Sage: Parallel Semi-Asymmetric Graph Algorithms for NVRAMs

Julian Shun

Slides made by Laxman Dhulipala

Joint work with Laxman Dhulipala, Charles McGuffey, Hongbo Kang, Yan Gu, Guy Blelloch, and Phil Gibbons (VLDB'20)

Shared-Memory Parallelism

Shared-Memory Machines

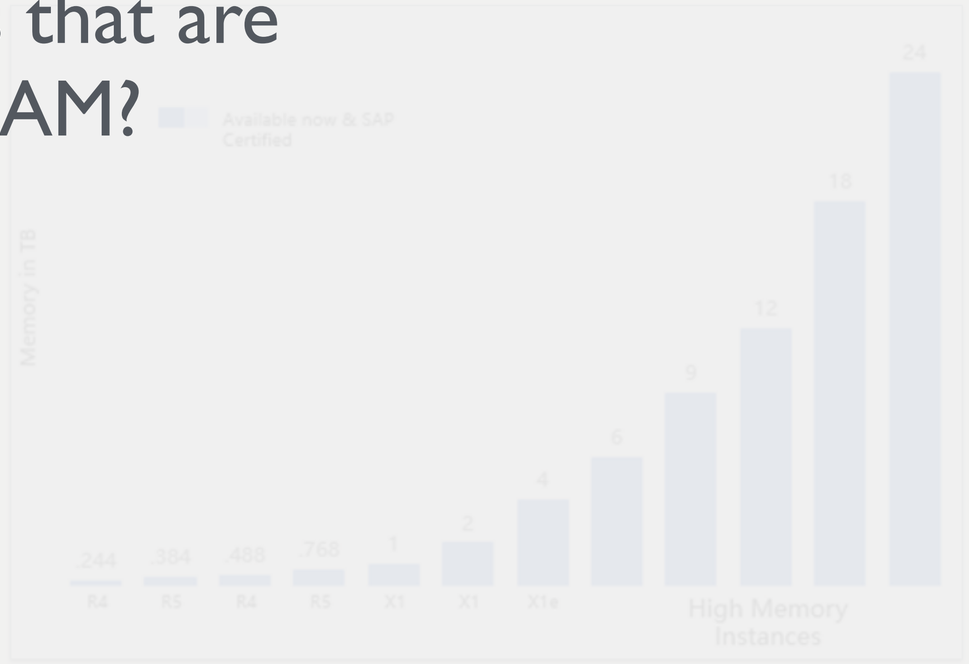
- Cost for a 1TB memory machine with 72 processors is about $20,000.
- Can rent a similar machine (96 processors and 1.5TB memory) for $11/hour on Google Cloud

WebDataCommons Graph

- 3.5 billion vertices and 128 billion edges

A single shared-memory machine can already store the largest publicly available graph datasets, with plenty of room to spare

What about graphs that are larger-than-DRAM?

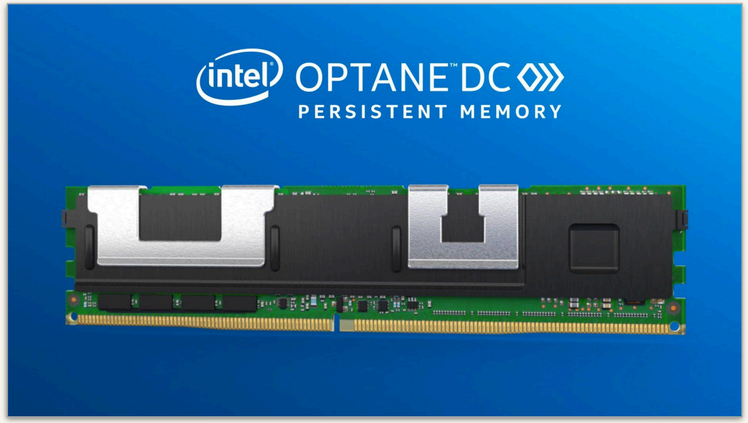


NVRAM Graph Processing

Non-Volatile Memory (NVRAM)

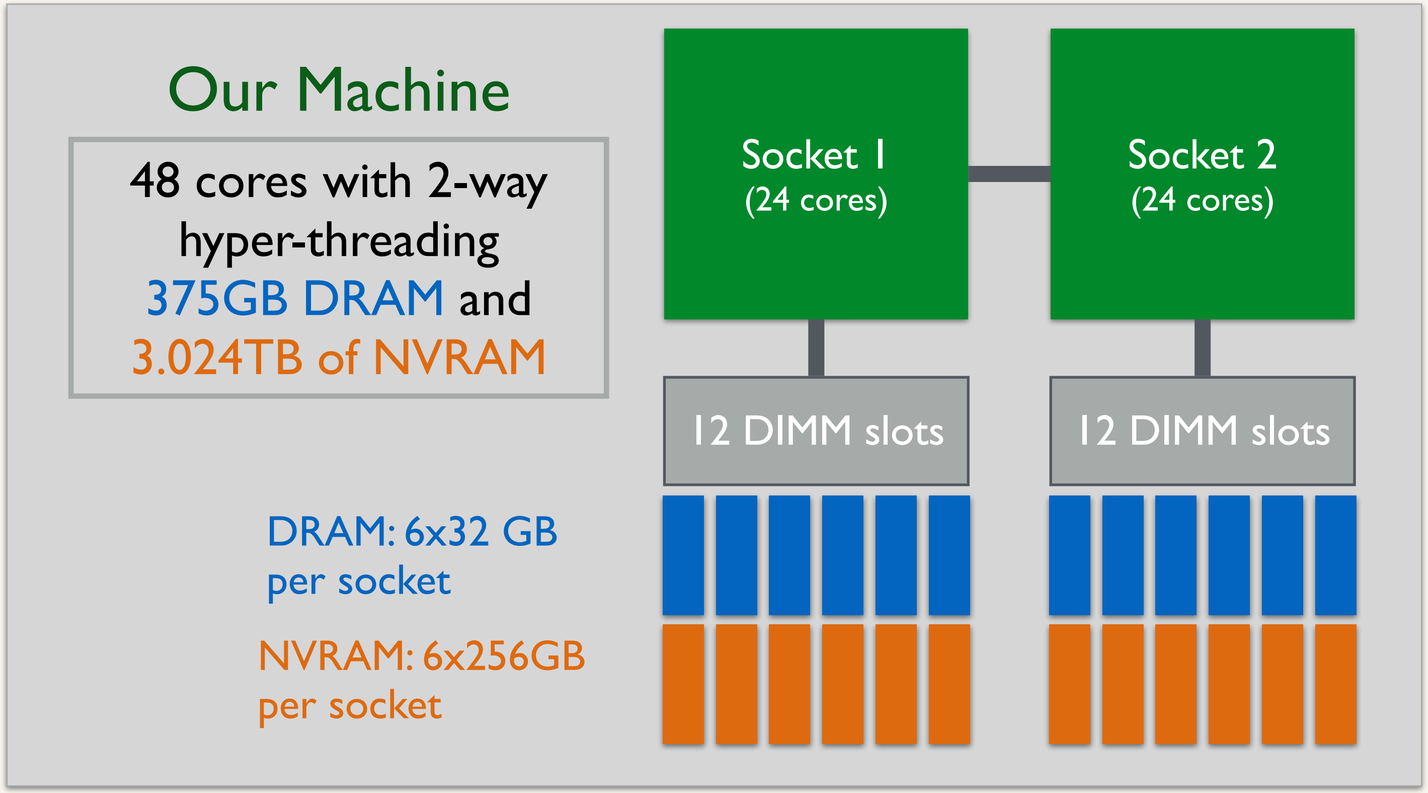
Intel Optane DC Memory

- Cheaper than DRAM on a per-byte basis
- Order of magnitude more capacity
- Memory is *persistent* and *byte-addressable*



Can we design algorithms that effectively use NVRAM as a higher-capacity memory while achieving DRAM-competitive performance?

NVRAM Characteristics

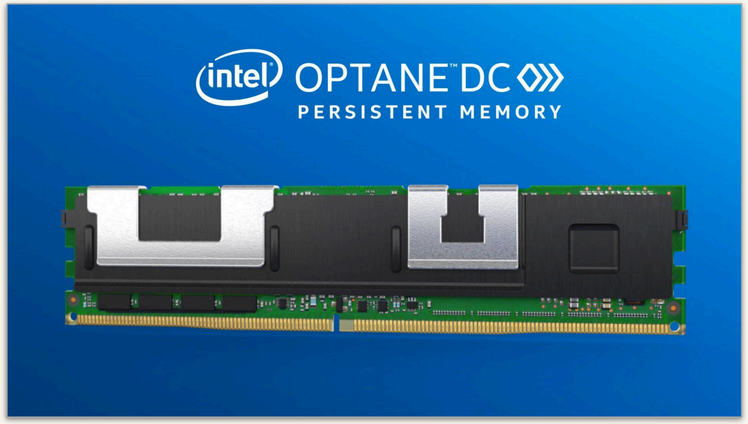


- 8x more NVRAM than DRAM
- NVRAM read throughput ~3x lower than DRAM read
- NVRAM write throughput further 4x lower

Recent work on Asymmetry

Benchmarking

- Two recent studies by Izraelevitz et al. [0] and van Renen et al. [1] perform careful benchmarking of Optane memory, and report similar asymmetries



Algorithms and Systems for Asymmetric Settings

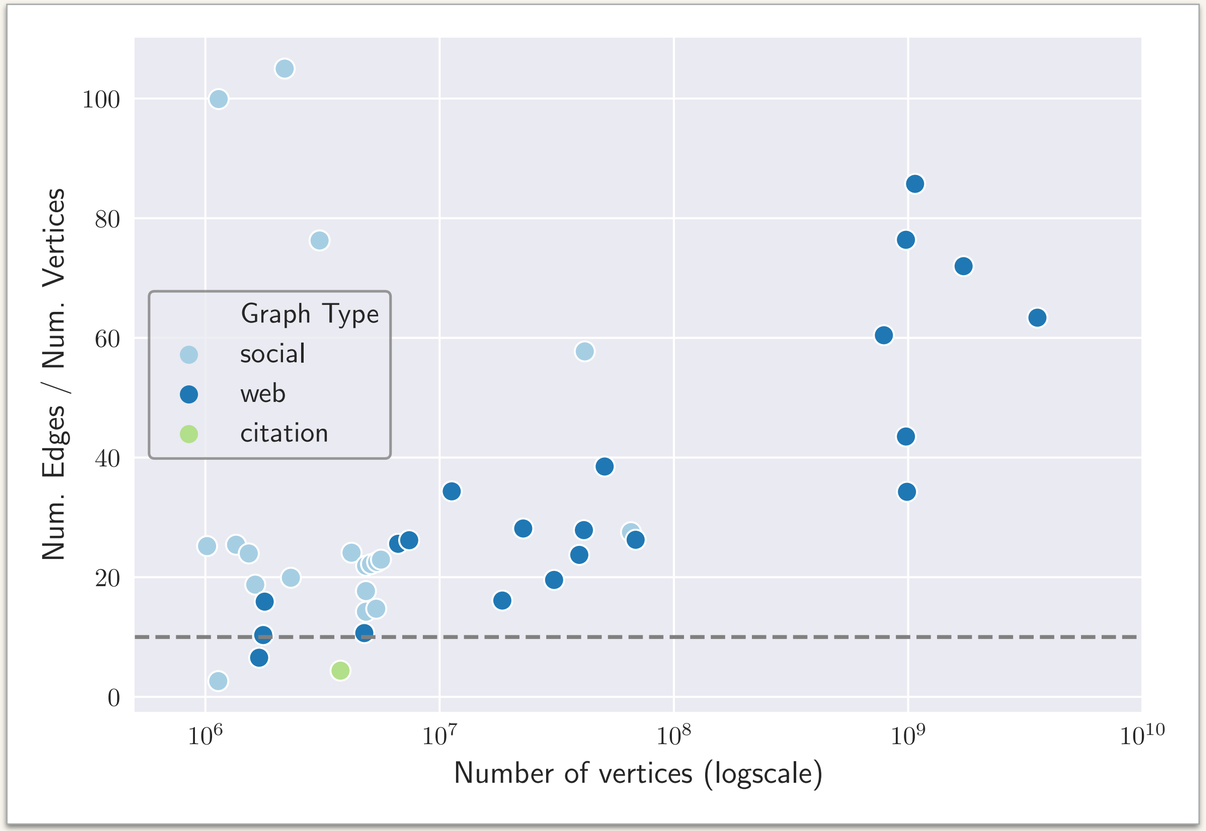
- Recent work explores how to minimize the number of NVRAM writes, e.g., [2 – 4], including many other papers
- Also significant work from systems, architecture, and database communities, e.g., [5 – 7], amongst many other papers

Sources:
[0] Izraelevitz et al. Basic performance measurements of the Intel Optane DC persistent memory module. (2019)
[1] van Renen et al. Persistent Memory I/O Primitives (2019)
[2] Ben-David et al. Parallel algorithms for asymmetric read-write costs (2016)
[3] Blelloch et al. Efficient algorithms with asymmetric read and write costs (2016)
[4] Carson et al. Write-avoiding algorithms (2016)
[5] Peng et al. System Evaluation of the Intel Optane byte-addressable NVM (2019)
[6] Ni et al. SSP: Eliminating Redundant Writes in Failure-Atomic NVRAMs via Shadow Sub-Paging (2019)
[7] Yang et al. An Empirical Guide to the Behavior and Use of Scalable Persistent Memory (2020)

Semi-Asymmetric Parallel Graph Algorithms for NVRAMs [DMKGBGS'20]

Can we design practical and theoretically-sound techniques to overcome read/write asymmetry for graph problems on NVRAMs?

Real World Graphs are not Ultra-Sparse



Over 90% of graphs with > 1M vertices from SNAP and LAW datasets have $m/n \geq 10$

We expect that ratio of NVRAM/DRAM in future systems will be similar (our ratio is 8x)

Sources:
https://snap.stanford.edu/data/
http://law.di.unimi.it/datasets.php

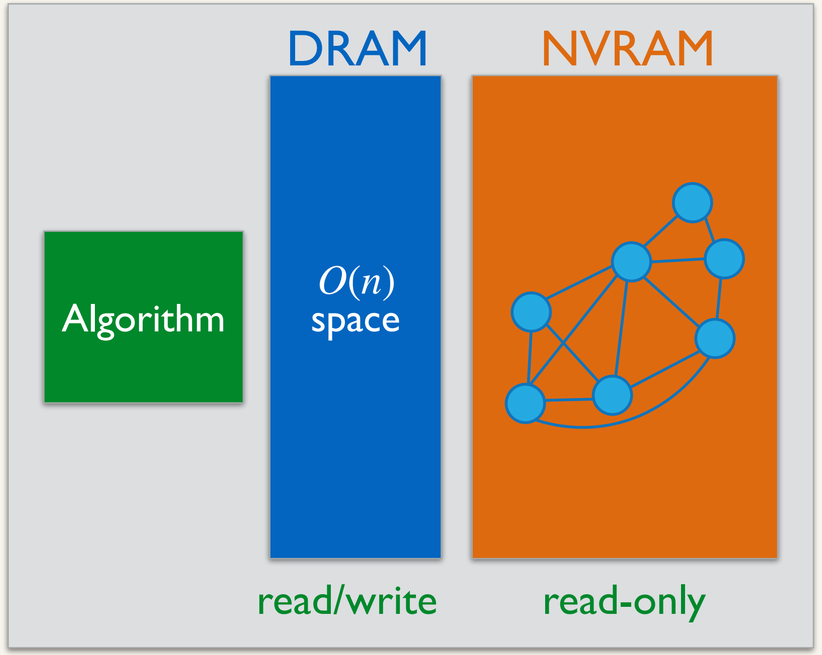
Our Approach

Semi-Asymmetric Approach

- Graph stored in NVRAM and accessed in a *read-only mode*
- Amount of DRAM is proportional to the number of vertices

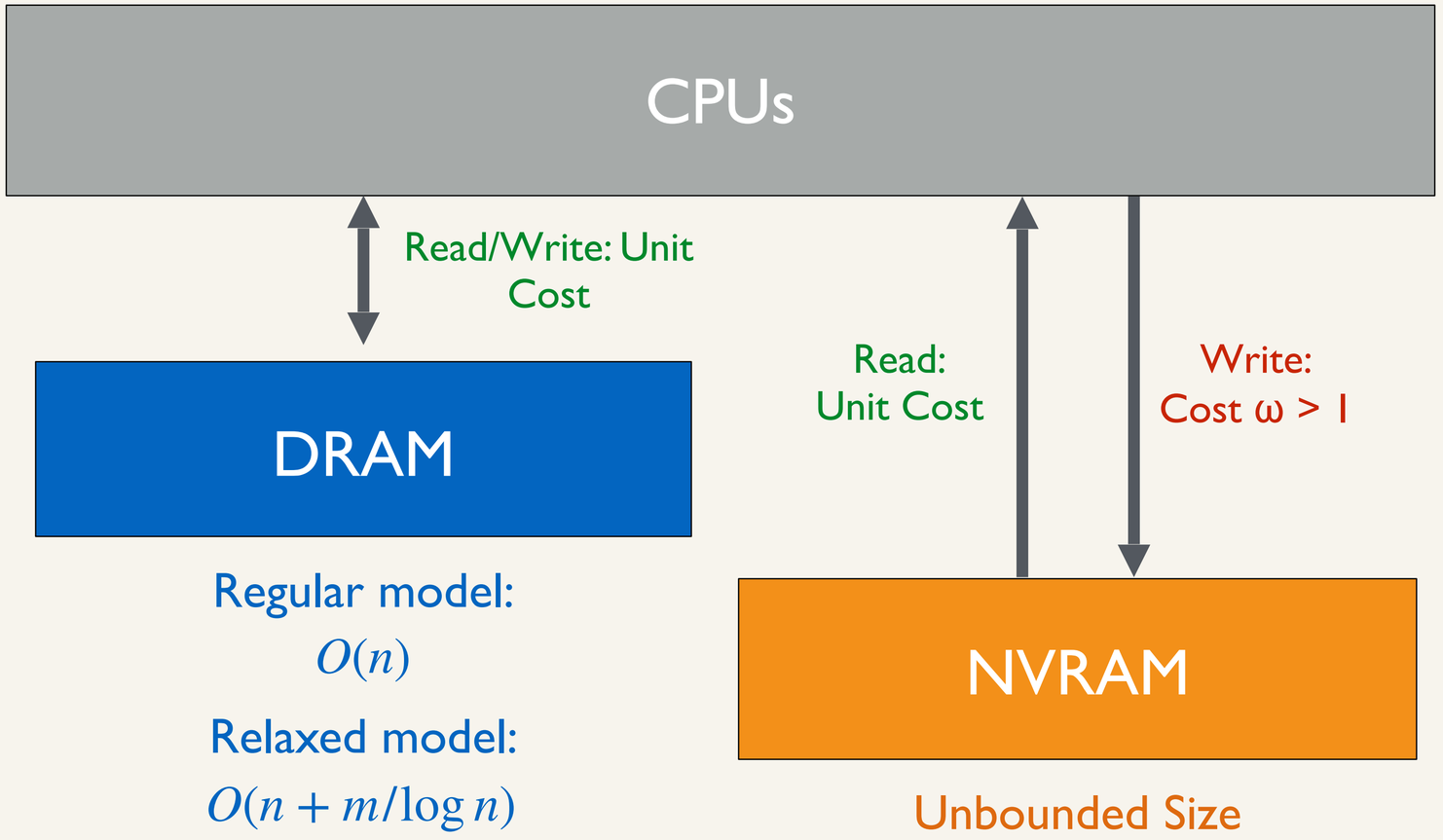
Benefits

- Algorithms avoid costly NVRAM writes, and algorithm design is independent of this cost
- Algorithms do not contribute to NVRAM wear-out



Our contribution:
This (restrictive) semi-asymmetric approach is effective for designing fast parallel graph algorithms

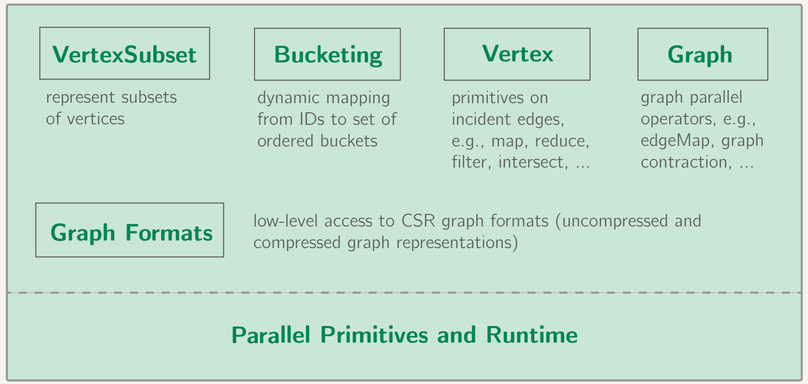
Parallel Semi-Asymmetric Model (PSAM)



Overview of Semi-Asymmetric Algorithms

- Start with work-efficient shared-memory algorithms from the Graph Based Benchmark Suite (GBBS)
- Implement interface primitives used by GBBS algorithms (*edgeMap* and *filtering*) efficiently in the PSAM

GBBS Interface



edgeMap

	Problem	GBBS Work	Sage Work	Sage Depth
EDGEMAPCHUNKED	**Breadth-First Search**	$O(\omega m)$	$O(m)$	$O(\mathsf{d}_G \log n)$
	Weighted BFS	$O(\omega m)^*$	$O(m)^*$	$O(\mathsf{d}_G \log n)^\ddagger$
	Bellman-Ford	$O(\omega \mathsf{d}_G m)$	$O(\mathsf{d}_G m)$	$O(\mathsf{d}_G \log n)$
	Single-Source Widest Path	$O(\omega \mathsf{d}_G m)$	$O(\mathsf{d}_G m)$	$O(\mathsf{d}_G \log n)$
	Single-Source Betweenness	$O(\omega m)$	$O(m)$	$O(\mathsf{d}_G \log n)$
	$O(k)$-Spanner	$O(\omega m)^*$	$O(m)^*$	$O(k \log n)^\ddagger$
	LDD	$O(\omega m)^*$	$O(m)^*$	$O(\log^2 n)^\ddagger$
	Connectivity	$O(\omega m)^*$	$O(m)^*$	$O(\log^3 n)^\ddagger$
	Spanning Forest	$O(\omega m)^*$	$O(m)^*$	$O(\log^3 n)^\ddagger$
	Graph Coloring	$O(\omega m)^*$	$O(m)^*$	$O(\log n + L \log \Delta)^*$
	Maxmial Independent Set	$O(\omega m)^*$	$O(m)^*$	$O(\log^2 n)^\ddagger$

GBBS work indicates the work of naively converting exisitng shared-memory algorithms from GBBS to NVRAM algorithms

Filtering (relaxed model)

	Problem	GBBS Work	Sage Work	Sage Depth
Both	**Biconnectivity**¶	$O(\omega m)^*$	$O(m)^*$	$O(\mathsf{d}_G \log n + \log^3 n)^\ddagger$
	Apx. Set Cover†	$O(\omega m)^*$	$O(m)^*$	$O(\log^3 n)^\ddagger$
Filter	**Triangle Counting**†	$O(\omega(m+n)+m^{3/2})$	$O(m^{3/2})$	$O(\log n)$
	Maximal Matching†	$O(\omega m)^*$	$O(m)^*$	$O(\log^3 m)^\ddagger$

Other Techniques

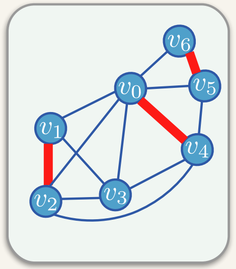
PageRank Iteration	$O(m+\omega n)$	$O(m)$	$O(\log n)$
PageRank	$O(P_{it}(m+\omega n))$	$O(P_{it} m)$	$O(P_{it} \log n)$
k-core	$O(\omega m)^*$	$O(m)^*$	$O(\rho \log n)^\ddagger$
Apx. Densest Subgraph	$O(\omega m)$	$O(m)$	$O(\log^2 n)$

Semi-Asymmetric Filtering

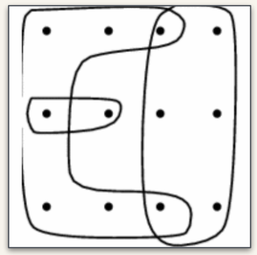
Motivation

- Some algorithms *remove*, or *batch-delete edges* over the course of their operation for work-efficiency
- Modifying the graph directly requires writing to NVRAM

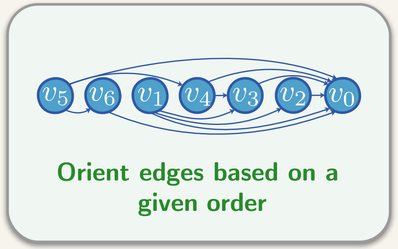
Maximal Matching



Parallel Approximate Set Cover

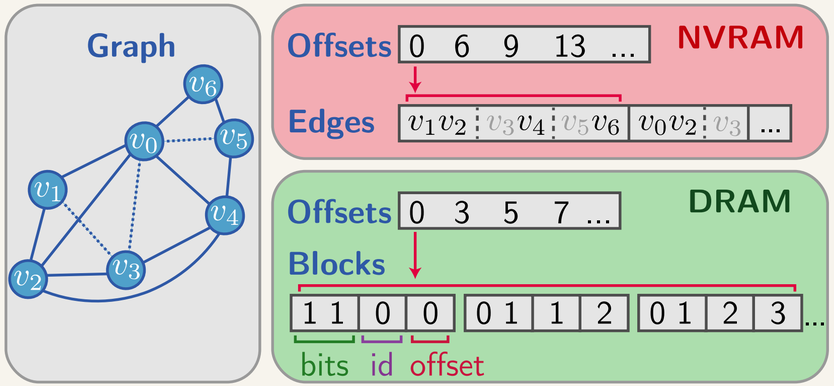


Triangle Counting

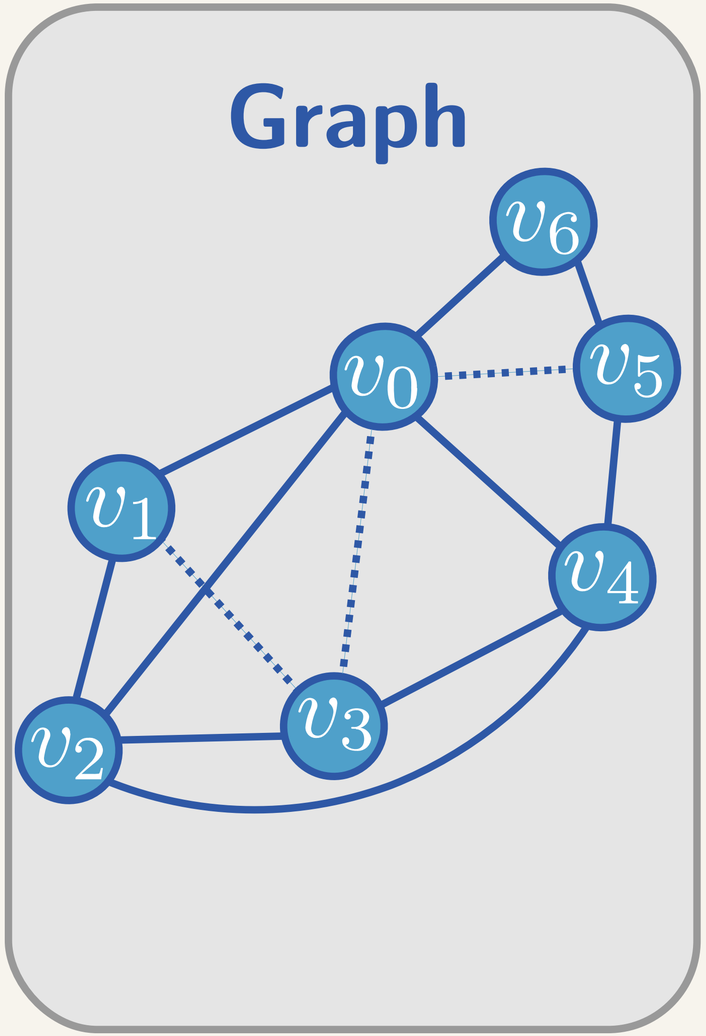


Semi-Asymmetric Filtering

- Work in the relaxed model
- Use one bit per edge and mirror the CSR structure (in NVRAM) using a blocked approach in DRAM



Semi-Asymmetric Filtering



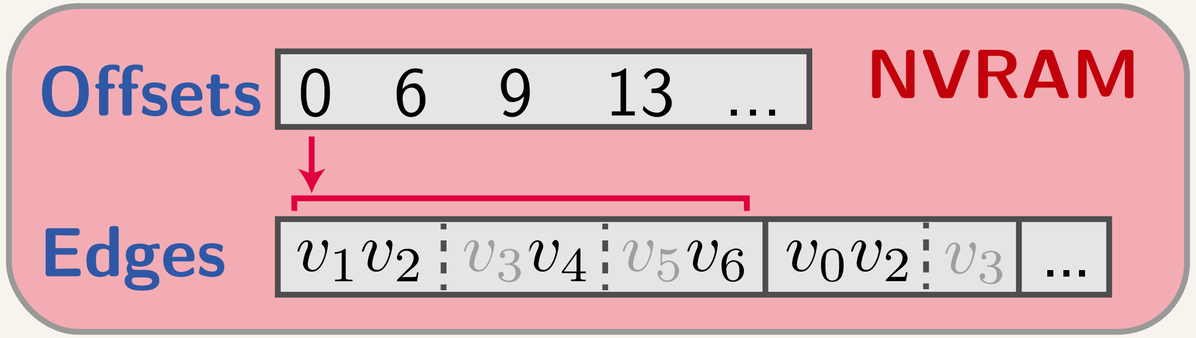
logically deleted

present in graph

Semi-Asymmetric Filtering

High-level Approach

(i) Set a filter block size, and logically chunk the CSR structure into chunks of this size

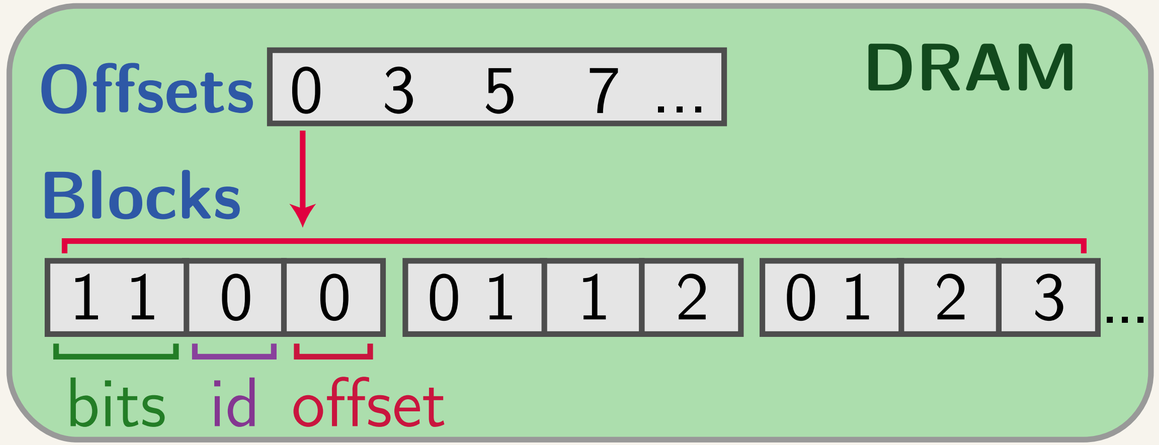


Graph in CSR format, stored in NVRAM $(\mathscr{F}_B = 2)$

Semi-Asymmetric Filtering

High-level Approach

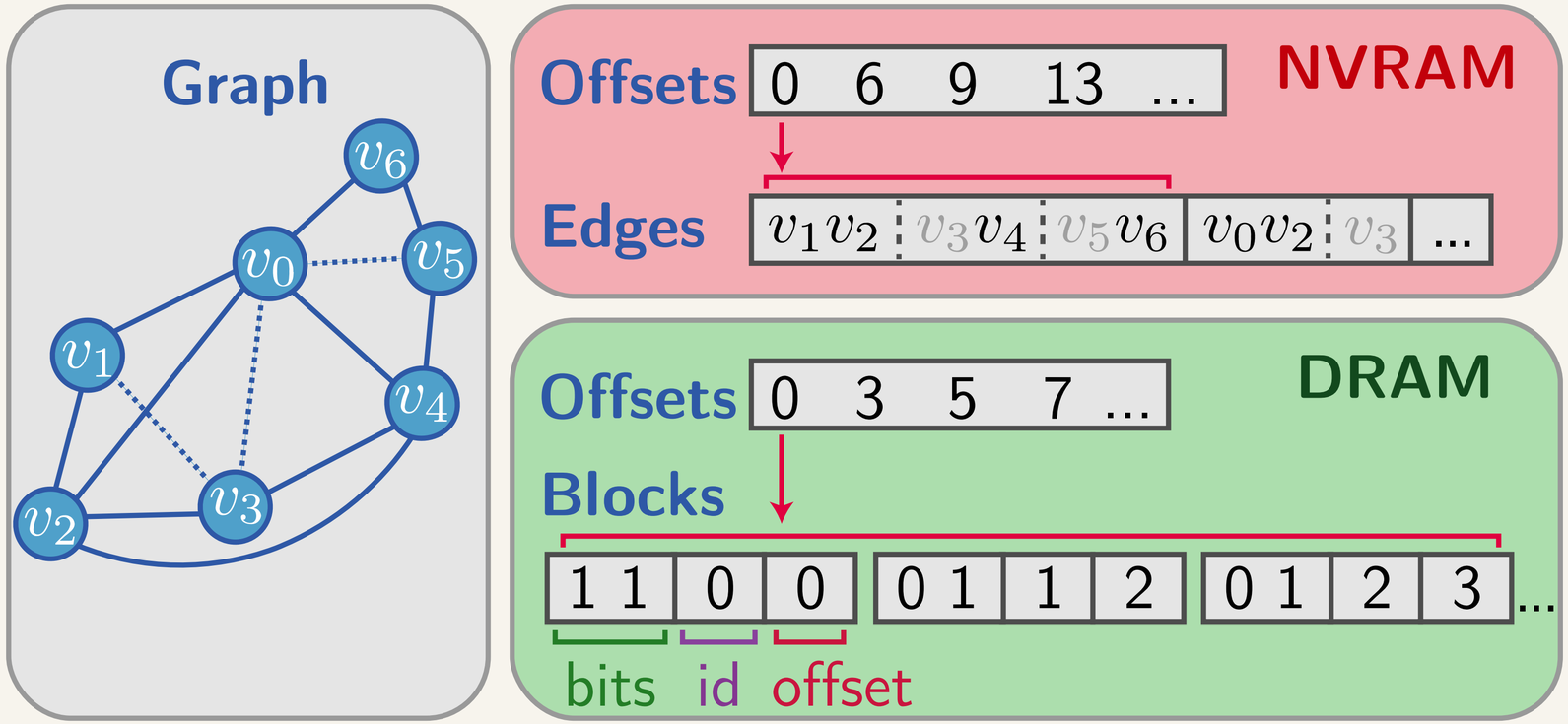
(ii) Create a "mirrored" filter structure in DRAM, storing 1 bit per edge in NVRAM



GraphFilter in CSR format, stored in DRAM ($\mathscr{F}_B = 2$)

Semi-Asymmetric Filtering

Structure Overview



Note: Blocks with no "1" bits remaining are deleted

Relationship to Other Models

Semi-External Memory (SE) Model

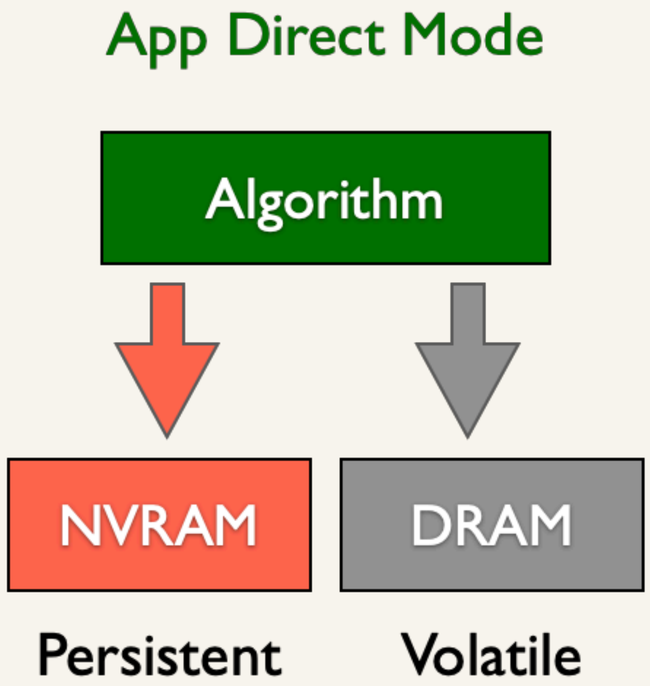
- SE model performs block-transfers, with a focus on I/O cost [0, 1]
- Both PSAM and SE models provide the same amount of DRAM, but SE does not account for DRAM reads and writes

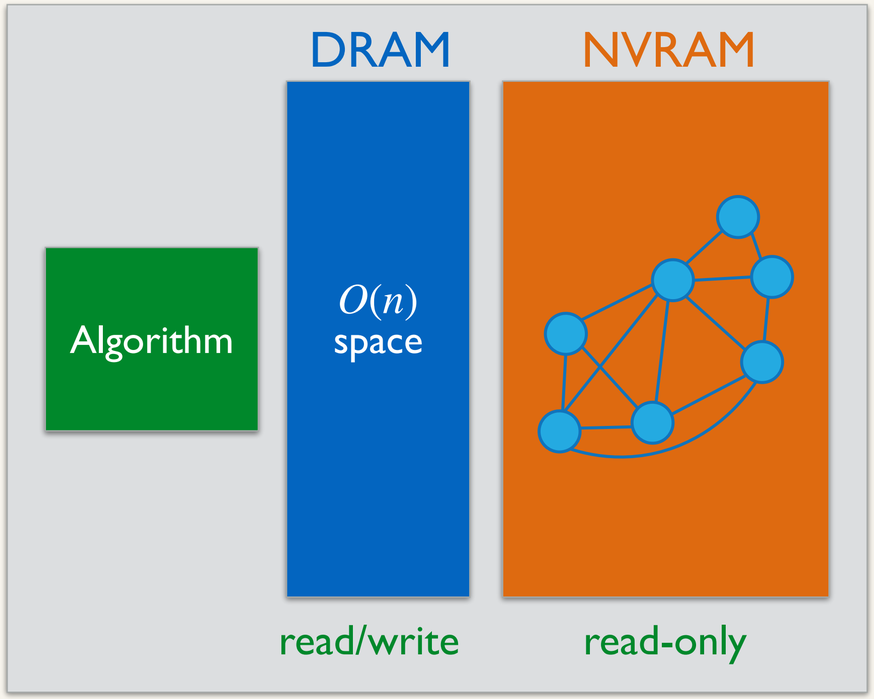
Asymmetric RAM and Asymmetric Nested Parallel Models

- Both ARAM [2] and ANP [3] models capture asymmetry of writing to NVRAM
- Unlike ARAM/ANP models, the PSAM includes a fast memory, and is specialized for graph problems

Sources:
[0] Abello et al. A Functional Approach to External Graph Algorithms (2002)
[1] Zheng et al. FlashGraph: Processing Billion-Node Graphs on an Array of Commodity SSDs (2015)
[2] Blelloch et al. Efficient algorithms with asymmetric read and write costs (2016)
[3] Ben-David et al. Parallel algorithms for asymmetric read-write costs (2016)

Semi-Asymmetric Graph Engine (Sage) Approach

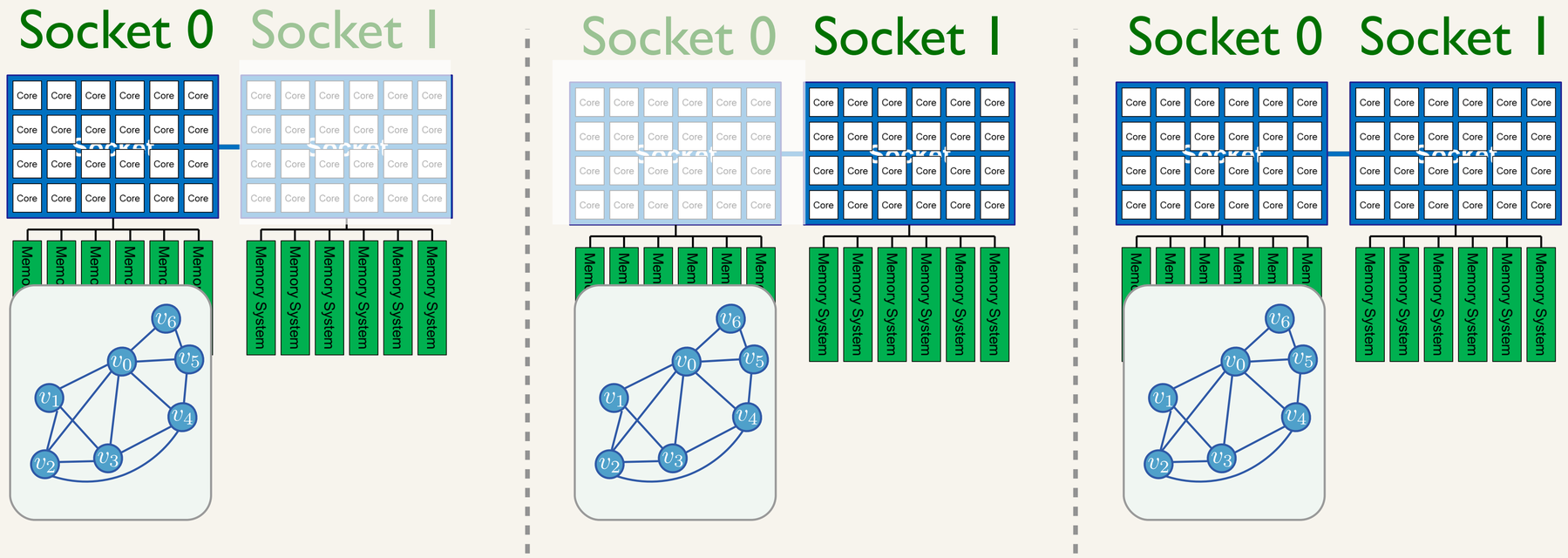




AppDirect Mode enables a direct implementation of PSAM algorithms

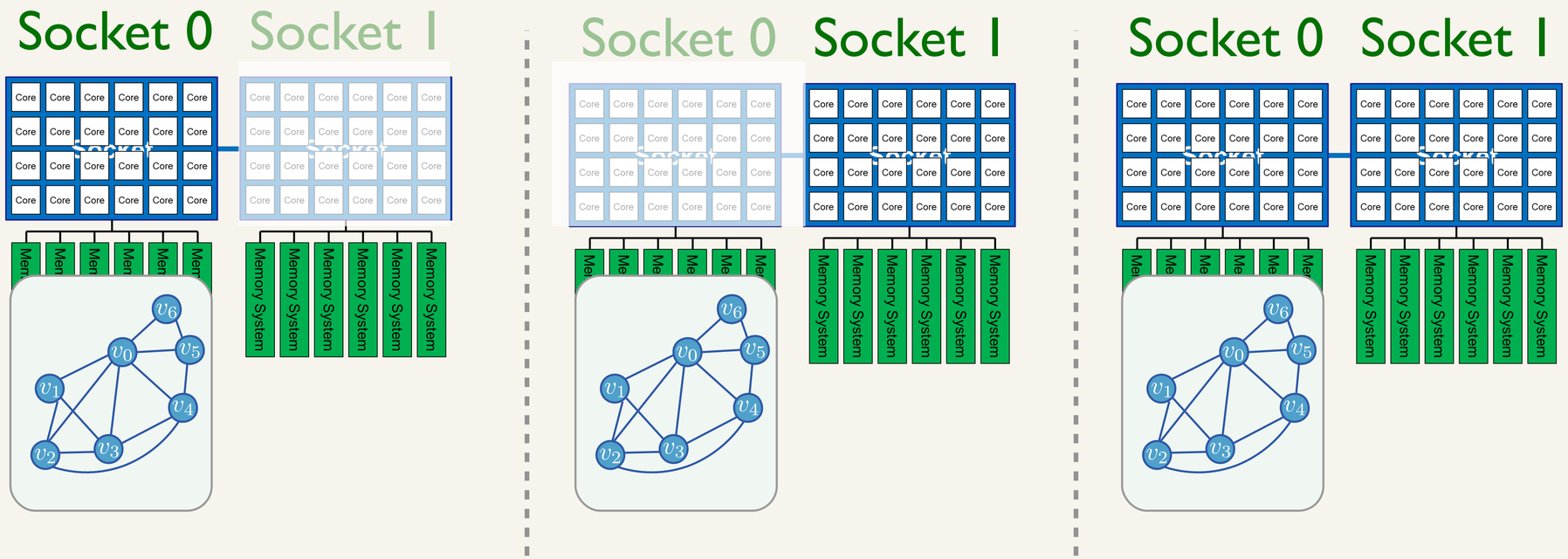
NUMA Optimization in Sage

Consider an algorithm that maps over all vertices, and for each vertex performs a reduction over the neighbors of the vertex



Three experiments based on (threads, storage)

NUMA Optimization in Sage



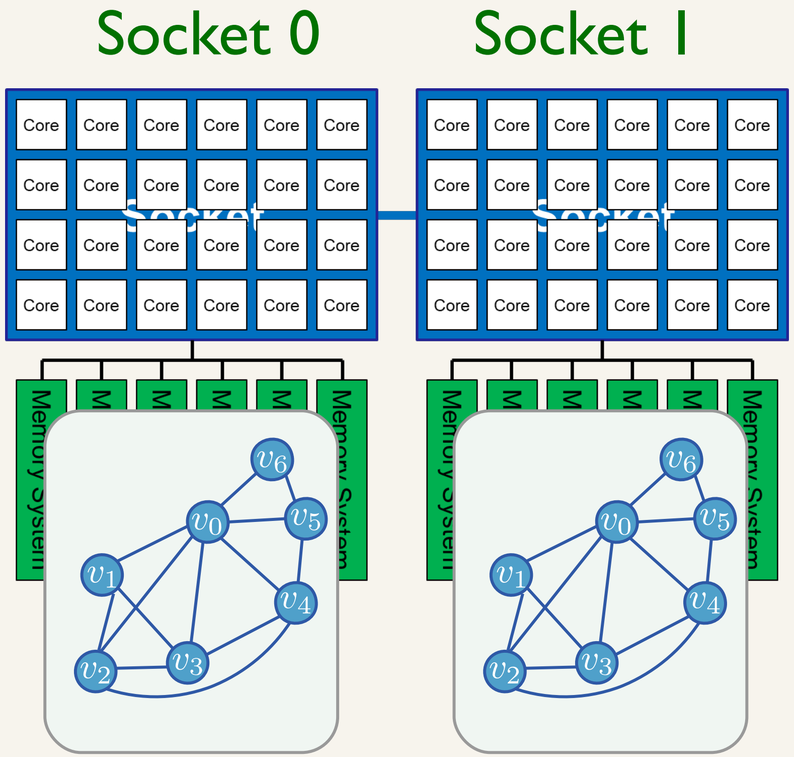
7 s

> 4x slower
first run
~7s subsequently

26 s

Cross-socket NVM reads should be avoided

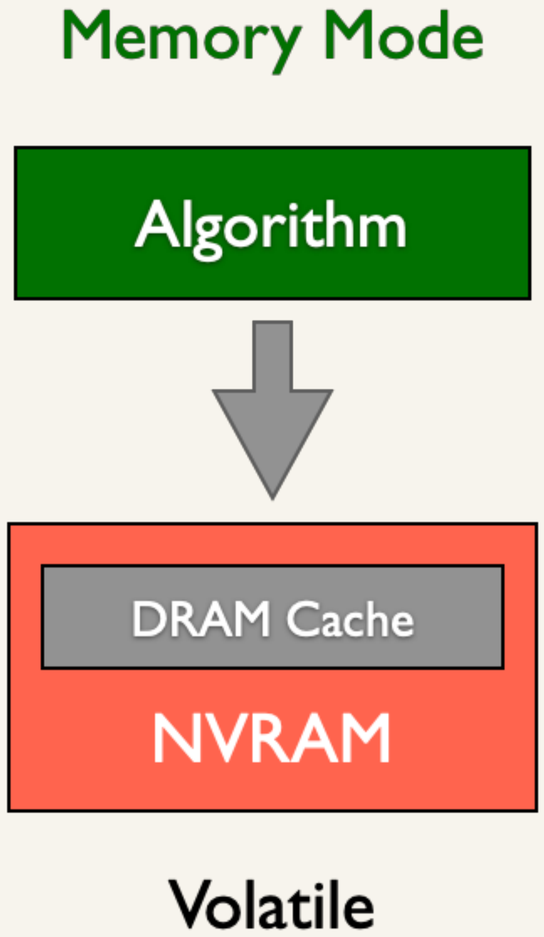
NUMA Optimization in Sage



4.3 s for microbenchmark

Both graphs stored in compressed CSR format

Existing Approaches: DRAM as a Cache



- Applications do not distinguish between DRAM and NVRAM
- Existing shared-memory software does not require modification
- Workloads that are larger than DRAM can involve costly NVRAM writes

Galois (Gill et al.)

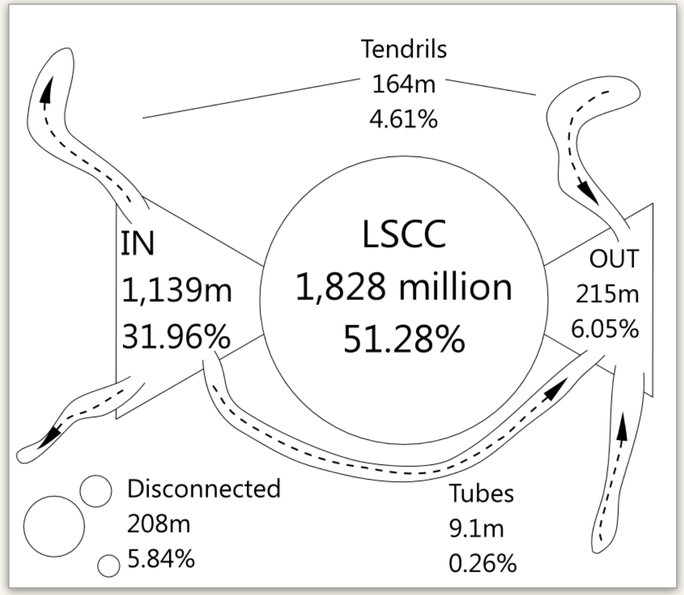
- Gill et al. study the performance of the Galois engine using MemMode
- They show promising results for scaling to larger than DRAM sizes

How does our approach compare?

Results for Larger-than-DRAM Graphs

WebDataCommons Graph

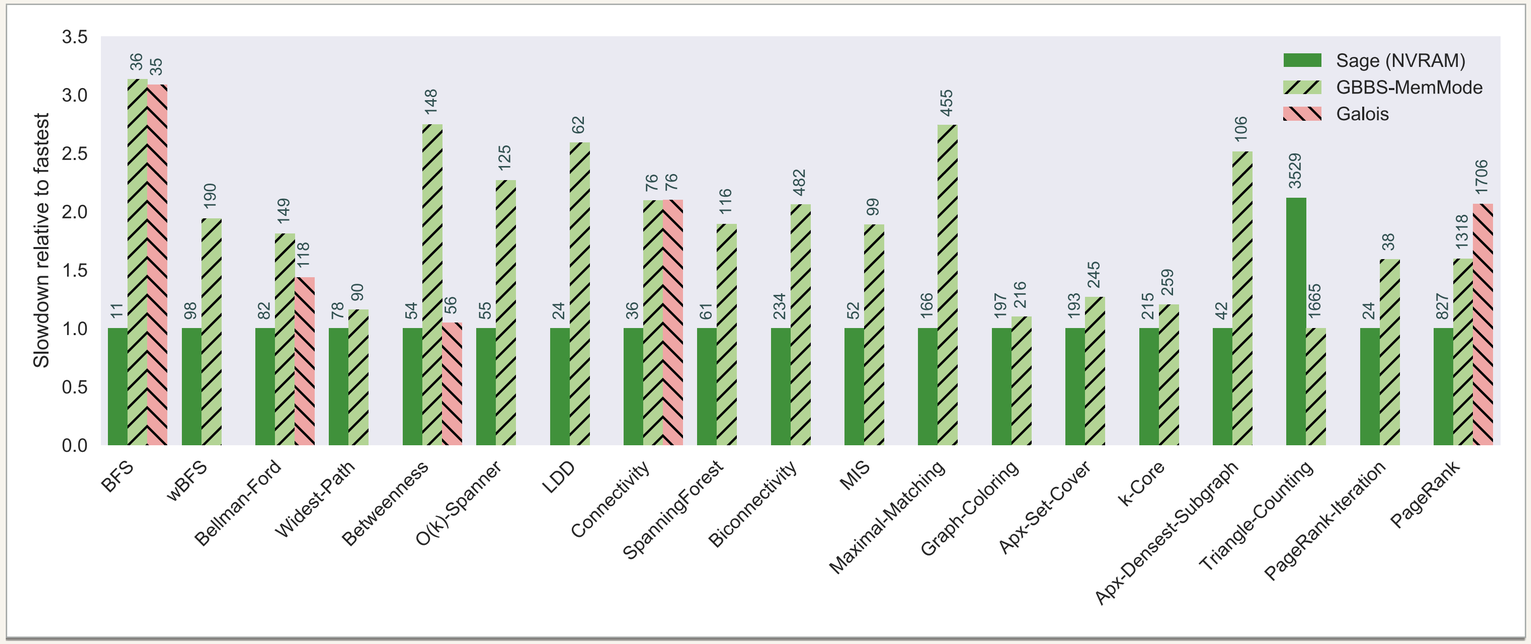
- Largest publicly available graph today
- 3.5B vertices connected by 128B edges (225B symmetrized)



Experiment

- Compare Sage results with
  - GBBS using MemMode (existing shared-memory codes)
  - Galois using MemMode (using numbers reported by authors on the same machine)

Results for Larger-than-DRAM Graphs



Run on a 48-core machine with 2-way hyper-threading, 375 GB of DRAM and 3 TB of NVRAM

1.94x speedup on average over Galois (state-of-the-art existing approach to NVRAM graph processing), and 1.87x speedup over simply running GBBS codes using MemMode

Results for Graphs Stored in Main Memory

ClueWeb Graph

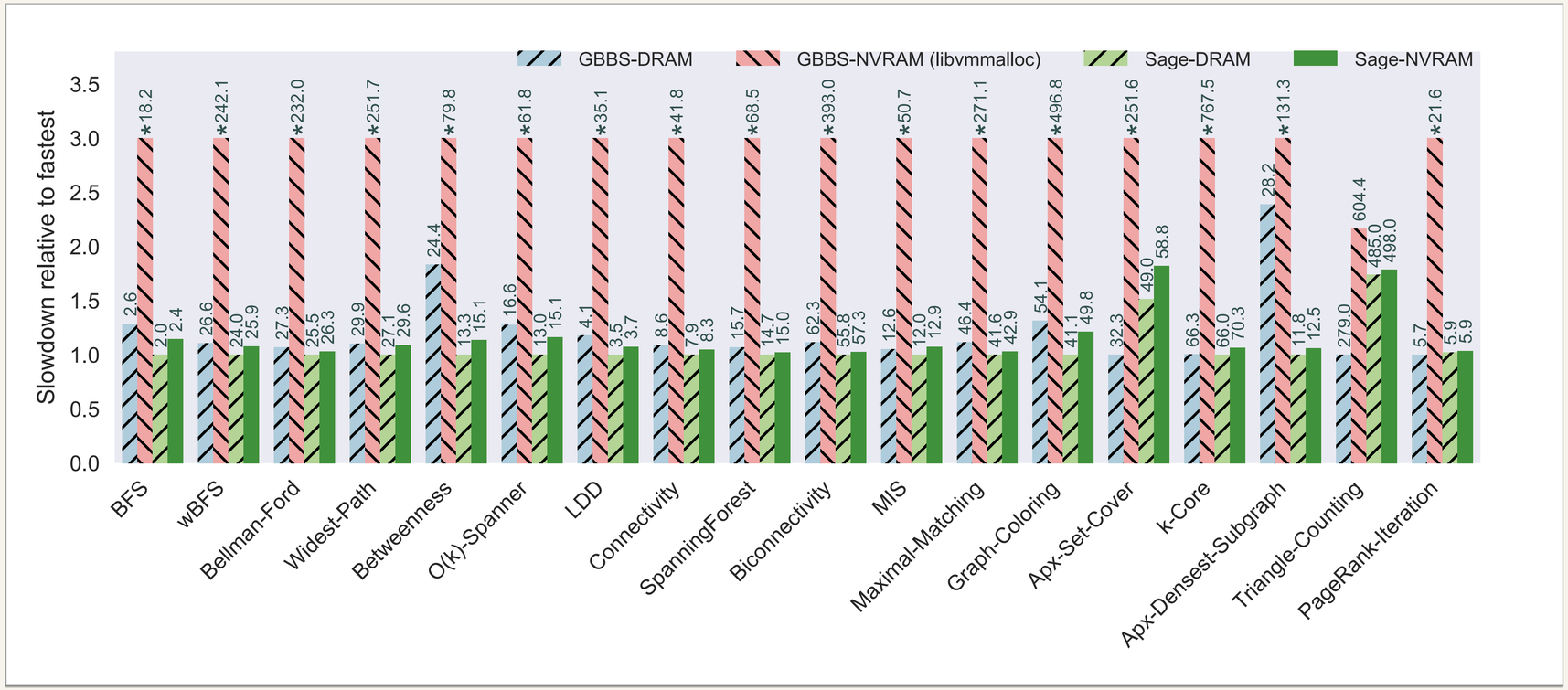
- Large web crawl with ~1B vertices connected by 42B edges (74B symmetrized)
- Graph fits entirely in the main memory of our machine

Experiment

- Compare Sage (graph stored on NVRAM) with
  - Sage (graph stored in DRAM)
  - GBBS (graph stored in DRAM)
  - GBBS with libvmmalloc (graph stored on NVRAM)

libvmmalloc: see https://pmem.io/pmdk/libvmmalloc/

Results for Graphs Stored in Main Memory



Run on a 48-core machine with 2-way hyper-threading, 375 GB of DRAM and 3 TB of NVRAM

Sage provides DRAM-competitive performance even when reading graph from NVRAM (only 5% slower on average)

Lessons and Directions for Future Work

Avoid Cross-Socket NVRAM Traffic

- NUMA optimization which reads from the copy of the read-only graph from the same socket achieves 6x speedup over cross-socket approach

Utilize App-Direct Mode

- Nearly 2x improvement for App-Direct based PSAM algorithms over two fast Memory Mode approaches

Avoid NVRAM Writes

- PSAM implementations which only read from NVRAM are over 6x faster than our algorithms which write to NVRAM (using libvmmalloc)