Single Machine graph analytics using Intel Optane DC persistent memory

Presentation by Yosef E Mihretie

Massive scale graph analytics: the choices

- Graphs today: billions of vertices, trillions of edges, and growing
- Most computers can’t fit them in memory, some can but DRAM is expensive
- Two choices: out-of-core(like GridGraph) and distributed memory(like D-Galois)
- Out-of-core: graph stored in SSD, chunks of it read to memory and processed as needed
- Out-of-core: algorithms need re-engineering, data layout must be changed, expensive IO etc
- Distributed memory: communication is a major bottleneck

Optane DC: adjusting the memory hierarchy

- Higher density, byte-addressable, lower cost and slower than DRAM, faster than SSDs, same form factor as DD4 DRAM

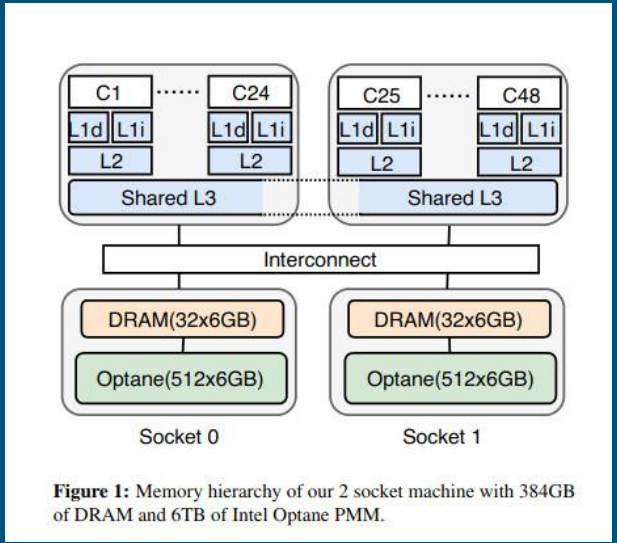


Figure 1: Memory hierarchy of our 2 socket machine with 384GB of DRAM and 6TB of Intel Optane PMM.

Optane DC: adjusting the memory hierarchy

- Two different modes: memory-mode and app-direct

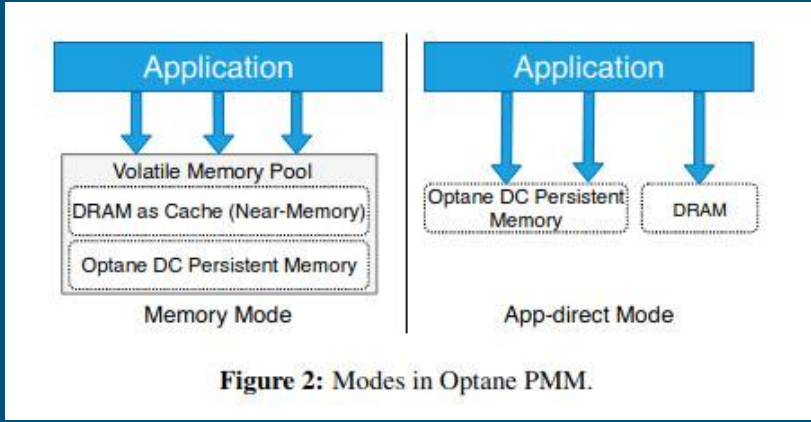


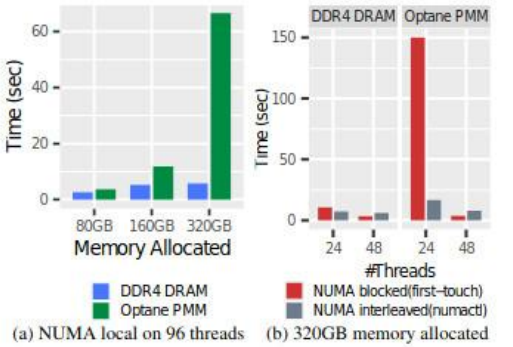
Figure 2: Modes in Optane PMM.

Graph analytics using Optane PMM: Results

- Found that Optane PMM in memory mode is a performant and affordable option
- Suggested runtime and algorithmic adjustment to make graph algorithms more performant on PMM
- NUMA-aware memory allocations that maximize near-memory utilizations are important
- Avoiding page-management overhead is key to performance
- Allowing programmers to implement flexible algorithms, specifically non-vertex and asynchronous programs, reduce memory accesses

Memory consideration

- Three main NUMA-aware allocations: local, blocked and interleaved
- Maximizing near-memory, DRAM, hit is critical

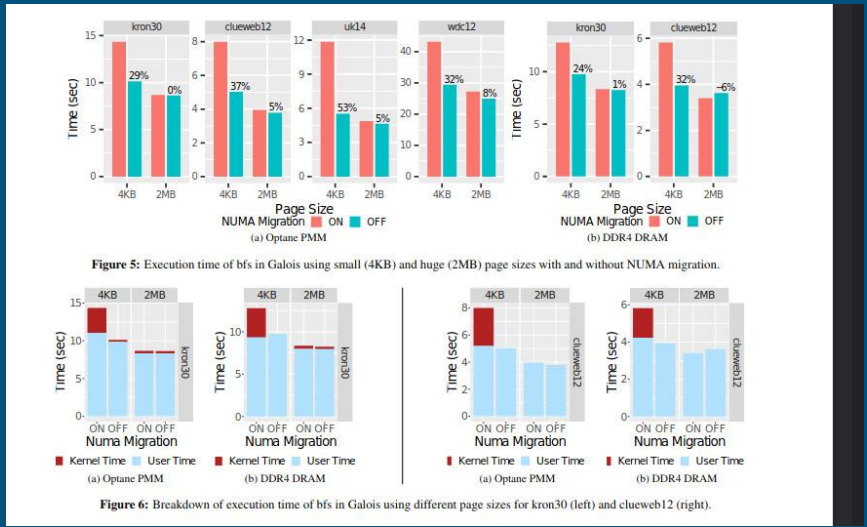


(a) NUMA local on 96 threads (b) 320GB memory allocated

Figure 4: Time to write memory allocated on Optane PMM and DDR4 DRAM using a micro-benchmark.

Memory considerations

- Requires bookkeeping to choose what pages to remove
- Changes virtual to physical page mappings -> TLB stale -> more TLB misses



Algorithmic principles

- Many graph algs: data kept on each vertex, set of active vertices, operators work on neighbors of an active vertex, data updated
- Vertex algorithms: the neighbors of an active vertex are only its immediate neighbors
- Non-vertex algorithms: the neighbors of an active vertex are any arbitrary portion of the graph
- Non-vertex algorithms: the neighbors of an active vertex are any arbitrary portion of the graph

Algorithmic principles

- Pull-style: neighbors used to update an active vertex
- Push-style: an active vertex used to update its neighbors
- Topology-driven: operates on all vertices
- Data-driven: a set of active vertices kept and operated on
- Bulk-synchronous: a dense worklist (bitvector) to keep active vertices, current and next vertex sets, continue until next list is empty
- Asynchronous: a sparse worklist (set of ids) to keep active vertices, pop and push vertices from it until nothing left

Algorithmic principles

- Non-vertex, asynchronous, data driven programs perform better on Optane PMM systems for real-world graphs

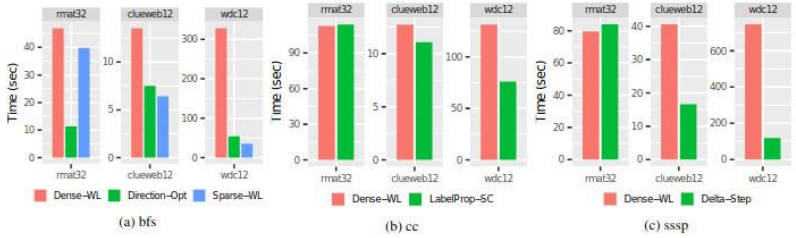


Figure 7: Execution time of different data-driven algorithms in Galois on Optane PMM using 96 threads.

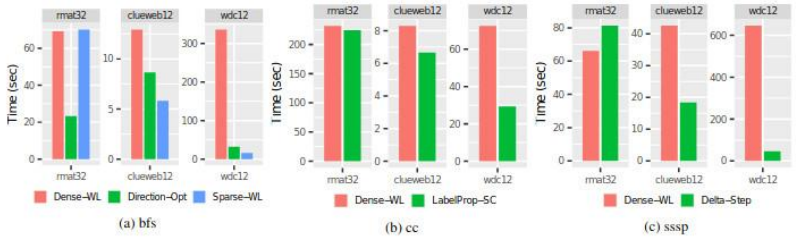


Figure 8: Execution time of different data-driven algorithms in Galois on Entropy (1.5TB DDR4 DRAM) using 56 threads.

Experiments: Uno

- Comparing Galois, GraphIt, GAP and GBBS(Ligra)

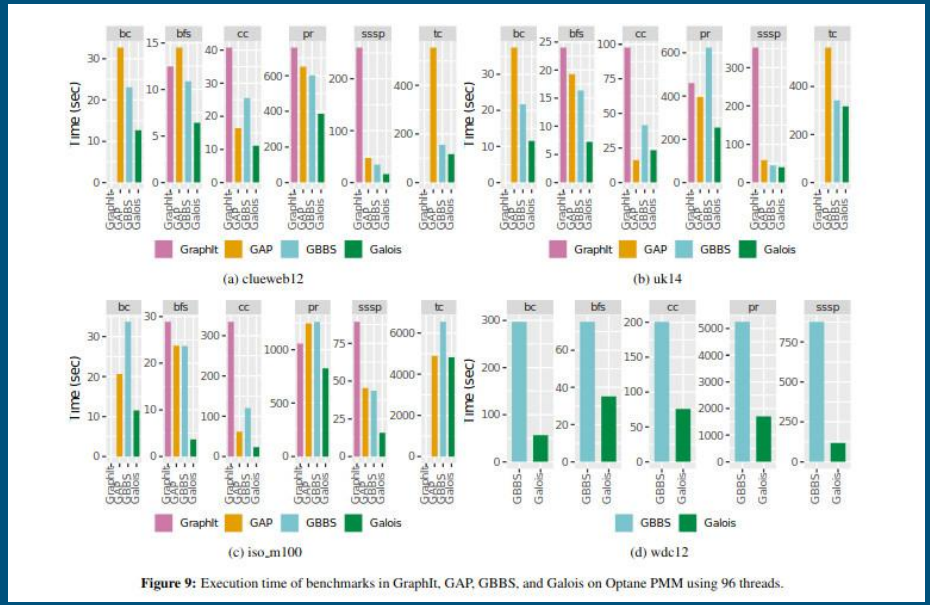


Figure 9: Execution time of benchmarks in GraphIt, GAP, GBBS, and Galois on Optane PMM using 96 threads.

Experiments: Dos

- Comparing Galois on Optane PMM vs DRAM with medium sized graphs

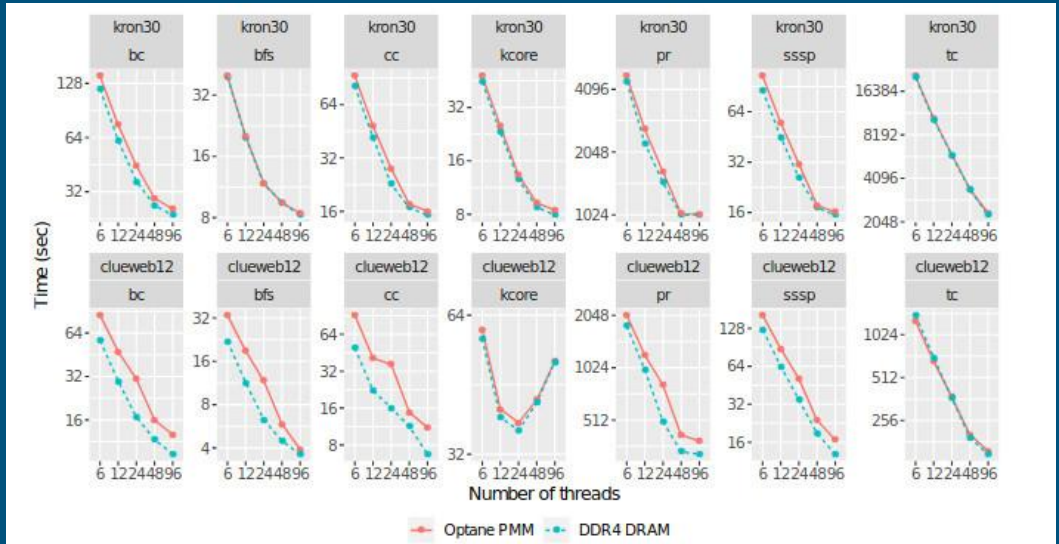


Figure 10: Strong scaling in execution time of benchmarks in Galois using DDR4 DRAM and Optane PMM.

Time for a joke

Experiments: Tres

- Comparing Optane PMM vs Distributed memory (D-Galois)

Table 4: Execution time (sec) of benchmarks in Galois on Optane PMM (OB) machine using efficient algorithms (non-vertex, asynchronous) and D-Galois on Stampede cluster (DM) using vertex programs with minimum # of hosts that hold the graph. Speedup of Optane PMM over Stampede. Best times highlighted in green.

Graph	App	Stampede (DM)	Optane PMM (OB)	Speedup (DM/OB)
clueweb12	bc	51.63	12.68	4.07×
	bfs	10.71	6.43	1.67×
	cc	13.70	11.08	1.24×
	kcore	186.03	51.05	3.64×
	pr	155.00	385.64	0.40×
	sssp	33.87	16.58	2.04×
uk14	bc	172.23	11.53	14.9×
	bfs	28.38	7.22	3.93×
	cc	14.56	21.30	0.68×
	kcore	56.08	7.94	7.06×
	pr	82.77	254.95	0.32×
	sssp	52.49	39.99	1.31×
iso_m100	bc	6.97	11.57	0.60×
	bfs	7.94	3.69	2.15×
	cc	16.32	23.69	0.69×
	kcore	1.21	0.48	2.52×
	pr	191.21	824.54	0.23×
	sssp	61.90	15.66	3.95×
wdc12	bc	775.84	56.48	13.7×
	bfs	71.50	35.25	2.03×
	cc	69.21	76.00	0.91×
	kcore	105.42	49.22	2.14×
	pr	118.01	1706.35	0.07×
	sssp	136.47	118.81	1.15×

Experiments: Cuatro

- Comparing Optane PMM vs Out-of-core (GridGraph)

Table 5: Execution time (sec) of benchmarks in Galois on Optane PMM in Memory Mode (MM) and the out-of-core framework GridGraph on Optane PMM in App-direct Mode (AD). Best times highlighted in green. "—" indicates it did not finish in 2 hours.

Graph	App	GridGraph (AD)	Galois (MM)	Speedup (AD/MM)
clueweb12	bfs	5722.75	6.43	890.0×
	cc	5411.23	11.08	488.4×
uk14	bfs	—	7.22	NA
	cc	5700.48	21.30	267.6×

Summary

- Optane PMM outperforms distributed memory and out-of-core systems
- Optane PMM is as easy to program as DRAM, less expensive
- Frameworks should allow flexibility, memory allocation should maximize DRAM usage and migrations aren’t helpful