YH-Light: Yielding Hierarchy-aware Partitioner for Large-scale Graph Processing

Xinbiao Gan
National University of Defense technology
Changsha, China
xinbiaogan@nudt.edu.cn

Tiejun Li
National University of Defense Technology
Changsha, China
litiejun@nudt.edu.cn

Chunye Gong
National University of Defense Technology
Changsha, China
gongchunye@nudt.edu.cn

Jie Liu
National University Of Defense Technology
Changsha, China
liujie@nudt.edu.cn

Kai Lu
National University of Defense Technology
Changsha, China
lukai@nudt.edu.cn

Abstract

Large-scale graph tasks often have to be conducted in parallel on partitioned graphs. However, current partitioning methods, designed for smaller-scale clusters with a limited number of computing nodes, struggle to scale effectively due to their inability to handle messages transferred through traditional partitioning grids. We present YH-Light, an hierarchy-aware partitioning engine on Tianhe supercomputers to minimize communication. The key idea of YH-Light is to take advantage of the hierarchical communication topology to perform a graph partition based on communication hierarchies, where scattered messages are (i) clustered with hub vertices according to the organization of the computing nodes and (ii) grouped and then exchanged messages according to hierarchical communication domains. We demonstrate YH-Light's effectiveness with synthetic benchmarks and real-world graphs. In particular, the YH-Light-based Graph 500 tests on the Tianhe supercomputer outperform the leading systems in the latest Graph 500 list. Furthermore, YH-Light significantly advances graph processing, surpassing the current state-of-the-art graph partitioning engines and graph systems by orders of magnitude.

CCS Concepts

• **Computing methodologies → Distributed algorithms**; Massively parallel algorithms.

Keywords

Hierarchy-aware, graph partitioning, communication domain, graph processing, Graph500

ACM Reference Format:
Xinbiao Gan, Tiejun Li, Chunye Gong, Jie Liu, and Kai Lu. 2025. YH-Light: Yielding Hierarchy-aware Partitioner for Large-scale Graph Processing. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3721145.3725740

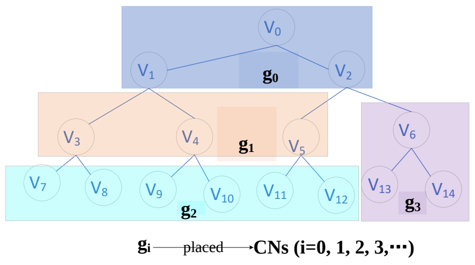
1 Introduction

The development of a supercomputer has always been a strategic goal for many countries [42]. Currently, exascale supercomputers pose severe efficiency and scalability challenges. Facing the challenges of the exascale computing and unprecedented large model training. It is vital for next-generation supercomputers to find appropriate applications with high social and economic benefit. It has been widely accepted that graph computation is a promising killer application for supercomputers.

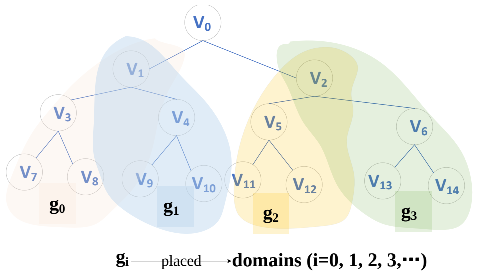
As the graph scale (i.e. the number of vertices and edges) increases explosively, the required computation resources also increase. For example, the Sogou graph [28, 46], comprising 271.8 billion vertices and 12.3 trillion edges, is processed using millions of cores on the TaihuLight supercomputer [28]. Similarly, the Kronecker graph [21] on a scale of 43 (a.k.a. Kron-43), with 6.6 trillion vertices and 105.6 trillion edges, is managed by the Fugaku supercomputer, which utilizes 152,064 CNs[1].

Graph 500, a serious approach to complement the Top 500, reflects the ability of supercomputers to deal with large-scale graphs generated from practical data-intensive applications. Although Tianhe supercomputers have led the world-wide competition of supercomputers (ranked No. 1 in the Top 500 list six times), they had been inefficient in large-scale graph processing according to the Graph 500 ranking. This was mainly because (i) typical graph characteristics, such as poor data locality and low memory access would result in a large number of scattered message exchanging within supercomputers, and (ii) the current graph partition policy cannot leverage the advanced architecture features of hierarchical communication domains, that is, a group of CNs connected to the same Routing Nodes (RN) [13, 16]. As we develop the latest generation Tianhe supercomputer, the mismatch between hardware and software co-designs becomes even more challenging.

[1]A CN may include one or multiple CPUs or accelerators[14, 42].



(a) CN-based Graph Partition



(b) RN-based Graph Partition

Figure 1: Different Graph Partitions

The Graph 500 benchmark ranks supercomputers with data-intensive applications [1]. Unlike the Top 500 benchmark, which compares supercomputers using FLOPS (Floating Point Operations Per Second) for computing-intensive applications [39, 42], Graph 500 instead measures graph processing performance using TEPS (Traversed Edges Per Second) or GTEPS (Giga TEPS). The most popular test of Graph500 is the breadth-first search (BFS) evaluation, which can be used as the kernel for many more complex graph algorithms [11, 41], such as Connected Component(CC) and Betweenness Centrality (BC). Most graph processing frameworks, including Ligra [35], Gemini [50], Gluon [7], and TopoX [25], have been optimized for efficient BFS implementation.

Recent advances in large-scale graph processing perform optimally on high-performance computing (HPC) machines with a limited number of available CNs [7, 8, 25, 50]. A key reason for this is the assumption that the communication overhead remains relatively uniform across hierarchical communication domains within HPC systems. This assumption holds for small-sized HPC systems; it is not true for large-scale HPC systems (e.g., supercomputers) involving hundreds or more CNs. There are mainly two types of nodes in large-scale HPC systems, namely CNs and routing nodes (RNs). CNs are responsible for computation that performs graph tasks on graphs, while RNs are mainly responsible for message communication among CNs. To maximize the communication ability of HPC systems, we can organize CNs and RNs into a hierarchical communication topology, that is, a communication tree, where CNs are all leaf nodes and RNs are all branch nodes. Moreover, communication among lower-level nodes is much faster than that among high-level ones.

For large-scale distributed graph processing, substantial communication overhead is the dominant factor in determining performance. Graph partition is the key step for communication distribution. There are many existing graph partitioning methods that can partition a graph into different sub-graphs. When graph algorithms run on HPC systems, we place those subgraphs onto different CNs, resulting in different communication costs. Figure 1 gives an example of two partitions for one graph, which are named CN-based graph partition and RN-based graph partition, respectively. In the CN-based graph partition, the graph vertices v_7 and v_8 are in subgraph g_2, but the graph vertex v_3 is in subgraph g_1. Since v_3 are connected to v_7 and v_8, there exist many information exchanges between v_3, v_7, and v_8 when performing graph algorithms. However, if these subgraphs are on different CNs and not attached to the same domains, then they require a lot of inter-domain communication, which is expensive.

Although graph processing is relatively straightforward, irregular graph partition and distribution are the bottleneck in such scenarios, since mismatching graph partition and target hierarchical communication topology cause huge cross-domain messages exchanging, particularly when numerous cross-domain communications are walking among hundreds of CNs equipped in supercomputers. As such, we present YH-Light to bridge the gap between large-scale graph partitioning and target HPC systems by utilizing hierarchical communication domains.

To validate YH-Light, we conducted a comprehensive evaluation using a variety of graph algorithms, including BFS, DFS-based algorithms, Single Source Shortest Path (SSSP), PageRank (PR), CC, and BC [7, 8, 13, 16, 50], across both synthetic and real-world graphs. The evaluation was performed on two distinct HPC systems with varying scales, utilizing the Tianhe supercomputer and a commercial Intel cluster. Extensive results demonstrate that YH-Light significantly reduces communication overhead and outperforms state-of-the-art graph partitioning methods. Specifically, YH-Light achieves a peak performance of 162,494 GTEPS for BFS, surpassing the fastest BFS systems by a substantial margin. Furthermore, when applied to real-world graphs, YH-Light achieves orders of magnitude superior performance compared to state-of-the-art graph partitioning and graph processing systems.

This paper makes the following contribution:

- It proposes a communication hierarchy-aware graph partitioning approach to improve data locality and reduce cross-domain communication costs.
- It highlights the effectiveness of co-design approaches in optimizing large-scale graph partitioning and hierarchical communication within Tianhe supercomputers.
- Extensive experiments show that YH-Light leads to fast processing time over state-of-the-art graph engines. More specially, YH-Light leads the leaderboard in Graph500 BFS ranking using fewer nodes.

2 Background

2.1 Graph Processing

Graphs represent a fundamental data structure for modeling relationships among discrete entities. Graphs have become a cornerstone in the representation of complex systems, which emerge as a powerful abstraction for efficiently modeling the relationships between objects. Specifically, vertices and edges are used to represent objects and their relationships, respectively. Such graph-based representations find extensive applications across machine learning, data mining, and social network analysis [3, 26, 43, 49], path queries on road networks [31]. Graph processing on a single machine has been well studied [19, 23, 34, 35] and has achieved high computation efficiency. However, the rapidly-growing graph scales have exceeded the memory capacity and processing speed of any individual machine [15, 44]. This has recently driven the study of distributed graph processing systems, including Pregel [30], GraphLab [29], PowerGraph [17], GraphX [18], PowerLyra [6], Gemini [50], and TopoX [47], etc. While reducing per-node computation load, parallel graph processing causes high communication cost. Although existing studies have extensively studied the partitioning problem for reducing communication by various methods like edge-cut [29, 30], vertex-cut [17, 18], hybrid-cut [6], and refactorization [47], the emergence of extremely large graphs (e.g., Sogou graph and Kron-43) consisting of trillions of vertices and edges necessitates the adoption of large-scale distributed systems(e.g., supercomputers) for efficient graph processing. Indeed, many graph processing algorithms (such as BFS, SSSP, PR, CC and BC, etc.) have been working efficiently on supercomputers [12, 13, 16, 50] which achieve significantly higher performance than their distributed HPC counterparts.

However, current distributed graph processing still suffers from unsatisfactory graph partitioning strategies [13, 16, 50]. That is because current graph partitioning solutions hardly match large-scale graph partition and target HPC systems with hierarchical communication domains. To address this problem, we present YH-Light: a partitioning engine based on the hierarchy of communication for large-scale graph processing using the hierarchical communication topology.

2.2 Communication Topology of Large-scale HPC Systems

Recently, massively parallel HPC systems, the underlying network connecting hundreds of CNs, have become popular for handling various graph processing tasks [14, 28, 42, 50]. Large-scale HPC systems often adopt a hierarchical communication topology [27, 42] to link different CNs. Due to the good scalability and the tree-based Mellanox network widely used in HPC systems [13, 16, 37], many large-scale HPC systems have hierarchical communication domains of which the architecture can be modeled as a multilevel domain tree.

As shown in Fig. 2, leaf nodes represent CNs, and inner nodes such as HRM, HNR, CPM and MCU are switching cells for specific communication domains at different communication hierarchies, The sets of computing nodes attached to $\{CMU_1, CMU_2, \ldots, CMU_7\}$ are organized into the lowest level communication domains, eg, $\{CN_1, CN_2, CN_3, CN_4\}$ attached to CMU_1 is a lowest level communication domain, and the lower level communication domains include all CNs equipped in $\{CPM_1, CPM_2, CPM_3, CPM_4\}$. while CNs attached to $\{HNR_1, HNR_2, HNR_3, HNR_4\}$ represent low-level communication domains. All CNs attached to the HRM organized a high-level communication domain. Many large-scale HPC systems

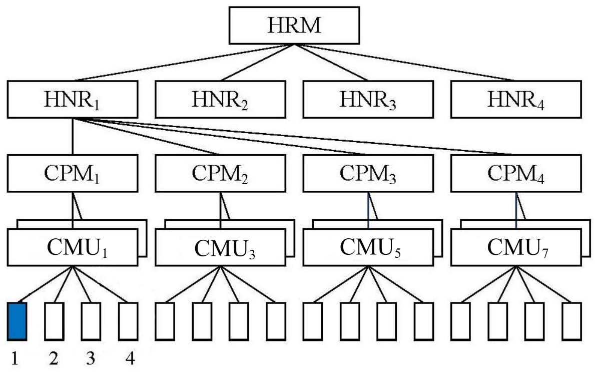


Figure 2: Hierarchical communication tree with 4-level communication domains, where 4 CNs are equipped in a MCU (memory control unit) at the lowest-level domain (i.e. level-0), and 2 CMUs are included in a CPM (computing processor mainboard) at level-1. The level-2 communication domain attached to a HNR (hyper networking router) has 4 CPMs, and 4 HNRs are organized into a four levels communication tree attached to thehyper networking machine (HRM).

have hierarchical communication domains of which the architecture can be modeled as a multilevel domain tree as shown in Fig. 2, a level-i node inside a tree has x_i level-$(i-1)$ child nodes, and a h level domain tree accommodates totally $\prod_{i=1}^{h}(x_i)$ CNs. Given $h = 4$, if we take $x_1 = 4$, $x_2 = 2$, $x_3 = 8$, $x_4 = 16$, then we will have a 1024-node system. In practice, communication latency in lower-level communication domains is significantly faster than in higher ones. The communication latency between CNs for the lowest communication domain can be 1 $\mathcal{U}$[2], while it can increase by almost 10 times when across a higher-level communication domain.

Define $\Theta(CN_x, CN_y)$ as communication latency between two computing nodes CN_x and CN_y, which can vary significantly depending on their locations. For example, consider three CNs, CN_1, CN_5, and CN_9, which are installed in the lowest level CMUs CMU_1, CMU_2, and CMU_3, respectively. When measuring the latency between CN_1 and its adjacent node CN_2 within the same CMU_1, the latency is minimal,i.e., $\Theta(CN_1, CN_2) = 1\mathcal{U}$. In contrast, the latency between CN_1 and CN_5, which are connected through the higher-level switch CPM_1, increases to $\Theta(CN_1, CN_5) = 10\mathcal{U}$. Furthermore, the latency between CN_1 and CN_9, which reside in the same HNR_1 communication domain, can escalate to $\Theta(CN_1, CN_9) = 100\mathcal{U}$. This latency variation is mainly attributed to the hierarchical communication domains, where nodes within the same CMU exhibit lower latency compared to those in higher-level switches or domains. The downside of such a design is that cross-dimension communication will become increasingly inefficient as the number of dimensions grows. To mitigate cross-domain data movement, we present YH-Light to orchestrate graph division and distribution by matching graph partitioning and hierarchical communication domains.

[2] A $\mathcal{U}$ maybe one microsecond, millisecond, or another scale, depending on the target system.

2.3 2D decomposition

A commonly used method for partitioning large-scale graph tasks is the 2D decomposition strategy [5, 13, 16, 45, 50], which divides the graph into small blocks (a.k.a., subgraph). In this approach, a big graph is segmented into "m" blocks in the horizontal direction and "n" blocks in the vertical direction, resulting in "m × n" blocks distributed across "m × n" CNs, with each CN handling a block. Many partitioning solutions are variations of the 2D-decomposition(e.g., 1.5D [4] and XTree [16]), fully utilizing graph properties but often overlooking the communication hierarchies built in the large-scale HPC systems. This can lead to frequent cross-domain data transfers and damage graph processing performance, providing both challenges and opportunities for improving the performance of graph processing. Accordingly, we introduce YH-Light, an advanced partitioning method to minimize cross-domain communication by orchestrating graph division and distributions across different communication hierarchies.

3 Modeling Communication Latency

Generally, given $d\_vertices$ is the communication latency (i.e., networking hops) between any two vertices, let

$$d\_vertices = d\_internode + d\_intranode \tag{1}$$

Here $d\_internode$ and $d\_intranode$ are communication latency within the local domain and across domains traveling between CN, respectively.

Using a quadruple coordination system [13], denoted as $(x_l^i, y_l^i, x_a^i, y_a^i)$ for the position of CN_i within the local and across the communication domain, we recursively determine the coordinates of the CNs and estimate the communication path within the local domain and across domains by factoring in the position of the CNs according to the hierarchical communication topology.

Define V_{ij}^{local} and H_{ij}^{local} to express the communication path (i.e., communication hops) between CN_i and CN_j within the local domain, similarly, V_{ij}^{across} and H_{xy}^{across} to express the communication path across domains.

$$H_{ij}^{local} = |x_l^i - x_l^j| \tag{2}$$

$$V_{ij}^{local} = |y_l^i - y_l^j| \tag{3}$$

$$H_{ij}^{across} = |x_a^i - x_a^j| \tag{4}$$

$$V_{ij}^{across} = |y_a^i - y_a^j| \tag{5}$$

As such, we can approximate $d\_internode$ and $d\_intranode$ by determining the distances between the nodes within each communication domain.

$$d\_internode = \epsilon_v V_{ij}^{local} + \epsilon_h H_{ij}^{local} \tag{6}$$

$$d\_intranode = \epsilon_v V_{ij}^{across} + \epsilon_h H_{ij}^{across} \tag{7}$$

Such that ϵ_h and ϵ_v represent the transmission delay per hop along the horizontal and vertical directions, respectively, at each domain level.

Given a graph $\mathbf{G} = (V, E)$ partitioned and distributed into the CNs of the HPC system with hierarchical communication domains. A partitioning engine aims to minimize communication hops from all vertices. Taking the worst case into account, there is message transfer between any pair of vertex so that the objective can be formulated as follows.

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} \Theta(v_i, v_j), \quad \text{subject to } v_i, v_j \in \mathbf{HirCS}.\text{CNs}. \tag{8}$$

where N is the total number of vertices in $\mathbf{G}$, and **HirCS**.CNs refers to the set of computing nodes belonging to a large-scale HPC system (named **HirCS**). $\Theta(v_i, v_j)$ is the cost of communication of messages between v_i and v_j distributed into CNs equipped in **HirCS**.

Solving Eq. 8 is in NP [10, 20], which can approximate the accumulative communication hops between any two CNs as Eq. 9.

$$\min \sum_{x=1}^{N} \sum_{y=1}^{N} \Theta(N_x, N_y) = d\_internode + d\_intranode \tag{9}$$

where $d\_internode$ and $d\_intranode$ are the inter-domain and intra-domain communication hops through RNs, respectively, Eq. 9 can be further simplified as:

$$\begin{cases} \min \sum_{x=1}^{N} \sum_{y=1}^{N} \Theta(N_x, N_y) = \\ \sum_{l=1}^{h} \left(\sum_{i=1}^{N} \Theta(N_l, N_i) + \sum_{j=1}^{N} \Theta(N_l, N_j) \right) \end{cases} \tag{10}$$

Clearly, h is the lowest domain level where both N_l and N_i / N_j are located. It is easy to see that cross-domain communication takes the majority of accumulative communication costs and dominates the communication overhead in large-scale graph processing in HPC systems according to Eq. 9~10. That is because (i) the inter-domain communication is orders of magnitude higher than that of intra-domain communication, and (ii) there are a large number of inter-domain communication in large-scale graph processing within hierarchical communication domains [13, 16, 28, 40].

To fully exploit advanced communication hierarchies, we present YH-Light to release powerful architectural features with hierarchical communication topology. YH-Light is built on the 2D decomposition, dividing the input graph into subgraphs and assigning each subgraph to a low-level communication domain. With this policy, many cross-domain communication can be translated into local communication within intra-domain communication by carefully orchestrating graph partitioning.

4 YH-Light philosophy

Our newly developed graph partitioning engine aims to improve classical 2D decomposition by considering the hierarchical communication topology and the varying costs across communication hierarchies. The key idea of YH-Light is to partition the graph based on the spatial locality of subgraphs, each subgraph is then assigned within the lowest communication domain, ensuring communication remains within a single subgraph. thereby minimizing cross-domain communication as much as possible.

4.1 Core Tree

Similarly to many graph partitioning strategies [8, 13, 16, 50], we begin by extracting the spatial locality of subgraphs from the input graph. This can be achieved by multiple methods, e.g., subgraph [4], subtree [16], and vertex clustering [13]. Owing to the strong cohesion of MST (Minimal Spanning Tree), in this work, we present a core tree policy based on MST to complete core tree extraction quickly and efficiently. Furthermore, each core tree would be orchestrated into CNs close to each other within the lowest communication domains.

The structure of core tree relies on the properties of the MST obtained during the DFS traversal of the graph, which involves iterating through the vertices and edges, ensuring that each edge selection maintains the acyclic property of the tree while also optimizing for the characteristics specified. The final core tree provides a robust representation of hierarchical communication topology. YH-Light differs from previous work in that it aims to place all vertices of an MST in the same CN close to each other within communication domains.

The core tree algorithm based on DFS is summarized in Algorithm 1. The process begins by extracting vertices and edges from the graph (Lines 1~3), followed by sorting the vertices in descending order of their degrees in parallel (Line 5). Subsequently, multiple MSTs are constructed (Lines 7~9) through DFS traversal of the input graph. A core tree is then formed through the interconnection of these MSTs (Line 10) until the core tree's volume satisfies the advanced features,i.e., **vol-st** that is a tunable hyperparameter that can be optimized through the network bandwidth and the memory capacity of CN.

Algorithm 1: Core-tree algorithm based on DFS

```
   Input: edge.bin
   Output: subtree
1  while $e \in$ edge.bin parallel do
2      vertexset.insert($e.v_1, e.v_2$);
3      build_adjacency($e.v_1, e.v_2$);
4  end
5  hub ← sorted(vertexset) by descending in parallel;
6  $i \leftarrow 0$;
7  foreach $v \in$ hub parallel do
8      if $v.visited = 0$ then
9          MST$_i$ ← dfs($v$);
10         core-tree ← $\bigcap_{j=0}^{i}$ MST$_j$;
           // vol-st is a tuning hyperparameter
11         if core-tree.vol = vol-st then
12             return core-tree;
13         end
14         $i \leftarrow i + 1$;
15     end
16 end
```

Algorithm 2: Hierarchy-Aware Partitioning

```
   Input: Vertex list, V, sorted according to edge degrees in
          descending order, a list of allocated CNs, $\mathbb{N}$
1  $\mathcal{H}$ ←Building communication domain tree ($\mathbb{N}$) and return the levels
     of communication hierarchy;
2  Move isolated vertices from V to $\tilde{\mathbf{V}}$
3  Set all vertices to be unvisited
   // Construct the core tree by calling algoritm 1
4  $\mathfrak{C}$ ← Core-tree($\tilde{\mathbf{V}}$)
5  HierarchicalPartition(V, $\mathfrak{C}$)
6  return SUCCESS
   // Distribute remaining isolated vertices
7  if !empty($\tilde{V}$) then
8      Distribution( $\tilde{V}$, 0)
9  end
10 Function HierarchicalPartition(vertex list V, Core-tree $\mathfrak{C}$)
11     while !empty(V) do
           // dequeue the highest degree vertex
12         $v \leftarrow V.dequeue()$
13         $\tilde{V}$ ← $v$'s (unpartitioned) sorted neighbor list
14         foreach $v_i$ in $\tilde{V}$ && $v_i \notin \mathfrak{C}$ do
15             $\mathfrak{C}_\mathfrak{v} \leftarrow \mathfrak{C} \cup \{v_i\}$
               // $\mathbf{V} \backslash \{v_i\}$ is remving $v_i$ from V
16             HierarchicalPartition($\mathbf{V} \backslash \{v_i\}$, $\mathfrak{C}_\mathfrak{v}$)
17         end
18     end
       // Recursively distribute vertices within $\mathfrak{C}$ into
          hierarchical communication domains, starting
          from the lowest-level communication domain
          (i.e., level 0).
19     while !empty($\mathfrak{C}_v$) do
20         Distribution($\mathfrak{C}_v$, 0)
21     end
22 end
23 Function Distribution($\mathfrak{C}$, domain − level   l)
24     if empty($\mathfrak{C}$) then
25         return
26     end
27     else
28         Distribute vertices in $\mathfrak{C}$ to CNs within the same
             communication domain, by first filling up an CN before
             moving to another within the same communication
             domain
29         Remove all allocated vertices from V and $\mathfrak{C}$
30         if $l \le \mathcal{H}$ then
31             Distribution($\mathfrak{C}$, $l$);
                $\mathfrak{C}$ ← Remove all allocated vertices from $\mathfrak{C}$
32             if !empty($\mathfrak{C}$) then
                   // Distributing to a higher level
                      communication domain
33                 Distribution($\mathfrak{C}$, $l + 1$)
34             end
35         end
36     end
37     return
38 end
```

4.2 Hierarchy-aware Graph Partitioning

Algorithms 2 outlines how to partition and distribute graph into CNs, taking into consideration the graph's spatial locality and hierarchical communication domains.

Given a list of allocated CNs, $\mathbb{N}$, containing node IDs or IP addresses, we create a communication tree by grouping CNs based on their communication domains. This can be accomplished by checking the node's local gateway or querying the HPC resource manager. The resulting communication tree encodes the communication pattern among CNs in the hierarchical communication topology. After constructing the communication tree, we then record the communication tree height to $\mathcal{H}$ (line 1), which indicates multi-level communication hierarchies. For example, there is a hierarchical communication tree with 4-level communication domains, i.e., $\mathcal{H} = 4$, as shown in Figure 2.

As a preprocessing step in the partitioning algorithm (see Algorithm 2), we first remove isolated vertices, which lack connections to other vertices, from the graph's vertex list (**V**), and store them in a separate list $\tilde{\mathbf{V}}$. Following this, we proceed with the main steps of the partitioning algorithm, applying the cohesion of MST to manage how vertices are grouped and ensuring connectivity by calling algorithm 1 (lines 2~4) to build a core tree before calling the partitioning function **HierarchicalPartition** (line 5).

To further reduce cross-domain message exchanging between core-trees, `Partitioning` function (lines 3-6 in Algorithm 2). begins with high-degree vertices from the (**V**) rather than the $\tilde{\mathbf{V}}$ to check whether they are included in a core-tree (lines 11~15) and then recursively call partitioning function **HierarchicalPartition** (lines 16), in which every core-tree will be recursively distributed vertices within $\mathfrak{C}_{\mathfrak{v}}$ into hierarchical communication domains by calling **Distribution** function (lines 19~20).

In the **Distribution** function that is a recursive function, all vertices within a core tree are orchestrated into CNs within the same communication domain, by first filling up a low-level domain(lines 24~31) before moving to another higher-level communication domain(lines 32~35) until all vertices of core trees are assigned to CNs in the lowest-level communication domain.

It is worth noting that current graph partitioning methods typically treat CNs as equal entities in the communication topology, randomly distributing vertices to any available node, which can lead to poor communication efficiency. However,YH-Light addresses this issue taking into account the locality of the subgraph and the varying communication latency within the communication hierarchies of large-scale HPC systems.

5 Evaluation

5.1 Graph Data

We test YH-Light on both synthetic and real-world graph data. Synthetic data are generated using the Graph 500 data generator that takes in two parameters, graph factor (m) and edge factor (n), to produce a graph with 2^m vertices and $n \times 2^m$ edges. For our evaluation, we vary the graph factor between 26 and 41 while using the default edge factor of 16 to create graphs of different sizes to validate YH-Light in various hardware setups. Table 1 lists the synthetic graphs and real-world data used in our evaluation.

Table 1: Synthetic and real-world graphs used in evaluations

G. Scale	Edge factor	#Edges	#Comp. Nodes
real-world web graphs			
26	16	1 billion	1
28	16	4 billions	4
30	16	16 billions	16
32	16	64 billions	64
34	16	256 billions	256
38	16	4 trillions	4,096
clueweb12		42.6 billions	64
enwiki-2022		159 millions	64

Table 2: Hardware systems used in our evaluations

System	Max. #nodes used	CPU	RAM/node	Top-level bandwidth
Tianhe-Exa	79,024	16-core FT-2000 ARMv8 CPU @ 2.2 GHz	16G	200Gbps
Intel Cluster	512	12-core Intel Xeon CPU @ 2.93 GHz	64G	160Gbps

5.2 Hardware Platforms

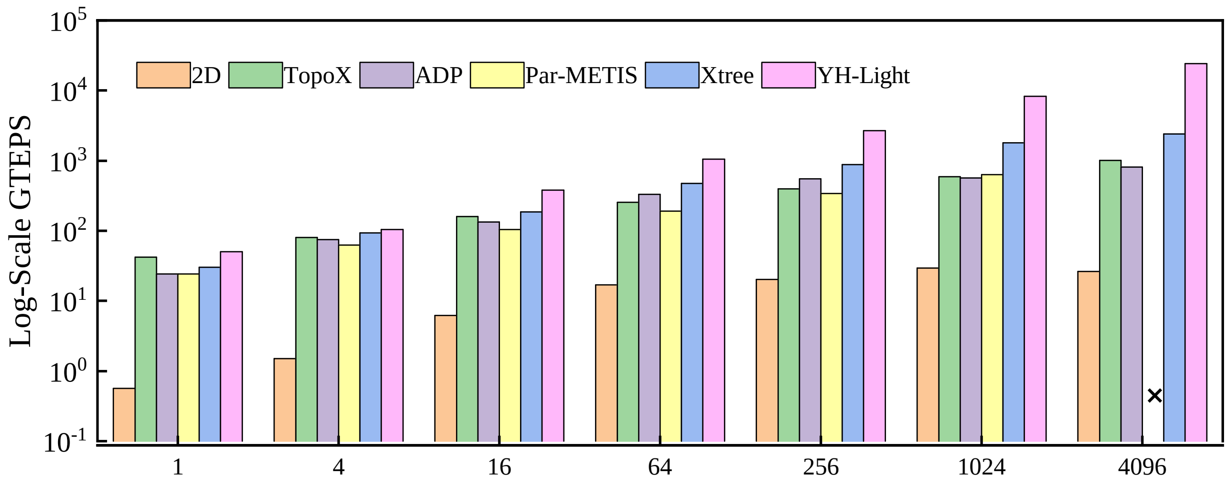
To evaluate the portability of YH-Light, we apply it to two HPC systems with different CPU architectures and interconnection components. Table 2 lists the details of the two HPC systems used in our testing, including the maximum number of computing nodes used in the experiments. Both of evaluating systems run Linux with the Linux kernel 9.3.0. We use MPICH 10.2.0 and libgomp 4.5. We compile the benchmark using GCC 10.2.0 with "-O3" as the compiler option. Note that the default is to perform experiments using the first one, a subset of the next-generation Tianhe supercomputer (Tianhe-Exa) [48] with up to 79024 nodes. The second is a commercial system, that is, a general cluster installed in the national supercomputer center at Changsha [2].

5.3 Competing Baselines

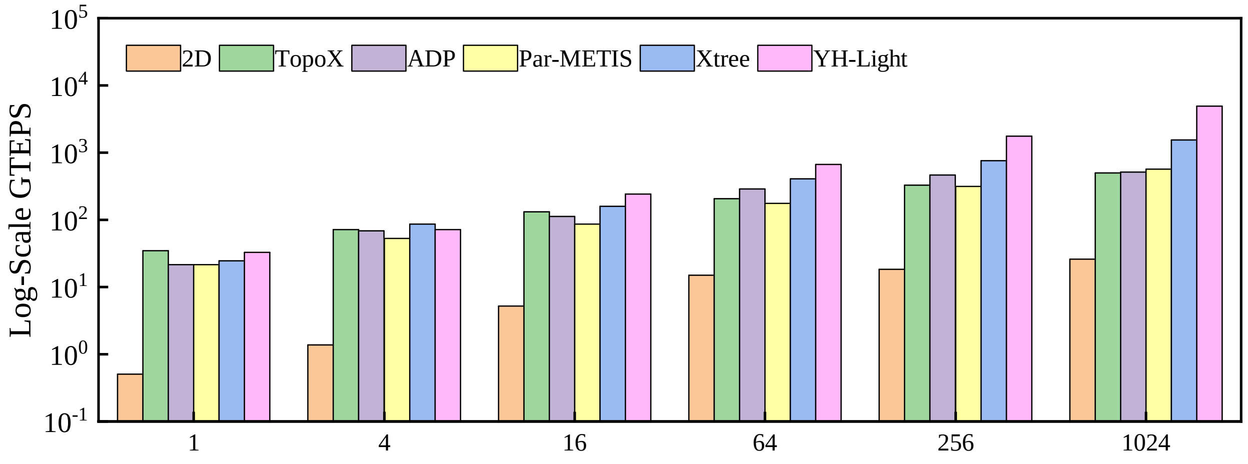
We compare YH-Light with five representative graph partitioning strategies including 2D decomposition [14, 16, 28, 45, 50], XTree [16], Par-Metis (the parallel version of METIS [22]), ADP [9] and TopoX [24]. We note that Par-Metis is widely used in commercial graph processing applications [22, 32], while ADP and TopoX are state-of-the-art graph partitioning schemes that implement optimized versions of 2D decomposition partitioning. We also compare YH-Light against Gemini [50] and GraphScope [8], two state-of-the-art graph processing engines.

5.4 Performance Report

Following the Graph 500 test methodology, we use GTEPS (Giga Traversed Edge Per Second), a higher-is-better metric, to measure graph processing throughput for each test case. For BFS, we performed a method ten times and obtained the geometric mean of GTEPS to compute the final performance result by calculating the harmonic mean [33] across 64 source vertices. Furthermore, we execute graph operators twenty times and compute the average time to validate YH-Light.



(a) Tianhe-Exa



(b) Intel Cluster

Figure 3: Overall BFS performance with various partitions for varying nodes.

5.5 Benchmarking Graph 500

In this subsection, we evaluate the overall performance improvement of YH-Light in Tianhe-Exa over 2D decomposition that is widely used in large-scale HPC systems [13, 15, 16, 16, 28, 40, 50]. The increase in the number of increases from 16 to 1024, according to YH-Light and 2D for BFS, the performance of YH-Light significantly outperforms 2D for BFS in Tianhe-Exa (by 12.36× for 1024 nodes); Note that the performance of Tianhe-Exa based on YH-Light achieves 2103.13 GTEPS with 1024 available CNs, outperforming the 2061.48 GTEPS of Tianhe-2 (a.k.a. MilkyWay-2 using 8192 CNs. The primary reason is because hierarchy-aware YH-Light effectively reduces communication cost compared to its 2D counterpart. We also evaluate SSSP performance using Tianhe-Exa based on YH-Light and 2D, respectively. The results are similar to those of the BFS tests.

Finally, YH-Light has been deployed in Tianhe-exa to compete with BFS and SSSP for the latest Graph 500 rankings[3], respectively. Although the Fugaku and Wuhan Supercomputer are the fastest supercomputers (except Tianhe-Exa) for BFS and SSSP, respectively, YH-Light-based Tianhe-exa significantly exceeds them.

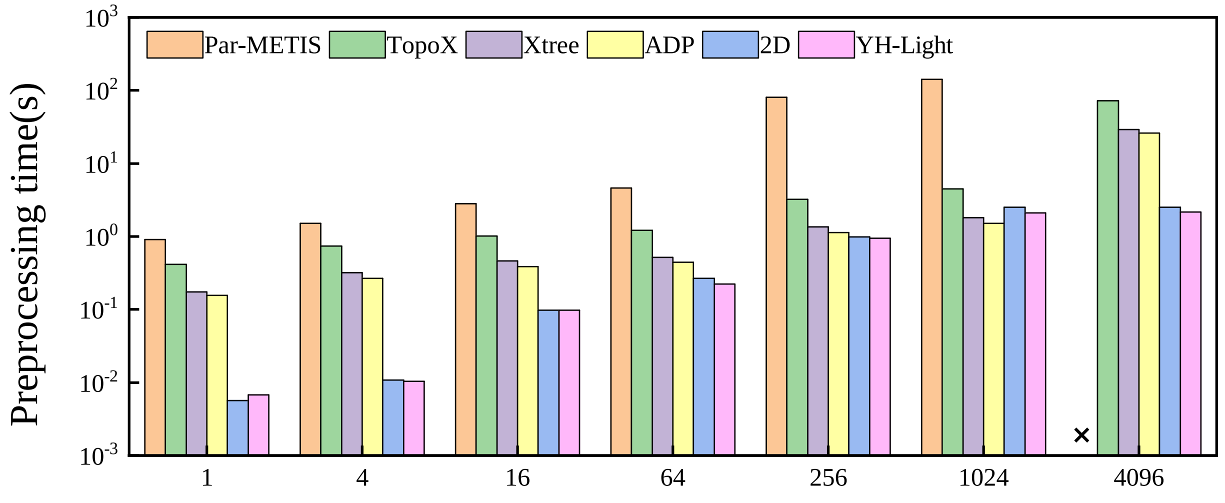
[3]Tianhe does not attend the Graph500 BFS ranking for non-technical reasons [38].

5.6 YH-Light vs. Popular Graph Partitioning

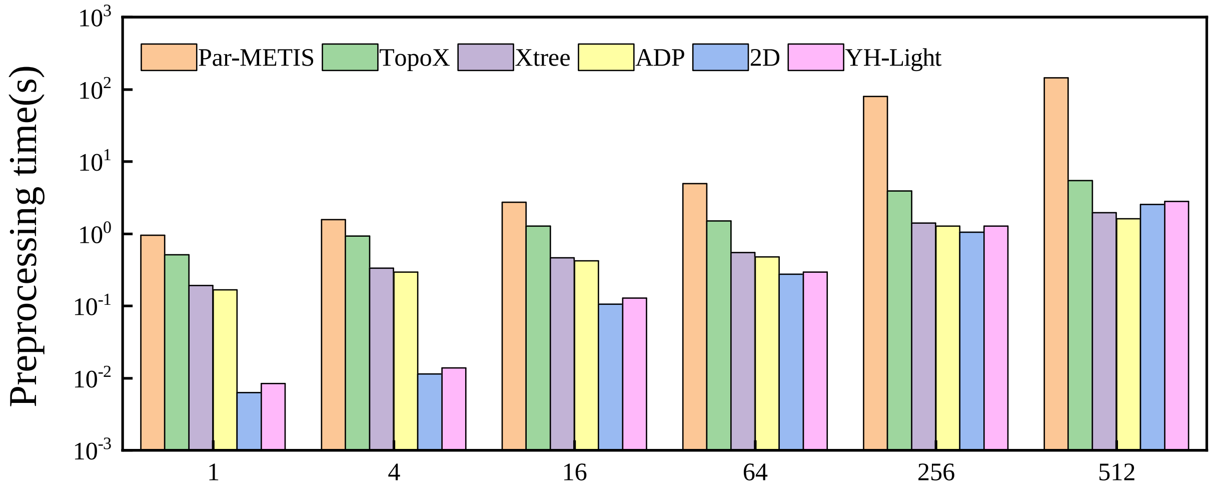
Figure 3 compares YH-Light with state-of-the-art competing baselines on the main evaluation system for Graph 500 BFS. In this experiment, we use up to 4096 computing nodes because all competing methods running beyond this scale lead to run-time errors owing to a communication buffer overflow. Note that Par-METIS could not execute some test cases on more than 1024 Tian-Exa computing nodes (marked as **X**).

We first compare YH-Light with classical graph partitioning policies, including 2D, Par-METIS, and ADP. More importantly, we further compare YH-Light with the state-of-the-art vertex-cut XTree and the state-of-the-art hybrid cut TopoX.

As shown in Figure 3, the performance of the BFS based on YH-Light is up to orders of magnitude higher than that of any other



(a) Tianhe-Exa



(b) Intel Cluster

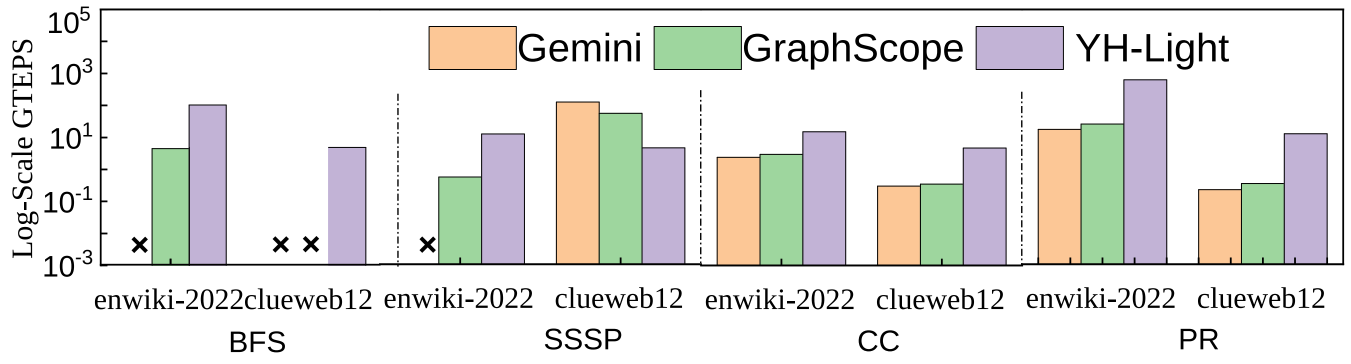
Figure 4: Pre-processing overhead of BFS with various graph partitions.

graph partitioning method. As the number of nodes increases to 4096 nodes, the GTEPS of YH-Light is significantly higher than that of the state-of-the-art hybrid TopoX and vertex-cut-XTree, where the peak performance of YH-Light using 4096 nodes is as high as 22,490.17 GTEPS when running a graph at *size* = 38. That is because YH-Light leverages knowledge of the hierarchical topology, while other partitioning strategies hardly exploit multilevel communication domains. We also obtain similar results on SSSP, where YH-Light achieves a 17.6× improvement over the best-performing baseline when using 4,096 Tianhe-Exa nodes.

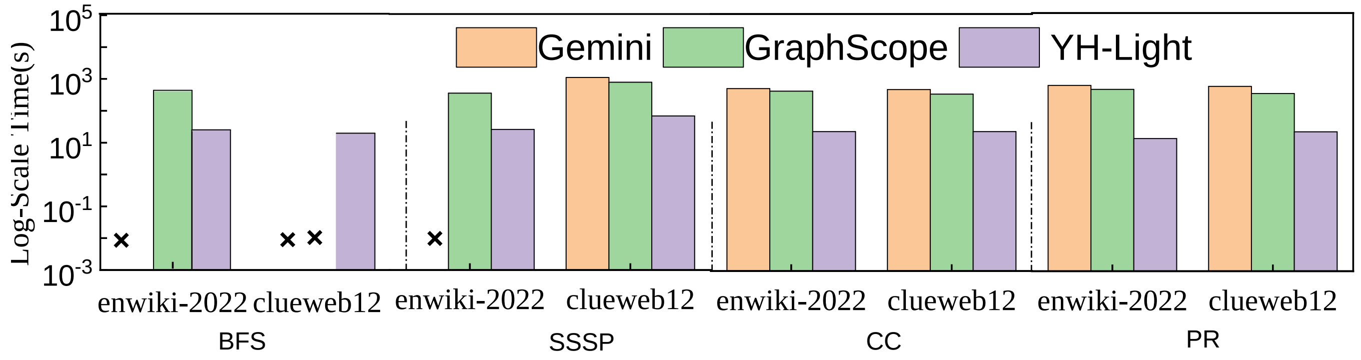
The extensive results show that the preprocessing cost of YH-Light is significantly lower than that of the others. In contrast, the communication reduction achieved by many advanced partitioning solutions often comes at the expense of redundant computational overhead; e.g., TopoX is a state-of-the-art partitioning method based on refactorization and requires many factorizations for partitioning.

5.7 YH-Light for Real-world Graphs

We further validate YH-Light and compare it with two famous graph systems, including Gemini and GraphScope by running various graph kernels with real-world graphs listed in Table 1. Note that GraphScope is the latest and state-of-the-art graph system available [8], so we take it as the representative graph system for large-scale graph tasks. Finally, we deployed YH-Light into an Intel clusters. and test it against GraphScope[8]. The performance of YH-Light-based BFS, and PR and CC (Connected Component[8, 16, 36, 50]) is superior to that of Gemini and GraphScope on a 64-CN testbed of Tianhe-Exa. YH-Light achieves the best performance in all graphs against two baselines, as shown in Figure 5(a). Note that Gemini failed to run the enwiki-2022 and neither GraphScope nor Gemini can run the bigger clueweb12 (marked as **X**) due to communication congestion. Furthermore, YH-Light has the smallest



(a) Performance



(b) Overhead

Figure 5: Runtime for deploying YH-Light on two real-world graphs.

preprocessing cost over Gemini and GraphScope as shown in Figure 5(b). That is because YH-Light greatly mitigates communication costs and improves graph processing performance, which beats state-of-the-art partitioning methods and strategies. We can also observe that GraphScope is only a few seconds away from YH-Light. But YH-Light outperforms GraphScope by a clear margin, which is very significant in the experiment of PR on enwiki-2022. This highlights the potential of YH-Light.

6 Conclusion and Future Work

To harness the full potential of HPC systems for large-scale graph processing, we propose YH-Light, a hierarchy-aware partitioning engine that fully utilizes hierarchical communication domains within large-scale HPC systems. We validate YH-Light with the Graph 500 benchmark and fundamental graph operations on two famous HPC systems, including up to 79K+ computing nodes and more than 1.2 million processor cores, YH-Light consistently outperforms state-of-the-art graph partitioning methods and graph processing systems. Looking ahead, the philosophy of YH-Light can be seamlessly integrated into graph learning frameworks, boosting their efficiency by optimizing communication and computation across hundreds of computing nodes in distributed large-scale HPC systems.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under grant Nos. 62372455, 62272476, 62032023, 42104078, and 6190241, partially supported by the PDL innovation research fund under grant No. 2023-JKWPDL-09, partially supported by China's National Key Research and Development Program under grant No. 2023YFB3001903. Xinbiao Gan and Chunye Gong are the corresponding authors.

References

[1] [n. d.]. Graph500:http://www.graph500.org/. ([n. d.]).

[2] 2022. National supercomputing Center in Changsha. http://nscc.hnu.edu.cn/info/1013/1011.htm 2022.

[3] Yasir Arfat, Rashid Mehmood, and Aiiad Albeshri. 2017. Parallel shortest path graph computations of United States road network data on apache spark. In *International Conference on Smart Cities, Infrastructure, Technologies and Applications*. Springer, 323–336.

[4] Huanqi Cao, Yuanwei Wang, Haojie Wang, Heng Lin, Zixuan Ma, Wanwang Yin, and Wenguang Chen. 2022. Scaling graph traversal to 281 trillion edges with 40 million cores. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 234–245.

[5] Fabio Checconi, Fabrizio Petrini, Jeremiah Willcock, Andrew Lumsdaine, Anamitra Roy Choudhury, and Yogish Sabharwal. 2012. Breaking the speed and scalability barriers for graph exploration on distributed-memory machines. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

[6] R. Chen, J. Shi, Y. Chen, and H. Chen. 2015. Powerlyra: Differentiated graph computation and partitioning on skewed graphs. *European Conference on Computer Systems* (2015), 1–15.

[7] Roshan Dathathri, Gurbinder Gill, Loc Hoang, Hoang-Vu Dang, Alex Brooks, Nikoli Dryden, Marc Snir, and Keshav Pingali. 2018. Gluon: A communication-optimizing substrate for distributed heterogeneous graph analytics. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 752–768.

[8] Wenfei Fan, Tao He, Longbin Lai, Xue Li, Yong Li, Zhao Li, Zhengping Qian, Chao Tian, Lei Wang, Jingbo Xu, et al. 2021. GraphScope: a unified engine for big graph processing. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2879–2892.

[9] Wenfei Fan, Ruiqi Xu, Qiang Yin, Wenyuan Yu, and Jingren Zhou. 2022. Application-driven graph partitioning. *The VLDB Journal* (2022), 1–24.

[10] Wenfei Fan, Ruiqi Xu, Qiang Yin, Wenyuan Yu, and Jingren Zhou. 2023. Application-driven graph partitioning. *The VLDB Journal* 32, 1 (2023), 149–172.

[11] Pablo Fuentes, Enrique Vallejo, José Luis Bosque, Ramón Beivide, Andreea Anghel, Germán Rodríguez, Mitch Gusat, and Cyriel Minkenberg. 2016. Synthetic traffic model of the Graph500 communications. In *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 675–683.

[12] Xinbiao Gan, Tiejun Li, Feng Xiong, Bo Yang, Xinhai Chen, Chunye Gong, Shijie Li, Kai Lu, Qiao Li, and Yiming Zhang. 2024. MST: Topology-aware message aggregation for exascale graph processing of traversal-centric algorithms. *ACM Transactions on Architecture and Code Optimization* 21, 4 (2024), 1–22.

[13] Xinbiao Gan, Guang Wu, Shenghao Qiu, Feng Xiong, Jiaqi Si, Jianbin Fang, Dezun Dong, Chunye Gong, Tiejun Li, and Zheng Wang. 2024. GraphCube: Interconnection hierarchy-aware graph processing. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 160–174.

[14] Xinbiao Gan, Yiming Zhang, Ruibo Wang, Tiejun Li, Tiaojie Xiao, Ruigeng Zeng, Jie Liu, and Kai Lu. 2021. TianheGraph: Customizing Graph Search for Graph500 on Tianhe Supercomputer. *IEEE Transactions on Parallel and Distributed Systems* 33, 4 (2021), 941–951.

[15] Xinbiao Gan, Yiming Zhang, Ruibo Wang, Tiejun Li, Tiaojie Xiao, Ruigeng Zeng, Jie Liu, and Kai Lu. 2021. TianheGraph: Customizing Graph Search for Graph500 on Tianhe Supercomputer. *IEEE Transactions on Parallel and Distributed Systems* (2021), 1–1. https://doi.org/10.1109/TPDS.2021.31007852

[16] Xinbiao Gan, Yiming Zhang, Ruigeng Zeng, Jie Liu, Ruibo Wang, Tiejun Li, Li Chen, and Kai Lu. 2022. XTree: Traversal-Based Partitioning for Extreme-Scale Graph Processing on Supercomputers. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2046–2059.

[17] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*. 17–30.

[18] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. 2014. GraphX: graph processing in a distributed dataflow framework. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*. 599–613.

[19] Samuel Grossman, Heiner Litz, and Christos Kozyrakis. 2018. Making pull-based graph processing performant. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 246–260.

[20] Juris Hartmanis. 1982. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review* 24, 1 (1982), 90.

[21] http://graph500.org/. 2021. The Graph 500 List. https://graph500.org/ Last accessed 03 March 2022.

[22] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20, 1 (1998), 359–392.

[23] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. 2012. Graphchi: Large-scale graph computation on just a pc. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 31–46.

[24] Dongsheng Li, Yiming Zhang, Jinyan Wang, and KianLee Tan. 2019. TopoX: Topology Refactorization for Efficient Graph Partitioning and Processing. *PVLDB* 12, 8 (2019), 891–905.

[25] Dongsheng Li, Yiming Zhang, Jinyan Wang, and Kian-Lee Tan. 2019. TopoX: Topology refactorization for efficient graph partitioning and processing. *Proceedings of the VLDB Endowment* 12, 8 (2019), 891–905.

[26] Yawen Li, Ye Yuan, Yishu Wang, Xiang Lian, Yuliang Ma, and Guoren Wang. [n. d.]. Distributed Multimodal Path Queries. *IEEE Transactions on Knowledge and Data Engineering* ([n. d.]). https://doi.org/10.1109/TKDE.2020.3020185 unpublished.

[27] Xiang-Ke Liao, Zheng-Bin Pang, Ke-Fei Wang, Yu-Tong Lu, Min Xie, Jun Xia, De-Zun Dong, and Guang Suo. 2015. High performance interconnect network for Tianhe system. *Journal of Computer Science and Technology* 30, 2 (2015), 259–272.

[28] Heng Lin, Xiaowei Zhu, Bowen Yu, Xiongchao Tang, Wei Xue, Wenguang Chen, Lufei Zhang, Torsten Hoefler, Xiaosong Ma, Xin Liu, et al. 2018. Shentu: processing multi-trillion edge graphs on millions of cores in seconds. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 706–716.

[29] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. 2012. Distributed GraphLab: A Framework for Machine Learning in the Cloud. *PVLDB* 5, 8 (2012), 716–727.

[30] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2009. Pregel: a system for large-scale graph processing. *Sigmod* (2009), 135–146.

[31] Rashid Mehmood and Gary Graham. 2015. Big data logistics: a health-care transport capacity sharing model. *Procedia computer science* 64 (2015), 1107–1114.

[32] Joel Nishimura and Johan Ugander. 2013. Restreaming graph partitioning: simple versatile algorithms for advanced balancing. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1106–1114.

[33] Sheldon M Ross. 2020. *Introduction to probability and statistics for engineers and scientists*. Academic press.

[34] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. 2013. X-stream: Edge-centric graph processing using streaming partitions. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 472–488.

[35] Julian Shun and Guy E Blelloch. 2013. Ligra: a lightweight graph processing framework for shared memory. In *ACM Sigplan Notices*, Vol. 48. ACM, 135–146.

[36] Stergios Stergiou, Dipen Rughwani, and Kostas Tsioutsiouliklis. 2018. Short-cutting label propagation for distributed connected components. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 540–546.

[37] Craig B Stunkel, Richard L Graham, Gilad Shainer, Michael Kagan, SS Sharkawi, B Rosenburg, and GA Chochia. 2020. The high-speed networks of the Summit and Sierra supercomputers. *IBM Journal of Research and Development* 64, 3/4 (2020), 3–1.

[38] theregister. 2022. biden-china-supercomputers. https://www.theregister.com/2021/04/09/biden_china_supercomputers/ 2022.

[39] TOP500.org. 2021. TOP 500 List. https://www.top500.org/ Last accessed 01 March 2022.

[40] Koji Ueno, Toyotaro Suzumura, Naoya Maruyama, Katsuki Fujisawa, and Satoshi Matsuoka. 2016. Extreme scale breadth-first search on supercomputers. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 1040–1047.

[41] Erik Vermij, Leandro Fiorin, Christoph Hagleitner, and Koen Bertels. 2017. Boosting the efficiency of HPCG and Graph500 with near-data processing. In *2017 46th International Conference on Parallel Processing (ICPP)*. IEEE, 31–40.

[42] Ruibo Wang, Kai Lu, Juan Chen, Wenzhe Zhang, Jinwen Li, Yuan Yuan, Pingjing Lu, Libo Huang, Shengguo Li, and Xiaokang Fan. 2020. Brief introduction of TianHe exascale prototype system. *Tsinghua Science and Technology* 26, 3 (2020), 361–369.

[43] Sibo Wang, Xiaokui Xiao, Yin Yang, and Wenqing Lin. 2016. Effective indexing for approximate constrained shortest path queries on large road networks. *Proceedings of the VLDB Endowment* 10, 2 (2016), 61–72.

[44] Da Yan, Yingyi Bu, Yuanyuan Tian, and Amol Deshpande. 2017. Big Graph Analytics Platforms. *Found. Trends Databases* 7, 1-2 (2017), 1–195. https://doi.org/10.1561/1900000056

[45] Andy Yoo, Edmond Chow, Keith Henderson, William McLendon, Bruce Hendrickson, and Umit Catalyurek. 2005. A scalable distributed parallel breadth-first search algorithm on BlueGene/L. In *SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*. IEEE, 25–25.

[46] Yiming Zhang, Kai Lu, and Wenguang Chen. 2021. Processing extreme-scale graphs on China's supercomputers. *Commun. ACM* 64, 11 (2021), 60–63.

[47] Yiming Zhang, Haonan Wang, Menghan Jia, Jinyan Wang, Dong sheng Li, Guangtao Xue, and K. Tan. 2020. TopoX: Topology Refactorization for Minimizing Network Communication in Graph Computations. *IEEE/ACM Transactions on Networking* 28 (2020), 2768–2782.

[48] Yishui Li etc. Zhe Li, Chengkun Wu. 2021. FEP-Based Large-Scale Virtual Screening for Effective Drug Discovery against COVID-19.

https://www.hpcwire.com/2021/11/18/gordon-bell-special-prize-goes-to-world-shaping-covid-droplet-work/

[49] Andy Diwen Zhu, Hui Ma, Xiaokui Xiao, Siqiang Luo, Youze Tang, and Shuigeng Zhou. 2013. Shortest path and distance queries on road networks: towards bridging theory and practice. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 857–868.

[50] Xiaowei Zhu, Wenguang Chen, Weimin Zheng, and Xiaosong Ma. 2016. Gemini: A Computation-Centric Distributed Graph Processing System. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, Kimberly Keeton and Timothy Roscoe (Eds.). USENIX Association, 301–316. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/zhu