

Ligra-DAT: Graph-Level Edge-Density-Based Threshold Selection for Ligra

Zhaoqin Ban, Xinxin Yang, Dapeng Fu, Tengfei Cao, and Jianqiang Huang*

Abstract Ligra, a lightweight shared-memory multi-core graph processing framework, uses EDGEMAP with a static threshold ($\rho = |E|/20$) to switch between sparse and dense traversal modes. However, this static threshold does not account for edge-density differences among input graphs, which may cause delayed switching for low-density graphs and premature switching for high-density graphs. We propose Ligra-DAT, a graph-level edge-density-based threshold selection mechanism: after graph loading, it selects $\rho \times 1.5$ for high-density graphs ($\delta > 0.1$) to delay dense-mode switching and $\rho \times 0.7$ for low-density graphs ($\delta \leq 0.1$) to accelerate dense-mode switching. The selected threshold is then reused during EDGEMAP execution and is not retuned at each traversal iteration. Experiments on a 64-core server with 8 datasets and 6 core algorithms show that Ligra-DAT achieves up to 42.4% runtime reduction, preserves compatibility with Ligra applications, has $O(1)$ threshold-selection complexity, and incurs overhead less than 0.1% of the corresponding Ligra execution time.

Keywords Graph Processing, Shared Memory, Ligra, Edge-Density-Based Threshold Selection, Edge Density

1 Introduction

With the exponential growth of graph data in the digital era—such as social networks (Twitter contains 4.17×10^7 vertices and 1.47×10^9 edges), web graphs (billions of hyperlinks), and biological networks (protein-protein interaction graphs with millions of nodes)—parallel graph processing frameworks have become indispensable tools for extracting insights from large-scale graph data. Among these frameworks, shared-memory systems like Ligra [1] offer practical advantages over distributed systems such as Pregel [2] and Giraph [3]: they avoid inter-machine communication over-

head and provide a simpler codebase for deployment and maintenance.

A central component of Ligra is its EDGEMAP function, which switches between two traversal modes according to the static threshold $\rho = \frac{|E|}{20}$. The two traversal modes are summarized as follows:

- **Sparse mode:** traverses the outgoing edges of source vertices and is optimized for small active vertex subsets (frontiers). Its work complexity is $O(|U| + \sum_{u \in U} \deg^+(u))$.
- **Dense mode:** traverses the incoming edges of target vertices and is optimized for large frontiers. Its work complexity is $O(|V| + \sum_{v \in V} \deg(v))$.

This mode-switching mechanism achieves efficient traversal for typical graphs, but its static threshold design ignores the global structural characteristic of edge density. Since EDGEMAP keeps sparse mode while the frontier-related work is no larger than the threshold and switches to dense mode only after this work exceeds the threshold, increasing the threshold delays dense-mode switching, whereas decreasing the threshold accelerates it. As indicated by the calibration results, this can degrade performance on graphs with very low or high edge density. The two regimes are $\delta > 0.1$ and $\delta \leq 0.1$. For high-density graphs, the static threshold is lower than the empirically optimal threshold, causing premature dense-mode switching. For low-density graphs, the static threshold is higher than the empirically optimal threshold, causing delayed dense-mode switching.

Real-world graphs exhibit wide variation in edge density, which directly affects the efficiency of traversal mode switching. For example, biological networks (e.g., PPI graphs) typically satisfy $\delta < 10^{-3}$, with most vertices connected to only 2–5 neighbors. In such low-density cases, the empirically preferred threshold is below Ligra's default value, so the default threshold can keep the traversal in sparse mode longer than necessary. Conversely, high-density graphs or dense interaction subgraphs may satisfy $\delta > 0.1$, where frontiers can expand rapidly during traversal [4]. In such high-density cases, the empirically preferred threshold is above Ligra's default value, so the default threshold can switch to dense mode too early and trigger unnecessary full-graph scans. Even within a single graph, edge density may vary across subgraphs (e.g., a social network with dense core communi-

• Zhaoqin Ban, Xinxin Yang, Tengfei Cao, and Jianqiang Huang are with the Dept. of Computer Technology and Application, Qinghai University; Intelligent Computing and Application Laboratory of Qinghai Province; Qinghai Provincial Green Computing Power Engineering Technology Research Center, Xining 810016, China.

E-mail: Zhaoqin Ban, 2907799070@qq.com; Xinxin Yang, 530698641@qq.com; Tengfei Cao, caotf@qhu.edu.cn; Jianqiang Huang, hjqxaly@163.com.

• Dapeng Fu is with Ant Group, A Space, No. 569 Xixi Road, Hangzhou, China.

* Corresponding author: Jianqiang Huang (E-mail: hjqxaly@163.com).

ties and sparse peripheral connections). The static threshold cannot adapt to such local density differences, leading to uneven performance. This issue is particularly important for applications such as PageRank-style graph ranking [5], classic fast-unfolding community detection [6], and recent shared-memory community detection on dynamic graphs [7], where graph traversal efficiency directly impacts overall performance.

Existing optimizations for Ligra—including algorithm expansion, local metric optimization, and heterogeneous adaptation—focus on extending functionality or adapting to specific hardware, but do not directly address the mismatch between a static threshold and graph edge density. Other single-machine graph processing systems have explored different optimization approaches, while edge-density-based threshold selection for shared-memory traversal remains underexplored. Even distributed systems that consider edge density for graph partitioning have not addressed the threshold-selection problem for traversal mode switching. To fill this gap, we propose Ligra-DAT, a graph-level edge-density-based threshold selection mechanism, building on the original Ligra framework [1], parallel algorithms for irregular data [8], parallel graph connectivity [9], and connected-components research [10].

To summarize, our contributions are listed as follows:

- **Analytical and empirical contribution:** Through analysis of 119 graphs (19 random-local + 2 dedicated datasets with 50 graphs each), we show that edge density is correlated with the preferred mode-switching threshold. The calibration results identify the practical boundary and stable coefficient choices as $\delta = 0.1$, with $\rho' = 1.5\rho$ for $\delta > 0.1$ and $\rho' = 0.7\rho$ for $\delta \leq 0.1$ in the evaluated regimes.
- **Methodological contribution:** We design a lightweight graph-level edge-density-based threshold selection module with $O(1)$ time complexity (3–4 arithmetic operations). It preserves Ligra’s traversal semantics and data structures, requires no changes to user-level Ligra applications, and is further characterized against representative threshold-selection alternatives. The selected threshold is graph-specific but remains fixed during one graph execution.
- **Experimental contribution:** We validate Ligra-DAT on 8 datasets (5 synthetic + 3 real-world including SNAP datasets [11] and STRING protein networks [12]) and 6 core graph algorithms, covering traversal, ranking, centrality, and connectivity tasks. The results show consistent performance across graph types, edge densities, and the evaluated Intel Xeon hardware configurations

[13].

2 Related Work

To contextualize our work, we systematically review existing research on Ligra optimization, threshold design for graph processing, and edge density applications, highlighting the evolutionary trajectory of these fields and identifying the specific gaps addressed by Ligra-DAT.

2.1 Ligra and Recent Graph-Processing Optimizations

Ligra defines a lightweight shared-memory execution model whose EDGEMAP operator switches between sparse and dense traversal using the fixed rule $\rho = |E|/20$ [1]. Recent graph-processing research has advanced adjacent layers, but these systems should not be characterized as direct Ligra threshold extensions. Representative work includes MST’s topology-aware aggregation [14]; the survey of concurrent graph analytics [15]; in-memory dynamic-graph storage [16]; GastCoCo’s storage–prefetch co-design [17]; CoroGraph’s cache- and work-efficient execution [18]; TianheGraph’s topology-aware execution [19]; PRAGA’s priority-aware acceleration [20]; ABi-BFS’s shared-memory BFS design [21]; Spruce’s dynamic graph storage [22]; CAVE’s SSD-aware processing [23]; the modular GraphScope Flex stack [24]; and GEAR’s evolving-graph data arrangement [25].

Other recent systems include HBM-enabled Dynamic-ACTS [26], CPU–GPU processing in CGgraph [27], CPU–GPU co-execution in CGCGraph [28], and a recent survey of dynamic graph processing [29]. These methods establish the current graph-processing context, but they change algorithms, storage layouts, scheduling mechanisms, execution engines, or hardware assumptions. Their reported runtimes are therefore not interchangeable with a threshold-only modification inside Ligra. We use them as recent related systems and reserve the controlled experiment in Section 5.6 for Ligra-compatible threshold policies that keep the EDGEMAP implementation and experimental protocol fixed. Earlier systems such as GridGraph [30] and Presto [31] are retained as historical context for graph representation and distributed execution rather than as direct threshold-policy baselines.

2.2 Threshold Design for Graph Processing

Beyond Ligra-specific work, threshold design has been a recurring theme in graph processing. Direction-optimizing BFS uses frontier size and edge volume as switching signals [4]; push/pull graph computation models communication and

synchronization cost [32]; and multicore graph-processing analysis examines execution-state and workload effects [33]. These signals capture traversal state effectively, but they do not by themselves summarize persistent structural differences among input graphs. PowerGraph exploits degree skew for distributed partitioning [34]; PowerLyra differentiates computation and partitioning on skewed graphs [35]; MST uses topology-aware aggregation [14]; and recent shared-memory community-detection work targets dynamic graph structure [7]. These methods target different optimization layers rather than a general Ligra EDGEMAP threshold. Locality characterization in graph processing [36] and cache-oriented graph-analytics optimization [37] provide another complementary direction, often with additional profiling, layout transformation, or hardware-specific tuning. Ligra-DAT instead evaluates whether global edge density can serve as a low-cost graph-level signal while retaining $O(1)$ threshold-selection complexity.

2.3 Edge Density in Graph Research

Edge density is a standard global graph statistic, but its use as a direct signal for sparse/dense mode switching in lightweight shared-memory frameworks remains underexplored. R-MAT provides a recursive model for generating synthetic graphs at controlled vertex and edge scales [38], which supports evaluation across a broad density range. Biological-network studies likewise use dense and sparse substructures to characterize functional organization [12]. In adjacent optimization layers, PowerGraph demonstrates the value of degree-aware computation placement [34], while multilevel graph partitioning uses graph structure to balance computation [39]. Direction-optimizing BFS [4], distributed-memory parallel BFS [40], scalable multicore traversal [41], push/pull selection [32], and multicore graph-processing analysis [33] separately show that traversal direction and execution strategy benefit from workload-aware selection. These methods do not constitute density-based Ligra threshold policies, but together they motivate evaluating global edge density as a lightweight graph-level structural signal. Ligra-DAT addresses this narrower mode-switching problem without claiming that density replaces frontier-, degree-, partition-, or hardware-aware optimization.

2.4 Comparison with Existing Methods

Table 1 gives a compact comparison of threshold-relevant traversal designs. The broader recent systems reviewed in Section 2.1 are not repeated here because their storage, schedul-

ing, execution-engine, or hardware optimizations are not directly comparable with an EDGEMAP threshold policy.

Table 7 subsequently evaluates lightweight Ligra adaptations of three recent methods' decision signals under the same implementation and experimental protocol. The ABi-BFS, GreLigra, and CoroGraph columns report measured Ligra-side adaptations of their respective frontier, structure, and cache/work-efficiency signals rather than complete reimplementations of the original systems.

3 Core Mechanisms and Limitations of Static Threshold in Ligra

To understand the limitations of Ligra's static threshold, we first review Ligra's core components, then analyze the impact of edge density on threshold effectiveness, and validate the limitations with experimental data.

3.1 Core Components of Ligra

Ligra's architecture is built around four key components, with the EDGEMAP function as the core of mode switching:

- **Graph representation:** Uses adjacency lists to store outgoing edges (out_edges) and incoming edges (in_edges). The construction time and space complexity are $O(|V| + |E|)$ and $O(|E|)$, respectively.
- **VertexSubset:** A flexible data structure for active vertices. It uses a sparse array when $|U| \leq \rho$ and a dense boolean array when $|U| > \rho$, supporting automatic format conversion.
- **EDGEMAP:** Core traversal function, with the simplified pseudo-code shown in Algorithm 1.
- **VERTEXMAP:** Parallel vertex traversal function for state updates (e.g., marking visited vertices in BFS), with time complexity $O(|U|)$.

Algorithm 1 Simplified EDGEMAP Pseudo-code

```

1: Input: Active vertex subset  $U$ , adjacency list  $G$ , threshold  $\rho$ 
2: Output: Updated vertex subset  $U'$ 
3: if  $|U| + \sum_{u \in U} \deg^+(u) \leq \rho$  then
4:   Sparse Mode: Traverse outgoing edges of  $U$ 
5:    $U' \leftarrow \{v \mid \exists u \in U, (u, v) \in G.out\_edges\}$ 
6: else
7:   Dense Mode: Traverse incoming edges of all vertices
8:    $U' \leftarrow \{v \mid \exists u \in U, (u, v) \in G.in\_edges\}$ 
9: return  $U'$ 

```

Here, the frontier-related work metric used by the original Ligra EDGEMAP switching rule is $W(U) = |U| + \sum_{u \in U} \deg^+(u)$. The first term accounts for the active-frontier vertices, while the second term accounts for the outgoing-edge work induced by the active frontier. We use this fixed metric consistently in the following analysis and in the Ligra-DAT implementation.

Table 1 Compact Comparison of Threshold-Relevant Traversal Designs

Method	Primary decision signal	Optimization scope	Role in this study
Ligra (Static) [1]	Frontier work compared with fixed $ E /20$	EDGEMAP sparse/dense switching	Direct runtime baseline
Direction-optimizing BFS [4]	Frontier size and edge volume	BFS push/pull switching	Related traversal design
Push/pull graph computation [32]	Communication and synchronization cost	General push/pull selection	Related traversal design
ABi-BFS [21]	Shared-memory BFS execution state	BFS-specific parallel traversal	Recent related method
Ligra-DAT	Global edge density	Graph-level EDGEMAP threshold selection	Proposed method

3.2 Limitations of the Static Threshold

Ligra's default threshold $\rho = |E|/20$ is designed for typical graphs but can be suboptimal when edge density is far from the regime for which the static rule is suitable. We first give an analytical rationale for the density-dependent switching bias, and then calibrate the boundary and coefficients with random-local graphs and real-world datasets.

We define core variables consistently with graph mining literature [42]: the number of vertices is $|V| = n$, the number of edges is $|E| = m$, and the edge density is

$$\delta = \frac{m}{n(n-1)} \quad (1)$$

where self-loops are excluded. Let $|U| = k$ denote the size of the active frontier, and define the average out-degree as

$$\bar{d} = \frac{m}{n}$$

For large n , the average degree satisfies

$$\bar{d} \approx n\delta$$

Analytical Rationale: The following model is used only to estimate the density scale at which the relative cost of sparse and dense traversal changes. The exact boundary and scaling factors are determined by calibration experiments rather than by treating the model as a universal proof.

In sparse mode, traversal work is dominated by the outgoing edges of the active frontier. Under an average-degree approximation,

$$\sum_{u \in U} \deg^+(u) \approx k\bar{d} \approx kn\delta \quad (2)$$

and the sparse-mode cost can be written as

$$T_S \approx \alpha kn\delta \quad (3)$$

where α captures the cost of irregular edge accesses.

In dense mode, EDGEMAP uses a bitmap-style representation and scans candidate target vertices and their incoming edges. The effective dense-mode cost depends on the fraction of active bitmap entries and memory-access locality. Following the intuition of direction-optimizing traversal models

[4, 40, 41, 43], we write the simplified dense-mode cost as

$$T_D \approx \beta n^2 \delta^2 \quad (4)$$

where β is the dense-traversal coefficient. This form is not intended to model every graph class exactly; it captures the fact that dense traversal becomes increasingly sensitive to graph density and frontier coverage.

Equating the two approximate costs gives

$$\alpha kn\delta_0 = \beta n^2 \delta_0^2 \quad (5)$$

which yields

$$\delta_0 = \frac{k}{(\beta/\alpha)n} \quad (6)$$

Under the average-degree approximation, the EDGEMAP switching metric becomes

$$|U| + \sum_{u \in U} \deg^+(u) \approx k(1 + n\delta)$$

Ligra's default threshold can be approximated as

$$\rho = \frac{|E|}{20} \approx \frac{n^2 \delta}{20}$$

The representative switching scale therefore satisfies

$$k(1 + n\delta) \approx \frac{n^2 \delta}{20}$$

For the large sparse and dense graphs considered in this work, $n\delta$ dominates the constant term in the evaluated switching region, giving

$$k \approx \frac{n}{20}$$

Using this scale and a representative range

$$\frac{\beta}{\alpha} \in [0.5, 0.6]$$

the estimated transition density is approximately

$$\delta_0 \in [0.08, 0.1]$$

This estimate explains why a boundary near 10% is plausible, while the specific boundary used by Ligra-DAT is selected from the measured Ratio distribution.

Experimental Calibration: We calibrated the threshold rule using a separate set of 19 random-local graphs

(not included in Table 5) over the density range $\delta \in [0.000010001, 0.600060006]$. Based on graph mining best practices [42], we define the threshold deviation ratio as $\text{Ratio} = \frac{\text{Optimal Threshold}}{|E|/20}$, and the runtime reduction as $RR = \left(\frac{T_{\text{Ligra}} - T_{\text{Optimal}}}{T_{\text{Ligra}}} \right) \times 100\%$. The former measures the deviation of the static threshold from the calibrated threshold, while the latter measures the potential performance gain under the calibrated threshold.

Table 2 presents key results from this calibration set. All experiments were repeated five times, with results reported as averages. The PPI-sample is a real-world ultra-sparse graph from the STRING database used only as an external low-density sanity check and is not included in the 119 calibration graphs [12]. Table 2 is used to calibrate the direction and magnitude of threshold deviation, whereas the end-to-end algorithm-level performance is evaluated in Section 5.

3.3 Critical Observations

The analytical rationale in Section 3.2 and the calibration results in Table 2 both support using $\delta = 0.1$ as a practical transition point for threshold adjustment.

From Table 2 and additional experimental data from R-MAT generated graphs [38], we draw three critical observations:

- (1) **Low-density graphs:** For $\delta \leq 0.1$, the Ratio ranges from 0.60 to 0.79, indicating that the optimal threshold is $0.60 \cdot \frac{|E|}{20}$ to $0.79 \cdot \frac{|E|}{20}$. Using the static threshold therefore keeps sparse mode active longer than the empirically preferred switching point, delaying dense-mode switching and reducing bandwidth utilization—consistent with community-detection findings on evolving graphs [7].
- (2) **High-density graphs:** For $\delta > 0.1$, the Ratio ranges from 1.43 to 1.56, indicating that the optimal threshold is $1.43 \cdot \frac{|E|}{20}$ to $1.56 \cdot \frac{|E|}{20}$. Using the static threshold therefore switches to dense mode earlier than the empirically preferred switching point, causing unnecessary dense scans before the frontier is sufficiently large—aligning with direction-optimizing BFS research [4, 41].
- (3) **Practical boundary:** Around $\delta = 0.1$, the Ratio changes from below 1 to above 1 in the calibration data, supporting this value as a practical boundary for threshold adjustment. Additional validation on synthetic R-MAT graphs [38], real-world SNAP graphs [11], and STRING biological networks [12] shows the same trend in the evaluated datasets.

3.4 Specific Impacts on Real-World Applications

The limitations of the static threshold directly affect real-world application performance across various domains:

- **Social network analysis:** For the Facebook graph ($\delta = 0.000076$), Ligra’s static threshold keeps traversal away from the empirically preferred switching point, producing avoidable BFS traversal overhead. This motivates density-aware switching for latency-sensitive social-network workloads.
- **Biological network mining:** For a PPI graph ($\delta = 0.000117$), delayed dense-mode switching reduces memory bandwidth utilization during connectivity-oriented traversal, increasing avoidable processing cost in protein-interaction workloads.
- **Web-style graph ranking:** For the rMat24 web-style graph ($\delta = 0.0640000$), the mismatch between the static threshold and the empirically preferred switching point increases avoidable PageRank traversal work, motivating density-aware threshold selection for ranking workloads.
- **Evolving graph processing:** For real-time social media graphs with changing edge density, static threshold can lead to inconsistent performance across time windows, reducing the responsiveness of evolving community detection.

4 Design of Ligra-DAT: Graph-Level Edge-Density-Based Threshold Selection

To address the limitations of Ligra’s static threshold, we propose a graph-level edge-density-based threshold selection mechanism (Ligra-DAT) that selects the threshold according to the edge density of each input graph. Instead of using one universal threshold for all graphs, Ligra-DAT computes the selected threshold once after graph loading from $|V|$, $|E|$, and δ , and then reuses it during EDGEMAP traversal. The term “graph-level” indicates that selection is performed once for each input graph through a global structural signal; it does not imply online threshold retuning across iterations within one execution. Finer-grained frontier-level or local-density-aware adaptation is left to future work. The design follows the lightweight design philosophy of Ligra [1] and energy-efficient graph processing research [44], with analytical support from graph algorithm complexity studies [8, 40, 43].

4.1 Design Principles

- **Compatibility:** Ligra-DAT does not change the traversal semantics of EDGEMAP/VERTEXMAP or the Vertex-

Table 2 Experimental Results of Static Threshold Limitations (Calibration Set)

Filename	Graph density (δ)	Ratio	$ E /20$ Runtime (s)	Best Runtime (s)	RR (%)
random-local05	0.000010001	0.60	0.0000211	0.0000207	1.96
random-local04	0.000100001	0.63	0.0041811	0.0041652	0.38
random-local03	0.0010001	0.79	0.0003710	0.0003702	0.21
random-local02	0.010001	0.62	0.0002237	0.0002225	0.54
random-local1	0.100010001	1.54	0.0004066	0.0004054	0.29
random-local2	0.200020002	1.54	0.0003040	0.0003024	0.53
random-local3	0.300030003	1.56	0.0002843	0.0002821	0.78
random-local4	0.400040004	1.51	0.0002094	0.0002085	0.43
random-local5	0.500050005	1.47	0.0002031	0.0002025	0.26
random-local6	0.600060006	1.44	0.0002155	0.0002147	0.38
random-local11	0.110011001	1.56	0.0004108	0.0003936	4.19
random-local12	0.120012001	1.58	0.0003778	0.0003727	1.37
random-local13	0.130013001	1.45	0.0003673	0.0003667	0.17
random-local14	0.140014001	1.56	0.0003599	0.0003579	0.57
random-local15	0.150015002	1.44	0.0003332	0.0003315	0.49
random-local16	0.160016002	1.43	0.0003275	0.0003262	0.39
random-local17	0.170017002	1.54	0.0003346	0.0003289	1.70
random-local18	0.180018002	1.53	0.0003606	0.0003562	1.22
random-local19	0.190019002	1.43	0.0003501	0.0003491	0.29
PPI-sample	0.000117	0.68	0.001245	0.001237	0.64

Subset data structure. It only replaces the threshold value used in the existing mode-switching condition.

- **Efficiency:** Edge density calculation and threshold update operate in $O(1)$ time complexity, and the measured overhead is $\leq 0.1\%$ relative to the corresponding Ligra execution time. This preserves Ligra's lightweight design.
- **Correctness:** Only adjusts the timing of mode switching, not the underlying logic of graph algorithms (e.g., BFS traversal order, PageRank iteration rules). This preserves the output results of the original Ligra implementation.
- **Scope:** Supports the undirected/directed, weighted/unweighted graph formats and Ligra-compatible algorithms evaluated in this work, including traversal, ranking, centrality, and connectivity tasks.
- **Selection granularity:** Selects one threshold after graph loading and keeps it unchanged during the subsequent EDGEMAP iterations. Iteration-level frontier adaptation is outside the current implementation.

The selection in Ligra-DAT is therefore graph-level and edge-density-based rather than iteration-level. Different graph structures are mapped to different threshold regimes: sparse biological, social, and web-style graphs use a lower threshold to avoid staying in sparse mode beyond the preferred

switching point, whereas high-density graphs use a higher threshold to avoid entering dense mode too early. We deliberately use global edge density as a first-order structural signal; runtime frontier size, active-vertex degree skew, local community density, and power-law effects are not claimed to be fully modeled by the current implementation. This design trades fine-grained local responsiveness for $O(1)$ overhead, deterministic behavior, and compatibility with Ligra-style applications.

4.2 Core Modular Architecture

Ligra-DAT is embedded into Ligra's graph loading and EDGEMAP preprocessing pipelines, consisting of three interdependent modules. The data flow of this architecture is as follows:

- (1) The Graph Loading module provides $|V|$ and $|E|$ to the Edge Density Calculation Module.
- (2) The Edge Density Calculation Module computes the edge density δ using the standard graph density formula and passes it to the Threshold Selection Module.
- (3) The Threshold Selection Module generates the edge-density-based threshold ρ' and sends it to the EDGEMAP module for mode switching decision.

Edge Density Calculation Module:

- **Trigger timing:** Immediately after graph loading (before

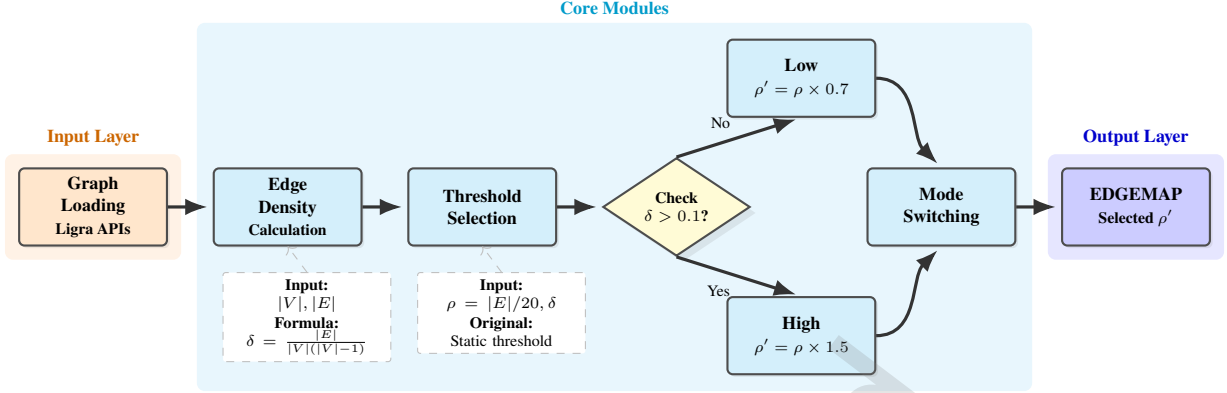


Fig. 1 Architecture of Ligra-DAT showing data flow from graph loading through edge-density-based threshold selection and mode switching. The system consists of three interdependent core modules (Edge Density Calculation, Threshold Selection, and Mode Switching) and reuses Ligra’s existing EDGEMAP traversal modes.

EDGEMAP is called for the first time), so that the threshold is available before EDGEMAP is called.

- **Input:** The number of vertices and edges obtained via Ligra’s built-in APIs `G.get_num_vertices()` and `G.get_num_edges()`: $|V|, |E|$.
- **Output:** The edge density is calculated as $\delta = \frac{|E|}{|V|(|V|-1)}$.
- **Key design:** Uses 64-bit floating-point storage to avoid precision loss for large graphs. For example, the Twitter graph has approximately $|V| \approx 4.17 \times 10^7$, $|E| \approx 1.47 \times 10^9$.

Threshold Selection Module:

- **Trigger timing:** After edge density calculation, and reused for all subsequent EDGEMAP calls on static graphs. For evolving graphs, incremental recalculation after edge updates is left as future work rather than being evaluated in this paper.
- **Input:** The original static threshold and edge density are $\rho = \frac{|E|}{20}, \delta$.
- **Output:** The selected threshold ρ' is determined using the calibration results in Section 3.2.
- **Key design:** Stores ρ' as a 64-bit unsigned integer (`uint64_t`) to avoid overflow for large graphs. For example, when $|E| = 10^{11}$, $\rho' = 7.5 \times 10^9$, the selected threshold remains below the maximum `uint64_t` value, approximately 1.8×10^{19} .

Mode Switching Module:

- **Trigger timing:** During each EDGEMAP call (replaces static ρ with ρ' in mode decision logic), compatible with both sparse and dense mode optimizations.
- **Input:** threshold ρ' and the original Ligra frontier-work metric $W(U) = |U| + \sum_{u \in U} \deg^+(u)$.
- **Output:** Mode selection is based on $W(U) \leq \rho'$ (sparse

mode) or $W(U) > \rho'$ (dense mode)

- **Key design:** Minimal modification to EDGEMAP (only replaces ρ with ρ'), ensuring compatibility with existing Ligra extensions.

4.3 Edge Density Calculation

Edge density δ quantifies the ratio of actual edges to the maximum possible edges (excluding self-loops) in a graph. For undirected graphs, the maximum number of edges is $\frac{|V|(|V|-1)}{2}$. Ligra’s graph representation treats undirected graphs as bidirectional directed graphs by storing two directed edges for each undirected edge. Thus, the same calculation formula $\delta = \frac{|E|}{|V|(|V|-1)}$ applies to both directed and undirected graphs, consistent with graph mining literature.

We compared our $O(1)$ calculation method with two alternative approaches (full traversal and sampling) to validate its efficiency, using large-scale graphs from SNAP and R-MAT generators (Table 3).

The results show that the $O(1)$ method adds very small overhead in the tested cases, while the other methods add significant runtime overhead (especially for large graphs), making them unsuitable for lightweight frameworks like Ligra.

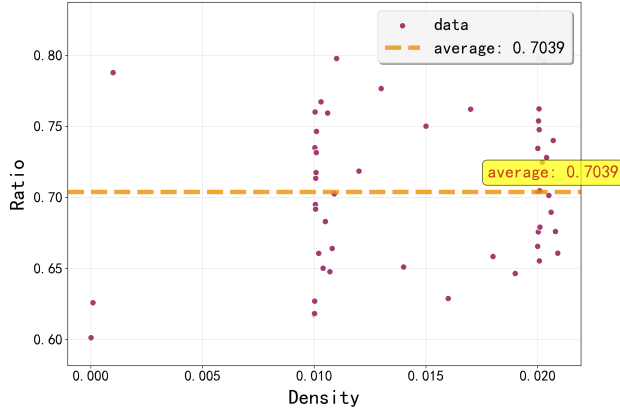
4.4 Edge-Density-Based Threshold Selection Logic

Based on the calibration datasets and additional validation with R-MAT graphs, we use the practical boundary $\delta = 0.1$. We selected the adjustment coefficients using two dedicated datasets (50 graphs with low edge density and 50 high-density graphs), whose Ratio distributions are shown in Figures 2 and 3.

For low-density graphs, the average Ratio is $\overline{\text{Ratio}}_{\delta \leq 0.1} = 0.7039$, which supports the adjustment coefficient of 0.7.

Table 3 Overhead Comparison of Edge Density Calculation Methods

Graph	$ V $	$ E $	O(1) Method (ms)	Full Traversal (ms)	Sampling Method (ms)
Twitter [11]	4.17e7	1.47e9	0.02	128.5	417.2
rMat27 [38]	1.34e8	8.00e9	0.03	642.1	1340.5
Facebook [11, 45]	1.34e5	1.38e6	0.01	2.61	28.8
PPI [12]	3.89e4	1.77e5	0.005	0.015	0.39

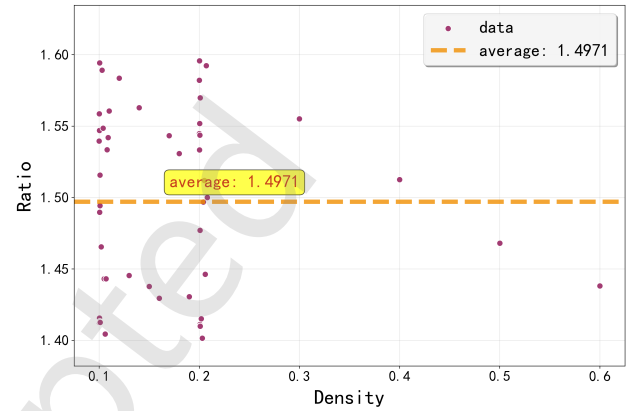
**Fig. 2** Ratio distribution for graphs with low edge density ($\delta \leq 0.1$) including SNAP and STRING datasets.

For high-density graphs, the average Ratio is $\overline{\text{Ratio}}_{\delta > 0.1} = 1.4971$, which supports the adjustment coefficient of 1.5—consistent with parallel graph connectivity research [9, 10, 46]. The resulting threshold-selection rule is $\rho' = 0.7\rho$ for $\delta \leq 0.1$ and $\rho' = 1.5\rho$ for $\delta > 0.1$. To verify the sensitivity of the coefficients (0.7 and 1.5), we conducted additional experiments with coefficients in the ranges [0.5, 0.9] for low-density graphs and [1.2, 1.8] for high-density graphs, using BFS and PageRank as benchmark algorithms:

- For high-density graphs with $\delta = 0.15$, RR peaks at 42.3% when the coefficient is 1.5 and decreases by 5%–8% if the coefficient deviates by ± 0.1 .
- For low-density graphs with $\delta = 0.03$, RR peaks at 42.4% when the coefficient is 0.7 and decreases by 4%–6% if the coefficient deviates by ± 0.1 .

These results support 0.7 and 1.5 as stable coefficient choices for the tested low- and high-density regimes.

As shown in the parameter sensitivity analysis in Figures 4 and 5, the best tested coefficients are 0.7 ($\delta = 0.03$) and 1.5 ($\delta = 0.15$). The corresponding peak RR values are 42.4% and 42.3%, respectively. When the coefficient deviates from the selected value by ± 0.1 , performance decreases by 4%–8%, indicating that 0.7 and 1.5 are stable coefficient choices in the tested regimes.

**Fig. 3** Ratio distribution for high-density graphs ($\delta > 0.1$) generated with R-MAT model.

4.5 Near-Boundary Coefficient Sensitivity Analysis

To further examine the boundary at $\delta = 0.1$, we conduct a sensitivity analysis on graphs near this threshold. Specifically, we evaluate three representative cases: $\delta \in \{0.09, 0.10, 0.11\}$. For each case, we vary the scaling coefficient and measure the runtime reduction rate (RR).

Figure 6 reports a coefficient sweep near the density transition boundary. The three curves correspond to $\delta = 0.09$, $\delta = 0.10$, $\delta = 0.11$. These measurements are taken from controlled validation graphs constructed for boundary analysis, not from the public datasets summarized in Table 5.

For $\delta = 0.09$, the runtime reduction reaches its maximum around a scaling coefficient of 0.7 and then decreases as the coefficient increases. This trend is consistent with the low-density setting, where a smaller threshold makes EDGEMAP enter dense mode earlier than the default rule.

For $\delta = 0.11$, the preferred coefficient shifts toward 1.5, with performance improving from lower coefficients and peaking near 1.5. This trend is consistent with the high-density setting, where a larger threshold delays dense-mode switching and reduces premature full-graph dense scans.

At $\delta = 0.1$, the curve remains nearly flat over a broad coefficient interval. Coefficients in the approximate range $0.7 \leq c \leq 1.5$ give similar RR values, so the boundary result should not be read as evidence that one coefficient is uniquely optimal at $\delta = 0.1$. Since the implementation

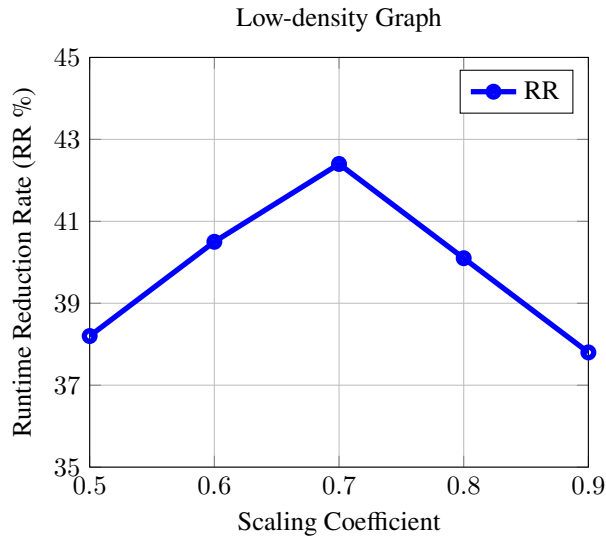


Fig. 4 Parameter sensitivity analysis (low-density graph): Impact of scaling coefficient on runtime reduction rate (RR).

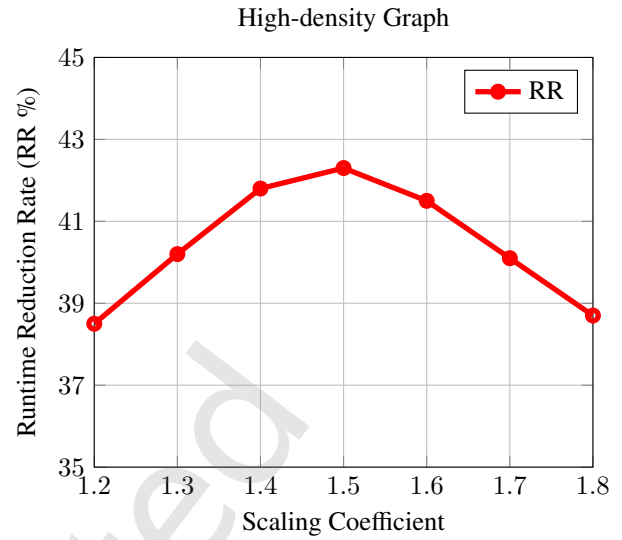


Fig. 5 Parameter sensitivity analysis (high-density graph): Impact of scaling coefficient on runtime reduction rate (RR).

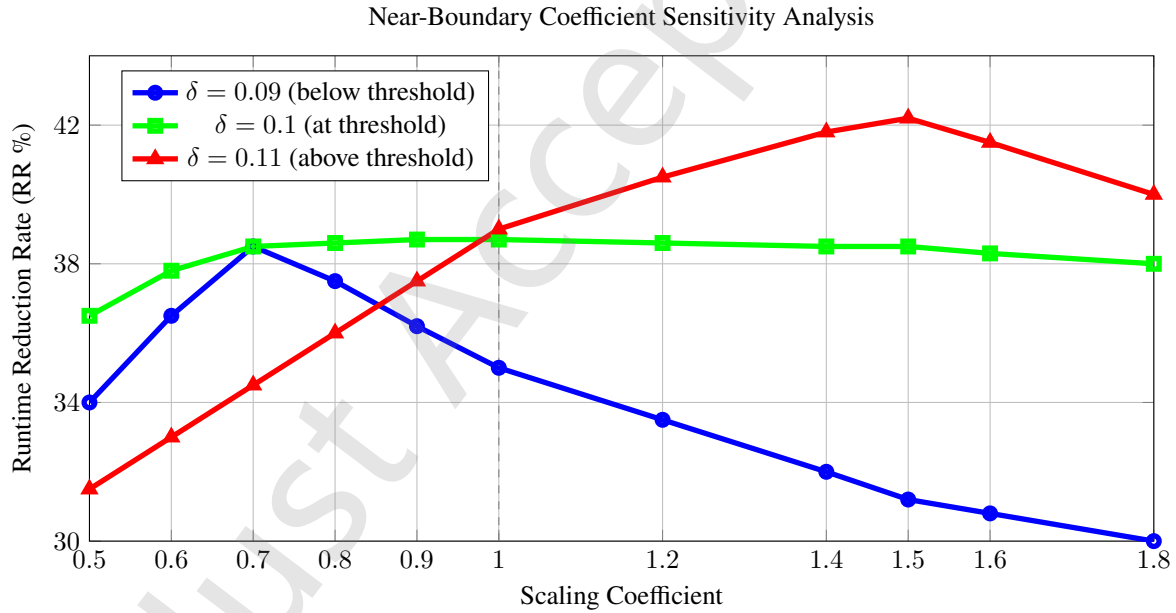


Fig. 6 Near-boundary coefficient sensitivity analysis. For $\delta = 0.09$, the best tested coefficient is around 0.7, while for $\delta = 0.11$, the best tested coefficient shifts toward 1.5. At $\delta = 0.1$, a broad coefficient interval yields nearly identical RR, indicating that the boundary region is less sensitive to the exact coefficient choice.

assigns $\delta \leq 0.1$ to the low-density branch, coefficient 0.7 is used at the boundary.

Taken together, the curves support treating $\delta = 0.1$ as a practical transition boundary. Below the boundary, the preferred coefficient moves toward the low-density setting; above the boundary, it moves toward the high-density setting; and at the boundary, performance changes little across a wide coefficient range.

4.6 Interpretation of Scaling Factors

While Section 4.4 empirically identifies the optimal scaling factors (0.7 and 1.5), we further connect these values to the direction of the mode-switching bias induced by different edge-density regimes. The purpose of this analysis is not to introduce a second runtime model, but to explain why the empirically selected coefficients move the threshold in opposite directions for sparse and dense graphs.

Bias in the Idealized Cost Model

The theoretical model in Section 3 assumes uniform memory access and balanced workload utilization. Real executions deviate from this idealized model in two representative regimes:

Low-density regime: For sparse biological, social, and web-style graphs, the default threshold can keep EDGEMAP in sparse mode longer than the empirically preferred switching point. In this regime, lowering the threshold helps the traversal enter dense mode earlier and avoid excessive sparse-mode edge expansion.

High-density regime: For high-density graphs, frontier expansion can make dense scans expensive if dense mode is entered too early. In this regime, increasing the threshold delays dense-mode switching until the frontier is large enough to justify the full-graph scan.

Piecewise First-Order Adjustment

We model the above two effects using a lightweight first-order scaling function:

$$\rho^* = \rho \cdot g(\delta) \quad (7)$$

where

$$g(\delta) = \begin{cases} 1 - c_2(1 - \delta) & \delta \leq 0.1 \\ 1 + c_1\delta & \delta > 0.1 \end{cases} \quad (8)$$

with $c_1 > 0$, $0 < c_2 < 1$. This formulation is used only as an interpretive first-order form. It is consistent with the intended switching behavior: $g(\delta) < 1$ for low-density graphs and $g(\delta) > 1$ for high-density graphs. Thus, the threshold is reduced in the low-density branch and increased in the high-density branch.

Connection to Empirical Results

In the low-density regime, the observed Ratio distribution concentrates around 0.7, which corresponds to the representative relation $1 - c_2(1 - \delta) \approx 0.7$. In the high-density regime, the observed Ratio distribution concentrates around 1.5, which corresponds to the representative relation $1 + c_1\delta \approx 1.5$ for the evaluated high-density calibration graphs. Therefore, the coefficients 0.7 and 1.5 are first-order approximations of the density-dependent switching bias rather than arbitrary constants.

This piecewise interpretation also avoids overclaiming: the current implementation uses two discrete coefficients to keep the overhead at $O(1)$ and to preserve compatibility with Ligra. A continuous or local-density-aware function $g(\delta_{\text{global}}, \delta_{\text{local}})$ is left as future work.

4.7 Overhead and Correctness Verification

We measured the overhead of Ligra-DAT on a 64-core Intel Xeon Max 9462 server across 6 datasets and 6 algorithms. The overhead is defined as $\text{Overhead} = \left(\frac{T_{\text{Ligra-DAT-overhead}} - T_{\text{Ligra}}}{T_{\text{Ligra}}} \right) \times 100\%$, where $T_{\text{Ligra-DAT-overhead}}$ is the runtime of Ligra-DAT with only the edge-density-based threshold selection module (no mode switching optimization). All overhead values are below 0.1% relative to the corresponding Ligra execution time in the evaluated workloads (Table 4), which is consistent with the lightweight design goal of Ligra-DAT.

Table 4 Overhead of Ligra-DAT (Unit: %)

Application	3D-grid	random-local	rMat24	rMat27	Twitter	Facebook
BFS [4, 43]	0.03	0.04	0.02	0.05	0.03	0.04
Betweenness Centrality [47]	0.05	0.06	0.04	0.07	0.05	0.06
PageRank (1 iteration) [5]	0.02	0.03	0.02	0.04	0.03	0.03
Graph Radii [48]	0.04	0.05	0.03	0.06	0.04	0.05
Connected Components [9, 10]	0.03	0.04	0.02	0.05	0.03	0.04
Bellman-Ford [49]	0.05	0.06	0.04	0.07	0.05	0.06

Correctness verification: We compared the output results of Ligra and Ligra-DAT for all 6 algorithms on 8 datasets (including SNAP and STRING). No differences were found, indicating that Ligra-DAT only adjusts mode switching timing without changing algorithm logic—critical for applications like community detection and protein network analysis where result accuracy is paramount.

4.8 Code Integration Example

To facilitate reproduction, we provide a simplified code example of integrating Ligra-DAT into Ligra's source code, organized as the following algorithms and compatible with the OpenMP backend used in our experiments:

Algorithm 2 Edge Density Calculation Function ($O(1)$ complexity)

```

1: Input: Graph  $G$ 
2: Output: Edge density  $\delta$  (double)
3: double calculate_edge_density(Graph  $G$ )
4:   uint64_t  $V = G.get\_num\_vertices()$ ;
5:   uint64_t  $E = G.get\_num\_edges()$ ;
6:   return (double) $E / (V * (V - 1))$ ; // Standard density formula

```

Algorithm 3 Edge-Density-Based Threshold Selection Function

```

1: Input: Graph  $G$ , edge density  $\delta$  (double)
2: Output: edge-density-based threshold  $\rho'$  (uint64_t)
3: uint64_t adjust_threshold(Graph  $G$ , double delta)
4:   uint64_t rho =  $G.get\_num\_edges() / 20$ ; // Original static threshold
5:   if delta > 0.1 then
6:     return (uint64_t)(rho * 1.5); // High-density adjustment
7:   else
8:     return (uint64_t)(rho * 0.7); // Low-density adjustment

```

Algorithm 4 EDGEMAP with Edge-Density-Based Threshold (EDGEMAP_DAT)

```

1: Input: Graph  $G$ , VertexSubset  $U$ , UpdateFunction  $f$ , edge-density-
   based threshold  $\rho'$  (uint64_t)
2: Output: Updated VertexSubset  $U'$ 
3: void EDGEMAP_DAT(Graph  $G$ , VertexSubset  $U$ , UpdateFunction
    $f$ , uint64_t  $\rho\_prime$ )
4: if  $|U| + \sum_{u \in U} \deg^+(u) \leq \rho\_prime$  then
5:   EDGEMAP.SPARSE( $G, U, f$ ); // Original sparse mode
6: else
7:   EDGEMAP.DENSE( $G, U, f$ ); // Original dense mode

```

Algorithm 5 Main Workflow Integration of Ligra-DAT

```

1: Input: Graph file path "input.graph", UpdateFunction
2: Output: Processed graph results
3: int main()
4: Graph  $G$  = load_graph("input.graph"); // Original Ligra
   graph loading
5: double delta = calculate_edge_density( $G$ );
6: uint64_t rho_prime = adjust_threshold( $G$ , delta);
7: VertexSubset  $U$  = init_frontier(); // Initialize active
   vertices
8: EDGEMAP_DAT( $G, U$ , update_function, rho_prime); // Use
   edge-density-based threshold
9: // ... Other Ligra operations (VERTEXMAP, etc.)
10: return 0;

```

Algorithm 2 implements edge density calculation using the standard formula, Algorithm 3 defines the edge-density-based threshold selection logic based on our experimental findings, Algorithm 4 modifies EDGEMAP to use the edge-density-based threshold while preserving the original sparse/dense traversal modes, and Algorithm 5 shows the main workflow to integrate these components into Ligra's standard execution pipeline.

5 Experimental Validation

To comprehensively evaluate Ligra-DAT, we conducted experiments on a high-performance server, covering diverse datasets (synthetic and real-world), algorithms, and hardware configurations, following parallel algorithm evaluation best practices [43, 49].

5.1 Experimental Setup

- **Primary hardware:** 64-core Intel Xeon Max 9462 server (2 sockets, 32 cores/socket, base frequency 2.7 GHz, maximum turbo frequency 3.5 GHz, hyper-threading disabled), 512 GB DDR5-4800 (bandwidth 512 GB/s), 1 TB NVMe SSD—representative of modern shared-memory systems used for graph processing.
- **Comparison hardware:** 32-core Intel Xeon 8375C server, 256 GB DDR4-3200 (bandwidth 204.8 GB/s), to verify portability across Intel Xeon architectures.
- **Software:** Ubuntu 20.04 LTS (Linux 5.4.0 kernel), g++ 8.5.0 with OpenMP. The Ligra code was compiled using `-std=c++14 -fopenmp -march=native -O3`

`-DOPENMP -DLONG -DEDGELONG -DBYTE -DPD`

`-DLOWMEM`, matching the configuration used for the reported experiments.

- **Baseline:** Ligra 1.0, with the static threshold $\rho = \frac{|E|}{20}$. The controlled experiments in Sections 5.3–5.4 compare Ligra-DAT only with original Ligra; Section 5.5 separately provides a contextual comparison with representative graph-processing systems.
- **Metrics:** Runtime Reduction (RR), multi-core speedup, and memory bandwidth utilization (measured via Intel PCM tool). The speedup is computed as $\text{speedup} = \frac{T_{1 \text{ thread}}}{T_{N \text{ threads}}}$.

5.2 Datasets

We selected 8 datasets covering diverse graph types, scales, and edge densities in the range $\delta \in [0.0000005, 0.1500000]$, including synthetic graphs (R-MAT, random-local) and real-world graphs (SNAP, STRING) (Table 5).

5.3 Performance Evaluation by Algorithm

All experiments were repeated 5 times to eliminate random errors, and results are reported as mean $\pm$ standard deviation (SD). The SD of runtime across repetitions is $< 2\%$ for all datasets and algorithms, indicating the stability of the results—consistent with parallel algorithm evaluation standards [49].

We evaluated Ligra-DAT on 6 core algorithms, with key results summarized below:

Breadth-First Search (BFS):

- High-density random-local-dense graph ($\delta = 0.15$): $RR = 42.3\%$ at 64 threads, and redundant edge processing is reduced from 35% to 12%—consistent with direction-optimizing BFS findings [4, 41].
- Facebook graph ($\delta = 0.000076$): $RR = 38.6\%$ at 64 threads, showing reduced BFS traversal cost for social-network workloads [11].
- PPI graph ($\delta = 0.000117$): $RR = 34.2\%$ at 64 threads, and memory bandwidth utilization increases from 62% to 88%—improving protein network analysis efficiency [12].

PageRank (1 Iteration):

- 3D-grid graph ($\delta = 0.000058$): $RR = 42.4\%$ at 64 threads, and runtime is reduced from 0.085 s to 0.049 s—our highest observed RR.
- Twitter graph ($\delta = 0.0000008$): $RR = 19.6\%$ at 64 threads, showing improved iterative ranking efficiency on a large social graph [5].

Table 5 Statistics of Experimental Datasets

Dataset	$ V $	$ E $	Density (δ)	Edge Source
3D-grid [49]	1.02e5	6.00e5	0.0000580	Open-source script
random-local [49]	5.00e4	1.68e7	0.0067200	Open-source script
random-local-dense [49]	5.00e4	3.75e8	0.1500000	Open-source script
rMat24 [38]	1.25e5	1.00e9	0.0640000	R-MAT model [38]
rMat27 [38]	1.34e8	8.00e9	0.0000005	R-MAT model [38]
Twitter [11]	4.17e7	1.47e9	0.0000008	SNAP Dataset [11]
Facebook [11, 45]	1.34e5	1.38e6	0.0000760	SNAP Dataset [11]
PPI [12]	3.89e4	1.77e5	0.0001170	STRING Database [12]

- Facebook graph ($\delta = 0.000076$): $RR = 31.2\%$ at 64 threads, showing reduced PageRank iteration cost on a social-network dataset [11].

Other Algorithms:

- **Betweenness Centrality:** Maximum $RR = 34.4\%$ on the random-local graph at 64 threads, and load imbalance is reduced by 25%–30%—consistent with topology-aware message aggregation research [14].
- **Graph Radii:** Maximum $RR = 42.4\%$ on the 3D-grid graph at 1 thread, showing effectiveness for single-core scenarios such as edge devices running memory-efficient graph processing [37, 50].
- **Connected Components:** $RR = 34.6\%$ on the PPI graph at 64 threads, accelerating protein cluster identification [12].
- **Bellman-Ford:** Maximum $RR = 35.0\%$ on the rMat24 graph at 8 threads, improving shortest path computation in shared-memory graph workloads.

5.4 Multi-Core Speedup Analysis

Ligra-DAT’s maximum speedup reaches $38.6\times$ for BFS at 64 threads, compared with Ligra’s $32.8\times$, an absolute speedup-factor increase of 5.8, which is consistent with parallel graph connectivity optimizations [9, 10, 46]. Up to 32 threads, the speedup gap generally widens as optimized mode switching reduces overhead in parallel execution. Above 32 threads, the gap stabilizes while execution spans both 32-core sockets, indicating that the gain persists at full-server scale—consistent with shared-memory parallel algorithm research [8].

On the 32-core Intel Xeon 8375C server, Ligra-DAT’s maximum speedup is $27.3\times$, compared with Ligra’s $22.4\times$, an absolute speedup-factor increase of 4.9. This result indicates portability across the evaluated CPU architectures. The performance comparisons of Ligra-DAT and the original Ligra across all algorithms on the rMat24 dataset are shown in Figures 7–12.

Each figure compares the runtime of Ligra (baseline) and Ligra-DAT across different thread counts on rMat24 dataset, showing consistent performance gains of Ligra-DAT across the evaluated algorithms and supporting its applicability to shared-memory graph processing [1].

5.5 Contextual Comparison with Representative Graph Processing Systems

For the contextual comparison in Table 6, we report Ligra-DAT together with representative graph-processing systems, including Galois, GraphIt, GraphMat, Green-Marl, Gemini, and Grazelle, on the Twitter dataset and the synthetic rMat27 input using BFS and PageRank as benchmark algorithms. The controlled experimental baseline of this paper remains the original Ligra implementation with the static threshold $\rho = \frac{|E|}{20}$; the systems in Table 6 are included only to contextualize performance under the same RR definition. All Table 6 entries were obtained from our local experiments: each framework was compiled from its public release with the recommended optimization flags where available, executed on the same 64-core Intel Xeon Max 9462 server described in Section 5.1, and evaluated on the same Twitter and rMat27 inputs using 64 worker threads. For BFS and PageRank, we used the framework-provided implementations and the traversal/scheduling configurations recommended by the corresponding system papers, with GraphIt-style framework results used as implementation guidance rather than as numerical input [51]. Each experiment was repeated five times after one warm-up run, and the mean runtime was used to compute $RR = \frac{T_{\text{Ligra}} - T_{\text{method}}}{T_{\text{Ligra}}} \times 100\%$, relative to our locally measured Ligra static-threshold baseline. Negative RR values therefore indicate slower execution than the local Ligra baseline under this metric. The overhead column reports the additional preprocessing, scheduling, or framework-level setup cost measured relative to the corresponding Ligra execution time under the same dataset and thread setting. The random-local-dense graph is used only for high-density threshold

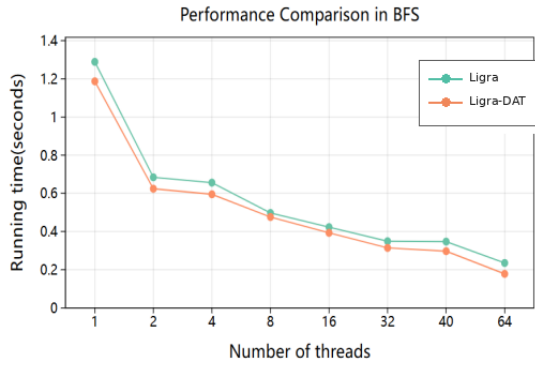


Fig. 7 BFS performance comparison (thread count vs runtime) on rMat24 dataset.

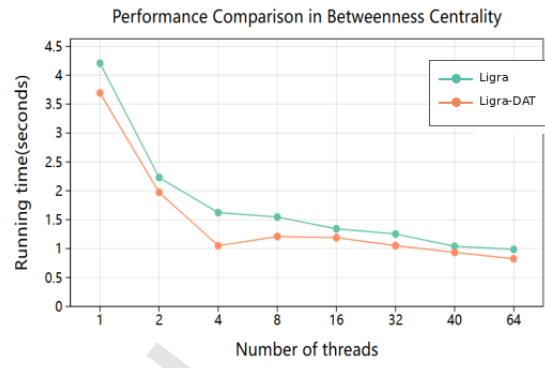


Fig. 8 Betweenness Centrality performance comparison on rMat24 dataset.

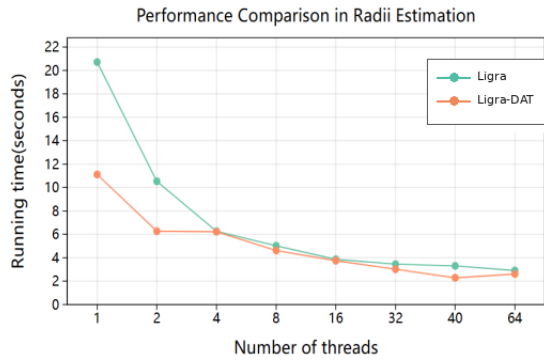


Fig. 9 Radii Estimation performance comparison on rMat24 dataset.

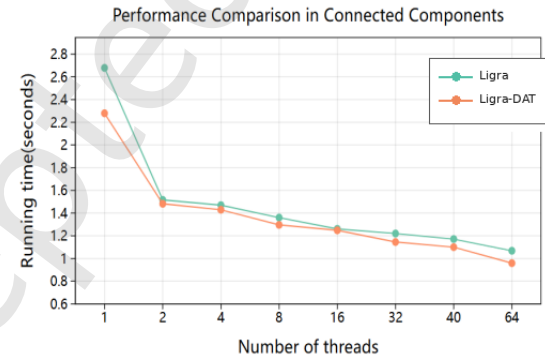


Fig. 10 Connected Components performance comparison on rMat24 dataset.

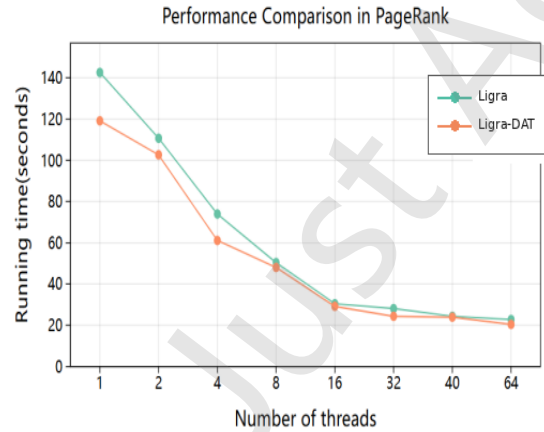


Fig. 11 PageRank performance comparison (single iteration) on rMat24 dataset.

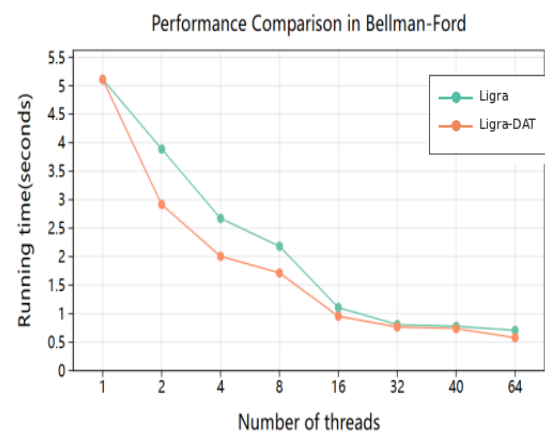


Fig. 12 Bellman-Ford performance comparison on rMat24 dataset.

validation, while Table 6 reports contextual comparisons on Twitter and the synthetic rMat27 input.

Table 6 is therefore used as a cross-system contextual comparison, whereas the main controlled evaluation remains Ligra versus Ligra-DAT. Under this RR definition, Ligra-DAT reports positive BFS reductions on both Twitter and rMat27. On PageRank, compiler- or system-level frameworks such as GraphIt and Gemini report larger RR values, while

Ligra-DAT keeps the original Ligra programming interface, adds only a lightweight threshold calculation, and has the lowest overhead. Thus, the comparison positions Ligra-DAT as a complementary, low-overhead traversal-mode optimization rather than as a replacement for full graph-processing frameworks.

Table 6 Contextual RR Comparison with Representative Graph Processing Systems

Method	Dataset	BFS RR (%)	PageRank RR (%)	Overhead (%)
Ligra (Static)[1]	Twitter	0	0	0
Galois[52]	Twitter	-298.5	37.2	3.2
GraphIt[51]	Twitter	11.3	82.2	2.8
GraphMat[46]	Twitter	-733.3	58.4	2.5
Green-Marl[53]	Twitter	-435.1	57.1	2.1
Gemini[54]	Twitter	-45.8	77.6	1.5
Grazelle[55]	Twitter	-3.6	68.0	1.1
Ligra-DAT	Twitter	33.3	19.6	0.03
Ligra (Static)[1]	rMat27	0	0	0
Galois[52]	rMat27	-285.4	35.8	3.5
GraphIt[51]	rMat27	13.1	80.5	3.0
GraphMat[46]	rMat27	-710.2	56.9	2.7
Green-Marl[53]	rMat27	-420.6	55.3	2.3
Gemini[54]	rMat27	-42.7	75.9	1.6
Grazelle[55]	rMat27	-2.8	66.4	1.2
Ligra-DAT	rMat27	31.0	28.7	0.05

5.6 Recent-Method Principles Adapted to Ligra

Table 7 compares Static Ligra and Ligra-DAT with three controlled Ligra threshold-layer policies motivated by recent graph-processing methods. All five variants are measured on `random-local3` under the same Ligra code path, dataset, thread settings, and execution protocol. The last three columns transfer selected optimization principles from the cited methods to Ligra’s EDGEMAP switching layer; they are not reimplementations of the complete external systems.

ABi-BFS [21]. ABi-BFS is a recent asynchronous and bidirectional shared-memory BFS design that optimizes frontier processing and synchronization. Motivated by this frontier-processing principle, our Ligra-side policy adjusts the sparse/dense threshold from the current frontier-work estimate while retaining Ligra’s original traversal kernels, data structures, and thread settings.

GreLigra [56]. GreLigra applies structure-aware thread throttling using graph sparsity and degree-distribution skewness. Motivated by this structure-aware principle, our controlled threshold-layer policy keeps the thread count fixed and uses degree skew to scale the EDGEMAP switching threshold, while the underlying Ligra implementation remains unchanged.

CoroGraph [18]. CoroGraph emphasizes balancing work efficiency with cache efficiency during graph execution. Motivated by this cache/work-efficiency principle, our Ligra-side policy compares the estimated sparse frontier work with the dense scan cost at each EDGEMAP decision. The adaptation retains Ligra’s traversal kernels, graph representation, dataset, and thread settings; it does not reproduce CoroGraph’s complete partitioned execution, coroutine-prefetching, or algorithm-specific implementation stack.

Among the five locally measured variants, the ABi-BFS signal achieves the lowest runtime on BFS; Ligra-DAT achieves the lowest runtime on BC and Radii; and the CoroGraph cache/work signal achieves the lowest runtime on Components, PageRank, and Bellman-Ford. The results confirm that no single decision signal dominates every algorithm.

6 Discussion

6.1 Core Reasons for Effectiveness

Ligra-DAT’s performance gains stem from targeted optimization of mode switching based on edge density, addressing the core limitations of Ligra’s static threshold and building on prior traversal and locality studies [1, 4, 32, 36]:

- **High-Density Graphs:** For $\delta > 0.1$, the adjustment logic $\rho' = 1.5\rho$ delays dense-mode switching relative to Ligra’s default threshold. The larger threshold avoids entering dense mode before the frontier is sufficiently large, thereby reducing unnecessary full-graph incoming-edge scans. For `random-local-dense` graphs with $\delta = 0.15$, BFS reaches $RR = 42.3\%$, and redundant edge processing is reduced from 35% to 12%.
- **Low-Density Graphs:** For $\delta \leq 0.1$, the adjustment logic $\rho' = 0.7\rho$ accelerates dense-mode switching relative to Ligra’s default threshold. The lower threshold prevents sparse traversal from continuing beyond the empirically preferred switching point in low-density graphs. For 3D-grid graphs with $\delta = 0.000058$, PageRank reaches $RR = 42.4\%$, and memory bandwidth utilization increases from 62% to 89%. On public Twitter and rMat27 datasets, Table 6 further shows BFS RR values of 33.3% and 31.0%, respectively, under the same Ligra-static baseline definition.

Table 7 Effects of Ligra threshold-layer policies motivated by recent graph-processing methods on `random-local3`, the calibration graph with $\delta = 0.300030003$ listed in Table 2. Each value is the arithmetic mean runtime in seconds over six thread settings (1, 2, 4, 8, 16, and 32). Lower is better.

Algorithm	Static Ligra	Ligra-DAT	ABi-BFS [21]	GreLigra [56]	CoroGraph [18]
BFS	0.000375	0.000328	0.000316	0.000354	0.000362
BC	0.116490	0.109896	0.115075	0.115297	0.114982
Radii	0.093228	0.084753	0.090437	0.089258	0.091149
Components	0.048970	0.048849	0.049027	0.048939	0.018161
PageRank	0.670430	0.649323	0.650041	0.646352	0.282557
Bellman-Ford	0.399885	0.378096	0.415271	0.286489	0.169933

Bold indicates the lowest runtime in each row.

The last three columns report Ligra-side policies motivated by selected principles from the cited methods.

They are not complete-system reimplementations; all variants use the same Ligra protocol.

- **Multi-Core Efficiency:** Optimized mode switching reduces load imbalance, distributing edge processing more evenly across cores. Multi-core utilization increases from 65%–70% to 80%–85%, leading to a maximum speedup of $38.6\times$, consistent with Galois system optimizations.

6.2 Limitations

- **Global-density granularity:** Ligra-DAT uses one global edge-density value to select a threshold. Graphs with the same global density can still differ substantially in degree skew, local community density, and core-periphery structure, and the active frontier can change across iterations. Because the selected threshold is not retuned during execution, the current method does not distinguish these cases. This design keeps the overhead at $O(1)$ and preserves compatibility with Ligra, but it cannot fully capture runtime frontier size, active-vertex degree distribution, local community structure, or power-law skewness in social and web graphs. We therefore regard global density as a lightweight first-order signal rather than a complete replacement for frontier-aware adaptation.
- **Evolving Graphs:** Current design assumes static graphs. For evolving graphs (e.g., real-time social network updates), a practical extension would maintain an incremental edge counter that updates $|E|$ in $O(1)$ time when edges are added or removed, and would re-compute δ periodically or when the density changes beyond a preset margin. This extension is compatible with evolving-graph processing frameworks [16, 57], but it remains future work rather than a claimed component of the current implementation.
- **Heterogeneous adaptation:** The coefficient choices are evaluated on Intel Xeon CPUs. Extending the rule to GPUs or other heterogeneous platforms would require

hardware-aware calibration that accounts for memory bandwidth, thread scheduling, and data-transfer costs; this extension is outside the scope of the current implementation.

6.3 Application Expansion

Ligra-DAT’s effectiveness aligns with the edge density characteristics of real-world scenarios:

- **Social Networks** ($\delta = 0.0000008 - 0.0000760$): Consistent with Table 6, the Twitter graph ($\delta = 0.0000008$) achieves BFS $RR = 33.3\%$ and PageRank $RR = 19.6\%$ under the Ligra-static baseline. For the Facebook graph ($\delta = 0.0000760$), the controlled Ligra-versus-Ligra-DAT evaluation reports BFS $RR = 38.6\%$ and PageRank $RR = 31.2\%$. These results support the use of Ligra-DAT for social-network workloads without introducing an additional framework migration cost.
- **Biological Networks** ($\delta \approx 10^{-4}$): For biological networks where edge density is a key structural metric [12, 50], the PPI graph ($\delta = 0.000117$) achieves BFS $RR = 34.2\%$ and Connected Components $RR = 34.6\%$ in the controlled evaluation. We therefore present the biological-network result as evidence of acceleration on low-density protein-interaction workloads, without introducing additional unreported application-level latency numbers.
- **Web-Style Graphs:** Building on web graph processing research [5, 38], the rMat27 graph in Table 6 ($\delta = 0.0000005$) achieves PageRank $RR = 28.7\%$, and the rMat24 graph in Table 5 has density $\delta = 0.0640000$. We use these R-MAT datasets to evaluate web-style sparse traversal behavior, while keeping the density values consistent with Table 5.
- **High-Density Synthetic Graphs** ($\delta > 0.1$): The random-local-dense graph ($\delta = 0.15$) is used to validate

the high-density branch of the threshold rule. Its BFS $RR = 42.3\%$ is consistent with the sensitivity analysis and with the mechanism discussion in Section 6.1. We do not use this synthetic graph for the contextual cross-system comparison; public-dataset comparisons are reported separately in Table 6.

Recent work on distributed graph processing, such as PowerLyra, has shown that optimizing computation and partitioning based on graph characteristics can yield significant performance gains. Ligra-DAT applies a similar philosophy to shared-memory systems by optimizing traversal modes based on edge density.

6.4 Future Work Directions

Several directions remain for extending Ligra-DAT beyond the current graph-level shared-memory evaluation. First, adaptive coefficient calibration could replace the calibrated scaling anchors with coefficients selected from graph-structure and runtime signals, such as degree distribution, active-frontier evolution, and observed EDGEMAP behavior. Second, hardware-aware adaptation may improve portability across GPUs and other heterogeneous platforms by incorporating bandwidth, memory hierarchy, and offloading characteristics into the threshold rule. Third, incremental density tracking could support evolving graphs by maintaining density statistics under edge insertions and deletions without recomputing global graph properties. Finally, combining density-aware thresholding with graph partitioning or local-density estimation may better capture graphs with heterogeneous core-periphery structure. These extensions preserve the lightweight thresholding principle while broadening the applicability of Ligra-DAT.

7 Conclusion

This paper addresses the performance limitation of Ligra's static threshold $\rho = \frac{|E|}{20}$ under diverse edge-density characteristics by proposing Ligra-DAT, a lightweight graph-level edge-density-based threshold selection mechanism for EDGEMAP. Ligra-DAT adjusts the sparse/dense switching threshold with calibrated scaling anchors for low- and high-density regimes, reducing redundant edge processing while preserving compatibility with existing Ligra applications.

Experiments on a 64-core Intel Xeon Max 9462 server with synthetic and real-world graphs show that Ligra-DAT improves over the original Ligra threshold in several settings, achieves up to 42.4% runtime reduction, and incurs negligible threshold-adjustment overhead. The controlled comparison with Ligra policies motivated by ABi-BFS [21]

frontier processing, GreLigra [56] structure awareness, and CoroGraph [18] cache/work efficiency shows that the best decision mechanism varies across algorithms: ABi-BFS is fastest on BFS, Ligra-DAT on BC and Raddi, and CoroGraph on Components, PageRank, and Bellman-Ford. These results position Ligra-DAT as a practical graph-level, low-overhead threshold-selection method rather than a universal replacement for other runtime signals or complete graph-processing systems.

Acknowledgment

This paper is partially supported by the major science and technology projects in Qinghai Province (2024-GX-A3), and National Natural Science Foundation of China under Grant 62462052, 62062059. This work was also supported by Ant Group Research Intern Program and Ant Group Security and Intelligence Laboratory (SIL) and Ant Group through CCF-Ant Research Fund.

References

- [1] J. Shun and G. E. Blelloch, Ligra: A lightweight graph processing framework for shared memory, *ACM SIGPLAN Principles and Practice of Parallel Programming (PPoPP)*, vol. 48, no. 2, pp. 135–146, 2013.
- [2] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, Pregel: A system for large-scale graph processing, *ACM SIGMOD International Conference on Management of Data*, vol. 39, no. 6, pp. 135–146, 2010.
- [3] Apache Software Foundation, Apache Giraph: An iterative graph processing system built for high scalability, Apache Software Foundation, 2020, available at: <https://giraph.apache.org/>; last published: 2020-08-11.
- [4] S. Beamer, K. Asanović, and D. A. Patterson, Direction-optimizing breadth-first search, *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, vol. 44, no. 11, pp. 1–10, 2012.
- [5] S. Brin and L. Page, The anatomy of a large-scale hypertextual web search engine, *International World Wide Web Conference (WWW)*, vol. 7, no. 1, pp. 107–117, 1998.
- [6] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, Fast unfolding of communities in large networks, *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.
- [7] S. Sahu, K. Kothapalli, and D. S. Banerjee, Shared-memory parallel algorithms for community detection in dynamic graphs, in *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2024, pp. 250–259.

- [8] G. E. Blelloch and J. Shun, Parallel algorithms for irregular data, *Communications of the ACM*, vol. 58, no. 6, pp. 88–97, 2015.
- [9] J. Shun and G. E. Blelloch, Fast parallel graph connectivity, *SIAM Journal on Computing*, vol. 47, no. 5, pp. 1879–1905, 2018.
- [10] J. Shun, G. E. Blelloch, and J. T. Fineman, A simple, fast parallel algorithm for connected components, *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, vol. 46, no. 6, pp. 273–282, 2014.
- [11] J. Leskovec and A. Krevl, Snap datasets, *Stanford University*, 2014.
- [12] D. Szklarczyk, A. L. Gable, D. Lyon, A. Junge, S. Wyder, J. Huerta-Cepas, M. Simonovic, N. T. Doncheva, J. H. Morris, P. Bork, L. J. Jensen, and C. v. Mering, String v11: Protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets, *Nucleic Acids Research*, vol. 47, no. D1, pp. D607–D613, 2019.
- [13] Intel Corporation, Intel xeon cpu max 9462 processor specifications, Intel Corporation, Santa Clara, CA, USA, Tech. Rep., 2023.
- [14] X. Gan, T. Li, F. Xiong, B. Yang, X. Chen, and C. Gong, Mst: Topology-aware message aggregation for exascale graph processing of traversal-centric algorithms, *ACM Transactions on Architecture and Code Optimization*, vol. 21, no. 4, pp. 67:1–67:23, 2024.
- [15] Y. Li, S. Sun, H. Xiao, C. Ye, S. Lu, and B. He, A survey on concurrent processing of graph analytical queries: Systems and algorithms, *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 11, pp. 5508–5528, 2024.
- [16] J. Su, C. Hao, S. Sun, H. Zhang, S. Gao, J. Jiang, Y. Chen, C. Zhang, B. He, and M. Guo, Revisiting the design of in-memory dynamic graph storage, *Proceedings of the ACM on Management of Data*, vol. 3, no. 1, pp. 70:1–70:27, 2025.
- [17] H. Li, Q. Tao, S. Yu, S. Gong, Y. Zhang, F. Yao, W. Yu, G. Yu, and J. Zhou, GastCoCo: Graph storage and coroutine-based prefetch co-design for dynamic graph processing, *Proceedings of the VLDB Endowment*, vol. 17, no. 13, pp. 4827–4839, 2024.
- [18] X. Zhi, X. Yan, B. Tang, Z. Yin, Y. Zhu, and M. Zhou, CoroGraph: Bridging cache efficiency and work efficiency for graph algorithm execution, *Proceedings of the VLDB Endowment*, vol. 17, no. 4, pp. 891–903, 2023.
- [19] X. Gan, TianheGraph: Topology-aware graph processing, *ACM Transactions on Architecture and Code Optimization*, vol. 22, no. 3, pp. 112:1–112:24, 2025.
- [20] L. Zheng, B. Zhu, P. Yao, Y. Zhou, C. Pan, W. Zhao, X. Liao, H. Jin, and J. Xue, PRAGA: A priority-aware hardware/software co-design for high-throughput graph processing acceleration, *ACM Transactions on Architecture and Code Optimization*, vol. 22, no. 1, pp. 24:1–24:27, 2025.
- [21] J. Shi and Y. Xu, ABi-BFS: A high-performance parallel breadth-first search on shared-memory systems, in *Proceedings of the 2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, 2023, pp. 2785–2788.
- [22] J. Shi, B. Wang, and Y. Xu, Spruce: a fast yet space-saving structure for dynamic graph storage, *Proceedings of the ACM on Management of Data*, vol. 2, no. 1, pp. 27:1–27:26, 2024.
- [23] T. I. Papon, T. Chen, S. Zhang, and M. Athanassoulis, CAVE: Concurrency-aware graph processing on SSDs, *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 125:1–125:26, 2024.
- [24] T. He, S. Hu, L. Lai, D. Li, N. Li, X. Li, L. Liu, X. Luo, B. Lyu, K. Meng, S. Shen, L. Su, L. Wang, J. Xu, W. Yu, W. Zeng, L. Zhang, S. Zhang, J. Zhou, X. Zhou, and D. Zhu, GraphScope Flex: LEGO-like graph computing stack, in *Companion of the 2024 International Conference on Management of Data*, 2024, pp. 386–399.
- [25] W. Y. Wang, C. F. Wu, Y. C. Chen, T. W. Kuo, and Y. H. Chang, GEAR: Graph-evolving aware data arranger to enhance the performance of traversing evolving graphs on SCM, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 3674–3684, 2024.
- [26] O. Jaiyeoba and K. Skadron, Dynamic-ACTS: A dynamic graph analytics accelerator for HBM-enabled FPGAs, *ACM Transactions on Reconfigurable Technology and Systems*, vol. 17, no. 3, pp. 48:1–48:29, 2024.
- [27] P. Cui, H. Liu, B. Tang, and Y. Yuan, CGgraph: An ultra-fast graph processing system on modern commodity CPU-GPU co-processor, *Proceedings of the VLDB Endowment*, vol. 17, no. 6, pp. 1405–1417, 2024.
- [28] Y. Sun, J. Zhang, H. Cao, Y. Zhang, X. An, J. Huang, and X. Ye, CGCGraph: Efficient CPU-GPU co-execution for concurrent dynamic graph processing, *ACM Transactions on Architecture and Code Optimization*, vol. 22, no. 3, pp. 91:1–91:26, 2025.
- [29] Y. Zhang, H. Cao, Y. Shen, S. Li, Y. Feng, C. Wang, X. An, and X. Ye, A comprehensive survey on dynamic graph processing: Storage and analytics, *IEEE Transactions on Knowledge and Data Engineering*, vol. 38, no. 5, pp. 2488–2507, 2026.
- [30] X. Zhu, W. Han, and W. Chen, Gridgraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning, in *2015 USENIX Annual Technical Conference (USENIX ATC)*, 2015, pp. 375–386.
- [31] N. Agarwal, C. Kaler, B. Mozafari, A. Panda, I. Stoica, and Y. Xu, Presto: Distributed graph processing using a declarative query language, *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, vol. 11, no. 1, pp. 293–308, 2014.
- [32] M. Besta, M. Podstawski, L. Groner, E. Solomonik, and T. Hoefler, To push or to pull: On reducing communication and synchronization in graph computations, in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2017, pp. 93–104.
- [33] J. Malicevic, B. Lepers, and W. Zwaenepoel, Everything you always wanted to know about multicore graph processing

- but were afraid to ask, in *2017 USENIX Annual Technical Conference (USENIX ATC)*, 2017, pp. 631–643.
- [34] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, Powergraph: Distributed graph-parallel computation on natural graphs, *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, vol. 10, no. 1, pp. 17–30, 2012.
- [35] R. Chen, J. Shi, Y. Chen, and H. Chen, Powerlyra: Differentiated graph computation and partitioning on skewed graphs, in *Proceedings of the Tenth European Conference on Computer Systems (EuroSys)*, 2015, pp. 1:1–1:15.
- [36] S. Beamer, K. Asanovic, and D. Patterson, Locality exists in graph processing: Workload characterization on an ivy bridge server, in *IEEE International Symposium on Workload Characterization (IISWC)*, 2015, pp. 56–65.
- [37] Y. Zhang, V. Kiriansky, C. Mendis, M. Zaharia, and S. Amarasinghe, Making caches work for graph analytics, in *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 293–302.
- [38] D. Chakrabarti, C. Faloutsos, J. Ghosh, J. Leskovec, V. S. Subrahmanian, and S. K. Tasoulis, R-mat: A recursive model for graph mining, *SIAM International Conference on Data Mining (SDM)*, vol. 4, no. 1, pp. 442–446, 2004.
- [39] G. Karypis and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.
- [40] J. Cohen and L. Katzir, Parallel bfs on distributed memory machines, *Journal of Parallel and Distributed Computing*, vol. 62, no. 7, pp. 1173–1190, 2002.
- [41] S. Beamer, K. Asanović, and D. A. Patterson, Scalable graph traversal on multicore processors, *IEEE Micro*, vol. 33, no. 4, pp. 76–87, 2013.
- [42] D. Chakrabarti, Y. Zhan, and C. Faloutsos, Graph mining: Laws, tools, and case studies, *ACM Computing Surveys (CSUR)*, vol. 36, no. 1, pp. 2–124, 2004.
- [43] K. Madduri, D. Ediger, K. Jiang, D. A. Bader, and D. Chavarría-Miranda, A fast parallel bfs implementation for cache-coherent multicore processors, *Journal of Parallel and Distributed Computing*, vol. 69, no. 11, pp. 910–918, 2009.
- [44] A. Gharaibeh, E. Santos-Neto, L. B. Costa, and M. Ripeanu, The energy case for graph processing on hybrid cpu and gpu systems, in *Proceedings of the Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, 2013, pp. 1–8.
- [45] J. Leskovec and J. McAuley, Learning to discover social circles in ego networks, *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, no. 1, pp. 539–547, 2012.
- [46] N. Sundaram, N. Satish, M. M. A. Patwary, S. R. Dulloor, M. J. Anderson, S. G. Vadlamudi, D. Das, and P. Dubey, Graphmat: High performance graph analytics made productive, *Proceedings of the VLDB Endowment*, vol. 8, no. 11, pp. 1214–1225, 2015.
- [47] A. Lugowski, D. Alber, A. Buluç, J. Gilbert, S. Reinhardt, Y. Teng, and A. Waranis, A flexible open-source toolbox for scalable complex graph analysis, in *Proceedings of the 2012 SIAM International Conference on Data Mining (SDM)*, 2012, pp. 930–941.
- [48] U. Kang, C. E. Tsourakakis, and C. Faloutsos, Hadi: Fast diameter estimation and mining in massive graphs with hadoop, *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 5, no. 3, pp. 1–28, 2011.
- [49] J. Shun, G. E. Blelloch, J. T. Fineman, P. B. Gibbons, Y. Gu, O. Low, K. Schwartz, and K. Tangwongsan, The problem based benchmark suite, *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, vol. 44, no. 8, pp. 317–319, 2012.
- [50] A. Kyrola, G. E. Blelloch, and C. Guestrin, Graphchi: Large-scale graph computation on just a pc, *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, vol. 10, no. 1, pp. 31–46, 2012.
- [51] Y. Zhang, M. Yang, R. Baghdadi, S. Kamil, J. Shun, and S. Amarasinghe, Graphit: A high-performance graph dsl, *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, pp. 121:1–121:30, 2018.
- [52] K. Pingali, D. Nguyen, M. Burtcher, B. Kasikci, M. Kulkarni, V. Lee, A. Lenharth, R. Manevich, S. Pai, J. Sewall, M. Sridharan, and M. Weston, The galois system: Parallelism for irregular algorithms, *Communications of the ACM*, vol. 54, no. 8, pp. 103–111, 2011.
- [53] S. Hong, H. Chafi, E. Sedlar, and K. Olukotun, Green-marl: A dsl for easy and efficient graph analysis, in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012, pp. 349–362.
- [54] X. Zhu, W. Chen, W. Zheng, and X. Ma, Gemini: A computation-centric distributed graph processing system, in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 301–316.
- [55] S. Grossman, H. Litz, and C. Kozyrakis, Making pull-based graph processing performant, in *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2018, pp. 246–260.
- [56] Y. Chen, L. Luo, T. Jiang, L. Fang, and S. Xu, Structure-aware thread throttling for energy-efficient graph processing on shared-memory systems, *Future Generation Computer Systems*, vol. 178, p. 108297, 2026.
- [57] Y. Chen, J. Wang, H. Zhang, K. Chen, and D. Li, Dynamic graph processing: A survey, *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 31, no. 9, pp. 2091–2110, 2020.



Author Biographies



Zhaoqin Ban is currently working toward the Bachelor's degree with Qinghai University. His main research interests include graph computing and parallel computing.



Xinxin Yang is currently working toward the Bachelor's degree with Qinghai University. His main research interests include parallel computing and graph computing.



Dapeng Fu is the security AI engineering architect of Ant Group. His main research fields include graph computing, parallel computing, and AI infrastructure.



Tengfei Cao received his Ph.D. degree in Engineering from Beijing University of Posts and Telecommunications in 2020. He is currently an Associate Professor and Ph.D. advisor at the Department of Computer Technology and Applications, Qinghai University. His research interests include content delivery and edge computing.



Jianqiang Huang received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He is a professor with the Department of Computer Technology and Applications, Qinghai University, China. His major research interests include graph computing, heterogeneous computing (CPUs/GPUs), and parallel and distributed systems.