BFS-FAR: Fragmented Asynchronous Breadth-First Search on UPMEM Processing-in-Memory

Tasmia Jannat[1] (tjv7w@mst.edu), Gharret Wilcut[1], Sanjukta Bhowmick[2], Satish Puri[1]
[1]Department of Computer Science, Missouri University of Science and Technology, Rolla, MO, USA
[2]University of North Texas, Denton, Texas, USA

Introduction

- **Breadth-first search (BFS)** is a core graph primitive for reachability and shortest-hop distance.
- BFS is difficult to accelerate because of
  - irregular memory accesses
  - low computation per edge
  - frequent synchronization.

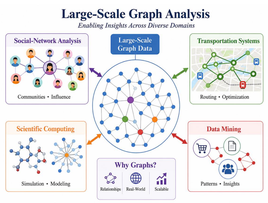


Figure 1: Applications of Large-Scale Graph Analytics

- Conventional systems face a **memory wall**, making data movement a major bottleneck.
- **Processing-in-Memory (PIM)** moves computation closer to memory to reduce data movement.

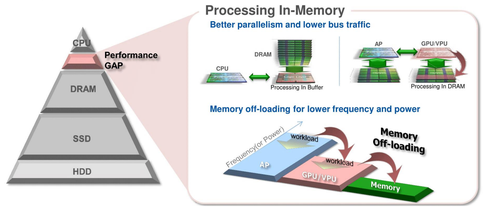


Figure 2: Processing In Memory (PIM) Concept [1]

Motivations

- **BFS is memory-bound and irregular**, causing poor locality and high synchronization costs.
- **UPMEM reduces data movement**, but DPUs cannot communicate directly, that's why Level-synchronous BFS requires frequent host coordination.
- Existing PIM BFS, **PrIM-BFS [2]**, repeatedly exchanges frontiers through the host; coordination accounts for **87–92%** of its runtime.
- Prior UPMEM work has explored graph analytics [3], [4] and tree-based spatial processing [5], [6]. BFS-FAR extends this line of work to asynchronous graph traversal.

Goal: Reduce **host coordination** through fragment-local relaxation and lightweight reconciliation.

Methodology

- **Fragment:** Partition the graph into local CSR fragments across DPUs.
- **Relax:** DPUs run in parallel and reuse improved levels within the same kernel launch.
- **Reconcile:** The host merges replicated levels using the minimum value and propagates improvements.
- *Repeat Relax and Reconcile until convergence.*

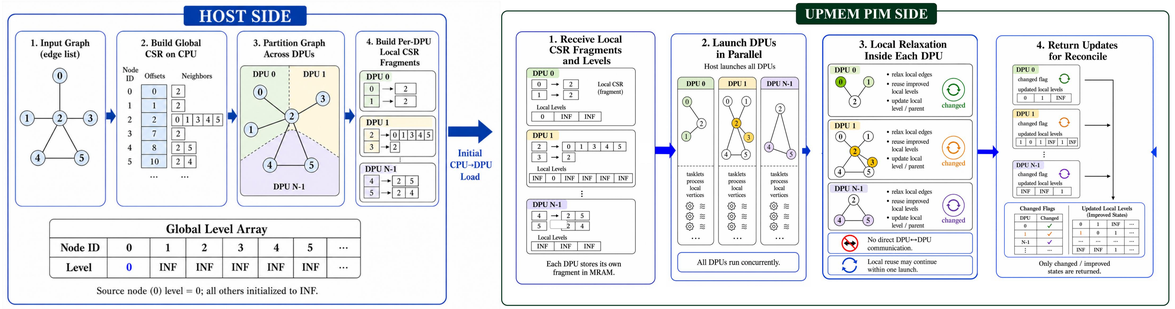


Figure 3: BFS-FAR workflow across the host and UPMEM DPUs: graph fragmentation, parallel local relaxation, and host-side reconciliation.

Experimental Setup

Each UPMEM PIM enabled memory DIMM contains 16 PIM chips, totaling 128 DPUs and 8 GB of memory. Scale: 128 DPUs and 8 GB per DIMM. Total 20 DIMM → **2560 DPUs, 160 GB**

Each DPU has exclusive access to (1) 64-MB Main RAM (MRAM), (2) 24-KB Instruction RAM (IRAM), (3) 64-KB Working RAM (WRAM).

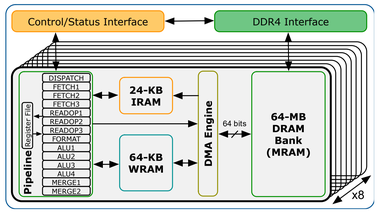


Figure 4: UPMEM PIM DPU Architecture [2]

Table 1: Experimental Platform Configuration

Component	Details
UPMEM system	**2,540** of 2,560 DPUs; **16** of 24 tasklets/DPU
Memory	64 MB MRAM and 64 KB WRAM per DPU
Toolchain	UPMEM SDK v12.0.0, C/C++
Baselines	PrIM-BFS [2] and parallel CPU with 64 threads
Datasets:	R-MAT and controlled-size random graphs

Results

Table 2: BFS-FAR vs. PrIM-BFS Algorithm-Only Performance

Dataset	**BFS-FAR**	PrIM-BFS	Speedup over PrIM-BFS
RMAT 18	0.61	14.31	23.49
RMAT 19	1.09	22.41	20.65
RMAT 20	0.99	41.67	42.00
RMAT 21	1.71	72.26	**42.33**
RMAT 22	7.22	129.96	17.99
Graph 1	0.13	10.33	80.69
Graph 2	0.13	10.79	81.73
Graph 3	0.16	13.78	**86.55**
Graph 4	0.20	14.23	71.72
Graph 13	0.79	42.61	54.00
Graph 25	15.97	298.79	18.71

BFS-FAR achieves **18–86.5**× algorithm-only speedup over PrIM-BFS.

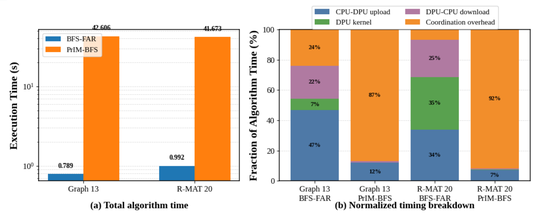


Figure 5: Performance and timing breakdown of BFS-FAR and PrIM-BFS on Graph 13 and R-MAT 20.

PrIM-BFS spends **87–92**% of its time in host-mediated coordination.

Why BFS-FAR Is Faster

PrIM-BFS	BFS-FAR
One BFS level per launch	Multiple local relaxations per launch
Exchanges a global frontier	Reuses improved local levels
Host-side frontier union	Minimum-level reconciliation
Frequent host round trips	Fewer host round trips

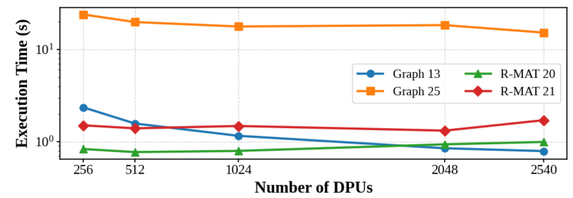


Figure 6: BFS-FAR execution time as the number of DPUs increases.

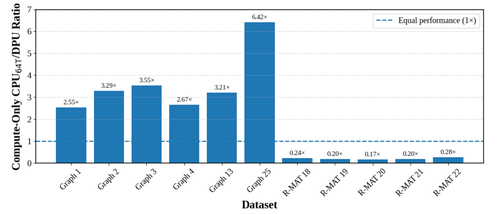


Figure 7: CPU_{64T} time divided by DPU-kernel time; values above 1×indicate a faster DPU kernel.

Key Takeaway:

- R-MAT scales less.
- Round-robin partitioning hurts locality.
- DPU work becomes uneven.

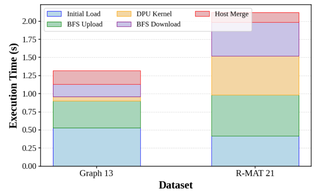


Figure 8: BFS-FAR Timing Breakdown

Conclusion

- BFS-FAR reduces global frontier exchange through local relaxation and minimum-level reconciliation.
- It achieves **18.0–86.5**× algorithm-only speedup over PrIM-BFS.
- Performance is still limited by locality and load imbalance, especially on R-MAT graphs.

Future Work

- Replace round-robin assignment with **locality- and edge-balanced partitioning**.
- Replicate or separately manage high-degree hub vertices.
- Communicate only changed vertex states.

References

1. Samsung, "Processing In Memory: PIM – Fill the Performance Gap and Deliver Energy-Efficient Solutions," Hot Chips 28, 2016.
2. J. Gómez-Luna et al., "Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System," IEEE Access, 2022.
3. M. Barkhordar et al., "ALPHA-PIM: Analysis of Linear Algebraic Processing for High-Performance Graph Applications on a Real Processing-In-Memory System," IISWC, 2025.
4. S. Cai et al., "PimPam: Efficient Graph Pattern Matching on Real Processing-in-Memory Hardware," Proc. ACM Manag. Data, 2024.
5. T. Jannat, M. Gowanlock, and S. Puri, "Parallel R-tree-Based Spatial Query Processing on a Commercial Processing-in-Memory System," ISC High Performance, 2026.
6. T. Jannat, M. Gowanlock, and S. Puri, "Challenges in Scaling R-tree Spatial Search on Processing-in-Memory," HPDC, 2026.

Acknowledgements : NSF Grant #2344585 and NSF Grant #2402987