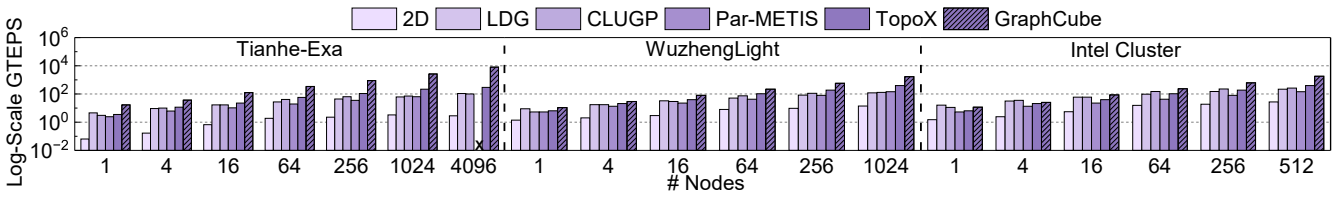

Figure 1. SSSP performance on our evaluation platforms.

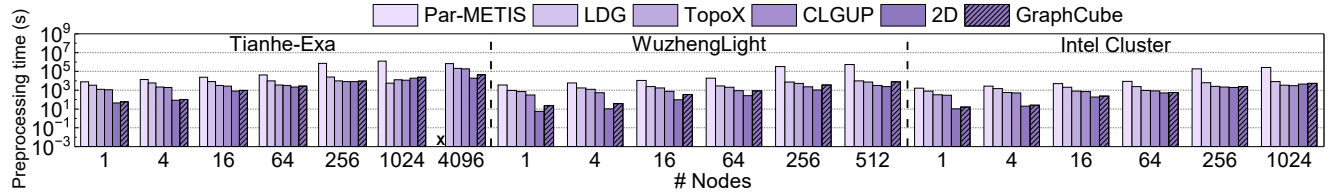

Figure 2. SSSP preprocessing overhead (***lower-is-better***).

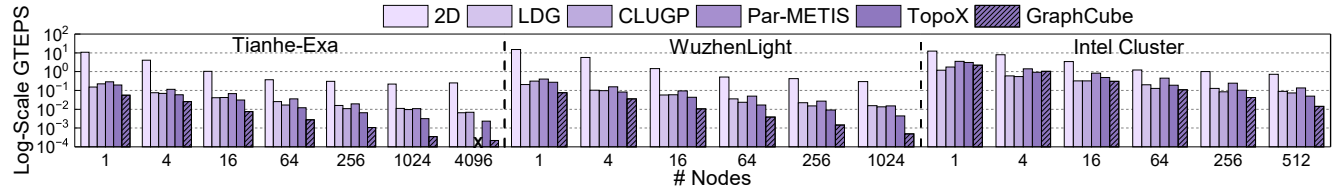

Figure 3. CC performance on our evaluation platforms.

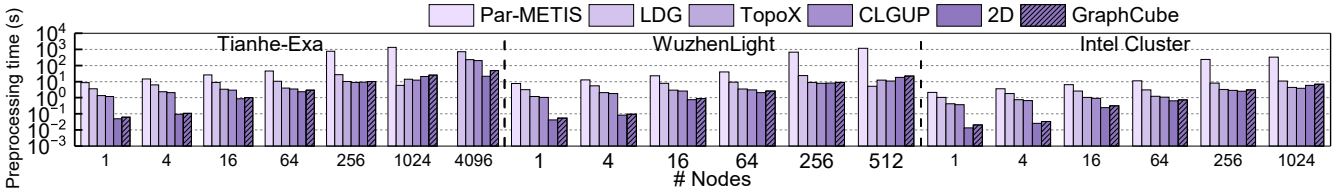

Figure 4. CC preprocessing overhead (***lower-is-better***).

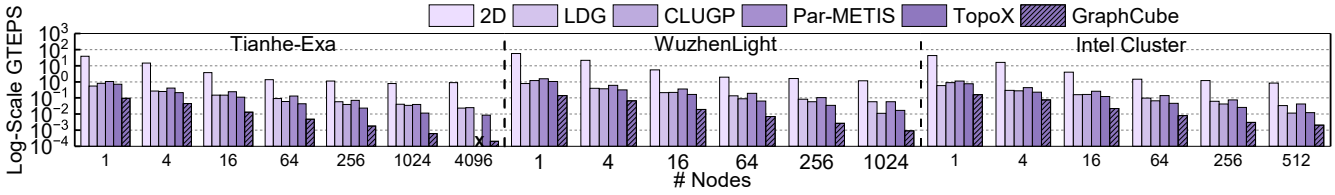

Figure 5. PR performance on our evaluation platforms.

Supplemental Material

S.1 Evaluation on SSSP, CC, PR and CDLP

The evaluation presented in our main paper compares GraphCube against the baseline methods on BFS. Here, we give results for other graph operations: single-source shortest paths (SSSP) in Figures 1 and 2, connected component detection (CC) in Figures 3 and 4, page ranking (PR) in Figures 5 and 6, and community detection with label propagation (CDLP) in Figures 7 and 8. These results, combined with the BFS results in the paper, show that GraphCube consistently outperform the baseline methods when using 16 or more computing nodes by better communication overhead across the blade level.

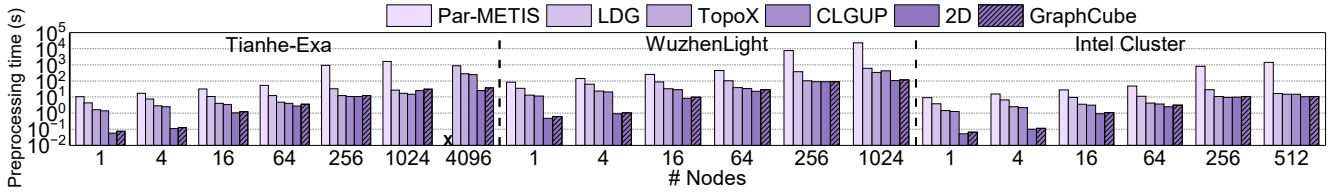

Figure 6. PR preprocessing overhead (***lower-is-better***).

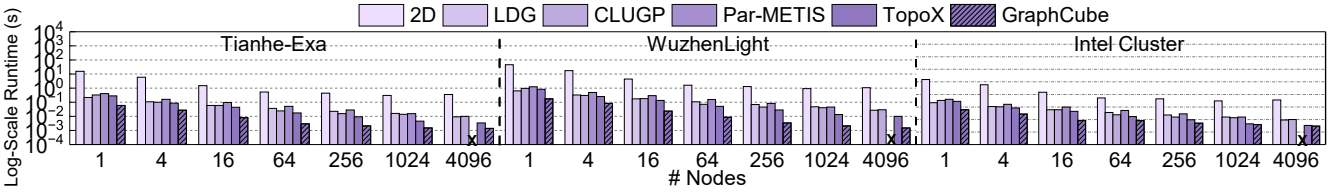

Figure 7. CDLP performance on our evaluation platforms.

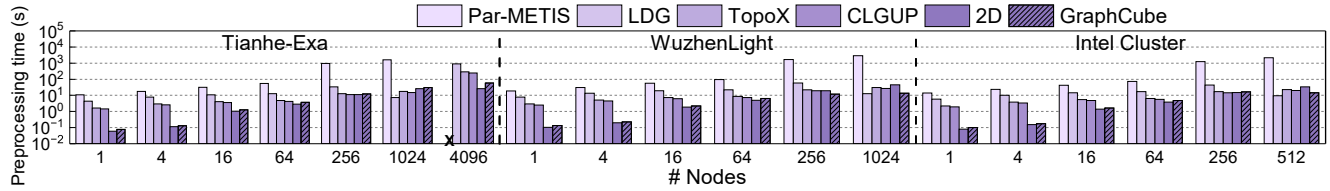

Figure 8. CDLP preprocessing overhead (***lower-is-better***).

S.2 Complexity Analysis for Graph Partitioning

In this section, we present an analysis of the time complexity of our partitioning algorithm, described in Section 5 of the main paper. Our algorithm operates on a sorted graph and takes advantage of the properties of the sorted graph. It consists of two main steps: vertex clustering and distribution. In this analysis, we focus on the time complexity of each step. Furthermore, the storage overhead of our algorithm remains constant throughout the entire graph partitioning process. Note that many graph algorithms require sorting the graph as a preprocessing step, and our algorithm leverages this property.

Time complexity of graph clustering. The first step of our partitioning algorithm is to group vertices into clusters $\mathfrak{C}$ according to their distance to a given high-degree degree. The overhead of vertex clustering is dominated by the time spent visiting each row of the graph adjacency matrix to collect the edge-degree information of each vertex. Therefore, the time of graph clustering is $\sum_{i=0}^{v_i \in \mathfrak{C}_i} v_i * \theta$, where θ is the average latency of visiting a row (i.e., a vertex) of the graph adjacency matrix. As such, the time spent on graph clustering is close to visiting all vertices once, with a time complexity of O(N), where N is the number of vertices.

Time complexity of graph partitioning. Assume the average communication latency within a domain is d, i.e., the intra-domain latency is $\hat{d}$. See also Sec. 4.4. Since we attempt to allocate every vertex (i.e., v_i) within a cluster, $\mathfrak{C}_i$, to the same domain, the time spent on graph partitioning is $\sum_{i=0}^{v_i \in \mathfrak{C}_i} v_i * \hat{d}$. Like graph clustering, the time spent on graph partitioning is close to visiting each vertex once, with a time complexity of O (N), where N is the number of graph vertices.

Space complexity. The memory footprint of GraphCube results from storing the vertex clusters, $\mathfrak{C}$. Hence, its space complexity is $\sum_{i=0}^{v_i \in \mathfrak{C}_i} v_i = \mathrm{O}(N)$.

Putting together, the time and space complexities of GraphCube's graph partitioning algorithm are $2 * \mathrm{O}(N) \approx \mathrm{O}(N)$ (time complexity of graph clustering plus graph partitioning) and $\mathrm{O}(N)$, respectively, where N is the number of graph vertices. Such complexity is in line with the $\mathrm{O}(N)$ complexity of most graph partitioning algorithms based on a sorted graph and lower than the $\mathrm{O}(N^2)$ complexity of methods performed on unsorted graphs [1, 7, 9]. Even when including the overhead for graph

sorting, which is required by many graph computation algorithms themselves, our time complexity is $O(logN) + O(N)$, which is still lower than $O(N^2)$.

S.3 Discussions and Future Work

Naturally, there is room for further work, and we discuss a few points here.

GPU optimization. GraphCube targets CPU executions. An interesting future direction would be porting our optimization to a CPU-GPU mixed system, where graph partitioning needs to consider the communications latency between the CPU and GPU memory and their computation capability. Balancing the communications across the interconnection hierarchy will still be important in such a setting, and GraphCube can benefit from GPU-specific optimization [5, 8].

Fault tolerance. Our graph partitioning algorithm does not explicitly consider fault tolerance. With our current implementation, fault tolerance is achieved through the classical techniques of replication and restarting crashing MPI nodes. Therefore, distribute fault tolerance techniques [2, 4] are orthogonal to our approach.

Ring-based interconnection topology. Our routing and graph partitioning algorithms target the commonly used tree-based interconnection topology in HPC systems. Although a ring-based topology is also used in some data centers [6, 10], it is rarely used in large-scale HPC systems due to the potential for performance degradation when messages traverse multiple nodes to reach their destination. Our routing optimization (Section 4.1) may not apply to ring-based topology, but our node-level optimization (Section 6) remains applicable. Furthermore, a ring-based topology can simplify our analytical models, as the partitioning algorithm only needs to consider two possible data transmission directions.

Auto-tuning kernels. We have showcased how pGAS can be combined with vectorization to improve performance (Sec. 6.2). It would be interesting to integrate our techniques auto-tuning methods [3, 11] to optimize low-level kernel implementation.

References

[1] Baruch Awerbuch and R Gallager. 1987. A new distributed algorithm to find breadth first search trees. *IEEE Transactions on information theory* 33, 3 (1987), 315–322.

[2] Cristóbal Camarero, Carmen Martínez, and Ramón Beivide. 2017. Random folded Clos topologies for datacenter networks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 193–204.

[3] Ravikumar V Chakaravarthy, Hua Jiang, Raghav Chakravarthy, and Siddharth Das. 2022. Auto-tuning of AI/ML Graphs for Optimal Performance in a Heterogenous Processor System. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 215–222.

[4] Poulami Das, Christopher A Pattison, Srilatha Manne, Douglas M Carmean, Krysta M Svore, Moinuddin Qureshi, and Nicolas Delfosse. [n.d.]. AFS: Accurate, Fast, and Scalable Error-Decoding for Fault-Tolerant Quantum Computers. ([n. d.]).

[5] Zhihao Jia, Yongkee Kwon, Galen Shipman, Pat McCormick, Mattan Erez, and Alex Aiken. 2017. A distributed multi-gpu system for fast graph processing. *Proceedings of the VLDB Endowment* 11, 3 (2017), 297–310.

[6] Hanjoon Kim, Gwangsun Kim, Seungryoul Maeng, Hwasoo Yeo, and John Kim. 2014. Transportation-network-inspired network-on-chip. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 332–343.

[7] Lijuan Luo, Martin Wong, and Wen-mei Hwu. 2010. An effective GPU implementation of breadth-first search. In *Proceedings of the 47th design automation conference*. 52–55.

[8] Duane Merrill, Michael Garland, and Andrew Grimshaw. 2015. High-performance and scalable GPU graph traversal. *ACM Transactions on Parallel Computing (TOPC)* 1, 2 (2015), 1–30.

[9] Maharshi J Pathak, Ronit L Patel, and Sonal P Rami. 2018. Comparative analysis of search algorithms. *International Journal of Computer Applications* 179, 50 (2018), 40–43.

[10] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2013. Integrating microsecond circuit switching into the data center. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 447–458.

[11] Richard Schoonhoven, Ben van Werkhoven, and K Joost Batenburg. 2022. Benchmarking optimization algorithms for auto-tuning GPU kernels. *IEEE Transactions on Evolutionary Computation* (2022).