Foundational Lecture: Workload Properties, Roofline Model, and Measurements

Scott Beamer
sbeamer@ucsc.edu

1/9/19

University of California, Santa Cruz

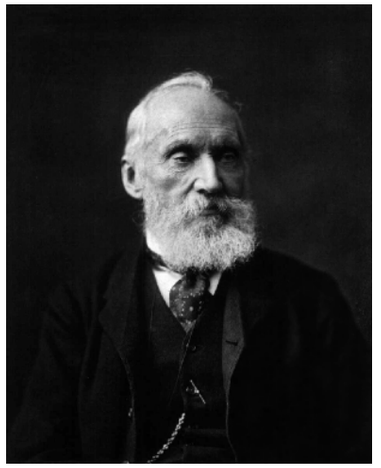
CMPE 293: Programmable Hardware Accelerators

https://cmpe293-winter19-01.courses.soe.ucsc.edu

Motivation for Measuring Workload

- Lord Kelvin 1883

> I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science, whatever the matter may be. https://en.wikiquote.org/wiki/William_Thomson



- If going to specialize hardware for workload, better understand characteristics of workload *well*
  - Need workload properties to exploit for efficiency
  - Understand workload's need for flexibility

Topics for Today

- *Goal*: Provide background to appreciate many of the papers for this course
- Workload characterization considerations
- Types of parallelism
- Memory hierarchy challenges
- Roofline Model
- Workload characterization methods

Challenges Summarizing Workload

- There are many dimensions to analyze a workload from, so typically impractical to measure (or present) all of them
- Need to summarize most relevant results
  - What matters most? (significance)
  - How will results be used? (context)
  - “Workload has high X” - Relative to what?

Summarizing Sanely

- In addition to qualitative statements, provide measurements too
- Understand consumers of characterization
  - What do they want to learn from it?
- Iterate on characterization with consumer
  - Are claims supported by measurements?
  - Alternative explanations?
  - Other things to measure?

General Things to Look For

- *Common traits* across target workload
  - especially if different from general purpose workloads
- *Underutilized resources* can be scaled down, or at least hint bottlenecks nearby
- *Bottlenecks* are a saturated resource that limits overall performance

Ex: Graph Characterization Workload

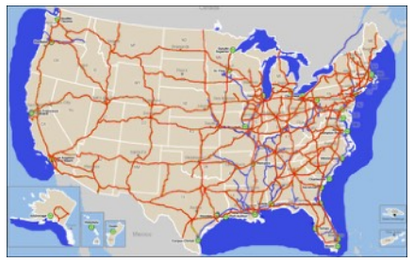
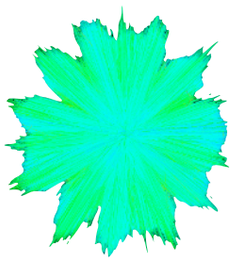
Code

Galois | GAP | Ligra

Kernel

Breadth-First Search (**BFS**)	Single-Source Shortest Paths (**SSSP**)	Connected Components (**CC**)	Betweenness Centrality (**BC**)	PageRank (**PR**)

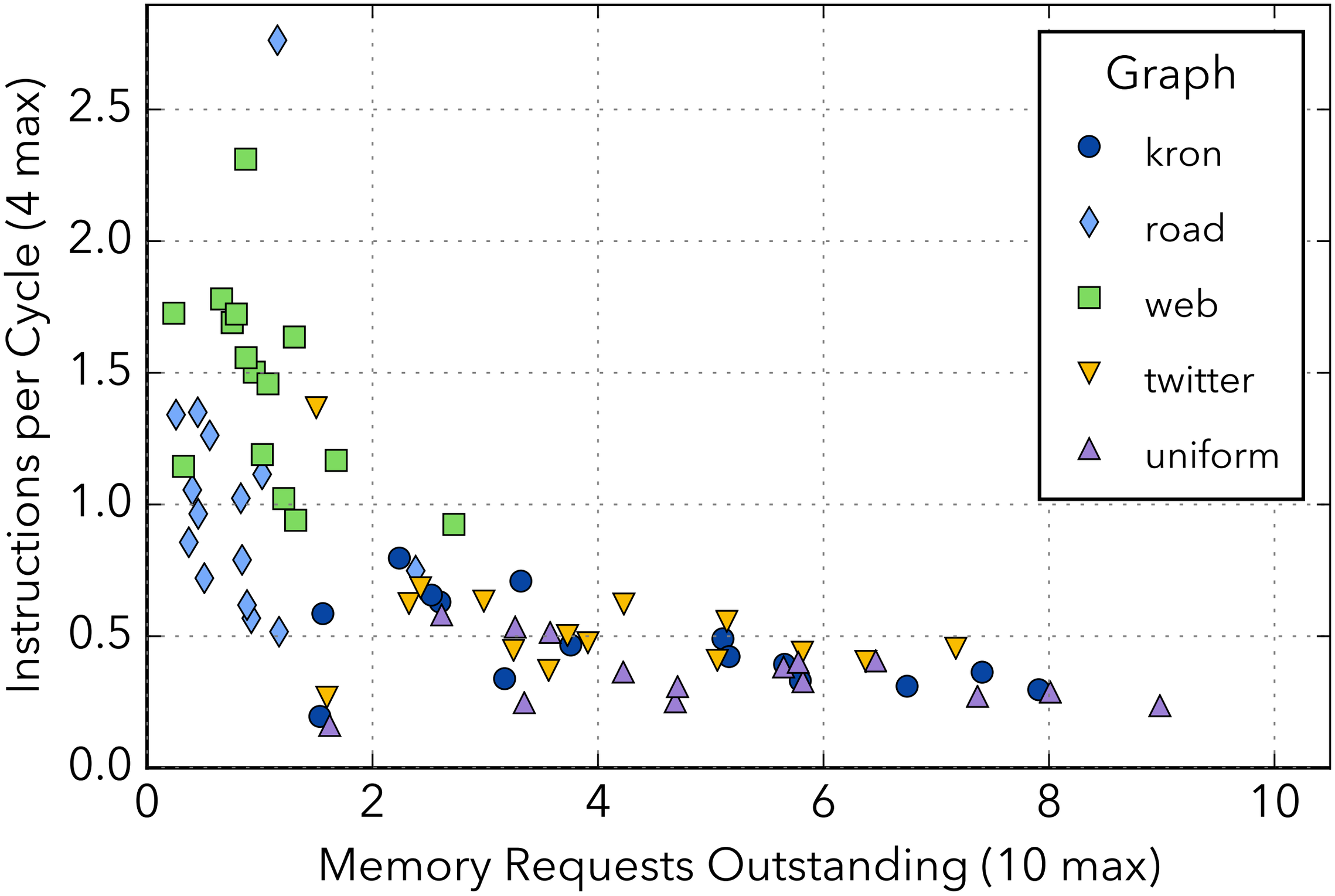
Graph

				
Roads of USA	**Kronnecker** Synthetic	**Web** Crawl	**Uniform** Random	**Twitter**

Ex: Evaluation Platform

- Target Platform
  - Dual-socket *Intel Ivy Bridge* (E5-2667 v2)
  - *Socket*: 8 cores with 25MB L3 cache
  - *Core*: 3.3 GHz 2 threads
  - *DRAM*: 256 GB DDR3-1600
- Hardware performance counters
  - Intel Performance Counter Monitor (PCM)
  - PAPI

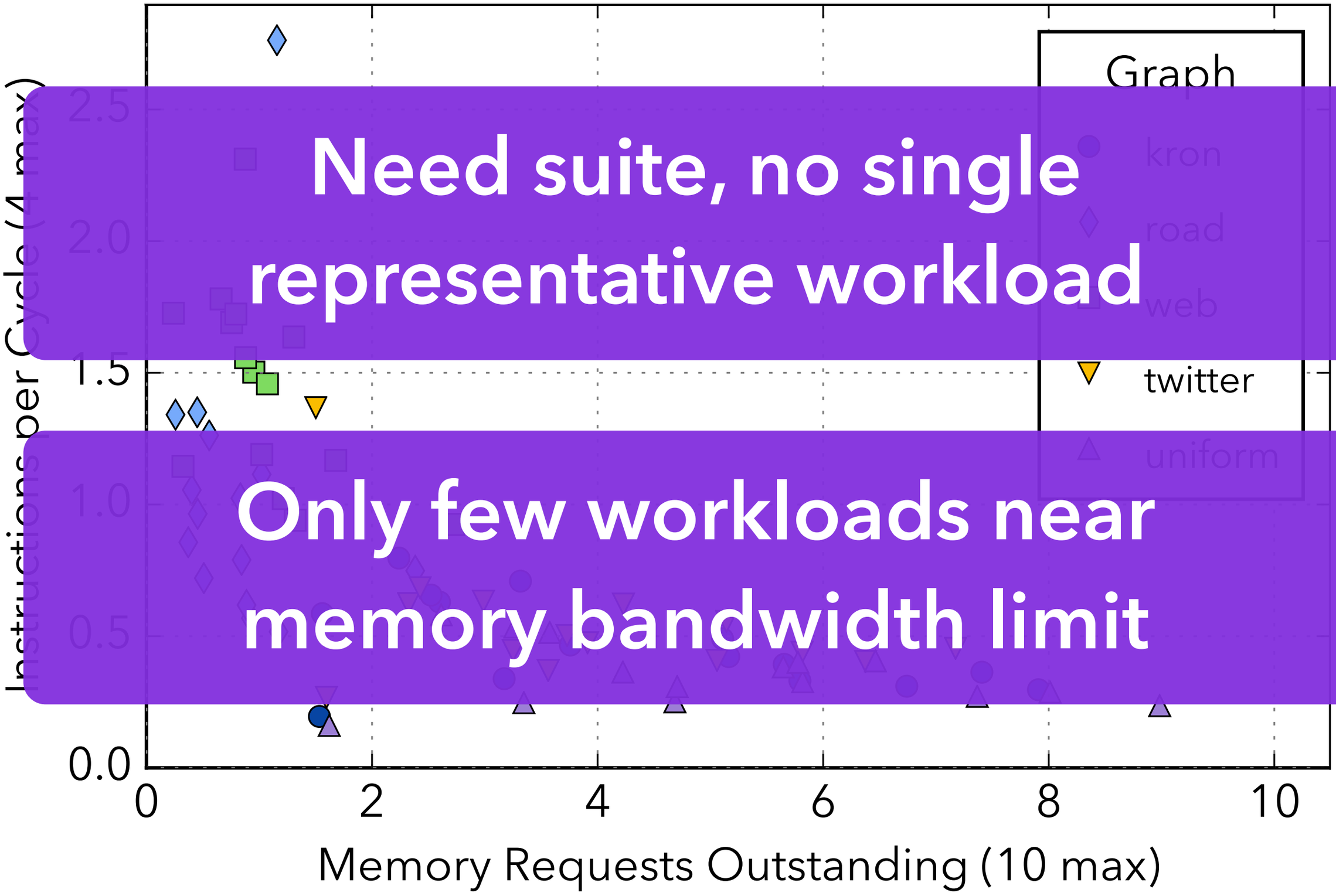
Ex: Graph Workloads Are Diverse



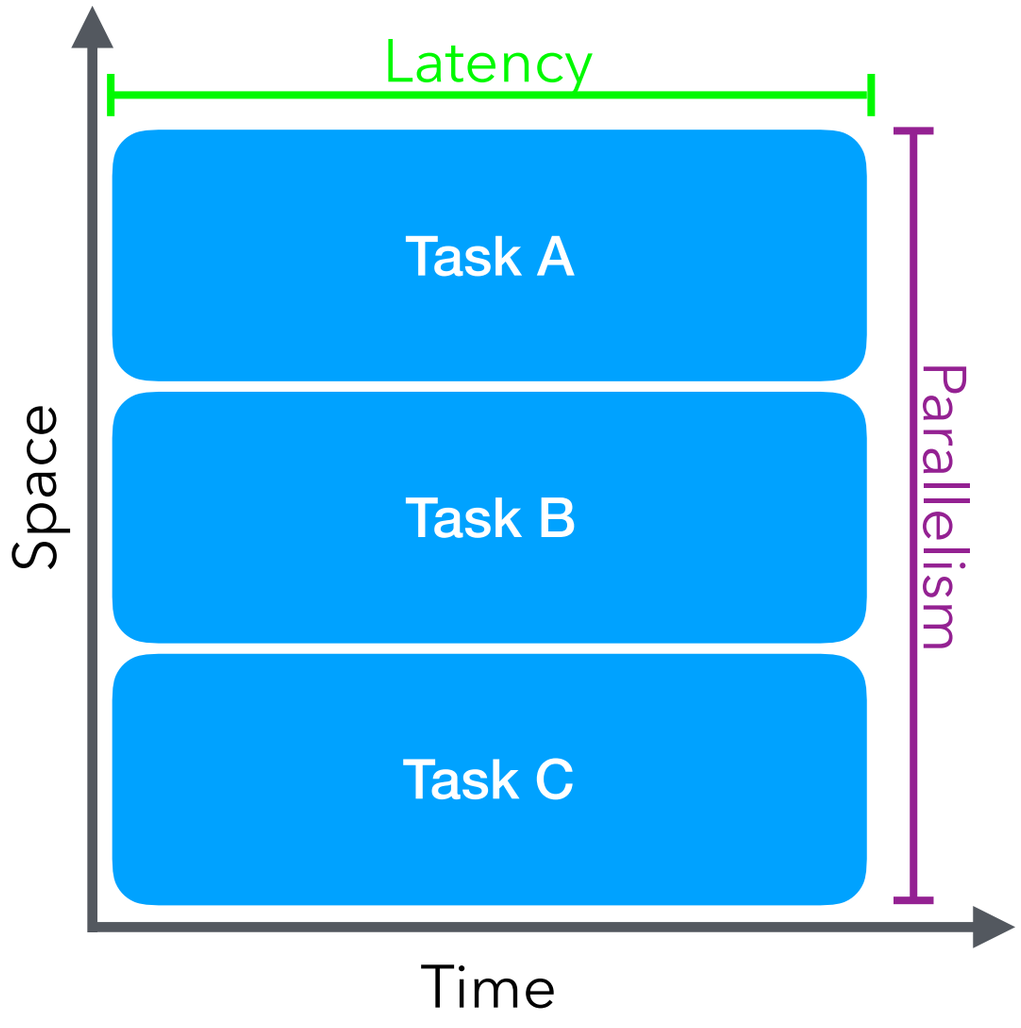
Ex: Graph Workloads Are Diverse

Need suite, no single representative workload

Only few workloads near memory bandwidth limit



Little's Law



$$\text{Throughput} = \frac{\text{Parallelism}}{\text{Latency}}$$

Little's Law

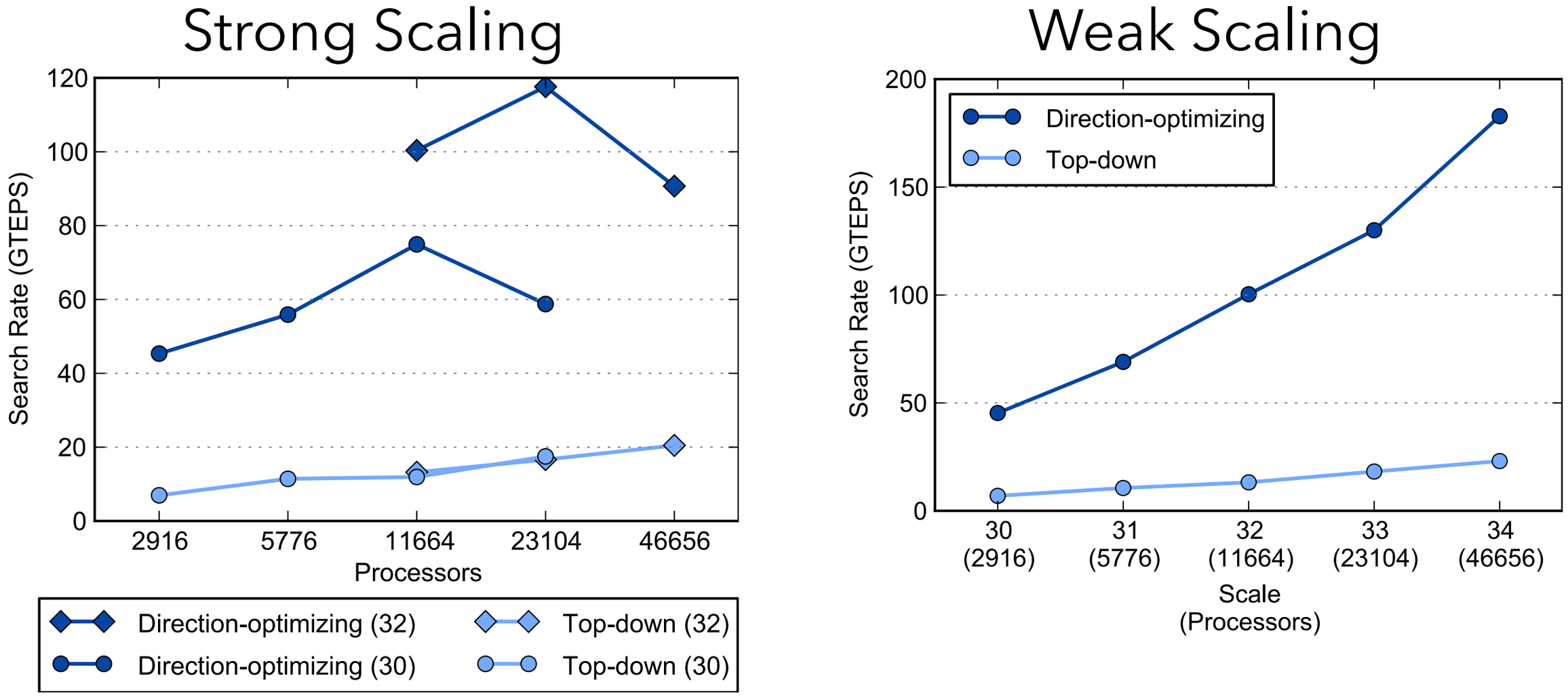
Parallelism = Latency x Throughput

- Typically, latency is fixed and want to increase throughput, so must increase parallelism
- If variance in inputs, Little's Law is for average

Measuring Scalability

- *Scaling* (increasing parallel resources), can be hindered by many factors
- *Strong scaling* - constant problem size, so decreasing problem/resource
- *Weak scaling* - constant problem/resource, so increasing problem size
- *Caveat*: When looking at scaling curves, ask what is relative to? Absolute performance matters more than linear speedup curves.

Example Scalability Measurements

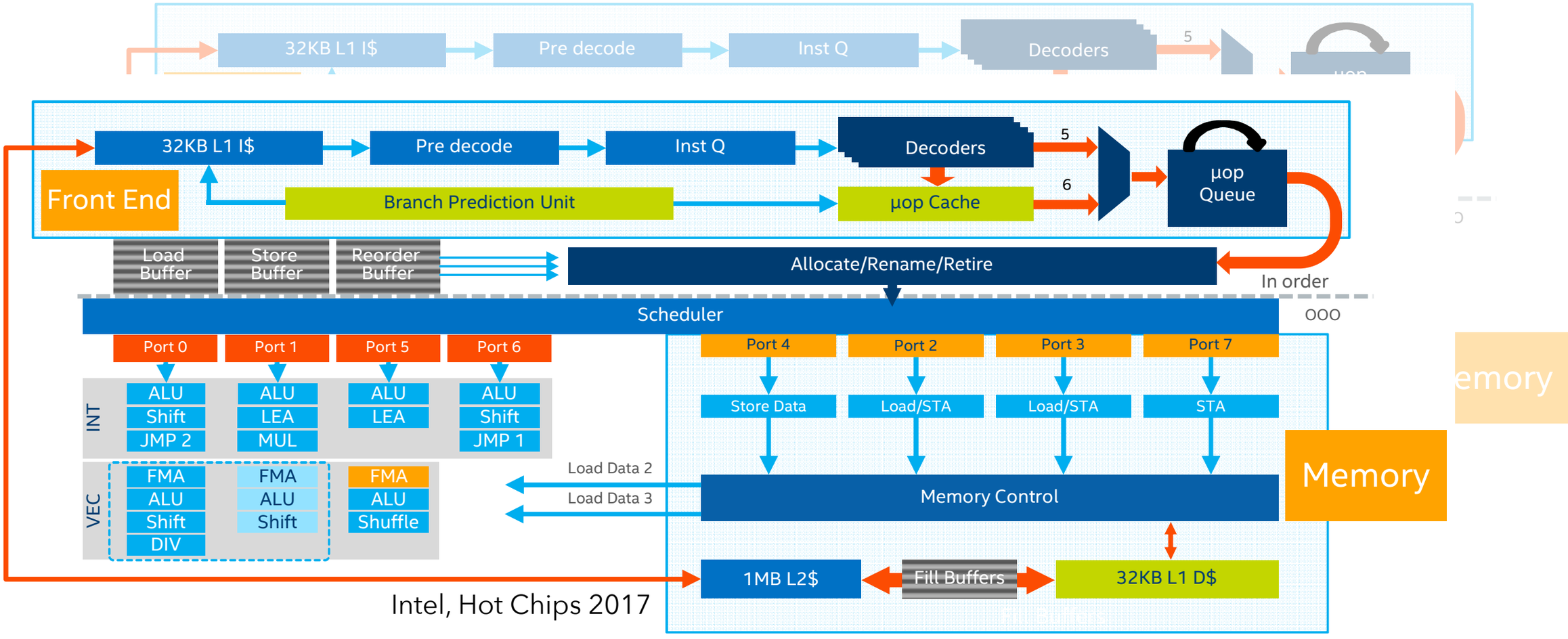


- Direction-optimizing implementation slows down when problem not big enough
- Despite non-linear scaling, still faster

Types of Parallelism in Processors

- Instruction-level Parallelism (ILP)
  - instructions are independent, but often from same "thread"
  - example uses: pipelining, superscalar
- Thread-level Parallelism (TLP)
  - threads have independent control flow
  - example uses: multicore, multithreading
- Data-level Parallelism (DLP)
  - data elements are independent, but similar control flow
  - example uses: SIMD extensions, vector
- Memory-level Parallelism (MLP)
  - memory accesses are independent

Example Parallelism Uses in Skylake



Intel, Hot Chips 2017

- ILP - can execute multiple instructions simultaneously
- DLP - vector instructions operate on multiple data elements
- TLP - multiple cores execute simultaneously
- MLP - out-of-order allows multiple memory ops at once

Administrivia

- Reading preferences due 1/10 @ 12PM
- 1st reading summary due 1/11 @ 9AM
- Sign up for Piazza
- Start looking for project partners and workloads
- Course website
  - https://cmpe293-winter19-01.courses.soe.ucsc.edu/

Improving Communication Efficiency

Reduce

(move less data)

- Improve data reuse
- Improve data layout

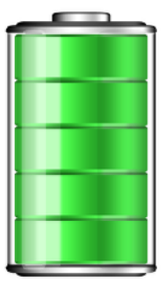
Accelerate

(move data faster)

- Utilize more bandwidth
- Build more bandwidth

Improve Communication Efficiency

Improve Performance



Save Energy

Reduce Cost

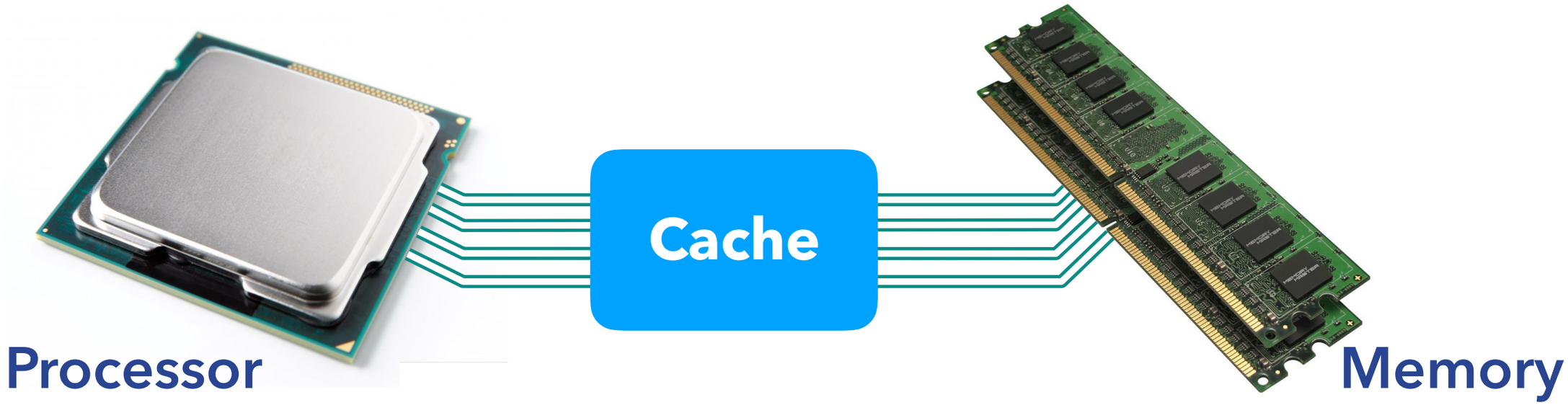
Communication-centric Perspective

	Reduce (move less data)	**Accelerate** (move data faster)
Algorithms	• What data is needed? • How to arrange data?	• When is data needed? • What is parallelizable?

Efficient Communication

Architecture	• Memory technology? • Memory hierarchy?	• Link technology? • Topology design?

Memory Hierarchy Recap



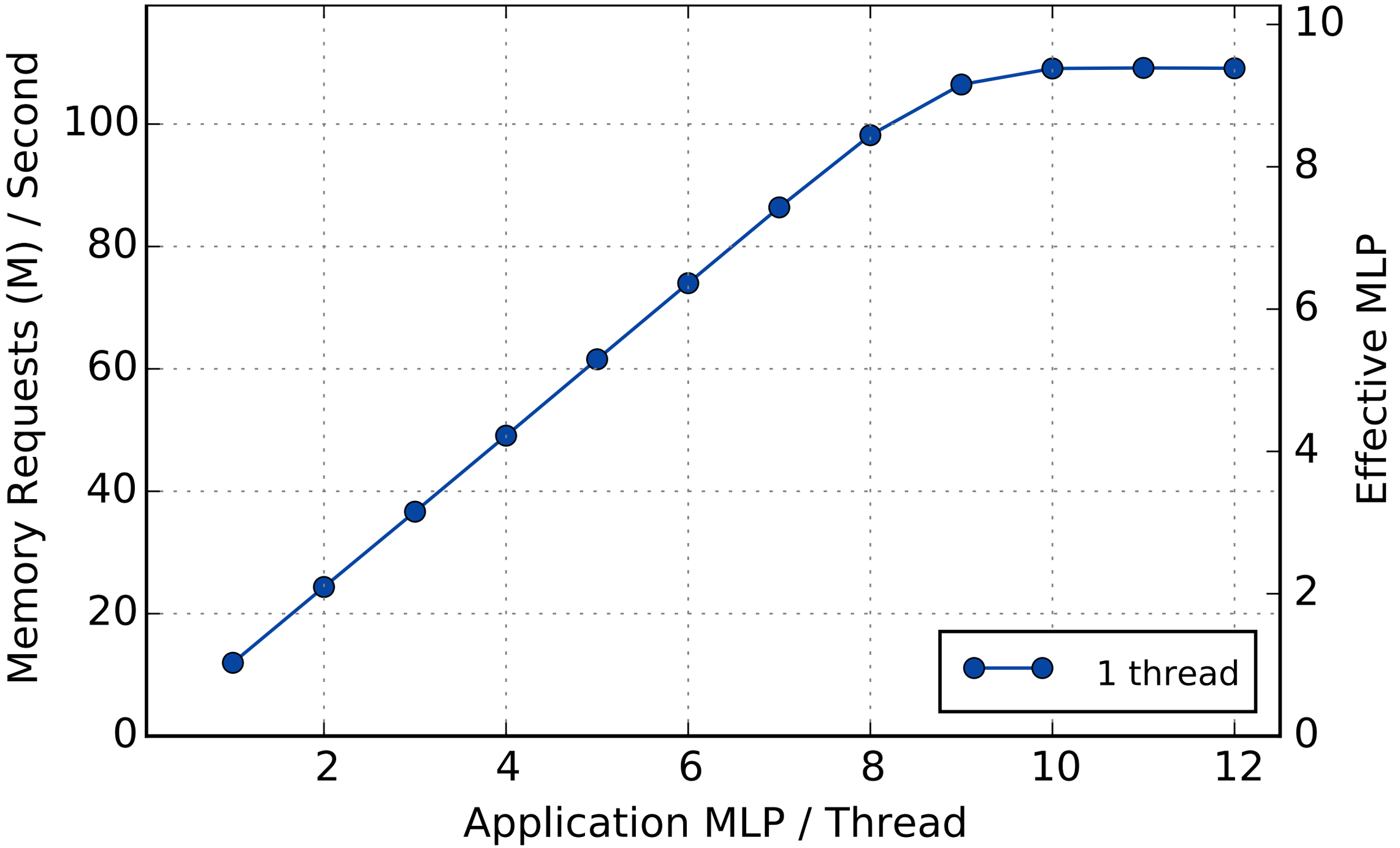
- Memory is “far” (latency & energy) from CPU
- *Locality* - property that some data is more likely to be accessed than other data
- Use cache to hold subset of data “closer” that is likely to be accessed (exploits locality)

Types of Locality

- *Temporal Locality* - likely to *reuse* data that has been accessed
- *Spatial Locality* - likely to use data *near* data that has been accessed

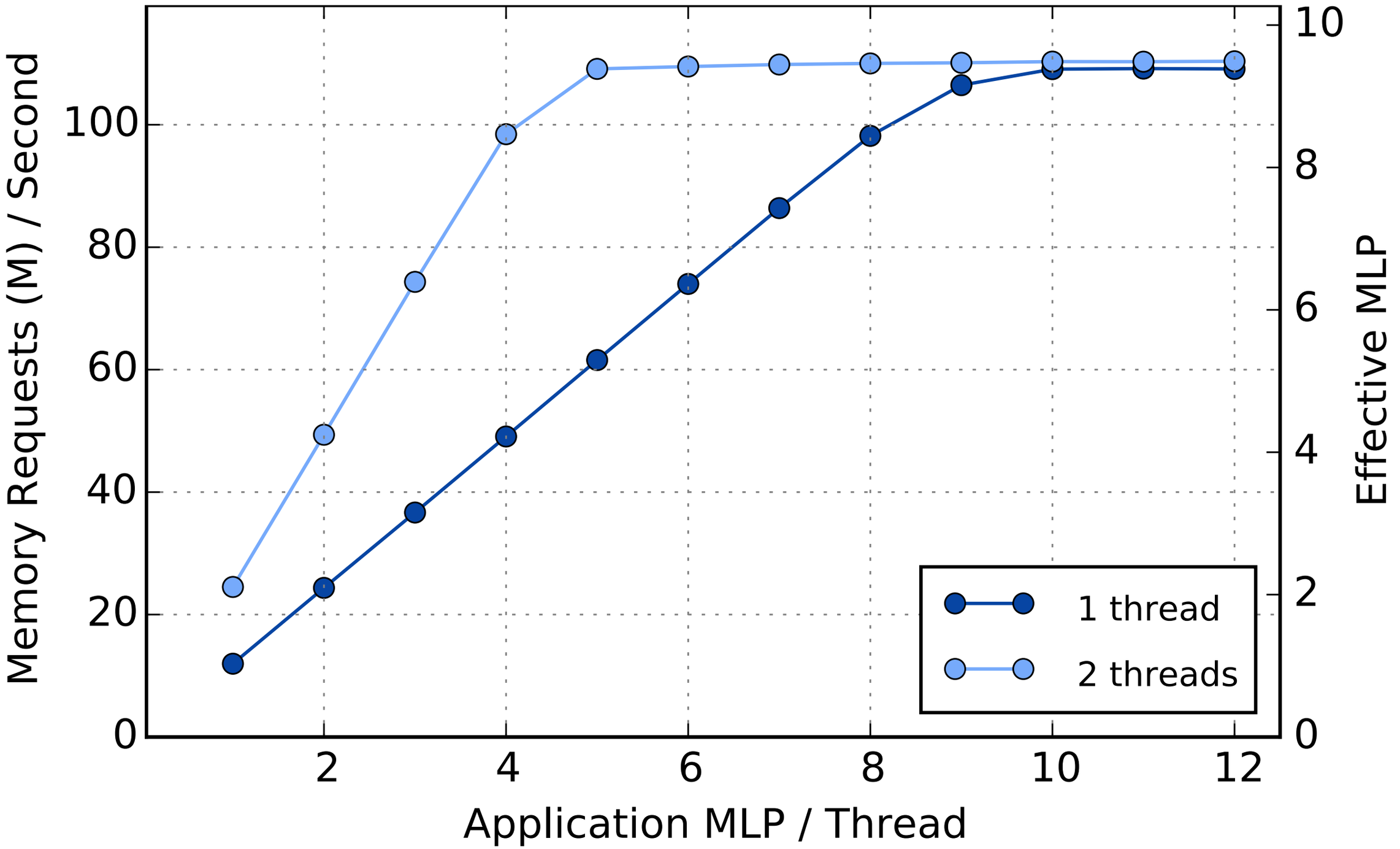
- Example access pattern: 10, 10, 0, 1, 2, 11
  - (10) has temporal locality
  - (0-2) and (10-11) have spatial locality

Single-Core Memory Bandwidth



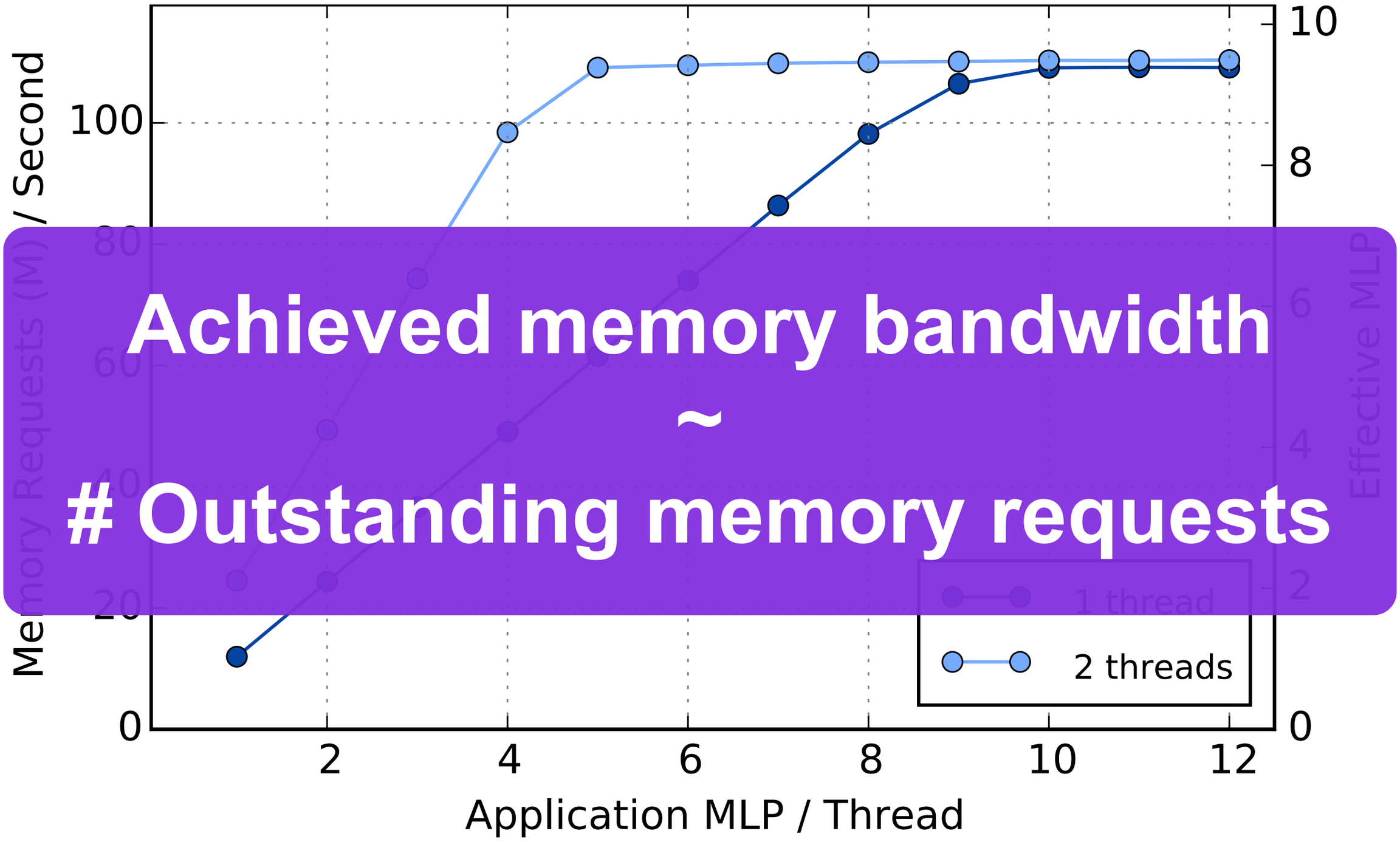
1 core (Ivy Bridge)

Single-Core Memory Bandwidth



1 core (Ivy Bridge)

Single-Core Memory Bandwidth



Achieved memory bandwidth

~

 # Outstanding memory requests

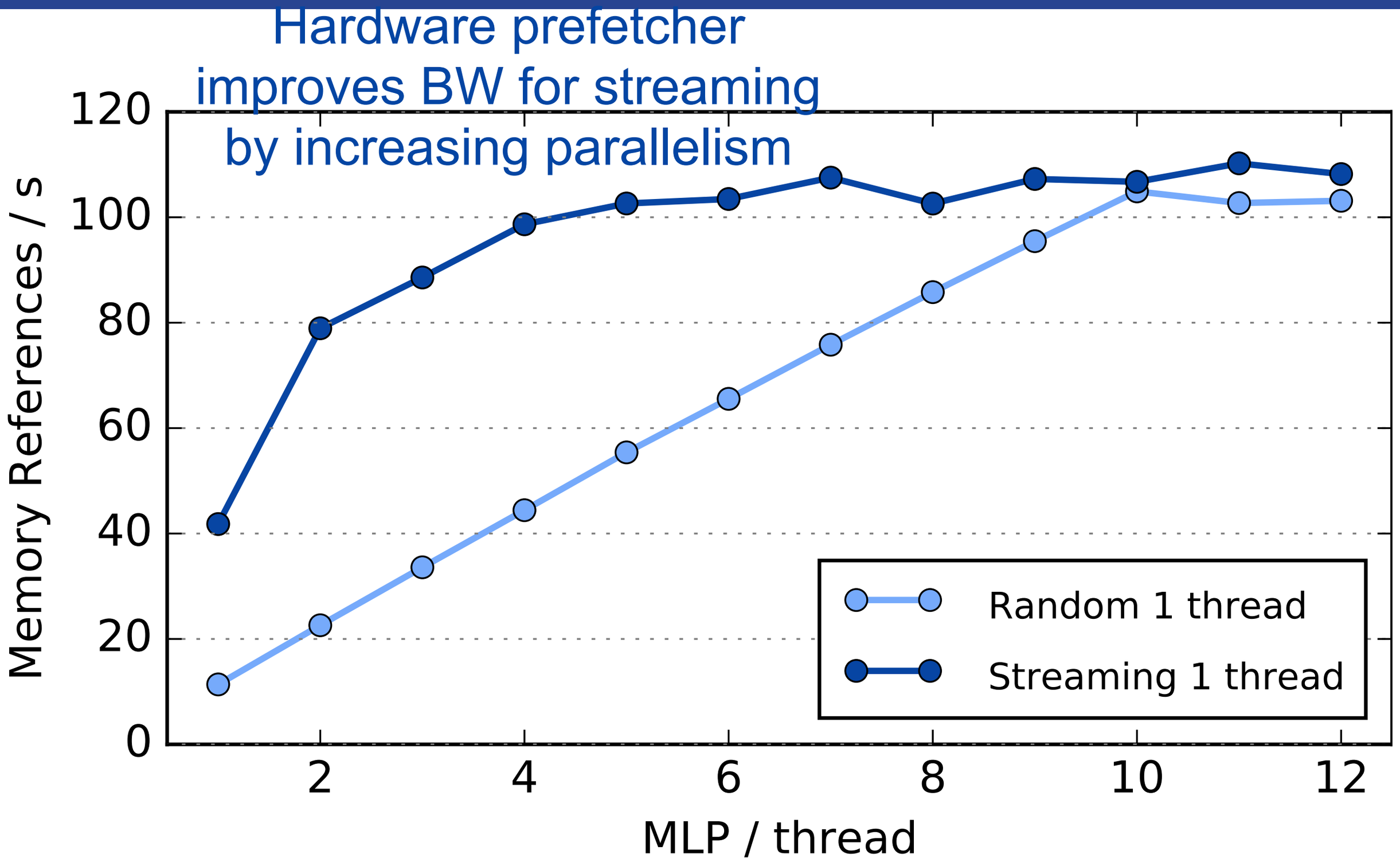
1 core (Ivy Bridge)

Processors Design for Spatial Locality

- *Cache lines*/blocks are typically far larger than a single word (64B vs 4B)
- *Hardware prefetchers* try to predict memory access patterns and bring data automatically (in advance) to cache
  - Prefetchers excel at streaming access patterns but struggle with "random"
- DRAM row hit - can save time and energy if requested page already open

Streaming Benefit Decreases w/ MLP

Hardware prefetcher improves BW for streaming by increasing parallelism

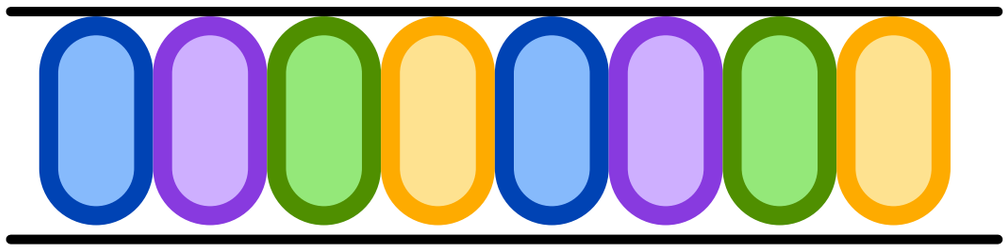


1 core (Sandy Bridge)

Types of Wasteful Communication

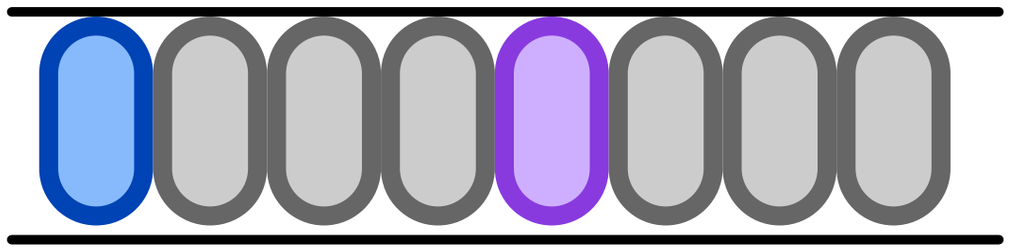
Refetch

- poor *temporal locality*
  - re-load previously accessed data



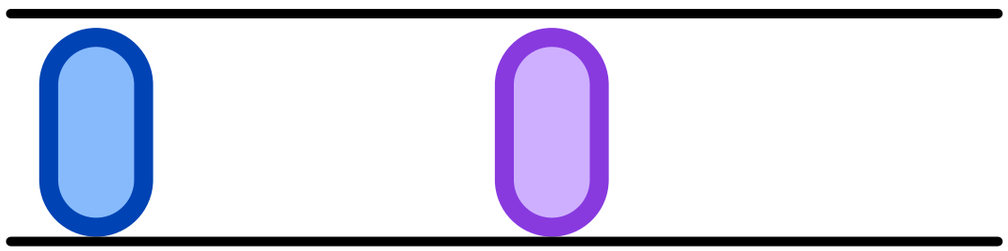
Overfetch

- poor *spatially locality*
  - transfer unnecessary data



Underfetch

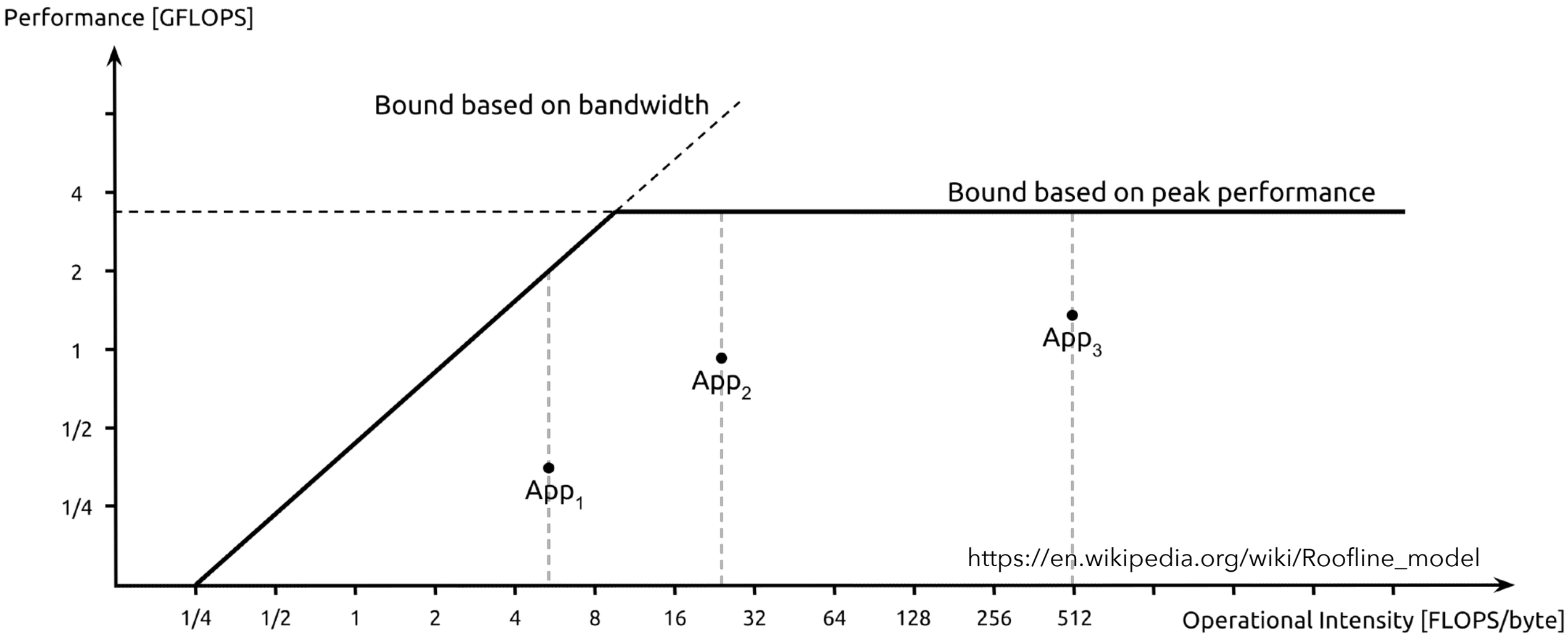
- poor *request parallelism*
  - transfer bandwidth underutilized



Peer Instruction

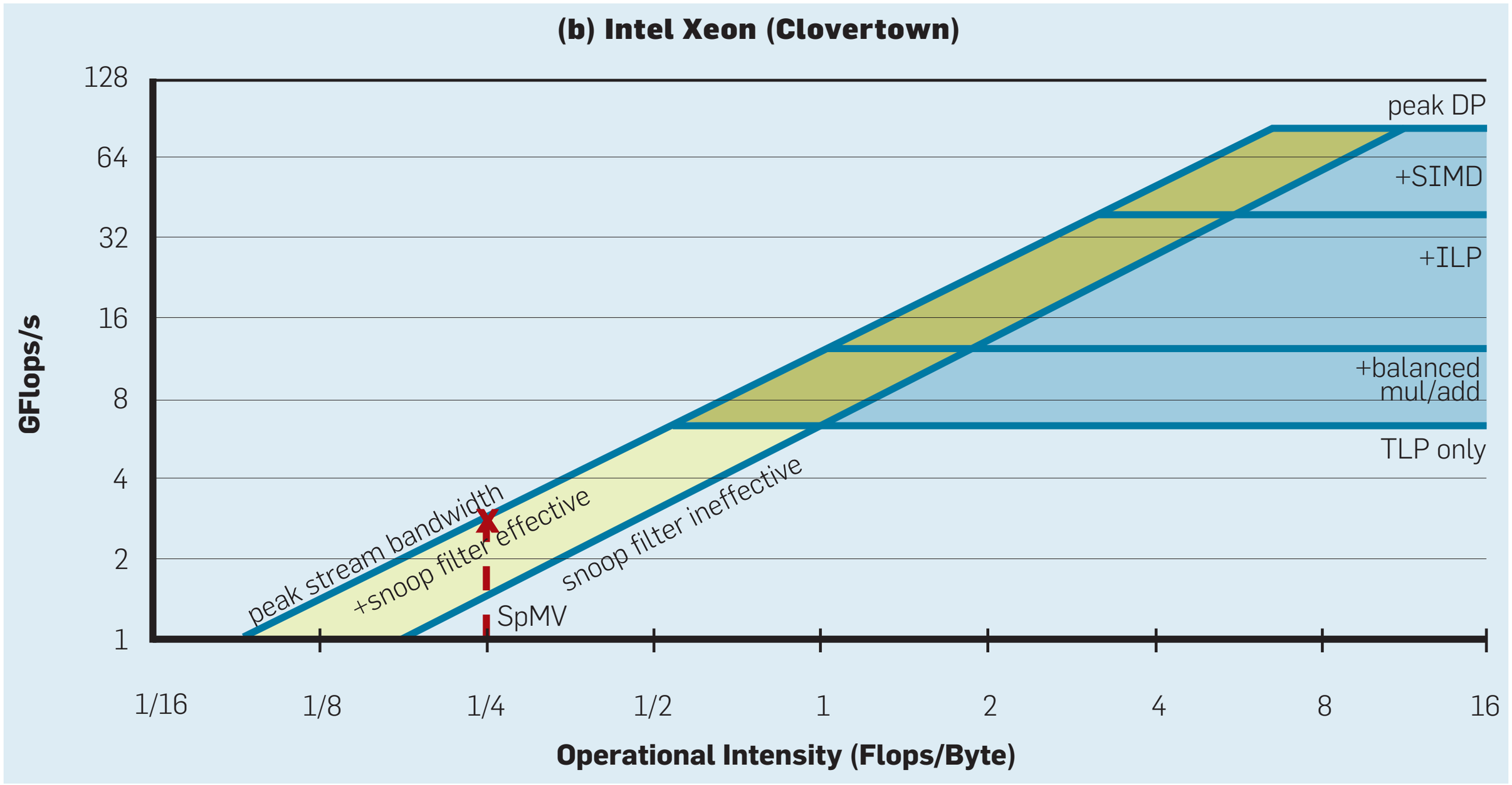
- Suppose you have a video decoder capable of playing 4K @ 60Hz, and you need to upgrade it to handle 8K @ 120 Hz. Which of the following accelerator optimizations will work? (Y/N for each)
  - 1) Halve latency & quadruple parallelism
  - 2) Reduce latency by 8x
  - 3) Increase parallelism by 8x
- Process: solve individually, vote individually, talk to neighbor, agree on answer, vote as pairs

Roofline Model



- Visual way to understand how close workload is to compute or bandwidth limits

Example Roofline



"Roofline: An insightful Visual Performance model for multicore Architectures" (CACM 2009)

How To Measure Workloads

- Qualitative approaches
  - Examining source code
  - Analytic models or complexity analysis
- Quantitative tools
  - Software Profilers
  - Hardware performance counters
  - Architecture Simulators
- Qualitative approaches are insufficient, so should use quantitative tools and compare
  - Do measurements and models agree?

Software Profilers

- Intermittently pause workload and sample state
- With enough samples, can get some statistics about workload's behavior

- Examples: linux perf, gprof, gperftools, valgrind
- (+) Easiest to get going
- (+) Great for finding where workload spends most of time (hot regions)
- (-) Can't measure too much of microarchitecture

Hardware Performance Counters

- Modern platforms typically contain extra hardware to monitor performance events with no overhead
- Software needs to be configured to read the counters and even control what they count
- Examples: linux perf, Intel PCM, PAPI, likwid
- (-) Can be very hard to use (poorly documented, buggy, limited)
- (+) Very accurate since measuring actual thing

Architecture Simulator

- Software that simulates architecture of interest
- Examples: gem5, sniper, ESESC
- (+) Can change architecture and see workload's sensitivity
- (+) Can measure arbitrary events
- (-) Slow (but maybe speed up with FireSim)
- (-) Need to verify models needed detail

Characterization Summary

- Iterate on characterization with its user to make sure it makes sense (& is useful)
  - Scripting reduces burden to rerun
- Memory accesses are a common bottleneck, so probably need to optimize
- Keep an eye out for *locality* & *parallelism*, as these are most often exploited by accelerators