EXERCISE 1
HIGH PERFORMANCE COMPUTING

Giulio Fantuzzi

PROBLEM OVERVIEW AND METHODOLOGY

PROJECT AIMS

- To assess the performance of various **openMPI** algorithms for specific collective operations
- To build a performance model for the latency

COLLECTIVE MPI OPERATIONS

- Broadcast
- Barrier

LOTS OF DEGREES OF FREEDOM

- Algorithm
- Message Size
- Number of MPI processes
- Allocation type
- Additional parameters (iterations, warm-up,...)

ARCHITECTURE

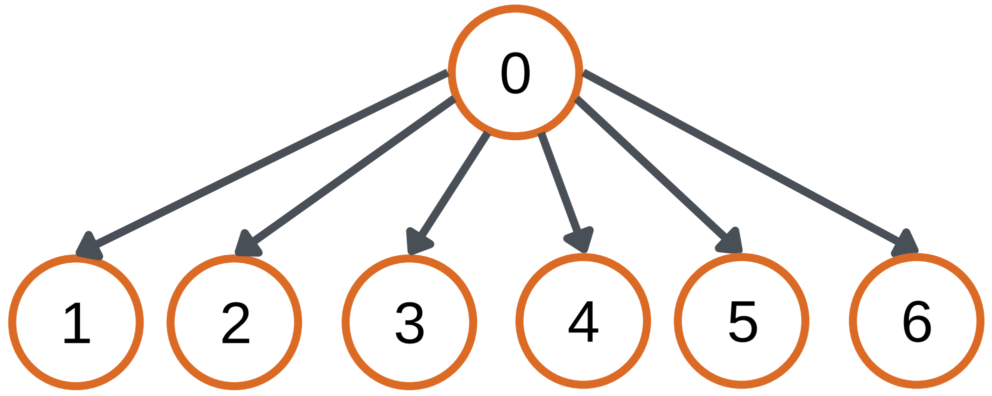
- **2 *THIN*** nodes of the **ORFEO** cluster
- 2 sockets per node
- 12 cores per socket

METHODOLOGY

- **OSU** benchmark to estimate the latency
- Bash scripts to automate data gathering phase
- **SLURM** workload manager
- R for data analysis and models

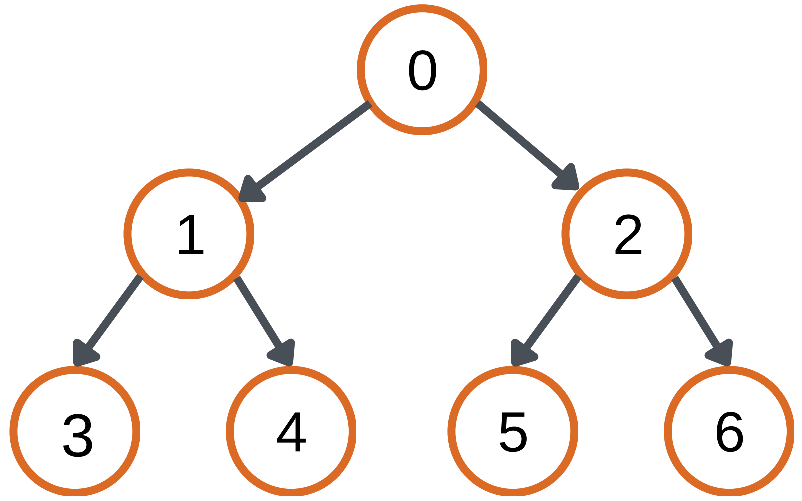
BROADCAST ALGORITHMS

1 LINEAR

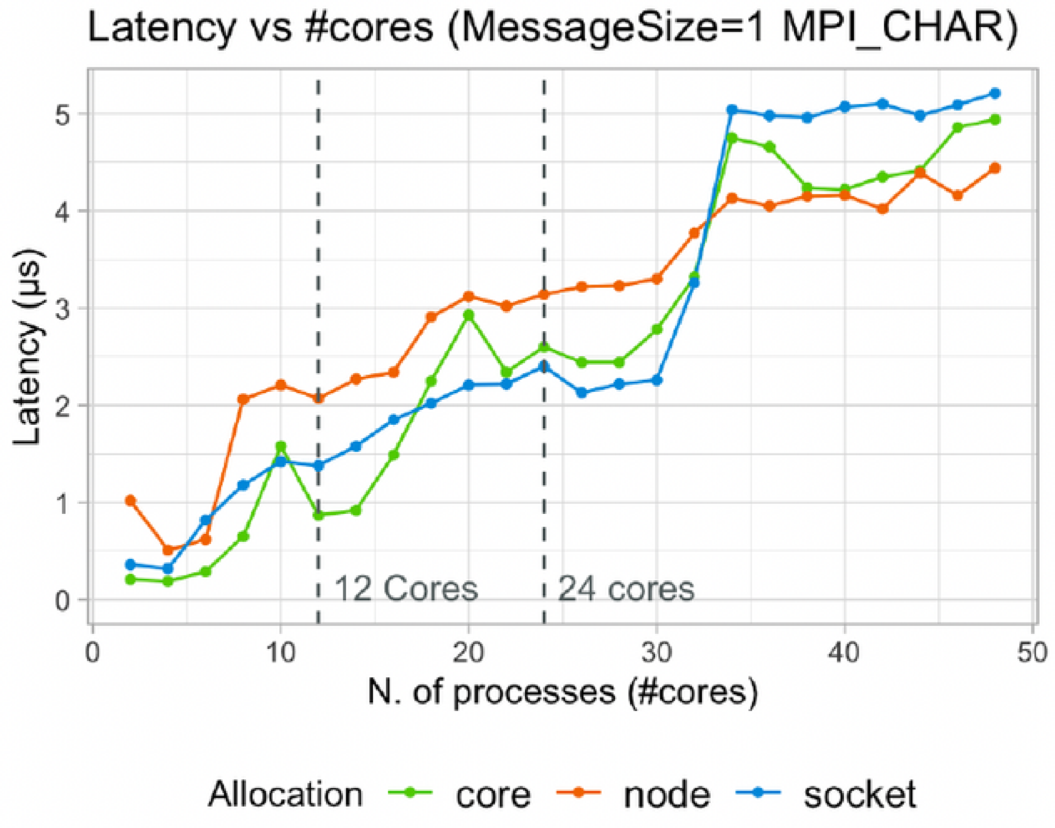


2 CHAIN

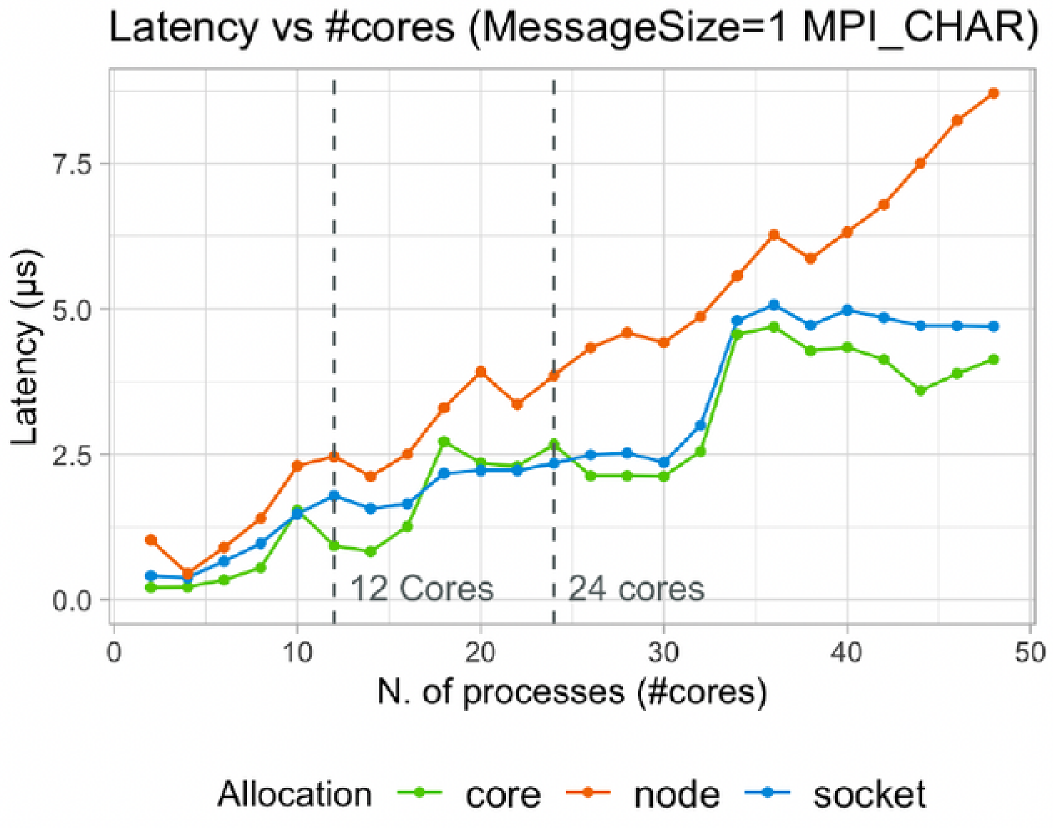
3 BINARY TREE



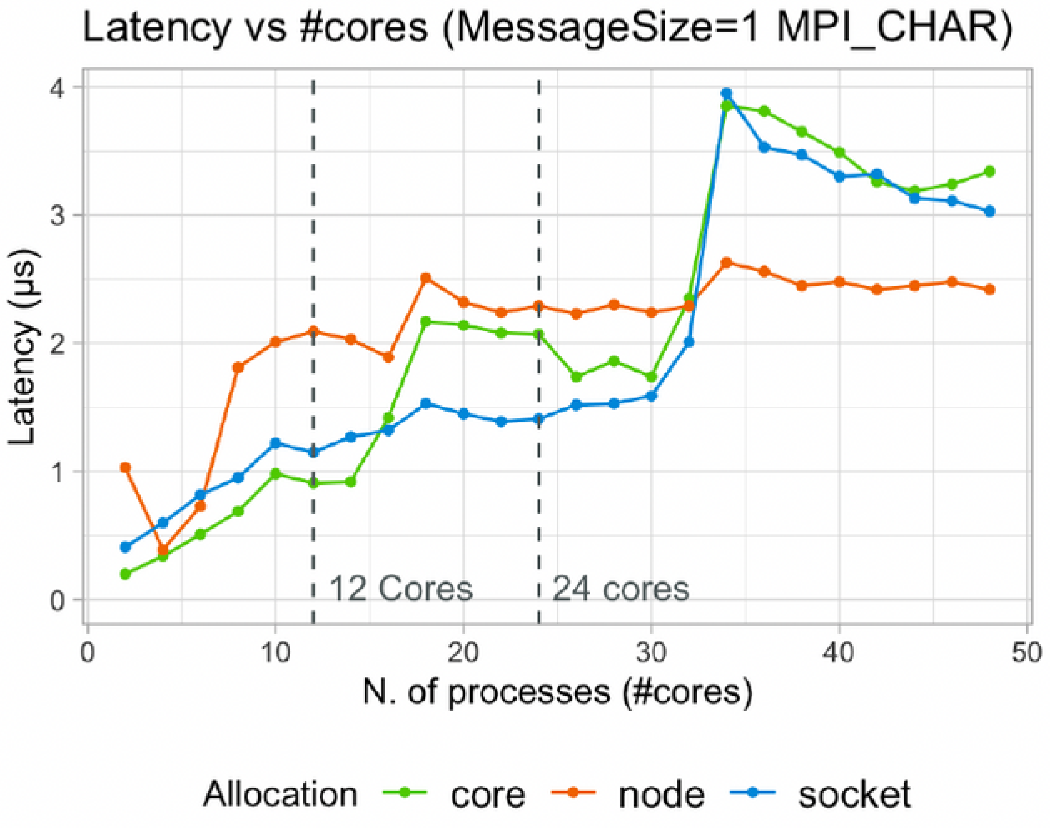
BCAST: ALLOCATION TYPE COMPARISON



(a) Linear algorithm



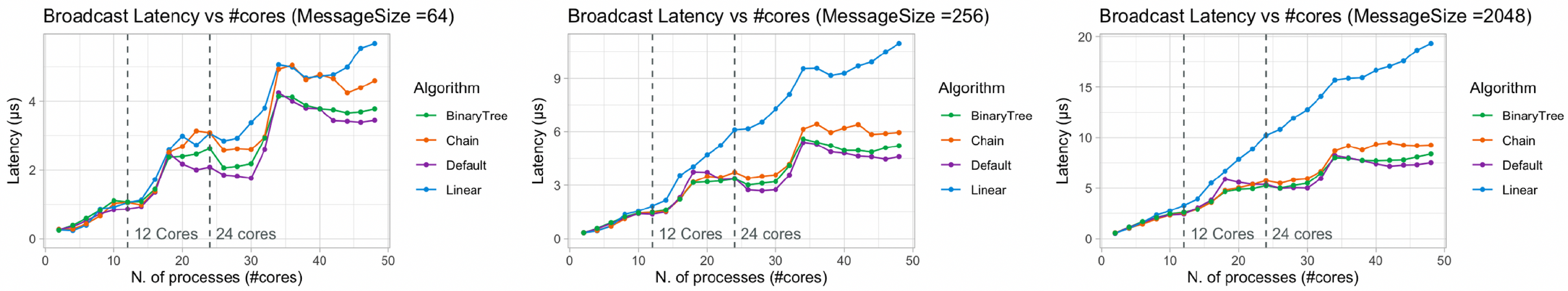
(b) Chain algorithm



(c) Binary Tree algorithm

- Socket and core allocations show latency “jumps” when changing node, while node allocation progresses seamlessly
- Similar considerations apply to socket changes
- Surprisingly, jumps occur at 16 and 32, hinting at additional factors influencing communication
- Lines convergence: Linear vs Chain algorithm
- Tree algorithm (expected) superior performance

BCAST: ALGORITHMS COMPARISON



- With a small fixed size, algorithmic implementation has a minimal impact
- Tree algorithm and default configuration outperform linear and chain
    - Tree structure efficiently distributes the communication among its branches
    - Default configurations are robust and efficient in different environments, leveraging platform-specific optimizations
- Linear vs chain insightful points of comparison:
    - Chain superior performance, primarily attributable to its utilization of contiguous cores
    - Chain capacity to partition messages into chunks during transmission, a feature absent in the linear

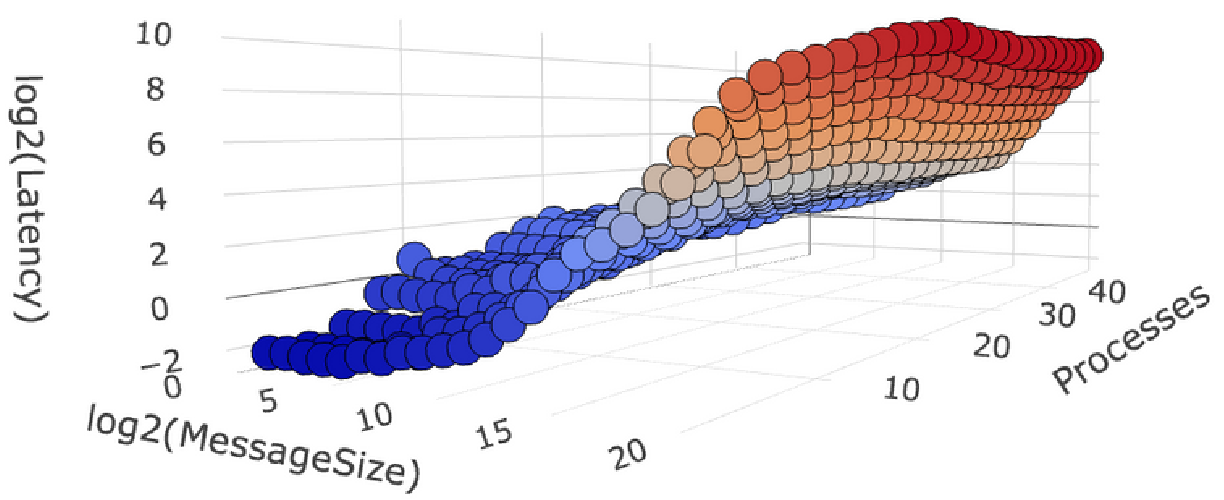
BCAST: PERFORMANCE MODELS

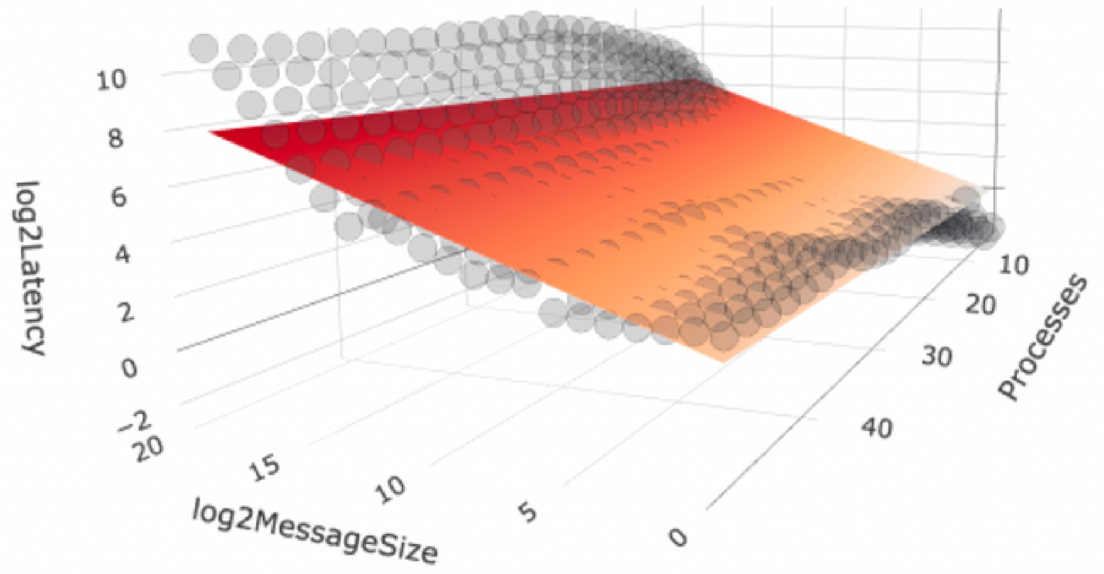
- **INITIAL APPROACH**: estimate pt2pt latency and bandwidth to construct a model resembling the *Hockney's model*
- Unfortunetely, results did not align well with the collected data, prompting me to explore statistical methodologies

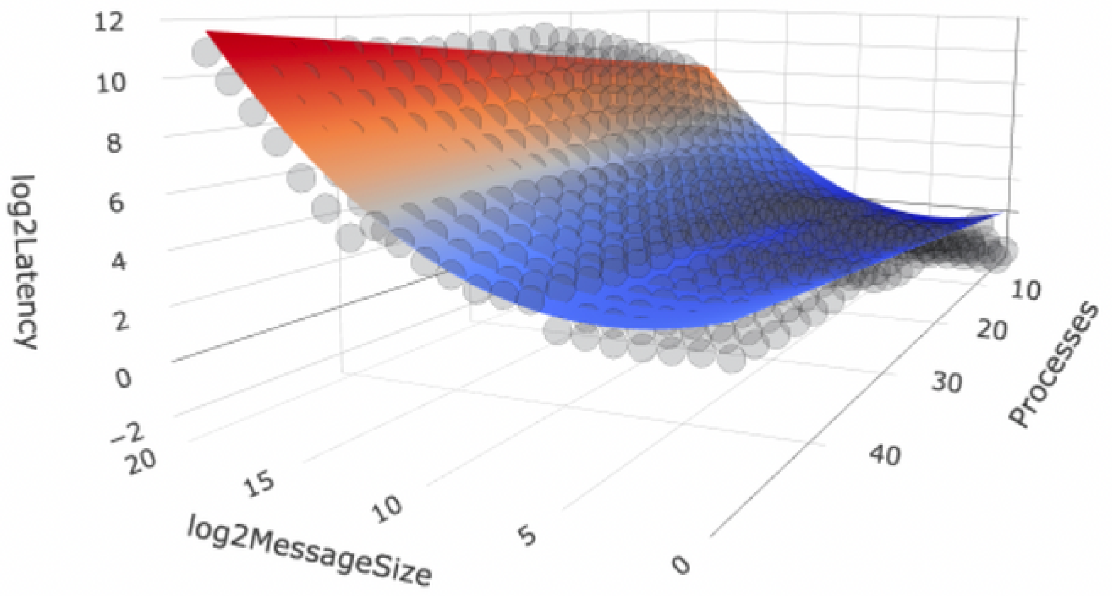
$$\boxed{log_2(\text{Latency}) = \beta_1 \cdot \text{Number of Processes} + \beta_2 \cdot log_2(\text{Message Size}) + \beta_3 \cdot log_2(\text{Message Size})^2}$$

Algorithm	β_1	β_2	R^2_{adj}
Linear	0.029815	0.321259	88,19 %
Chain	0.021957	0.306713	86,77 %
Binary Tree	0.016315	0.317449	86,58 %
Default	0.009586	0.321418	85,86 %

Algorithm	β_1	β_2	β_3	R^2_{adj}
Linear	0.0718655	-0.3022613	0.0355722	97,57 %
Chain	0.0646477	-0.3262914	0.0361133	98,00 %
Binary Tree	0.0600791	-0.3314744	0.0370215	98,38 %
Default	0.0525293	-0.3153369	0.0363273	97,74 %



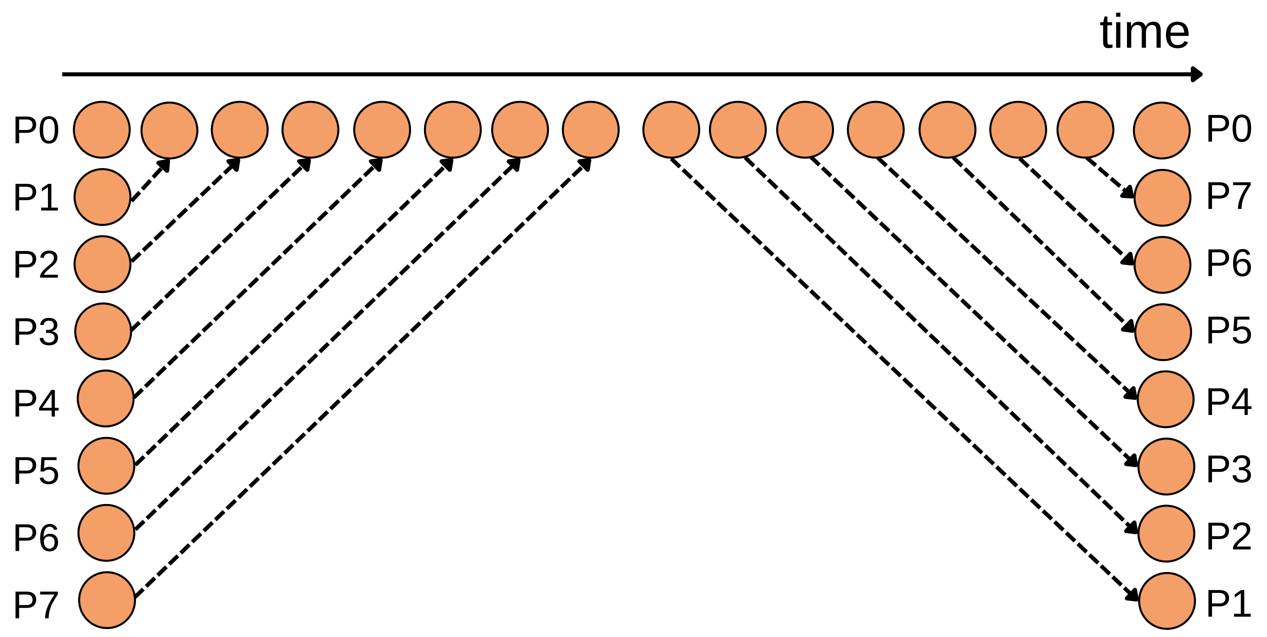




BARRIER ANALYSIS

- The analytical approach employed for Barrier closely follows the methodology followed for Broadcast
- Barrier aims for synchronization without involving message sizes → fewer degrees of freedom!
- My analysis includes the default MPI configuration, along with 3 additional algorithms:
  - **LINEAR ALGORITHM**

    All the processes report to a pre-selected root. Once every process has reported to the root, the root sends a releasing message to all participants

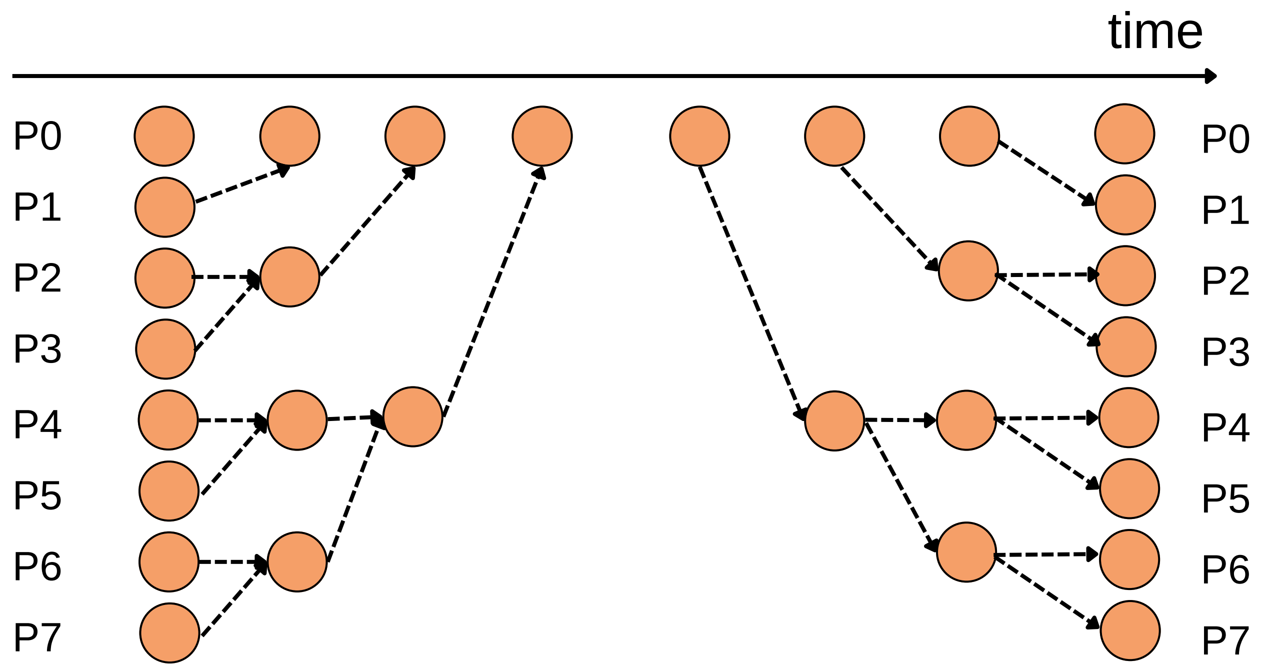


  - **BRUCK ALGORITHM**

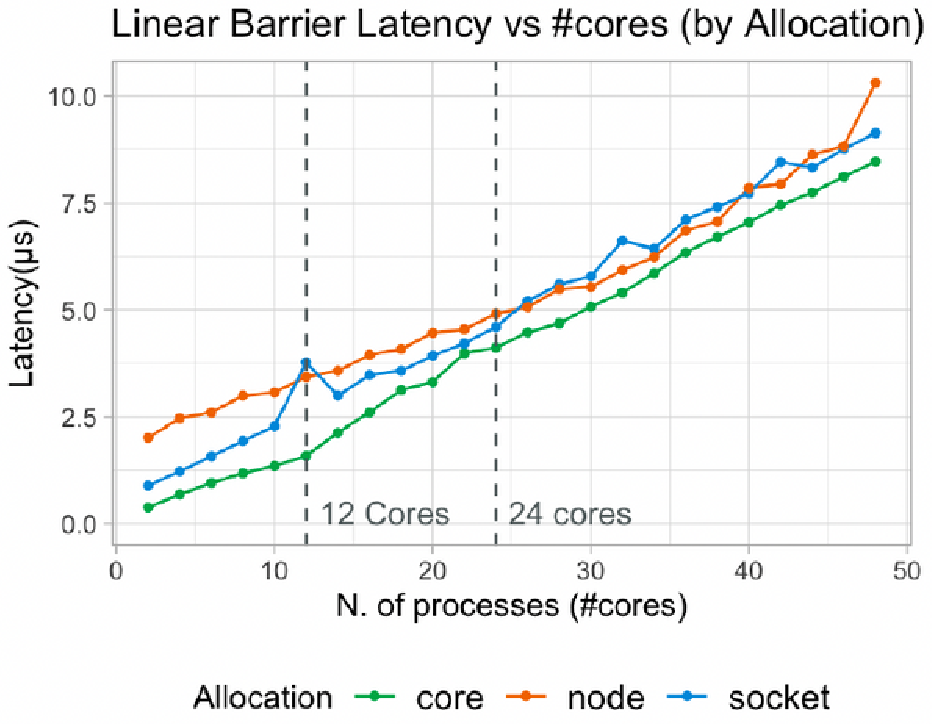
    It requires log2(P) communication steps: at **step k**, process **r** receives a zero-byte message and sends it to process **(r-2^k)** and **(r+2^k)**, with *wrap around*

  - **TREE ALGORITHM**

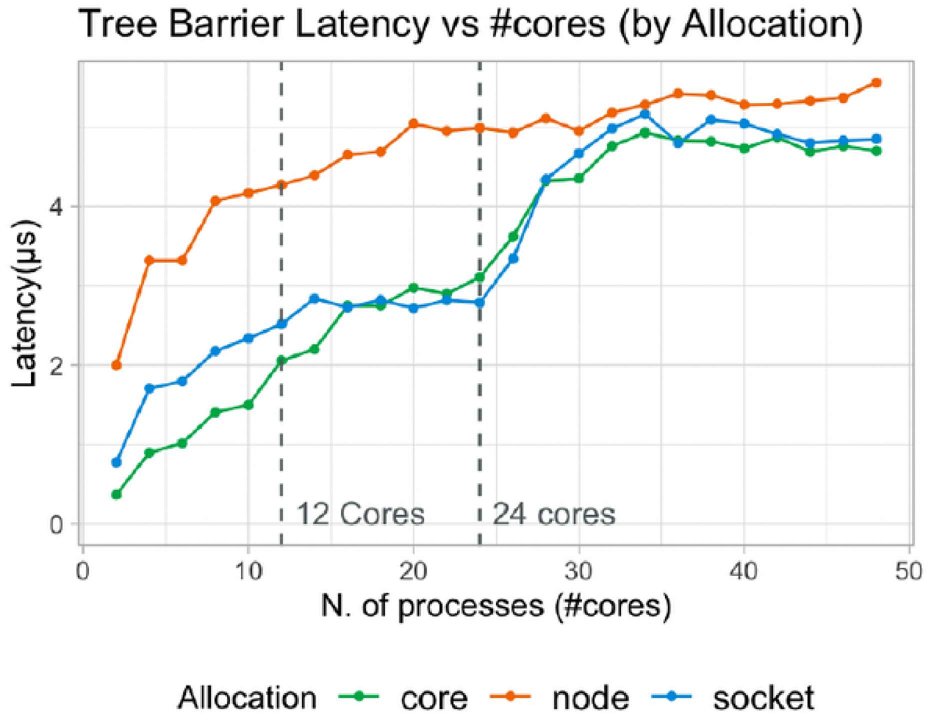
    All the processes report to a pre-selected root. Once every process has reported to the root, the root sends a releasing message to all participants



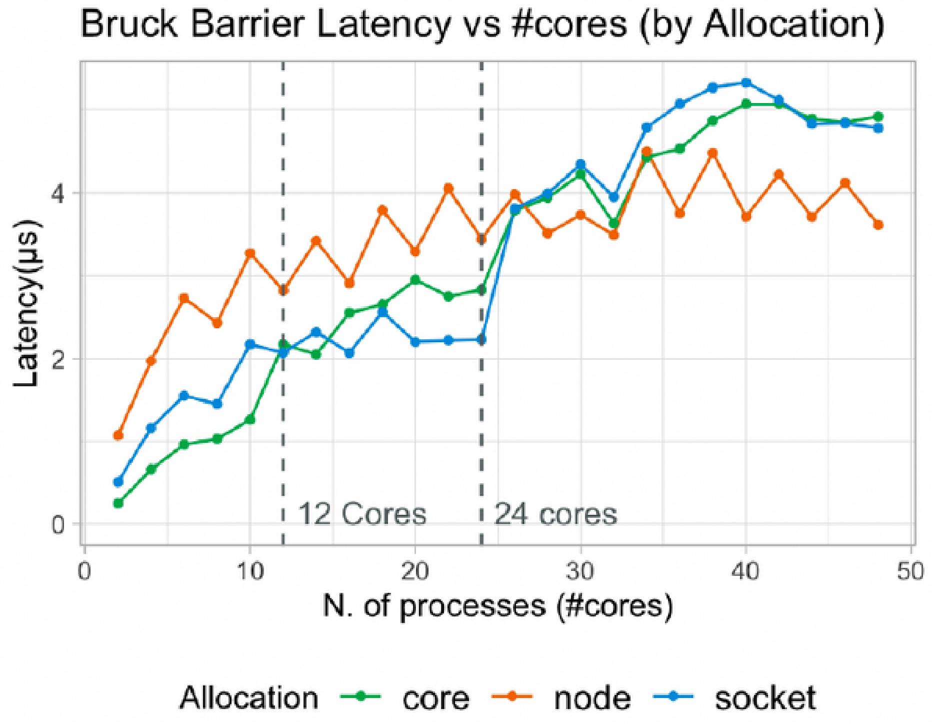
BARRIER: COMPARING ALLOCATION AND ALGORITHMS



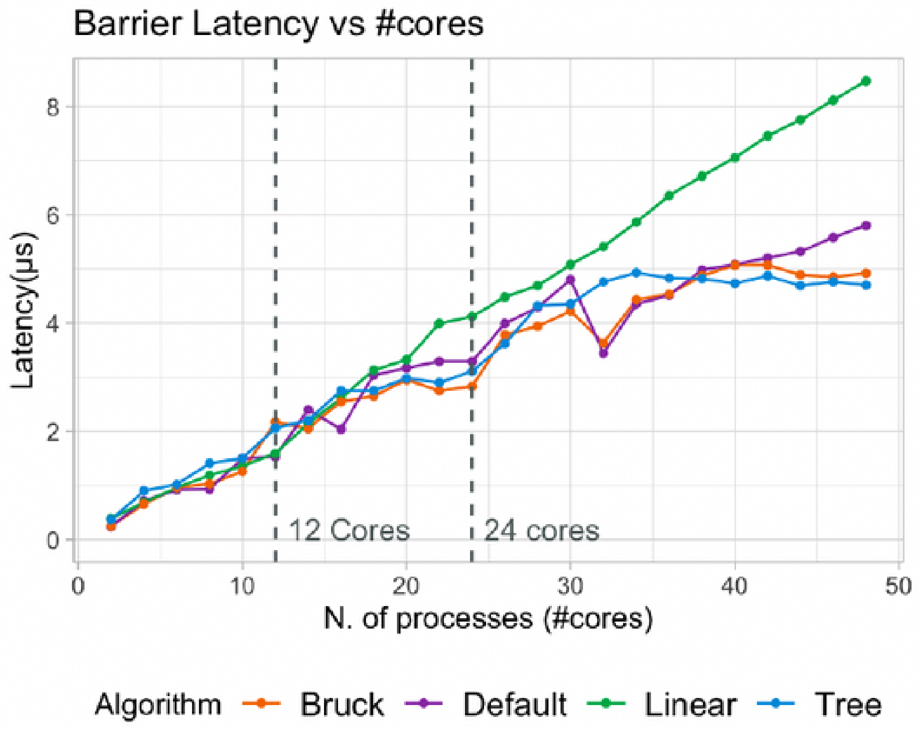
(a) Linear algorithm



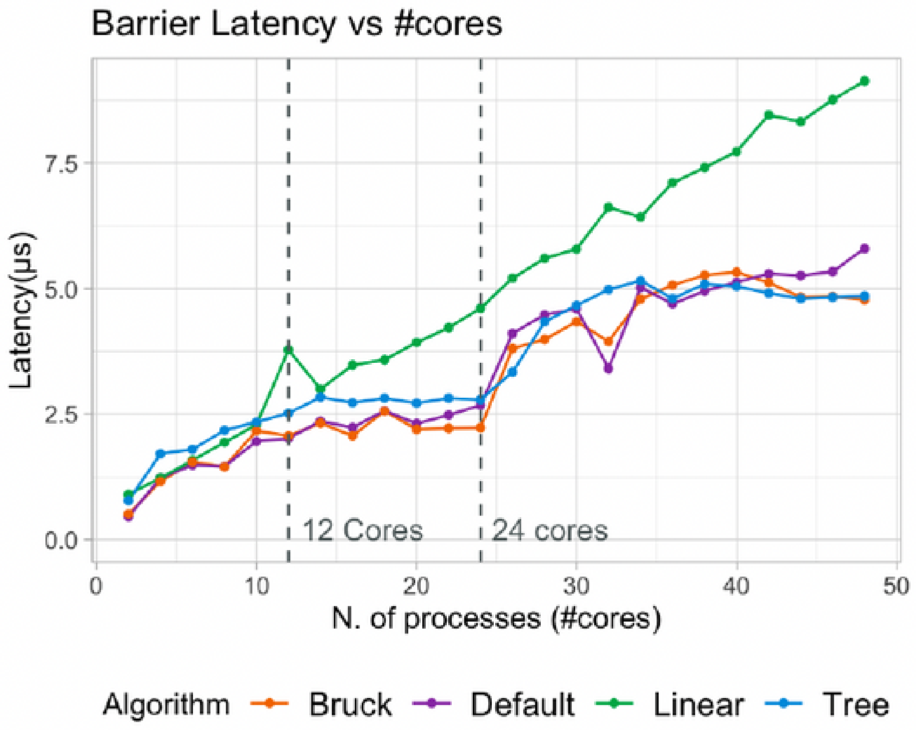
(b) Tree algorithm



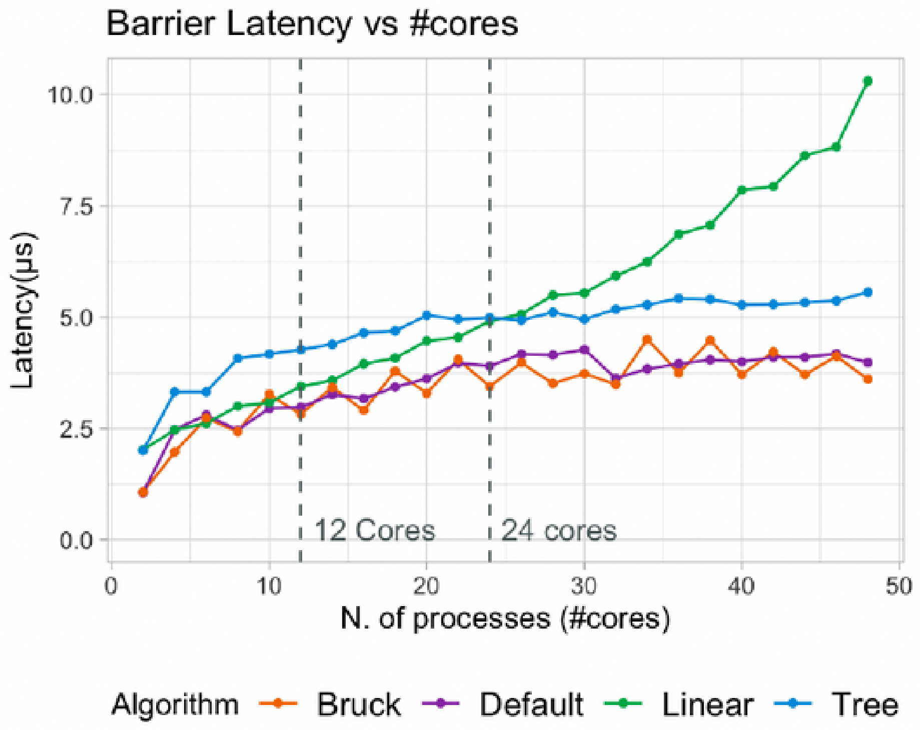
(c) Bruck algorithm



(a) Configuration by core



(b) Configuration by socket



(c) Configuration by node

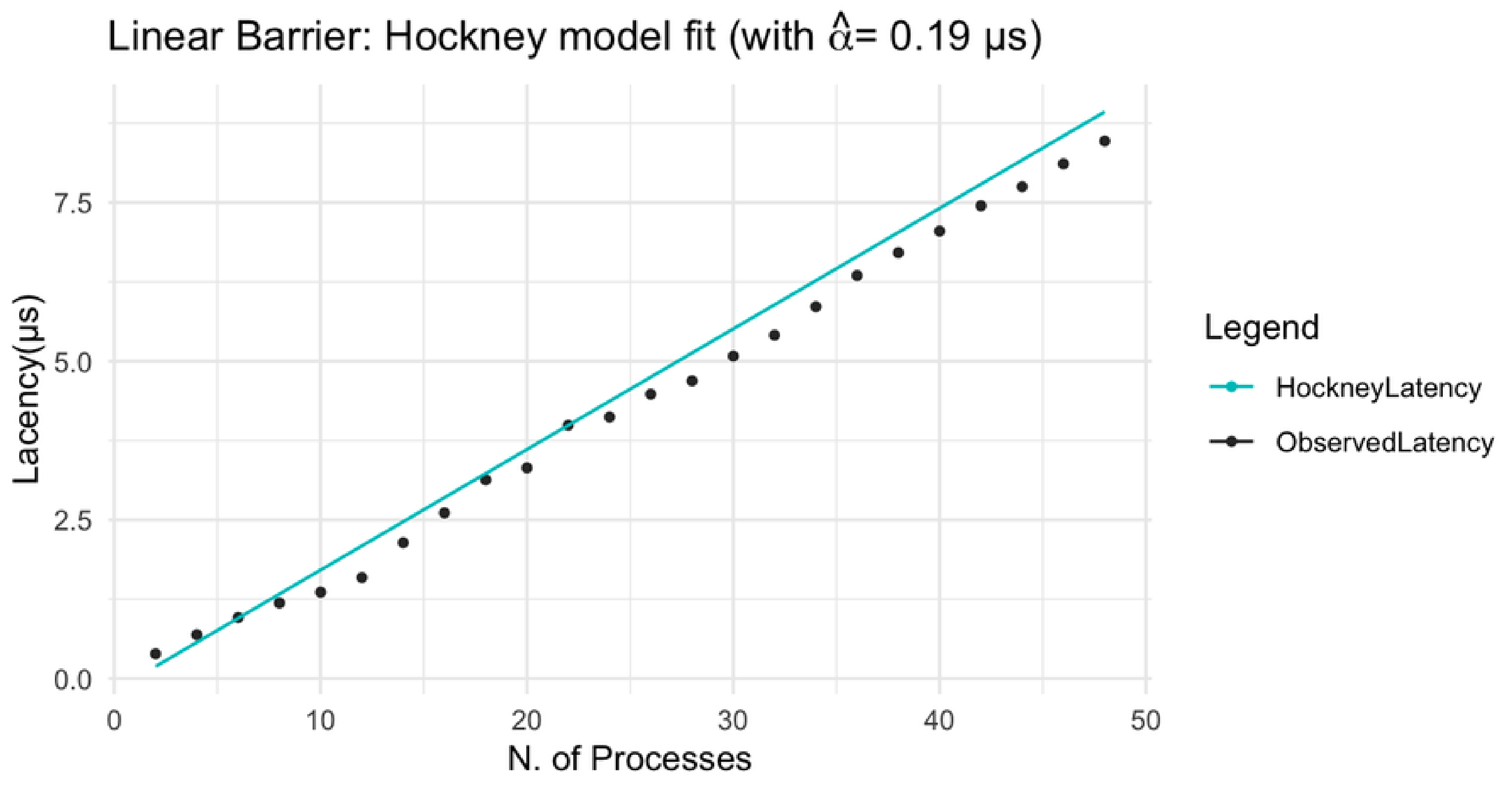
BARRIER: PERFORMANCE MODEL FOR LINEAR ALG

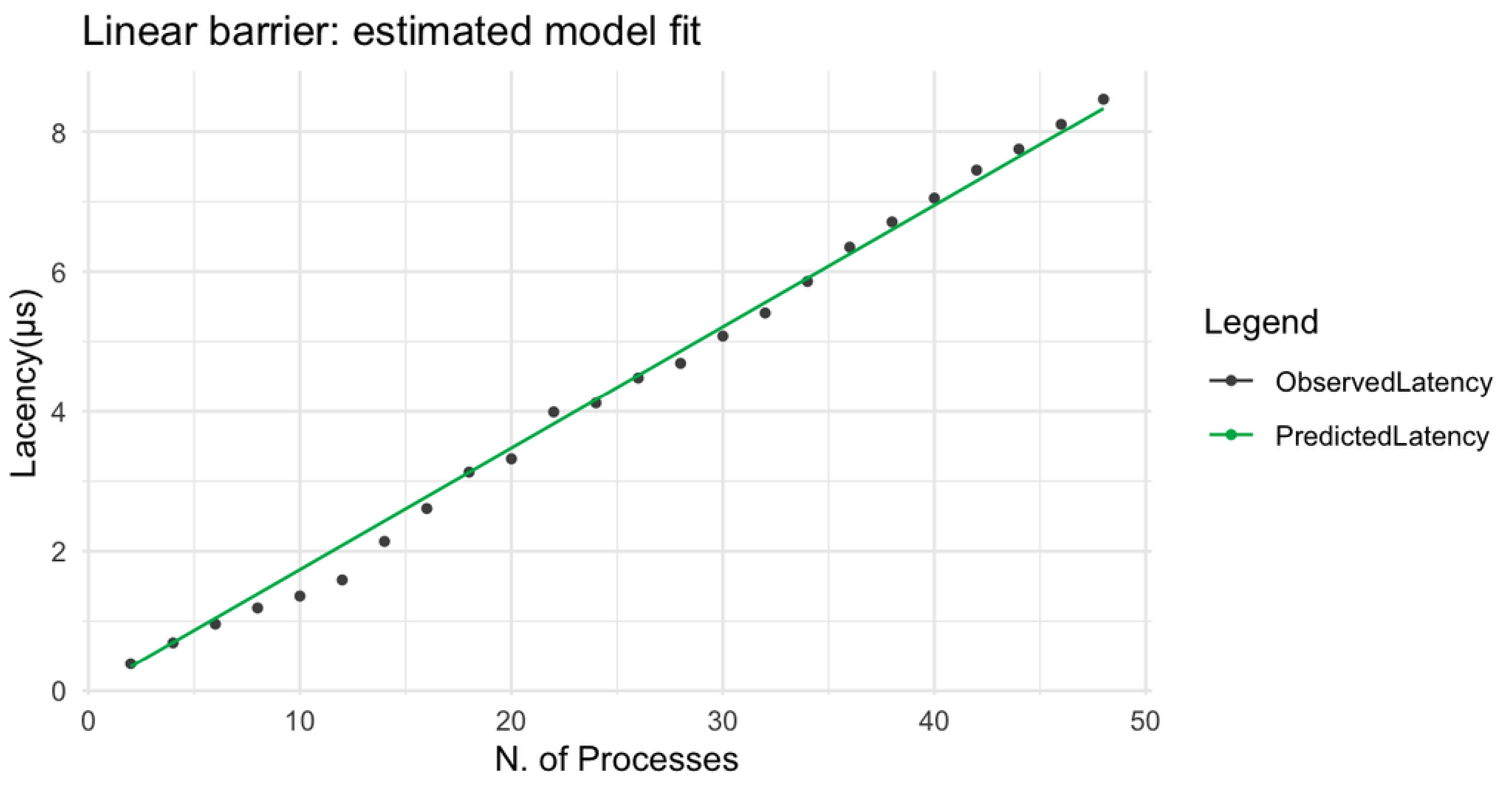
- The chosen approach was to construct **algorithm-specific** models, in accordance with established practices in the literature
- For linear algorithm, I successfully integrated the estimated pt2pt latency (from **OSU**) into a Hockney model

$$\boxed{\text{Latency} = \hat{\alpha}(P-1) = \hat{\alpha}P - \hat{\alpha}}$$

- As a curiosity, I implemented a linear regression model

$$\boxed{\text{Latency} = \beta_1 \cdot \text{Number of Processes}}$$





BARRIER: PERFORMANCE MODEL FOR NON-LINEAR ALGS

- Attempts to extend the Hockney model to other barrier algorithms were unsuccessful
- For all the non-linear models, the introduction of a quadratic term improved the model's fit

$$\boxed{\text{Latency} = \beta_1 \cdot \text{Number of Processes} + \beta_2 \cdot (\text{Number of Processes})^2}$$

Algorithm	β_1	β_2	R^2_{adj}
Linear	0.1736878	/	99,86 %
Tree	0.1950677	-0.0019060	99,46 %
Bruck	0.1700697	-0.0012954	99,45 %
Default	0.1690773	-0.0010560	99,26 %