Gnosis Research Center @ ILLINOIS TECH

Enlarging Effective DRAM Capacity through Hermes

Luke Logan

Llogan@hawk.iit.edu

Gnosis Research Center @ Illinois Tech

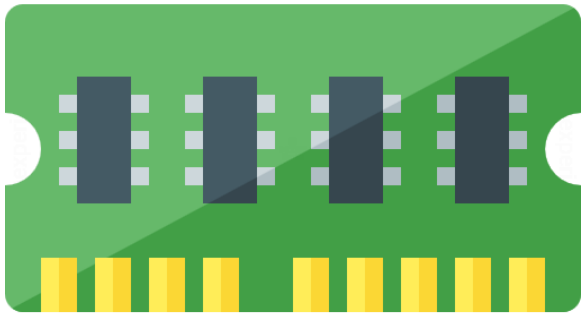
Logan, Luke, Xian-He Sun, and Anthony Kougkas. "MegaMmap: Blurring the Boundary Between Memory and Storage for Data-Intensive Workloads." *The International Conference for High Performance Computing, Networking, Storage, and Analysis*, 2024 (SC'24).

Memory-Centric, Data-Intensive Workloads

- Workloads are becoming increasingly data-intensive

The data volume is increasing beyond what main memory can hold!

Two Main Solutions

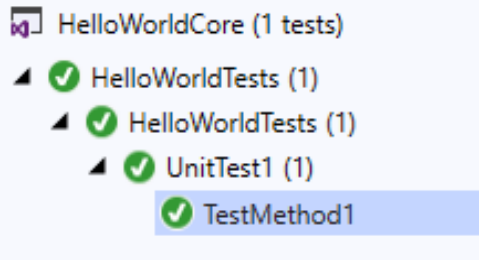


Just buy more DRAM…

Out-of-core Programming

Just Buy More DRAM

Pros:



Don't need to change applications

Cons:

DRAM is ***very*** expensive

DRAM has a ***very*** high energy cost

	Price	Capacity	$/GB
DRAM	$106	32GB	$3.30
PMEM	$275	128GB	$2.10
NVMe	$169	2TB	$0.08
SATA SSD	$175	4TB	$0.04
HDD	$369	18TB	$0.02

Out-Of-Core Programming

- Use storage to offload memory pressure

Pros:

Don't need to spend money on hardware

More energy efficient (less DRAM)

Cons:

Development Complexity

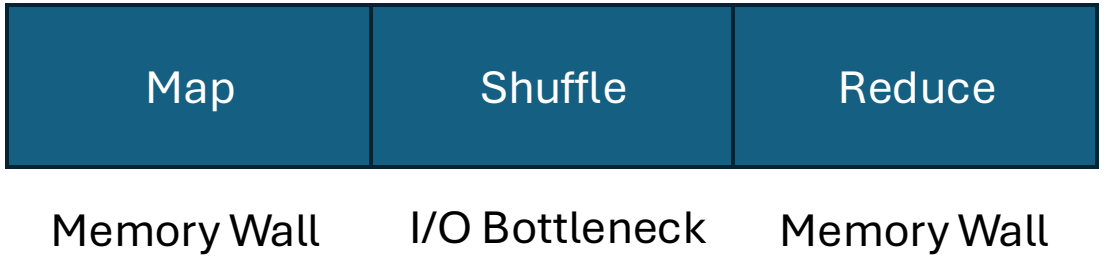
Suboptimal, one-off, manual solutions

Typical Out-Of-Core Programming Approach

- Have separate, synchronous phases for compute and I/O
- Incur memory wall in the compute phases
- Incur I/O bottleneck in the I/O phases
- **When combined, they make for a very slow program!**

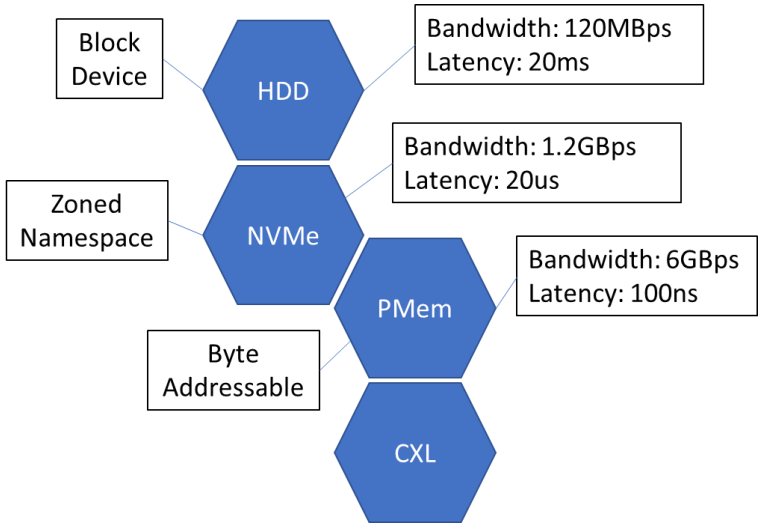
(e.g., many scientific simulations)

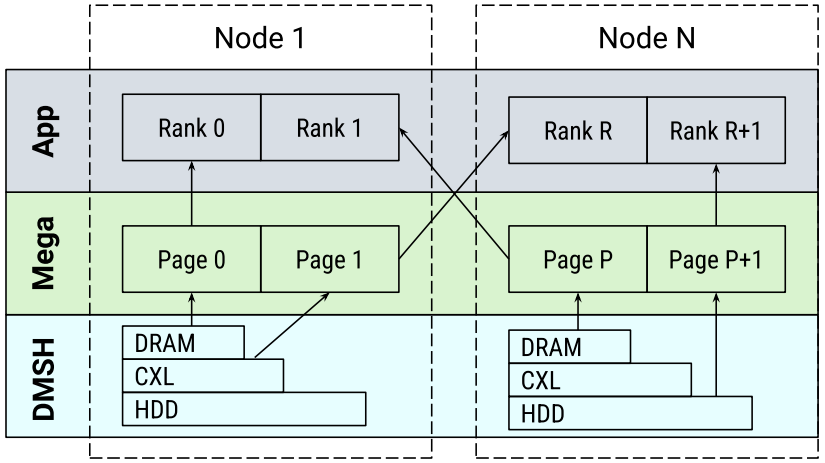
(e.g., Apache Spark)



We want an effectively infinite memory abstraction to interact with data, but without having to purchase more DRAM!

Blurring The Line Between Memory and Storage





- Storage has emerged with similar performance to DRAM
- Compute Express Link (CXL) is emerging that allows storage to be accessed as memory
- **We can combine DRAM and storage into a tiered distributed shared memory (DSM)**

But how can we efficiently hide this high-performance storage under a memory abstraction?

Limitations of Existing DSMs

- Agnostic to application behavior
  - Significant performance overhead to maintain coherence
  - Flurries of small messages to handle caching and replication
  - E.g., locking a page to ensure coherence
- DSMs are Cloud-focused
  - Provide replication to handle frequent hardware failures, incurring overhead
  - Must handle multi-tenancy, requiring acceptable performance for all but best for none
  - No optimizations for coordinated (e.g., MPI) workloads common in HPC
- No tiering
  - Most DSMs focus only on DRAM, but ignore storage tiers
  - Limited capacity

Tiered DSM Challenges

- Applications must be able to propagate access pattern intentions
  - Can reduce the need for fine-grained synchronizations
  - Can improve tiered data organization decisions
- Memory coherence policies must be optimized to address the characteristics of HPC
  - Coordinated workloads do not always need aggressive cache evictions
  - For example, read-only analysis workflows never need to invalidate caches
- New data placement + prefetching policies must be developed to address memory behavior, rather than storage

Our Approach: The MegaMmap Adapter

- A durable, persistent, and intuitive **DSM system** that allows massive datasets to be presented as memory
  - Reduces out-of-core complexity
- A user-driven **transactional memory access API**, which leads to improved decision-making in cache coherence and data organization policies by propagating memory access intent
- **Intent-aware memory coherence optimizations**, which improves the latency and bandwidth of memory accesses based on workload characteristics.
- **Tiered data organization policies**, which minimize I/O stall times by leveraging heterogeneous storage hardware and advance knowledge of access pattern intent.

An Infinite Nonvolatile Memory Abstraction

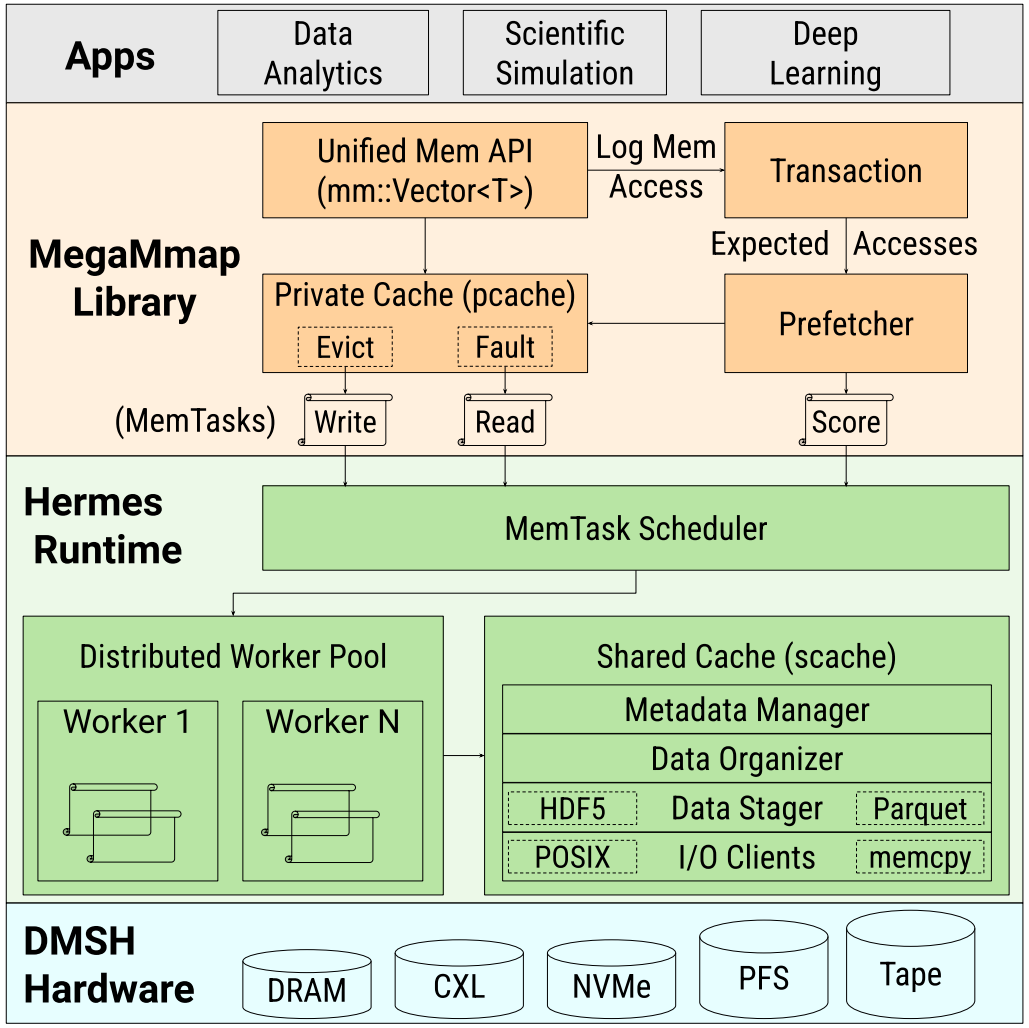
Presenting Datasets As Memory

```
1  #include <mega_mmap/vector.h>
2
3  void KMeansInertia(std::vector<Point3D> &ks) {
4    int rank = mpi::get_rank();
5    int nprocs = mpi::get_comm_size();
6    mm::Vector<Point3D> pts("/points.parquet");
7    pts.BoundMemory(MEGABYTES(1));
8    pts.Pgas(rank, nprocs);
9    float distance = 0;
10   tx = pts.SeqTxBegin(
11     pts.local_off(), pts.local_size(),
12     MM_READ_ONLY);
13   for (Point3D p : tx) {
14     distance += pow(NearestCentroid(p, ks), 2);
15   }
16   pts.TxEnd();
17   return distance;
18 }
```

- Mass datasets in memory
- Transactions mark the access pattern
- Transactions are not a significant burden to users since they are similar to iterators

Actively Ensuring Data Persistence and Consistency

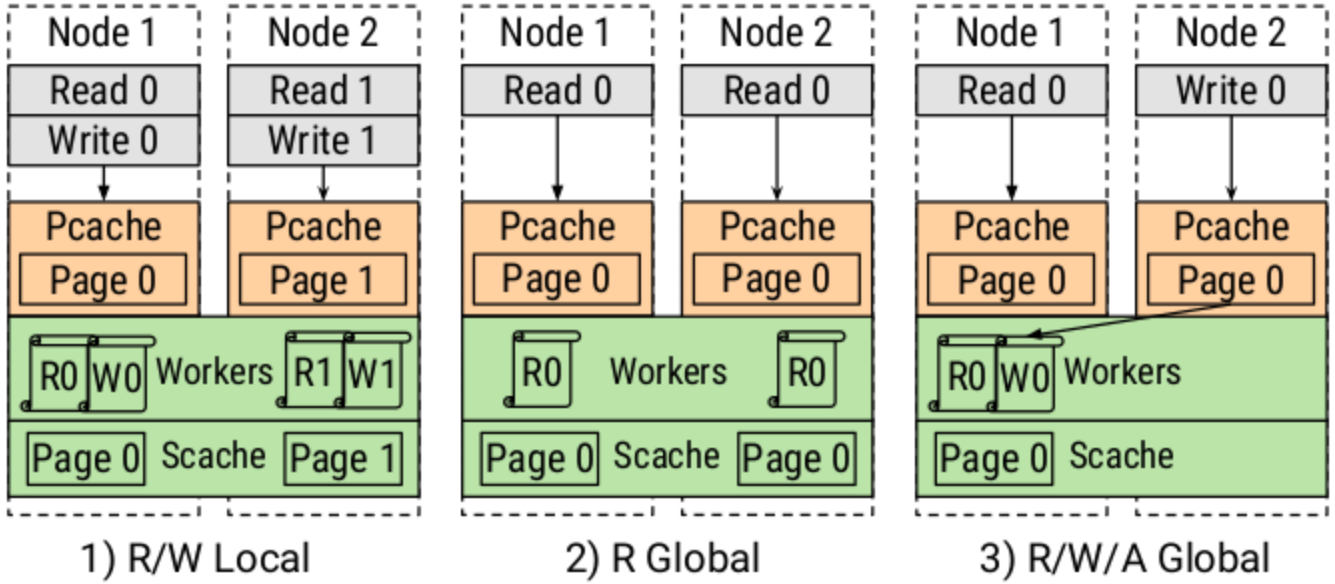
Actively Ensuring Persistence



- We use Hermes to persist data from applications
- Data must be persisted before the job ends
- The shared cache (Hermes) handles this situation
- The shared cache also handles consistency – all processes have the same view

Efficient Memory Coherence

An Infinite Non-Volatile Memory Abstraction



- R/W local avoids cache coherence overheads completely, since all data is accessed locally
- R global avoids cache coherence as well since no one modifies data
- R/W/A global minimizes the overhead using properties of queueing – modifications are sent asynchronously and sequenced

Masking I/O Stalls with Informed Tiered Data Movement Policies

Masking I/O Stalls

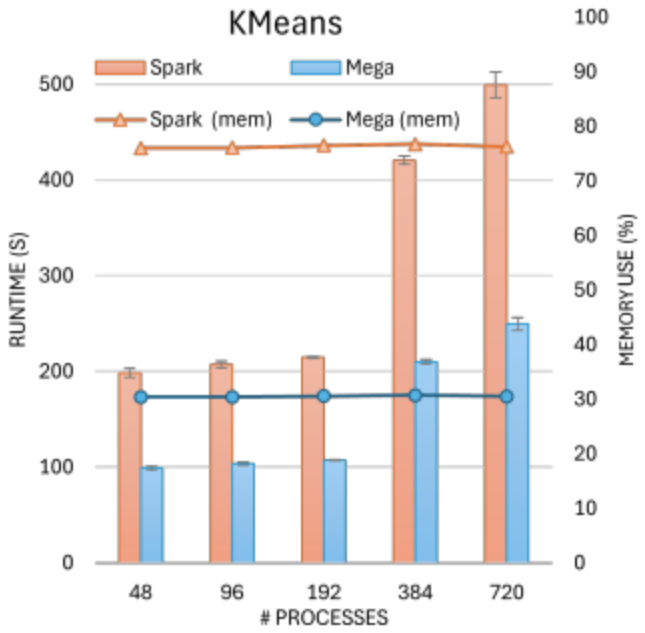
Algorithm 1 Private Cache Prefetcher

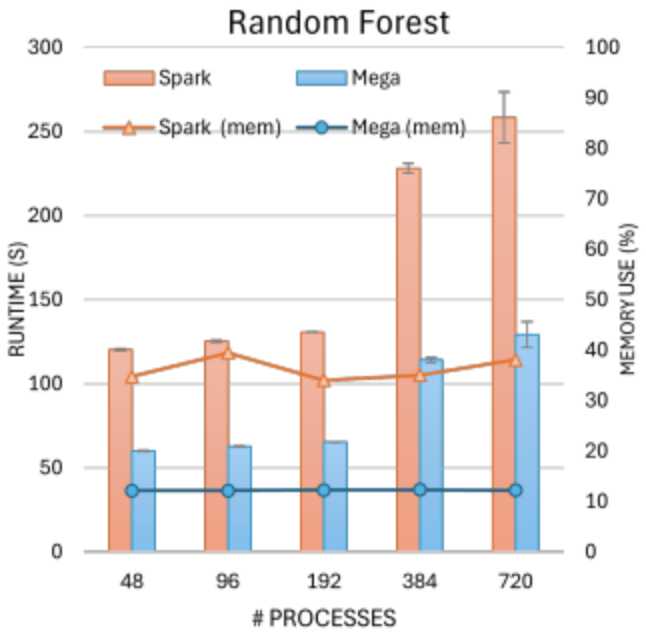
```
1:  function PREFETCHER(Vec, Tx, MinScore)
2:      Evict(Vec,Tx)
3:      Prefetch(Vec,Tx,MinScore)
4:      Tx.Head=Tx.Tail
5:  end function
6:  function EVICT(Vec, Tx)
7:      N=Vec.Max/Vec.PageSize
8:      for Page in Tx[Tx.Head:Tx.Tail] do
9:          Page.SetScore(0.0,Vec.NodeId)
10:     end for
11:     for Page in Tx[Tx.Tail:Tx.Tail+N] do
12:         Page.SetScore(1.0,Vec.NodeId)
13:     end for
14:     EvictIfZeroScore(Tx[Tx.Head:Tx.Tail])
15: end function
16: function PREFETCH(Vec, Tx, MinScore)
17:     BaseTime=0
18:     N=(Vec.Max−Vec.Cur)/Vec.PageSize
19:     for Page in Tx[Tx.Tail:Tx.Tail+N] do
20:         T=Page.GetTier()
21:         BaseTime+=Page.GetSize()/T.BW
22:     end for
23:     EstTime=BaseTime
24:     Score=1.0
25:     while Score>MinScore do
26:         Page=Tx[Tx.Tail+N]
27:         T=Page.GetTier()
28:         EstTime+=Page.GetSize()/T.BW
29:         Score=BaseTime/EstTime
30:         Page.SetScore(Score,Vec.NodeId)
31:         N=N+1
32:     end while
33: end function
```

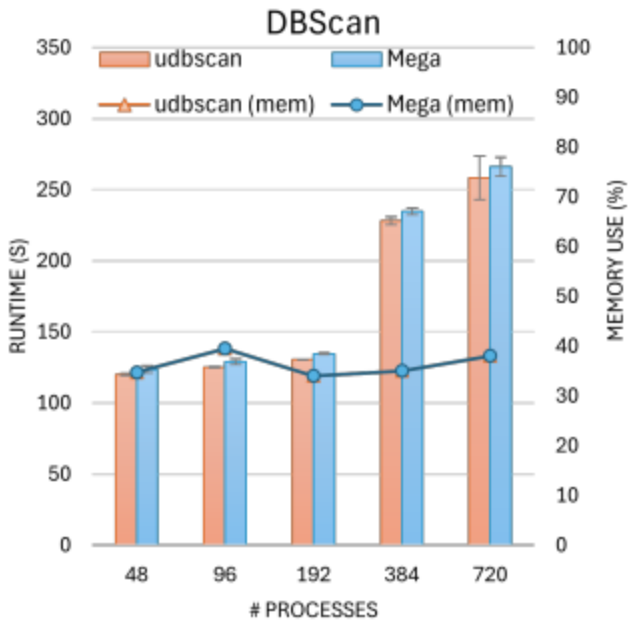
Evaluation

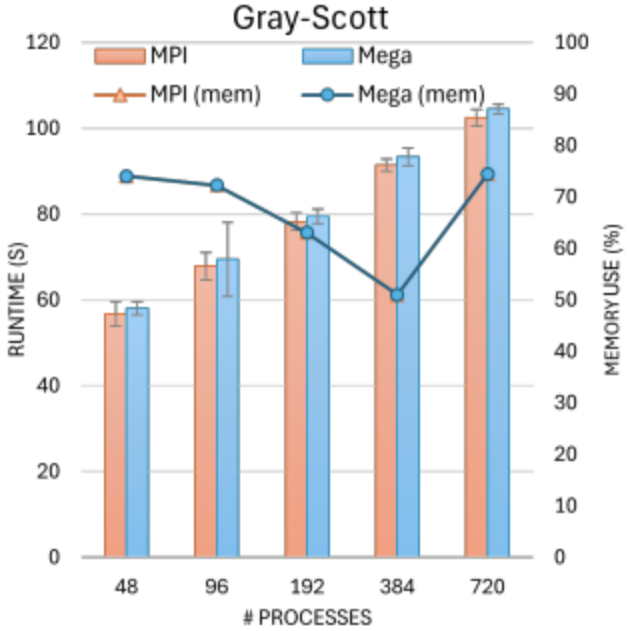
Evaluation Objectives

- Memory coherence of DSM are not a scalability bottleneck compared to leading HPC+Cloud communication solutions, such as MPI and Spark.
- Tiering memory can increase the resolution of scientific datasets by eliminating memory constraints, allowing more detail in the final simulation data.
- Intelligently tiering memory can bring performance benefits to out-of-core algorithms.
- DRAM consumption can be lowered by offloading memory to tiered storage.





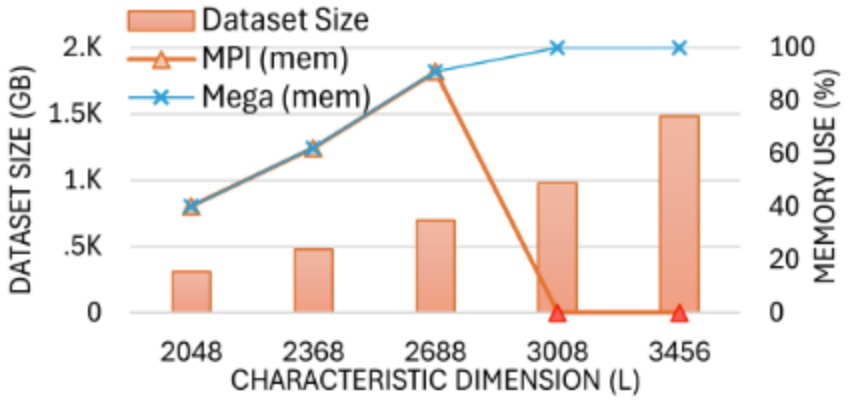




DSMs can perform well

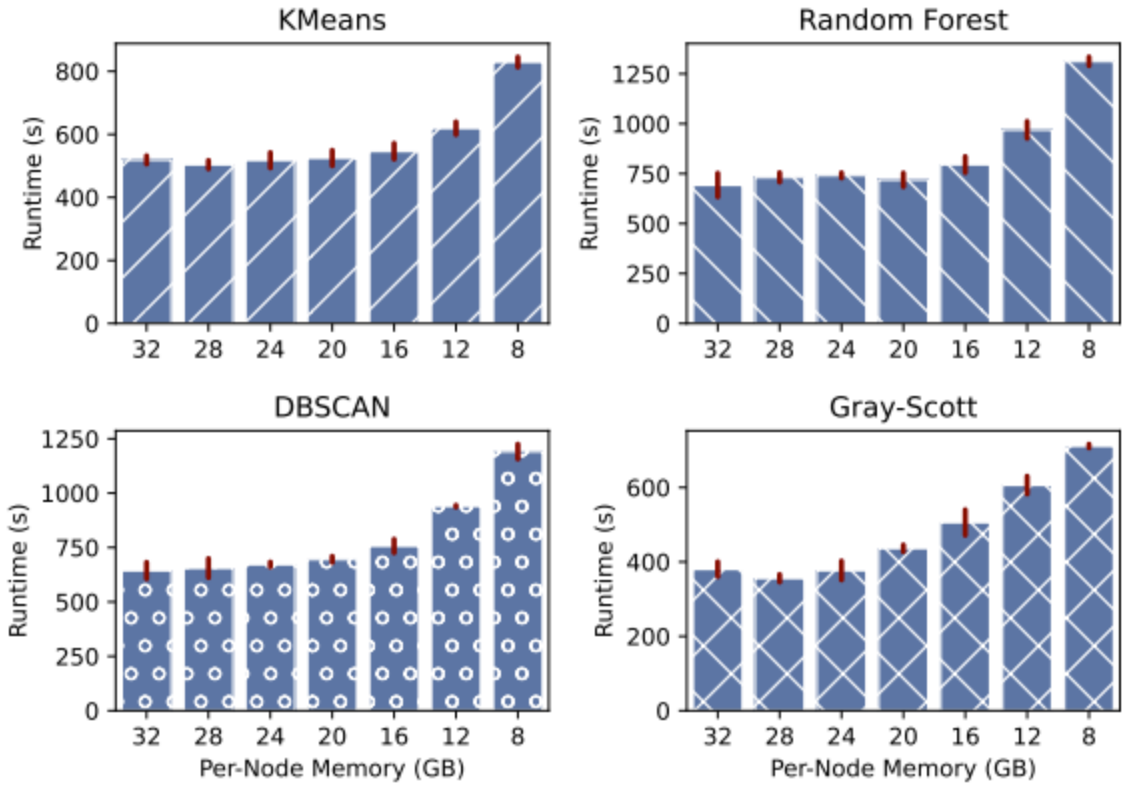
- MegaMmap scales just as well as MPI
- MegaMmap scales and performs better than spark

We can do more science!



- MegaMmap allows for twice the data to be processed without crashing

We can use less memory!



- MegaMmap algorithms can use half the memory with minimal performance impacts

Conclusion

Conclusion

- We demonstrate that the complexity of developing out-of-core algorithms can be reduced by enabling massive persistent datasets to be presented as memory
- We demonstrate how leveraging a transactional memory API can be used to reap substantial benefits in tiering performance.
- Evaluations showcase that developing algorithms with MegaMmap reduces peak memory utilization by as much as 2x