Algorithms for Non-Volatile RAM

Guy Blelloch
Carnegie Mellon University

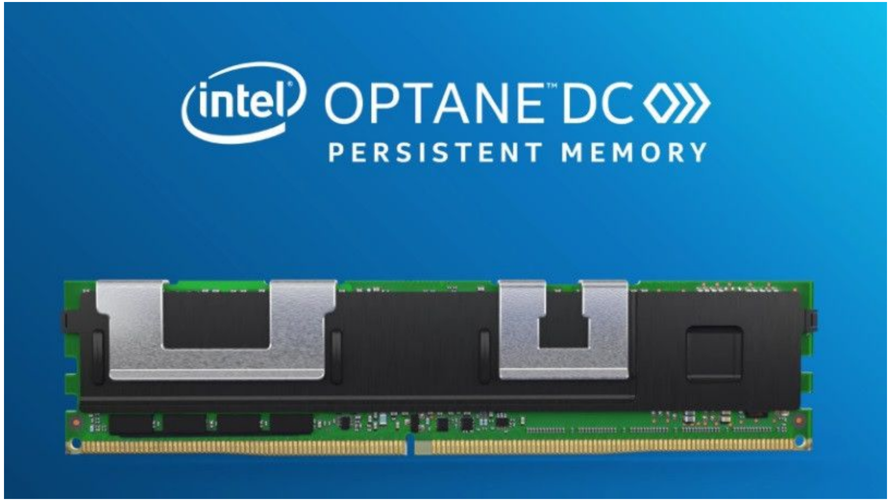
Contributions from many others including: Naama Ben-David, Laxman Dhulipala, Jeremy Fineman, Phillip Gibbons, Yan Gu, Hongo Kang, Charles McGuffey, Julian Shun, Yihan Sun, ++

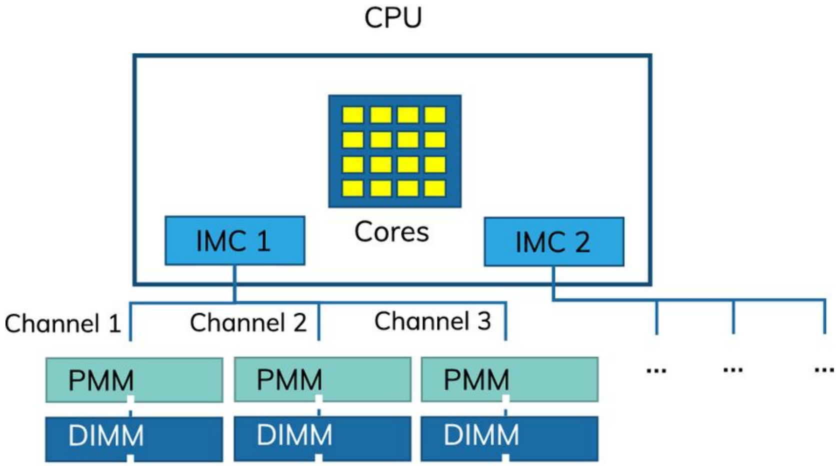
NVRAMS

Properties (of Intel Optane):

- + Plug in like any other RAM, byte addressable
- + Up to 512GB/DIMM (4 x RAM)
- + 32 Tbytes on an 4 chip server
- + No loss of data on power off
- - 2-3x slower for read throughput
- - 12x slower on write throughput

Other technologies on their way.





Write Efficient Algorithms

Try to reduce the number of writes.

Will outline three general techniques:

1. incremental updates
   - Reduce to sorting
2. “Anchors”
3. Rose

Focus on parallelism

Warmup : comparison-based sorting

How many writes are required?

- O(n) lower bound
- Quicksort, mergesort, heapsort? O(n log n)
- Any alternatives? Insert into a BST

Warmup : comparison-based sorting

For keys in random order

N = newNode(k_i)

P = pointer to root of a binary tree

While true do

If *P = null then

*P = N;

Break;

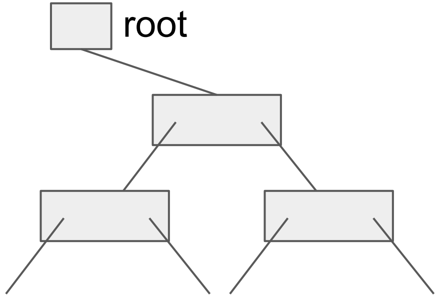
If N->key < *P->key then

P = &(*P->left)

else P = &(*P->right)

Only O(n) writes per insertion

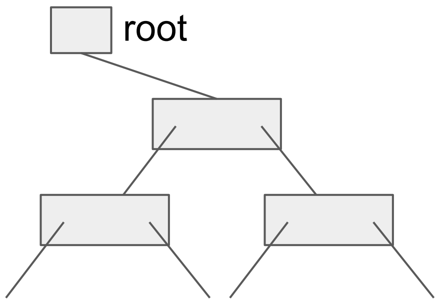
O(n log n) comparisons w.h.p.



Works on deterministic trees if number of rotations is O(n).

What about in parallel?

```
For keys in random order
  N = newNode(k_i)
  P = pointer to root of a binary tree
  While true do
     If *P = null then
        *P = N;
        Break;
    If N->key < *P->key then
       P = &(*P->left)
    else P = &(*P->right)
```



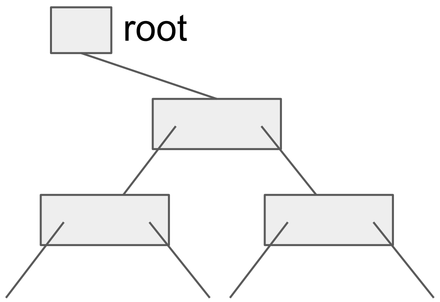
What about in parallel?

For keys in random order **in parallel**

```
  N = newNode(k_i)
  P = pointer to root of a binary tree
  While true do
     If *P = null then
        *P = N;   // priority write
        If (*P == N) Break;
    If N->key < *P->key then
       P = &(*P->left)
    else P = &(*P->right)
```

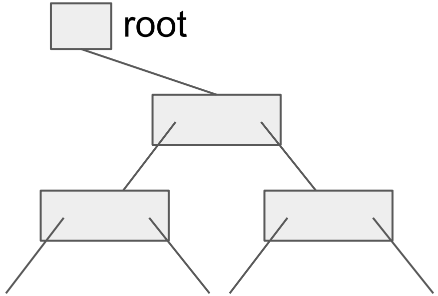
Is it write efficient?

No!



What about in parallel?

```
For j=1 to (lg n) sequentially
  for i=2^(j-1) to (2^j - 1) in parallel    // batch parallel with prefix doubling
    N = newNode(k_i)
    P = pointer to root of a binary tree
    While true do
      If *P = null then
        *P = N;   // arbitrary write
        If (*P == N) Break;
      If N->key < *P->key then
        P = &(*P->left)
      else P = &(*P->right)
```



O(n) writes

Some tricks to reduce span

Algorithms that can be reduced to sorting

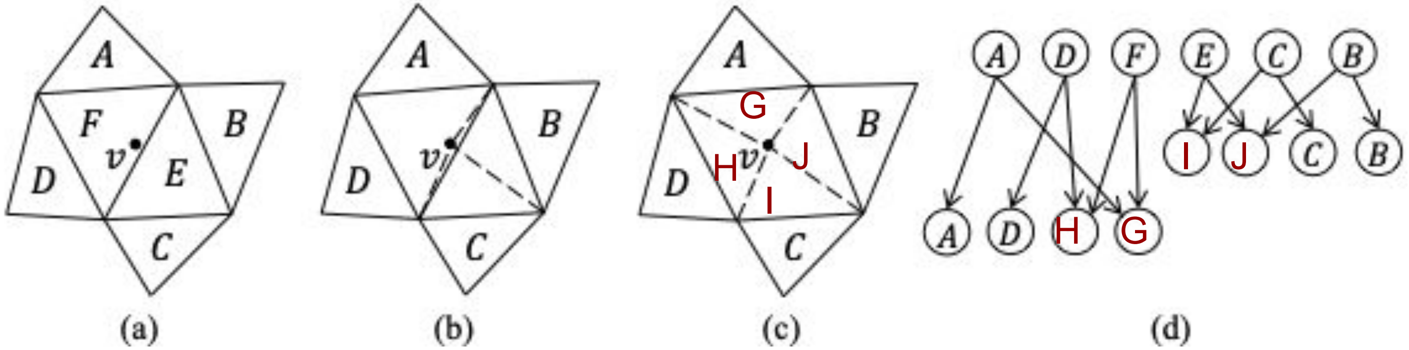
1. 2d Convex hull = sorting + O(n) reads and writes : Overmars + Van Leeuwen
2. Priority tree = sorting + O(n) reads and writes : BGSH
3. Interval tree = sorting + O(n) reads and writes : BGSH

All are parallelizable.

Other write-efficient random incremental algorithms

Random incremental Delaunay triangulation: (recently shown to be parallel)

- Each vertex adds 6 triangles in expectation, builds a search structure
- Tricky since search structure is a DAG, and searches can meet up.   Can't afford to write down what we searched.
- Developed a general DAG searching technique that requires O(log n) local memory if DAG has bounded degree.



Method 2: Anchors

Selecting a subset of elements such that writes are proportional to the size of the subset.

We use various different names (unfortunately):

- Centers : for graph connectivity
- Critical nodes : weight balanced trees, augmented values
- Partition nodes : tree contraction

Often involves starting with a random sample and then improving by adding more.

Sometimes involves a tradeoff (fewer writes = more reads)

Example: Graph Connectivity

Goal: support a data structure that supports

1. Build: graph -> struct with fewer than n writes
2. Query: struct x vertex x vertex -> bool in reasonable time

We achieve O(n/k) writes and O(kn) reads for build, and O(k) time for queries
Only for bounded-degree graphs.

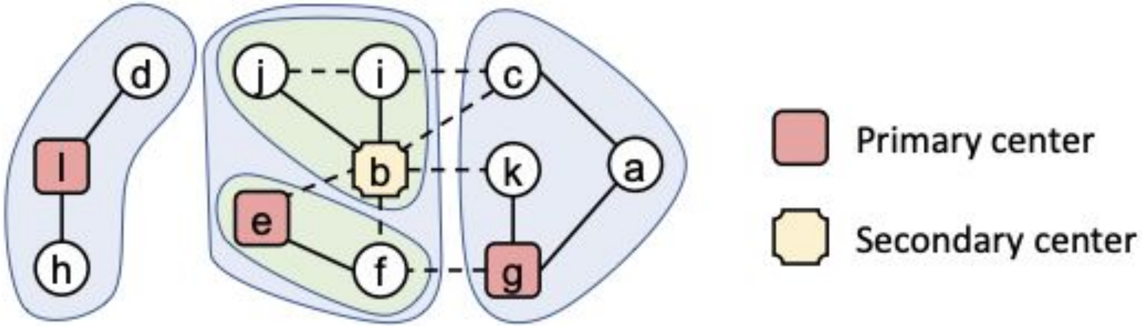
k is a parameter that can be adjusted (n = |V|)

Algorithm parallelizes.

Example: Graph Connectivity

Basic idea for finding centers (augmented random sampling):

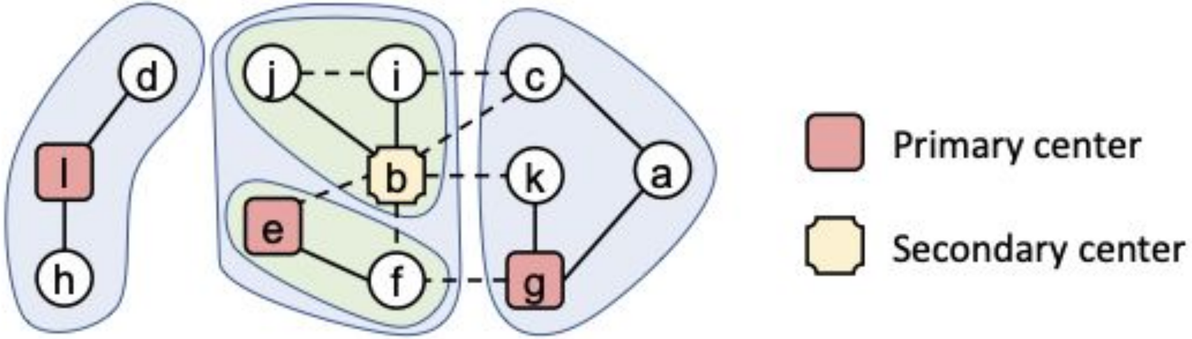
1. Pick random "primary" centers with probability 1/k.
2. Systematic BFS from each point to first center
3. Split BFS tree for each center with "secondary centers" so that resulting subtrees have size at most k



Example: Graph Connectivity

Using to build connectivity structure:

1. Run a connectivity algorithm using clusters as “supernodes”
2. Only stores info proportional to number of centers
3. For this step it is important that clusters are small (O(k)) so that we can list their edges.



Example : Partitioning for parallel tree contraction

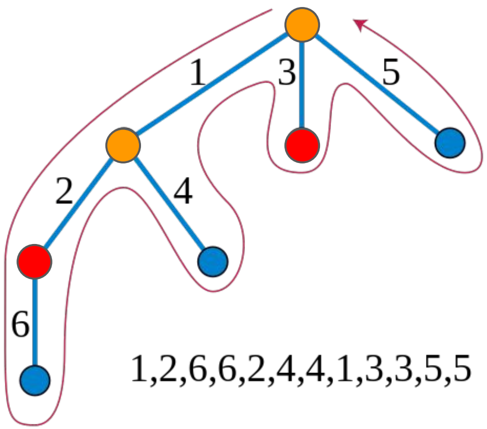
Input: a binary tree

Output: O(s) partition nodes that even split the tree

1. Take euler tour of the tree, and mark s nodes that partition it into runs of size approximately n/s (Uses augmented random sample)
2. Within each run find the highest node (closest to root) in the run
3. Add these to marked nodes from step 1.

Only need to write out O(s) data.

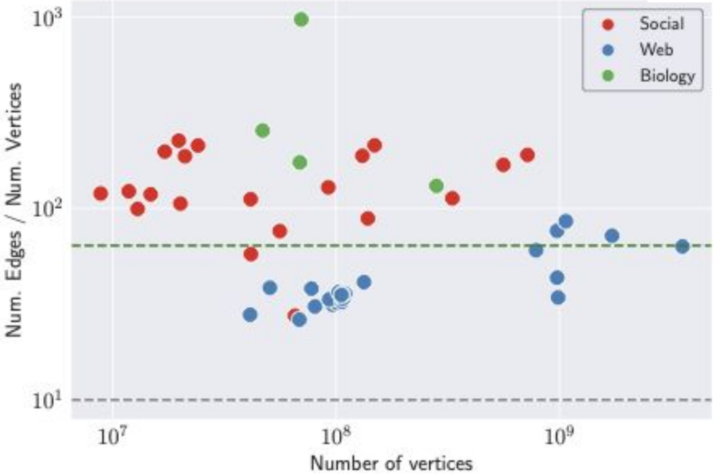
Easy to parallelize.



Rose : Read only semi-external model

Assume the symmetric memory can fit some part of the data but not some other part. The other part is stored in read-only memory.

I/O complexity: number of size B block reads from the read-only memory



Graphs, for example:

- Vertices fit
- Edges do not

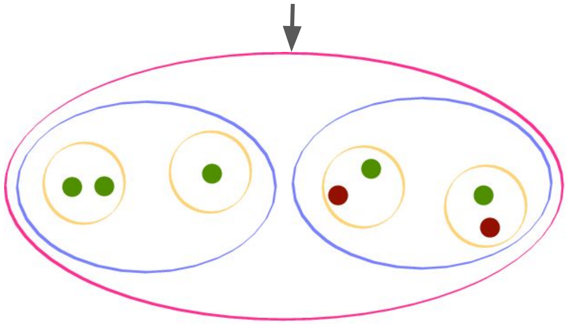
Recall that NVRAM is about 8x Denser than RAM

Some results:

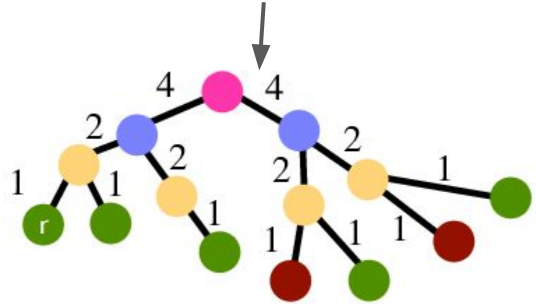
Problem	Work	Depth	I/O Complexity
Triangle Counting	$O(\alpha m)^*$	$O(\alpha \log n + \log^2 n)^\ddagger$	$O(\alpha(n + m/B))$
FRT Trees	$O(W_{SP} \log n)^*$	$O(D_{SP} \log n)^\ddagger$	$O(I_{SP} \log n)^*$
Strongly Connected Comp	$O(m \log n)^*$	$O(\mathrm{d}_G \log^3 n)^\ddagger$	$O((n + m/B) \log n)$
Breadth-First Search	$O(m)$	$O(\mathrm{d}_G \log n)^\dagger$	$O(n + m/B)$
Weighted BFS	$O(r_{src} + m)^*$	$O(\mathrm{r}_{src} \log n)^{\ddagger\dagger}$	$O(n + m/B)$
Bellman-Ford	$O(\mathrm{d}_G m)$	$O(\mathrm{d}_G \log n)^\dagger$	$O(\mathrm{d}_G(n + m/B))$
Single-Source Widest Path	$O(r_{src} + m)$	$O(\mathrm{r}_{src} \log n)^\dagger$	$O(n + m/B)$
Single-Source Betweenness	$O(m)$	$O(\mathrm{d}_G \log n)^\dagger$	$O(n + m/B)$
$O(k)$**-Spanner**	$O(m)^*$	$O(k \log n)^{\ddagger\dagger}$	$O(n + m/B)$
LDD	$O(m)^*$	$O(\log^2 n)^{\ddagger\dagger}$	$O(n + m/B)$
Connectivity	$O(m)^*$	$O(\log^3 n)^{\ddagger\dagger}$	$O(n + m/B)$
Spanning Forest	$O(m)^*$	$O(\log^3 n)^{\ddagger\dagger}$	$O(n + m/B)$
Graph Coloring	$O(m)^*$	$O(\log n + L \log \Delta)^{*\dagger}$	$O(n + m/B)$
Maximal Independent Set	$O(m)^*$	$O(\log^2 n)^{\ddagger\dagger}$	$O(n + m/B)$
Biconnectivity	$O(m)^*$	$O(\mathrm{d}_G \log n + \log^3 n)^{\ddagger\dagger}$	$O(n + m/B)$
PageRank Iteration	$O(m)$	$O(\log n)$	$O(m/B)$
k**-core**	$O(m)^*$	$O(\rho \log n)^\ddagger$	$O(n + m/B)$
Apx. Densest Subgraph	$O(m)$	$O(\log^2 n)$	$O(n + m/B)$

Example : FRT, probabilistic tree embeddings

Graph Metric

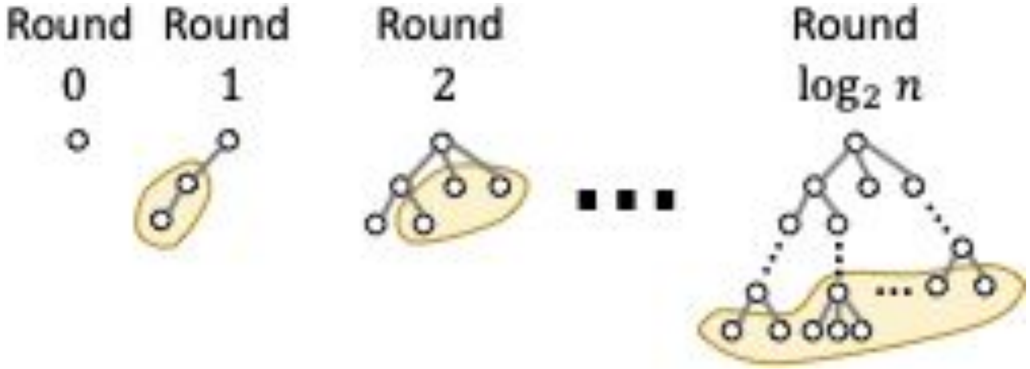


(a) Hierarchical decomposition.



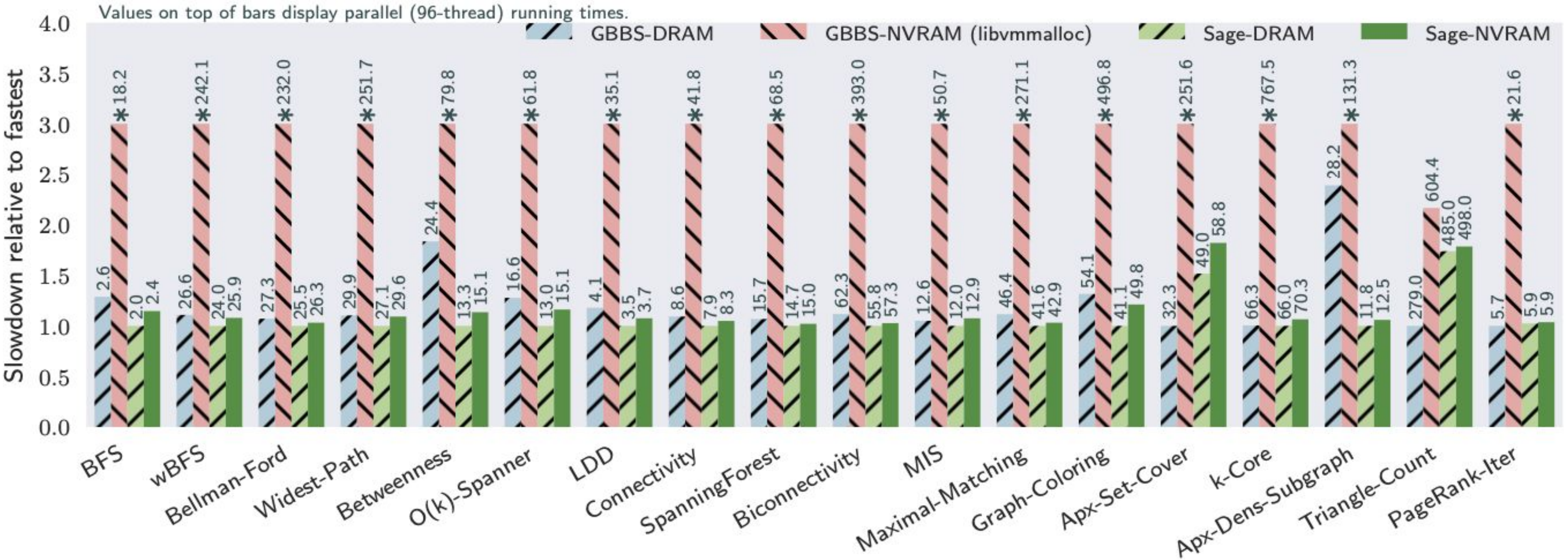
Previous parallel algorithms generate an O(n log n) size LE-list data structure as a substep (every point keeps a list of O(log n) other points and distances).

Idea, again, incrementally add points, only use part of LE list that is generated so far.   O(n) space.



Some timings on actual NVRAM

48 cores across two chips (VLDB 20) – Sage uses “Rose” algorithms



Conclusions

1. Designing write-efficient algorithms is fun.
2. There seem to be a handful of techniques that come up often.
3. It does make a difference both in theory and practice.
4. There are several lower-bounds…did not show them

Word of caution for APOCS community:

“If it is interesting from an algorithms point of view, and the good algorithm make a difference, it will be a while before it is of interest for practitioners”.