In order to better understant the models presented in this report, it's first necessary to briefly explain how the studied algorithms for the modelled collective operations work.
The two collective operations studied in this work are the **broadcast** operation, where a single process (the root) sends the same message to all processes, and the **reduce** operation, where all processes send their data to a single process (the root) that aggregates them.
OpenMPI architecture is based on a modular approach where different software components, plugged in the library kernel, provide specific implementation features. The `tuned` component, in particular, is responsible for the selection of different algorthms for a particular collective operation. Since an MPI collective operation can be seen as a set of point-to-point tranmission between processes, the algorithms differ one from the other in the way they manage these point-to-point communications. In the case of this study, a total of 4 different algorithms have been considered for both operations.
First of all, the **default** algorithm has been used for both operation as a baseline. In MPI, the default algorithm has no precise implementation since the library itself decides at runtime one of the many possible algorithms available for the specific collective operation requested, based on a decision routine.
The **linear** algorithm, also known as **flat tree** algorithm, has then been taken into account for this study. As depicted in figure 1a, this algorithm translates into a single level tree topology where the root has $P-1$ children and the message is sent (received)[1] to (from) children processes without segmentation. Since *OpenMPI* implements this algorithm using non-blocking send and blocking receive point-to-point communications, the tree topology of this algorithm is known in literature as a *Non-Blocking Fat Tree (NBFT)* [**?**]. The execution time $T^c_{NBFT}(P,m)$ of such algorithm, while transmitting a message of size m to $P-1$ processes throught a channel of communication c, can be upper bounded by the execution time $T^c_{BFT}(P,m)$ of its *Blocking Fat Tree (BFT)* counterpart, which only uses blocking send-receive communications. At the same time, $T^c_{NBFT}(P,m)$ must logically be grater or equal to the time of a single point-to-point communication, hence the following inequality holds:

$$T^c_{p2p}(m) \leq T^c_{NBFT}(P,m) \leq T^c_{BFT}(P,m) = (P-1)\cdot T^c_{p2p}(m) \tag{1}$$

where $T^c_{p2p}(m)$ is the time of a single point-to-point communication of size m and T^c_{p2p} is the time of a single point-to-point communication.
The third algorithm considered has been the **chain** algorithm. In this case, each node of the topology has one child, therefore the message is split in n_s segments of size m_s and transmission of segments continues in a pipeline where process i sends (receives) segments to (from) process $i+1$. Therefore, this algorithm describes a chain tree of height $P-1$, that can be completed in $P+n_s-2$ sequential stages. Since each stage consists of multiple concurrent NBFTs of $P=2$ nodes, as shown in figure 1b, the overall execution time of the algorithm will be equal to the sum of the maximum times $T^{c_i}_{NBFT}(2,m_s)$ of each i^{th} stage.

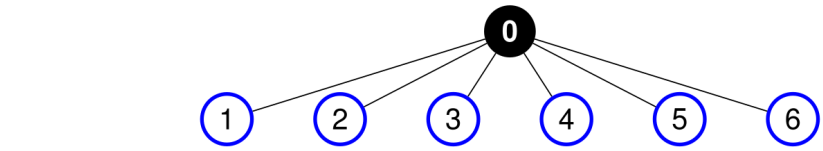
$$T_{chain}(P,m,n_s) = \sum_{i=1}^{P+n_s-2} \max_{j_i} T^{c_{j_i}}_{NBFT}(2,m_s) \tag{2}$$

Where c_{j_i} is the communication channel of the j^{th} NBFT running at the i^{th} stage.
The last algorithm studied is the **binary tree** algorithm (figure **??**c). As the name suggests, this algorithm is represented by a binary tree where each node i exchanges segmented messages with two children $2i$ and $2i+1$. Assuming for simplicity that the tree is complete, then its height will be $\lfloor \log_2(P) \rfloor$ and the algorithm will be completed in $\lfloor \log_2(P) \rfloor + n_s - 1$ stages, each cosisting of multiple NBFT of either 2 or 3 nodes. Therefore, in this case:

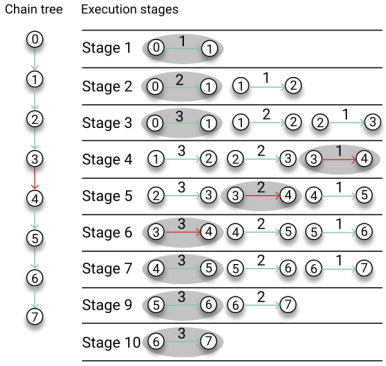
$$T_{binary}(P,m,n_s) = \sum_{i=1}^{\lfloor \log_2(P) \rfloor + n_s - 1} \max_{j_i} T^{c_{j_i}}_{NBFT}(P_i,m_s) \tag{3}$$

Where the number of NBFTs changes at each stage, starting from 1 from stage 1 and reaching a maximum of $2^{\lfloor \log_2(P) \rfloor}$ in the middle stages.

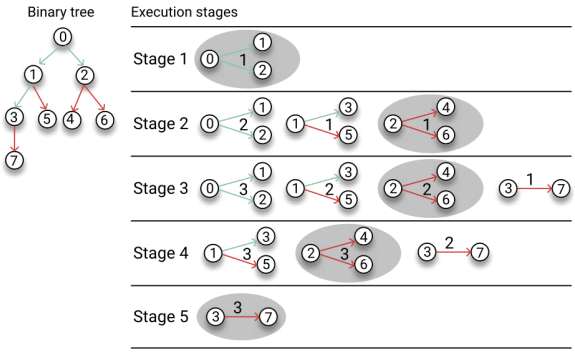
[1] This refers to the broadcast case, between brackets the analogous for the reduce one.



(a) Linear algorithm



(b) Chain algorithm



(c) Binary tree algorithm

Figure 1: Schematic representation of (a) linear, (b) chain and (c) binary tree algorithms.

Equation 1 allows to approximate the execution time of a NBFT as follows:

$$T^c_{NBFT}(P,m) \approx \gamma^c(P,m) \cdot T^c_{p2p}(m) \tag{4}$$

where $\gamma^c(P,m)$ is a *parallelization factor*, representing the increase in latency of $P-1$ cocurrent point-to-point communications in channel c, originating from the same root and transmitting a message of size m, with respect to a single point-to-point communication. Starting from this approximation, is therefore possible to build a model for the prediction of collective operations latencies based on single point-to-point communications latencies as it'll be explained in the next sections. In fact, the ability of predicting the latency of an arbitrarily large NBFT makes also possible to extend the model for predicting latencies of the previously introduced algorithms thanks to equations 2 and 3.