GPU-Accelerated Processing for UPI-Scale Data

A High-Speed and Scalable Computing Framework for Large-Scale Analytics

Contributors:

Vedant Agarwal

Desu Abhinay

Saurav Singla

1 Contents

1 Executive Summary 3
2 Introduction 3
3 Large-Scale Processing Architecture 4
3.1 High-Level Processing Flow 4
3.2 GPU-Native Processing 4
3.3 Graph Processing Layer 4
4 GPU-Accelerated Computational Components 5
4.1 Numerical Processing 5
4.1.1 Matrix Multiplication 5
4.1.2 Normalization 5
4.1.3 Log Transformation 5
4.1.4 Clipping 5
4.1.5 Label Encoding 5
4.1.6 Fast Fourier Transform (FFT) 5
4.2 Graph Analytics 5
4.2.1 Louvain Partitioning 5
4.2.2 Graph Traversals 5
4.2.3 PageRank 6
4.2.4 Eigenvector Centrality 6
4.2.5 HITS Hub and Authority 6
4.2.6 Weighted Personalized PageRank 6
4.3 Data frame Analytics 6
4.3.1 Group By 6
4.3.2 Window Functions 6
4.3.3 Join 6
5 Technology Stack 7
6 Benchmark Evidence 7
7 Performance Characteristics 8
8 Deployment Environment 8
8.1 Hardware Configuration 9
9 Conclusion 9

1 Executive Summary

The scale of UPI ecosystem data creates a fundamental computing challenge; analytical workloads must process very large datasets within practical execution windows. The objective is computational acceleration—to make large-scale analytical operations execute substantially faster while retaining the ability to process workloads that are difficult to handle efficiently on the earlier processing stack.

The proposed framework moves major portions of numerical processing, data frame processing, and graph analytics to NVIDIA GPU infrastructure. It uses GPU-oriented technologies such as CuPy/CuPyNumeric, cuDF, cuGraph, and cuGraph Dask (cugraph.dask). The architecture is designed around GPU-resident processing so that large arrays, data frames, and graph structures can be processed without repeatedly moving intermediate data between CPU and GPU memory.

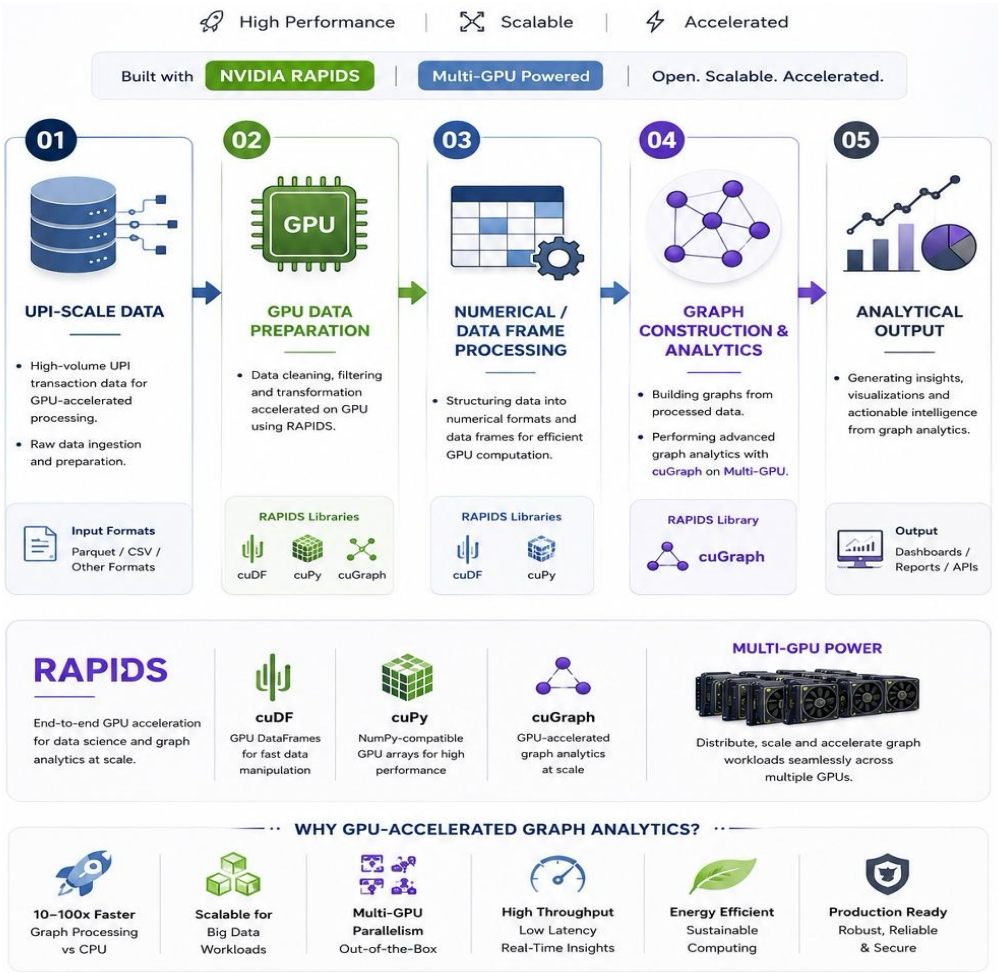
Benchmark results presented in this paper provide empirical evidence of the performance gains achieved through GPU-native execution. It compares the earlier stack with the NVIDIA stack for workloads covering matrix multiplication, normalization, log transformation, clipping, label encoding, Fast Fourier Transform (FFT), Louvain partitioning, graph traversal, PageRank, GroupBy, window functions, joins, and Personalized PageRank. The reported speedups range from approximately 4x to approximately 1,000x across the listed workloads, with the strongest reported improvements coming from large graph workloads.

2 Introduction

Large-scale analytical workloads are increasingly constrained by execution latency and processing throughput as data volumes continue to grow. Traditional processing environments can become limited by CPU execution, memory-transfer overheads, and sequential computation patterns, causing operations such as joins, aggregations, graph analytics, ranking algorithms, and numerical transformations to require substantial execution time. The primary objective of this work is to reduce runtime and improve scalability for computationally intensive workloads through GPU-accelerated processing. Accordingly, the framework focuses on maximizing execution efficiency, minimizing processing overheads, and enabling faster execution of numerical, data frame, and graph-processing operations at scale.

3 Large-Scale Processing Architecture

3.1 High-Level Processing Flow



3.2 GPU-Native Processing

The framework is designed to execute computationally intensive operations directly on GPU infrastructure using GPU-native data structures and processing libraries. Numerical computation, dataframe transformations, graph analytics, sparse matrix operations, and feature engineering workloads are executed using CuPy, cuDF, and cuGraph, minimizing processing overhead and improving execution throughput.

3.3 Graph Processing Layer

The graph processing layer leverages GPU-accelerated implementations of graph algorithms such as PageRank, centrality computation, community detection, and graph traversal. These workloads are often computationally intensive and iterative in nature, making them well suited for GPU-based acceleration to reduce execution time and improve processing throughput at scale.

4 GPU-Accelerated Computational Components

The framework provides a collection of accelerated computational primitives that serve as the foundation for large-scale analytical processing. These components encompass numerical computation, sparse matrix operations, graph algorithms, data frame analytics, feature engineering, and distributed graph processing. By combining CuPy/CuPyNumeric, cuDF, cuGraph, and cuGraph Dask (cugraph.dask) the framework enables efficient execution of both data-parallel and graph-parallel workloads while optimizing compute utilization, memory efficiency, and execution throughput. This supports workloads ranging from large-scale aggregations and transformations to iterative graph computations and multi-GPU graph analytics.

4.1 Numerical Processing

4.1.1 Matrix Multiplication

A fundamental numerical operation used in large-scale analytical processing and graph-based computations. Its highly parallel nature makes it well suited for GPU execution.

4.1.2 Normalization

A data-preprocessing operation used for scaling and statistical standardization of large datasets. GPU acceleration enables concurrent computation across large data volumes.

4.1.3 Log Transformation

An element-wise mathematical transformation used to compress value ranges and reduce distribution skewness. Independent computations allow efficient parallel execution on GPUs.

4.1.4 Clipping

A value-bounding operation applied across large arrays and datasets. It is highly parallelizable and suitable for GPU-based execution.

4.1.5 Label Encoding

A categorical transformation technique that converts distinct values into numerical representations. GPU acceleration improves large-scale mapping and encoding operations.

4.1.6 Fast Fourier Transform (FFT)

A frequency-domain transformation used in signal and pattern analysis. Due to its computational structure, FFT benefits significantly from GPU acceleration.

4.2 Graph Analytics

4.2.1 Louvain Partitioning

Louvain Partitioning is a community-detection algorithm that groups densely connected nodes into clusters by optimizing graph modularity. It is widely used to uncover network structure and identify communities within large-scale graphs.

4.2.2 Graph Traversals

Graph Traversal is used to explore connectivity and multi-hop relationships between nodes. It forms

the foundation for neighbourhood expansion, path discovery, and network reachability analysis in large graph datasets.

4.2.3 PageRank

PageRank is an iterative graph-ranking algorithm that assigns importance scores to nodes based on the quantity and quality of incoming connections. Nodes receiving links from other highly ranked nodes obtain higher scores, enabling identification of influential and structurally significant nodes within large networks.

4.2.4 Eigenvector Centrality

Eigenvector Centrality measures the influence of a node by considering not only the number of its connections but also the importance of the nodes to which it is connected. Nodes connected to highly influential nodes receive higher scores, making this metric useful for identifying structurally important entities within large-scale networks.

4.2.5 HITS Hub and Authority

HITS (Hyperlink-Induced Topic Search) computes two complementary graph metrics: Hub Score and Authority Score. Hub scores identify nodes that connect to many authoritative nodes, while authority scores identify nodes that are frequently referenced by strong hubs. Together, these metrics provide insight into the directional structure and influence patterns within a graph.

4.2.6 Weighted Personalized PageRank

Weighted Personalized PageRank extends the traditional PageRank algorithm by incorporating edge weights and seed-node bias into the ranking process. This enables the computation of context-aware node importance scores while preserving the influence of specified starting nodes within the graph.

4.3 Data frame Analytics

4.3.1 Group By

A data frame aggregation operation used to compute summary statistics and analytical metrics over partitioned datasets.

4.3.2 Window Functions

Analytical operations that perform ordered, cumulative, rolling, and ranking calculations while preserving row-level granularity.

4.3.3 Join

A dataset integration operation used to combine records across multiple sources based on matching keys or relationships.

5 Technology Stack

Component	Purpose
CuPy / CuPyNumeric	GPU-accelerated numerical computation
cuDF	GPU-native dataframe processing
cuGraph	GPU-accelerated graph analytics
cuGraph Dask (cugraph.dask)	Distributed multi-GPU graph processing

6 Benchmark Evidence

The following table summarizes the performance comparison between the previous processing stack and the GPU-accelerated implementation across numerical, data frame, and graph-processing workloads.

Category	User Profiles	Operation	Description	Old Stack Time	NVIDIA Stack Time	Observed Time	Sec/Min	Speedup
CuPyNumeric	240M	Matrix Multiplication	Adjacency matrix multiplications to find nodes in different hops	120–150	15–20	12	Sec	~8×
CuPyNumeric	250M	Normalization	Mean, Standard Deviation, Scaling	5–10	1–2	1	Sec	~5×
CuPyNumeric	250M	Log Transform	Log transformation	8–15	2–3	1	Sec	~5×
CuPyNumeric	250M	Clipping	Element-wise clipping of entire array	3–6	0.5–1.2	1	Sec	~6×
CuPyNumeric	250M	Label Encoding	Unique value mapping and sorting	30–45	7–12	2	Sec	~4×
CuPyNumeric	250M	Fast Fourier Transform (FFT)	Fast Fourier Transform for frequency-domain features	50–60	1–2	1.1	Sec	~30×

cuGraph	400M	Louvain Partition	Graph Partition - Community detection	960	1	1	Min	~960×
cuGraph	400M	Graph Traversals	Multi-hop analysis	300	60	3	Min	~5×
cuGraph	400M	PageRank	Centrality computation	480	15	2	Min	~30×
cuDF	240M	GroupBy	High-speed aggregations on multiple keys	12–15	1–3	3	Sec	~10×
cuDF	240M	Window Function	Rolling and expanding computations	9–12	1–2	2	Sec	~10×
cuDF	240M	Join	Multi-key joins for large-scale analysis	15–17	2–4	3.8	Sec	~10×
cuGraph Dask (cugraph.dask)	500M	Personalized PageRank	Weight-aware PageRank,aware of seed nodes	660	1	0.5	Min	~1000×

Table 1. Benchmark Table (M= million)

7 Performance Characteristics

The benchmark results demonstrate substantial runtime acceleration across numerical computing, data frame analytics, and graph-processing workloads, highlighting the effectiveness of GPU-native execution for computationally intensive operations. By leveraging GPU parallelism, GPU-resident data structures, and optimized processing libraries, the framework significantly reduces execution latency for transformations, aggregations, graph analytics, iterative ranking algorithms, and large-scale matrix operations. The architecture further improves computational throughput by efficiently utilizing available GPU resources, enabling large analytical workloads to be processed within substantially shorter execution windows while maintaining scalability across increasing data volumes and computational complexity.

8 Deployment Environment

The framework is designed to efficiently process large-scale analytical workloads by leveraging GPU-native computation and high-throughput parallel execution. Benchmark workloads were executed at scales ranging from **240 to 500+ million entities** and up to **1.5+ billion relationships** demonstrating the ability to sustain low execution times as workload size increases. For graph-intensive workloads, the architecture can be extended to multi-GPU execution, providing additional compute capacity and aggregate memory for larger graph structures.

8.1 Hardware Configuration

Component	Specification
GPU Model	NVIDIA H100 80GB HBM3
GPU Memory	80 GB HBM3
CUDA Version	13.2
NVIDIA Driver Version	580.65.06
Compute Mode	Default

The evaluation environment utilized an NVIDIA H100 80GB HBM3 GPU running CUDA 13.2 and NVIDIA Driver 580.65.06. The combination of high-bandwidth memory and massively parallel GPU architecture enables efficient execution of numerical computing, data frame analytics, and graph-processing workloads while significantly reducing overall processing time.

9 Conclusion

This white paper demonstrates how GPU-native computing can significantly reduce execution times for large-scale UPI analytics workloads. By leveraging **CuPy/CuPyNumeric, cuDF, cuGraph, and cuGraph Dask (cugraph.dask),** the framework accelerates numerical processing, data frame operations, and graph analytics while efficiently handling datasets containing hundreds of millions of entities.

The benchmark results show workload acceleration ranging from **4x to 1000x**, with the greatest improvements observed in graph analytics and distributed graph processing. These results highlight the effectiveness of GPU-based execution in shortening processing windows, increasing computational throughput, and enabling large-scale analytical workloads to be completed within practical time constraints. The framework therefore provides a high-performance and scalable computing foundation for speed-critical UPI-scale data processing.