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
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

![Graph showing time to write memory allocated on Optane PMM and DDR4 DRAM using a micro-benchmark.](image)
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
Experiments: Uno

- Comparing Galois, GraphIt, GAP and GBBS (Ligra)
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, asyn-chronous) and D-Galois are Stampede cluster (OB) using vertex programs with minimum # of hosts that hold the graph. Speedup of Optane PMM over Stampede. Best times highlighted in green.
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.

<table>
<thead>
<tr>
<th>Graph</th>
<th>App</th>
<th>GridGraph (AD)</th>
<th>Galois (MM)</th>
<th>Speedup (AD/MM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>clueweb12</td>
<td>bfs</td>
<td>5722.75</td>
<td>6.43</td>
<td>890.0×</td>
</tr>
<tr>
<td></td>
<td>cc</td>
<td>5411.23</td>
<td>11.08</td>
<td>488.4×</td>
</tr>
<tr>
<td>uk14</td>
<td>bfs</td>
<td>—</td>
<td>7.22</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>cc</td>
<td>5700.48</td>
<td>21.30</td>
<td>267.6×</td>
</tr>
</tbody>
</table>
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