18-344 Recitation 9

11/14/2025

Outline

1. Logistics
2. Review
3. Lab 4
4. Lab 2 postmortem

Logistics

- Homework 8 released, due November 17 (Monday) (in 3 days)
  - Covers lecture 17
- Homework 9 released, due November 24 (Monday) (in 10 days)
  - Covers lecture 18-19

- Lab 3
  - Code freeze: November 21 (Friday) (in 1 week)
    - Submit (push a commit to your repo) on GitHub Classroom
  - Report: November 24 (Monday) (in 1 week + 3 days)
    - Submit on Gradescope
- Lab 4 will be released on November 17
  - Code freeze: December 11
    - Submit (push a commit to your repo) on GitHub Classroom
  - Report: December 14
    - Submit on Gradescope

Review

Arithmetic Intensity*

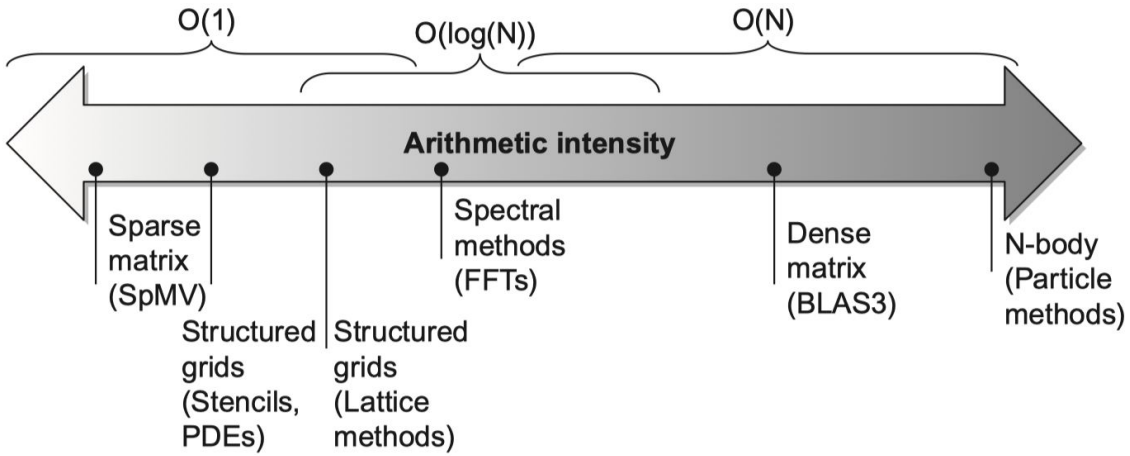
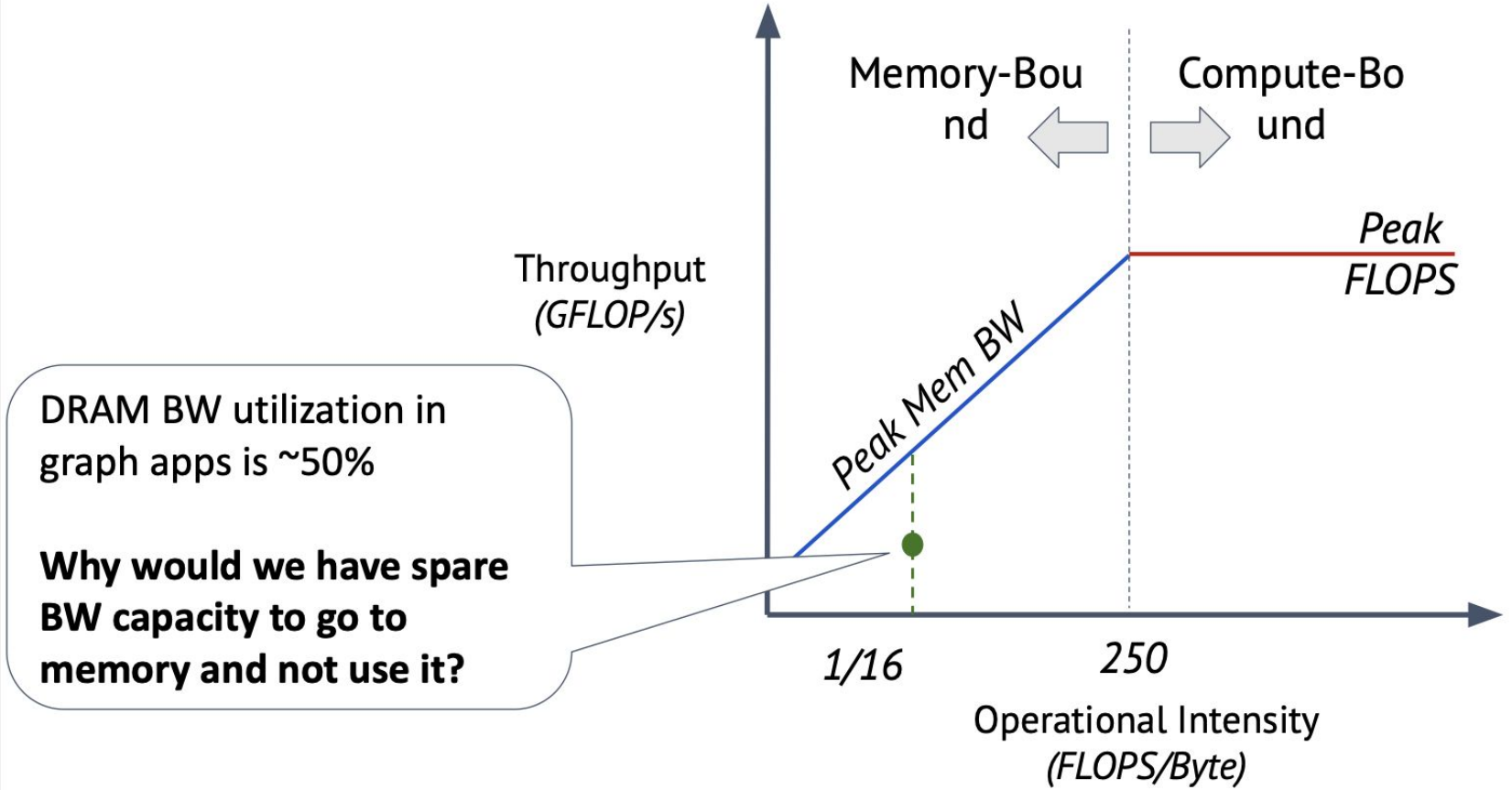


Figure 4.10 Arithmetic intensity, specified as the number of floating-point operations to run the program divided by the number of bytes accessed in main memory (Williams et al., 2009). Some kernels have an arithmetic intensity that scales with problem size, such as a dense matrix, but there are many kernels with arithmetic intensities independent of problem size.

Computer Architecture: A Quantitative Approach 6th Edition, Hennesy and Patterson

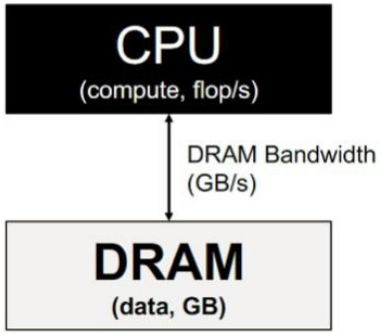
Graph Applications are Memory-Bound



DRAM BW utilization in graph apps is ~50%

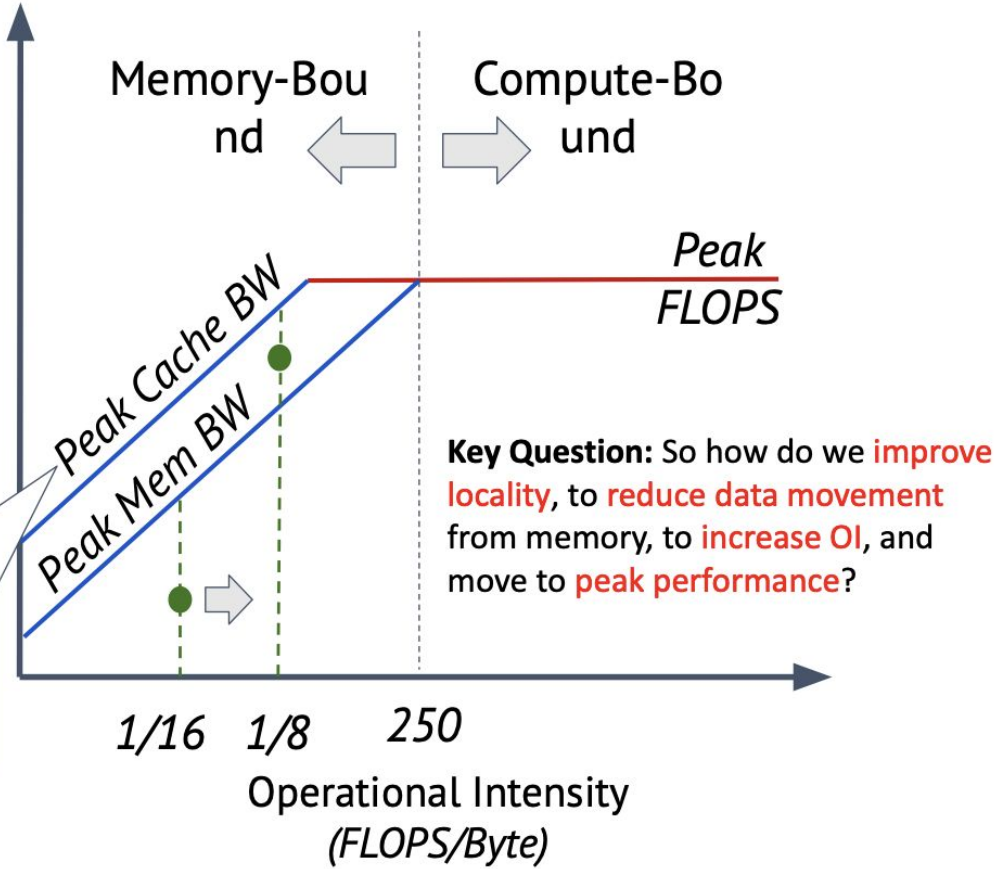
Why would we have spare BW capacity to go to memory and not use it?

Improving Operational Intensity (OI) by Improving Locality



Locality wins: If we can operate out of cache, higher ceiling & more leftward ridge point.

Why is cache BW > DRAM BW?
Smaller SRAM caches much faster.

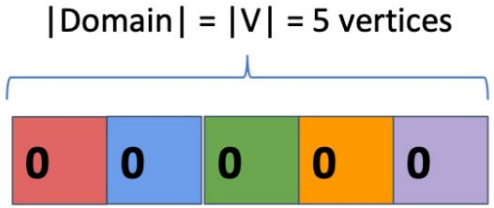


Key Question: So how do we improve locality, to reduce data movement from memory, to increase OI, and move to peak performance?

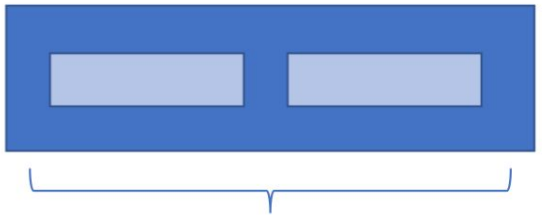
Propagation Blocking: Reorganize Input to Make Memory Being Randomly Written Fit in Cache

0	1
2	0
1	0
0	2
2	3
0	4
0	3

COO
(EdgeList)



Bad for the cache: the size of the *domain* of vertex data array entries is |V|, but the cache holds only |C| << |V| entries

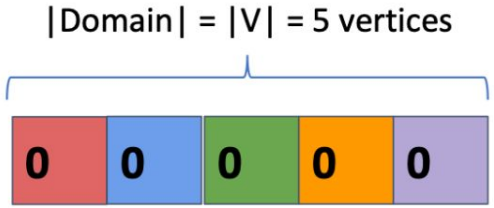


Recall: irregular accesses into vertex data array based on e.dst *which are essentially random*

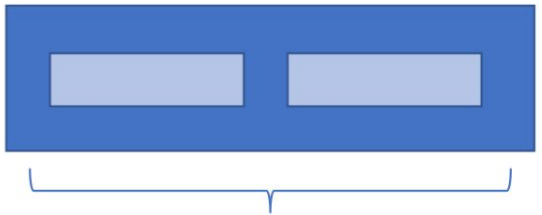
Propagation Blocking: Reorganize Input to Make Memory Being Randomly Written Fit in Cache

0	1
2	0
1	0
0	2
2	3
0	4
0	3

COO
(EdgeList)



Bad for the cache: the size of the *domain* of vertex data array entries is |V|, but the cache holds only |C| << |V| entries

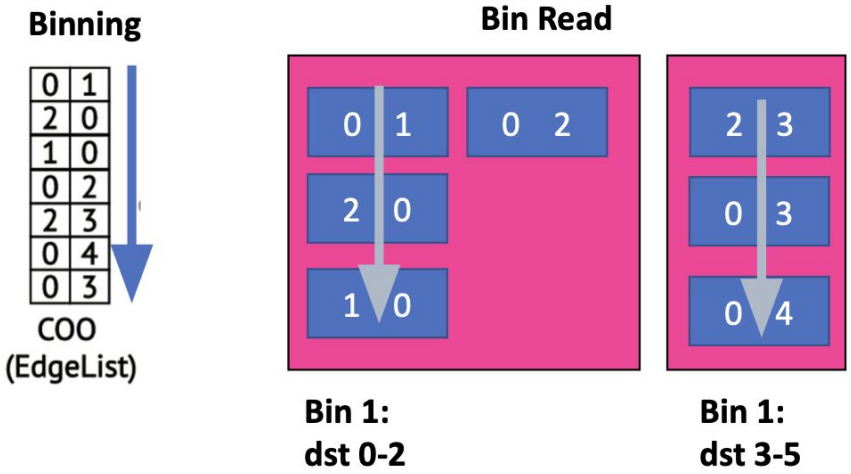


Recall: irregular accesses into vertex data array based on e.dst *which are essentially random*

Key idea in propagation blocking: Limit the domain of updates to a *sub-space* of vertices, **V***, so that |V*| <= |C| and do multiple sub-spaces of V*s, so that all V*s together = V

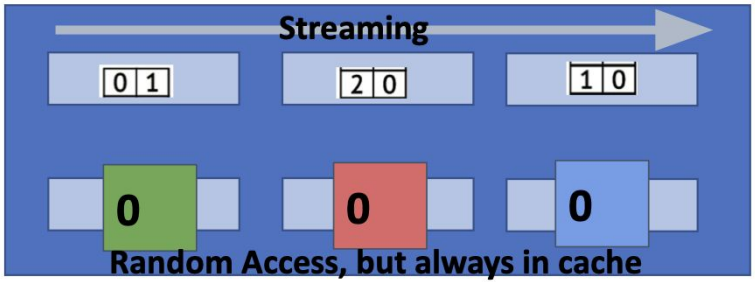
Propagation Blocking: Performance Analysis

Traverse the edge list twice instead of once



What about the performance of reading the edge list during binning?

Usually save a little space in cache for *streaming edge list* data. Easy to cache.



dstData

Remember: dstData[e.dst] ++
and e.dst is random, from edge list

Propagation Blocking: Edge Count

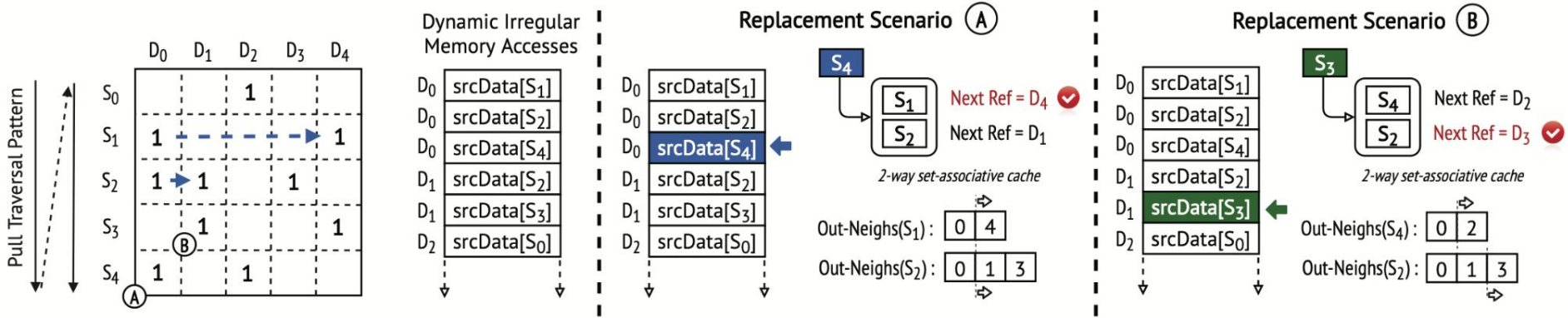
```
1  Bins B[];
2
3  for edge in E {
4      add_to_bin( find_bin(edge) )
5  }
6
7  for bin in B {
8      for e in bin {
9          dstData[e.dst]++
10     }
11 }
```

Costly pre-processing:
putting edges in distinct bins
(a.k.a. ***binning***)

(Hopefully) each bin is small enough to fit into the cache, so later repeated accesses to the bins have ***improved locality***.

T-OPT and P-OPT

- Transpose-based Cache Replacement (T-OPT)
  - Uses a graph's adjacency matrix to make near-optimal replacement decisions.
  - Guiding replacement based on graph structure in T-OPT allows emulating Belady's MIN policy *without* oracular knowledge of all future accesses.



V. Balaji, N. Crago, A. Jaleel and B. Lucia, "P-OPT: Practical Optimal Cache Replacement for Graph Analytics," *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA).*

T-OPT and P-OPT

- P-OPT: Providing architectural extensions for efficient T-OPT execution.
  - Using quantization to compress the transpose.

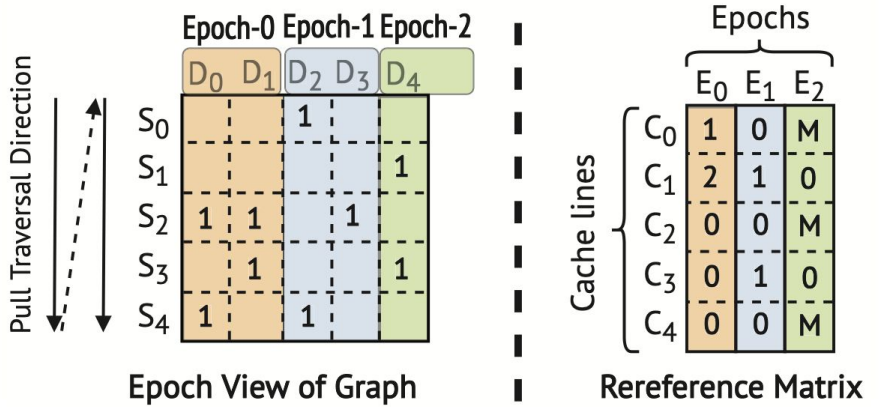


Fig. 5: **Reducing T-OPT overheads using the Rereference Matrix:** *Quantizing next references into cachelines and a small number of epochs reduces the cost of accessing next references.*

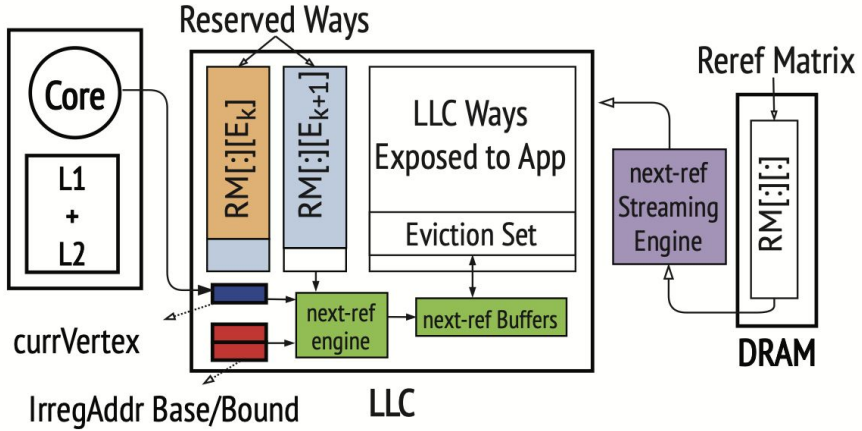


Fig. 9: **Architecture extensions required for P-OPT:** *Components added to a baseline architecture are shown in color.*

V. Balaji, N. Crago, A. Jaleel and B. Lucia, "P-OPT: Practical Optimal Cache Replacement for Graph Analytics," *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA).*

T-OPT and P-OPT

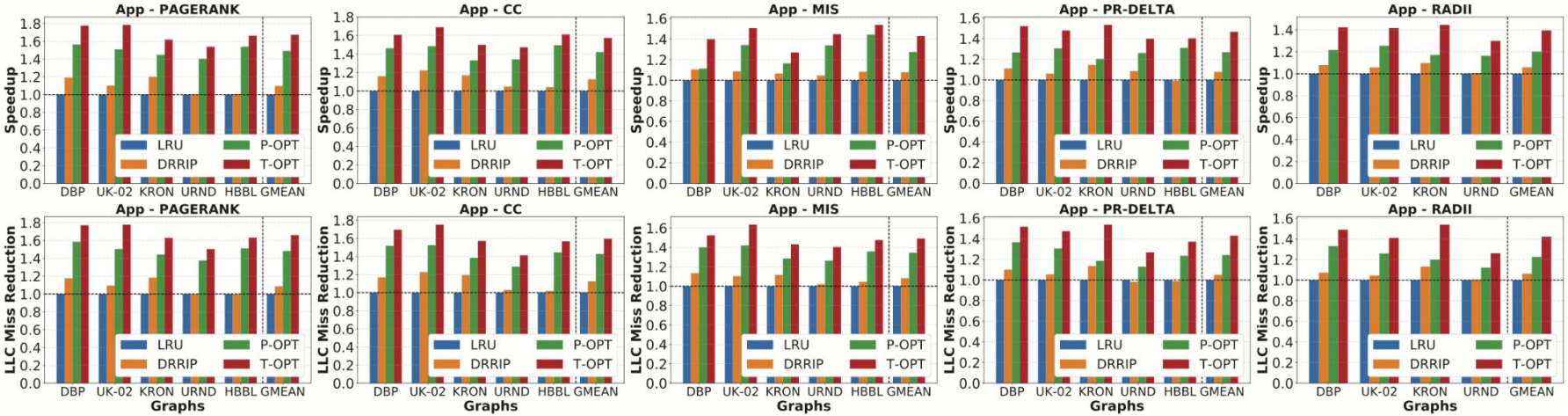
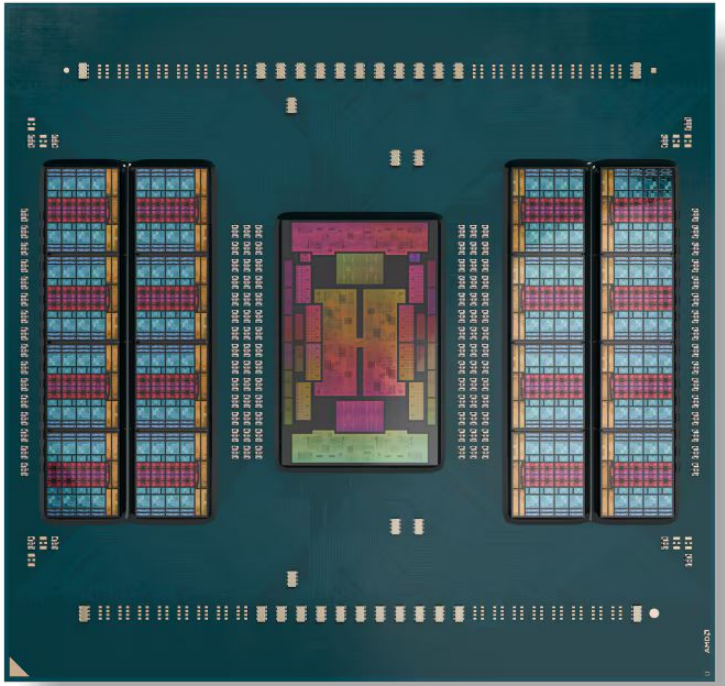
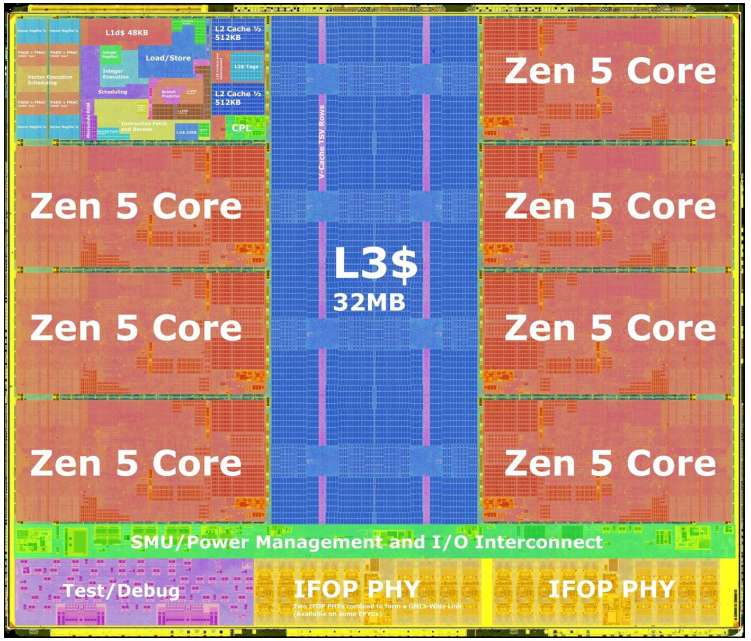


Fig. 10: **Speedups and LLC miss reductions with P-OPT and T-OPT:** *The T-OPT results represent an upper bound on performance/locality because T-OPT makes optimal replacement decisions using precise re-reference information without incurring any cost for accessing metadata. P-OPT is able to achieve performance close to T-OPT by quantizing the re-reference information and reserving a small portion of the LLC to store the (quantized) replacement metadata.*

V. Balaji, N. Crago, A. Jaleel and B. Lucia, "P-OPT: Practical Optimal Cache Replacement for Graph Analytics," *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA).*

Shared-memory multi-processors





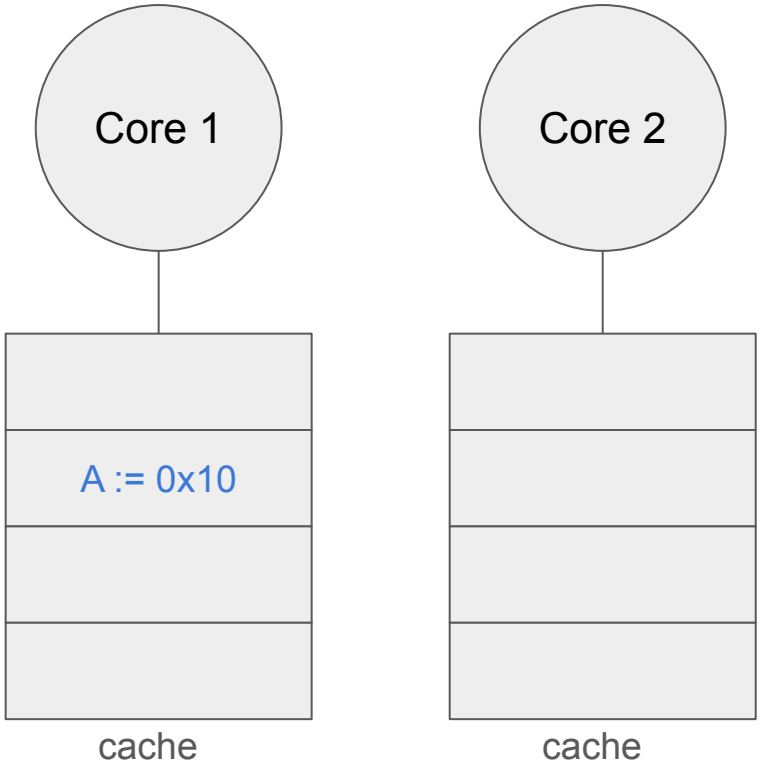
5th Generation AMD EPYC™ Processors, “Zen 5”. https://www.amd.com/en/products/processors/server/epyc/9005-series.html

AMD Granite Ridge "Zen 5" Processor Annotated. https://www.techpowerup.com/327388/amd-granite-ridge-zen-5-processor-annotated

Cache **In**coherence

T = 1:

Core 1:
`load r1, mem[A]`

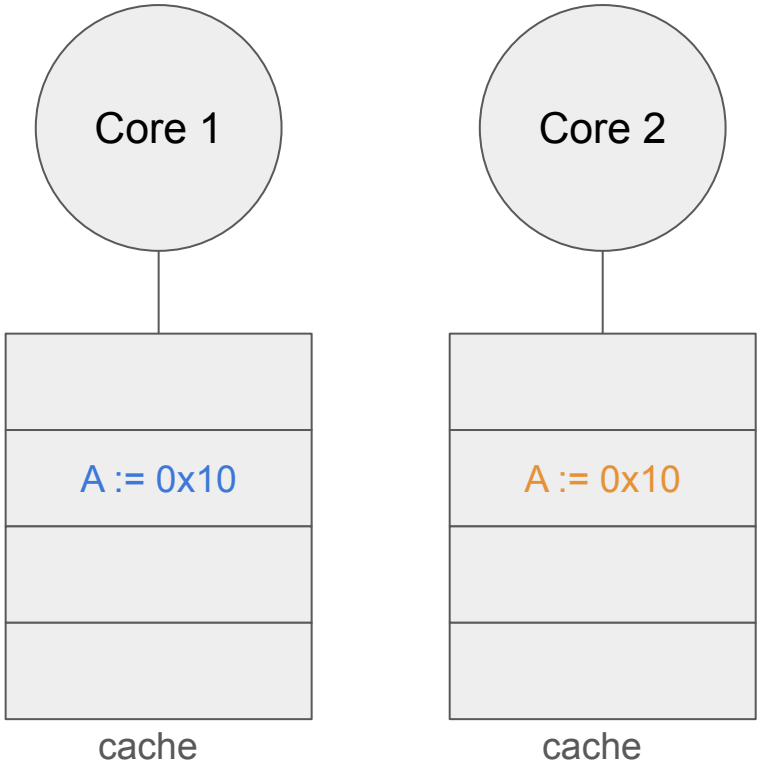


Cache **In**coherence

T = 2:

Core 2:

```
load r1, mem[A]
```

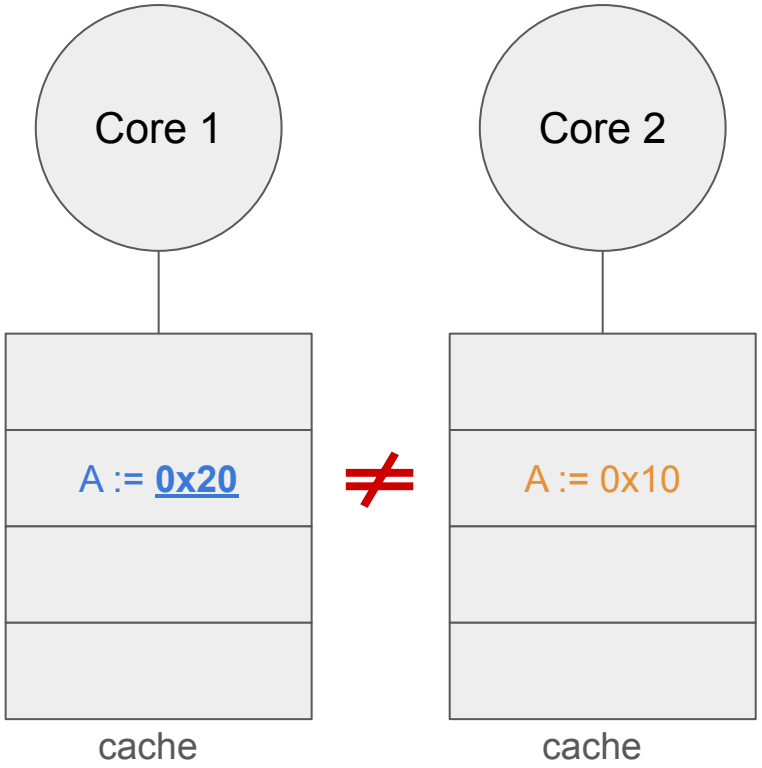


Cache **In**coherence

T = 3:

Core 1:

```
add r1, r1, r1
store r1, mem[A]
```



Cache Coherence Protocols

- Snooping-based
  - Coherency controllers in caches monitor or ***snoop*** the bus transactions.
  - Each coherency controller maintains the states of its own cache lines, and act (change its own state, tell other controllers to change state, etc.).
  - Need to broadcast coherence messages to determine the state of a line in the other caches.
- Directory-based
  - Avoid broadcasting by storing information about the status of the line in directories.
  - Directory entry for a cache line contains information about the state of the cache line in all caches.
    - Caches look up information from the directory as necessary
  - Cache coherence is maintained by point-to-point messages between the caches on a “need to know” basis.

- Revisit lecture 17 for more examples.

On Memory Consistency

I guess that I had no idea how a program should behave.
I may have been quite the only one;
indeed programmers seem to expect certain behaviours
and find others surprising when they run a program.

Excerpt from "A shared memory poetics."
Jade Alglave, 2010.

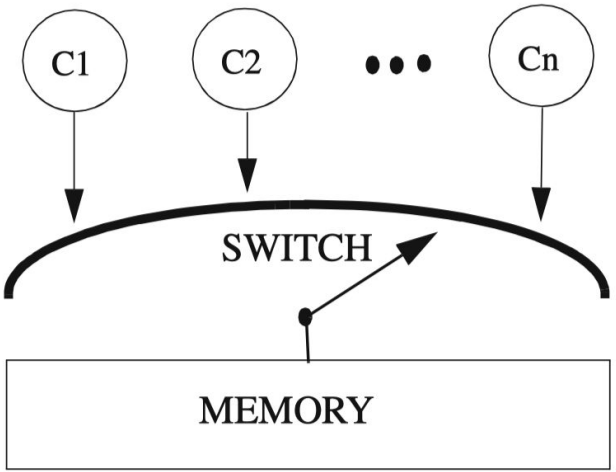
Coherence vs. Consistency

- Coherence defines the set of legal orders of accesses to a ***single*** memory location.

- Consistency defines the set of legal orders of accesses to ***multiple*** memory locations.

Memory Consistency Models

- Sequential Consistency (SC)
  - Invariant 1: Instructions are executed in program order.
  - Invariant. 2: All processors agree on a total order of executed instructions.
  - Think: memory switch model.



Each core Ci seeks to do its next memory access in its program order <p.

The switch selects one core, allows it to complete one memory access, and repeats; this defines memory order <m.

Sorin, D., Hill, M. and Wood, D., 2011. *A primer on memory consistency and cache coherence*. Morgan & Claypool Publishers.

Relaxed Memory Consistency Models

- What if we want *some* reordering of memory instructions?
  - *Why* would we want this?
    - Memory optimizations.
    - Compiler optimizations.
    - Hardware constraints.
- Turns out, it's *okay* to allow some reordering of memory instructions.
  - Use the happens-before graph to check if mem. instructions have been reordered.
  - Memory models that permit reordering are **relaxed**.
  - Modern machines do this all the time!
    - **x86-TSO** (Total Store Order): loads may complete before older stores to different locations complete. "The write buffer memory model."
    - arm's weakly ordered memory model.
  - **Synchronization** is needed to prevent reordering and eliminate data races.

Lab 4

Overview

- You'll study and optimize a kernel commonly used in sparse problems: **EL2CSR**.
  - Study: an **inefficient** implementation of `EL2CSR` that we provide.
  - Optimize: implement **propagation blocking** to improve locality.

- You are asked to study and demonstrate performance improvement quantitatively.
  - Demonstrate that propagation blocking improves the performance of `EL2CSR` compared to directly processing edge list data.
  - Choose metrics, baselines, characteristics, and hardware configurations carefully.
  - Evaluate the performance of your implementation using your cache simulator from lab2 with single-level cache.
  - Evaluate your implementation with **both** synthesized input and real-world input.

- This time, you'll be implementing a **binary tool**.

Tips

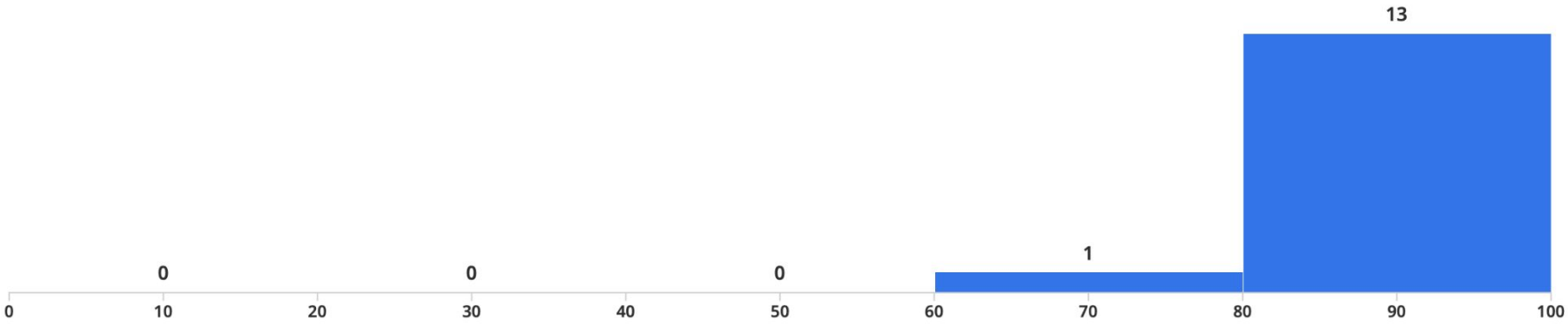
- Read the handout carefully.
  All of our suggestions for this lab are already in there.

- Think about these questions:
  When implementing propagation blocking, do you bin by source or destination? What's the difference? Is one better than the other? How do you evaluate it (analytically or by experiment)?

- Evaluating the effectiveness of propagation blocking using ***only synthesized*** inputs sometimes doesn't show much difference. Consider using large, real-world graphs: https://snap.stanford.edu/ (personal recommendation: `web-Google`).

Lab 2
postmortem

Nice work, everyone!

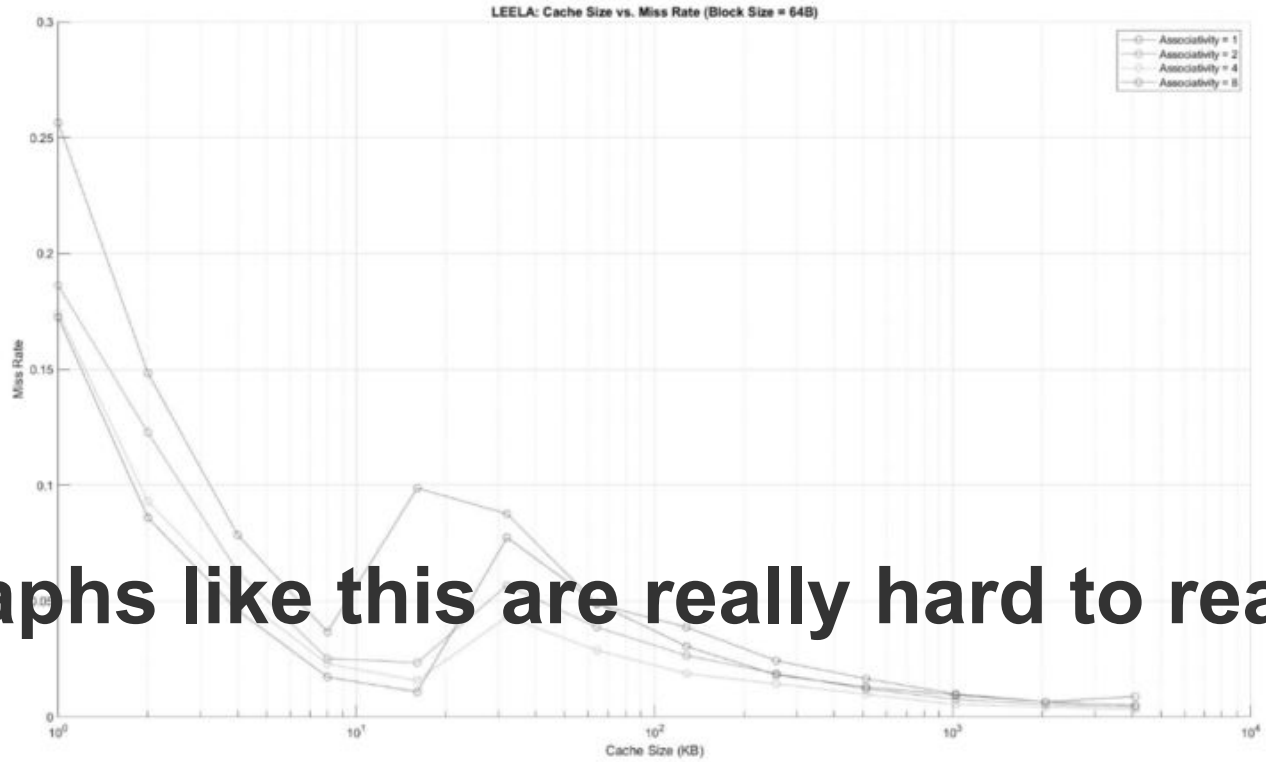


Minimum	Median	Maximum	Mean	Std Dev
75.0	89.0	100.0	88.39	6.95

Feedback from us

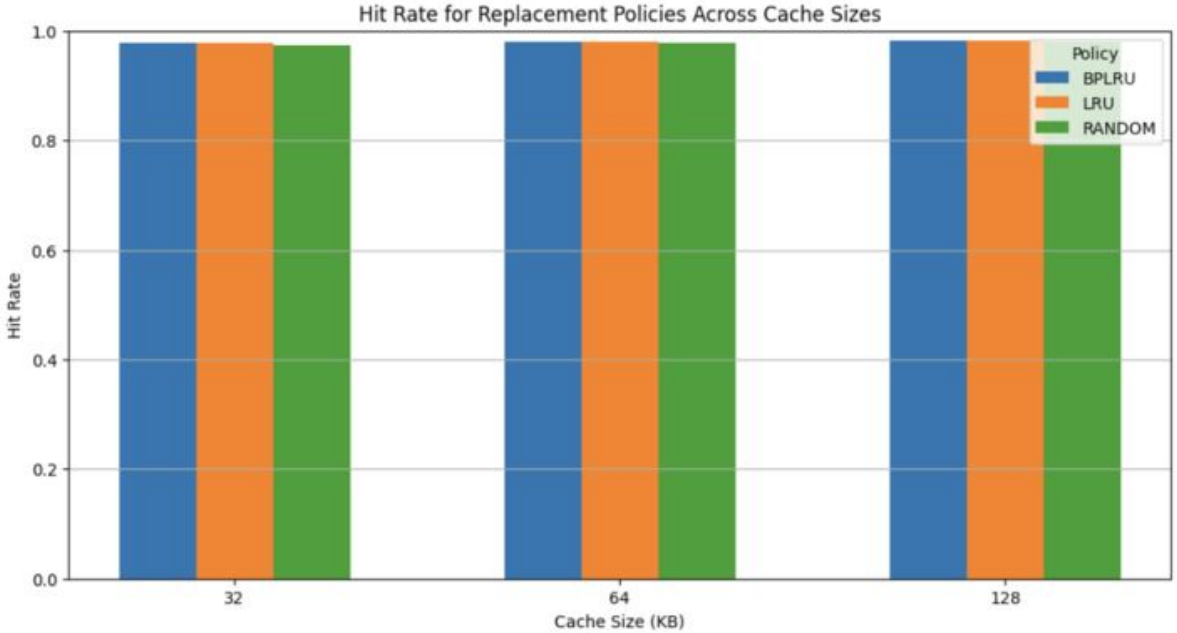
- **Avoid overly large tables of numbers.**
  Tables are not an effective way to show large amounts of numerical data.
- **Use plots instead of tables when possible**, and keep plots simple and readable.
- **Label axes clearly**, including **units**.
- **Write clear, informative figure captions** that explain what the reader should take away.
- **Declutter your graphs.**
  Avoid unnecessary labels or text (e.g., labeling every point on a Pareto plot).
- **Focus on showing trends.**
  Include plots only if they reveal something meaningful about your results.
- **“Less is more”** — both in writing and in the number of figures you include.
- **Follow the write-up instructions more closely.**
  Several required elements were missed. For example:
  You should include a bar plot for a set of Pareto optimal cache configurations that shows the AMAT for each of the SPEC2017 benchmarks that you run.

More feedback from us



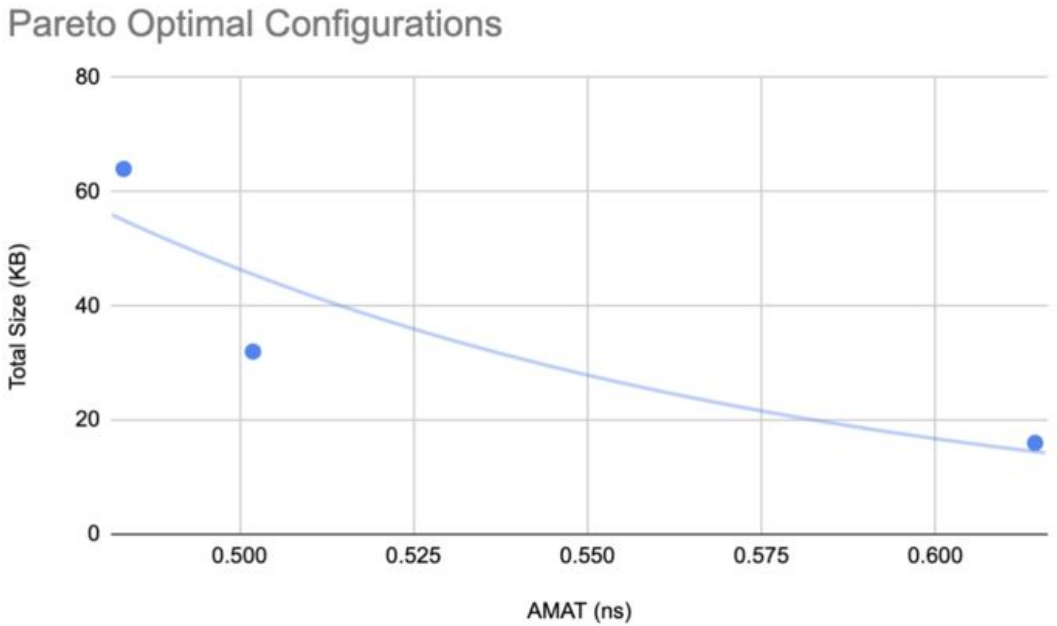
Graphs like this are really hard to read.

More feedback from us



Adding labels to data is also a good idea, especially when things are too close to visually tell the difference.

More feedback from us



Trendlines / curve fittings are NOT Pareto frontiers.