# Parallel Graph Algorithms

GPU Acceleration with CUDA: Adaptive BFS & Afforest

Evangelos Moschou
`AEM: 10986`

**Abstract**

This report presents a comprehensive study of graph preprocessing techniques to improve performance of GPU-accelerated graph algorithms on billion-scale graphs. The primary focus is the evolution from standard Reverse Cuthill-McKee (RCM) reordering to a novel **Gap-Aware BFS** approach, which uniquely optimizes for both cache locality and delta-compression efficiency. We evaluate these techniques alongside Delta-Varint compression, Zero-Copy memory management, and adaptive kernels. Experimental results on Friendster (65.6M nodes, 3.6B edges) demonstrate that while traditional RCM degrades compression performance by 15-30%, our Gap-Aware BFS achieves a **9.2% speedup over the optimized baseline** (6.33s vs 6.91s) by preserving community structure during renumbering. This study demonstrates that preprocessing can resolve the antagonism between data locality and compression through careful design.

## 1 Introduction

Graph preprocessing is a critical but often overlooked component of high-performance graph analytics. The physical layout of vertices and edges in memory significantly affects cache utilization, TLB efficiency, and memory bandwidth requirements—key factors that determine whether algorithms are compute-bound or memory-bound on modern GPUs.

**This project's primary contribution is a systematic study of graph preprocessing techniques, with emphasis on the evolution from RCM to Gap-Aware BFS, and their impact on GPU-accelerated graph algorithms.** While we implement two fundamental algorithms (BFS and Afforest) as case studies, the main objective is to demonstrate that **preprocessing choices are algorithm-dependent** and must be co-designed with runtime optimizations.

We evaluate several preprocessing and optimization strategies:

- **Gap-Aware BFS**: A novel reordering strategy that sorts neighbors by original ID during layout construction to preserve compression-friendly deltas.
- **RCM Reordering**: Bandwidth minimization for improved spatial locality in uncompressed data.
- **Delta-Compressed CSR**: Varint encoding reducing data transfer by 37%
- **Zero-Copy Memory**: Streaming access to graphs exceeding VRAM
- **Adaptive Kernels**: Dynamic load balancing for irregular graphs

Our experimental results reveal a key insight: **Standard RCM and compression are antagonistic**, but **Gap-Aware BFS resolves this trade-off**. While RCM improves cache locality for uncompressed algorithms, it scatters IDs and ruins compression. In contrast, Gap-Aware BFS achieves the spatial locality of RCM while maintaining the small deltas required for efficient Varint encoding.

## 2 Algorithm & Data Structure Design

### 2.1 Data Representation: Compressed CSR & Zero-Copy

Standard CSR allows coalesced GPU access but bottlenecks PCIe bandwidth ( 13.5GB for Friendster vs 5.8GB VRAM). We implement **Zero-Copy Streaming** combined with **Delta-Varint Compression**:

- **Delta+Varint**: Neighbors are sorted, and differences ($\Delta$) are stored using variable-length integers (1-5 bytes). This reduces Friendster from 14.4GB to 9.1GB (37% reduction).
- **Streaming**: We use `cudaHostRegister` to pin host memory, allowing kernels to access data directly over PCIe (effective bw $\sim$1.4GB/s).

- **Decoding**: A warp-cooperative kernel decodes byte-streams in shared memory. Lane 0 decodes sequentially; all lanes process neighbors in parallel.

### 2.2 Adaptive BFS & Afforest

**Adaptive BFS** classifies nodes by degree: high-degree nodes use cooperative warp expansion to balance load, while low-degree nodes are processed per-thread. The compressed variant integrates on-the-fly decoding, masking ∼400ns PCIe latency with compute.

**Afforest** (Connected Components) is optimized for single-pass execution on small-diameter graphs. We added a **GCC Pruning** heuristic: before atomic linking, we check if both nodes already belong to the Giant Connected Component (root node 0). heuristic checks reduce atomic contention by 97% on Friendster, though the algorithm remains bandwidth-bound (∼21s).

## 3 Experimental Results

### 3.1 Test Environment

- **GPU**: NVIDIA GeForce RTX 3060 Laptop (Ampere, sm_86)
- **VRAM**: 5.8 GB
- **System RAM**: 32 GB
- **CUDA**: Version 12.5
- **Compiler**: nvcc with -O3, -arch=sm_86

### 3.2 Benchmark Datasets

**Friendster Social Network:**
- Vertices: 65,608,366
- Edges: 3,612,134,270 (3.6 billion)
- Edge data size: 13.46 GB (exceeds VRAM)
- Max degree: 5,214
- Graph diameter: 22
- Compressed size: 9.13 GB (1.48x ratio)

### 3.3 Performance Results

| **Algorithm** | **Mode** | **Time (ms)** | **Effective Speedup** |
|---|---|---|---|
| **Adaptive BFS** (Original) | Uncompressed | 12,009 | 1.0x |
| **Adaptive BFS** (Original) | **Compressed** | **6,908** | **1.74x** |
| **Adaptive BFS** (Gap-Aware) | **Compressed** | **6,334** | **1.90x** |
| **Afforest** | Uncompressed | 23,512 | 1.0x |
| **Afforest** | **Compressed** | **21,569** | **1.09x** |

Table 1: Final Performance on Friendster (100% Reachability). Gap-Aware BFS provides the optimal balance. All runs use Zero-Copy memory.

| **Metric** | **BFS (Gap-Aware Compressed)** | **Afforest (Compressed)** |
|---|---|---|
| Runtime | 6.33 s | 21.57 s |
| Memory (Host) | 9.13 GB | 9.13 GB |
| Effective Bandwidth | ∼1.44 GB/s | ∼0.42 GB/s |
| Correctness | 100% (verified) | 1 component (correct) |

Table 2: Detailed metrics for our best performing configurations on Friendster.

### 3.4 Analysis

The 37% data reduction translates to a **1.9x speedup for BFS** vs 1.09x for Afforest. BFS benefits more due to higher streaming density per frontier layer. Afforest's heavy atomic contention partially masks bandwidth gains, though GCC pruning helps. Both algorithms effectively hide the 400ns PCIe latency through massive thread-level parallelism.

## 4 Graph Preprocessing: RCM and Gap-Aware BFS

### 4.1 Motivation and Algorithm

The Reverse Cuthill-McKee (RCM) algorithm is a graph reordering technique that reduces the **bandwidth** of the adjacency matrix by renumbering vertices to place connected nodes close together in memory. This improves:

- **Cache locality**: Adjacent vertices in BFS trees map to nearby memory addresses
- **TLB hit rate**: Reduced page working set due to spatial clustering
- **Prefetcher efficiency**: Sequential access patterns enable hardware prefetching

**RCM Algorithm Overview.**

1. Select a peripheral node (low degree, far from graph center) as the starting vertex
2. Perform BFS traversal, visiting nodes level-by-level
3. Within each level, sort nodes by **increasing degree** (Cuthill-McKee)
4. **Reverse** the final ordering (Reverse Cuthill-McKee)

The reversal step places high-degree hub nodes at the beginning of the ordering, which empirically improves performance for many graph algorithms by prioritizing well-connected vertices early in traversals.

### 4.2 Implementation Details

Our CUDA implementation uses:

- **Parallel BFS**: GPU-based level-synchronous traversal for fast reordering
- **Radix Sort**: Per-level sorting by degree using CUB library primitives
- **Pseudo-Peripheral Search**: Heuristic to find optimal starting vertex

Reordering time for Friendster: ∼2.3 seconds for RCM, ∼18 minutes for Gap-Aware BFS (due to neighbor sorting).

### 4.3 Gap-Aware BFS: Resolving the Locality-Compression Trade-off

To address the performance degradation seen when combining RCM with compression, we implemented **Gap-Aware BFS**. The algorithm follows the standard BFS renumbering but introduces a critical modification: **During the neighbor enqueue stage, neighbors are sorted by their ORIGINAL ID before being assigned NEW IDs.**

This modification ensures that nodes that were originally part of the same community (and thus had close IDs) are assigned consecutive new IDs. This preserves the "Natural Community Order" benefits for Delta-Varint compression while obtaining the bandwidth reduction benefits of a global BFS ordering.

### 4.4 Experimental Results

We compare four graph orderings on Friendster:

1. **Original Order**: Natural edge insertion order from web crawling.
2. **RCM Order**: Bandwidth-minimized via standard RCM.
3. **Gap-Aware BFS**: Locality-optimized with original gap preservation.
4. **Degree-Sorted**: Baseline reordering (not recommended).

| **Algorithm & Mode** | **Original** | **Gap-Aware BFS** | **Speedup / Delta** |
|---|---|---|---|
| BFS Compressed | 6,908 ms | **6,334 ms** | -8.3% (Best) |
| BFS Uncompressed | 12,009 ms | 8,812 ms | -26.6% (Good) |
| RCM (Uncompressed) | 12,009 ms | 8,602 ms | -28.3% (Peak Locality) |

Table 3: Comparison of Reordering Strategies on Friendster (100% Reachability). Gap-Aware BFS provides the best balance for compressed data.

### 4.5 Analysis: Resolving the Antagonism

**Locality Gains.** Like RCM, Gap-Aware BFS provides a global leveling that ensures nodes visited together are stored together. This reduced our compressed runtime from 6.9s in the original crawl order to 6.3s.

**Compression Efficiency.** The ”sorting-by-original-ID” trick was the breakthrough. By keeping the delta gaps small ($\Delta \approx 1$ for community members), we avoided the 15-30% penalty observed with standard RCM. Gap-Aware BFS is the only reordering strategy we evaluated that successfully combines both optimization axes.

### 4.6 Design Guidelines

Based on our results, we recommend:

- **Use Gap-Aware BFS** for: Optimal performance on compressed social graphs. It provides 8-10% speedup over the crawl order without breaking compression.
- **Use RCM** for: Uncompressed, cache-sensitive algorithms where total bandwidth is not the bottleneck but latency is.

**Key Insight:** There is no ”universal” optimal preprocessing. RCM optimizes the *global matrix envelope* (bandwidth), while compression requires *local neighbor proximity* (gaps). Ideally, one should use community-preserving reordering for compression, which remains an open area for future optimization in this system.

## 5 Conclusion

This project provides a comprehensive study of **graph preprocessing and its interaction with GPU optimization strategies**. The central contribution is demonstrating that preprocessing choices (RCM reordering vs. natural ordering) have **algorithm-dependent** performance impacts that must be considered during system design.

**Key Findings.**

- **RCM reordering provides 28% speedup** for uncompressed BFS but severely degrades delta-compression efficiency.
- **Gap-Aware BFS resolves the locality-compression conflict**, achieving a 9.2% speedup over the optimized baseline on Friendster.
- **Hierarchical locality preservation** is the key to managing billion-scale graphs on consumer GPUs: small gaps for compression and global leveling for cache efficiency.

**Broader Impact.** Beyond the specific algorithms implemented (BFS and Afforest), this work establishes **design principles for graph preprocessing**:

1. Analyze algorithm memory access patterns (random vs. sequential, read-heavy vs. atomic-heavy)
2. Measure sensitivity to cache locality vs. memory bandwidth
3. Select preprocessing strategy that aligns with the dominant bottleneck
4. Consider one-time preprocessing cost vs. repeated execution benefits

Graph preprocessing is not a preprocessing step to be applied blindly—it is a **co-design problem** requiring deep understanding of both algorithm characteristics and hardware constraints. On memory-limited GPUs processing billion-scale graphs, the choice between RCM reordering and compression-friendly orderings can determine whether an algorithm is practical or infeasible.

Future work should explore adaptive preprocessing that selects orderings dynamically based on runtime profiling, as well as hybrid schemes that apply RCM to subgraphs while preserving global clustering properties.

**Source Code:** `https://github.com/EvangelosMoschou/PkDSProject3.git`