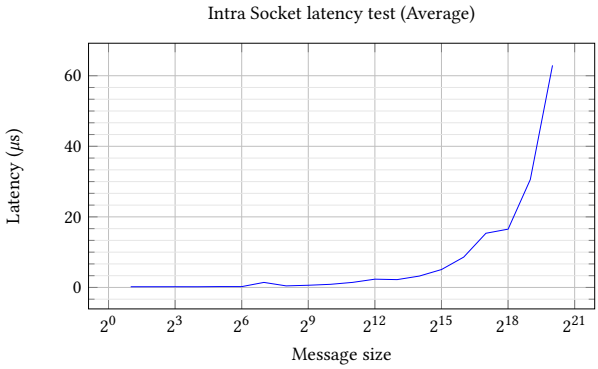




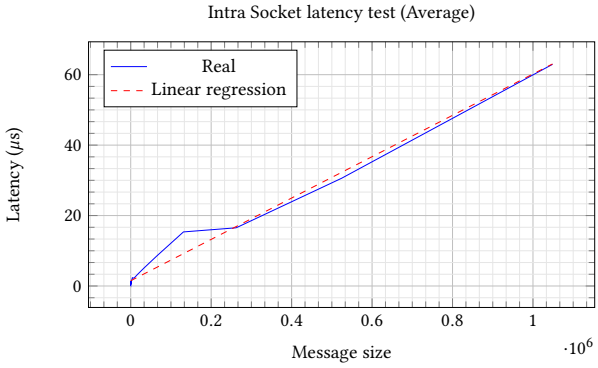
The communication is the slowest when the second process is bound to the same core as the first process. We do not represet this value in the plot for readability reasons. Hereafter we report some further statistics:

Message size	Average (μs)	Std (μs)	Std/Average
8 bytes	0.19	0.007	0.0362
1 Kb	0.88	0.009	0.0111
1 Mb	62.93	0.33	0.005

As we can see, the absolute variability of the latency increases with the message size. However, if we consider the relative variability, we can see that the latency is more stable for larger messages. Finally, we analyze all the tested message sizes averaging the latencies among each core combination.



Even with the significant increase in the latency for larger messages, its growth is not exponential. This is clearer when we consider the linear scale of the plot.

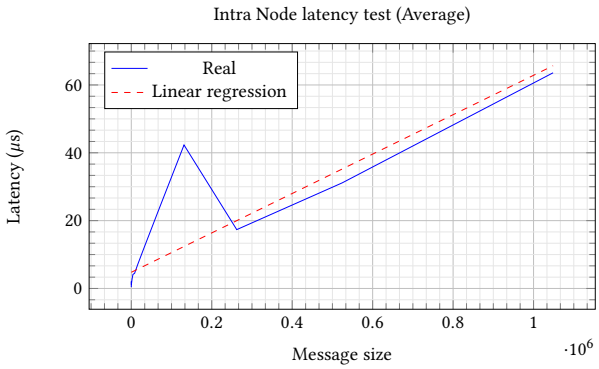


The data clearly exhibits a linear behaviour, with a slope of approximately 58.7μs/MB, starting from the intercept with value 1.49μs. This means that the latency grows linearly This simple yet effective model enable us to estimate the latency for larger messages, assuming the linearity of the system will still hold. The only deviation is for the smallest message size, where the latency is more than linear. This could be due to the overhead of the communication system, which is more significant for smaller messages.

2.1.2 Intra-Node. In this test we bind the two processes to different sockets. We fix the first process to socket 0 core 0 and the second process to socket 1 core 0. We vary the second process from core 0 to core 3, 6, and 10. We again expect the latency to be stable across the different cores and this is the reason we tested only four of them. Since we have less test points we directly report a table.

Message size	Average (μs)	Std (μs)	Std/Average
8 bytes	0.405	0.005	0.0123
1 Kb	1.9175	0.0217	0.0113
1 Mb	63.6575	0.8918	0.0140

This time we observe a different behaviour in the variability-to-average ratio. Instead of decreasing as in the previous test, it stays almost constant around an average of 0.0125. Having seen the previous result, we ask ourselves if again the latency grows linearly with the message size.



This time, even if starting from a higher intercept of value 4.76μs, the slope is approximately the same as the previous test, around 58.7μs/MB. This means that not only the latency grows linearly with the message size, but also the rate of growth is the same as in the previous test. This is a very interesting result, since it means that the latency is not only stable across different cores, but also across different sockets. However, the non-linear behaviour for the smallest message size is now more evident.

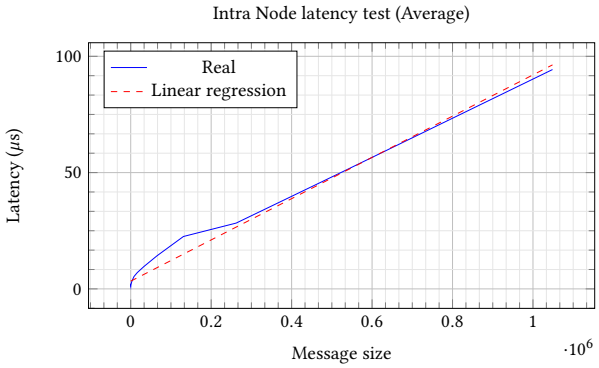
2.1.3 Intra-Cluster. Finally we test the latency between two processes on different nodes. We test the following combinations:

- N0 S0 C0 ↔ N1 S0 C0
- N0 S0 C0 ↔ N1 S1 C0
- N0 S1 C0 ↔ N1 S0 C0
- N0 S1 C0 ↔ N1 S1 C0

Again we report the results in a table.

Message size	Average (μs)	Std (μs)	Std/Average
8 bytes	1.1250	0.0622	0.0553
1 Kb	1.9550	0.1484	0.0759
1 Mb	94.2250	0.3829	0.0041

This time we observe again a different behaviour in the variability-to-average ratio. It increases when considering low values of the message size and then drops for larger values becoming more and more stable.
Once more we check for the linearity of the latency with the message size.



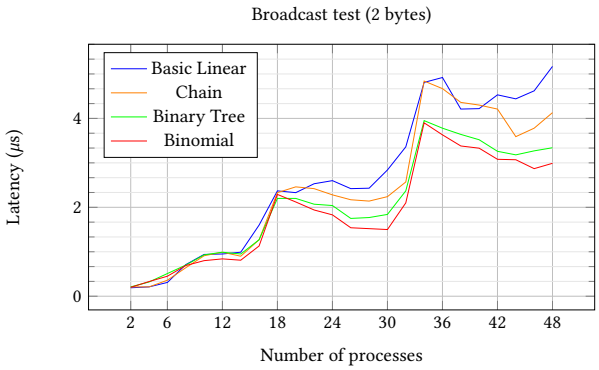
Suprisingly, the intercept of this last model is lower than the previous one. Obviously computing the latency for an empty message is not realistic, therefore this value should be taken with caution. However, the slope is 88.69μs/MB, which is higher than the previous tests. The initial deviation is suppressed by the higher slope making it less noticeable.

2.2 Broadcast

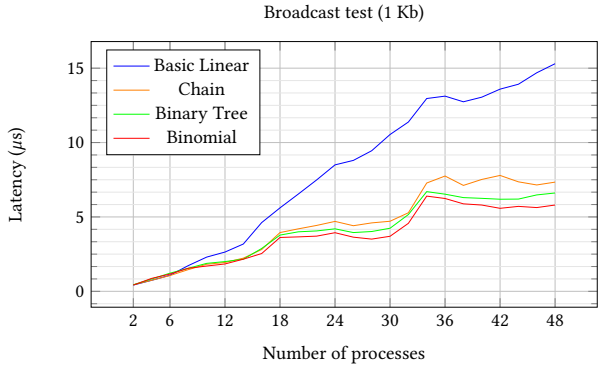
The broadcast operation is used to send a message from the root process to all the other processes in the communicator. This is useful when the root process has a block of data that needs to be entirely copied to all the other processes.
The parameters for the tests are:

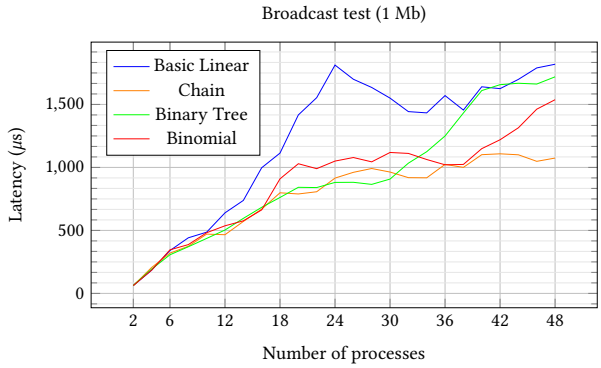
- iterations: 10000
- warmup: 1000
- algorithms: basic linear, chain, binary tree, binomial tree
- message size: 1byte → 1 Mb
- number of processes: 2, 4, . . . , 48

Herafter we report the results for this operation considering three different message sizes: 2 bytes, 1 Kb, and 1 Mb. We expect to notice a significant increase in the latency when the MPI processes fill the socket and even more when they fill the node. This is due to the fact that the communication is done through the network and not through the shared memory.



When processes are within the same socket, they can share cache more efficiently, leading to lower latency. However, once the first socket is fully utilized and processes start to run on the second socket, inter-socket communication requires maintaining cache coherence across sockets, which introduces additional latency. This causes the rapid increase in latency we can observe in the interval from 12 to 18 processes (remember that we have 12 cores per socket). After that, the beneficial effect of the cache helps reducing the latency until the second socket is fully utilized. Then the second node is used and the latency drastically increases again. This require network communication, which is even slower. Once more, the cache of the second node helps reducing the latency.
The best overall performing algorithm is the binomial, which consistently outperforms the other algorithms. Since the message size is very small, we cannot observe a wide difference between all the algorithms. However, this will become more evident in the two following tests.





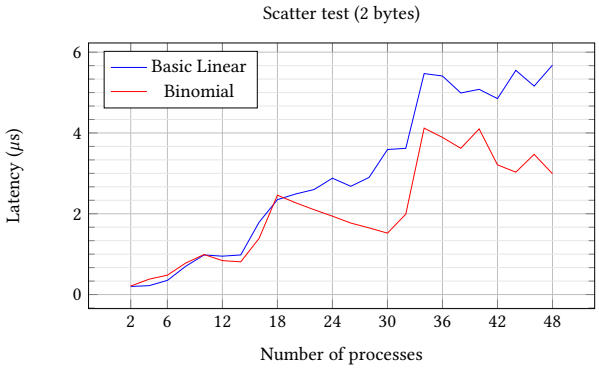
As expected, when increasing the message size the difference in latency between the 'optimized' algorithms and the basic linear algorithm becomes more evident. This effect can be clearly assessed from the two following plots. The increase in latency caused by the network communication is more pronounced for larger messages. Notice the constant linear trend of the basic linear algorithm with 1 Mb messages. This is due to the fact that the basic linear algorithm sends the message from the root process to the last process in a linear way. Consequently the last process will receive the message only after all the other processes have received it. This is the reason why the latency increases linearly with the number of processes. After the first node being fully utilized, the second node starts to be used and the latency decreases a bit to then start increasing again, this time less rapidly.

2.3 Scatter

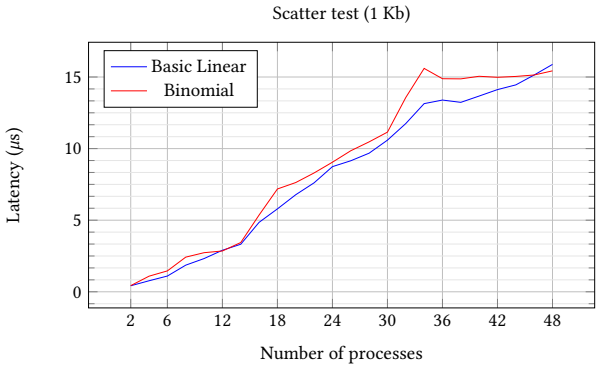
The scatter operation is similar to broadcast, but instead of sending the entire message to all the other processes, the root process sends a different chunks of the message to each process. This is useful when the root process has a block of data that needs to be divided among all the other processes. The parameters for the tests are:

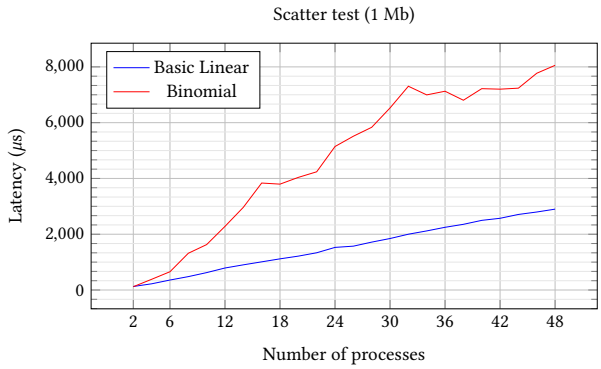
- iterations: 1000
- warmup: 100
- algorithms: basic linear, binomial
- message size: 1byte $\longrightarrow$ 1 Mb
- number of processes: $2, 4, \ldots, 48$

Like in the previous case we report the results considering the same three messages sizes. Again we expect to notice a significant increase in the latency when the MPI processes fill the socket and even more when they fill the node.



For a small message size of 2 bytes, the binomial tree algorithm outperforms the basic linear one.





3 CONCLUSIONS