# High Performance Computing 2023 - Exercise 1

**Marco Zampar - SM3800032**

October 01, 2024

# Problem Statement

This project aims to:

- Estimate the latency of the default OpenMPI collective operations Broadcast and Barrier.
- Vary the number of processes and message sizes for different allocations of the processes.
- Compare the results with other algorithms.
- Infer the underlying performance models of the algorithms.

# Computational Resources

- 2 THIN nodes from the ORFEO cluster.
- Each node has 2 CPUs with 12 cores in a single NUMA region.

# Broadcast Algorithms analysed

## Flat Tree Algorithm

- Single-level tree topology.
- Root node transmits to $P - 1$ child nodes, without segmentation.

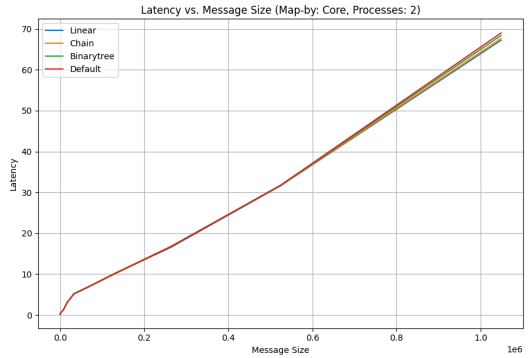
## Chain Tree Algorithm

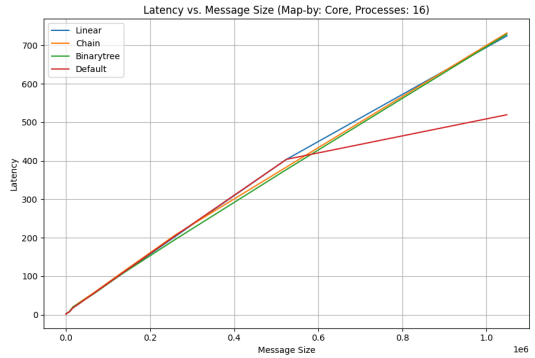
- Internal nodes have one child.
- Messages are split into segments, transmitted in a pipeline.

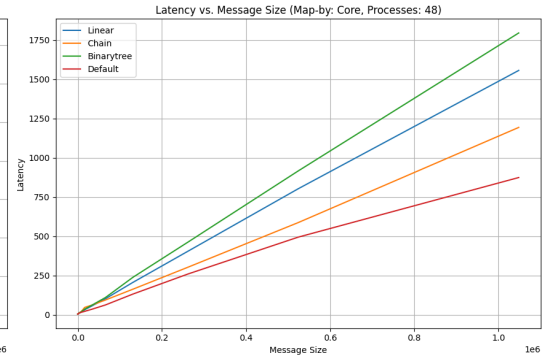
## Binary Tree Algorithm

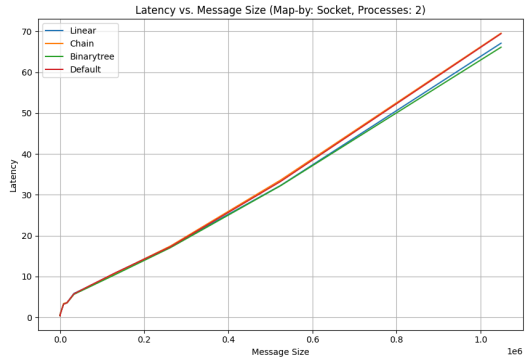
- Internal process has two children, then $P = 2^H - 1$, assuming the tree is complete, with $H = \log_2(P + 1)$ the height of the tree.
- Segmentation is used to improve communication parallelism.

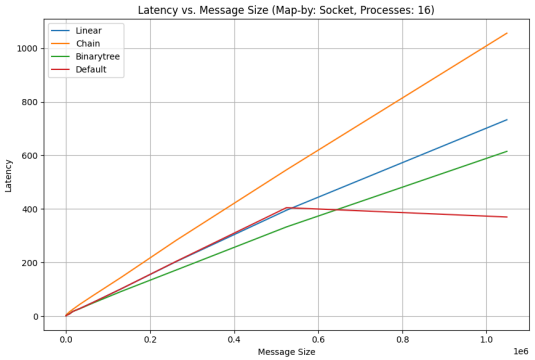
# Latency vs Message Size (Fixed Processes)

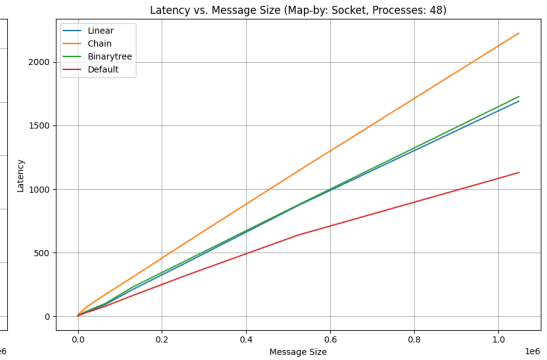


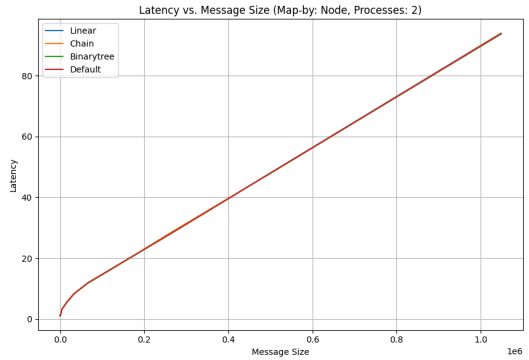



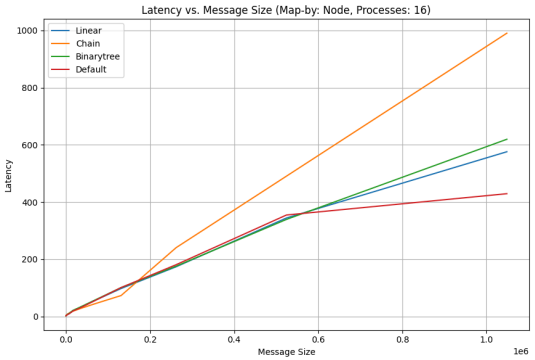



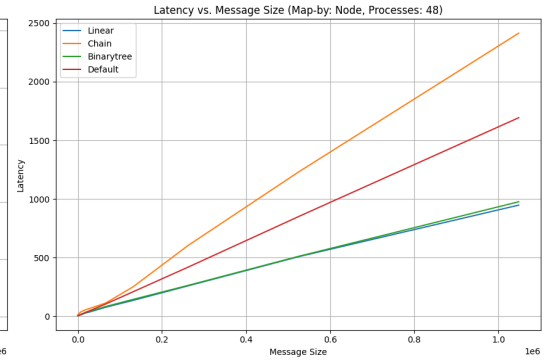



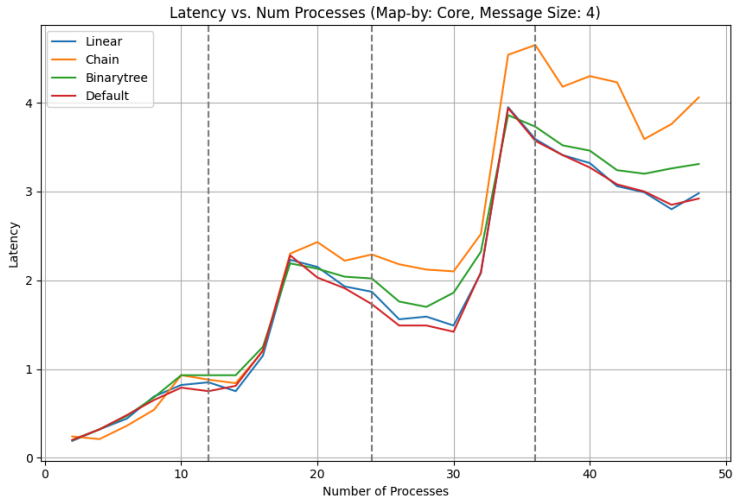



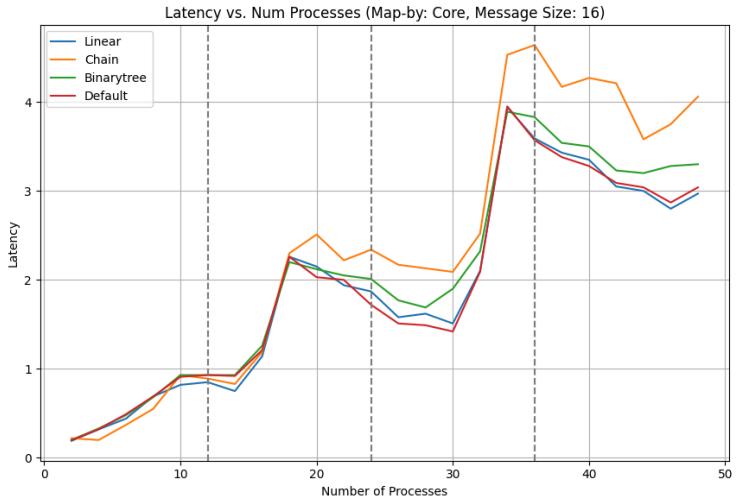



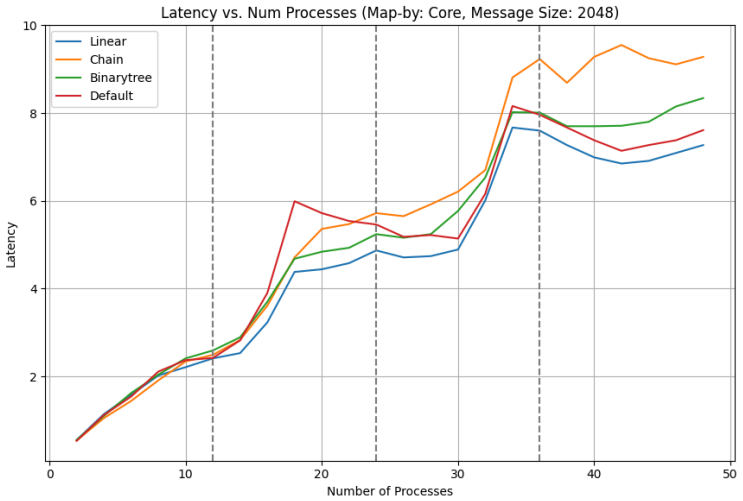



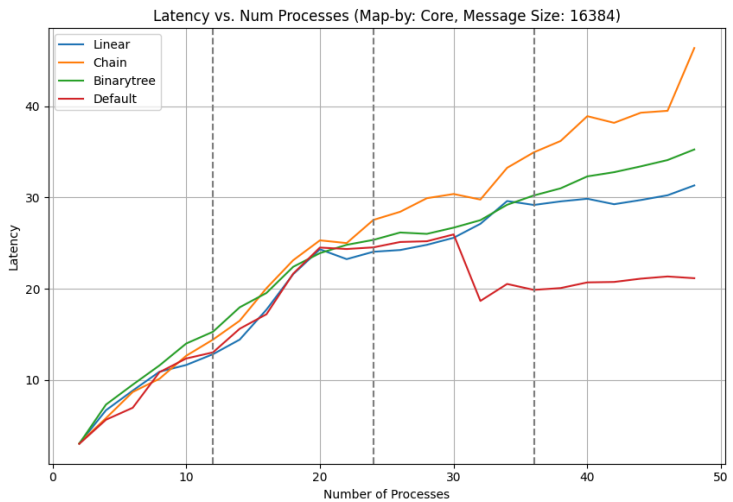



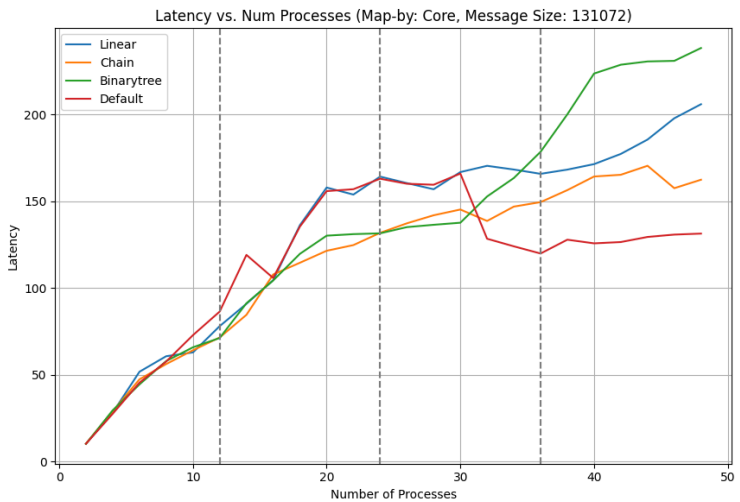
# Latency vs Number of Processes (Fixed Message Size and Core allocation)

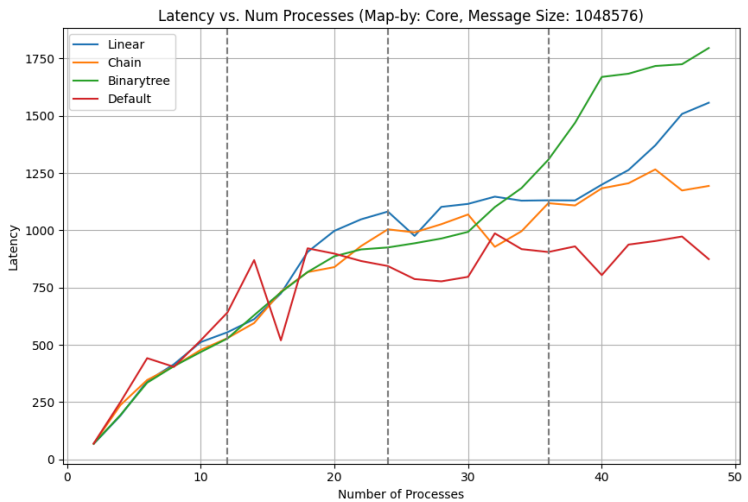


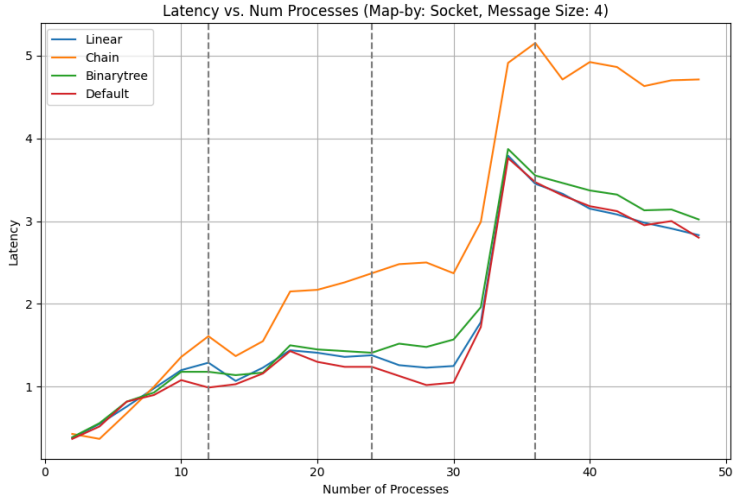



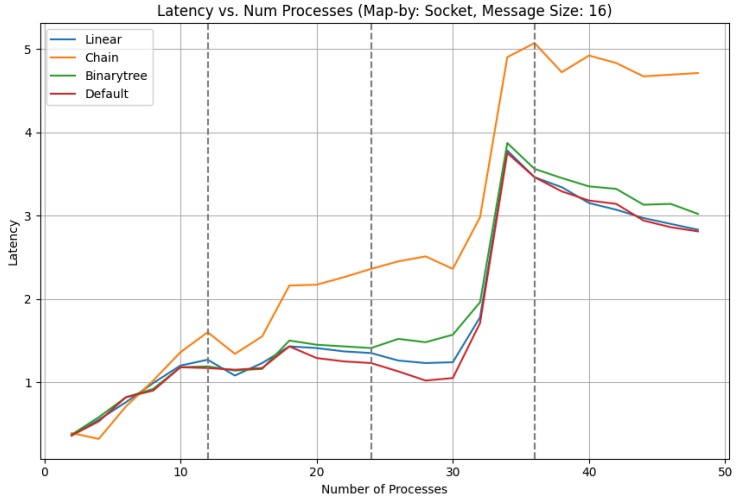



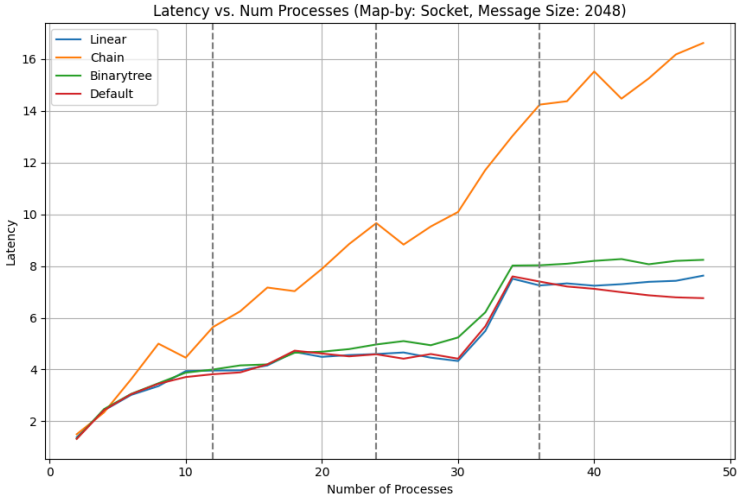



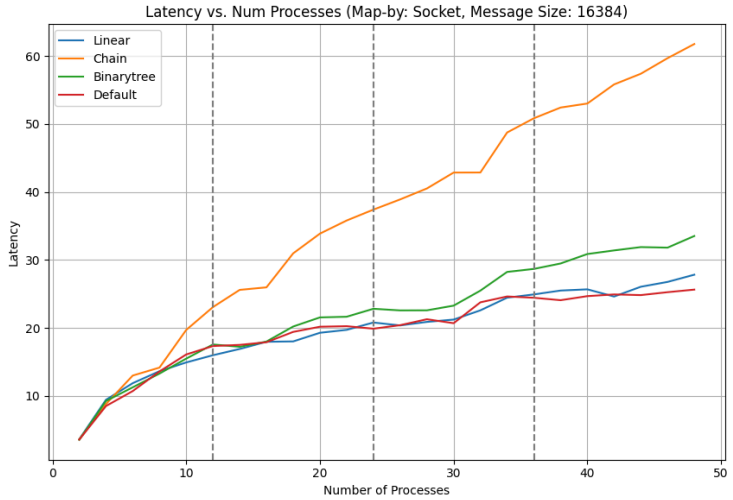



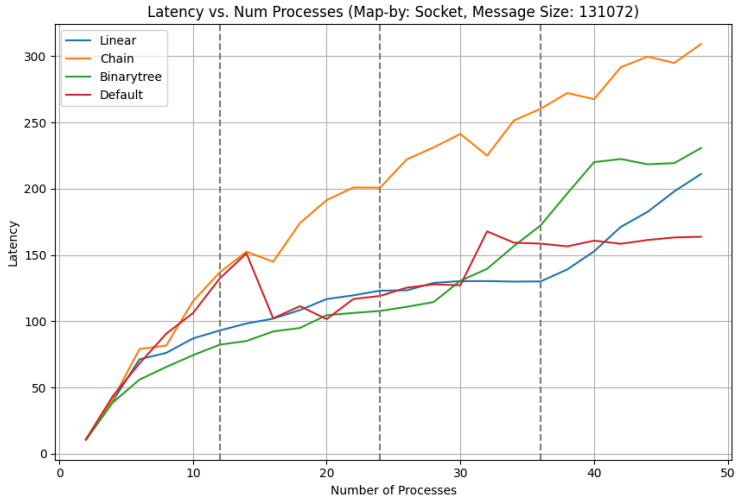
# Latency vs Number of Processes (Fixed Message Size and Socket allocation)

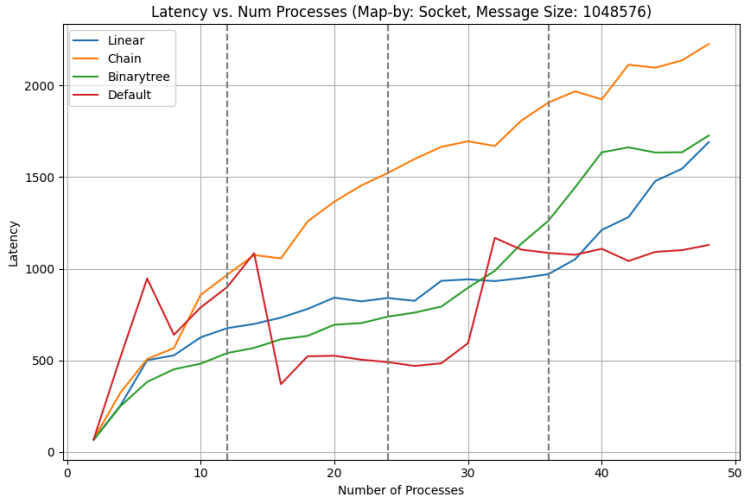


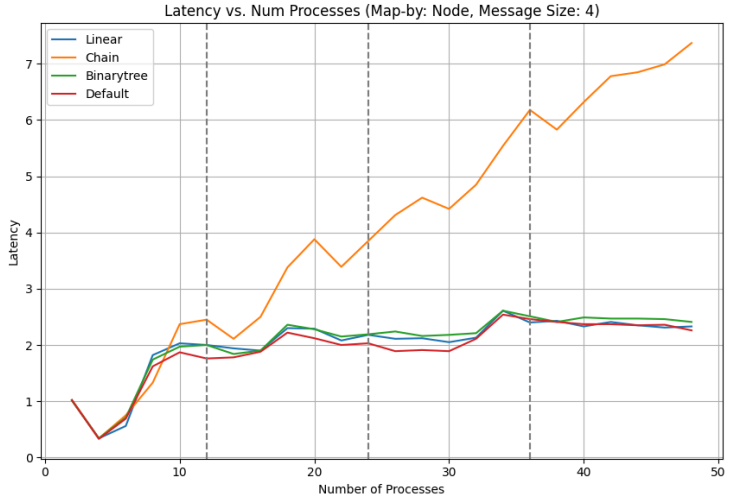



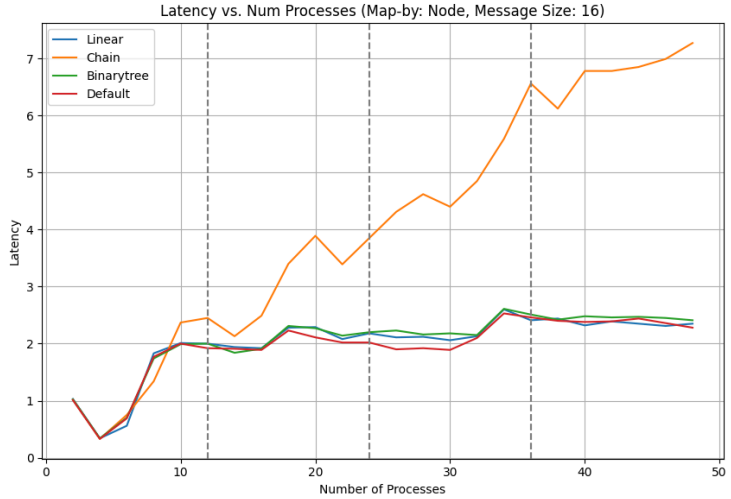



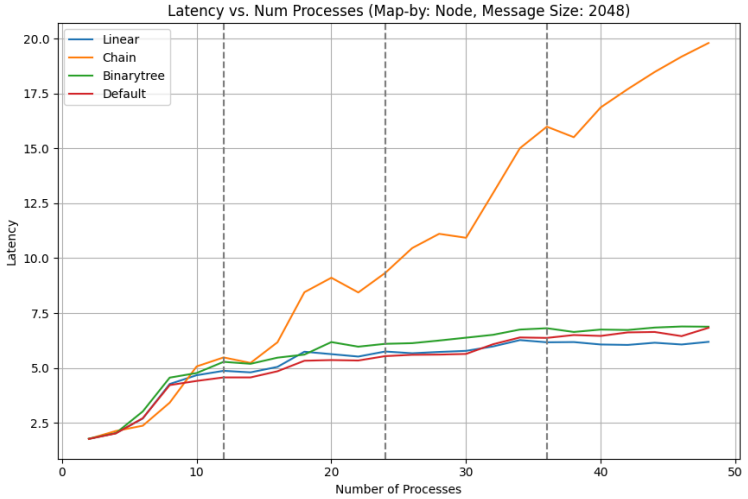



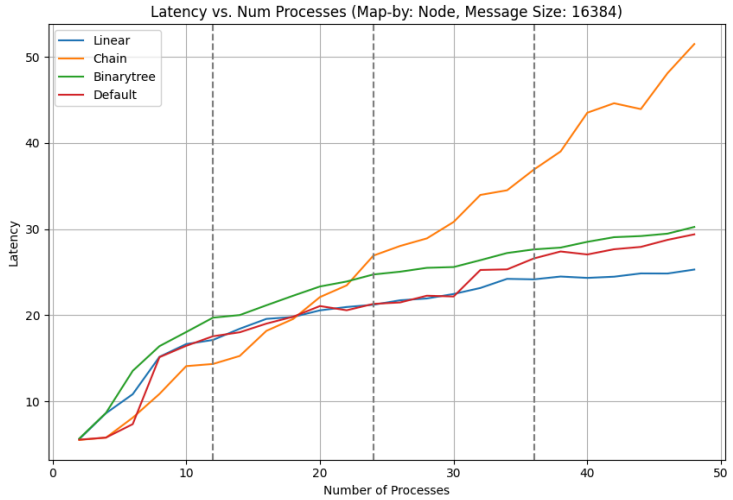



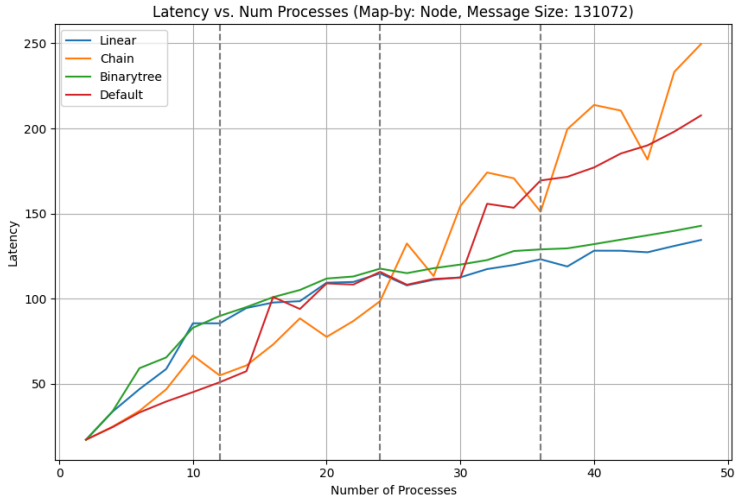
# Latency vs Number of Processes (Fixed Message Size and Node allocation)

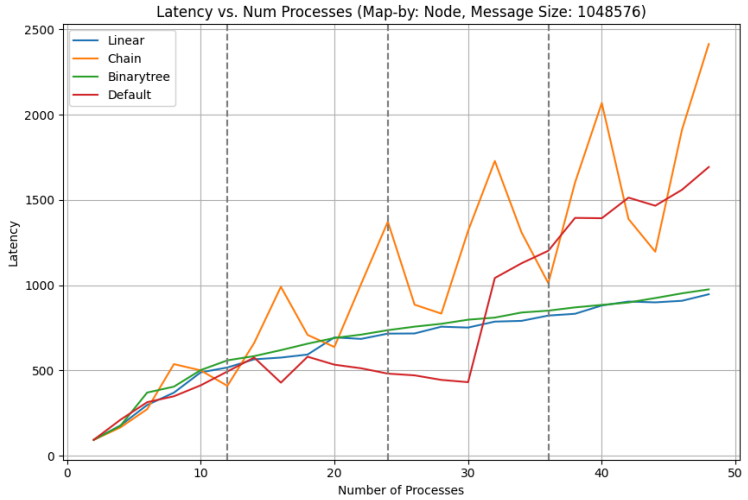


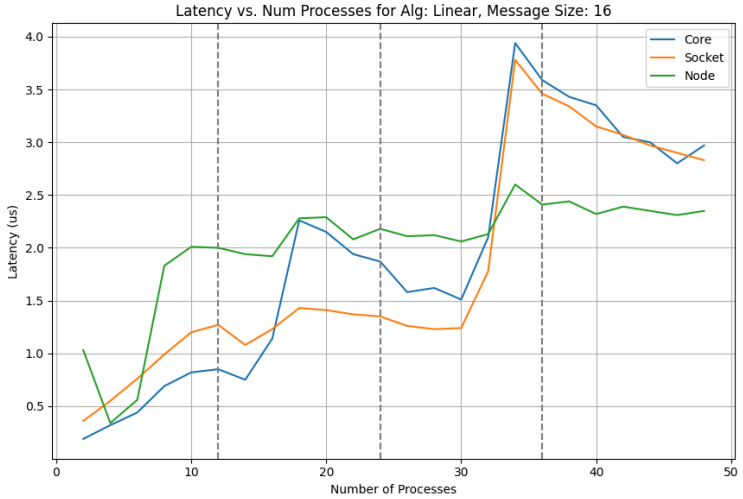



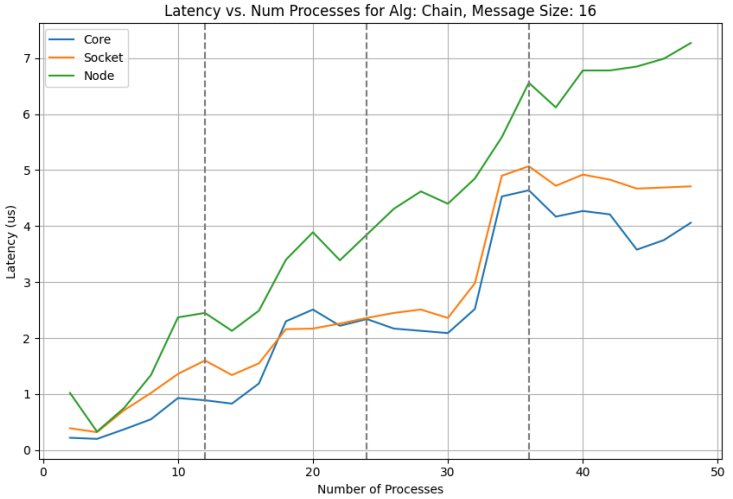



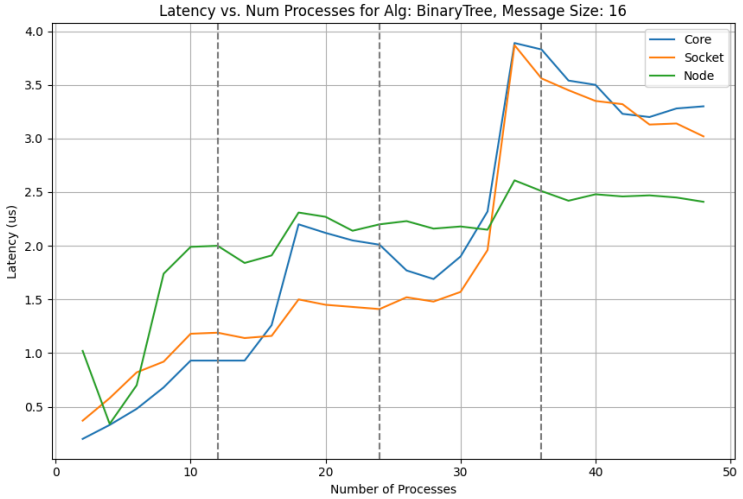



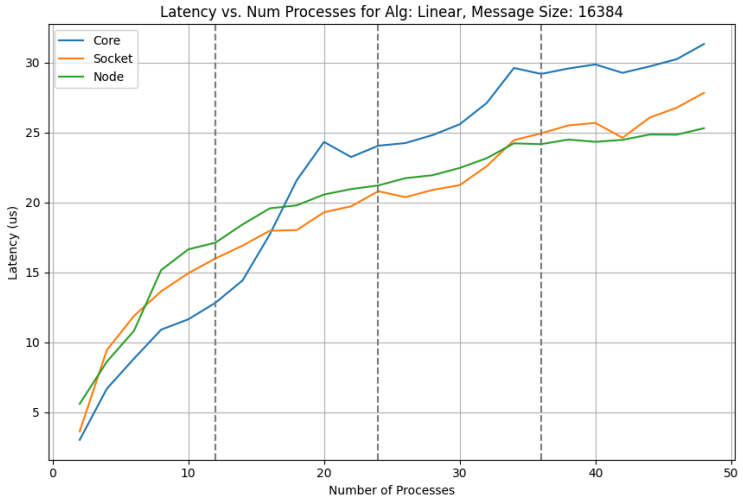



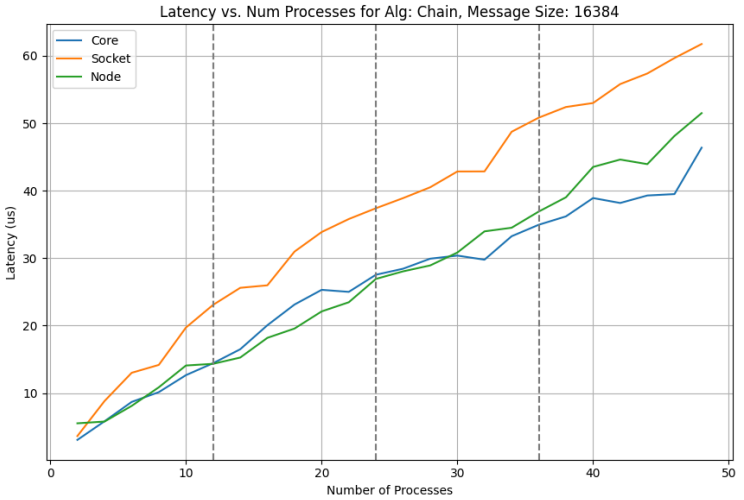
# Latency vs Number of Processes (Fixed Algorithm)

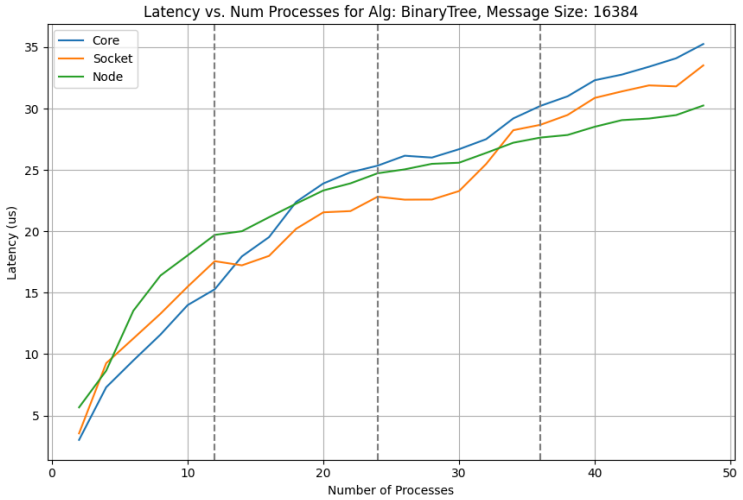


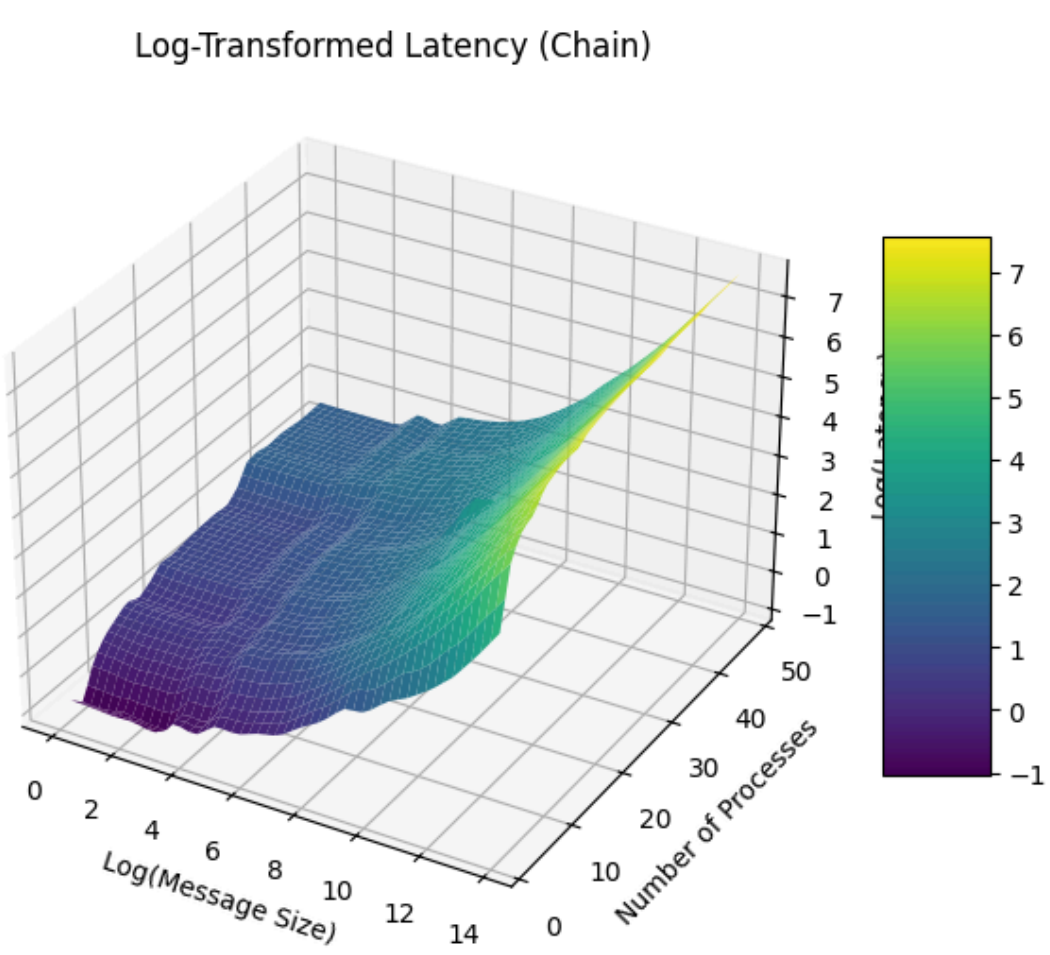



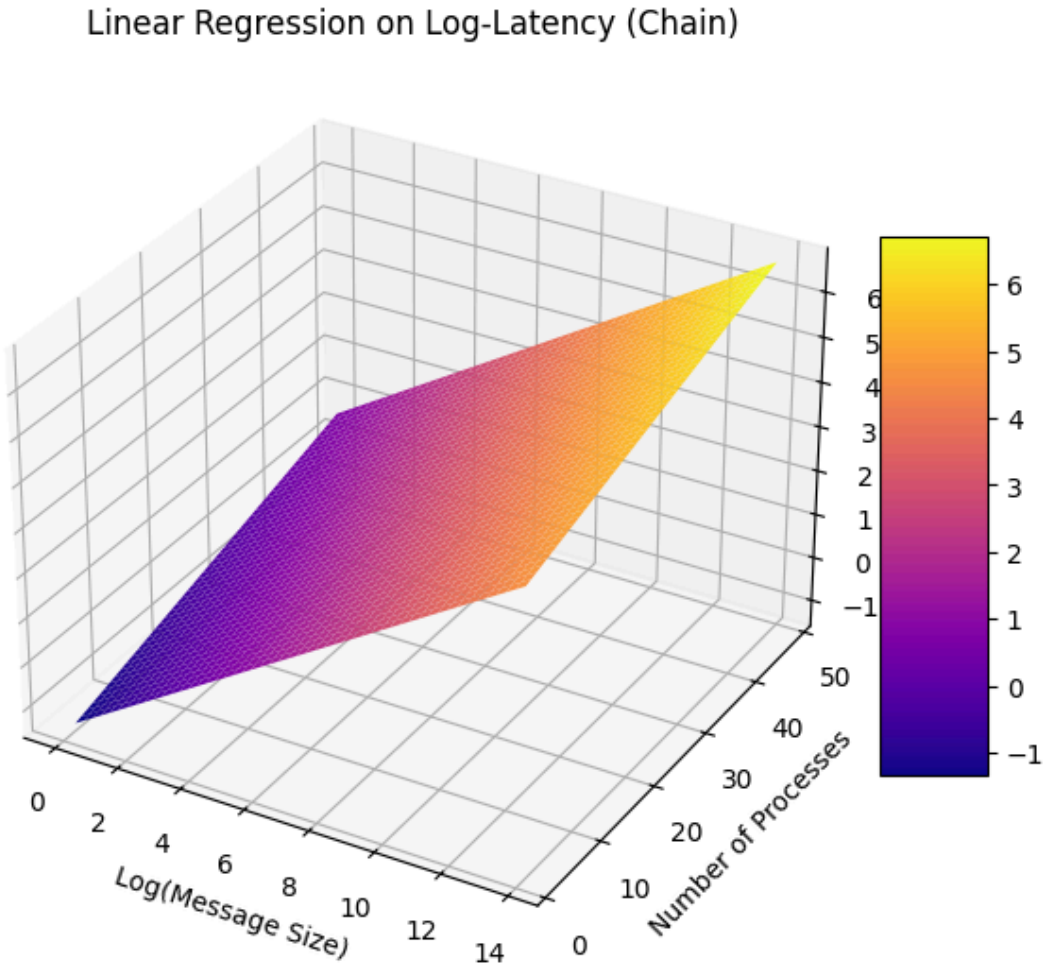



# Comparison of the Default algorithm with the others

| Allocation | Message Size | Processes | #Optimal | Optimal % | Mean Diff | Std Diff |
|---|---|---|---|---|---|---|
| core | Small | Few | 10 | 17.86 | 0.07 | 0.08 |
| core | Small | Medium | 41 | 73.21 | -0.05 | 0.07 |
| core | Small | Many | 30 | 53.57 | 0.09 | 0.61 |
| core | Medium | Few | 10 | 17.86 | 0.17 | 0.21 |
| core | Medium | Medium | 7 | 12.50 | 0.39 | 0.35 |
| core | Medium | Many | 26 | 46.43 | 0.77 | 3.92 |
| core | Large | Few | 16 | 28.57 | 17.28 | 55.45 |
| core | Large | Medium | 9 | 16.07 | 17.22 | 62.55 |
| core | Large | Many | 56 | 100.00 | -65.71 | 83.83 |
| socket | Small | Few | 32 | 57.14 | 0.02 | 0.13 |
| socket | Small | Medium | 52 | 92.86 | -0.11 | 0.09 |
| socket | Small | Many | 34 | 60.71 | -0.01 | 0.05 |
| socket | Medium | Few | 19 | 33.93 | 0.15 | 0.36 |
| socket | Medium | Medium | 12 | 21.43 | 0.12 | 0.16 |
| socket | Medium | Many | 33 | 58.93 | -0.30 | 0.80 |
| socket | Large | Few | 6 | 10.71 | 66.04 | 131.11 |
| socket | Large | Medium | 15 | 26.79 | -7.08 | 98.31 |
| socket | Large | Many | 33 | 58.93 | -33.76 | 125.58 |
| node | Small | Few | 28 | 50.00 | 0.04 | 0.18 |
| node | Small | Medium | 55 | 98.21 | -0.15 | 0.09 |
| node | Small | Many | 31 | 55.36 | 0.00 | 0.06 |
| node | Medium | Few | 24 | 42.86 | 0.22 | 0.50 |
| node | Medium | Medium | 41 | 73.21 | -0.03 | 0.23 |
| node | Medium | Many | 17 | 30.36 | 0.66 | 1.12 |
| node | Large | Few | 30 | 53.57 | -2.62 | 28.18 |
| node | Large | Medium | 19 | 33.93 | -12.58 | 89.62 |
| node | Large | Many | 0 | 0.00 | 146.32 | 192.01 |

# Performance Models



# Conclusion

The default algorithm doesn't always choose the best algorithm to perform the broadcast, expecially for node allocation, large message size and many processes, with a remarkable difference.

# Barrier Algorithms analysed

### Linear:

- All nodes report to a preselected root
- Linear communication steps.

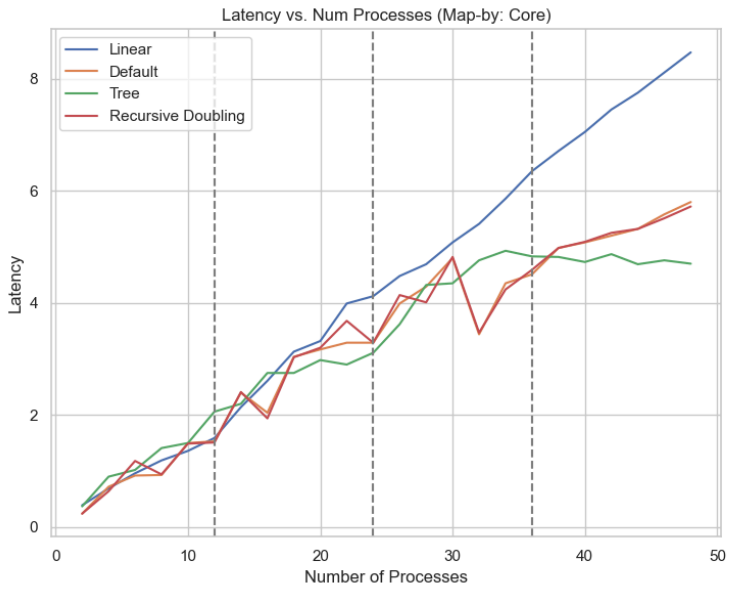
### Tree:

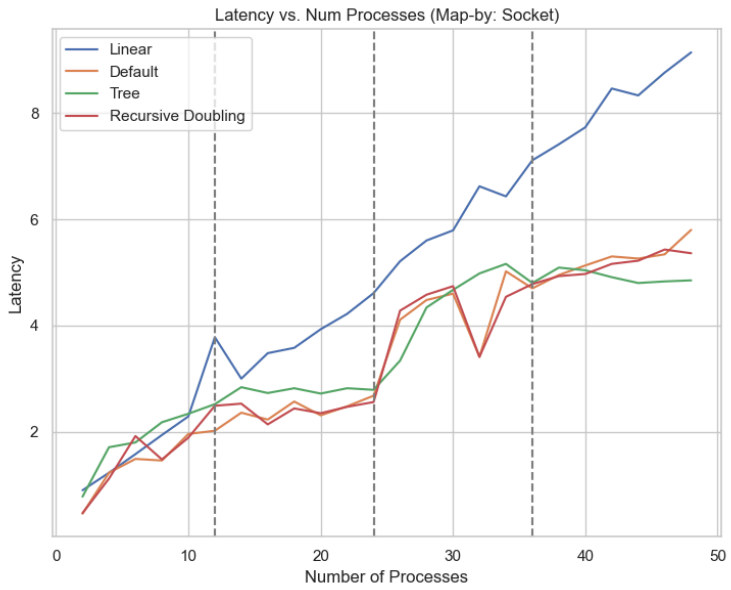
- Hierarchical synchronization in a tree-like structure.
- Logarithmic communication steps.

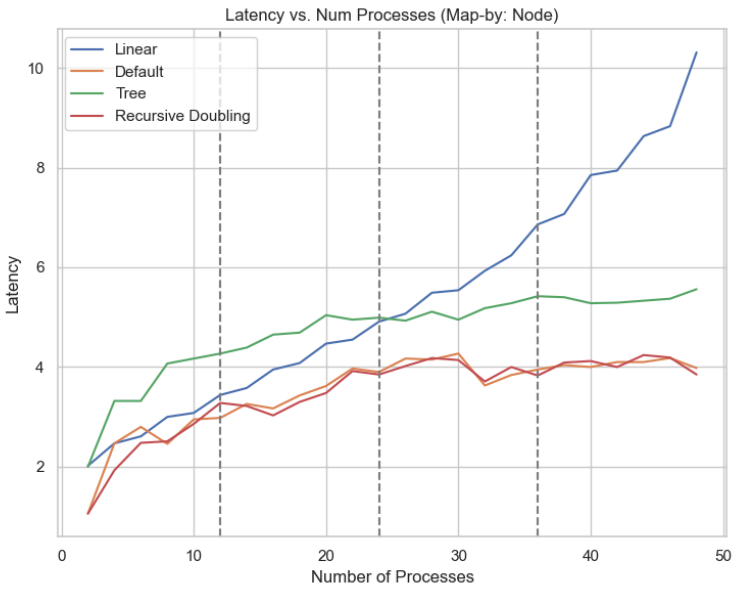
### Recursive Doubling:

- Optimal for powers of 2. At step $k$, node $r$ exchanges a message with node $(r \texttt{XOR} 2^k)$.
- Logarithmic communication steps.

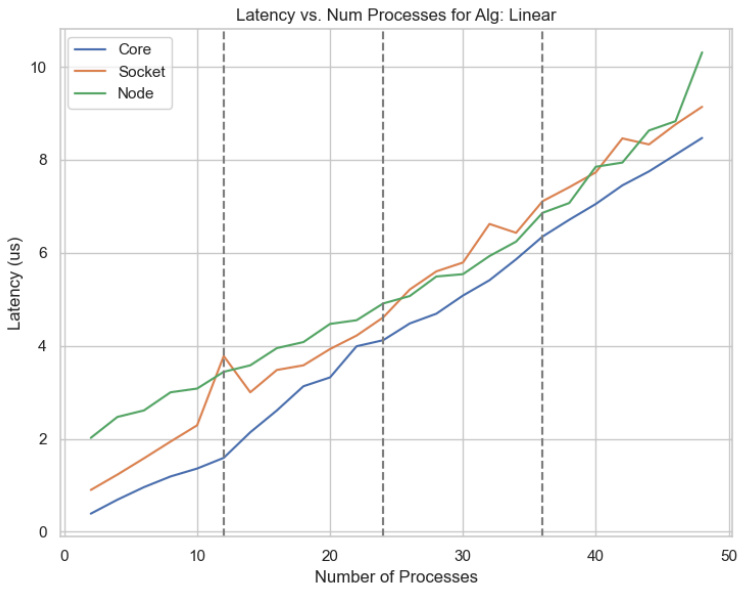
# Latency against number of processes (Fixed allocation)

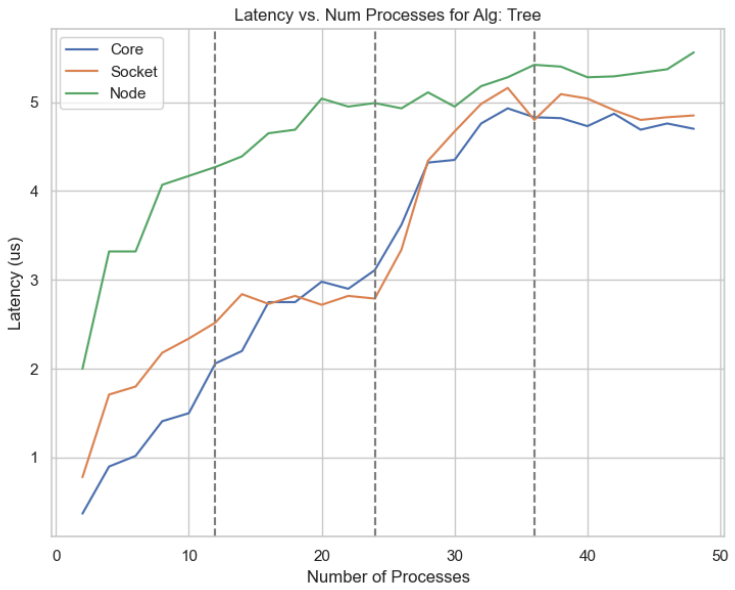


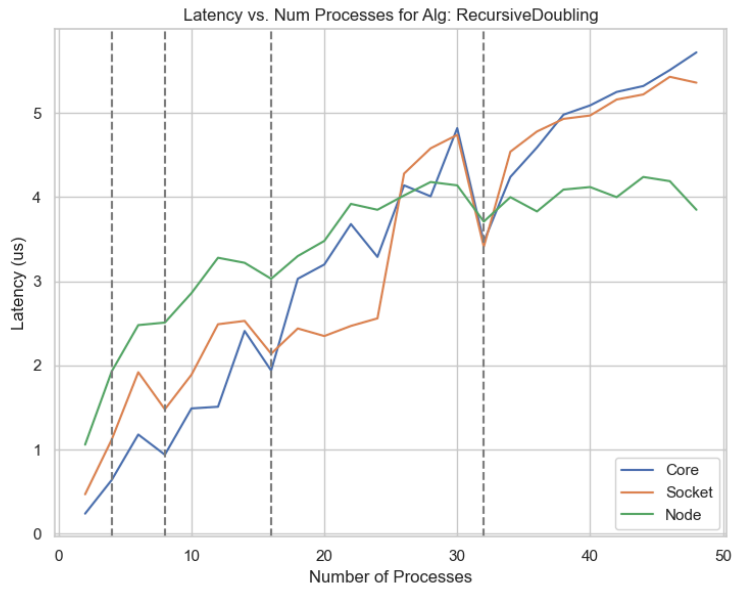



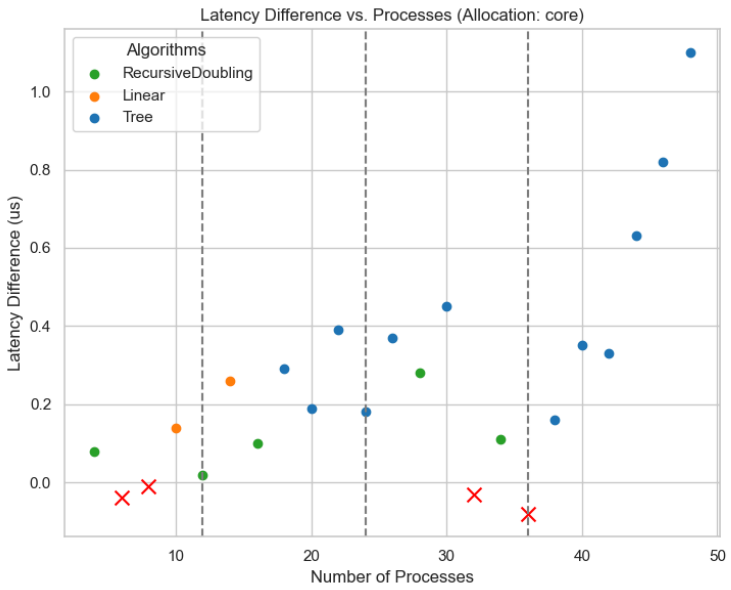
# Latency against number of processes (Fixed algorithm)

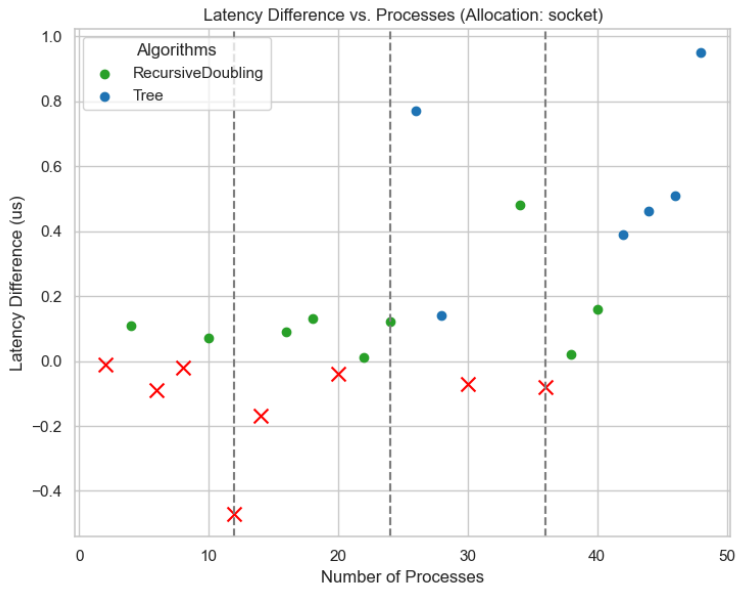


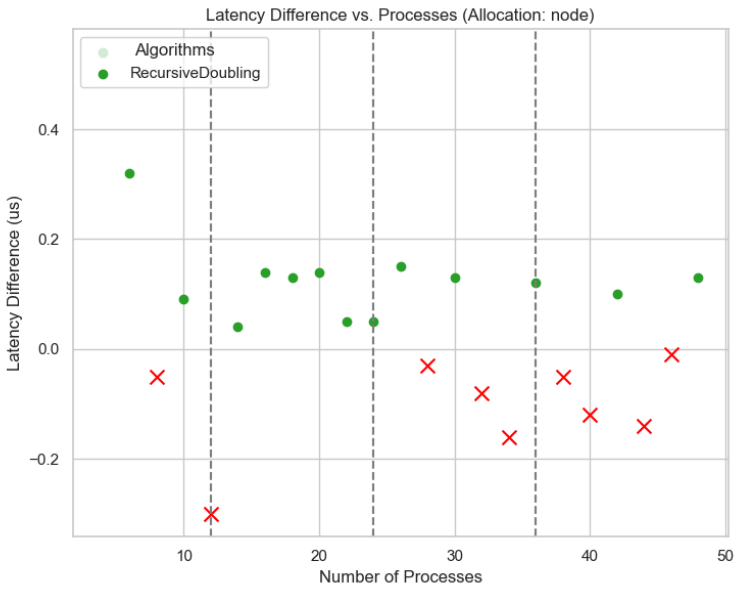



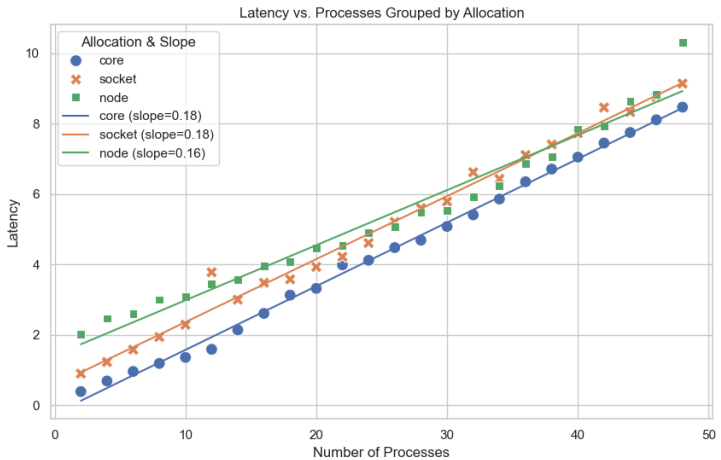



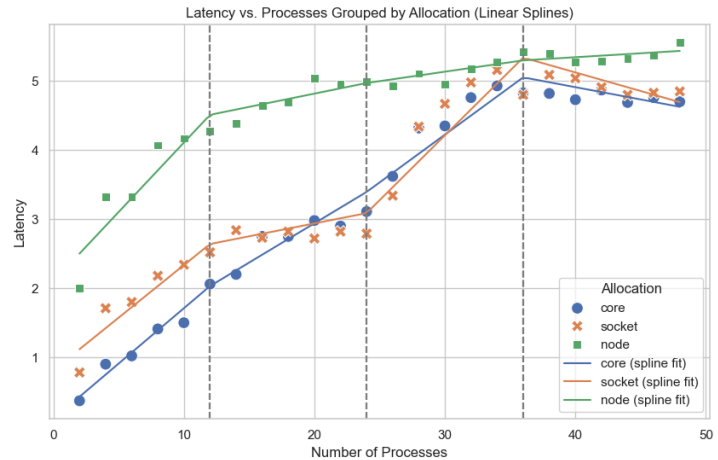
# Comparison of the Default algorithm with the others

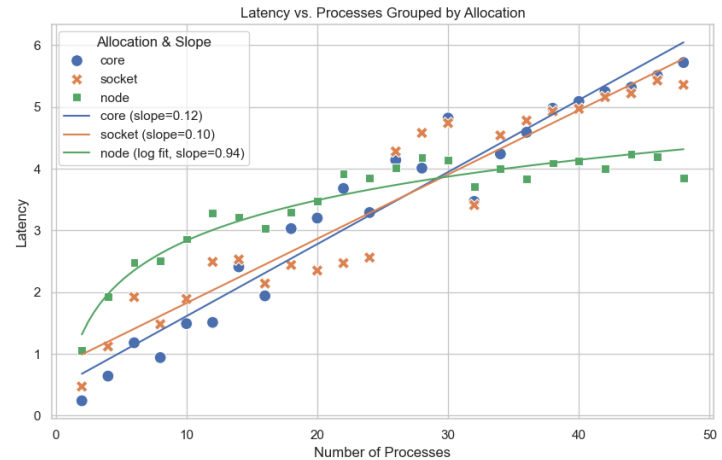



Latency vs. Processes Grouped by Allocation
Allocation & Slope
core
socket
node
core (slope=0.18)
socket (slope=0.18)
node (slope=0.16)
Latency
Number of Processes


Latency vs. Processes Grouped by Allocation (Linear Splines)
Allocation
core
socket
node
core (spline fit)
socket (spline fit)
node (spline fit)
Latency
Number of Processes


Latency vs. Processes Grouped by Allocation
Allocation & Slope
core
socket
node
core (slope=0.12)
socket (slope=0.10)
node (log fit, slope=0.94)
Latency
Number of Processes

## Conclusion

OpenMPI's default algorithm doesn't always choose the optimal algorithm in terms of latency, but in the case of `MPI_Barrier` the difference is not remarkable.