A Comparative Study of Ligra and Ligra+: Shared-Memory Graph Processing with and without Compression

Your Name ✉
Your Institution

Abstract

Today's shared-memory machines offer hundreds of gigabytes to several terabytes of RAM, making it possible to process graphs with billions of edges on a single host. Despite this, many graph algorithms remain bottlenecked by memory bandwidth rather than computation. Ligra addresses this imbalance through a lightweight, data-parallel interface built around two primitives—`VERTEXMAP` and `EDGEMAP`—that dynamically switch between sparse and dense traversal modes. Ligra+ extends this design with compression techniques such as delta encoding and compact variable-byte formats, reducing memory footprint while improving throughput on multi-core systems. This report synthesizes the central ideas of both frameworks, reconstructs their algorithmic design principles, contrasts their empirical performance, and analyzes how compression reshapes memory–computation trade-offs in shared-memory graph processing.

2012 ACM Subject Classification Theory of computation → Shared memory algorithms; Information systems → Graph-based data models

Keywords and phrases Ligra, Ligra+, shared-memory graph processing, compression, parallel algorithms

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Large graphs appear routinely in social networks, web infrastructure, biological interaction networks, and simulations of physical systems. While distributed graph-processing frameworks such as Pregel and GraphLab were historically dominant, modern commodity servers now provide enough RAM to store ten billion edges or more on a single machine. Shared-memory systems reduce communication delays, simplify programming, and often outperform distributed solutions for many workloads [5].

Ligra exploits this hardware trend by offering a minimal, efficient interface for data-parallel graph traversal. Rather than relying on message passing, Ligra organizes computations around dynamically changing *frontiers*—sets of currently active vertices. The system automatically switches between sparse and dense evaluation modes depending on frontier size.

Ligra+ extends the same interface but compresses adjacency lists using delta-encoded and variable-length formats [6]. Earlier compression systems achieved space savings at the cost of slower runtime [1]; Ligra+ demonstrates conditions under which compression yields speedups instead, particularly under memory bandwidth pressure on multi-core machines.

This report explains the design choices behind Ligra and Ligra+, analyzes their performance characteristics, and compares how compression modifies algorithmic behavior in shared-memory graph workloads.

2 Background and Graph Model

Both Ligra and Ligra+ operate on directed graphs $G = (V, E)$ where vertices are numbered $0, \ldots, |V| - 1$. For each vertex v, the tools store outgoing neighbors $N^+(v)$ and incoming neighbors $N^-(v)$ alongside their degrees [5]. Computation uses fork–join parallelism with atomic instructions such as compare-and-swap (CAS) ensuring correctness under concurrent writes.

The fundamental abstraction is the `vertexSubset`, which represents an active frontier either as a sparse list of vertex identifiers or as a dense bitmap. The representation is selected dynamically based on frontier size.

Ligra exposes two core parallel primitives:

- **VERTEXMAP(U, F):** Applies a Boolean function F to each $u \in U$ and returns those for which F succeeds.
- **EDGEMAP(G, U, F, C):** Applies an update function F to edges leaving vertices in U, generating candidate vertices, which are filtered by condition C.

`EDGEMAP` automatically selects sparse traversal over outgoing edges or dense traversal scanning all vertices and their incoming edges, depending on the number of active edges.

Ligra+ preserves this API but replaces raw adjacency lists with compressed, delta-encoded representations and partitions large neighbor lists for better load balancing [6].

3 Ligra

3.1 Motivation

Ligra aims to provide a lightweight shared-memory alternative to distributed frameworks. Systems like Pregel [4] or GraphLab [3] incur synchronization and communication overhead, whereas Ligra is optimized for uniform memory access with fast, shared RAM.

3.2 Core Techniques

Ligra's performance hinges on adaptively switching between sparse and dense modes when frontier size crosses a threshold proportional to $|E|$ [5]. Sparse traversal examines only outgoing edges from active vertices; dense traversal checks all vertices using incoming edges.

Graphs are stored in flat arrays for both in-edges and out-edges, minimizing overhead. The `vertexSubset` abstraction simplifies algorithms such as BFS and Brandes' betweenness centrality [2].

3.3 Applications

Common algorithms implemented in Ligra include:

- Breadth-First Search (BFS)
- Betweenness centrality via Brandes' algorithm
- Connected components
- Graph radius approximation
- PageRank
- Bellman–Ford shortest paths

All implementations retain textbook asymptotic complexity (e.g., BFS in $O(|V| + |E|)$) while achieving near-linear speedup up to 40 cores [5].

4 Ligra+

4.1 Motivation

On modern systems, graph processing is frequently memory-bandwidth-bound: fetching adjacency lists dominates runtime relative to arithmetic. Ligra+ tests whether compressed representations—decoded on the fly—reduce bandwidth consumption enough to offset decompression costs.

4.2 Compressed Representation

Adjacency lists in Ligra+ are sorted and stored as sequences of delta values encoded using:

- variable-byte codes,
- run-length–coded byte sequences,
- nibble (4-bit) codes.

Run-length encoding groups values requiring equal byte lengths, reducing branching during decoding [6].

Decoding uses two core operations:

- **FirstEdge**: decodes the first neighbor;
- **NextEdge**: decodes subsequent neighbors from the last offset.

High-degree vertices are partitioned into chunks of size at most T, allowing parallel decoding with independent starting points, improving load balance on multi-core hardware.

4.3 Empirical Behavior

Across real and synthetic datasets, Ligra+ uses roughly half the memory of Ligra. Compression benefits scale with repeating structural patterns.

Single-threaded runtime is often slightly slower due to decompression. However, on 40-core machines, byte-coded Ligra+ achieves:

- up to $2\times$ speedup,
- occasionally up to 10% slowdown,
- on average 14% faster than Ligra.

These gains appear when reduced memory traffic outweighs decoding overhead.

5 Comparative Analysis

5.1 Abstraction and API

Ligra and Ligra+ share the same programming interface; any algorithm written for Ligra runs on Ligra+ unchanged. The key difference lies entirely in edge storage and traversal cost.

5.2 Space–Time Trade-offs

Ligra maximizes per-edge traversal speed but consumes more memory. Ligra+ cuts memory by nearly half, often resulting in faster performance for memory-bound workloads or large core counts.

For small graphs or lightly threaded execution, Ligra often wins. For large graphs or memory-constrained settings, Ligra+ is typically superior.

5.3 Algorithmic Sensitivity

Algorithms like BFS that perform little computation per edge benefit most from compression. Algorithms with heavier per-edge arithmetic, such as PageRank or Bellman–Ford, may hide decoding overhead, sometimes making Ligra+ faster in practice.

6 Conclusion

Ligra demonstrates that a simple shared-memory framework can support a broad class of graph traversal algorithms both efficiently and expressively. Ligra+ extends this model by showing that lightweight compression can simultaneously reduce memory usage and improve performance on modern multicore processors. Together, both systems highlight the importance of data layout, memory bandwidth, and threading decisions in designing high-performance graph processing systems.

References

1 Daniel K. Blandford, Guy E. Blelloch, and Ian A. Kash. Compact representations of separable graphs. In *SODA*, 2003.

2 Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 2001.

3 Yucheng Low et al. Graphlab: A new parallel framework for machine learning. In *UAI*, 2010.

4 Grzegorz Malewicz et al. Pregel: a system for large-scale graph processing. In *SIGMOD*, 2010.

5 Julian Shun and Guy E. Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *PPoPP*. ACM, 2013.

6 Julian Shun, Laxman Dhulipala, and Guy E. Blelloch. Ligra+: graph processing with compression. In *Data Compression Conference*. IEEE, 2015.