Performance Tuning of Scientific Codes with the Roofline Model

1:30pm	Introduction to Roofline	Samuel Williams
2:00pm	Using Roofline in NESAP	Jack Deslippe
2:20pm	Using LIKWID for Roofline	Charlene Yang
2:40pm	Using NVProf for Roofline	Protonu Basu
3:00pm	break / setup NERSC accounts	
3:30pm	Introduction to Intel Advisor	Charlene Yang
3:50pm	Hands-on with Intel Advisor	Samuel Williams
4:45pm	closing remarks / Q&A	all

Introductions

Samuel Williams
Computational Research Division
Lawrence Berkeley National Lab
SWWilliams@lbl.gov

Jack Deslippe
NERSC
Lawrence Berkeley National Lab
JRDeslippe@lbl.gov

Charlene Yang
NERSC
Lawrence Berkeley National Lab
CJYang@lbl.gov

Protonu Basu
Computational Research Division
Lawrence Berkeley National Lab
PBasu@lbl.gov

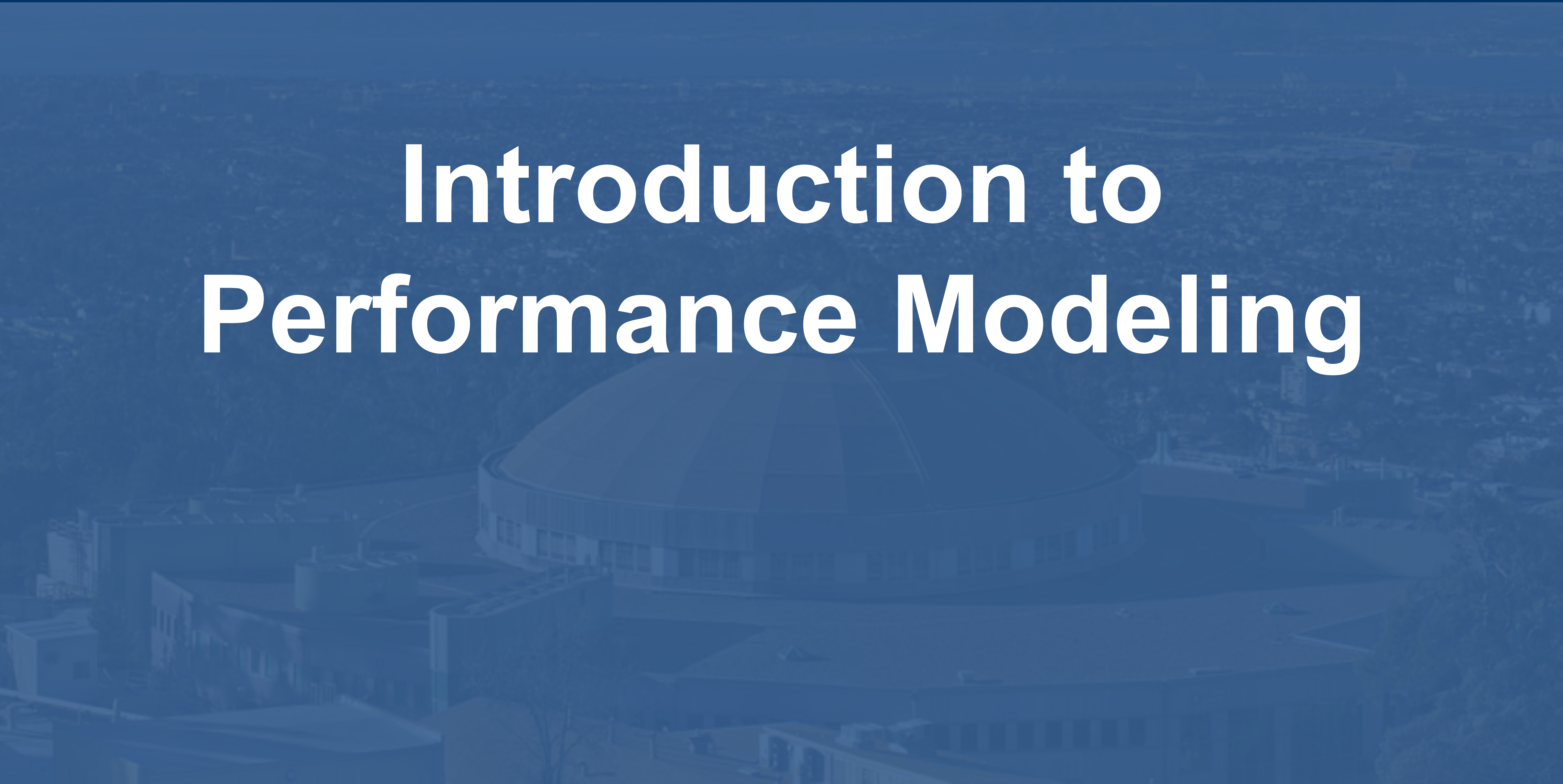
Introduction to the Roofline Model

Samuel Williams
Computational Research Division
Lawrence Berkeley National Lab
SWWilliams@lbl.gov

Acknowledgements

- This material is based upon work supported by the Advanced Scientific Computing Research Program in the U.S. Department of Energy, Office of Science, under Award Number DE-AC02-05CH11231.
- This material is based upon work supported by the DOE RAPIDS SciDAC Institute.
- This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.
- Special Thanks to:
  - Zakhar Matveev, Intel Corporation
  - Roman Belenov, Intel Corporation

Introduction to Performance Modeling



Why Use Performance Models or Tools?

- Identify performance bottlenecks
- Motivate software optimizations
- **Determine when we're done optimizing**
  - Assess performance relative to machine capabilities
  - Motivate need for algorithmic changes
- Predict performance on future machines / architectures
  - Sets realistic expectations on performance for future procurements
  - Used for HW/SW Co-Design to ensure future architectures are well-suited for the computational needs of today's applications.

Performance Models

- Many different components can contribute to kernel run time.
- Some are characteristics of the application, some are characteristics of the machine, and some are both (memory access pattern + caches).

#FP operations	Flop/s
Cache data movement	Cache GB/s
DRAM data movement	DRAM GB/s
PCIe data movement	PCIe bandwidth
Depth	OMP Overhead
MPI Message Size	Network Bandwidth
MPI Send:Wait ratio	Network Gap
#MPI Wait's	Network Latency

Performance Models

- Can't think about all these terms all the time for every application…

Computational Complexity

#FP operations	Flop/s
Cache data movement	Cache GB/s
DRAM data movement	DRAM GB/s
PCIe data movement	PCIe bandwidth
Depth	OMP Overhead
MPI Message Size	Network Bandwidth
MPI Send:Wait ratio	Network Gap
#MPI Wait's	Network Latency

Performance Models

- Because there are so many components, performance models often conceptualize the system as being dominated by one or more of these components.

#FP operations	Flop/s
Cache data movement	Cache GB/s
DRAM data movement	DRAM GB/s
PCIe data movement	PCIe bandwidth
Depth	OMP Overhead
MPI Message Size	Network Bandwidth
MPI Send:Wait ratio	Network Gap
#MPI Wait's	Network Latency

Roofline Model

Williams et al, "Roofline: An Insightful Visual Performance Model For Multicore Architectures", CACM, 2009.

Performance Models

- Because there are so many components, performance models often conceptualize the system as being dominated by one or more of these components.

	#FP operations	Flop/s
	Cache data movement	Cache GB/s
	DRAM data movement	DRAM GB/s
LogCA	PCIe data movement	PCIe bandwidth
	Depth	OMP Overhead
	MPI Message Size	Network Bandwidth
	MPI Send:Wait ratio	Network Gap
	#MPI Wait's	Network Latency

Bin Altaf et al, "LogCA: A High-Level Performance Model for Hardware Accelerators", ISCA, 2017.

Performance Models

- Because there are so many components, performance models often conceptualize the system as being dominated by one or more of these components.

#FP operations	Flop/s
Cache data movement	Cache GB/s
DRAM data movement	DRAM GB/s
PCIe data movement	PCIe bandwidth
Depth	OMP Overhead
MPI Message Size	Network Bandwidth
MPI Send:Wait ratio	Network Gap
#MPI Wait's	Network Latency

LogGP

Alexandrov, et al, "LogGP: incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation", SPAA, 1995.

Performance Models

- Because there are so many components, performance models often conceptualize the system as being dominated by one or more of these components.

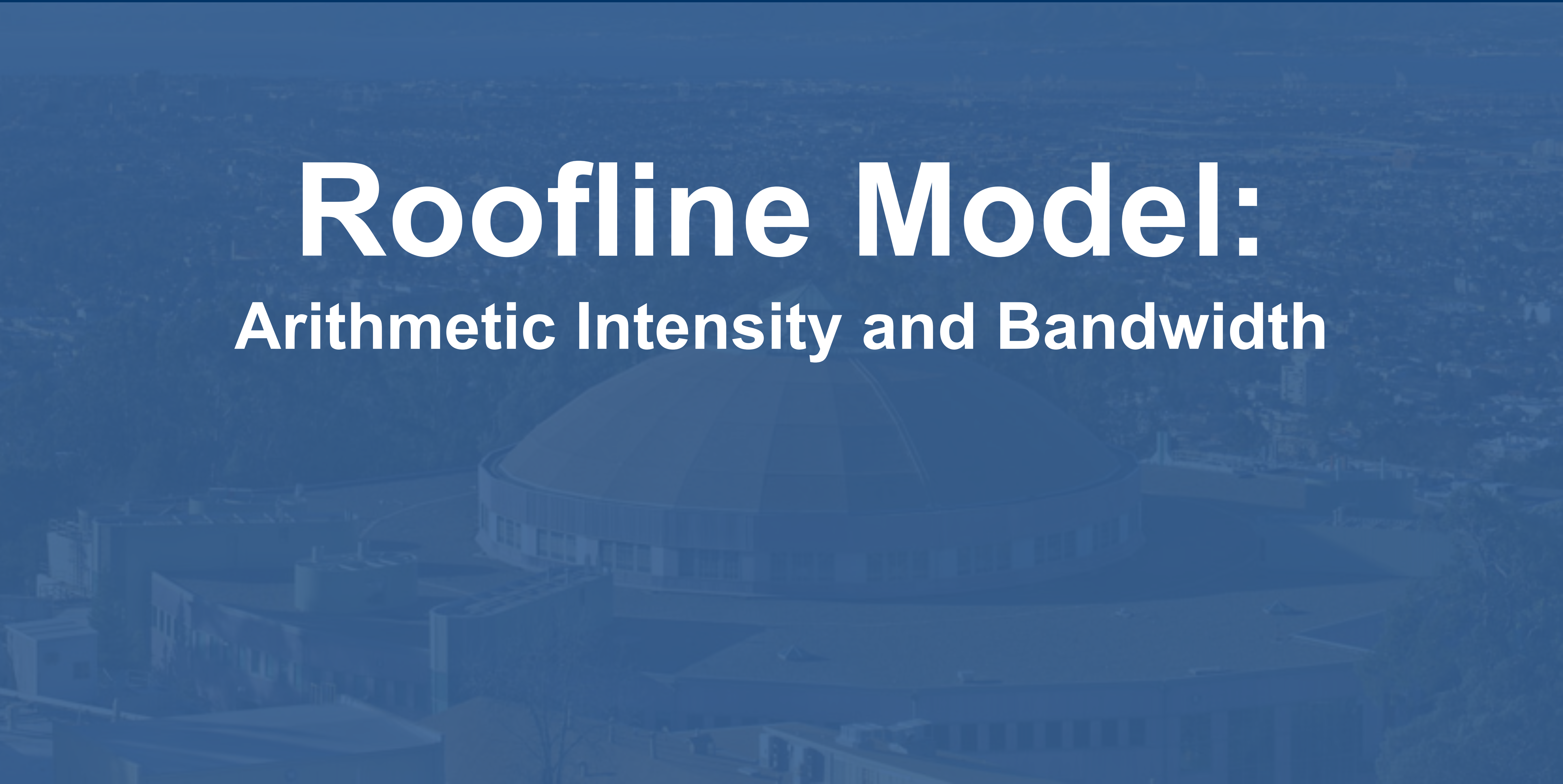
#FP operations	Flop/s
Cache data movement	Cache GB/s
DRAM data movement	DRAM GB/s
PCIe data movement	PCIe band
Depth	OMP O
MPI Message Size	Network B
MPI Send:Wait ratio	Network Ga
#MPI Wait's	Network Latency

LogP

Right model depends on app and problem size

!

Culler, et al, "LogP: a practical model of parallel computation", CACM, 1996.



Roofline Model:

Arithmetic Intensity and Bandwidth

Performance Models / Simulators

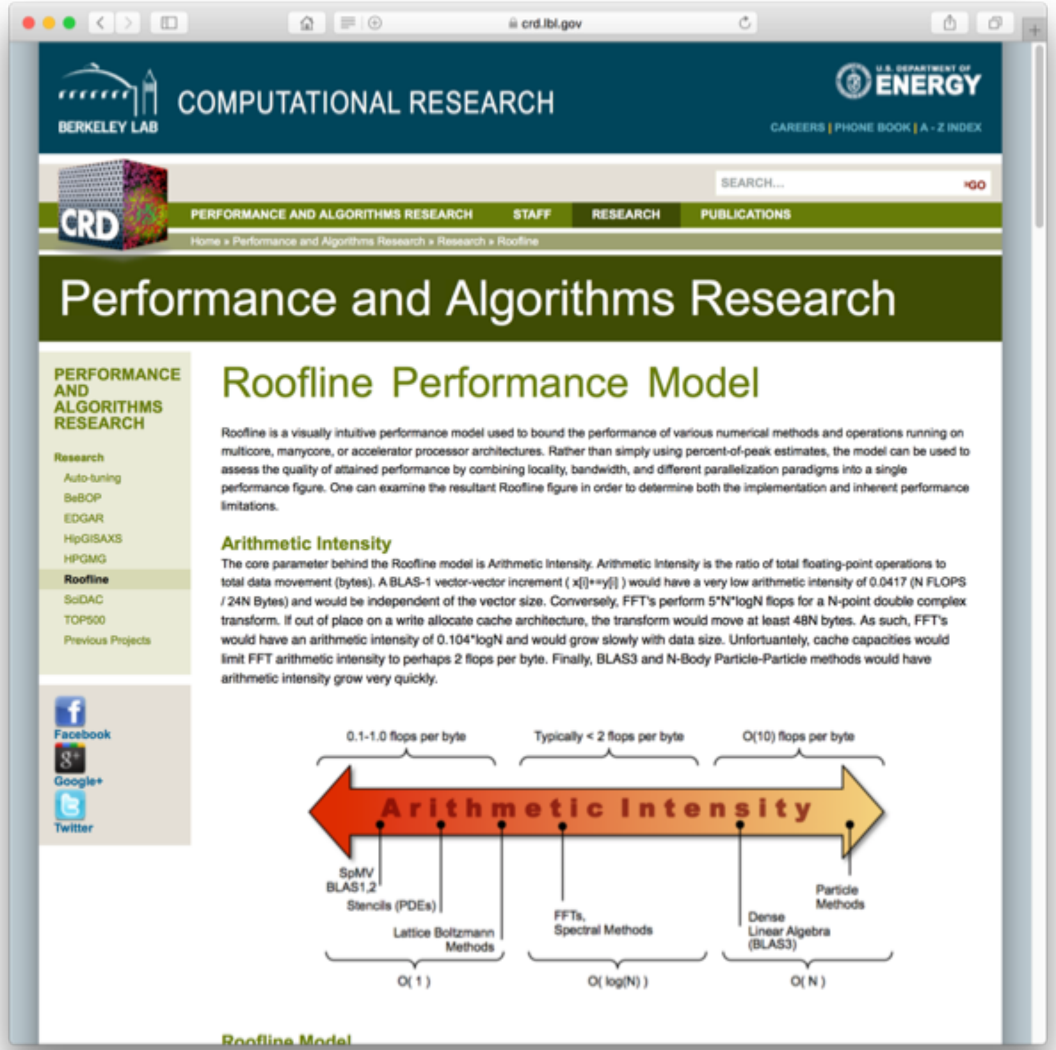
- Historically, many performance models and simulators tracked latencies to predict performance (i.e. counting cycles)

- The last two decades saw a number of latency-hiding techniques…
  - Out-of-order execution (hardware discovers parallelism to hide latency)
  - HW stream prefetching (hardware speculatively loads data)
  - Massive thread parallelism (independent threads satisfy the latency-bandwidth product)

- Effective latency hiding has resulted in a shift from a latency-limited computing regime to a **throughput-limited computing regime**

Roofline Model

- **Roofline Model** is a throughput-oriented performance model…
  - Tracks rates not times
  - Augmented with Little's Law
    (concurrency = latency*bandwidth)
  - Independent of ISA and architecture (applies to CPUs, GPUs, Google TPUs[1], etc…)



crd.lbl.gov

BERKELEY LAB COMPUTATIONAL RESEARCH

U.S. DEPARTMENT OF ENERGY

CAREERS | PHONE BOOK | A - Z INDEX

SEARCH... GO

CRD PERFORMANCE AND ALGORITHMS RESEARCH STAFF RESEARCH PUBLICATIONS

Home » Performance and Algorithms Research » Research » Roofline

Performance and Algorithms Research

PERFORMANCE AND ALGORITHMS RESEARCH

Research
Auto-tuning
BeBOP
EDGAR
HipGISAXS
HPGMG
Roofline
SciDAC
TOP500
Previous Projects

Facebook
Google+
Twitter

Roofline Performance Model

Roofline is a visually intuitive performance model used to bound the performance of various numerical methods and operations running on multicore, manycore, or accelerator processor architectures. Rather than simply using percent-of-peak estimates, the model can be used to assess the quality of attained performance by combining locality, bandwidth, and different parallelization paradigms into a single performance figure. One can examine the resultant Roofline figure in order to determine both the implementation and inherent performance limitations.

Arithmetic Intensity

The core parameter behind the Roofline model is Arithmetic Intensity. Arithmetic Intensity is the ratio of total floating-point operations to total data movement (bytes). A BLAS-1 vector-vector increment (x[i]+=y[i]) would have a very low arithmetic intensity of 0.0417 (N FLOPS / 24N Bytes) and would be independent of the vector size. Conversely, FFT's perform 5*N*logN flops for a N-point double complex transform. If out of place on a write allocate cache architecture, the transform would move at least 48N bytes. As such, FFT's would have an arithmetic intensity of 0.104*logN and would grow slowly with data size. Unfortuantely, cache capacities would limit FFT arithmetic intensity to perhaps 2 flops per byte. Finally, BLAS3 and N-Body Particle-Particle methods would have arithmetic intensity grow very quickly.

Roofline Model

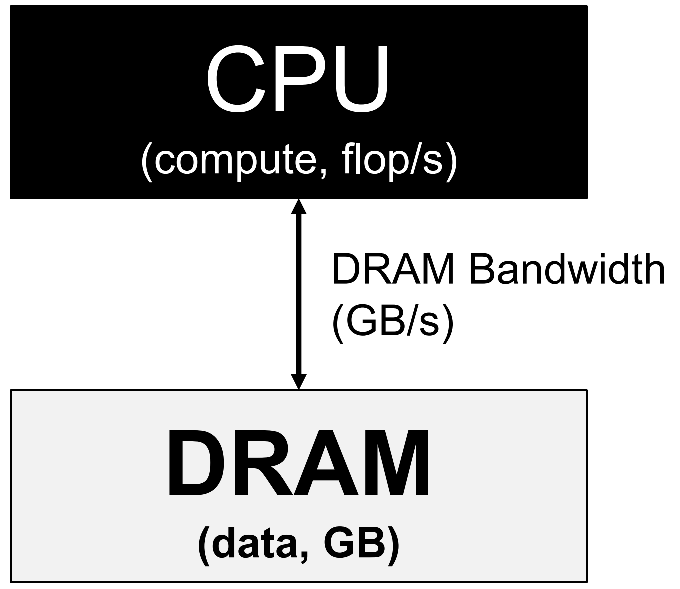
https://crd.lbl.gov/departments/computer-science/PAR/research/roofline

[1]Jouppi et al, "In-Datacenter Performance Analysis of a Tensor Processing Unit", ISCA, 2017.

(DRAM) Roofline

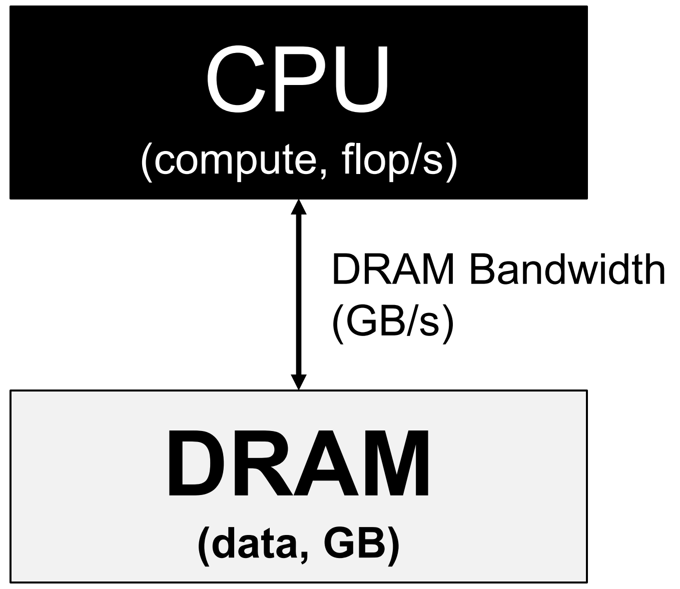
- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)

$$\text{Time} = \max \begin{cases} \#\text{FP ops / Peak GFlop/s} \\ \#\text{Bytes / Peak GB/s} \end{cases}$$



(DRAM) Roofline

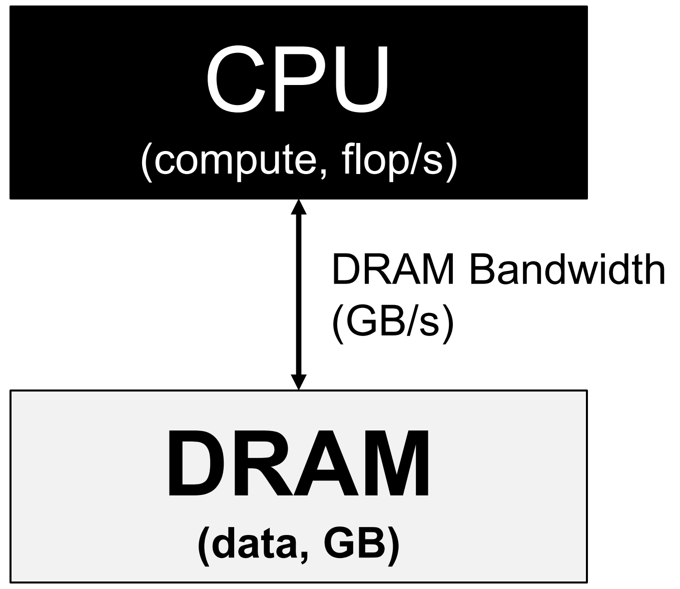
- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)



$$\frac{\text{Time}}{\text{\#FP ops}} = \max \begin{cases} 1 \,/\, \text{Peak GFlop/s} \\ \text{\#Bytes} \,/\, \text{\#FP ops} \,/\, \text{Peak GB/s} \end{cases}$$

(DRAM) Roofline

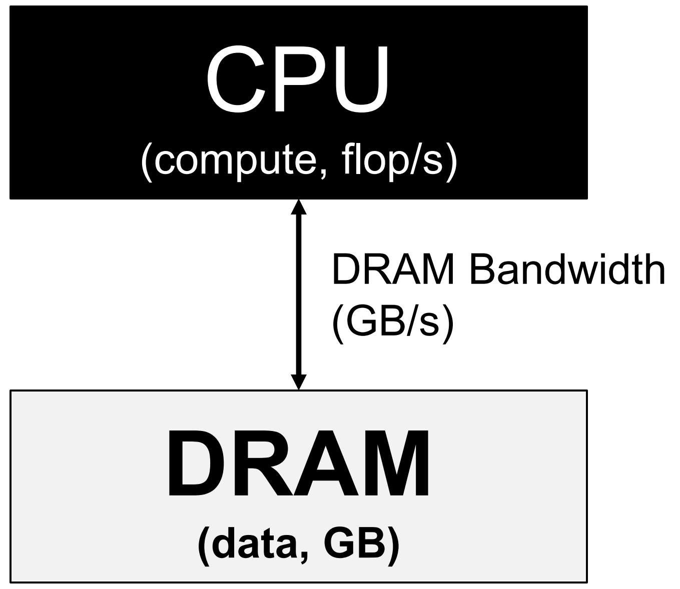
- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)



$$\frac{\#\text{FP ops}}{\text{Time}} = \min \begin{cases} \text{Peak GFlop/s} \\ (\#\text{FP ops} / \#\text{Bytes}) * \text{Peak GB/s} \end{cases}$$

(DRAM) Roofline

- One could hope to always attain peak performance (Flop/s)
- However, finite locality (reuse) and bandwidth limit performance.
- Assume:
  - Idealized processor/caches
  - Cold start (data in DRAM)

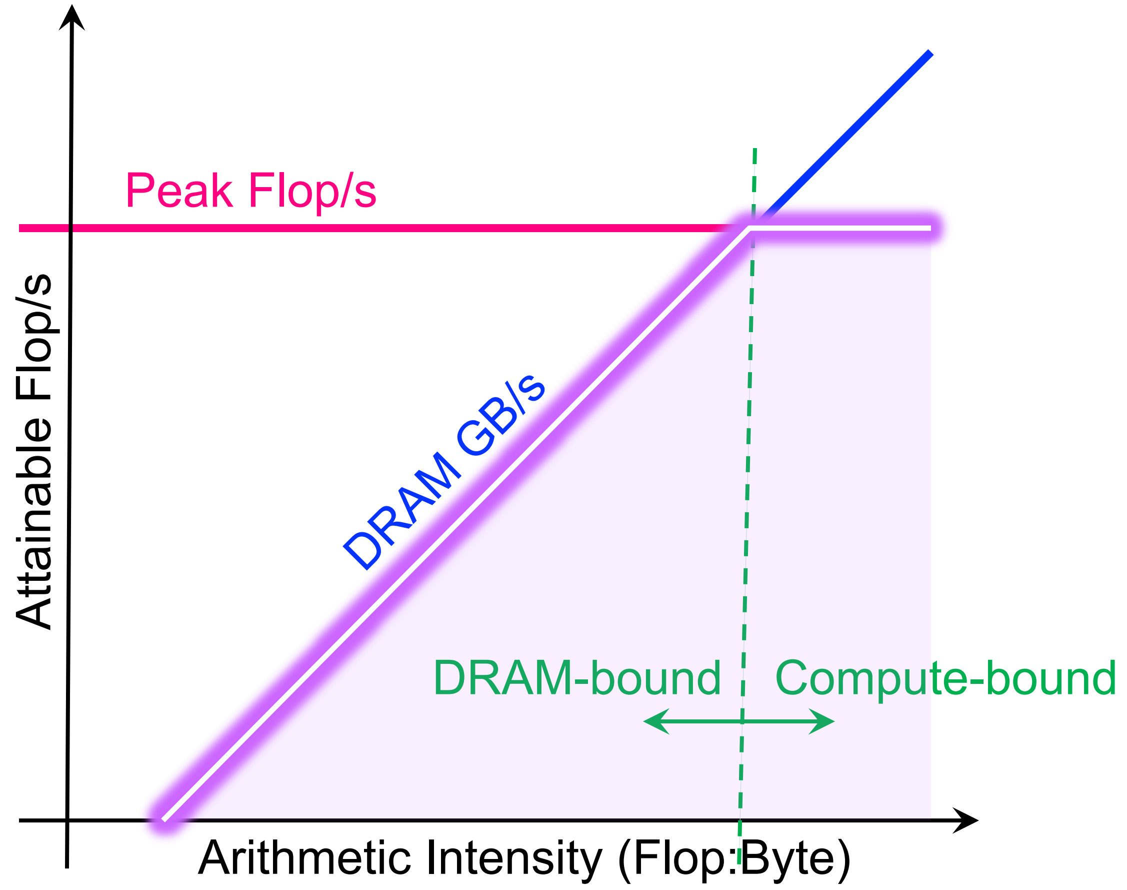


$$\textbf{GFlop/s} = \min \begin{cases} \textbf{Peak GFlop/s} \\ \textbf{AI * Peak GB/s} \end{cases}$$

Note, Arithmetic Intensity (AI) = Flops / Bytes (as presented to DRAM)

(DRAM) Roofline

- Plot Roofline bound using Arithmetic Intensity as the x-axis
- **Log-log scale** makes it easy to doodle, extrapolate performance along Moore's Law, etc…
- Kernels with AI less than machine balance are ultimately DRAM bound (we'll refine this later…)



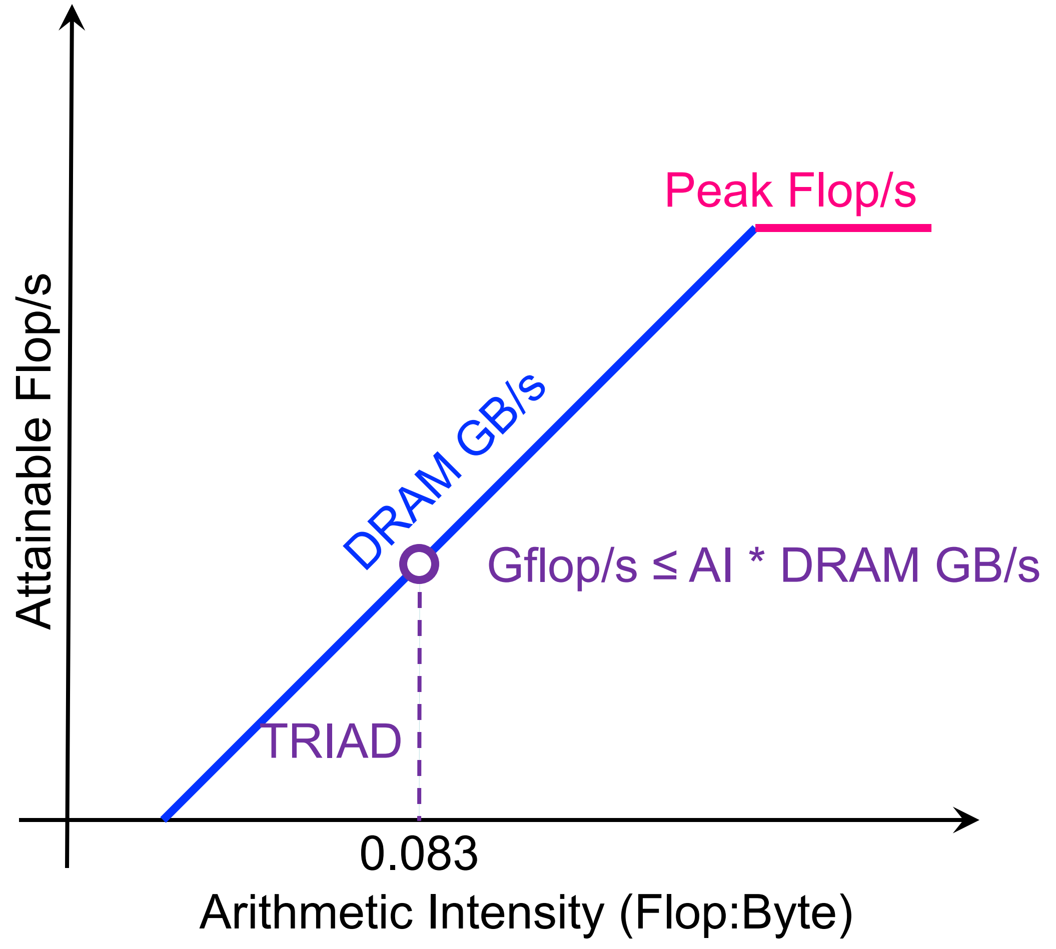
Roofline Example #1

- Typical machine balance is 5-10 flops per byte…
  - 40-80 flops per double to exploit compute capability
  - Artifact of technology and money
  - **Unlikely to improve**

- Consider STREAM Triad…

```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

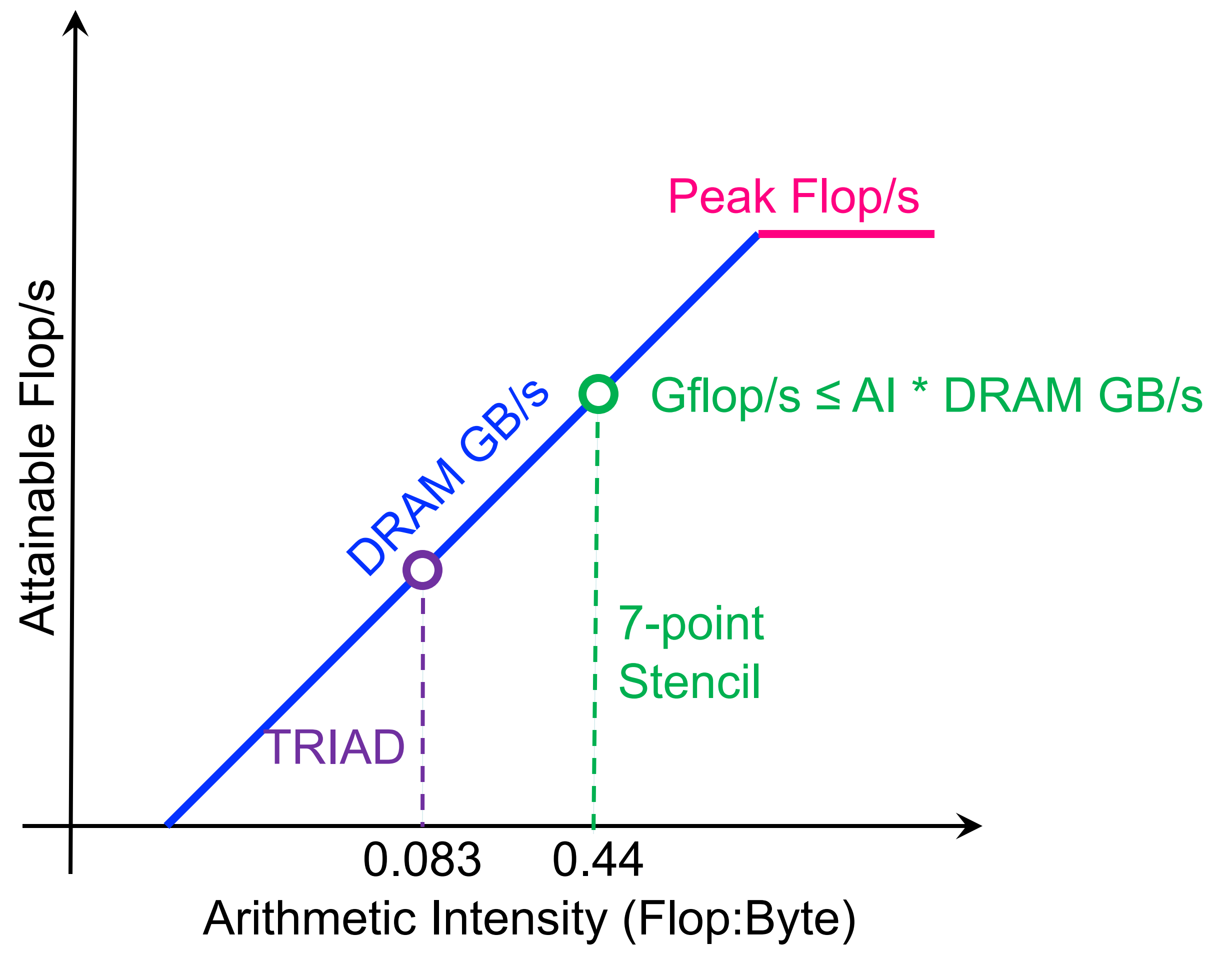
  - 2 flops per iteration
  - Transfer 24 bytes per iteration (read X[i], Y[i], write Z[i])
  - **AI = 0.083 flops per byte == Memory bound**



Roofline Example #2

- Conversely, 7-point constant coefficient stencil…
  - 7 flops
  - 8 memory references (7 reads, 1 store) per point
  - Cache can filter all but 1 read and 1 write per point
  - **AI = 0.44 flops per byte == memory bound, but 5x the flop rate**

```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                    + old[k  ][j  ][i-1]
                    + old[k  ][j  ][i+1]
                    + old[k  ][j-1][i  ]
                    + old[k  ][j+1][i  ]
                    + old[k-1][j  ][i  ]
                    + old[k+1][j  ][i  ];
}}}
```

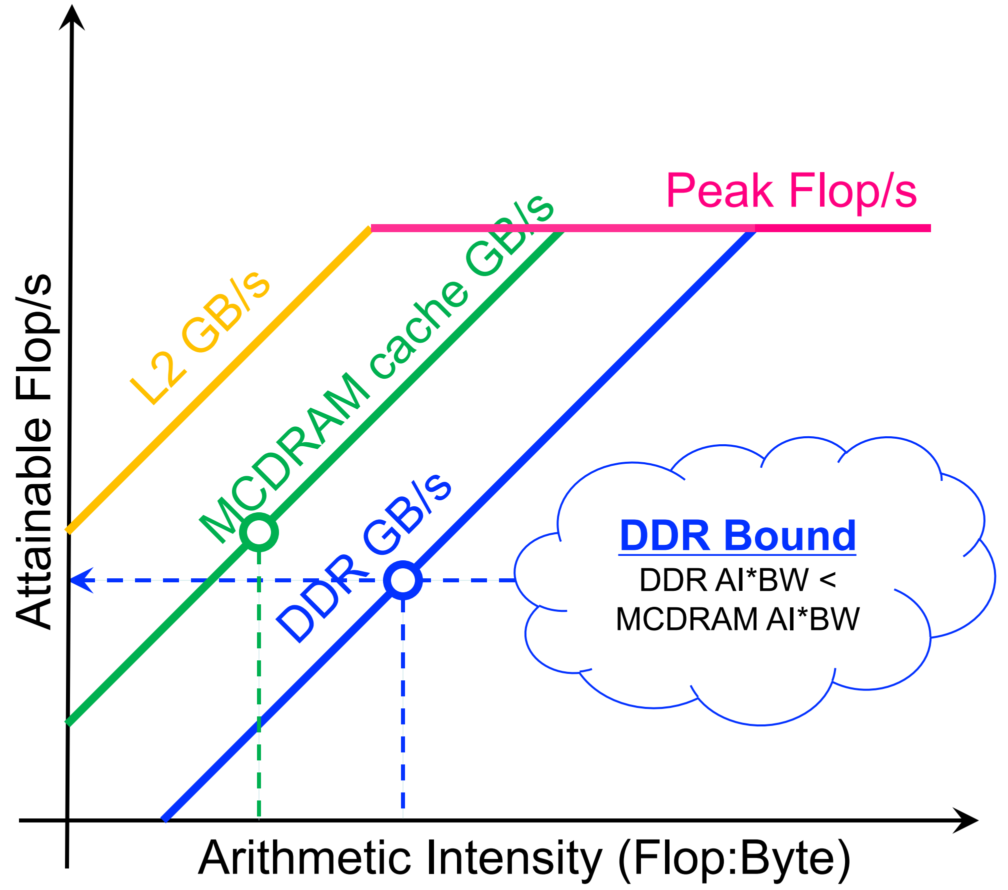


Hierarchical Roofline

- Real processors have multiple levels of memory
  - Registers
  - L1, L2, L3 cache
  - MCDRAM/HBM (KNL/GPU device memory)
  - DDR (main memory)
  - NVRAM (non-volatile memory)
- Applications can have locality in each level
  - Unique data movements imply unique AI's
  - Moreover, each level will have a unique bandwidth

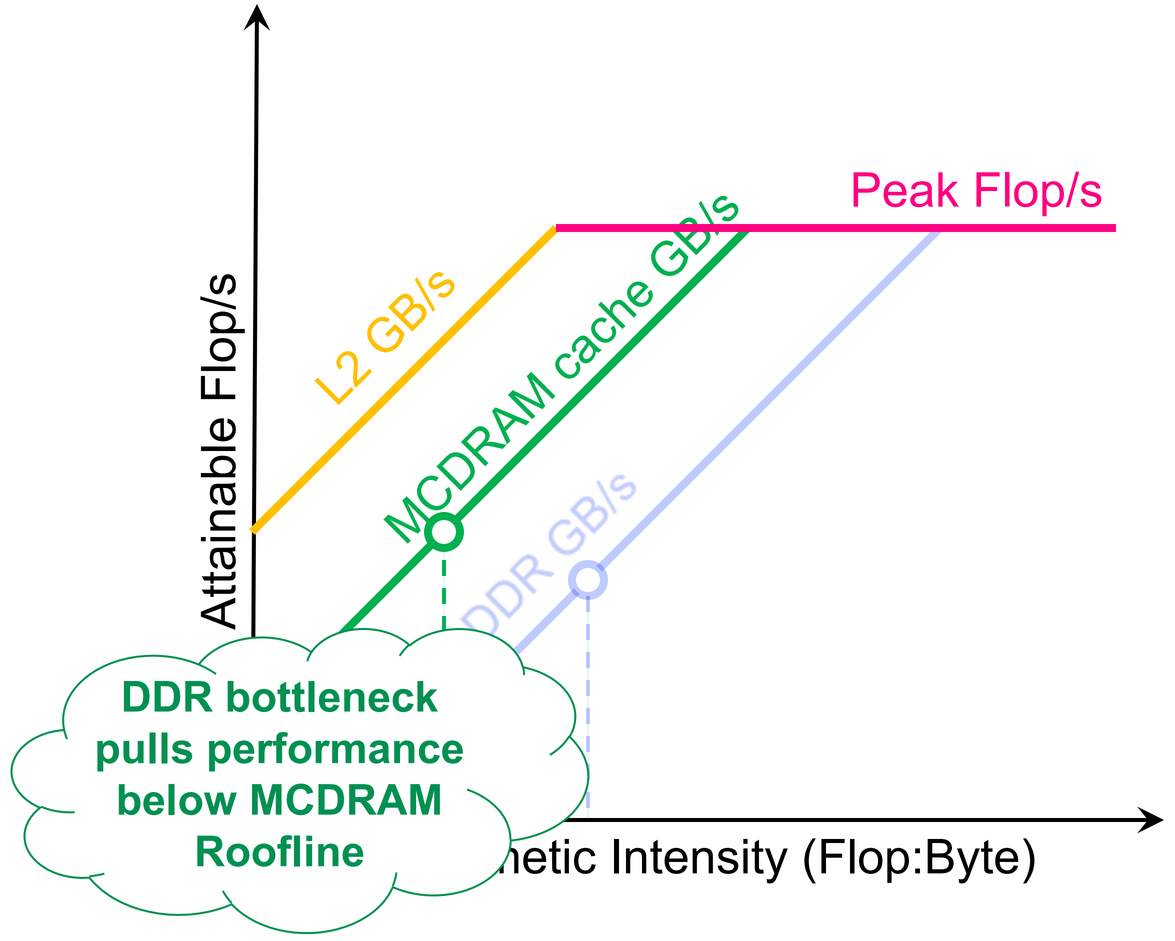
Hierarchical Roofline

- Construct superposition of Rooflines…
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM)…
  - **… performance is bound by the minimum**



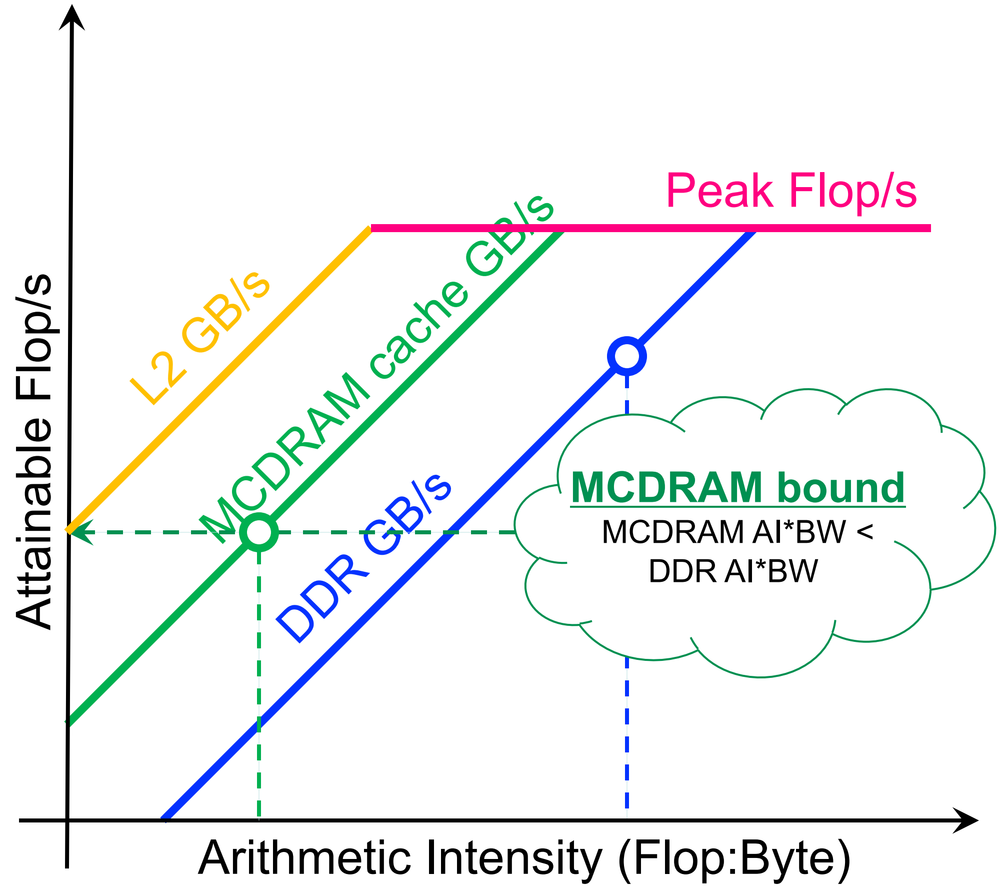
Hierarchical Roofline

- Construct superposition of Rooflines…
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM)…
  - **… performance is bound by the minimum**



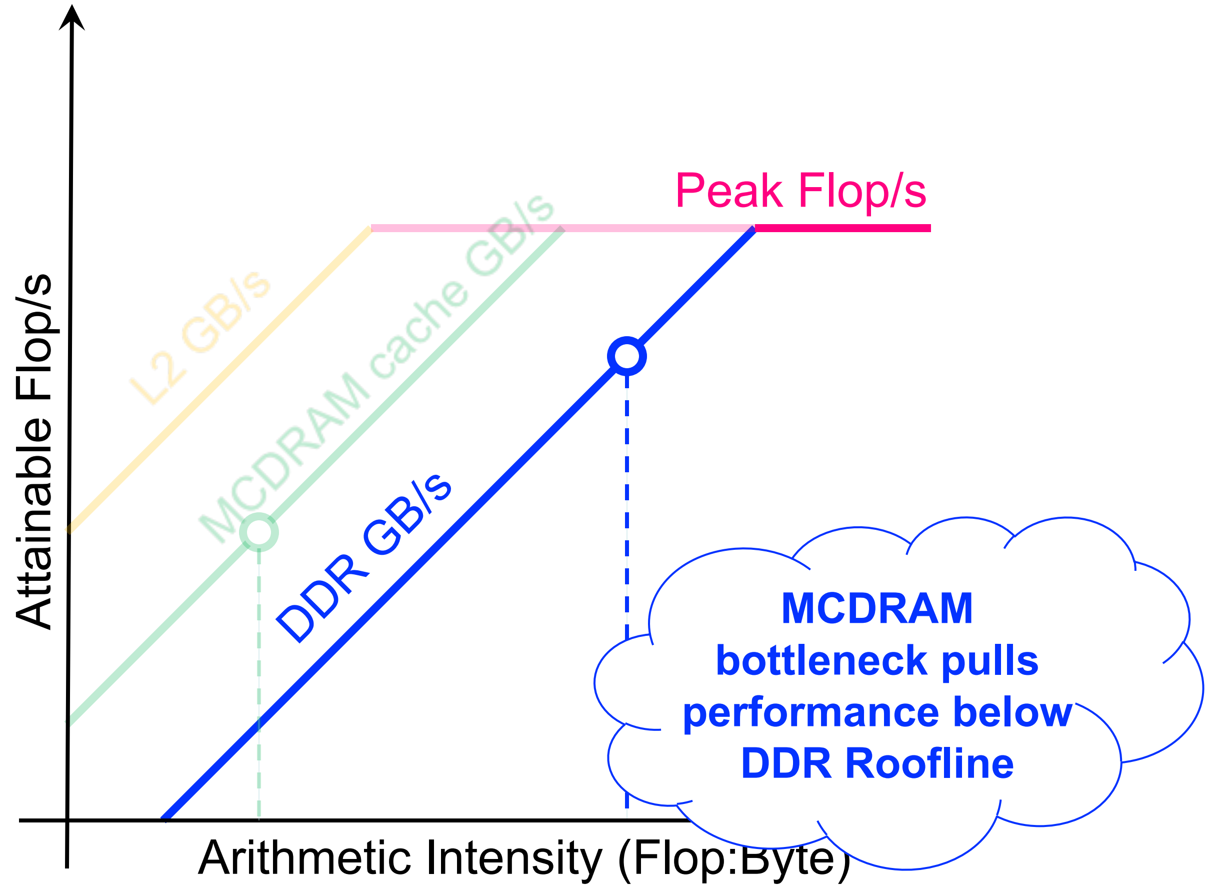
Hierarchical Roofline

- Construct superposition of Rooflines…
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM)…
  - **… performance is bound by the minimum**



Hierarchical Roofline

- Construct superposition of Rooflines…
  - Measure a bandwidth
  - Measure AI for each level of memory
  - Although an loop nest may have multiple AI's and multiple bounds (flops, L1, L2, … DRAM)…
  - **… performance is bound by the minimum**


Peak Flop/s
L2 GB/s
MCDRAM cache GB/s
DDR GB/s
Attainable Flop/s
Arithmetic Intensity (Flop:Byte)
MCDRAM bottleneck pulls performance below DDR Roofline

BERKELEY LAB

LAWRENCE BERKELEY NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

Roofline Model:

Modeling In-core Performance Effects

Data, Instruction, Thread-Level Parallelism…

- Modern CPUs use several techniques to increase per core Flop/s

Fused Multiply Add

- w = x*y + z is a common idiom in linea algebra
- Ra er f ving se d a se a add (FMA)
- Th U chains the mu tiply and add in a single pipeline so that it can complete FMA/cycle

Resurgence… Tensor Cores, QFMA, etc…

Vector Instructions

- Many HPC codes apply the same operation to a vector of elements
- Vendors provide vector instructions that apply the same operation to 2, 4, 8, 16 elements…

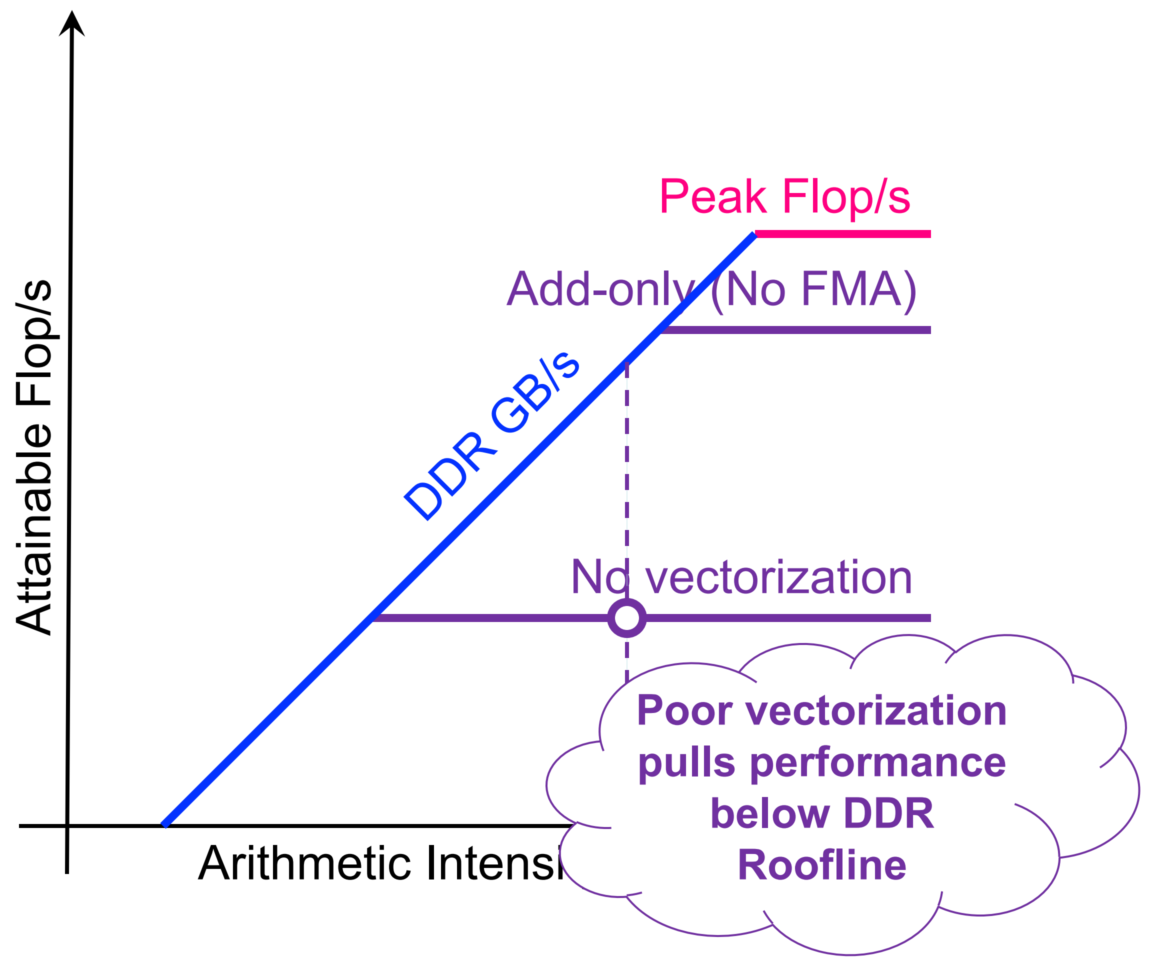
  x [0:7] *y [0:7] + z [0:7]
- Vector FPUs complete 8 vector operations/cycle

Deep pipelines

- The hardware for a FMA is substantial.
- Breaking a single FMA up into several smaller operations and pipelining them allows vendors to increase GHz
- Little's Law applies… need FP_Latency * FP_bandwidth independent instructions

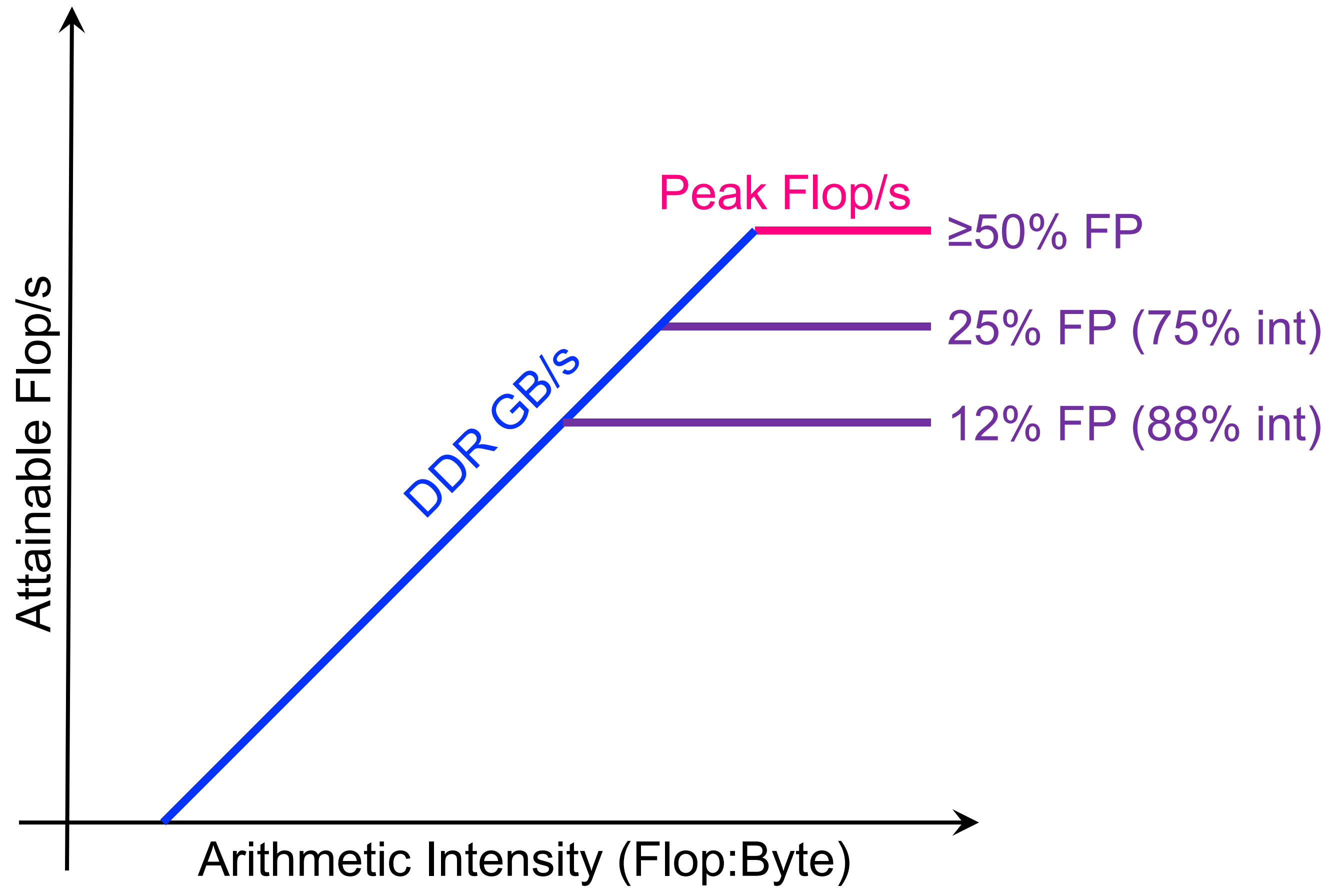
Data, Instruction, Thread-Level Parallelism…

- If every instruction were an ADD (instead of FMA), **performance would drop by 2x on KNL or 4x on Haswell**
- Similarly, if one failed to vectorize, performance would drop by **another 8x on KNL and 4x on Haswell**
- Lack of threading (or load imbalance) will reduce performance by another 64x on KNL.



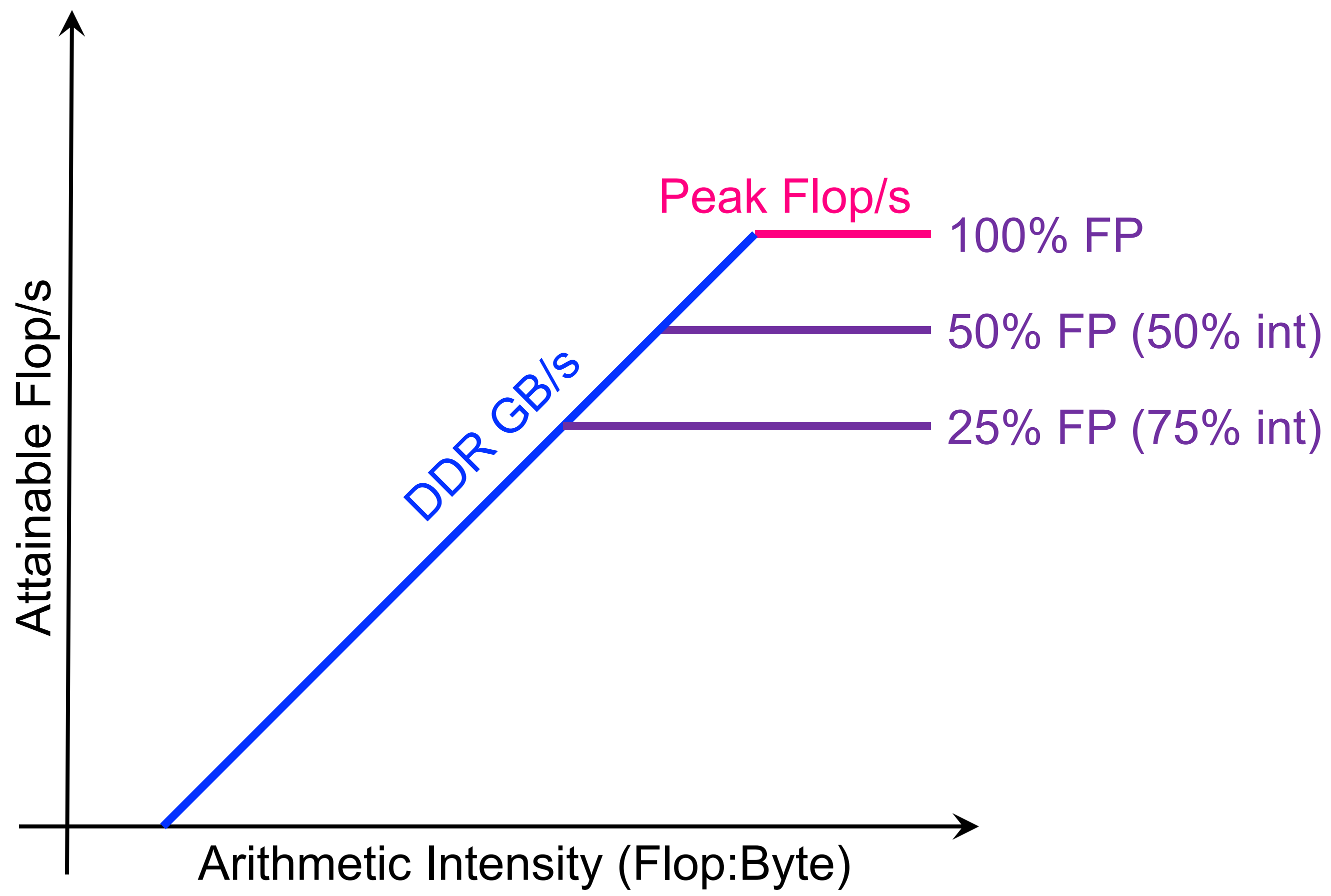
Superscalar vs. Instruction mix

- Define in-core ceilings based on instruction mix…
- e.g. Haswell
  - 4-issue superscalar
  - Only 2 FP data paths
  - Requires 50% of the instructions to be FP to get peak performance



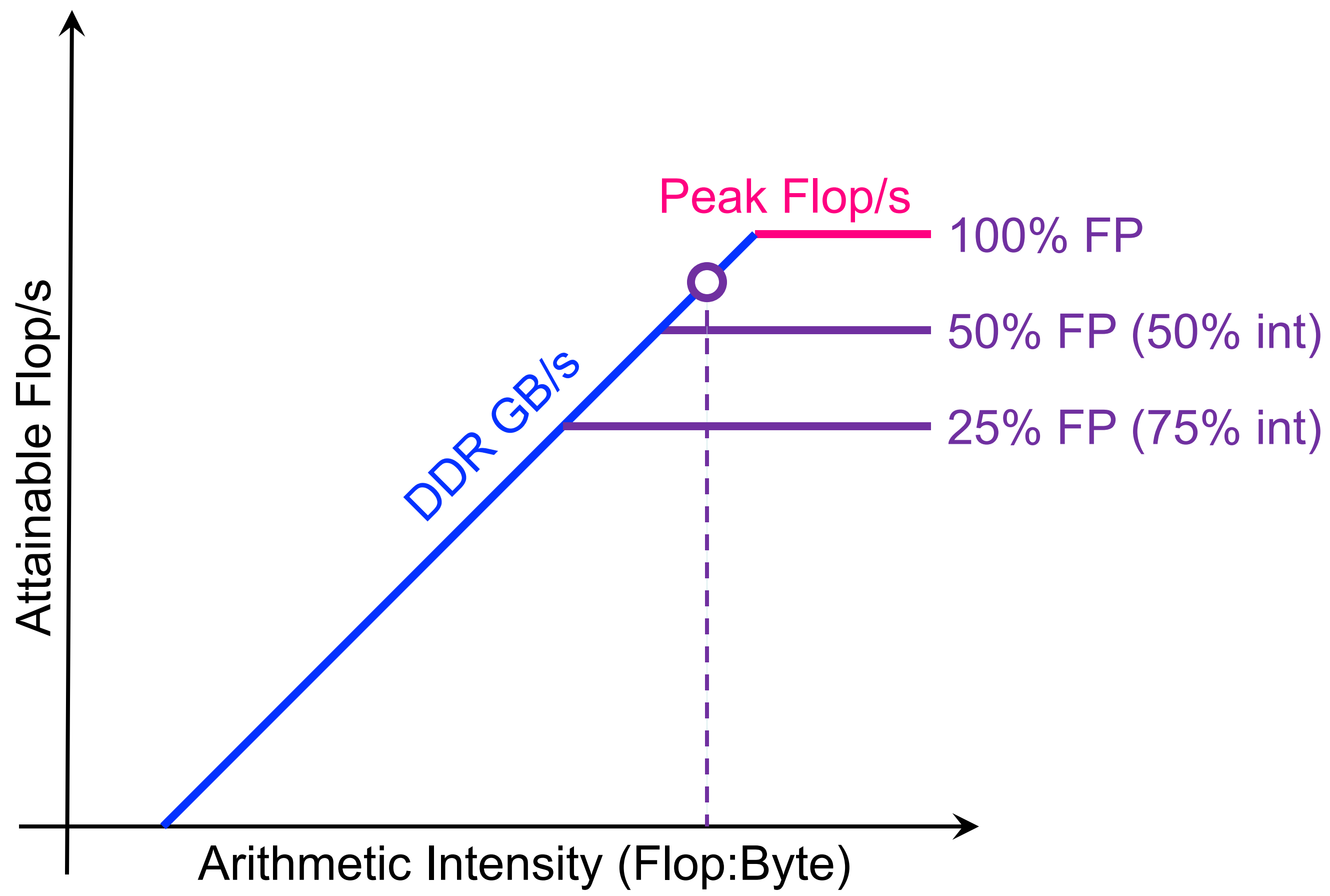
Superscalar vs. Instruction mix

- Define in-core ceilings based on instruction mix…
- e.g. Haswell
  - 4-issue superscalar
  - Only 2 FP data paths
  - Requires 50% of the instructions to be FP to get peak performance
- e.g. KNL
  - 2-issue superscalar
  - 2 FP data paths
  - Requires 100% of the instructions to be FP to get peak performance



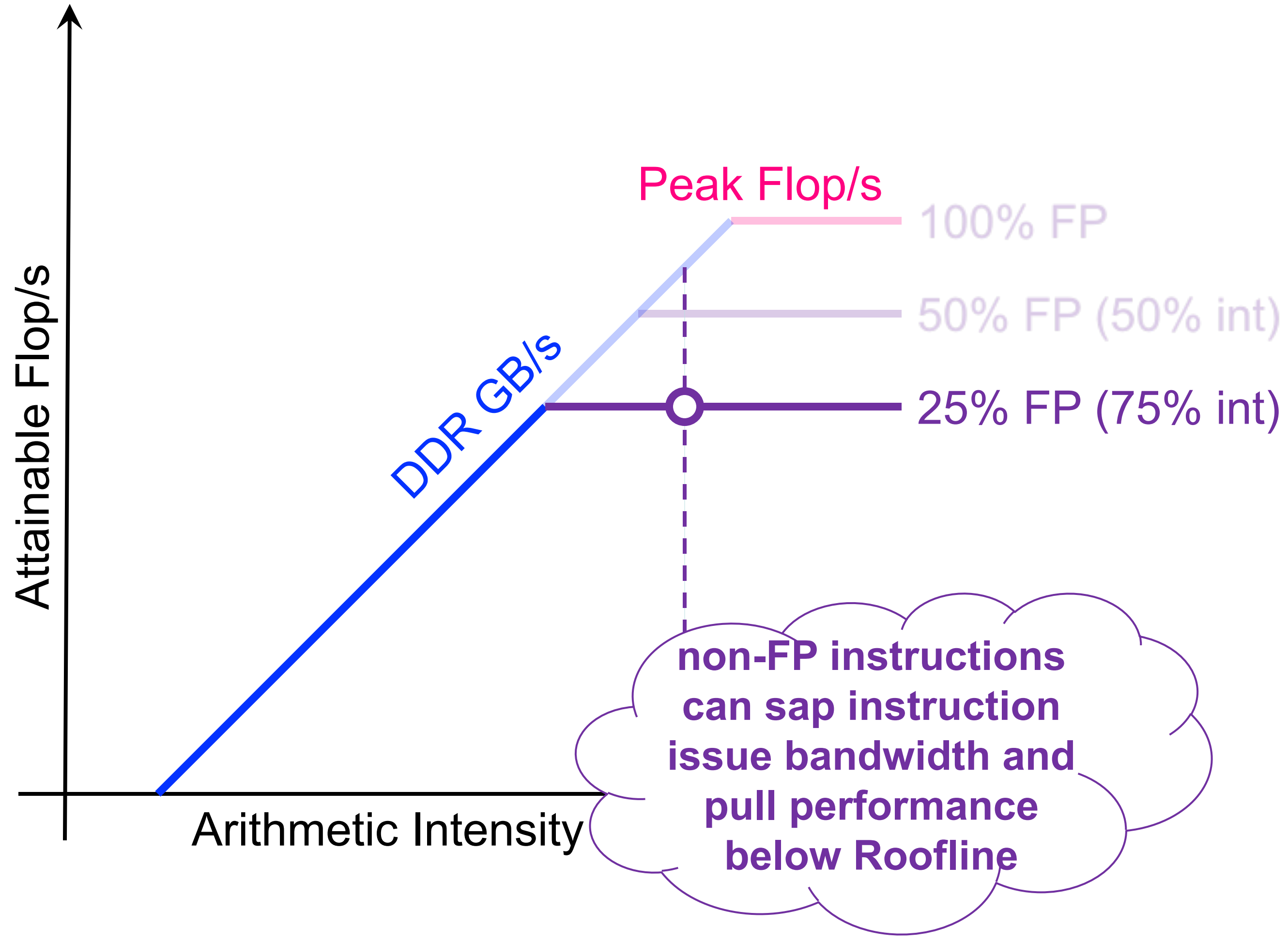
Superscalar vs. instruction mix

- Define in-core ceilings based on instruction mix…
- e.g. Haswell
  - 4-issue superscalar
  - Only 2 FP data paths
  - Requires 50% of the instructions to be FP to get peak performance
- e.g. KNL
  - 2-issue superscalar
  - 2 FP data paths
  - Requires 100% of the instructions to be FP to get peak performance



Superscalar vs. instruction mix

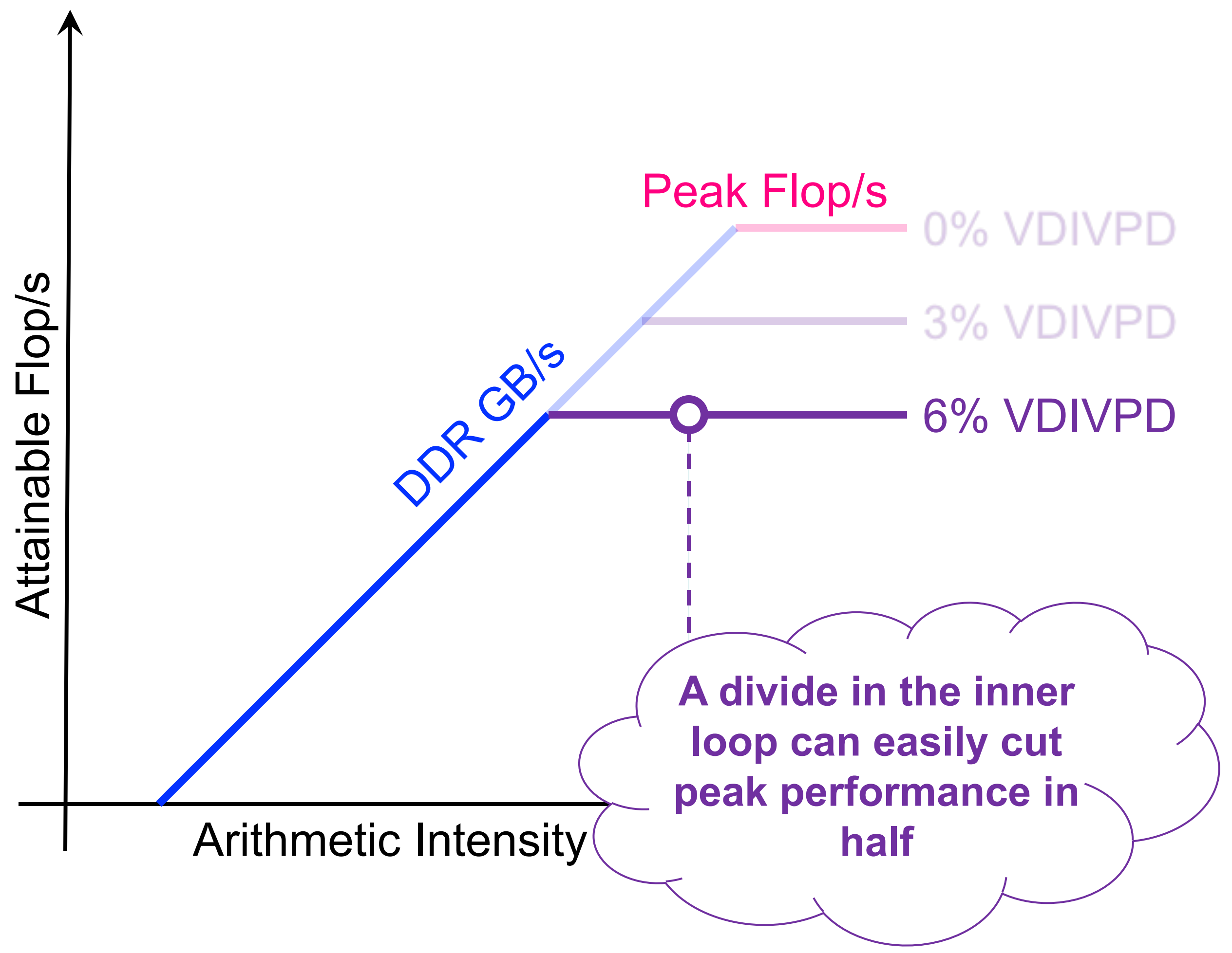
- Define in-core ceilings based on instruction mix…
- e.g. Haswell
  - 4-issue superscalar
  - Only 2 FP data paths
  - Requires 50% of the instructions to be FP to get peak performance
- e.g. KNL
  - 2-issue superscalar
  - 2 FP data paths
  - Requires 100% of the instructions to be FP to get peak performance

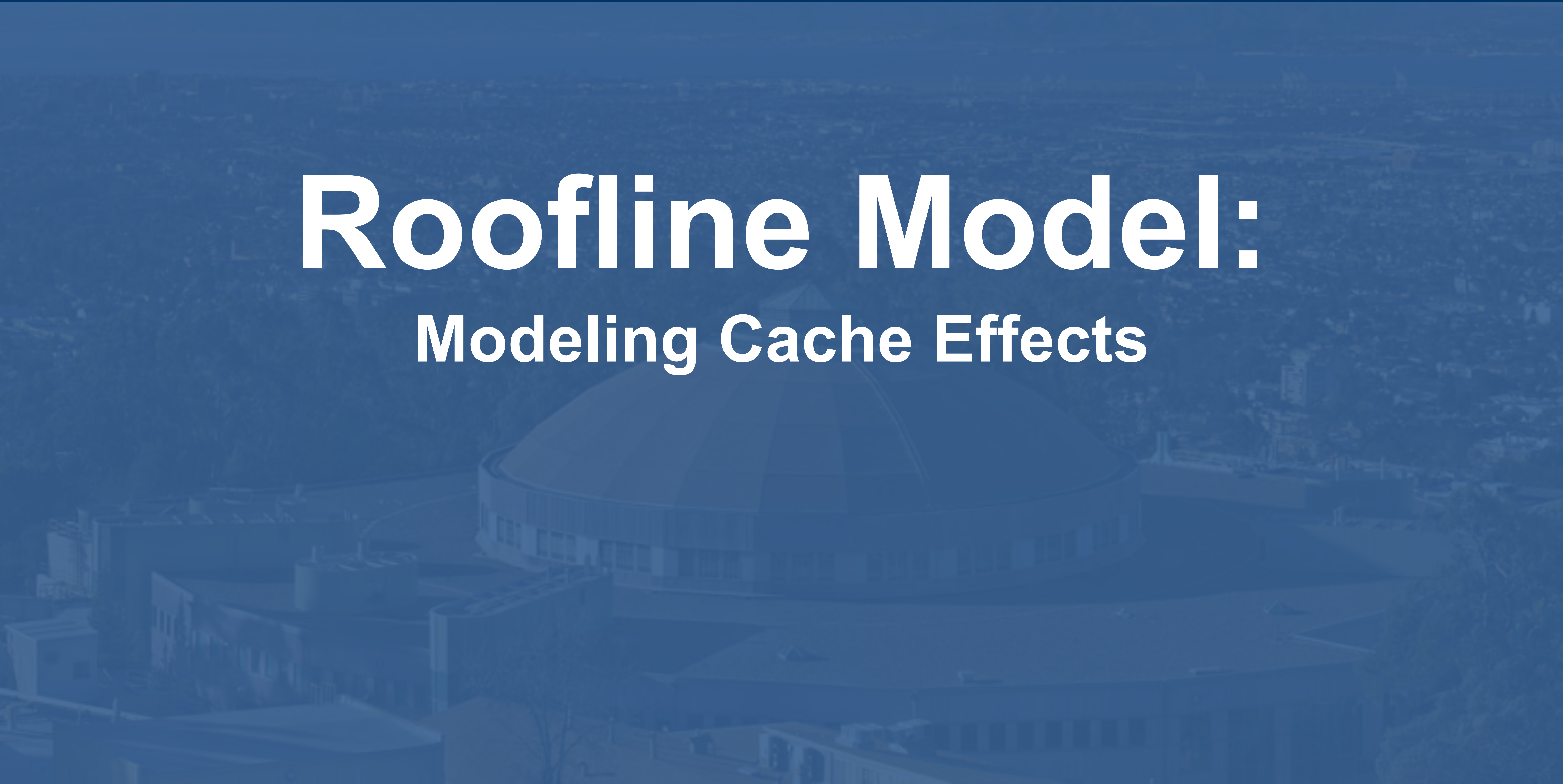


Divides and other Slow FP instructions

- FP Divides (sqrt, rsqrt, …) might support only limited pipelining
- As such, their throughput is substantially lower than FMA's
- If divides constitute even if 3% of the flop's come from divides, performance can be cut in half.

➢ ***Penalty varies substantially between architectures and generations (e.g. IVB, HSW, KNL, …)***



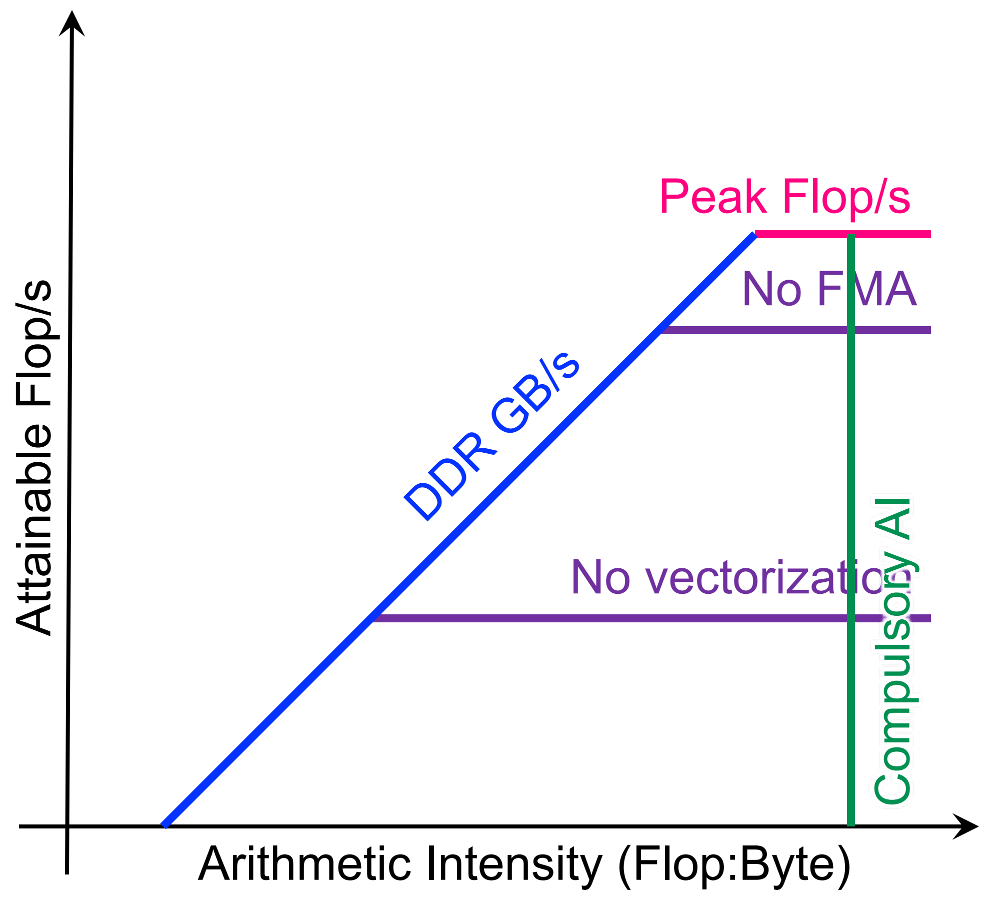


Roofline Model:

Modeling Cache Effects

Locality Walls

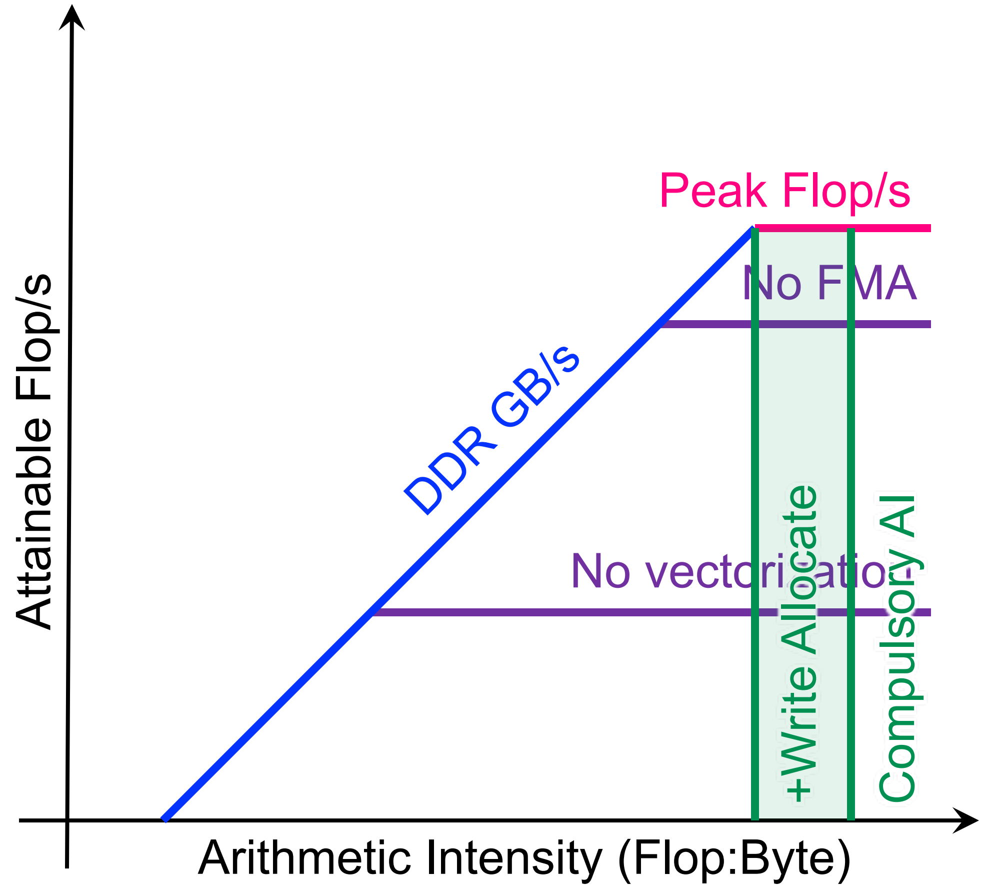
- Naively, we can bound AI using only compulsory cache misses



$$AI = \frac{\#Flop's}{Compulsory\ Misses}$$

Locality Walls

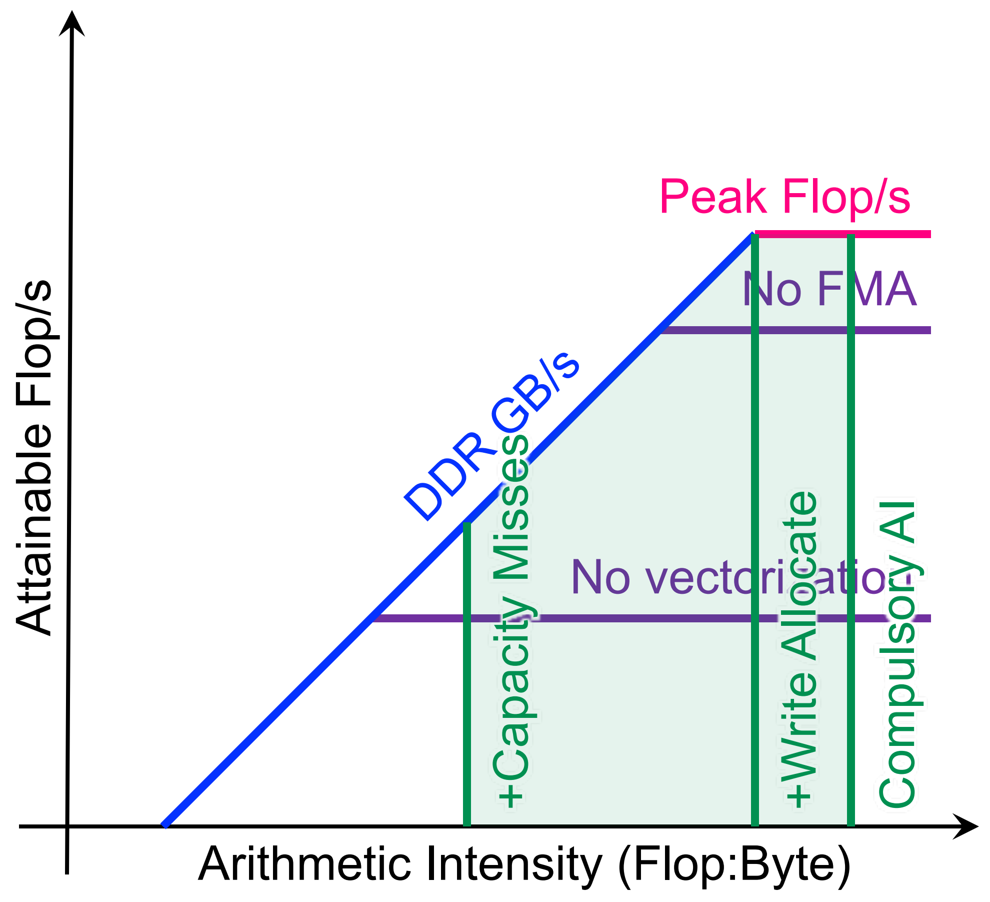
- Naively, we can bound AI using only compulsory cache misses
- However, write allocate caches can lower AI



$$AI = \frac{\#Flop's}{\text{Compulsory Misses + Write Allocates}}$$

Locality Walls

- Naively, we can bound AI using only compulsory cache misses
- However, write allocate caches can lower AI
- Cache capacity misses can have a huge penalty



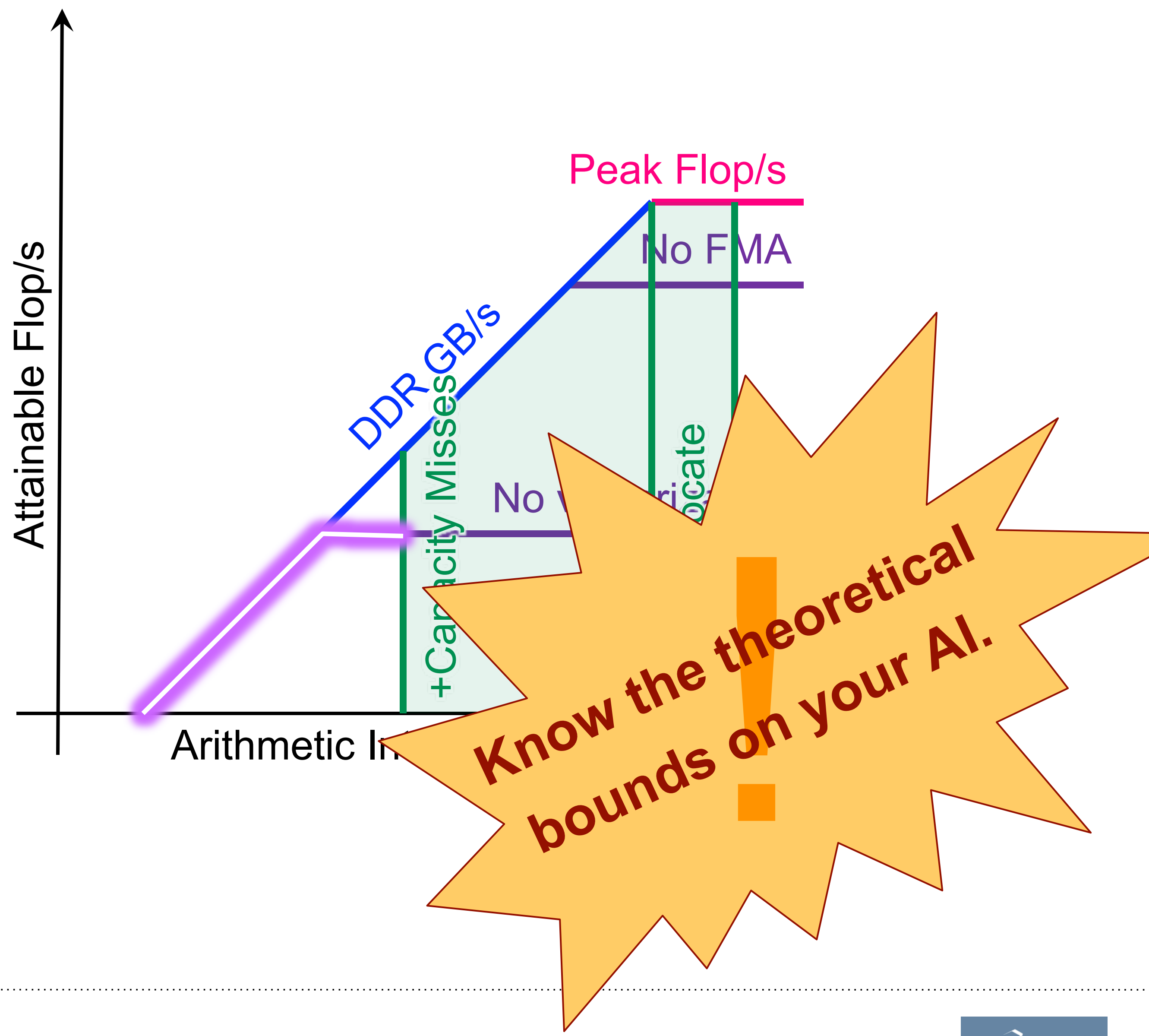
$$AI = \frac{\#Flop's}{\text{Compulsory Misses} + \text{Write Allocates} + \text{Capacity Misses}}$$

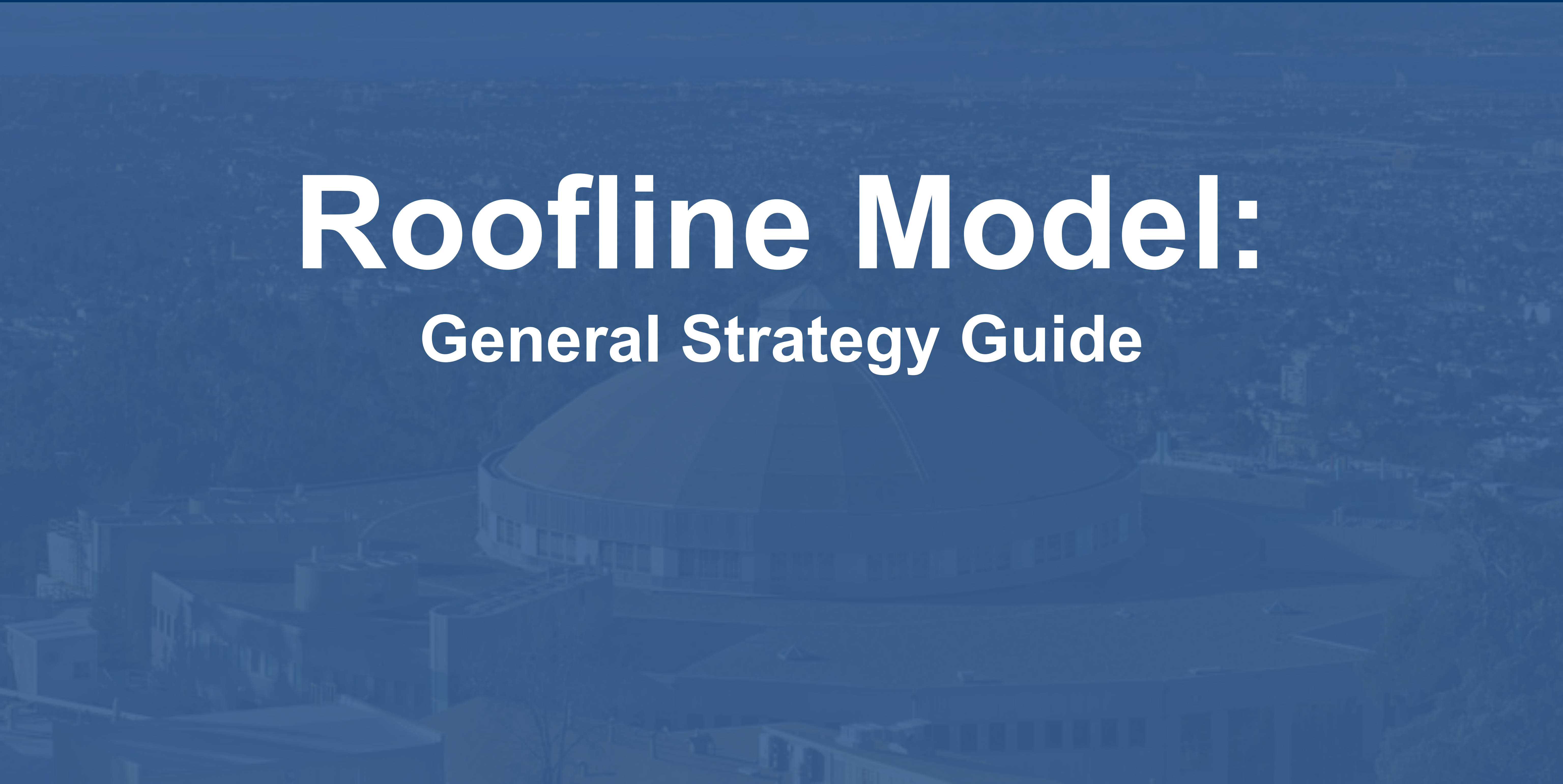
Locality Walls

- Naively, we can bound AI using only compulsory cache misses
- However, write allocate caches can lower AI
- Cache capacity misses can have a huge penalty

➢ **Compute bound became memory bound**

$$AI = \frac{\#Flop's}{\text{Compulsory Misses} + \text{Write Allocates} + \text{Capacity Misses}}$$



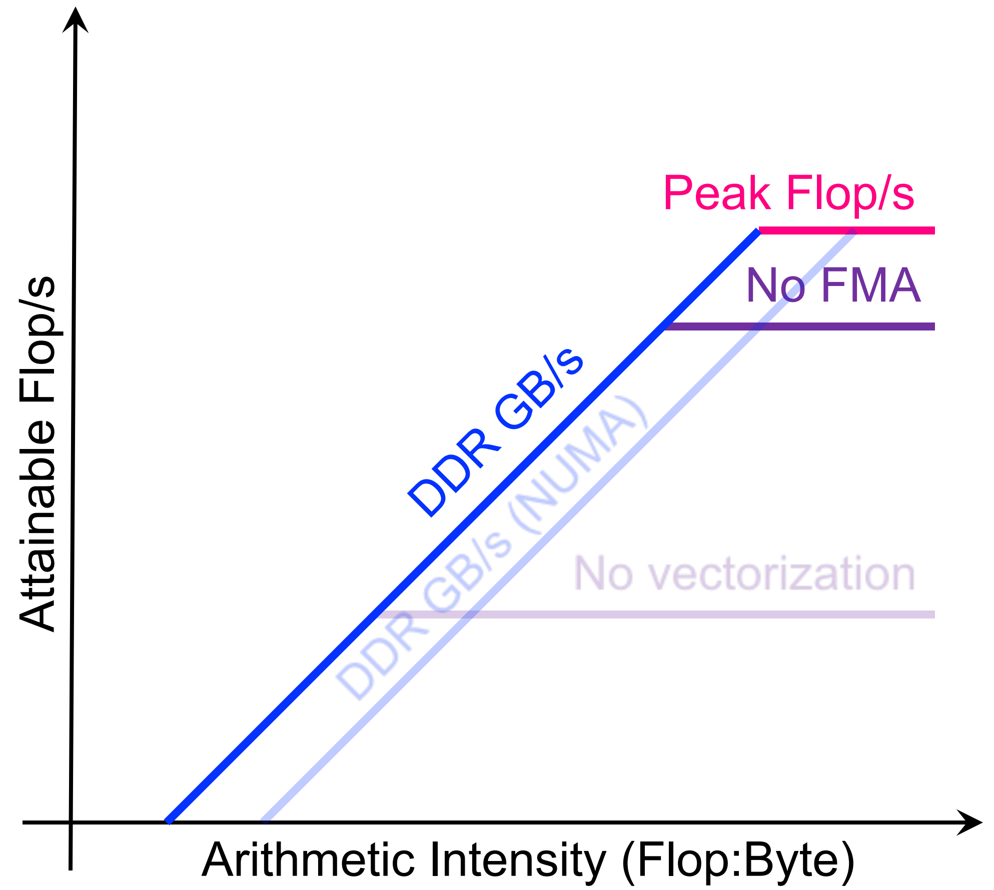


Roofline Model:

General Strategy Guide

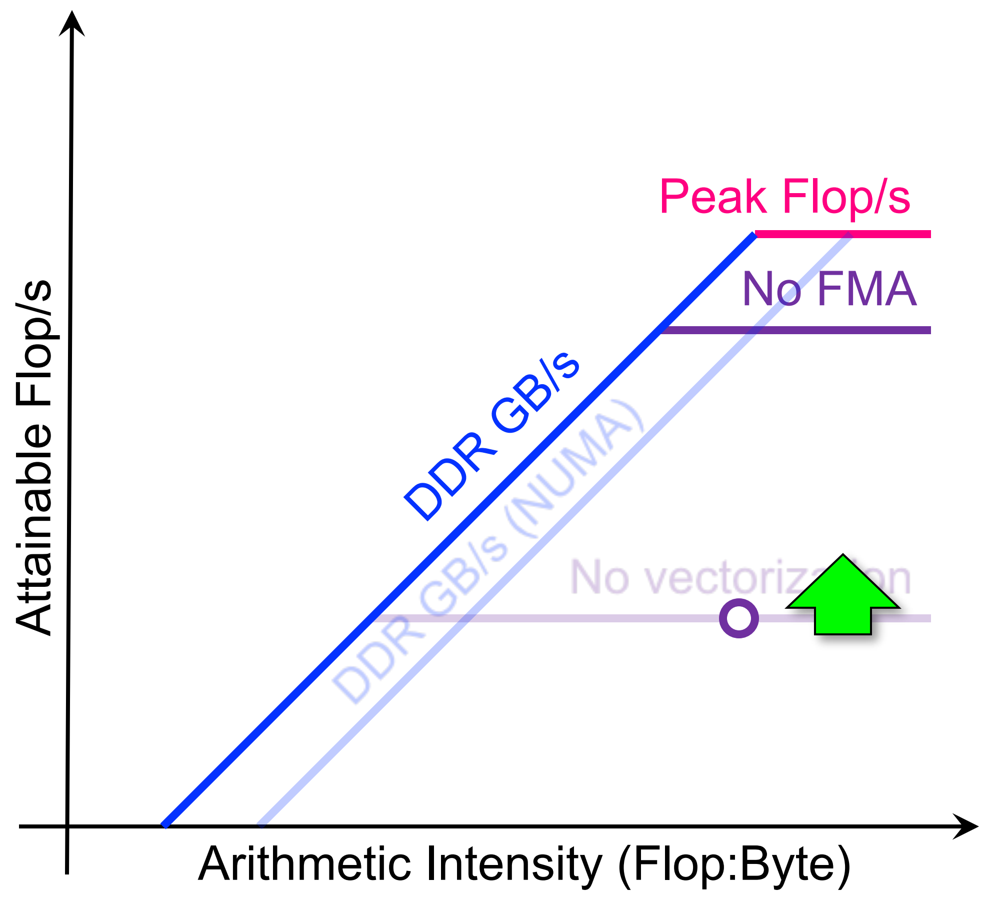
General Strategy Guide

- Broadly speaking, there are three approaches to improving performance:



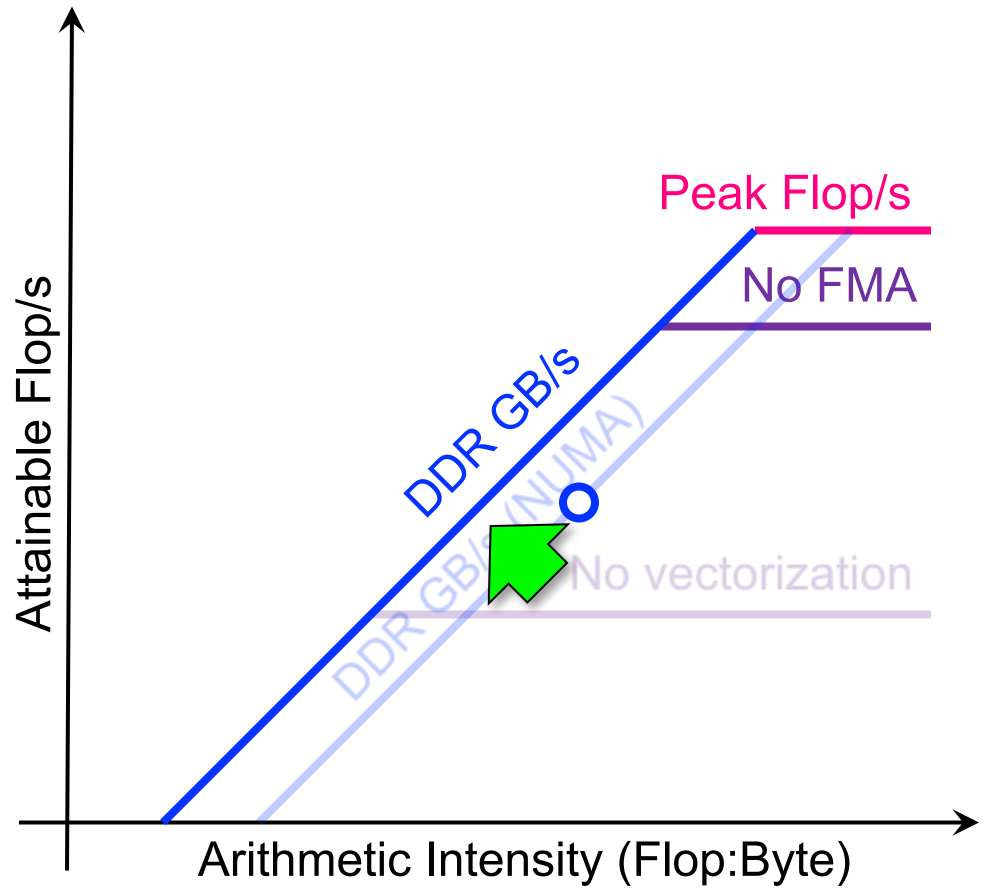
General Strategy Guide

- Broadly speaking, there are three approaches to improving performance:
- **Maximize in-core performance (e.g. get compiler to vectorize)**



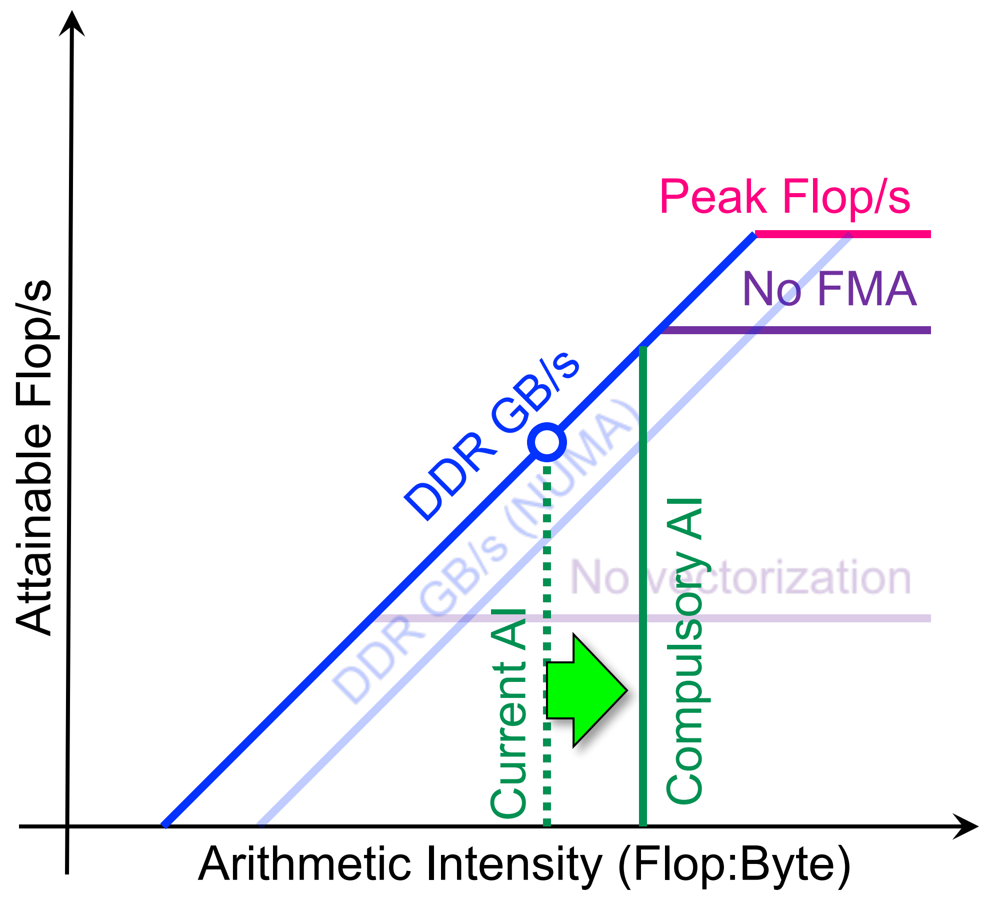
General Strategy Guide

- Broadly speaking, there are three approaches to improving performance:
- Maximize in-core performance (e.g. get compiler to vectorize)
- **Maximize memory bandwidth (e.g. NUMA-aware allocation)**



General Strategy Guide

- Broadly speaking, there are three approaches to improving performance:
- Maximize in-core performance (e.g. get compiler to vectorize)
- Maximize memory bandwidth (e.g. NUMA-aware allocation)
- **Minimize data movement (increase AI)**



Constructing a Roofline Model requires answering some questions…

Questions can overwhelm users…

What is my machine's peak flop/s?

How important is vectorization or FMA on my machine?

What is my machine's DDR GB/s?

L2 GB/s?

Properties of the target machine

(Benchmarking)

How much data did my kernel actually move?

Did my kernel vectorize?

How many flop's did my kernel actually do?

How much did that divide hurt?

Properties of an application's execution

(Instrumentation)

What is my kernel's compulsory AI? (communication lower bounds)

Can my kernel ever be vectorized?

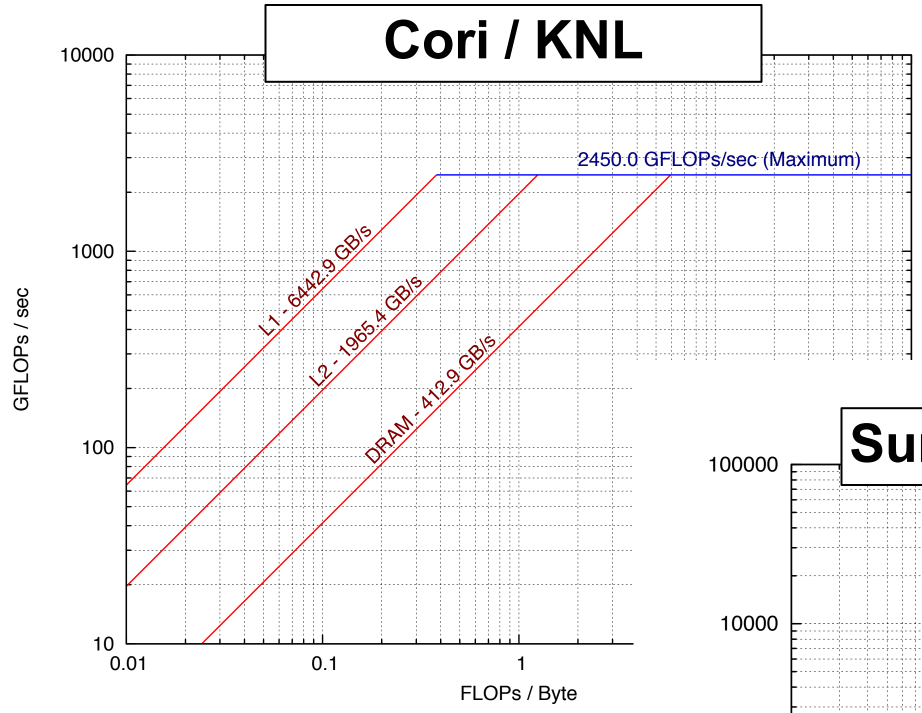
Fundamental properties of the kernel constrained by hardware

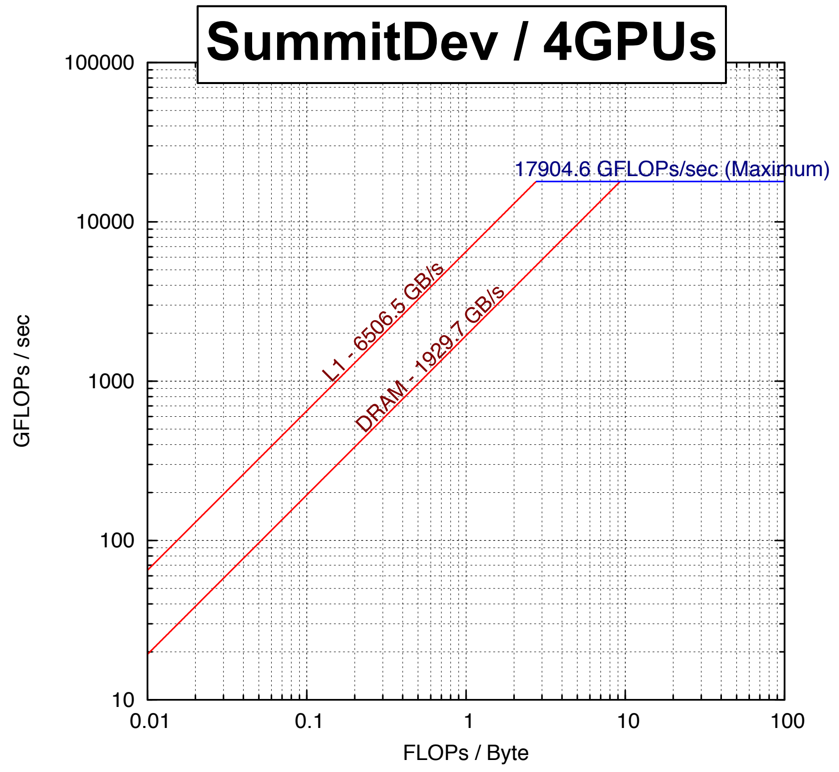
(Theory)

We need tools…

Node Characterization?

- **"Marketing Numbers"** can be deceptive…
  - Pin BW vs. real bandwidth
  - TurboMode / Underclock for AVX
  - compiler failings on high-AI loops.
- LBL developed the Empirical Roofline Toolkit (ERT)…
  - Characterize CPU/GPU systems
  - Peak Flop rates
  - Bandwidths for each level of memory
  - **MPI+OpenMP/CUDA == multiple GPUs**





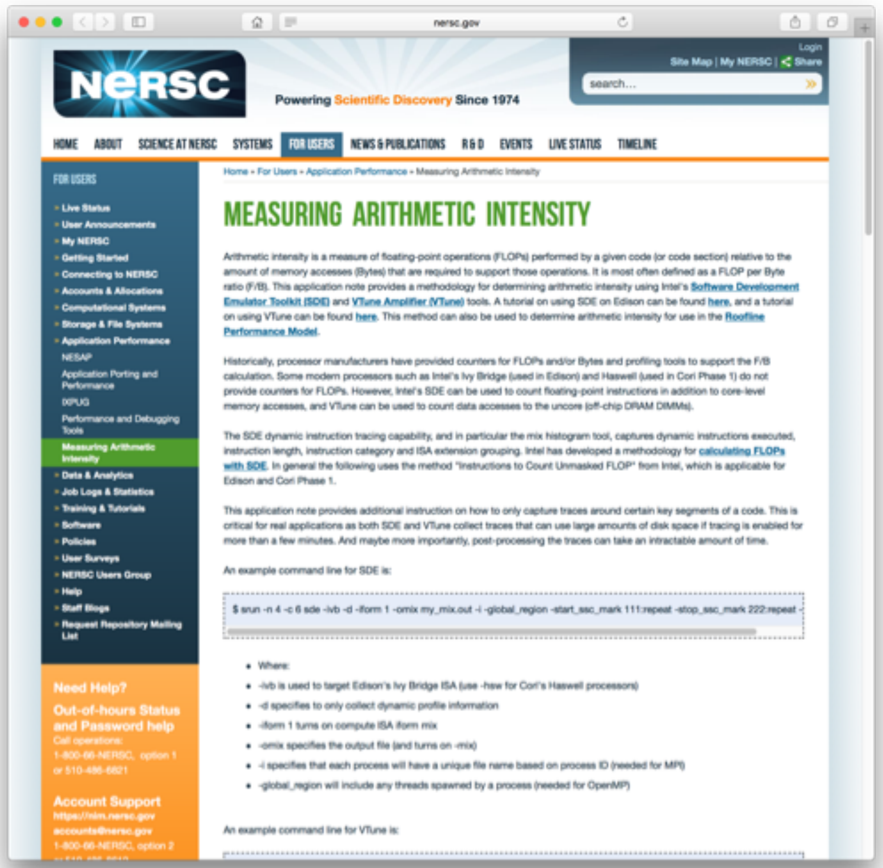
https://crd.lbl.gov/departments/computer-science/PAR/research/roofline/

Instrumentation with Performance Counters?

- ***Characterizing applications with performance counters can be problematic…***
  - x Flop Counters can be broken/missing in production processors
  - x Vectorization/Masking can complicate counting Flop's
  - x Counting Loads and Stores doesn't capture cache reuse while counting cache misses doesn't account for prefetchers.
  - x DRAM counters (Uncore PMU) might be accurate, but…
    - x are privileged and thus nominally inaccessible in user mode
    - x may need vendor (e.g. Cray) and center (e.g. NERSC) approved OS/kernel changes

Forced to Cobble Together Tools…

- Use tools known/observed to work on NERSC's Cori (KNL, HSW)…
  - Used **Intel SDE** (Pin binary instrumentation + emulation) to create software Flop counters
  - Used **Intel VTune** performance tool (NERSC/Cray approved) to access uncore counters
- ➢ Accurate measurement of Flop's (HSW) and DRAM data movement (HSW and KNL)
- ➢ Used by NESAP (NERSC KNL application readiness project) to characterize apps on Cori…



Home » For Users » Application Performance » Measuring Arithmetic Intensity

MEASURING ARITHMETIC INTENSITY

Arithmetic intensity is a measure of floating-point operations (FLOPs) performed by a given code (or code section) relative to the amount of memory accesses (Bytes) that are required to support those operations. It is most often defined as a FLOP per Byte ratio (F/B). This application note provides a methodology for determining arithmetic intensity using Intel's Software Development Emulator Toolkit (SDE) and VTune Amplifier (VTune) tools. A tutorial on using SDE on Edison can be found here, and a tutorial on using VTune can be found here. This method can also be used to determine arithmetic intensity for use in the Roofline Performance Model.

Historically, processor manufacturers have provided counters for FLOPs and/or Bytes and profiling tools to support the F/B calculation. Some modern processors such as Intel's Ivy Bridge (used in Edison) and Haswell (used in Cori Phase 1) do not provide counters for FLOPs. However, Intel's SDE can be used to count floating-point instructions in addition to core-level memory accesses, and VTune can be used to count data accesses to the uncore (off-chip DRAM DIMMs).

The SDE dynamic instruction tracing capability, and in particular the mix histogram tool, captures dynamic instructions executed, instruction length, instruction category and ISA extension grouping. Intel has developed a methodology for calculating FLOPs with SDE. In general the following uses the method "Instructions to Count Unmasked FLOP" from Intel, which is applicable for Edison and Cori Phase 1.

This application note provides additional instruction on how to only capture traces around certain key segments of a code. This is critical for real applications as both SDE and VTune collect traces that can use large amounts of disk space if tracing is enabled for more than a few minutes. And maybe more importantly, post-processing the traces can take an intractable amount of time.

An example command line for SDE is:

```
$ srun -n 4 -c 6 sde -ivb -d -iform 1 -omix my_mix.out -i -global_region -start_ssc_mark 111:repeat -stop_ssc_mark 222:repeat -
```

- Where:
- -ivb is used to target Edison's Ivy Bridge ISA (use -hsw for Cori's Haswell processors)
- -d specifies to only collect dynamic profile information
- -iform 1 turns on compute ISA iform mix
- -omix specifies the output file (and turns on -mix)
- -i specifies that each process will have a unique file name based on process ID (needed for MPI)
- -global_region will include any threads spawned by a process (needed for OpenMP)

An example command line for VTune is:

http://www.nersc.gov/users/application-performance/measuring-arithmetic-intensity/

NERSC is LBL's production computing division
CRD is LBL's Computational Research Division
NESAP is NERSC's KNL application readiness project
LBL is part of SUPER (DOE SciDAC3 Computer Science Institute)

BERKELEY LAB

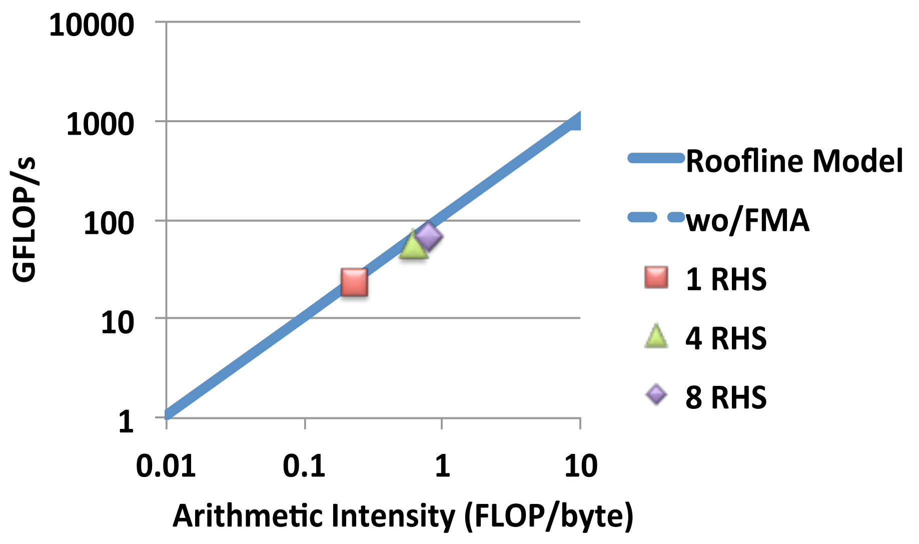
Initial Roofline Analysis of NESAP Codes

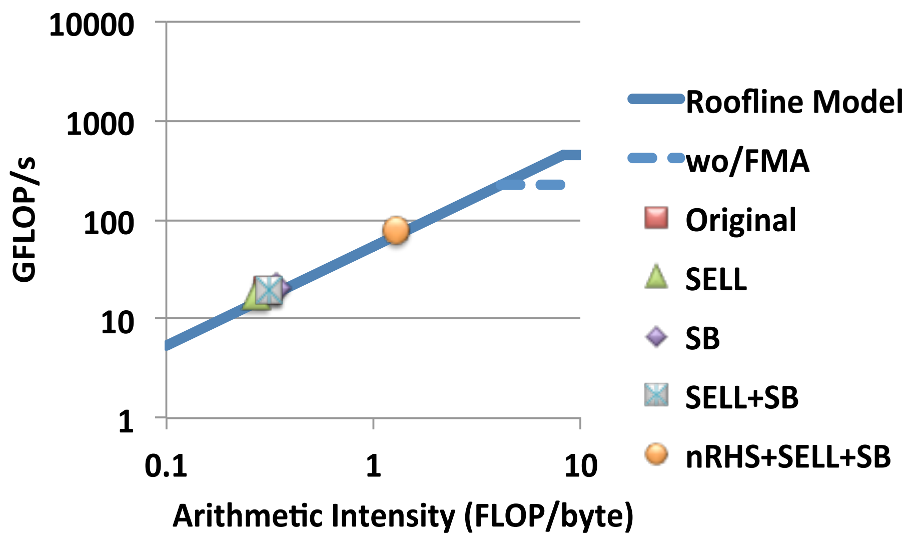
MFDn

EMGeo

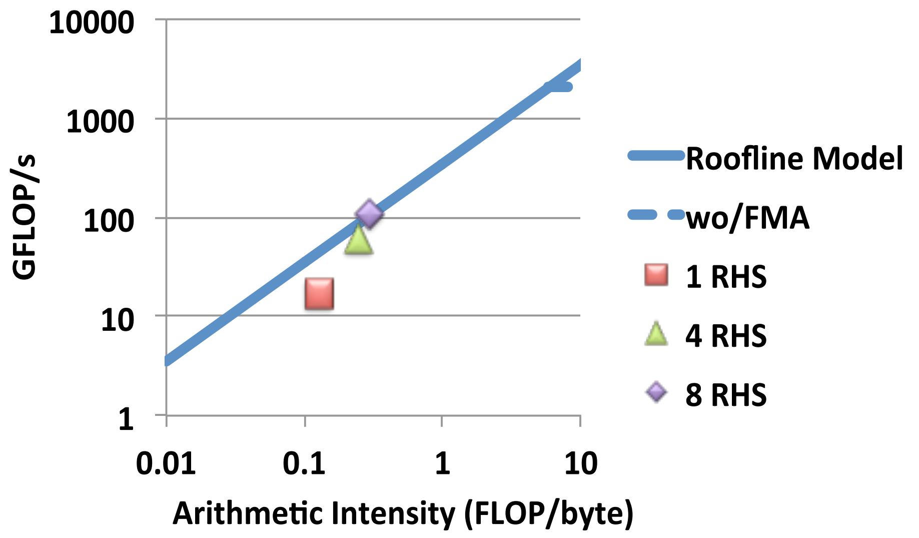
PICSAR

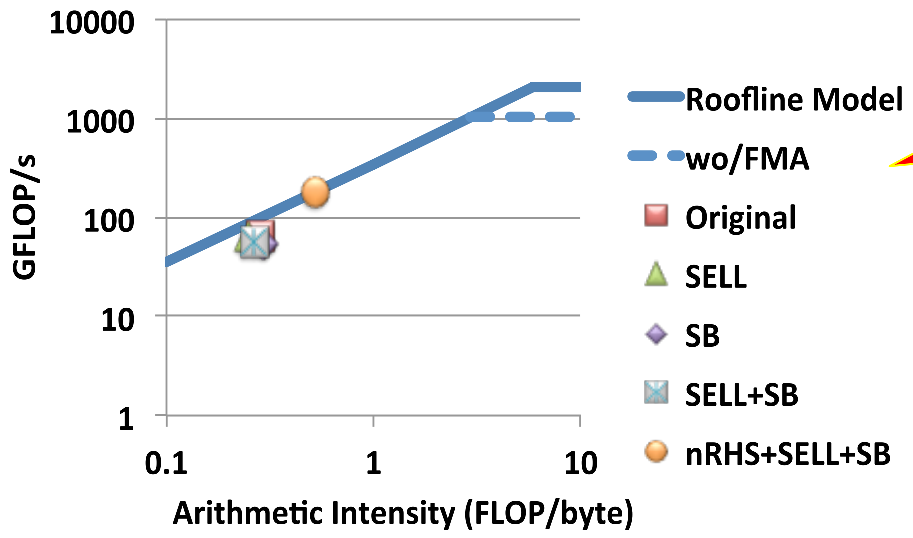
2P HSW

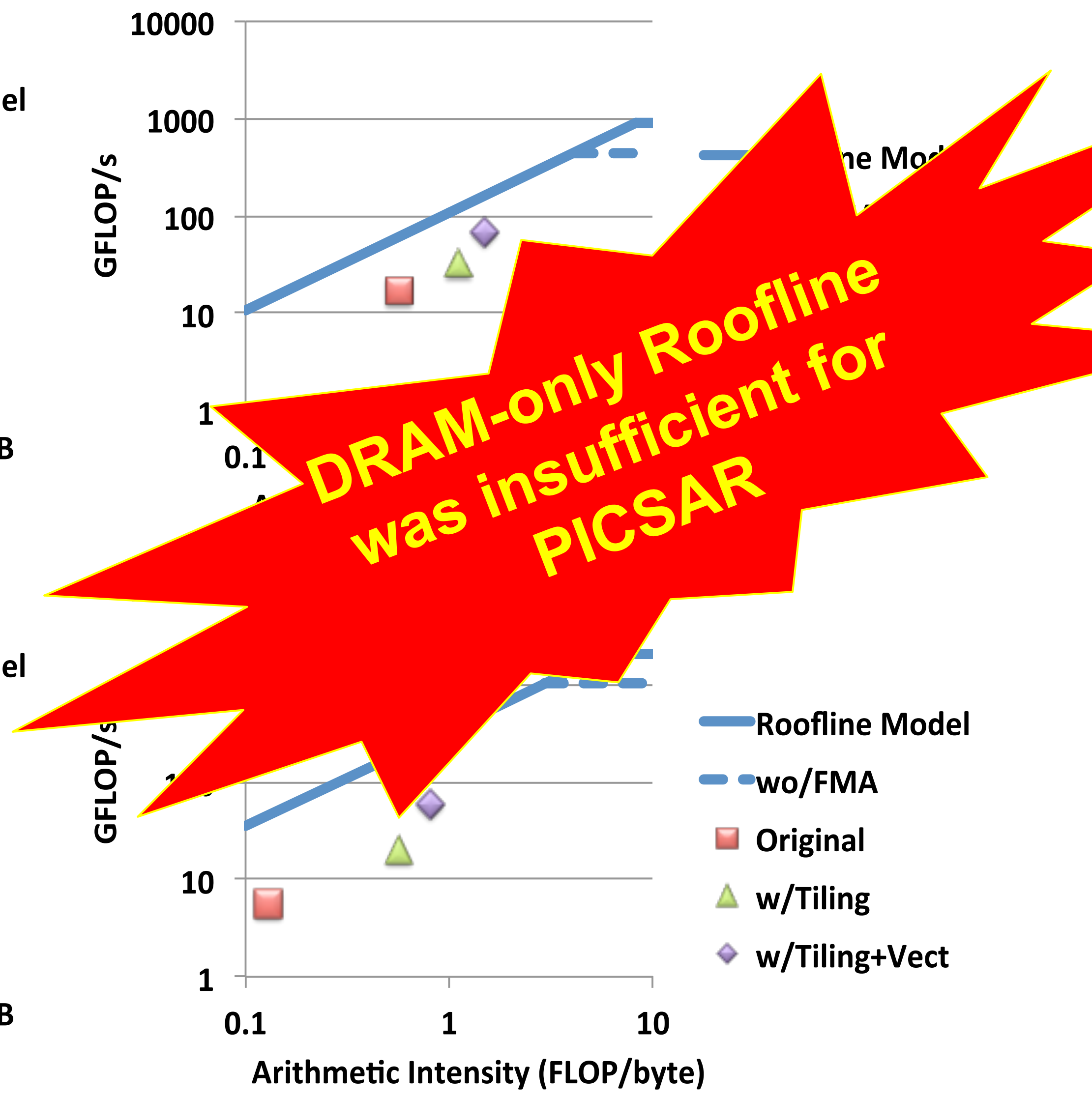




KNL





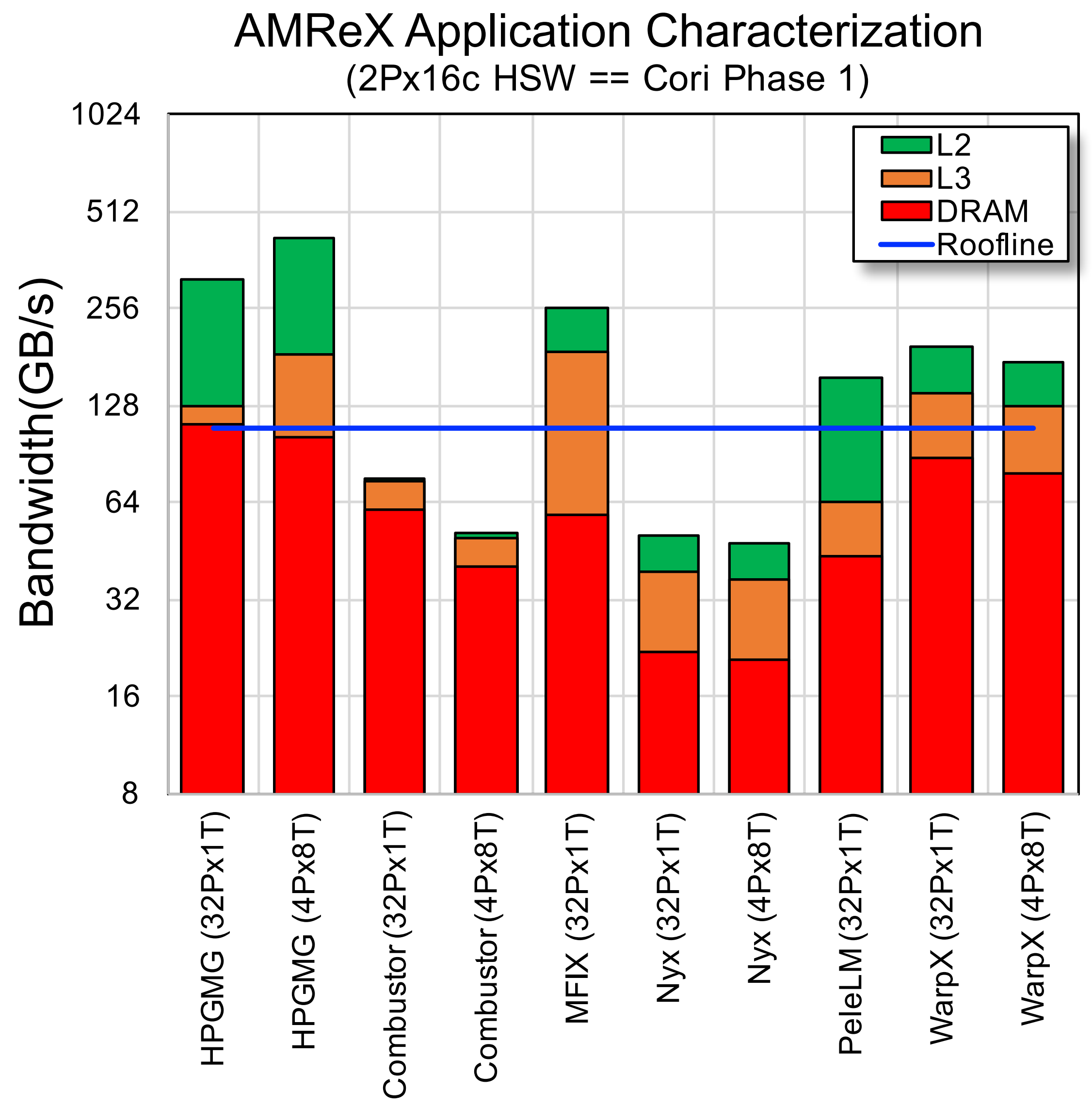


DRAM-only Roofline was insufficient for PICSAR

Evaluation of LIKWID

- LIKWID provides easy to use wrappers for measuring performance counters…
  - ✓ **Works on NERSC production systems**
  - ✓ Minimal overhead (<1%)
  - ✓ Scalable in distributed memory (MPI-friendly)
  - ✓ Fast, high-level characterization
  - x **No detailed timing breakdown or optimization advice**
  - x **Limited by quality of hardware performance counter implementation (garbage in/garbage out)**

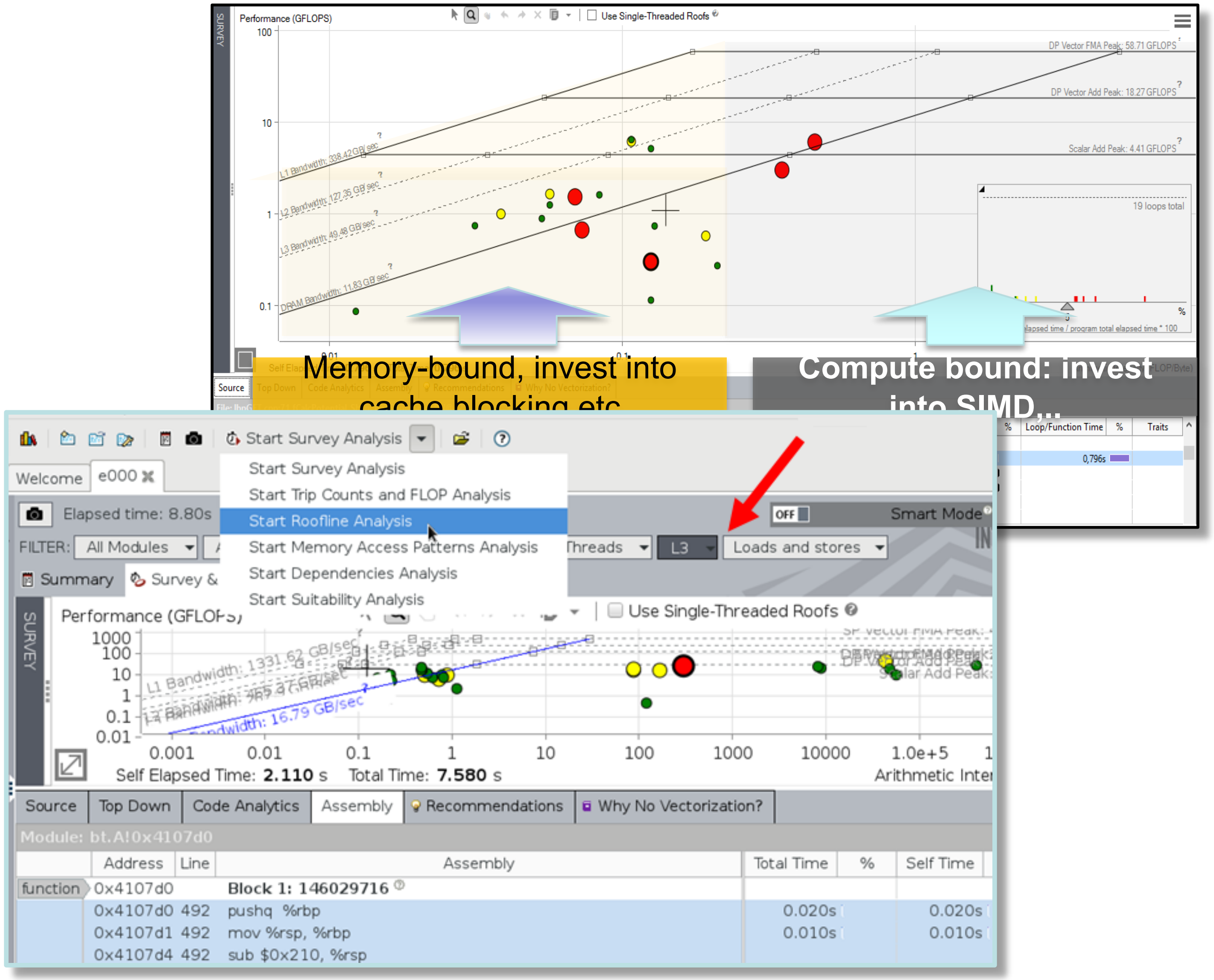
➢ **Useful tool that complements other tools**



https://github.com/RRZE-HPC/likwid

Intel Advisor

- Includes Roofline Automation…
  - ✓ Automatically instruments applications (one dot per loop nest/function)
  - ✓ Computes FLOPS and AI for each function (**CARM**)
  - ✓ AVX-512 support that incorporates masks
  - ✓ **Integrated Cache Simulator[1] (hierarchical roofline / multiple AI's)**
  - ✓ Automatically benchmarks target system (calculates ceilings)
  - ✓ Full integration with existing Advisor capabilities


Performance (GFLOPS)
Use Single-Threaded Roofs
DP Vector FMA Peak: 58.71 GFLOPS
DP Vector Add Peak: 18.27 GFLOPS
Scalar Add Peak: 4.41 GFLOPS
L1 Bandwidth: 208.42 GB/sec
L2 Bandwidth: 127.35 GB/sec
L3 Bandwidth: 49.48 GB/sec
DRAM Bandwidth: 11.83 GB/sec
19 loops total
Memory-bound, invest into cache blocking etc.
Compute bound: invest into SIMD...
Start Survey Analysis
Welcome
e000
Start Survey Analysis
Start Trip Counts and FLOP Analysis
Start Roofline Analysis
Start Memory Access Patterns Analysis
Start Dependencies Analysis
Start Suitability Analysis
Elapsed time: 8.80s
OFF
Smart Mode
FILTER: All Modules
Threads
L3
Loads and stores
Summary
Survey &
SURVEY
Performance (GFLOPS)
L1 Bandwidth: 1331.62 GB/sec
L3 Bandwidth: 365.47 GB/sec
Bandwidth: 16.79 GB/sec
1000
100
10
1
0.1
0.01
0.001
0.01
0.1
1
10
100
1000
10000
1.0e+5
Arithmetic Inte
Self Elapsed Time: 2.110 s
Total Time: 7.580 s
Source
Top Down
Code Analytics
Assembly
Recommendations
Why No Vectorization?
Module: bt.A!0x4107d0

	Address	Line	Assembly	Total Time	%	Self Time
function	0x4107d0		Block 1: 146029716			
	0x4107d0	492	pushq %rbp	0.020s		0.020s
	0x4107d1	492	mov %rsp, %rbp	0.010s		0.010s
	0x4107d4	492	sub $0x210, %rsp			

http://www.nersc.gov/users/training/events/roofline-training-1182017-1192017

[1]Technology Preview, not in official product roadmap so far.

BERKELEY LAB

Tools and Platforms for Roofline Modeling

Use LIKWID for fast, scalable app-level instrumentation

	Metric	STREAM	Intel SDE	Intel Advisor	NVIDIA NVProf
Benchmark	Peak MFlops	✗	✗	✓	✗
	Pe	✗	✗	✗	[illegible]
	[illegible]	✓	✗	✓	[illegible]
	[illegible]	✗	✗	[illegible]	[illegible]
Execution	[illegible]	✗	✓	✓	[illegible]
	%SI	✗	✓	✓	[illegible]
	MIPS	✗	✓	✗	[illegible]
	DRAM BW	✗	✗	✓	✓
	Cache BW	✗	✗	✓	✓
	Auto-Roofline	✗	✗	✓	✗
Platforms	Intel CPUs	✓	✓	✓	✗
	IBM Power8	✓	✗	✗	✗
	NVIDIA GPUs	✓	✗	✗	✓
	AMD CPUs	✓	?	?	✗
	AMD GPUs	✓	✗	✗	✗
	ARM	✓	✗	✗	✗

Use ERT to benchmark systems

Use Advisor for loop-level instrumentation and analysis on Intel targets

Questions?

Backup

Complexity, Depth, …

Why Use Performance Models or Tools?

- Identify performance bottlenecks
- Motivate software optimizations
- **Determine when we're done optimizing**
  - Assess performance relative to machine capabilities
  - Motivate need for algorithmic changes
- Predict performance on future machines / architectures
  - Sets realistic expectations on performance for future procurements
  - Used for HW/SW Co-Design to ensure future architectures are well-suited for the computational needs of today's applications.

Computational Complexity

- Assume run time is correlated with the number of operations (e.g. FP ops)
- Users define parameterize their algorithms, solvers, kernels
- Count the number of operations as a function of those parameters
- Demonstrate run time is correlated with those parameters

```
#pragma omp parallel for
for(i=0;i<N;i++){
  z[i] = alpha*x
}
```

DAX
N

```
#pragma omp parallel for
for(i=0;i<N;i++){
for(j=0;j<N;j++){
  double Cij=0;
    r(k=0;k<N;k++){
     ij += A[i][k] * B[k][j];

   ][j] = sum;
```

What are the scaling constants?

GEMM: O(N^3) complexity where N is the number of rows (equati

FFTs: O(NlogN) in the number of

CG: O($N^{1.33}$) in the number of

MG: O(N) in the number of ele

N-body: O(N^2) in the number o

Why did we depart from ideal scaling?

Data Movement Complexity

- Assume run time is correlated with the amount of data accessed (or moved)
- Easy to calculate amount of data accessed… count array accesses
- Data moved is more complex as it requires understanding cache behavior…
  - Compulsory[1] data movement (array sizes) is a good initial guess…
  - … but needs refinement for the effects of finite cache capacities

Operation	Flop's	Data
DAXPY	O(N)	O(N)
DGEMV	$O(N^2)$	$O(N^2)$
DGEMM	$O(N^3)$	$O(N^2)$
FFTs	O(NlogN)	O(N)
CG	$O(N^{1.33})$	
MG		
N-body		

Which is more expensive…

Performing Flop's, or

Moving words from memory

[1]Hill et al, "Evaluating Associativity in CPU Caches", IEEE Trans. Comput., 1989.

Machine Balance and Arithmetic Intensity

- Data movement and computation can operate at different rates
- We define machine balance as the ratio of…

Balance = $\frac{\textbf{Peak DP Flop/s}}{\textbf{Peak Bandwidth}}$

- …and arithmetic intensity as the ratio of…

AI = $\frac{\textbf{Flop's Performed}}{\textbf{Data Moved}}$

Operation	Flop's	Data	AI (ideal)
DAXPY	O(N)	O(N)	O(1)
DGEMV	O(N^2)	O(N^2	O(1)
DGEMM	[illegible]	[illegible]	O(N)
FFTs	O(N	[illegible]	O(logN)
CG	O(N	[illegible]	[illegible]
MG	[illegible]	[illegible]	O(1)
N-body	[illegible]	[illegible]	O(N)

Kernels with AI greater than machine balance are ultimately compute limited

Distributed Memory Performance Modeling

- In distributed memory, one communicates by sending messages between processors.
- Messaging time can be constrained by several components…
  - Overhead (CPU time to send/receive a message)
  - Latency (time message is in the network; can be hidden)
  - Message throughput (rate at which one can send small messages… messages/second)
  - Bandwidth (rate one can send large messages… GBytes/s)
- Bandwidths and latencies are further constrained by the interplay of network architecture and contention
- Distributed memory versions of our algorithms can be differently stressed by these components depending on N and P (#processors)

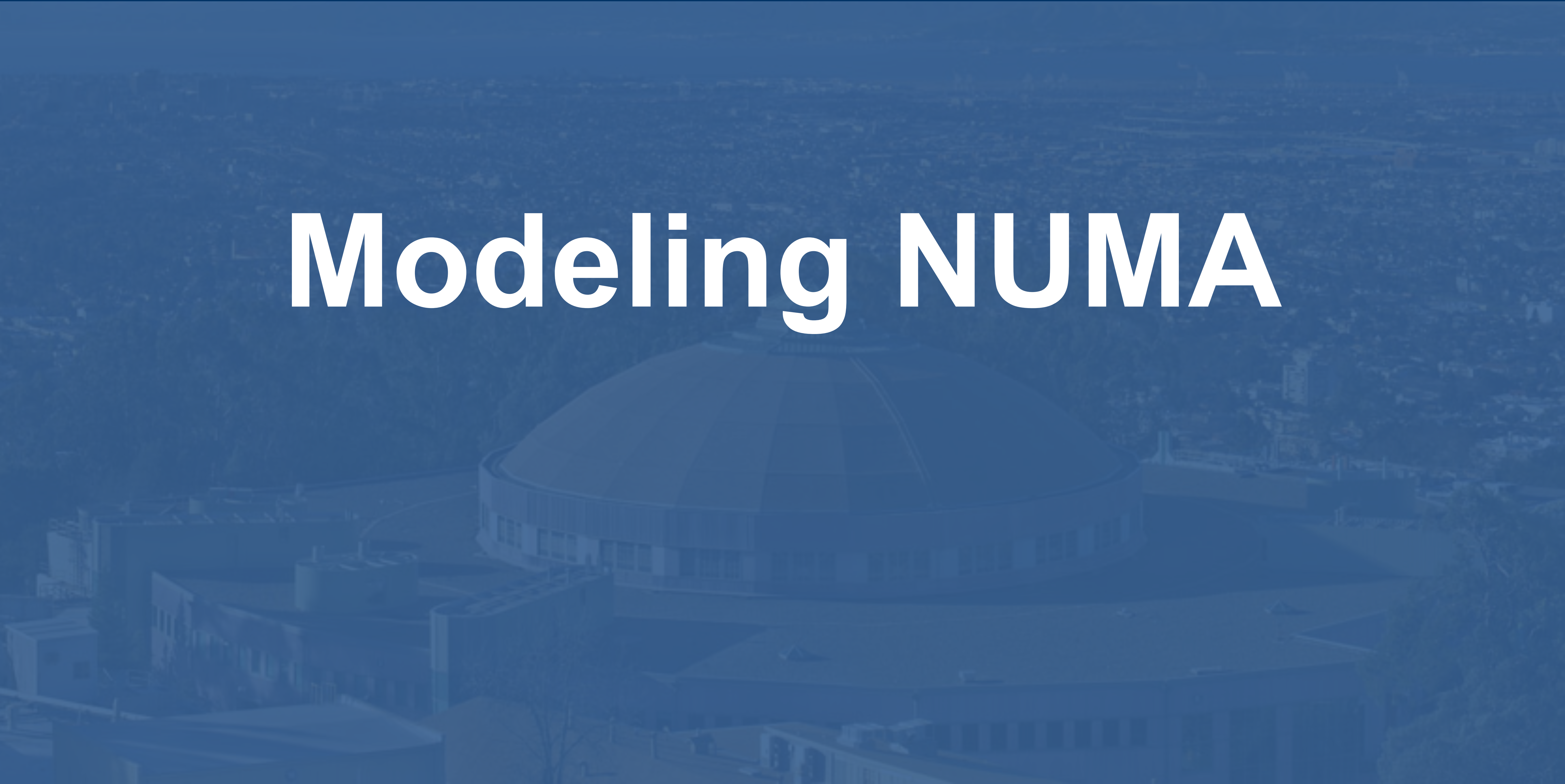
Computational Depth

- Parallel machines incur substantial overheads on synchronization (shared memory), point-to-point communication, reductions, and broadcasts.
- We can classify algorithms by **depth** (max depth of the algorithm's dependency chain)

➢ **If dependency chain crosses process boundaries, we incur substantial overheads.**

Operation	Flop's	Data	AI (ideal)	Depth
DAXPY	O(N)	O(N)	[illegible]	O(1)
DGEMV	$O(N^2)$	[illegible]	[illegible]	O(logN)
DGEMM	[illegible]	[illegible]	[illegible]	O(logN)
FFTs	[illegible]	[illegible]	[illegible]	O(logN)
CG	[illegible]	[illegible]	[illegible]	[illegible]
MG	[illegible]	[illegible]	[illegible]	O(logN)
N-body	[illegible]	[illegible]	[illegible]	O(logN)

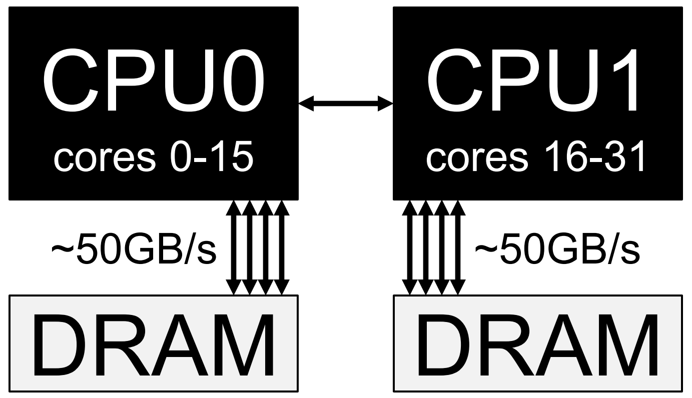
Overheads can dominate at high concurrency or small problems

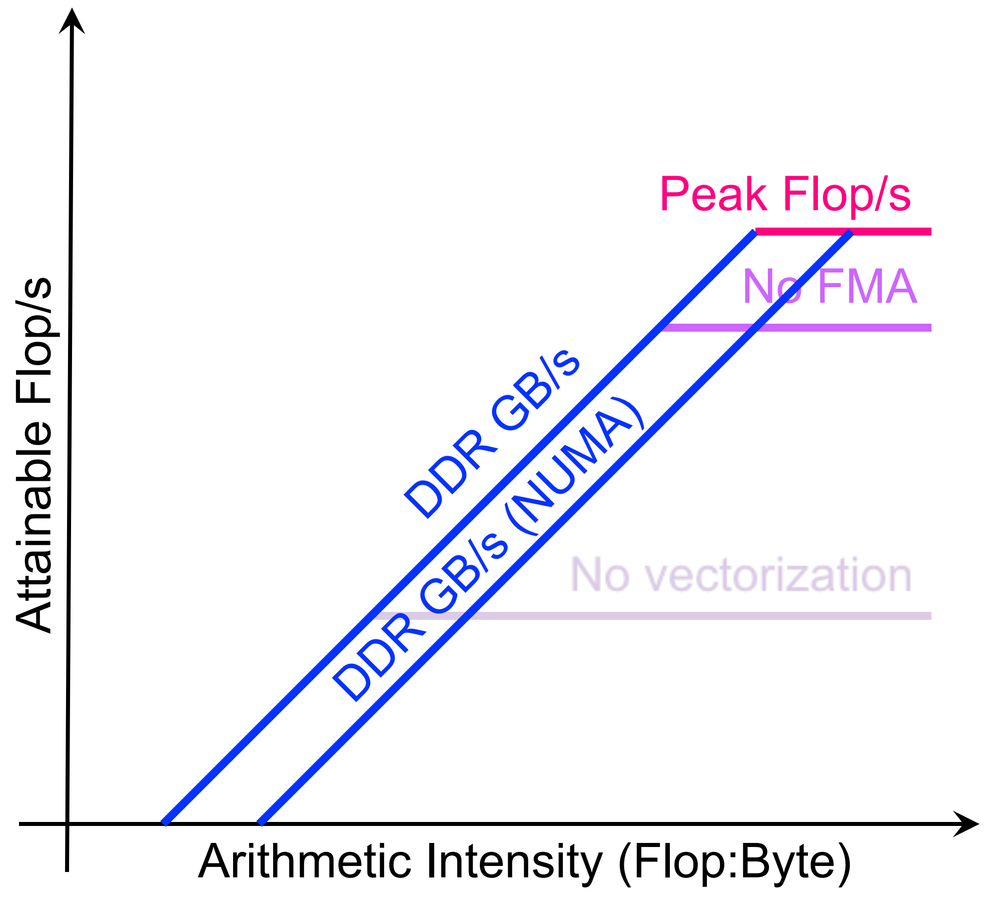


Modeling NUMA

NUMA Effects

- Cori’s Haswell nodes are built from 2 Xeon processors (sockets)
  - Memory attached to each socket (fast)
  - Interconnect that allows remote memory access (slow == NUMA)
  - Improper memory allocation can result in more than a 2x performance penalty





Hierarchical Roofline vs. Cache-Aware Roofline

…understanding different Roofline formulations in Advisor

There are two Major Roofline Formulations:

- Hierarchical Roofline (original Roofline w/ DRAM, L3, L2, …)…
  - **Williams, et al, “Roofline: An Insightful Visual Performance Model for Multicore Architectures”, CACM, 2009**
  - **Chapter 4 of “Auto-tuning Performance on Multicore Computers”, 2008**
  - Defines multiple bandwidth ceilings and multiple AI’s per kernel
  - Performance bound is the minimum of flops and the memory intercepts (superposition of original, single-metric Rooflines)
- Cache-Aware Roofline
  - **Ilic et al, "Cache-aware Roofline model: Upgrading the loft", IEEE Computer Architecture Letters, 2014**
  - Defines multiple bandwidth ceilings, but uses a single AI (flop:L1 bytes)
  - As one looses cache locality (capacity, conflict, …) performance falls from one BW ceiling to a lower one at constant AI
- Why Does this matter?
  - Some tools use the Hierarchical Roofline, some use cache-aware **== Users need to understand the differences**
  - Cache-Aware Roofline model was integrated into production Intel Advisor
  - Evaluation version of Hierarchical Roofline[1] (cache simulator) has also been integrated into Intel Advisor

[1]Technology Preview, not in official product roadmap so far.

Hierarchical Roofline

- Captures cache effects
- AI is Flop:Bytes after being ***filtered by lower cache levels***
- Multiple Arithmetic Intensities (one per level of memory)
- AI ***dependent*** on problem size (capacity misses reduce AI)
- Memory/Cache/Locality effects are ***observed as decreased AI***
- Requires ***performance counters or cache simulator*** to correctly measure AI

Cache-Aware Roofline

- Captures cache effects
- AI is Flop:Bytes ***as presented to the L1 cache (plus non-temporal stores)***
- Single Arithmetic Intensity
- AI ***independent*** of problem size
- Memory/Cache/Locality effects are ***observed as decreased performance***
- Requires static analysis or ***binary instrumentation*** to measure AI

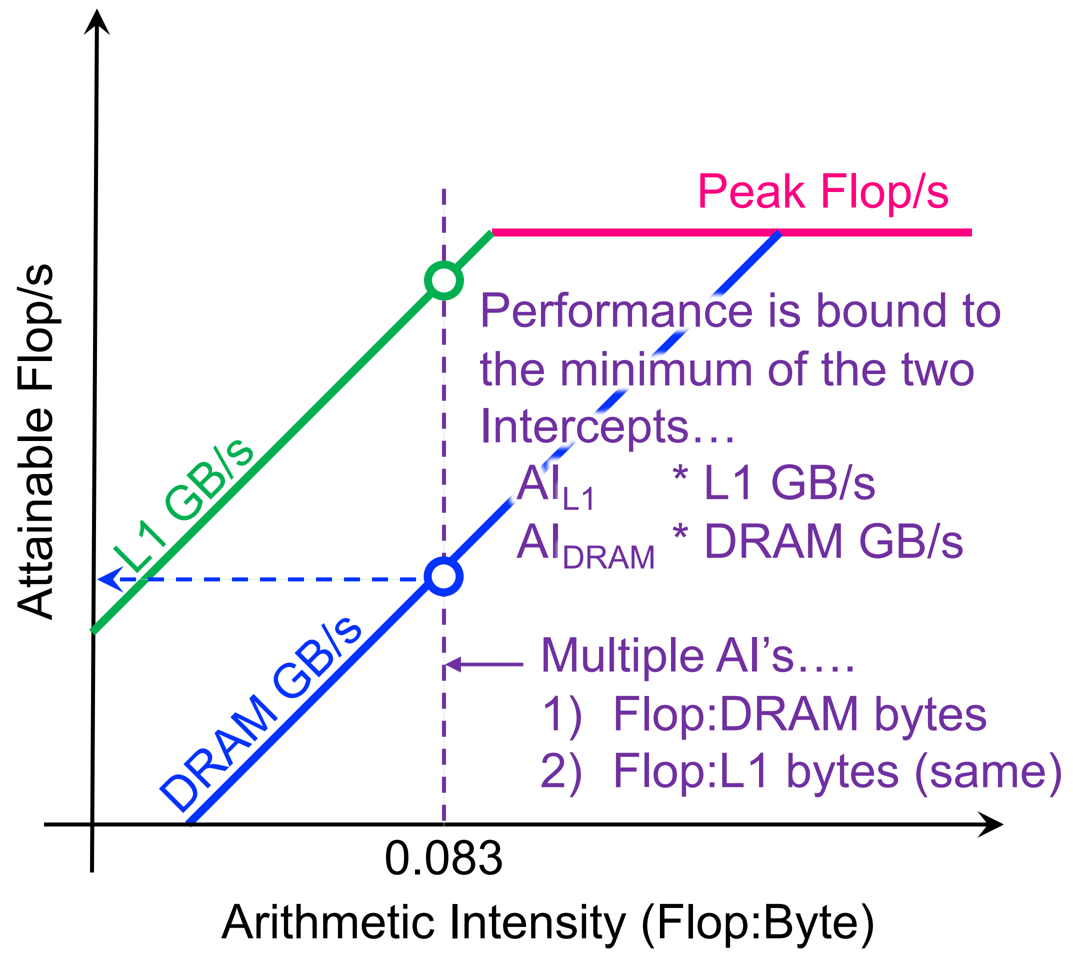
Example: STREAM

- L1 AI…
  - 2 flops
  - 2 x 8B load (old)
  - 1 x 8B store (new)
  - = 0.08 flops per byte
- No cache reuse…
  - Iteration i doesn't touch any data associated with iteration i+delta for any delta.
- … leads to a DRAM AI equal to the L1 AI

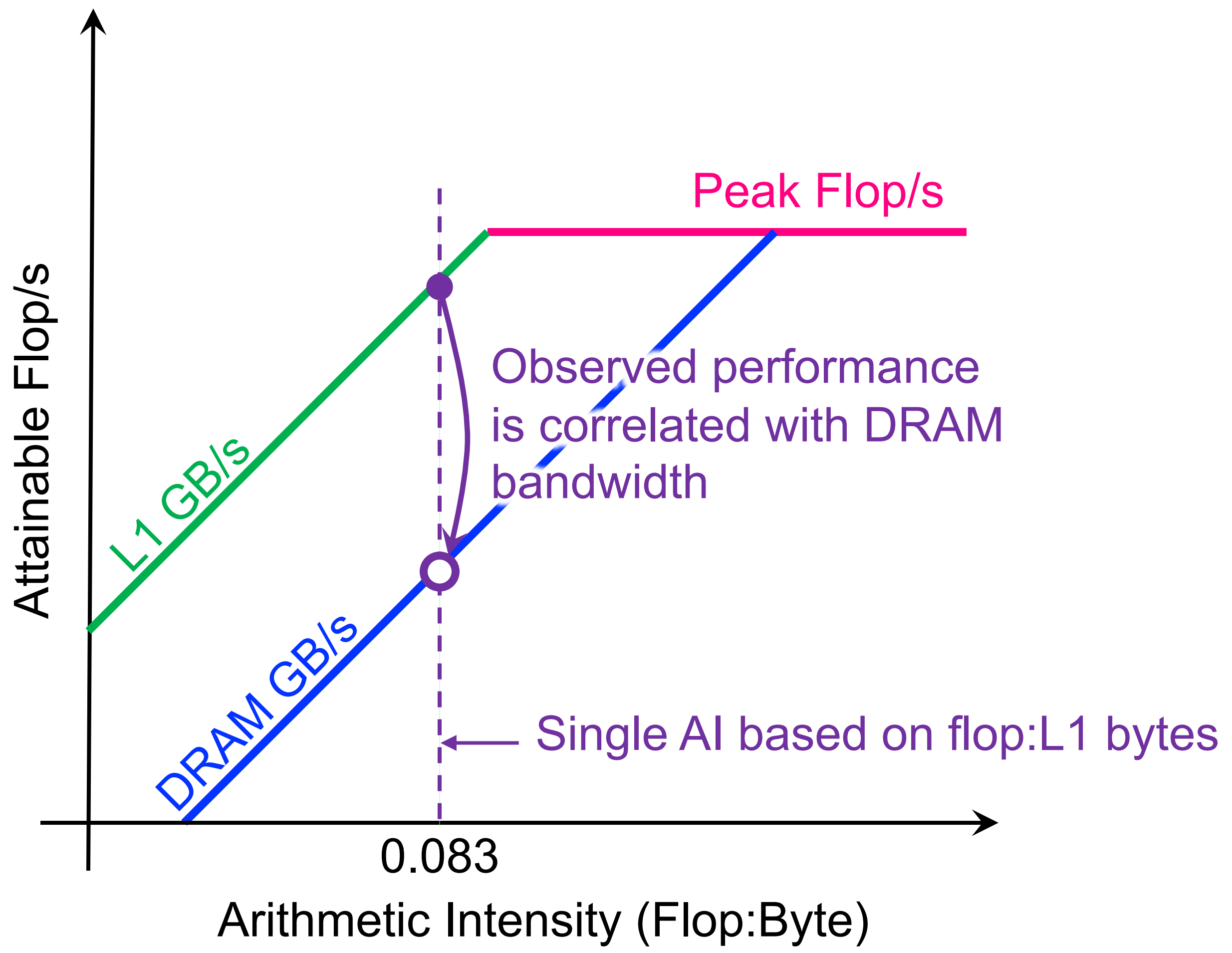
```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

Example: STREAM

Hierarchical Roofline



Cache-Aware Roofline


Peak Flop/s
Observed performance is correlated with DRAM bandwidth
Single AI based on flop:L1 bytes
L1 GB/s
DRAM GB/s
Attainable Flop/s
0.083
Arithmetic Intensity (Flop:Byte)

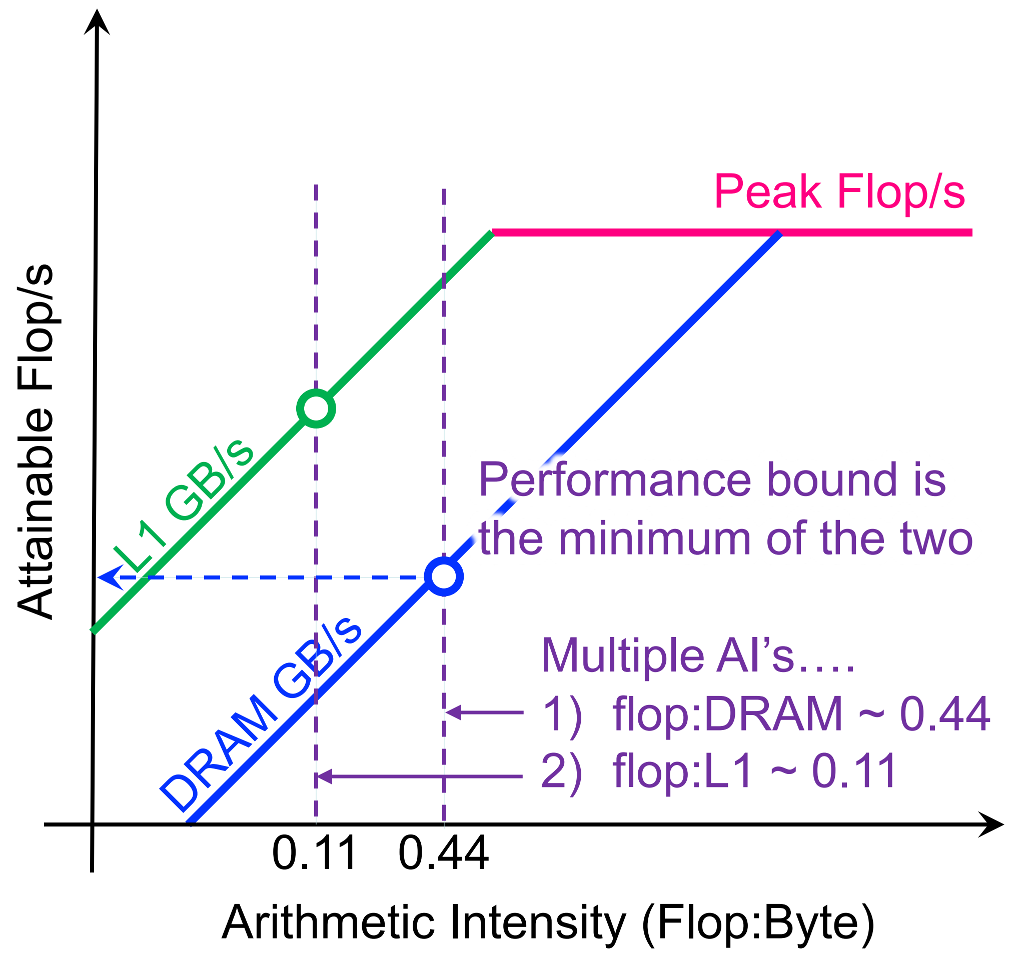
Example: 7-point Stencil (Small Problem)

- L1 AI…
  - 7 flops
  - 7 x 8B load (old)
  - 1 x 8B store (new)
  - = 0.11 flops per byte
  - some compilers may do register shuffles to reduce the number of loads.
- Moderate cache reuse…
  - old[ijk] is reused on subsequent iterations of i,j,k
  - old[ijk-1] is reused on subsequent iterations of i.
  - old[ijk-jStride] is reused on subsequent iterations of j.
  - old[ijk-kStride] is reused on subsequent iterations of k.
- … leads to DRAM AI larger than the L1 AI

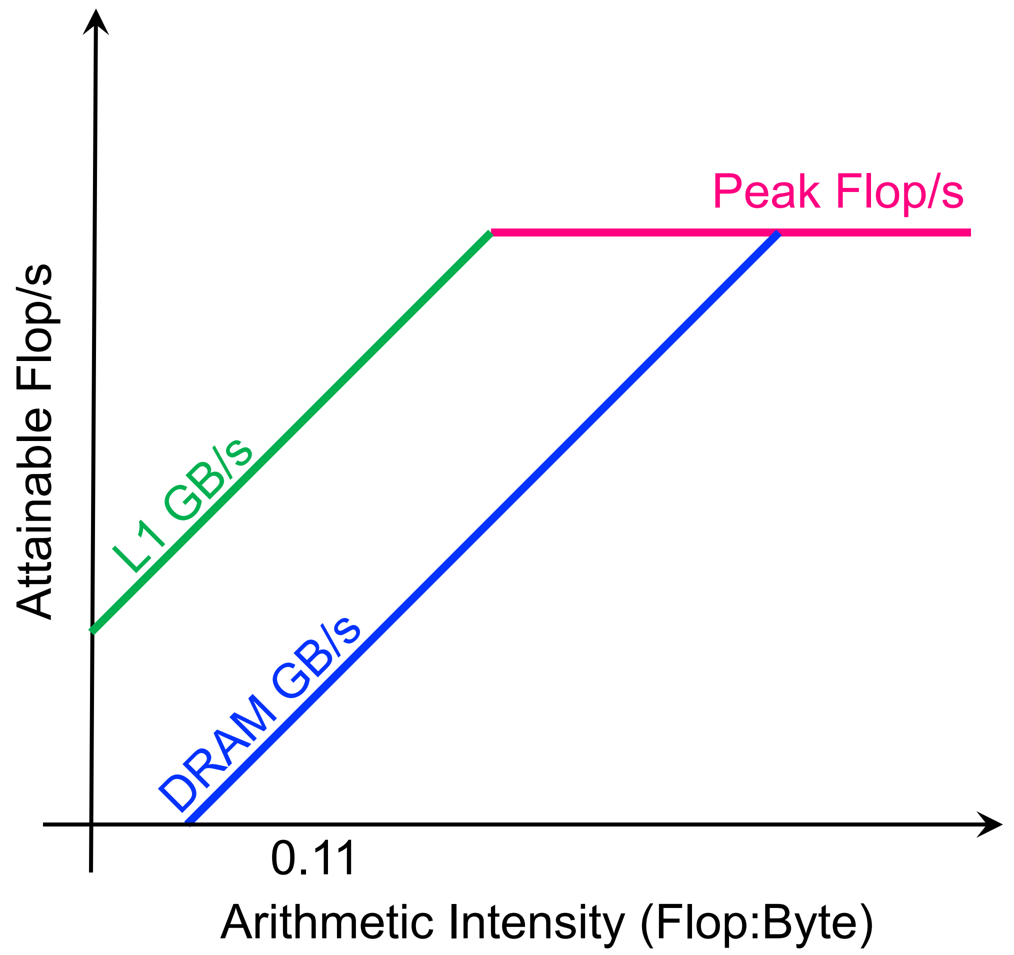
```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                    + old[k  ][j  ][i-1]
                    + old[k  ][j  ][i+1]
                    + old[k  ][j-1][i  ]
                    + old[k  ][j+1][i  ]
                    + old[k-1][j  ][i  ]
                    + old[k+1][j  ][i  ];
}}}
```

Example: 7-point Stencil (Small Problem)

Hierarchical Roofline

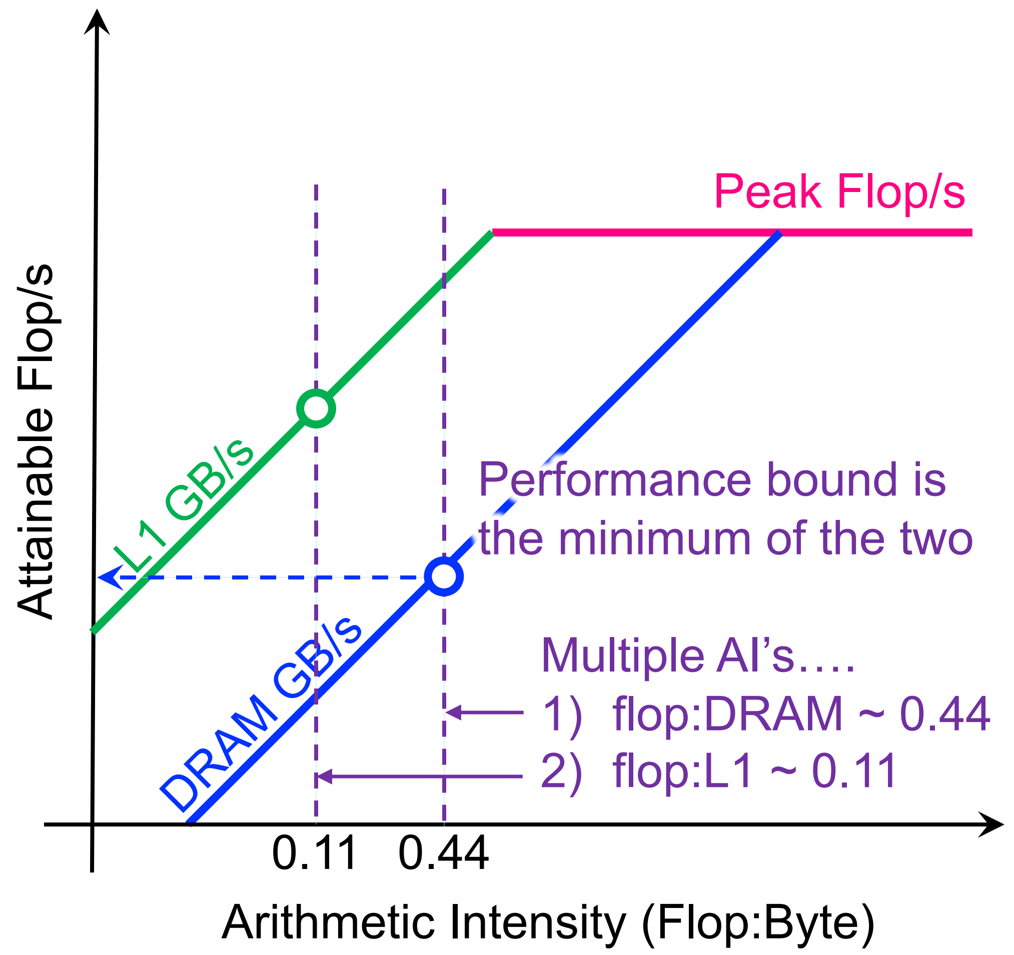


Cache-Aware Roofline

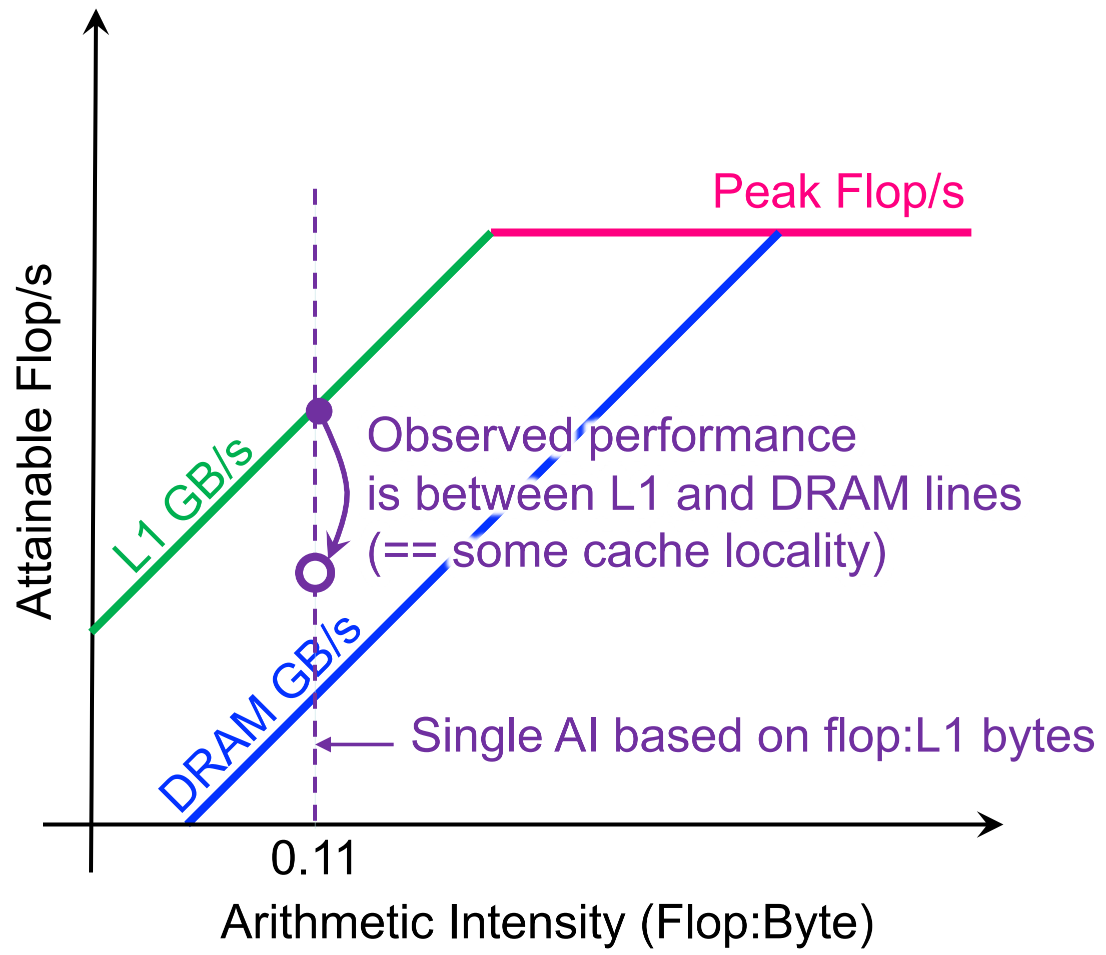


Example: 7-point Stencil (Small Problem)

Hierarchical Roofline

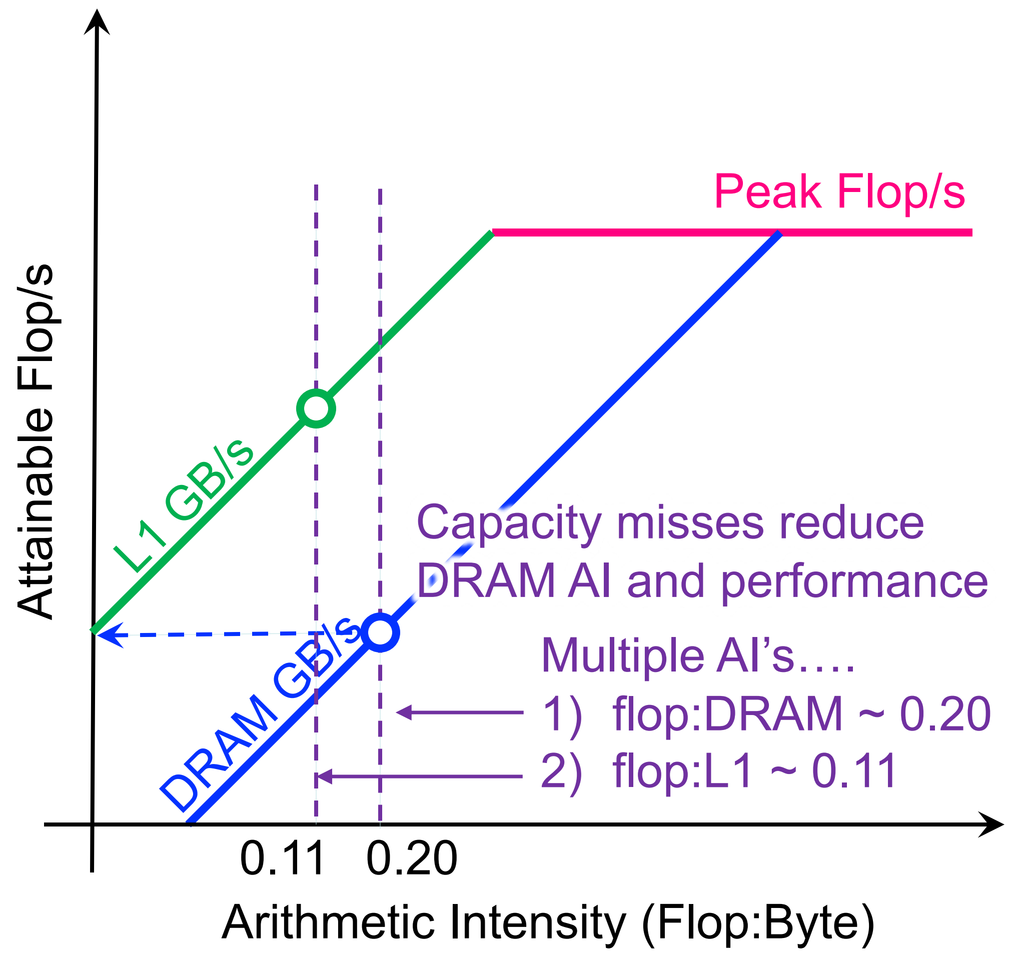


Cache-Aware Roofline

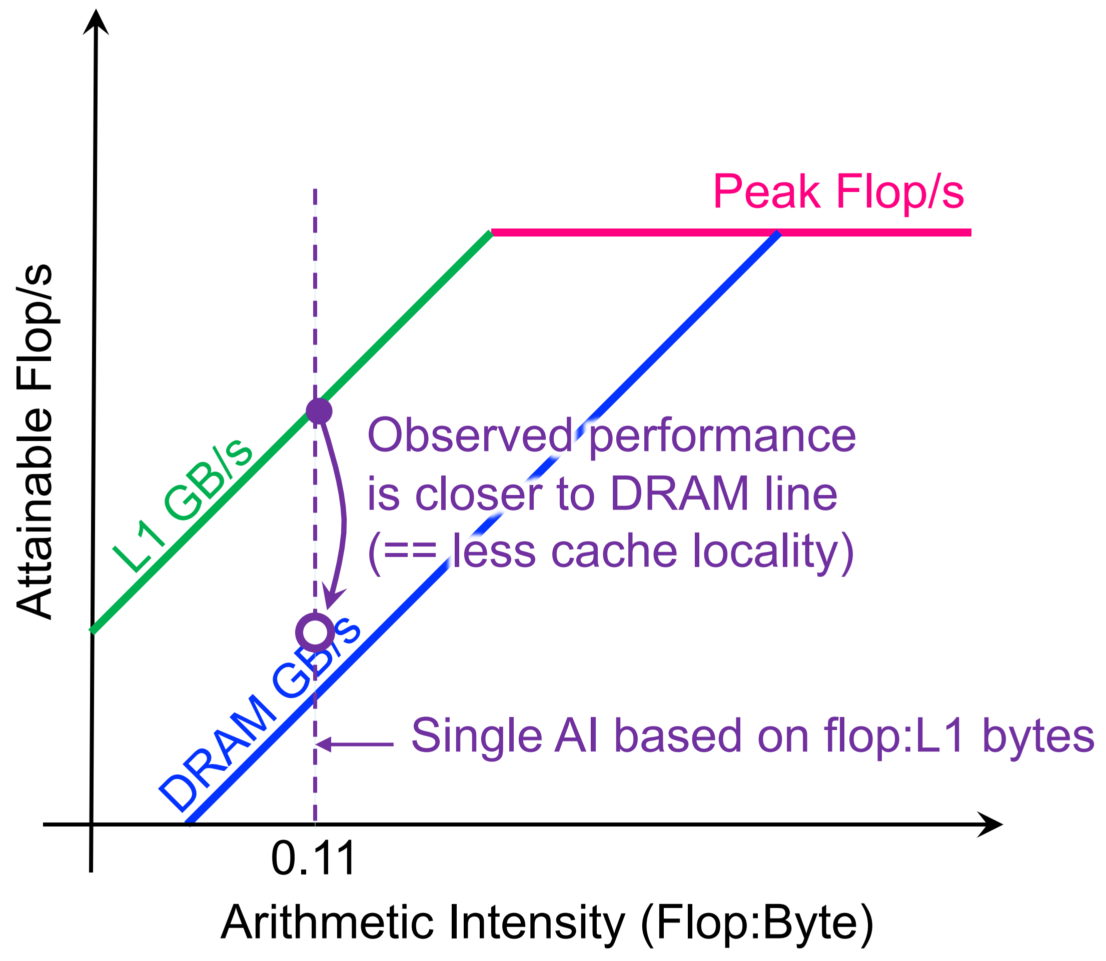

Peak Flop/s
Attainable Flop/s
L1 GB/s
DRAM GB/s
Observed performance
is between L1 and DRAM lines
(== some cache locality)
Single AI based on flop:L1 bytes
0.11
Arithmetic Intensity (Flop:Byte)

Example: 7-point Stencil (Large Problem)

Hierarchical Roofline

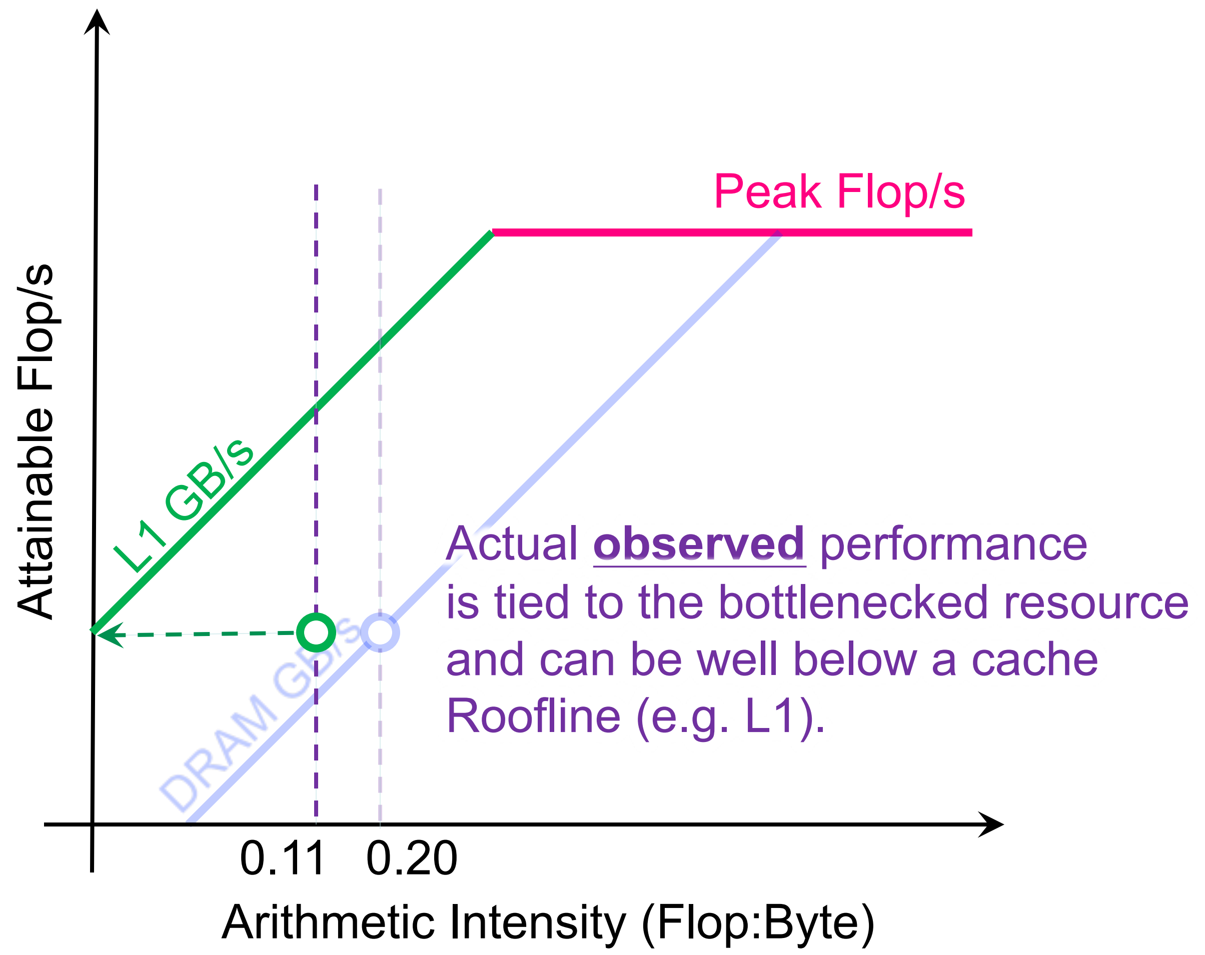


Cache-Aware Roofline


Peak Flop/s
L1 GB/s
DRAM GB/s
Attainable Flop/s
Observed performance is closer to DRAM line (== less cache locality)
Single AI based on flop:L1 bytes
0.11
Arithmetic Intensity (Flop:Byte)

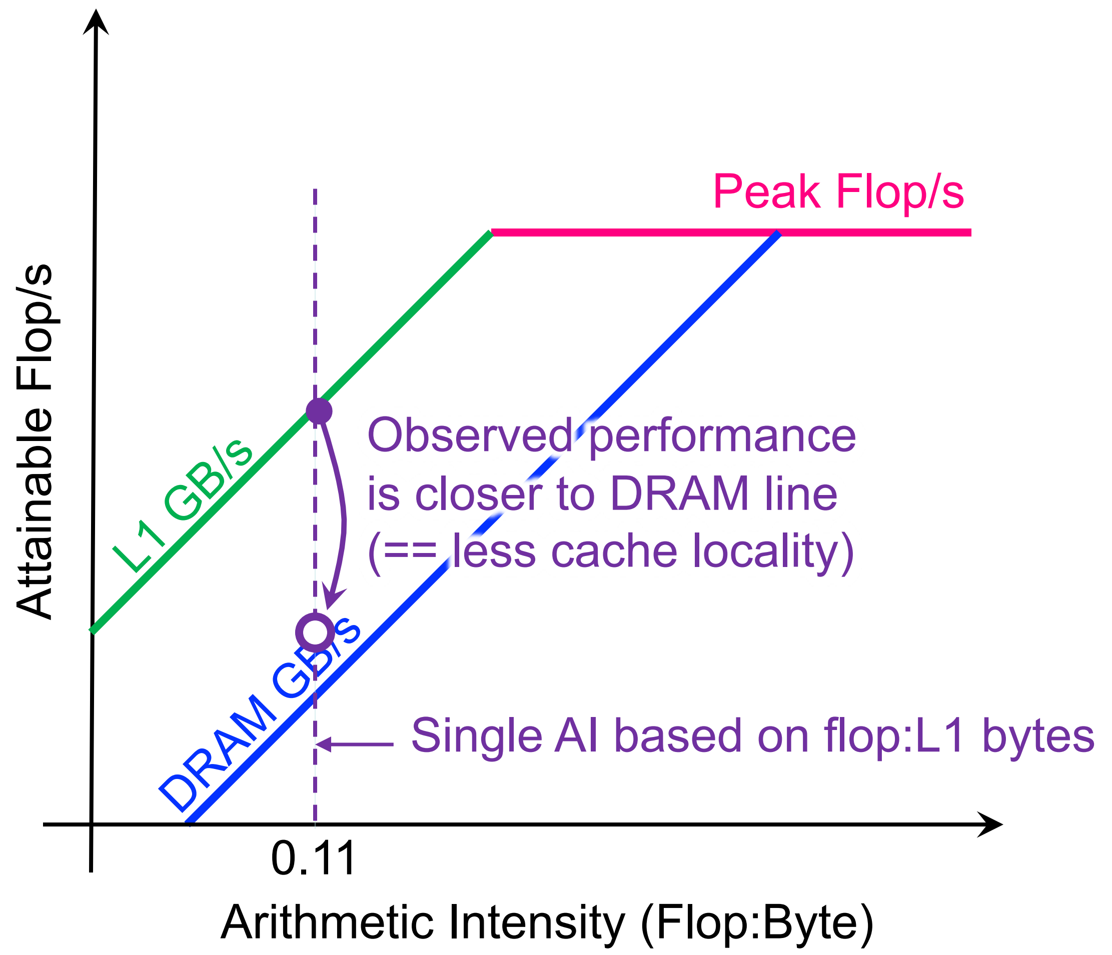
Example: 7-point Stencil (Observed Perf.)

Hierarchical Roofline



Actual **observed** performance is tied to the bottlenecked resource and can be well below a cache Roofline (e.g. L1).

Cache-Aware Roofline

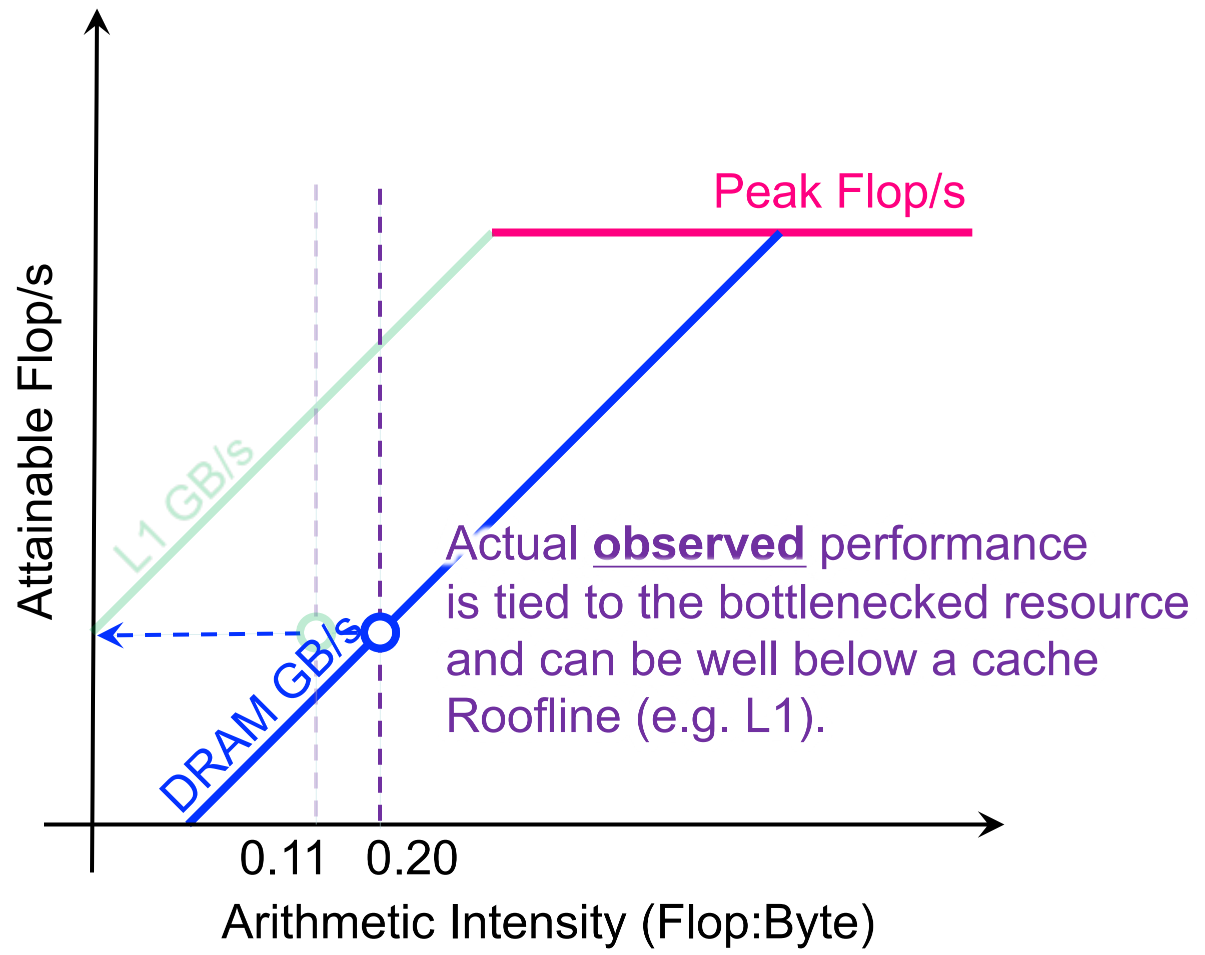


Observed performance is closer to DRAM line (== less cache locality)

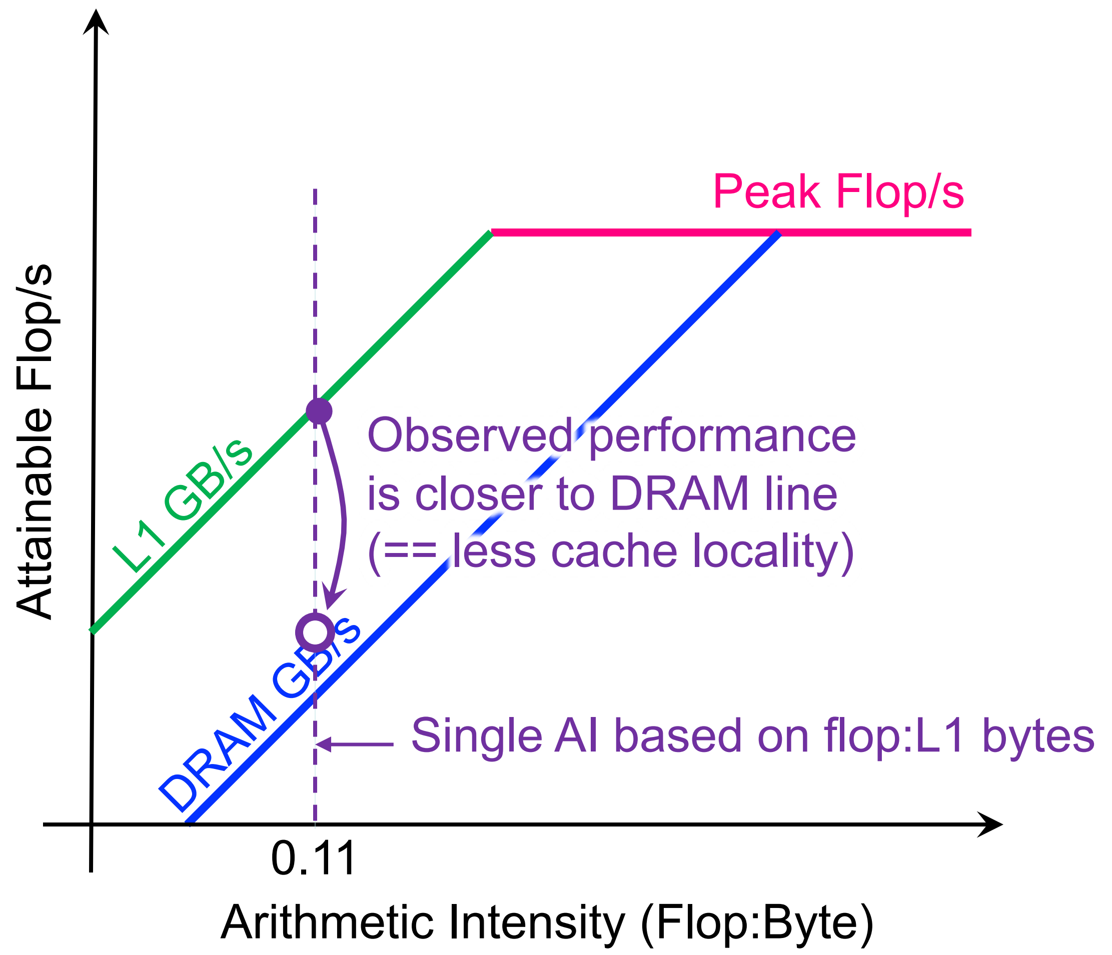
Single AI based on flop:L1 bytes

Example: 7-point Stencil (Observed Perf.)

Hierarchical Roofline


Peak Flop/s
Attainable Flop/s
L1 GB/s
DRAM GB/s
Actual **observed** performance is tied to the bottlenecked resource and can be well below a cache Roofline (e.g. L1).
0.11
0.20
Arithmetic Intensity (Flop:Byte)

Cache-Aware Roofline


Peak Flop/s
Attainable Flop/s
L1 GB/s
DRAM GB/s
Observed performance is closer to DRAM line (== less cache locality)
Single AI based on flop:L1 bytes
0.11
Arithmetic Intensity (Flop:Byte)