A MultiTree Example and Comparison with Double Binary Tree

Jiayi Huang, Pritam Majumder, Sungkeun Kim, Abdullah Muzahid, Ki Hwan Yum, Eun Jung Kim

***Abstract*—In this document, we illustrate the MultiTree all-reduce tree schedule construction by showing a more complicated example using (3×3) torus network. Then we contrast the MultiTree and double binary tree algorithm by visualizing their all-reduce communication steps.**

I. MultiTree All-Reduce for 3×3 Torus Network

Fig. 1 shows the MultiTree [1] all-reduce tree schedule construction for a 3×3 torus network, which shows the link allocation sequences for each time step to construct the spanning trees. In the example, node `n` in tree `T` is denoted as (`T-n`) and label `i` of an edge is the allocation sequence of that link to connect two nodes in a tree. The algorithm constructs 9 spanning trees for a 9-node network by starting with the 9 tree roots as initial states, as described in Fig. 1b. Then, a topology graph is used to allocate the links to connect the remaining nodes of all the trees, which take turns to get a link from the topology graph. When there is no available link to connect remaining nodes for any of the trees, a new topology graph is used to schedule for the next communication time step. In total, the trees can finish each phase (reduce-scatter or all-gather) in 3 time steps.

A. Time step 1

Fig. 1a–1e show the link allocation for time step 1, where Fig. 1a shows the topology graph and Fig. 1c–1e depict the partial tree built for this time step. The edge label in the trees denotes the link allocation sequence. For example, edge labeled with `0` that connects (`0-0`) to (`0-6`) in Fig. 1c (node `0` to node `6` in tree `0`) denotes that it is the first link that allocated from the topology graph (corresponding to the edge labeled with `0` in Fig. 1a). Then this edge is removed from the topology graph in Fig. 1a. Similarly, the next allocated link is the edge labeled with `1` (in both Fig. 1a and Fig. 1c), which connects (`1-1`) to (`1-7`) (node `1` to node `7` in tree `1`). When the topology graph runs out of links as shown on the right in Fig. 1a, all the links allocated in this time step can be used for scheduling communication concurrently. Then a new topology graph is used to schedule communication for the next time step.

B. Time step 2

After the topology graph in Fig. 1a runs out of links for time step 1, a new topology graph in Fig. 1f is used for link allocation for time step 2. Fig. 1g–1i show the partial trees after link allocation at time step 2, with the newly added edges highlighted. Since the remaining edges in the right topology graph in Fig. 1a cannot connect any remaining nodes of any tree, a new topology graph is used for the next time step.

C. Time step 3

Similarly, a new topology graph on the left in Fig. 1j is used to connect the remaining nodes to form the spanning trees. Fig. 1k–1m show the final complete trees with the newly added edges highlighted.

After tree construction, the trees are processed to generate reduce and broadcast schedules for the reduce-scatter and all-gather phases, as shown in Fig. 2. All the nodes participate in zero or one of the trees at a particular time step, interleaving the usage of the link bandwidth globally to achieve low to zero contention even with large data size. All the trees spend the first three time steps on reduce for reduce-scatter, and the second three time steps on broadcast for all-gather.

II. Double Binary Tree for a 9-node Network

We implemented double binary tree algorithm to generate two trees with fast coloring for 9-node all-reduce [2], as shown in the upper-right corner in Fig. 3. The incoming edge of a node for broadcast (outgoing edge for reduce) is colored as black or red. The node also has the same color as the incoming edge for clarity. The remaining plots in Fig. 3 show the pipelined double binary tree for reduce phase using the two binary trees. The all-reduce data are partintioned into 9 chunks in a 9-node network for pipelining, where tree 0 and tree 1 handle data chunks {0, 2, 4, 6, 8} and {1, 3, 5, 7}, respectively. (`D-n`) denotes the node number in the tree for data chunk `D` and edge color denotes the time for communication. The odd time steps are used for communication colored with red while the even time steps are used for communication colored with black, and the white node denotes only receiving but not sending data.

The rationale behind double binary tree is that the leaf nodes in a reduction tree only send data, so they can also be the internal nodes in another reduction tree to receive data. By proper pipelining with coloring, the communications can be scheduled to be faster compared to a single tree. In this example, data chunks 0 and 1 can be reduced using trees 0 and 1 as shown in the two left-most trees in Fig. 3. At time step 3, the leaf nodes of trees 0 and 1 have finished communication for data chunks 0 and 1, they can be reused to send new data chunks 1 and 2, in a pipeline fashion. Other remaining data chunks follow the same manner. In total, the reduce phase needs 15 time steps, where each communicated data between two nodes in a step is the same as MultiTree.

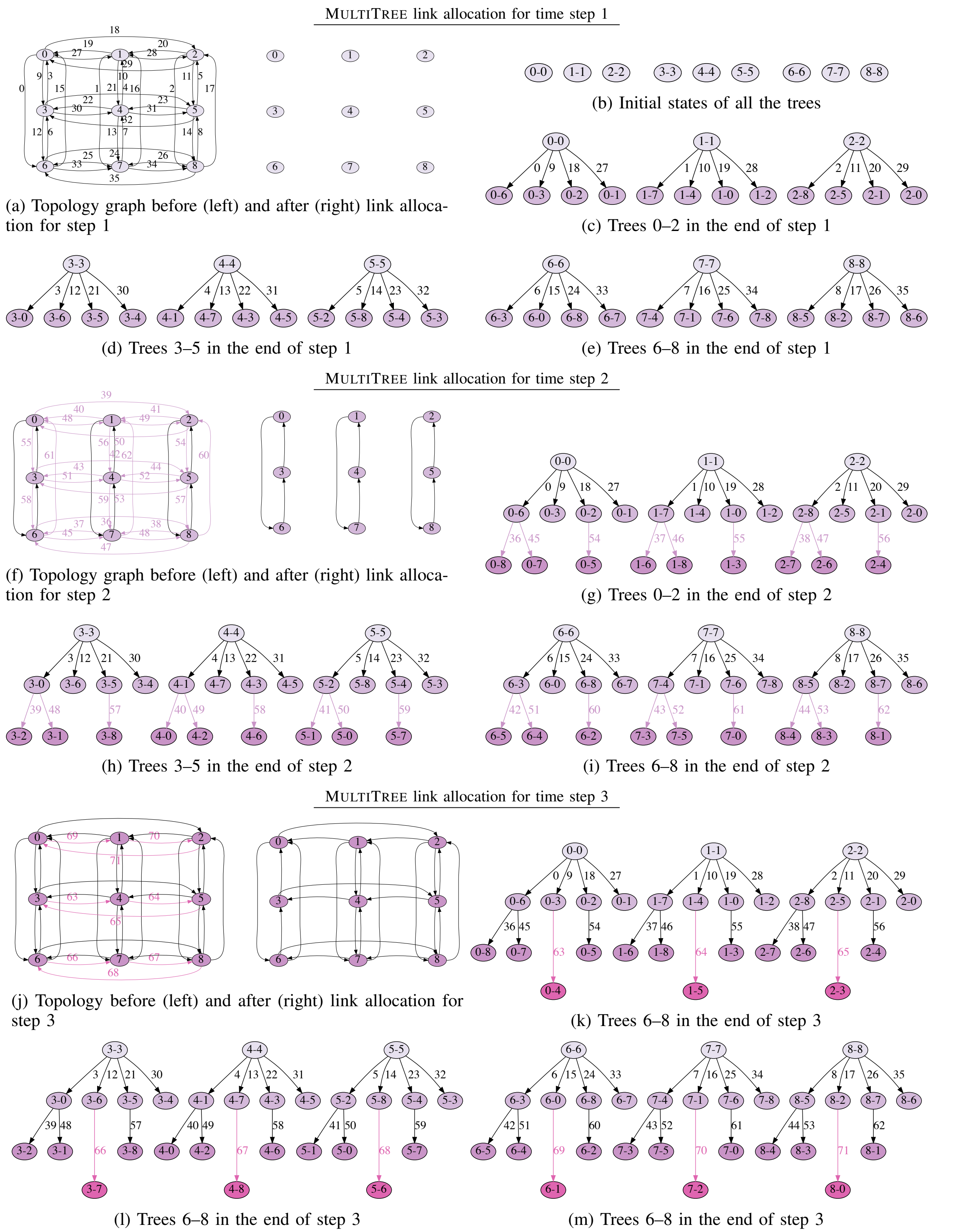
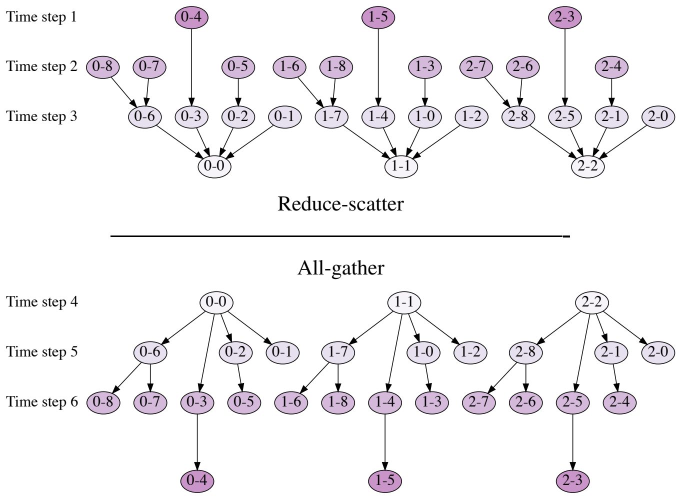
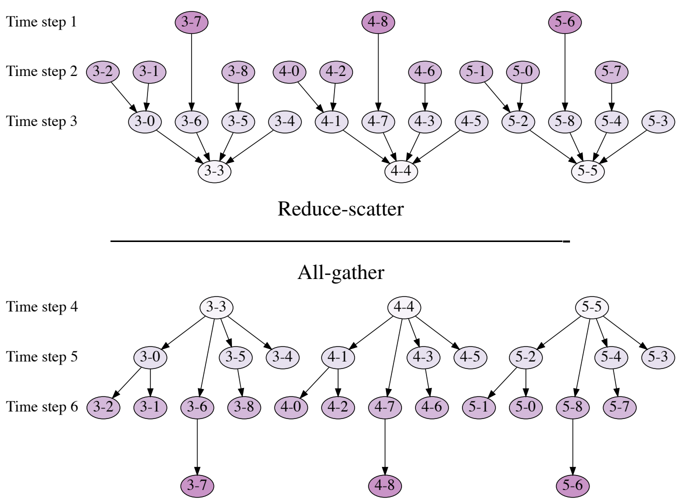


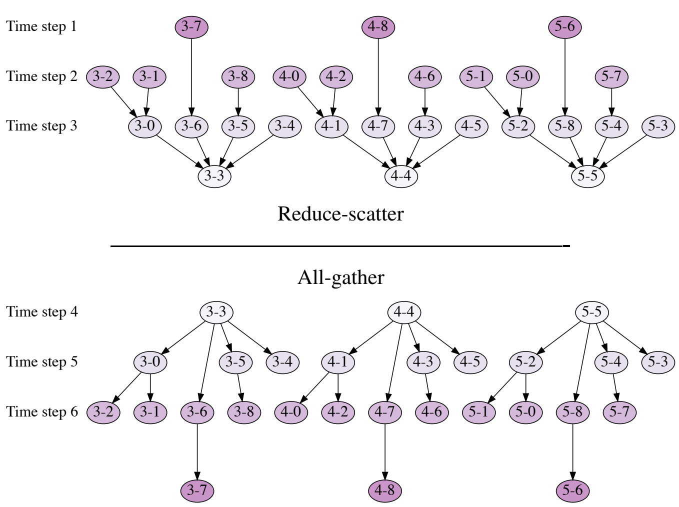
Fig. 1: MULTITREE construction for all-reduce of a (3×3) torus network. Node `n` in tree `T` is denoted as (`T-n`) and label `i` of an edge is the allocation sequence of that link to connect two nodes in a tree. The subplots show the link allocation for time step 1 (a)–(e), time step 2 (f)–(i), and time step 3 (j)–(m).



(a) Trees 0–2 all-reduce



(b) Trees 3–5 all-reduce



(c) Trees 3–5 all-reduce

Fig. 2: MULTITREE All-Reduce (reduce-scatter and all-gather) time steps for trees 0–2 (a), trees 3–5 (b) and trees 6–8 (c). Note that all the trees spend the first three time steps on reduce for reduce-scatter, and the second three time steps on broadcast for all-gather.

III. MULTITREE VERSUS DOUBLE BINARY TREE

As examples in Fig. 2 and Fig. 3 show, in a 3×3 torus network, MULTITREE can finish all-reduce (reduce-scatter and all-gather) in 6 time steps while double binary tree finishes reduce phase in **15** time steps. Although the broadcast phase of double binary tree can also be pipelined with the reduce phase, the time spent on reduce phase in double binary tree is sufficient to run MULTITREE all-reduce twice. Even though without pipelining for small message size, the reduce phase in double binary tree needs 8 time steps, which is larger than the number of time steps in MULTITREE for the whole all-reduce operation.

The main benefit of MULTITREE is its topology awareness in design, which leverages the topology to better coordinate the communications and construct multiple trees instead of just two. Moreover, the topology-aware design reduces the communication inside the network due to the fact that the nodes connected by an edge in MULTITREE are neighbors in the network, making it friendly to any network topology. On the contrary, it is non-trivial or even impossible to map the double binary tree to have the same feature depending on the network topology. This can make communication between two nodes in the two trees actually cross multiple nodes in the network, leading to high latency even with small message [3]. In addition, double binary tree is designed to utilize the network interface bandwidth at the end nodes while MULTITREE further co-design with the network to better utilize both end-node and network bandwidth.

In summary, double binary tree is designed for small to medium data size to achieve latency improvement while MULTITREE targets for both latency and bandwidth for communication with small to large data size. And double binary tree works well on network similar to all-to-all topology while MULTITREE applies well to various network topologies.

REFERENCES

[1] J. Huang, P. Majumder, S. Kim, A. Muzahid, K. H. Yum, and E. J. Kim, “Communication Algorithm-Architecture Co-Design for Distributed Deep Learning,” in *Proceedings of the 48th International Symposium on Computer Architecture (ISCA-48)*, June 2021, pp. 181–194.

[2] P. Sanders, J. Speck, and J. L. Träff, “Two-tree Algorithms for Full Bandwidth Broadcast, Reduction and Scan,” *Parallel Computing*, vol. 35, no. 12, pp. 581–594, 2009.

[3] S. Shi, Z. Tang, X. Chu, C. Liu, W. Wang, and B. Li, “Communication-Efficient Distributed Deep Learning: Survey, Evaluation, and Challenges,” *arXiv preprint arXiv:2005.13247*, 2020.

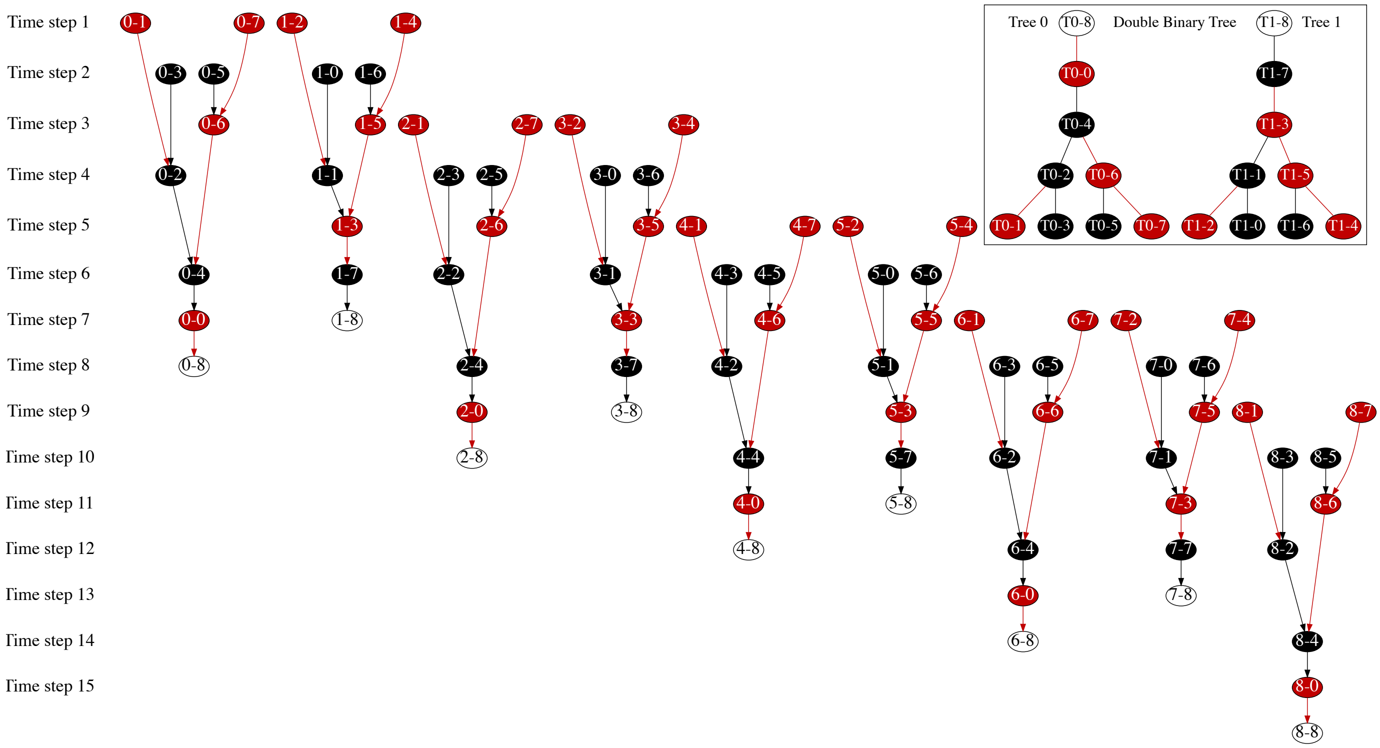


Fig. 3: Pipelined double binary tree for the *reduce* phase using the colored trees 0 and 1 in the upper-right corner, where a node's incoming edge for broadcast (outgoing edge for reduce) is colored to black or red. The node is also colored to be same of the incoming edge for clarity. The all-reduce data are partitioned into 9 chunks (9 nodes) for pipelining, where tree 0 and tree 1 handle data chunks {0, 2, 4, 6, 8} and {1, 3, 5, 7}, respectively. `D-n` denotes the node number in the tree for data chunk `D` and edge color denotes the time for communication. The odd time steps are used for communication colored with red while the even time steps are used for communication colored with black, and the white node denotes only receiving but not sending data. Each communicated data between two nodes in a step is the same as MULTITREE.