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

Tasmia Jannat[*,*], Gharret Wilcut[*,*], Sanjukta Bhowmick[†], and Satish Puri[*]
[*] Missouri University of Science and Technology, Rolla, Missouri, USA
Email: {tjv7w, gawgc7, satish.puri}@mst.edu
[†] University of North Texas, Denton, Texas, USA
Email: Sanjukta.Bhowmick@unt.edu

I. Introduction

Breadth-first search (BFS) is a fundamental graph primitive, but it is difficult to accelerate because of irregular memory accesses, low work per edge, and frequent synchronization. Processing-in-Memory reduces data movement, but UPMEM DRAM Processing Units (DPUs) cannot communicate directly. As shown in Fig. 1, DPUs rely on the host for global coordination. Therefore, PrIM-BFS [1] exchanges the global frontier through the host after every BFS level.

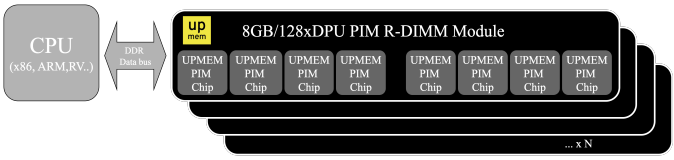


Fig. 1. Processing-in-Memory (PIM) System Organization [1].

Prior UPMEM studies have explored graph analytics, including BFS and other graph applications [2] and graph pattern matching [3]. Our previous work investigated tree-based spatial indexing and query processing on UPMEM [4], [5], demonstrating the potential of PIM for irregular data structures. Building on this line of work, we present **BFS-FAR**, which partitions the graph across DPUs, performs multiple local relaxations per launch, and reconciles replicated levels via host-side minimum reduction to reduce frontier exchange and coordination.

Using up to 2,540 DPUs, BFS-FAR achieves an 18.0–86.5× algorithm-only speedup over PrIM-BFS on R-MAT and controlled-size synthetic graphs.

II. Methodology

BFS-FAR has three stages: *Fragment*, *Asynchronous Relaxation*, and *Reconcile*. Fragmentation runs once; the other stages repeat until convergence.

*a) **Fragment**:* Vertex u and its outgoing edges are assigned to DPU $u \bmod P$. Each DPU stores a local CSR fragment and vertex state.

*b) **Asynchronous Relaxation**:* The host initializes the source level in the first iteration. Later, only improved levels are sent to relevant DPUs. Each DPU applies

$$L_d(v) \leftarrow \min\left(L_d(v), L_d(u)+1\right). \tag{1}$$

As Fig. 2 shows, updated levels may be reused later in the same launch, enabling multiple local relaxations before reconciliation.

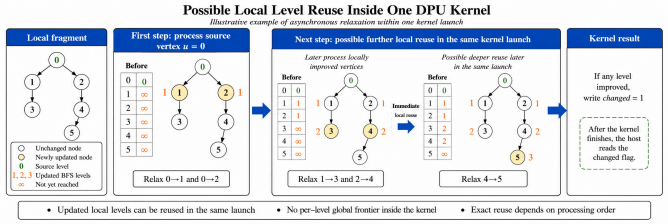


Fig. 2. Local level reuse within one DPU launch.

*c) **Reconcile**:* Levels from changed DPUs are returned to the host, which merges replicated states using minimum reduction.

$$L(v) = \min_{d:\, v \in V_d} L_d(v). \tag{2}$$

Only improved levels are sent back. Relaxation and reconciliation repeat until no level changes.

III. Results

A. Experimental Setup

We use 2,540 of 2,560 available UPMEM DPUs and 16 of 24 tasklets per DPU, PrIM-BFS as the baseline, and a 64-thread CPU for kernel comparison. We use two synthetic datasets: R-MAT and controlled-size graphs.

B. Overall Performance and Scaling

Table I shows that BFS-FAR is 18.0–86.5× faster than PrIM-BFS under the same UPMEM setup. Algorithm-only time includes DPU execution, transfers, and host coordination, but excludes preprocessing.

Figure 3 shows that coordination accounts for 87–92% of PrIM-BFS time. BFS-FAR reduces this cost through local relaxation and minimum-level reconciliation.

[*]Tasmia Jannat and Gharret Wilcut contributed equally to this work.

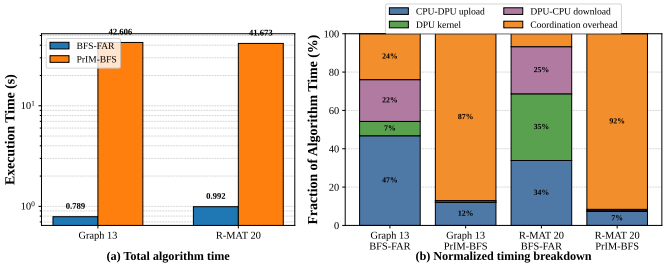


Fig. 3. Performance and timing breakdown of BFS-FAR and PrIM-BFS.

TABLE I
ALGORITHM-ONLY PERFORMANCE OF BFS-FAR COMPARED WITH PRIM-BFS.

Dataset	BFS-FAR (s)	PrIM-BFS (s)	Speedup
R-MAT 18	0.61	14.31	23.49×
R-MAT 19	1.09	22.41	20.65×
R-MAT 20	0.99	41.67	42.00×
R-MAT 21	1.71	72.26	**42.33**×
R-MAT 22	7.22	129.96	18.00×
Graph 1	0.128	10.325	80.69×
Graph 2	0.132	10.788	81.73×
Graph 3	0.159	13.781	**86.55**×
Graph 4	0.198	14.233	71.72×
Graph 13	0.789	42.607	54.00×
Graph 25	15.969	298.787	18.71×

Figure 4 shows the best scaling on Graph 13. R-MAT scales less consistently. Figure 5 further shows that the DPU kernel is faster than $\text{CPU}_{64\text{T}}$ on controlled-size graphs, but not on R-MAT. This suggests that R-MAT's irregular structure and round-robin partitioning limit locality and workload balance.

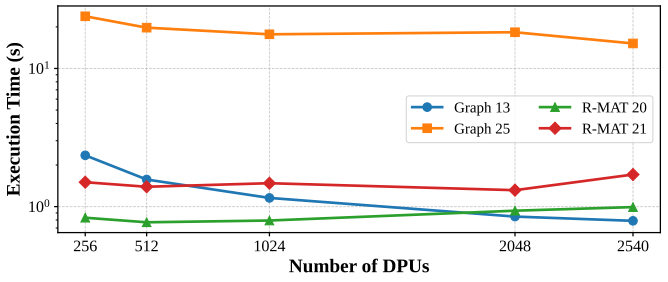


Fig. 4. BFS-FAR execution time with increasing numbers of DPUs.

IV. CONCLUSION

BFS-FAR reduces repeated host coordination through local relaxation and minimum-level reconciliation, achieving substantial speedups over PrIM-BFS. Performance remains limited by graph structure and round-robin partitioning, especially for irregular R-MAT graphs. Future work will explore graph-aware partitioning, reordering, sparse updates, and deeper local propagation to improve locality, balance DPU work, and reduce host round trips.

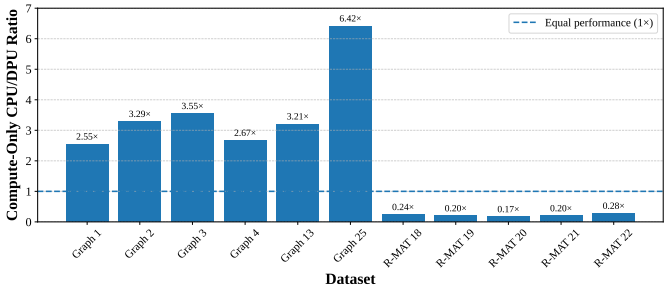


Fig. 5. Compute-only $\text{CPU}_{64\text{T}}$-to-DPU ratio, calculated as the $\text{CPU}_{64\text{T}}$ baseline time divided by DPU-kernel-only time. Values above 1× indicate a faster DPU kernel.

REFERENCES

[1] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a new paradigm: Experimental analysis and characterization of a real processing-in-memory system," *IEEE Access*, vol. 10, pp. 52565–52608, 2022.

[2] M. Barkhordar, A. Tabatabaeian, M. Sadrosadati, C. Giannoula, J. G. Luna, I. El Hajj, O. Mutlu, and A. R. Alameldeen, "Alpha-pim: Analysis of linear algebraic processing for high-performance graph applications on a real processing-in-memory system," in *2025 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 257–271, IEEE, 2025.

[3] S. Cai, B. Tian, H. Zhang, and M. Gao, "Pimpam: Efficient graph pattern matching on real processing-in-memory hardware," *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 1–25, 2024.

[4] T. Jannat, M. Gowanlock, and S. Puri, "Parallel r-tree—based spatial query processing on a commercial processing-in-memory system," in *ISC High Performance 2026 Research Paper Proceedings (40th International Conference)*, pp. 1–12, Prometeus GmbH, 2026.

[5] T. Jannat, M. Gowanlock, and S. Puri, "Challenges in scaling r-tree spatial search on processing-in-memory," in *Proceedings of the 35th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '26, pp. 609–610, Association for Computing Machinery, 2026.