High-Performance BFS on Billion-Scale Graphs

V5.3: 831ms on Friendster Using CUDA

Evangelos Moschou
AEM: 10986

January 2026

Abstract

This report presents the design and optimization of a GPU-accelerated Breadth-First Search (BFS) solver capable of traversing the Friendster social network (65.6M nodes, 3.6B edges) in **831 milliseconds**—a 14.5x speedup over baseline and 16.5% improvement over our previous best. The key innovations are: (1) **Direct Queue Emission** using warp-aggregated atomics to eliminate O(N) distance scans; (2) **Hybrid Direction-Optimization** switching between Top-Down and Bottom-Up traversal; (3) **Delta-Varint Compression** reducing memory footprint by 37%; and (4) **Zero-Copy Streaming** enabling processing of graphs larger than GPU VRAM. We also implement the Afforest algorithm for connected components. All code is publicly available.

1 Introduction

Graph algorithms are fundamental to social network analysis, recommendation systems, and scientific computing. However, billion-scale graphs present significant challenges for GPU acceleration due to:

- **Memory Constraints**: Large graphs exceed GPU VRAM (Friendster: 14.4GB vs 5.8GB available).
- **Irregular Access Patterns**: Power-law degree distributions cause load imbalance.
- **Atomic Contention**: Concurrent frontier updates create serialization bottlenecks.

This project addresses these challenges through a systematic optimization of the BFS algorithm, culminating in **Version 5.3** which achieves sub-second traversal on Friendster. We also implement the Afforest algorithm for connected components as a secondary case study.

1.1 Contributions

1. **Direct Queue Emission**: A novel technique where Bottom-Up kernels build the next frontier queue directly during traversal, eliminating expensive O(N) scans.
2. **Warp-Aggregated Atomics**: Reducing global atomic contention by 32x through hierarchical aggregation.
3. **Zero-Copy Streaming**: Enabling processing of out-of-core graphs via pinned host memory.
4. **Comprehensive Evaluation**: Detailed performance analysis across algorithm versions.

2 System Design

2.1 Data Representation: Delta-Compressed CSR

Standard CSR (Compressed Sparse Row) format stores:

- `row_ptr[N+1]`: Offsets into column array (8 bytes each for 64-bit indexing)
- `col_idx[E]`: Neighbor IDs (4 bytes each)

For Friendster, this requires 14.4GB—exceeding our 5.8GB VRAM. We implement **Delta-Varint Compression**:

1. Sort neighbors for each row
2. Store differences: $\Delta_i = \text{neighbor}_i - \text{neighbor}_{i-1}$
3. Encode deltas using Variable-Length Quantity (1-5 bytes per delta)

This reduces Friendster to **9.13 GB** (37% reduction), enabling Zero-Copy streaming from host RAM.

2.2 Zero-Copy Memory Management

Since compressed data still exceeds VRAM, we use `cudaHostRegister` to pin host memory and map it to device address space. Kernels access graph data over PCIe with effective bandwidth of ∼1.4 GB/s. The key insight is that BFS traversal exhibits high temporal locality per level—we stream each portion of the graph once per BFS iteration.

2.3 Hybrid Direction-Optimization

We implement the Beamer direction-optimization strategy:

- **Top-Down**: Process small frontiers by expanding each frontier node's neighbors.
- **Bottom-Up**: Process large frontiers by checking if each unvisited node has a frontier neighbor.

The switching threshold is $N/26$ (≈2.5M nodes for Friendster). This hybrid approach reduces redundant edge checks by up to 50% on high-diameter graphs.

3 V5.3 Optimizations

3.1 Direct Queue Emission (Critical)

In prior versions, Bottom-Up traversal marked newly discovered nodes by writing distances, then required a separate O(N) scan (`distancesToQueueKernel`) to rebuild the frontier queue. This scan dominated runtime on large graphs.

V5.3 Solution: Bottom-Up kernels now emit discovered nodes directly to the next frontier using warp-aggregated atomics:

1. Each warp processes one unvisited node
2. If a frontier neighbor is found, thread sets `found = true`
3. Warp ballot collects all `found` flags: `__ballot_sync()`
4. Leader thread atomically reserves space: `atomicAdd(queue_size, popcount)`
5. Each found thread writes its node ID at the computed offset

This eliminates the O(N) scan entirely, reducing Bottom-Up level time by 100–150ms.

3.2 Warp-Aggregated Atomics (Top-Down)

Top-Down kernels originally used per-discovery `atomicAdd` calls to append to the next frontier. With billions of edges, this created severe contention.

V5.3 Solution: Within each warp, discoveries are aggregated before a single atomic operation:

1. Each thread processes edges and flags discoveries
2. `__ballot_sync()` + `__popc()` count discoveries per warp
3. Leader performs one `atomicAdd` for the entire warp
4. `__shfl_sync()` broadcasts base index to all threads
5. Each thread computes its local offset and writes

This reduces global atomics from O(edges) to O(warps), a 32x reduction.

3.3 Synchronization Reduction

We removed unnecessary `cudaDeviceSynchronize()` calls after kernel launches, relying instead on CUDA's implicit stream serialization. This reduced per-level overhead by ∼10%.

4 Experimental Results

4.1 Test Environment

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

4.2 Benchmark: Friendster Social Network

- Vertices: 65,608,366
- Edges: 3,612,134,270 (3.6 billion)
- Uncompressed size: 14.4 GB
- Compressed size: 9.13 GB (1.58x ratio)
- Diameter: 22 levels

4.3 Performance Evolution

Version	Key Optimization	Time (ms)	Speedup
V3 Baseline	Shared Memory Kernels	12,009	1.0x
V4.1 Hybrid	Direction-Optimization	1,200	10.0x
V4.3 Compressed	Zero-Copy + Varint	996	12.1x
V5.3 Final	**Direct Queue Emission**	**831**	**14.5x**

Table 1: BFS Performance Evolution on Friendster (source=0, 100% reachability)

Metric	BFS V5.3	Afforest
Runtime	**831 ms**	21.6 s
Edges Processed	3.6 B	3.6 B
Throughput (GTEPS)	4.35	0.17
Memory Streamed	9.13 GB	9.13 GB
Effective Bandwidth	11.0 GB/s	0.42 GB/s

Table 2: Detailed metrics. GTEPS = Giga Traversed Edges Per Second.

4.4 Analysis

The 14.5x speedup is achieved through:

- **Hybrid Switching** (V4.1): Eliminates redundant edge checks, reducing iteration count.
- **Compression + Zero-Copy** (V4.3): Reduces data transfer by 37%, hiding PCIe latency.
- **Direct Emission** (V5.3): Eliminates O(N) scans, reducing per-level overhead by 100ms+.

Afforest remains bandwidth-bound due to its random atomic linking pattern, which cannot exploit the structured traversal of BFS.

5 Afforest Algorithm

We implement the Afforest algorithm for connected components:

1. **Sampling Phase** (Optional): Sample random edges to quickly link large components.
2. **Hook Phase**: Iterate over all edges, linking components via `atomicMin`.
3. **Compression**: Flatten component tree to root pointer.

For compressed graphs, we disable sampling and GCC pruning (as component IDs cannot be reliably estimated). Runtime: 21.6s—primarily bandwidth-bound by the random atomic pattern.

6 Conclusion

This project demonstrates that billion-scale graph traversal is achievable in sub-second time on consumer GPUs through careful algorithmic optimization:

- **831ms BFS on Friendster**: A 14.5x speedup over baseline, enabled by Direct Queue Emission.
- **Warp-Level Aggregation**: Reduces atomic contention by 32x.
- **Zero-Copy + Compression**: Enables out-of-core processing with 37% bandwidth reduction.

The key insight is that **algorithmic complexity matters more than raw bandwidth**. Eliminating the O(N) distance scan in V5.3 provided a larger improvement than all previous compression optimizations combined.

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