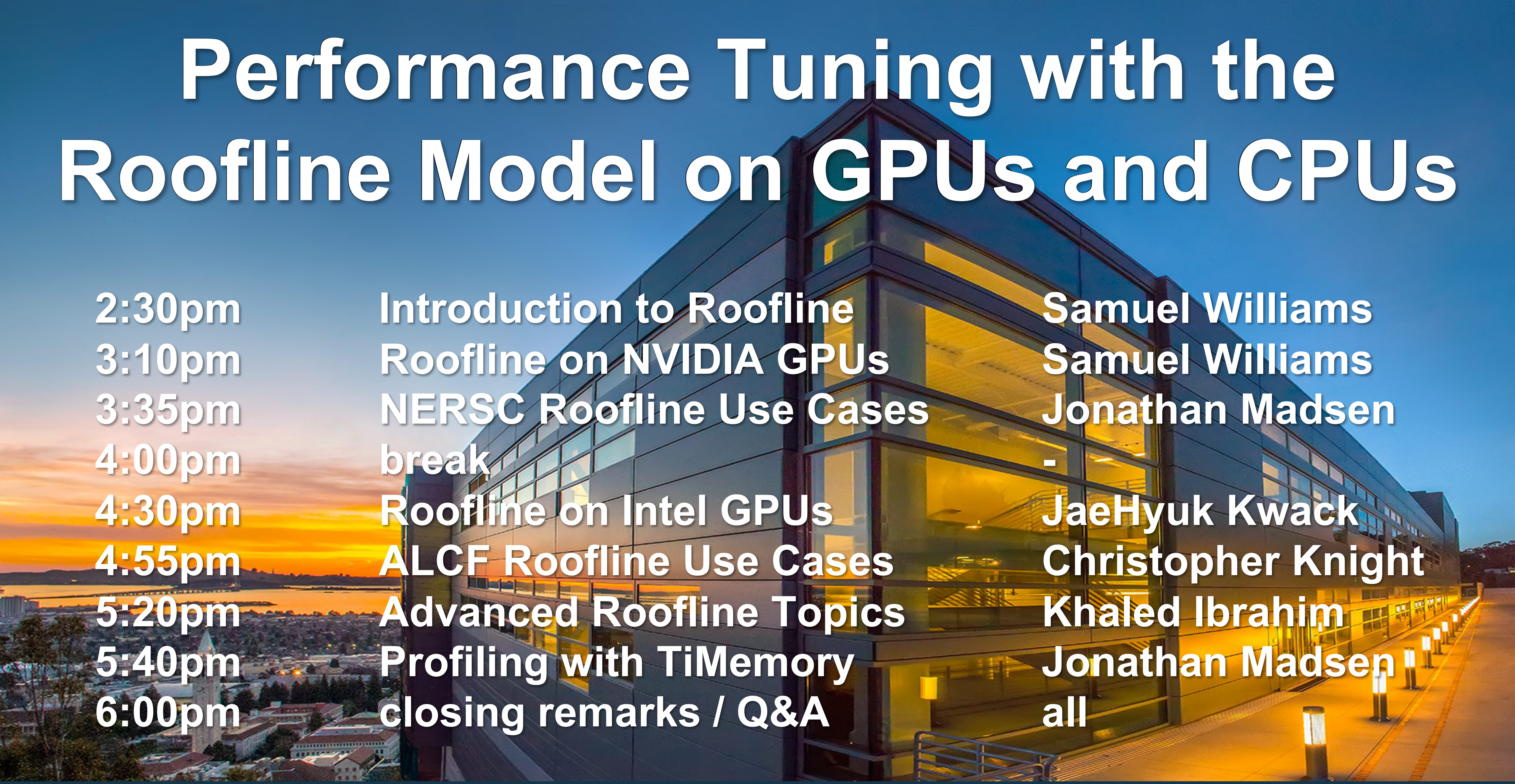


Performance Tuning with the Roofline Model on GPUs and CPUs

2:30pm	Introduction to Roofline	Samuel Williams
3:10pm	Roofline on NVIDIA GPUs	Samuel Williams
3:35pm	NERSC Roofline Use Cases	Jonathan Madsen
4:00pm	break	-
4:30pm	Roofline on Intel GPUs	JaeHyuk Kwack
4:55pm	ALCF Roofline Use Cases	Christopher Knight
5:20pm	Advanced Roofline Topics	Khaled Ibrahim
5:40pm	Profiling with TiMemory	Jonathan Madsen
6:00pm	closing remarks / Q&A	all

Presenters

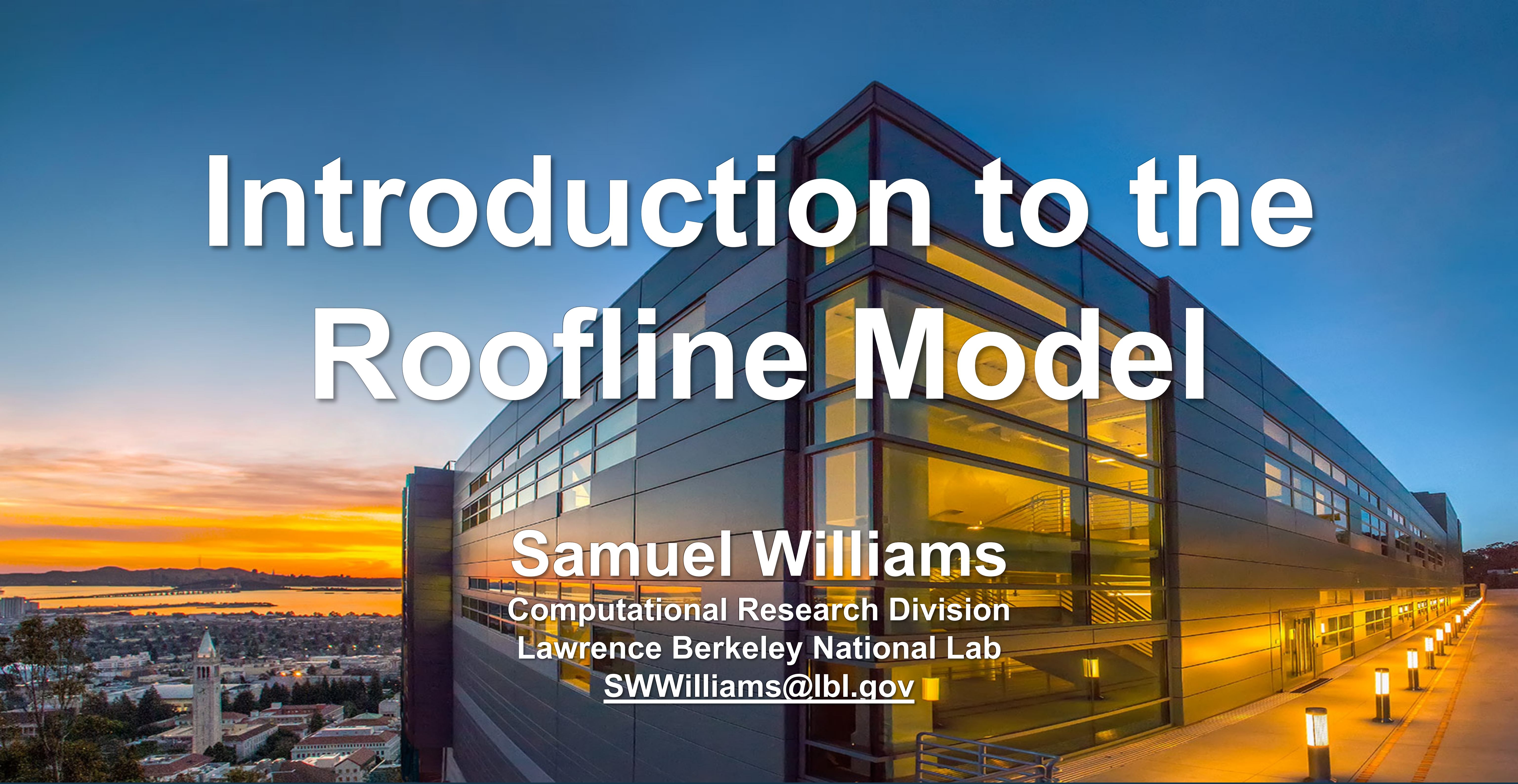
Samuel Williams
Computational Research Division
Lawrence Berkeley National Lab
SWWilliams@lbl.gov

JaeHyuk Kwack
ALCF
Argonne National Lab
jkwack@anl.gov

Jonathan Madsen
NERSC
Lawrence Berkeley National Lab
JRMadsen@lbl.gov

Khaled Ibrahim
Computational Research Division
Lawrence Berkeley National Lab
KZIbrahim@lbl.gov

Christopher Knight
ALCF
Argonne National Lab
knightc@anl.gov

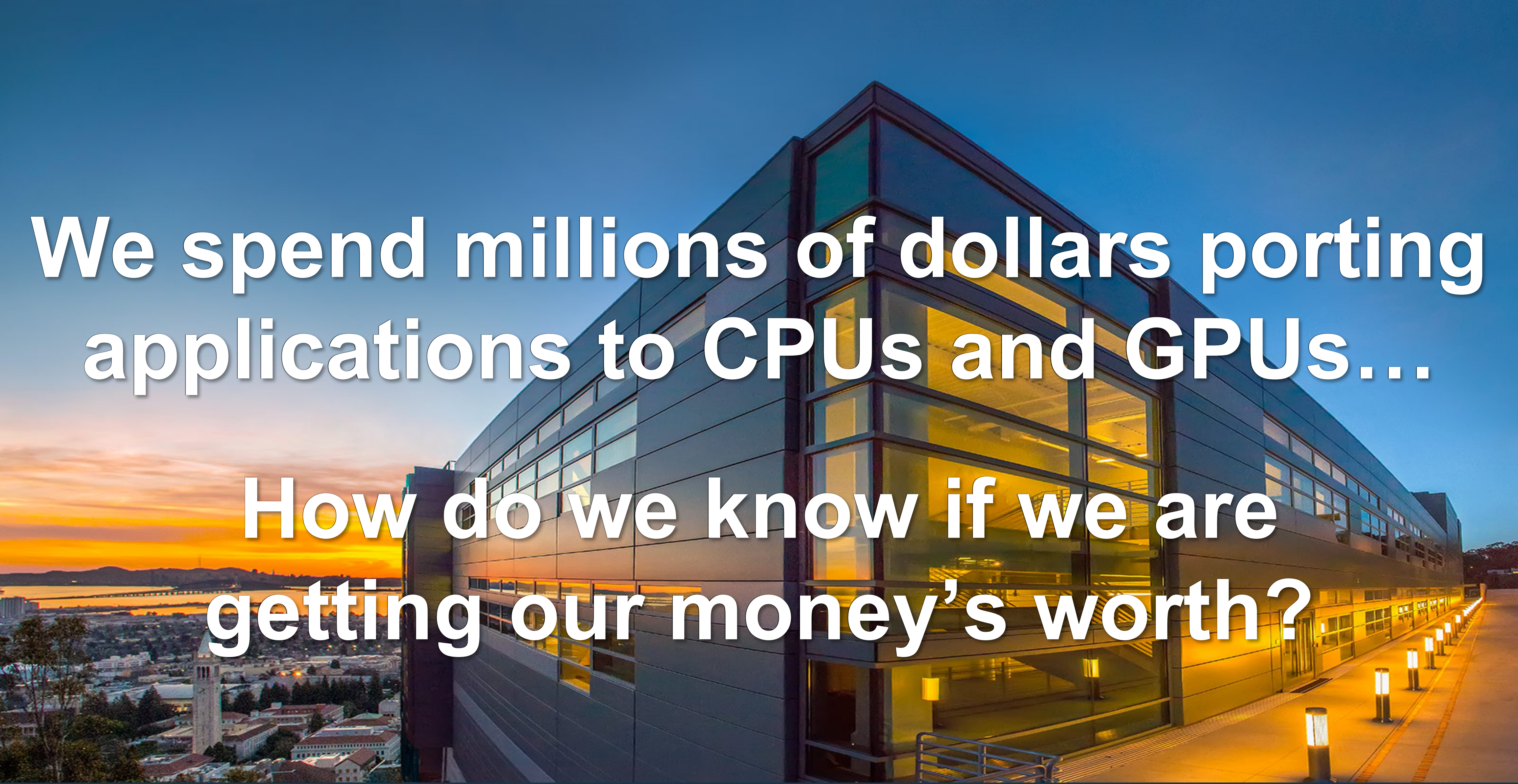


Introduction to the Roofline Model

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.
- This research used resources of the Oak Ridge Leadership Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.



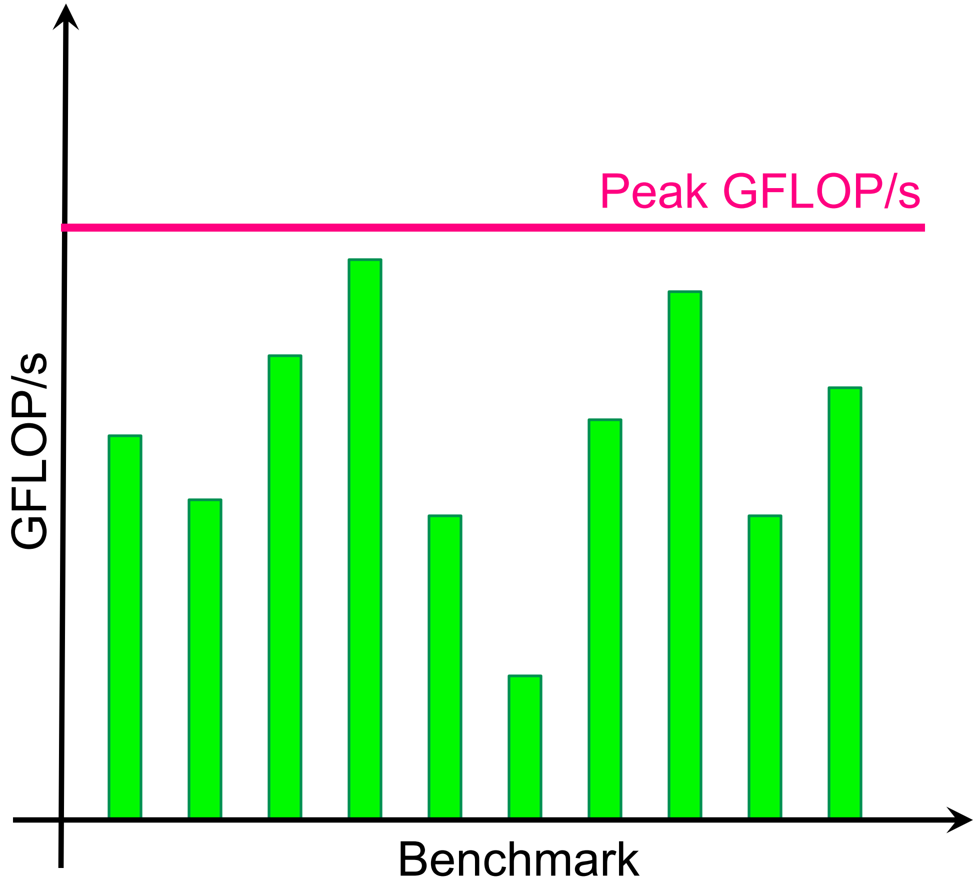
We spend millions of dollars porting applications to CPUs and GPUs…

How do we know if we are getting our money’s worth?

Getting our money’s worth?

- Do we get good performance on application benchmarks?

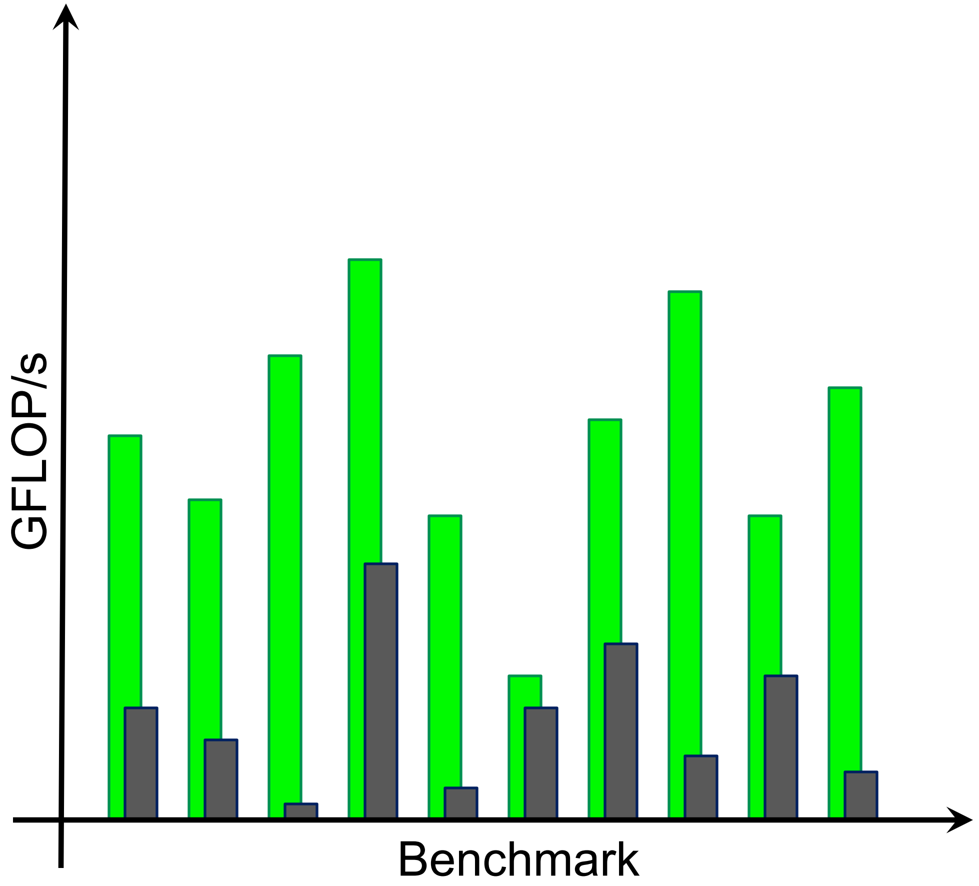
- Imagine profiling a mix of GPU-accelerated benchmarks …
- GFLOP/s alone may not be particularly insightful



Are we getting good performance?

- We could compare performance to a CPU…
  - ○ Speedup may seem random
  - ○ Aren't GPUs always 10x faster than a CPU?
  - ○ If not, what does that tell us about architecture, algorithm or implementation?

➢ **'Speedup' provides no insights into architecture, algorithm, or implementation.**



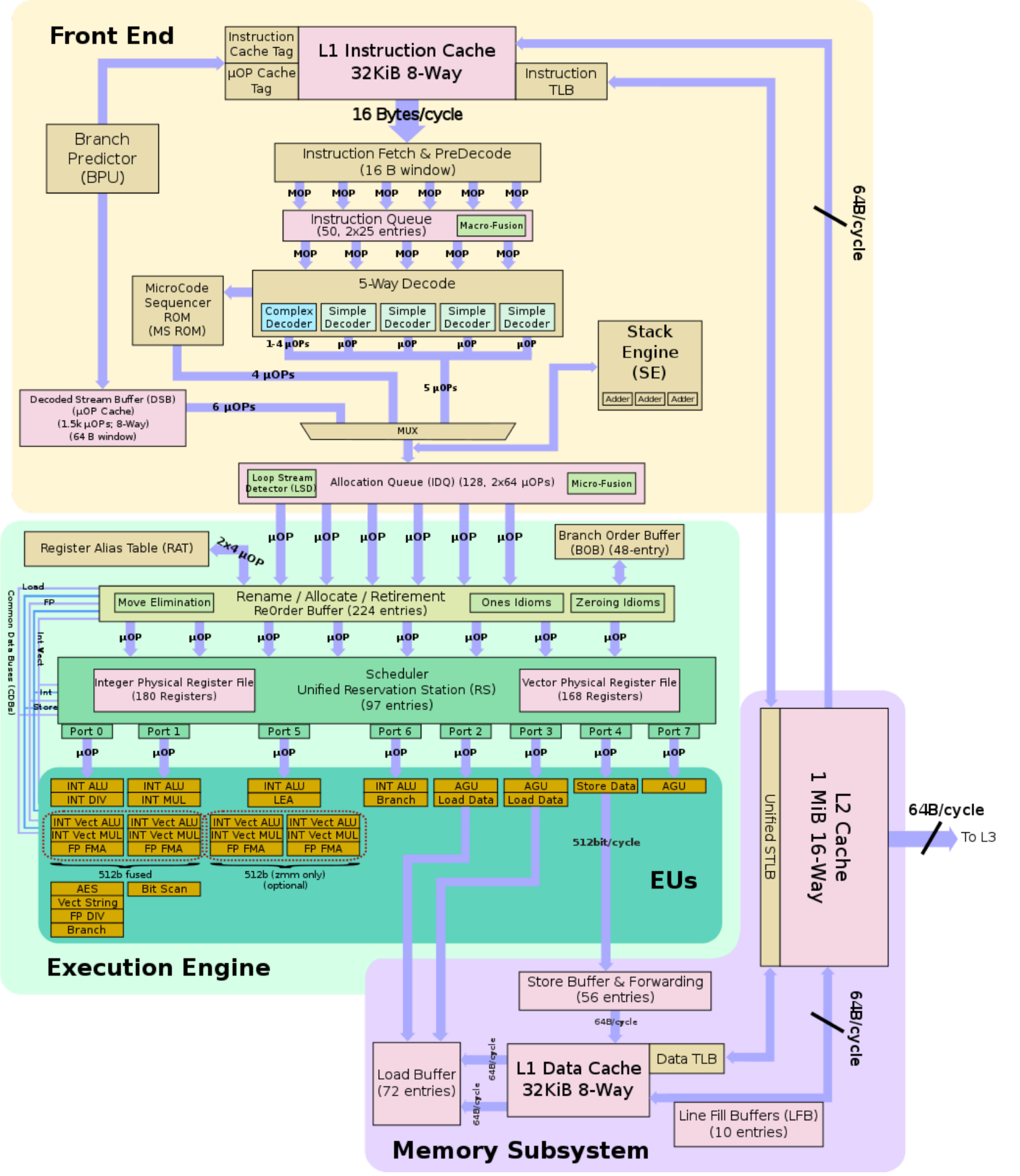
Are we getting good performance?

- We could take a CS approach and look at performance counters…
  - ○ Record microarchitectural events on CPUs/GPUs
  - ○ Use arcane, architecture-specific terminology
  - ○ May be broken
  - ○ We may be able to show correlation between events, but…
  - ➢ **…providing actionable guidance to CS, AM, applications, or procurement can prove elusive.**

```
.
.
.
FRONTEND_RETIRED.LATENCY_GE_8_PS
FRONTEND_RETIRED.LATENCY_GE_16_PS
FRONTEND_RETIRED.LATENCY_GE_32_PS
RS_EVENTS.EMPTY_END
FRONTEND_RETIRED.L2_MISS_PS
FRONTEND_RETIRED.L1I_MISS_PS
FRONTEND_RETIRED.STLB_MISS_PS
FRONTEND_RETIRED.ITLB_MISS_PS
ITLB_MISSES.WALK_COMPLETED
BR_MISP_RETIRED.ALL_BRANCHES_PS
IDQ.MS_SWITCHES
FRONTEND_RETIRED.LATENCY_GE_2_BUBBLES_GE_1_PS
BR_MISP_RETIRED.ALL_BRANCHES_PS
MACHINE_CLEARS.COUNT
MEM_LOAD_RETIRED.L1_HIT_PS
MEM_LOAD_RETIRED.FB_HIT_PS
MEM_LOAD_UOPS_RETIRED.L1_HIT_PS
MEM_LOAD_UOPS_RETIRED.HIT_LFB_PS
MEM_INST_RETIRED.STLB_MISS_LOADS_PS
MEM_UOPS_RETIRED.STLB_MISS_LOADS_PS
MEM_LOAD_RETIRED.L2_HIT_PSMEM_LOAD_UOPS_RETIRED.L2_HIT_PS
MEM_LOAD_RETIRED.L3_HIT_PS
MEM_LOAD_UOPS_RETIRED.LLC_HIT_PS
MEM_LOAD_UOPS_RETIRED.L3_HIT_PS
MEM_LOAD_RETIRED.L3_MISS_PS
MEM_LOAD_UOPS_RETIRED.LLC_MISS_PS
MEM_LOAD_UOPS_MISC_RETIRED.LLC_MISS_PS
MEM_LOAD_UOPS_RETIRED.L3_MISS_PS
MEM_INST_RETIRED.ALL_STORES_PS
MEM_UOPS_RETIRED.ALL_STORES_PS
ARITH.DIVIDER_ACTIVE
ARITH.DIVIDER_UOPS
ARITH.FPU_DIV_ACTIVE
INST_RETIRED.PREC_DIST
IDQ.MS_UOPS
INST_RETIRED.PREC_DIST
.
.
.
```

Are we getting good performance?

- We could take the computer architect’s approach and build a simulator to understand performance nuances…
  - ○ Modern architectures are incredibly complex
  - ○ Simulators may perfectly reproduce performance
  - ○ Deluge of information interpretable only by computer architects
  - ○ worse, might incur 10^6x slowdowns
  - ➢ **Provide no insights into quality or limits of algorithm or implementation.**
  - ➢ **Provide no guidance to CS, AM, applications, or procurement.**



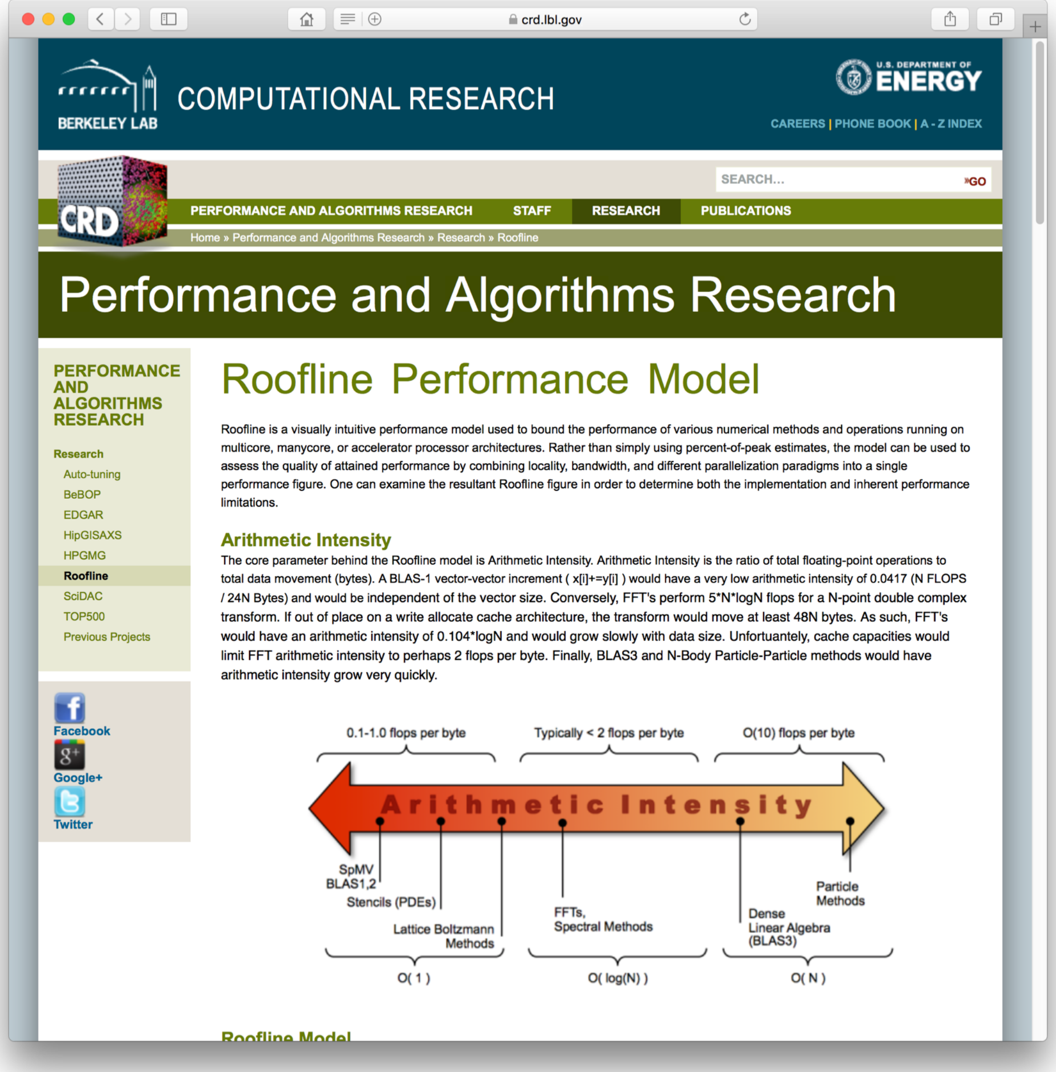
Front End
Instruction Cache Tag
µOP Cache Tag
L1 Instruction Cache 32KiB 8-Way
Instruction TLB
16 Bytes/cycle
Branch Predictor (BPU)
Instruction Fetch & PreDecode (16 B window)
MOP
Instruction Queue (50, 2x25 entries)
Macro-Fusion
MicroCode Sequencer ROM (MS ROM)
5-Way Decode
Complex Decoder
Simple Decoder
1-4 µOPs
µOP
4 µOPs
5 µOPs
6 µOPs
Stack Engine (SE)
Adder
Decoded Stream Buffer (DSB) (µOP Cache) (1.5k µOPs; 8-Way) (64 B window)
MUX
Loop Stream Detector (LSD)
Allocation Queue (IDQ) (128, 2x64 µOPs)
Micro-Fusion
64B/cycle
Register Alias Table (RAT)
2x4 µOP
Branch Order Buffer (BOB) (48-entry)
Load
FP
Int Vect
Int
Store
Common Data Buses (CDBs)
Move Elimination
Rename / Allocate / Retirement ReOrder Buffer (224 entries)
Ones Idioms
Zeroing Idioms
Integer Physical Register File (180 Registers)
Scheduler Unified Reservation Station (RS) (97 entries)
Vector Physical Register File (168 Registers)
Port 0
Port 1
Port 5
Port 6
Port 2
Port 3
Port 4
Port 7
INT ALU
INT DIV
INT MUL
LEA
Branch
AGU
Load Data
Store Data
INT Vect ALU
INT Vect MUL
FP FMA
512b fused
512b (zmm only) (optional)
AES
Vect String
FP DIV
Bit Scan
512bit/cycle
EUs
Execution Engine
Unified STLB
L2 Cache 1 MiB 16-Way
64B/cycle
To L3
Store Buffer & Forwarding (56 entries)
Load Buffer (72 entries)
L1 Data Cache 32KiB 8-Way
Data TLB
Line Fill Buffers (LFB) (10 entries)
Memory Subsystem

BERKELEY LAB

What’s missing…

- Each community speaks their own language and develops specialized tools/methodologies
- Need common mental model of application execution on target system
- Sacrifice accuracy to gain…
  - Architecture independence / extensibility
  - Readily understandable by broad community
  - Intuition, insights, and guidance to CS, AM, apps, procurement, and vendors

➢ **Roofline is just such a model**


crd.lbl.gov
BERKELEY LAB
COMPUTATIONAL RESEARCH
U.S. DEPARTMENT OF ENERGY
CAREERS | PHONE BOOK | A - Z INDEX
CRD
SEARCH...
»GO
PERFORMANCE AND ALGORITHMS RESEARCH
STAFF
RESEARCH
PUBLICATIONS
Home » Performance and Algorithms Research » Research » Roofline
PERFORMANCE AND ALGORITHMS RESEARCH
Research
Auto-tuning
BeBOP
EDGAR
HipGISAXS
HPGMG
Roofline
SciDAC
TOP500
Previous Projects
Facebook
Google+
Twitter

Performance and Algorithms Research

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

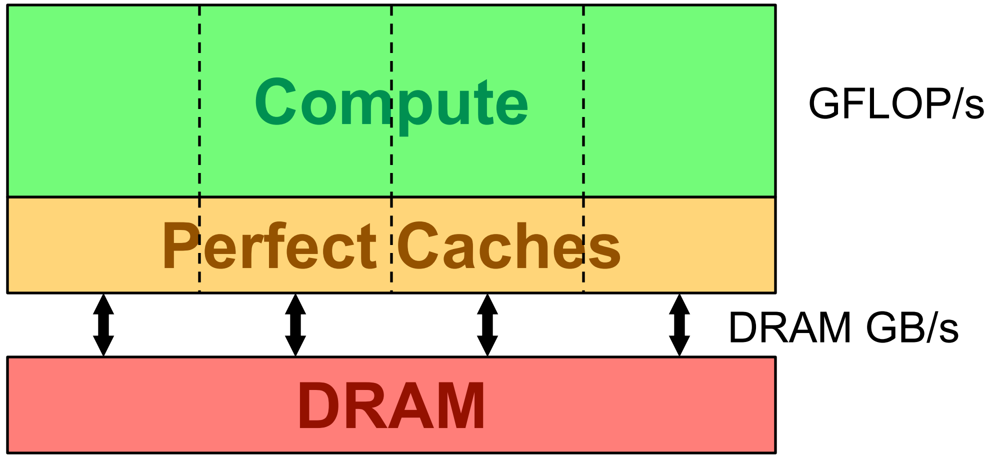
The core parameter behind the Roofline model is Arithmetic Intensity. Arithmetic Intensity is the ratio of total floating-point operations to total data movement (bytes). A BLAS-1 vector-vector increment (x[i]+=y[i]) would have a very low arithmetic intensity of 0.0417 (N FLOPS / 24N Bytes) and would 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

Data Movement or Compute?

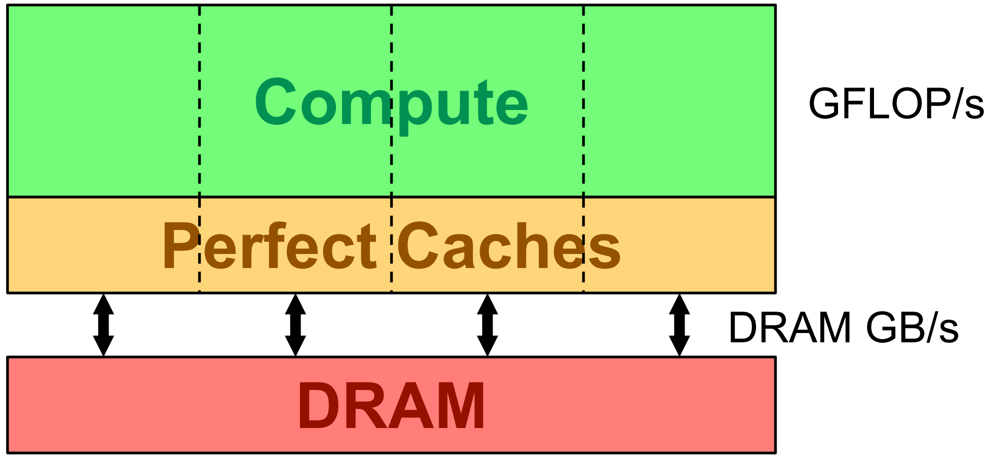
- Which takes longer?
  - Data Movement
  - Compute



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

Data Movement or Compute?

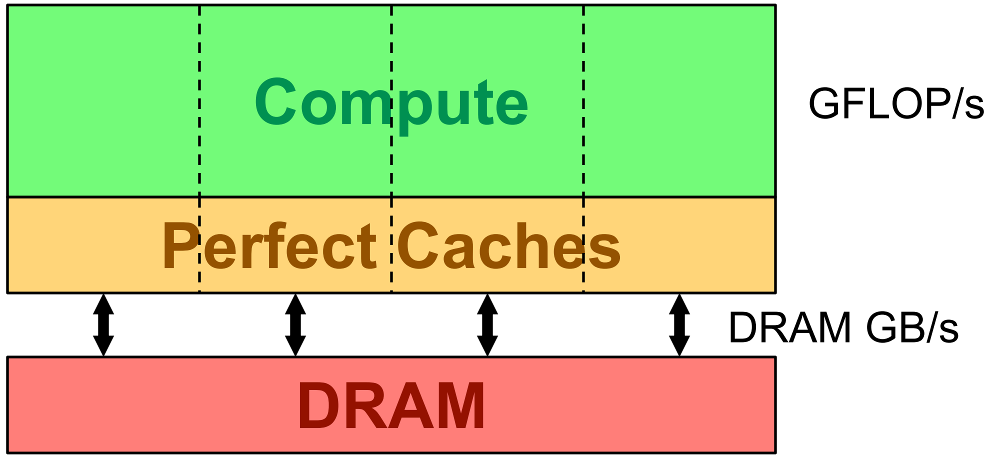
- Which takes longer?
  - Data Movement
  - Compute
- Is performance limited by compute or data movement?



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

Data Movement or Compute?

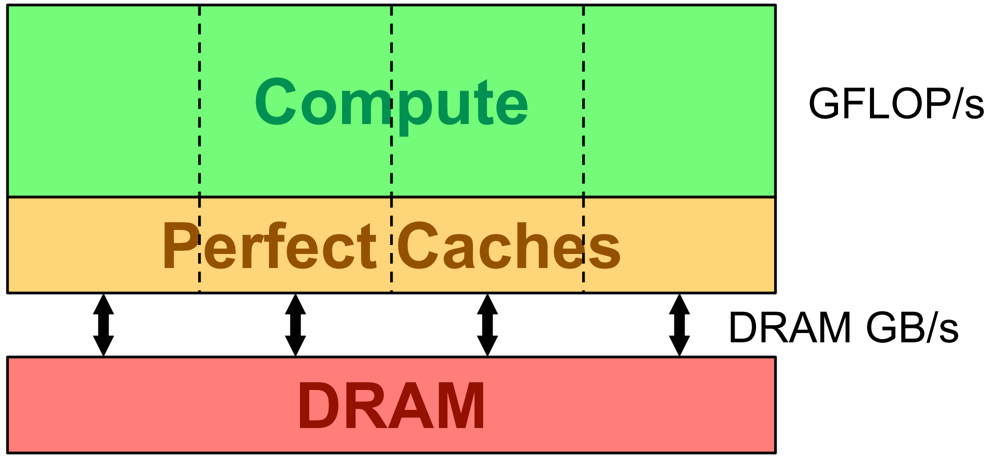
- Which takes longer?
  - Data Movement
  - Compute
- Is performance limited by compute or data movement?



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

Data Movement or Compute?

- Which takes longer?
  - Data Movement
  - Compute
- Is performance limited by compute or data movement?



$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI * Peak GB/s} \end{cases}$$

Arithmetic Intensity (AI) = measure of data locality

Arithmetic Intensity

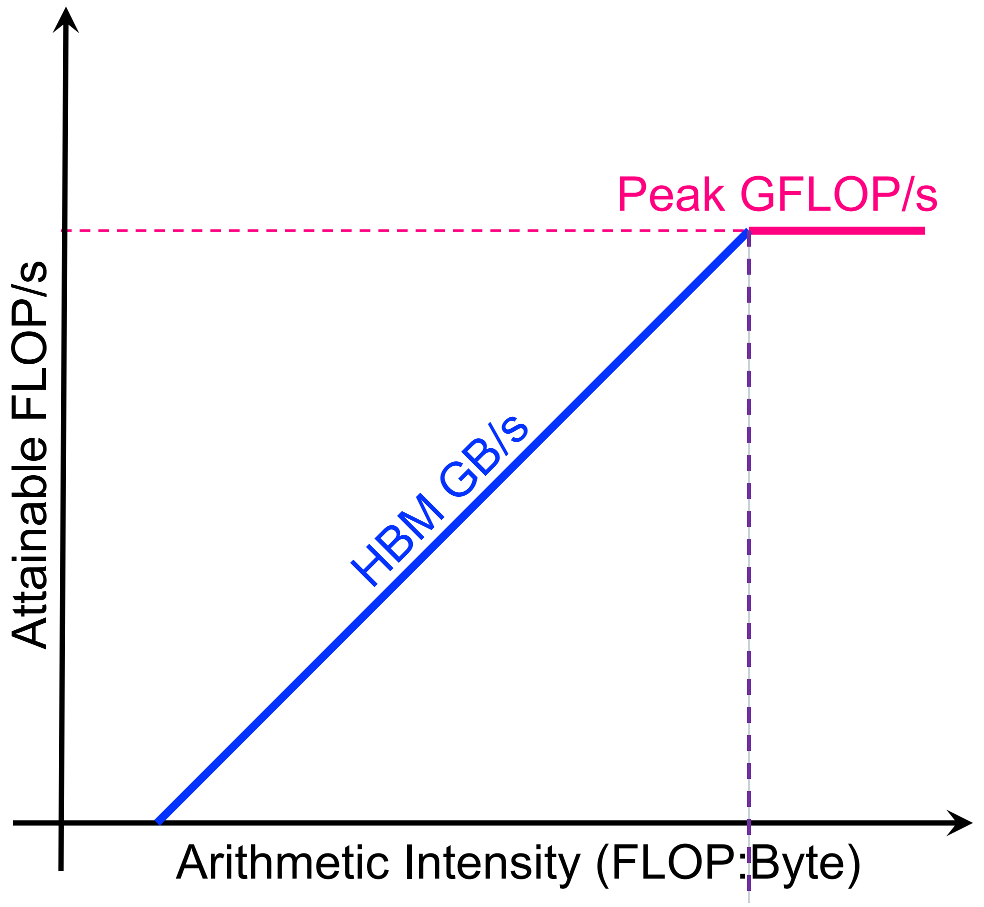
- Measure of data locality (data reuse)
- Ratio of **Total Flops** performed to **Total Bytes** moved
- For the DRAM Roofline…
  - Total Bytes to/from DRAM
  - Includes all cache and prefetcher effects
  - Can be very different from total loads/stores (bytes requested)
  - Equal to ratio of sustained GFLOP/s to sustained GB/s (time cancels)

(DRAM) Roofline Model

$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI * Peak GB/s} \end{cases}$$

AI (Arithmetic Intensity) = FLOPs / Bytes (moved to/from DRAM)

- Plot bound on **Log-log scale** as a function of AI (data locality)



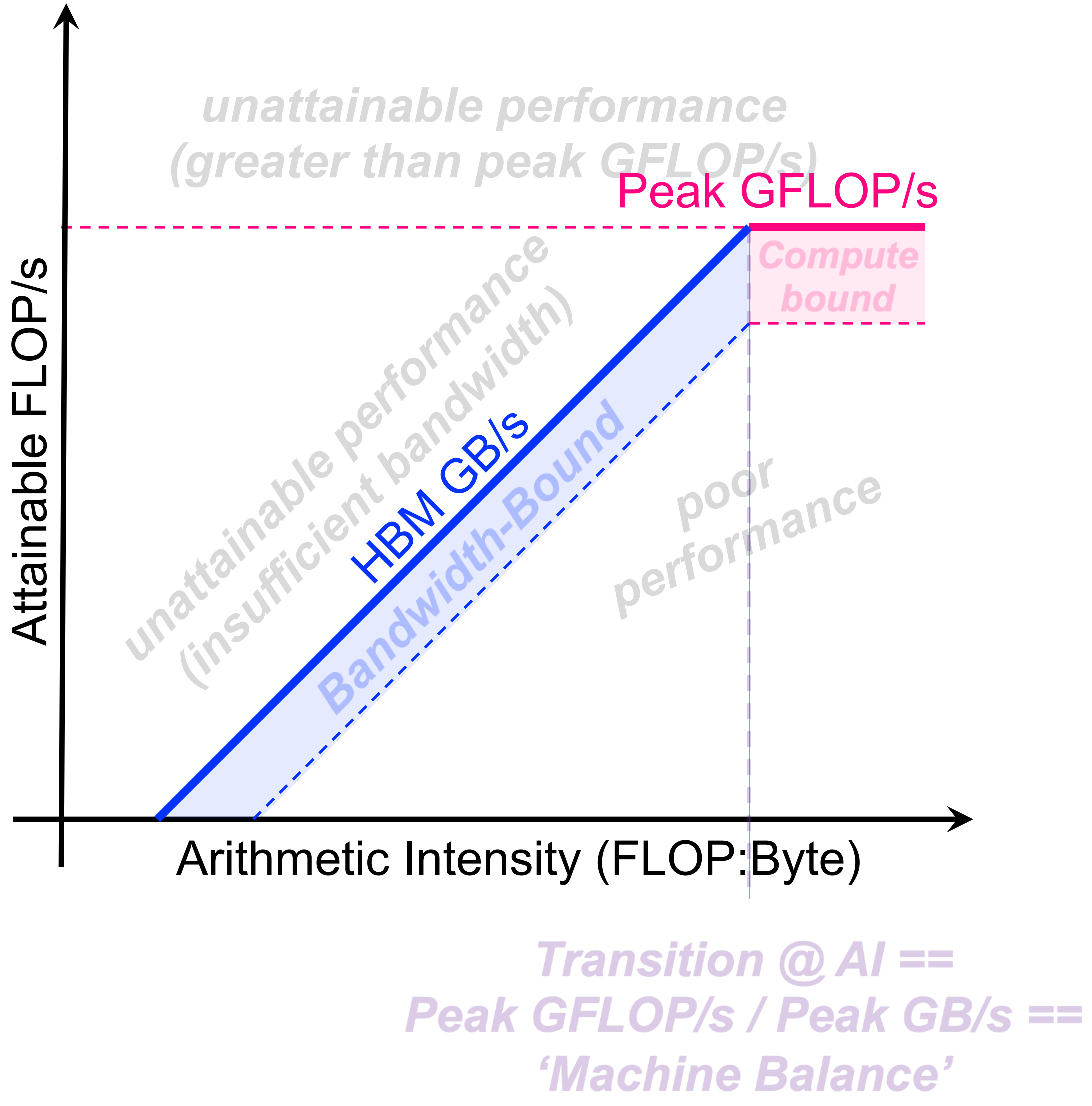
Transition @ AI ==
Peak GFLOP/s / Peak GB/s ==
'Machine Balance'

(DRAM) Roofline Model

$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI * Peak GB/s} \end{cases}$$

AI (Arithmetic Intensity) = FLOPs / Bytes (moved to/from DRAM)

- Plot bound on **Log-log scale** as a function of AI (data locality)
- Roofline tessellates the locality-performance plane into five regions…

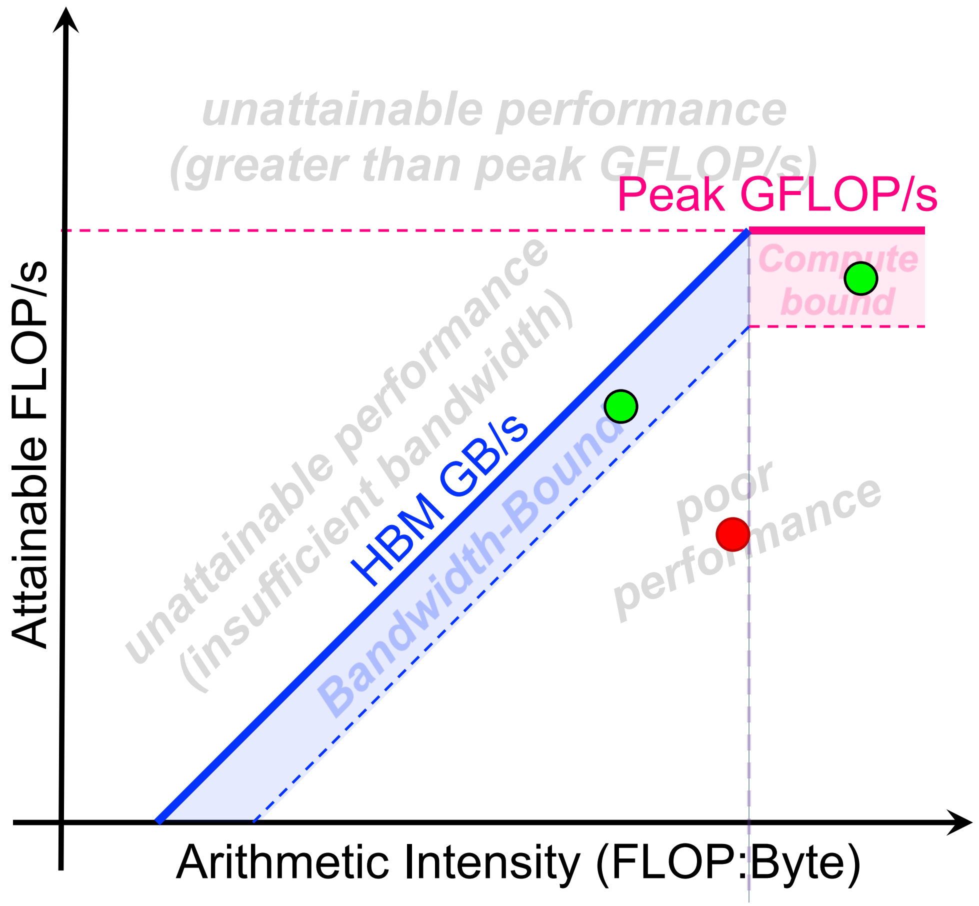


(DRAM) Roofline Model

$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI * Peak GB/s} \end{cases}$$

AI (Arithmetic Intensity) = FLOPs / Bytes (moved to/from DRAM)

- Plot bound on **Log-log scale** as a function of AI (data locality)
- Roofline tessellates the locality-performance plane into five regions…
- Measure application (AI,GF/s) and plot in the 2D locality-performance plane.



Transition @ AI ==
Peak GFLOP/s / Peak GB/s ==
'Machine Balance'

Roofline Examples

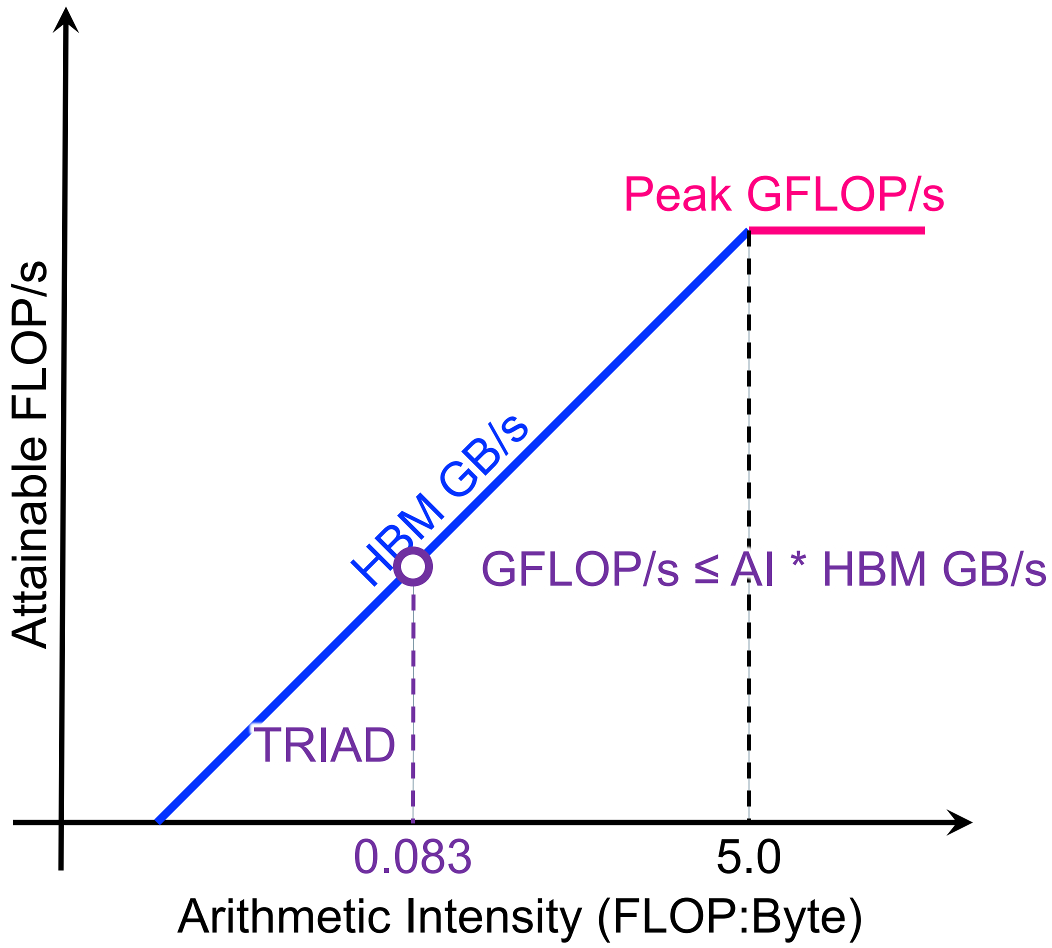
Roofline Example #1

- 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…

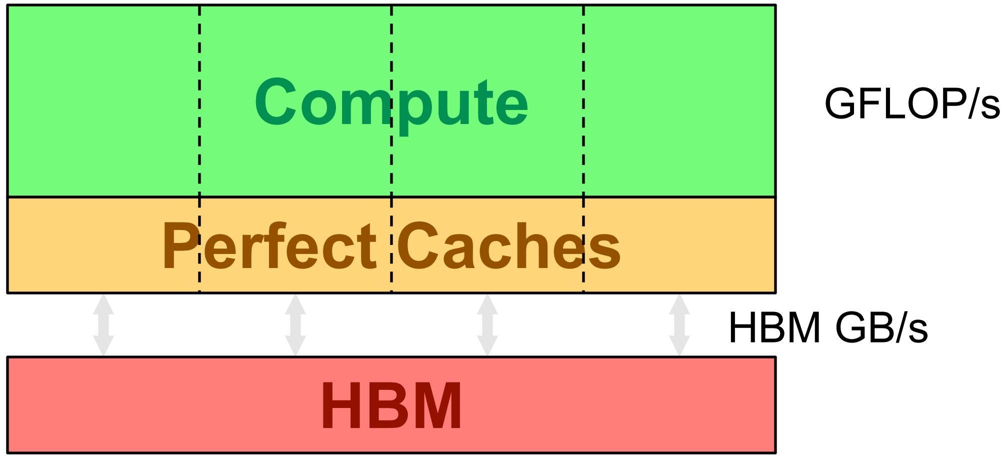
```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

  - 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

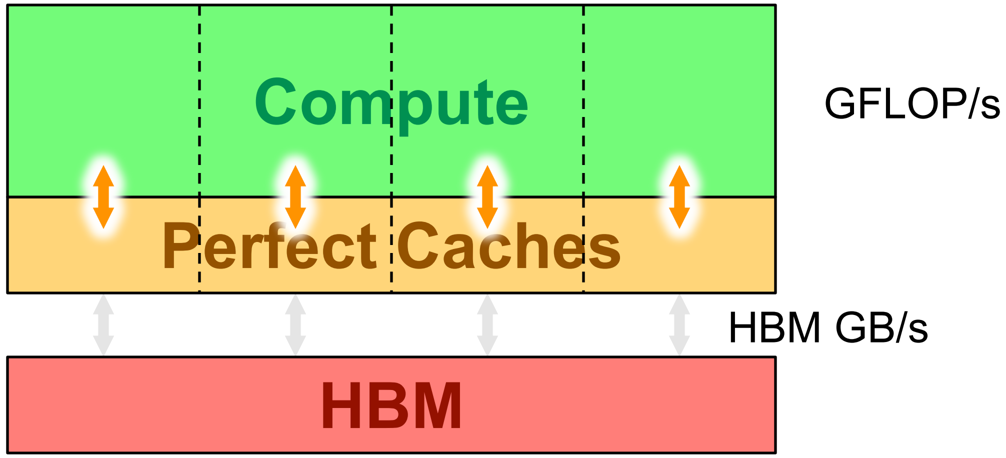
- Conversely, 7-point constant coefficient stencil…

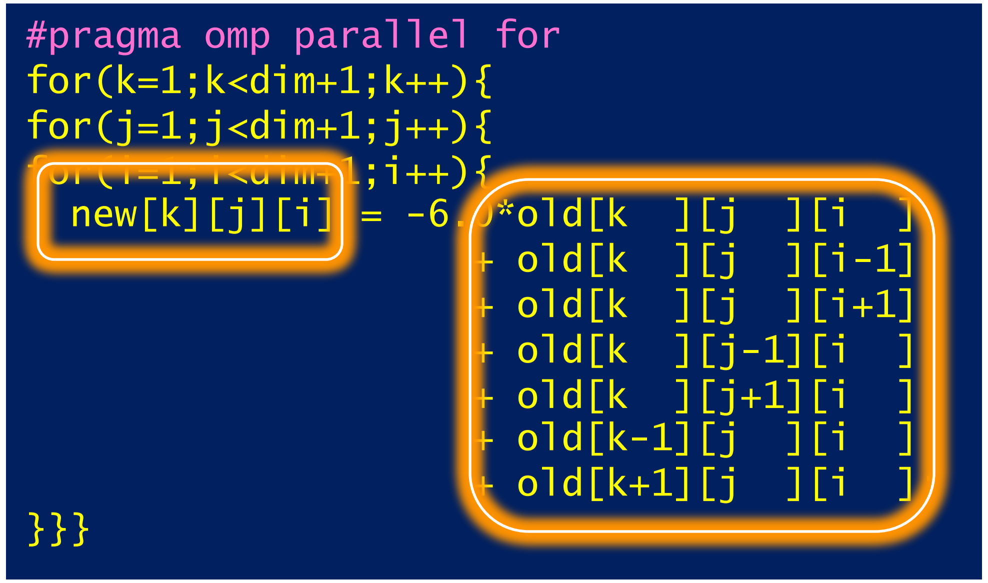


```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                    + old[k  ][j  ][i-1]
                    + old[k  ][j  ][i+1]
                    + old[k  ][j-1][i  ]
                    + old[k  ][j+1][i  ]
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

Roofline Example #2

- Conversely, 7-point constant coefficient stencil…
  - ○ 7 FLOPs
  - ○ 8 memory references (7 reads, 1 store) per point
  - ○ **AI = 7 / (8*8) = 0.11 FLOPs per byte (measured at the L1)**

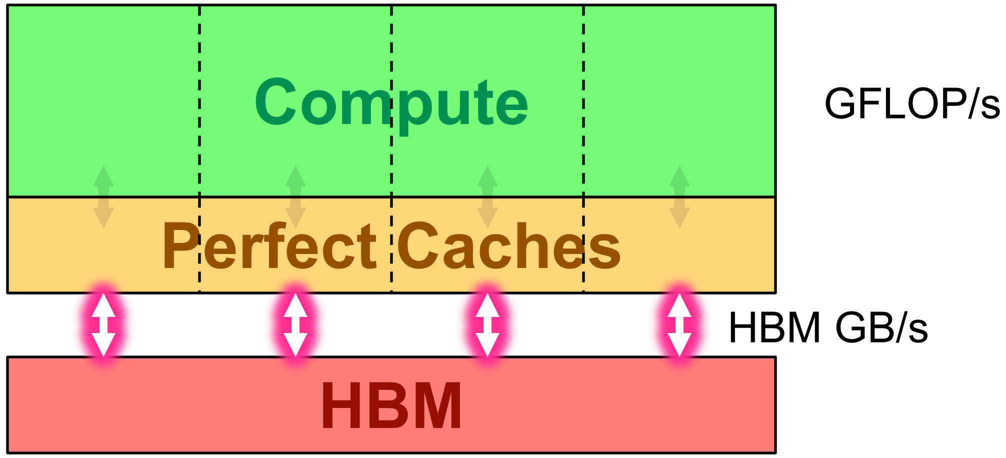




```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                    + old[k  ][j  ][i-1]
                    + old[k  ][j  ][i+1]
                    + old[k  ][j-1][i  ]
                    + old[k  ][j+1][i  ]
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

Roofline Example #2

- Conversely, 7-point constant coefficient stencil…
  - 7 FLOPs
  - 8 memory references (7 reads, 1 store) per point
  - **Ideally, cache will filter all but 1 read and 1 write per point**

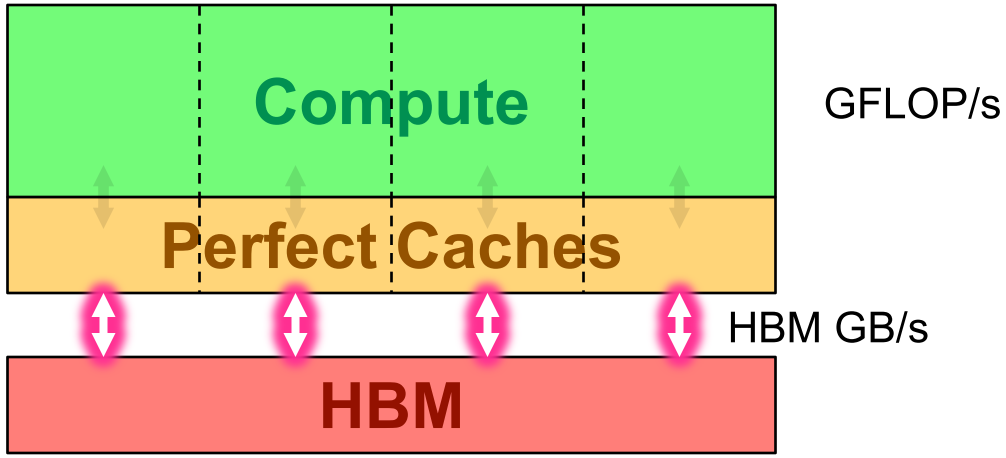


```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                    + old[k  ][j  ][i-1]
                    + old[k  ][j  ][i+1]
                    + old[k  ][j-1][i  ]
                    + old[k  ][j+1][i  ]
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

Roofline Example #2

- Conversely, 7-point constant coefficient stencil…
  - 7 FLOPs
  - 8 memory references (7 reads, 1 store) per point
  - Ideally, cache will filter all but 1 read and 1 write per point
  - ➢ **7 / (8+8) = 0.44 FLOPs per byte (DRAM)**

```
#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  ];
}}}
```

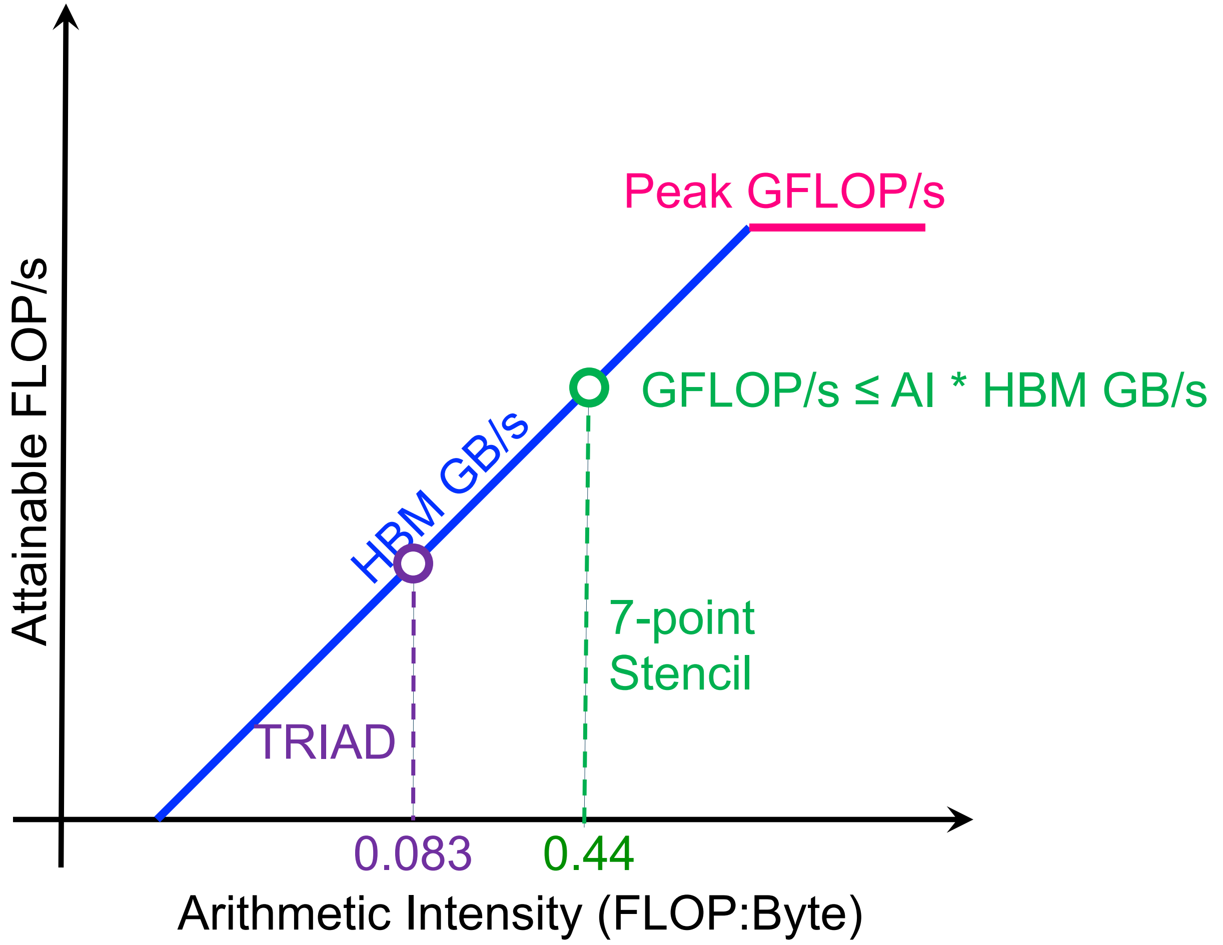


Roofline Example #2

- Conversely, 7-point constant coefficient stencil…
  - 7 FLOPs
  - 8 memory references (7 reads, 1 store) per point
  - Ideally, cache will filter all but 1 read and 1 write per point
  - ➢ **7 / (8+8) = 0.44 FLOPs per byte (DRAM)**

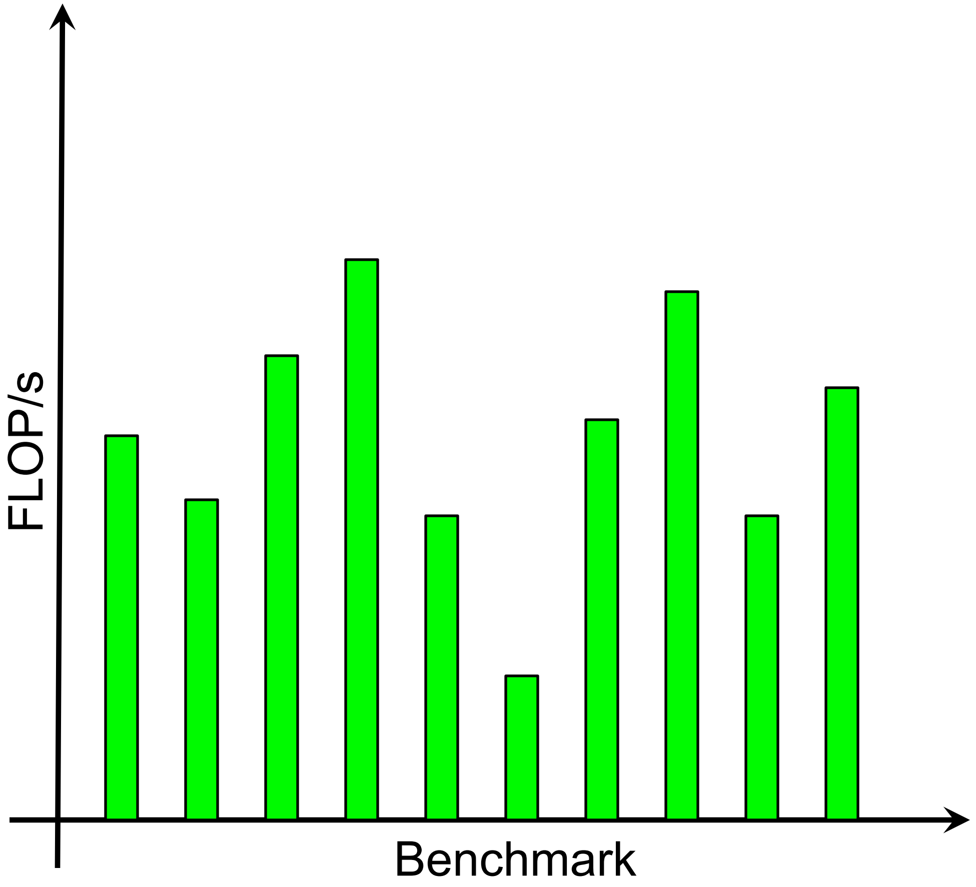
    == memory bound, but 5x the FLOP rate as TRIAD

```
#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  ];
}}}
```



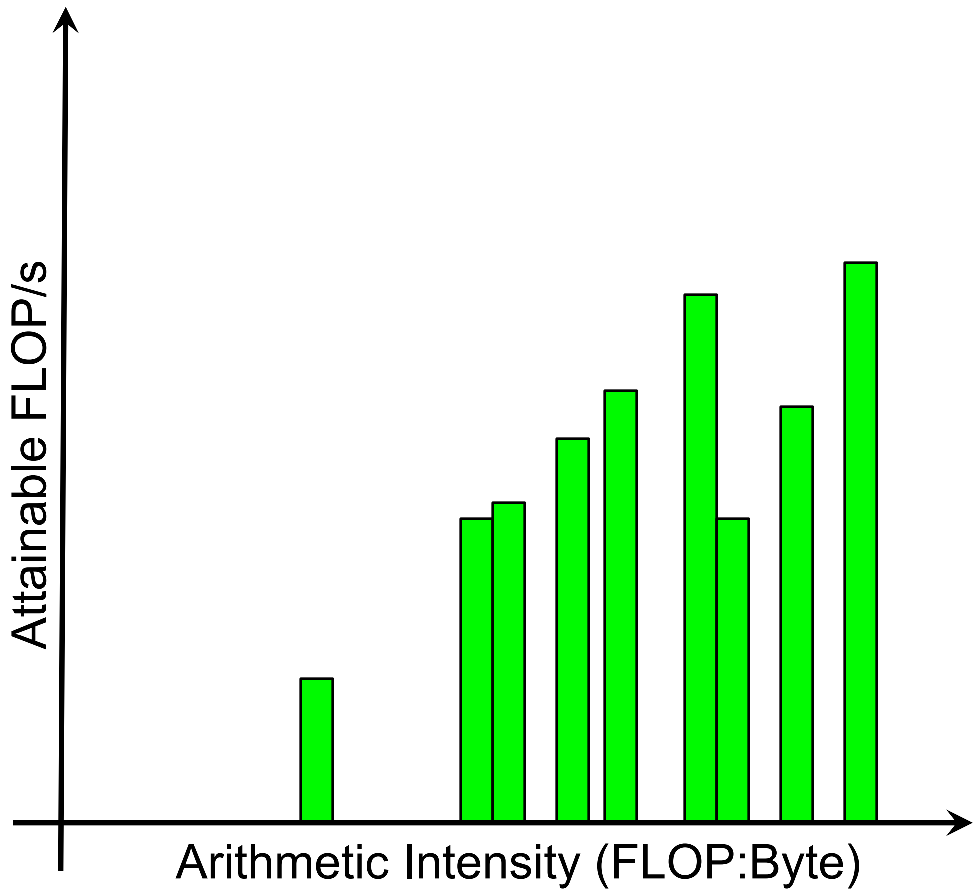
Are we getting good performance?

- Think back to our mix of benchmarks…



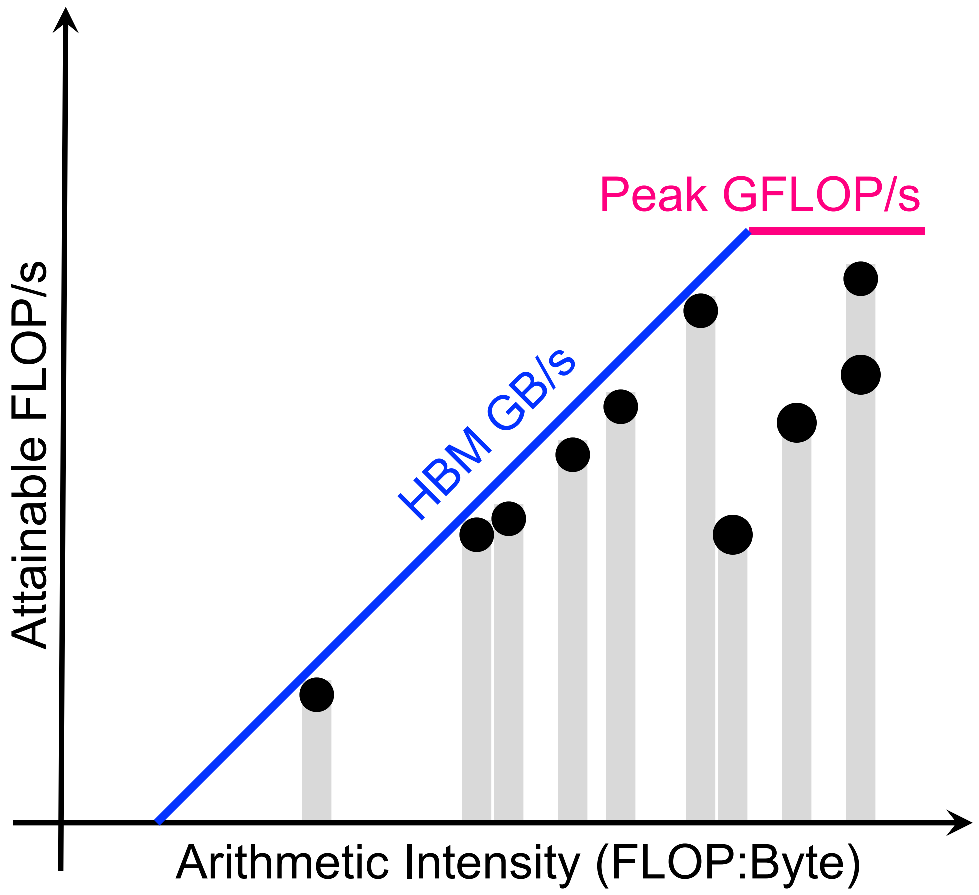
Are we getting good performance?

- We can sort benchmarks by arithmetic intensity…



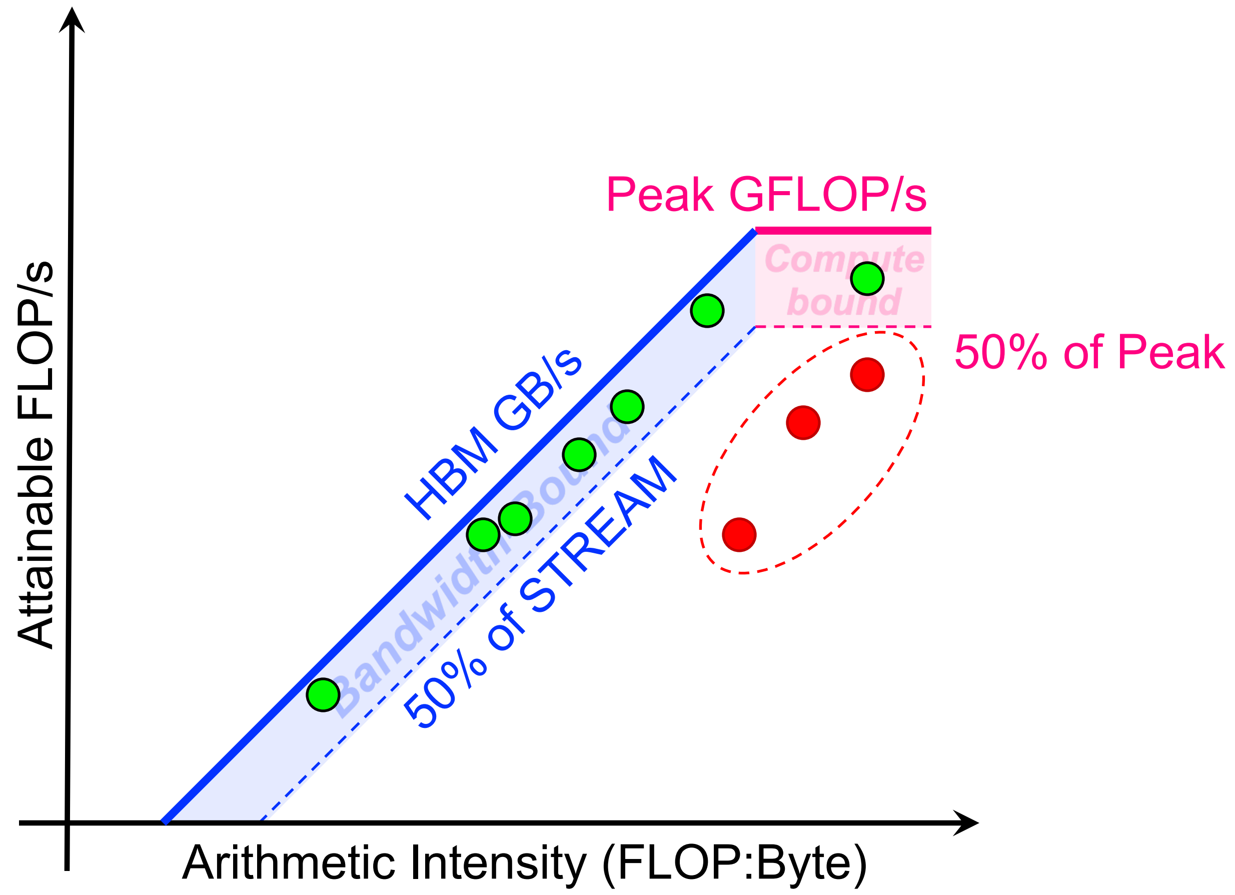
Are we getting good performance?

- We can sort benchmarks by arithmetic intensity…
- … and compare performance relative to machine capabilities



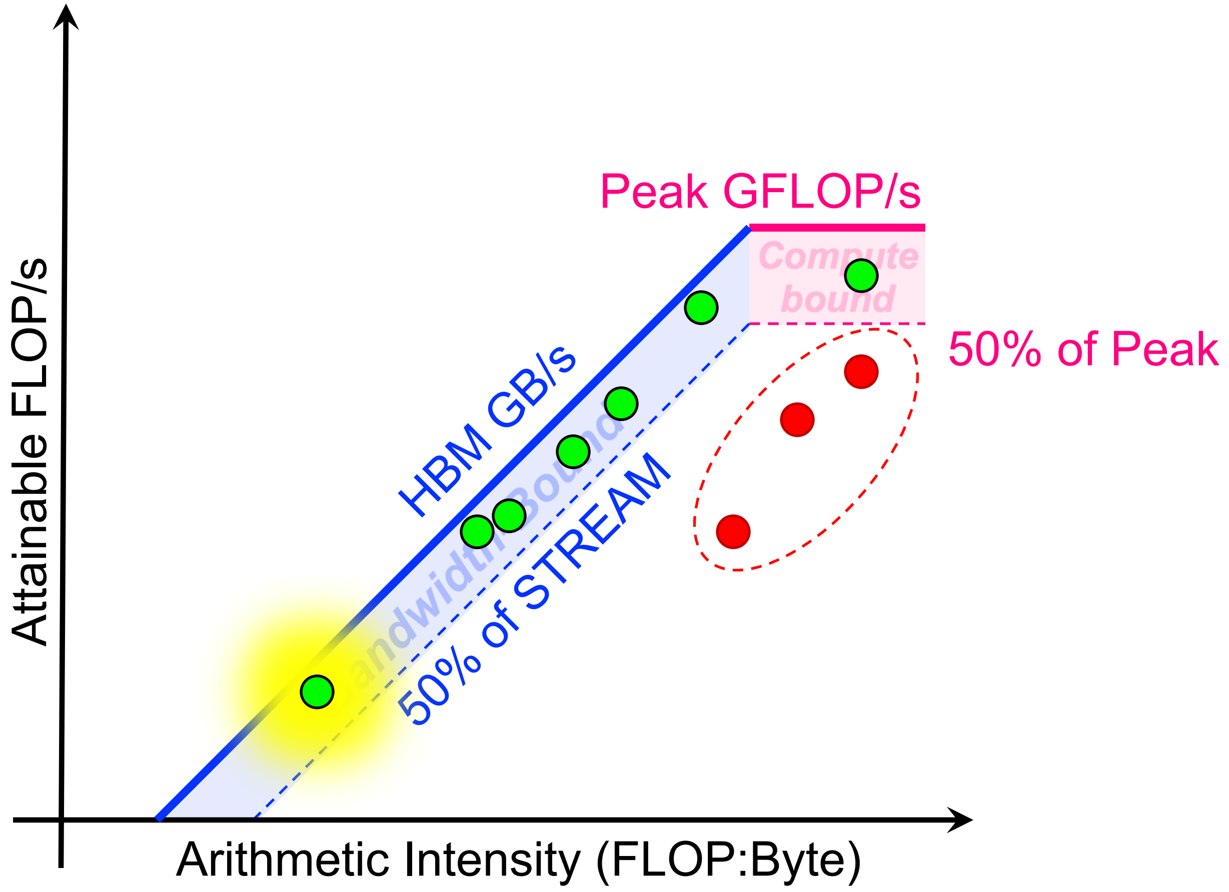
Are we getting good performance?

- Benchmarks near the roofline are making **good use** of computational resources



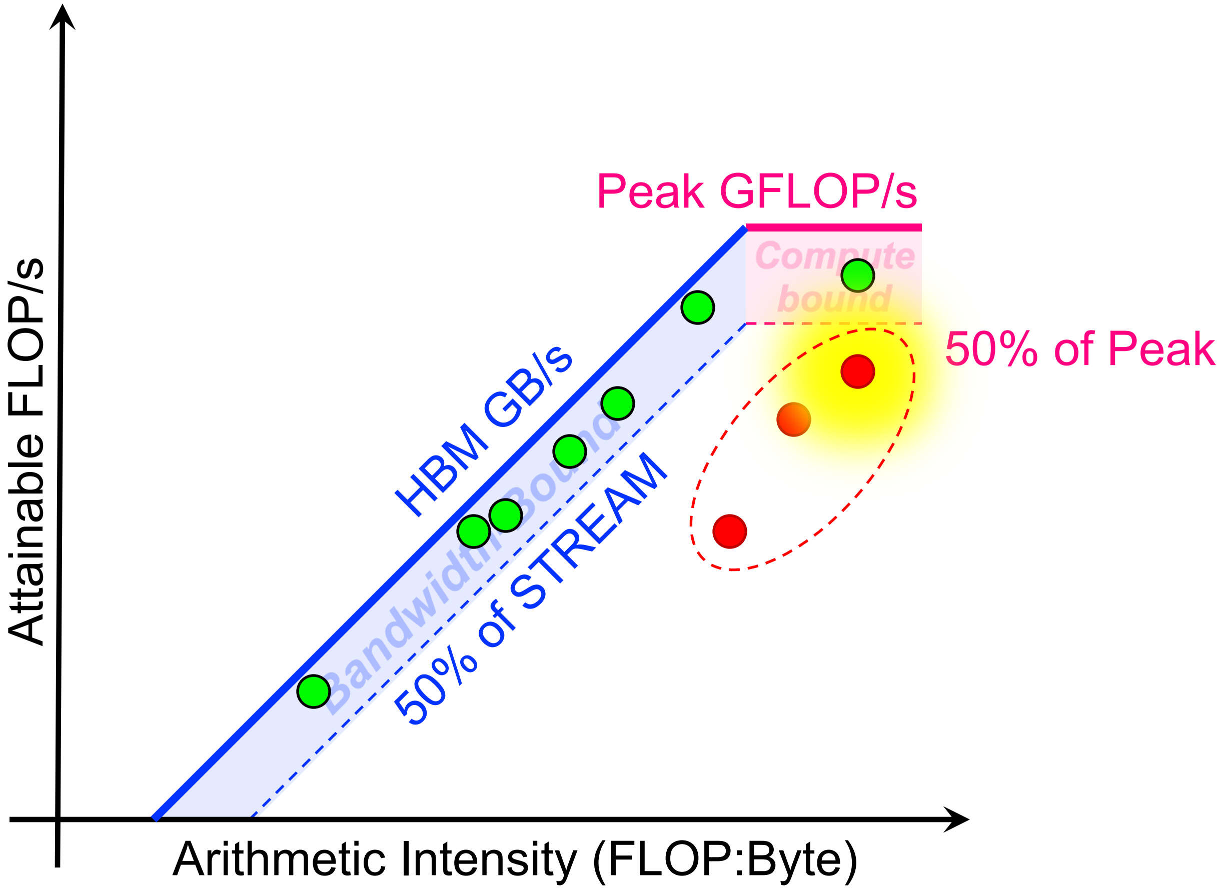
Are we getting good performance?

- Benchmarks near the roofline are making **good use** of computational resources
  - ➢ benchmarks can have **low performance** (GFLOP/s), but make **good use** (%STREAM) of a machine



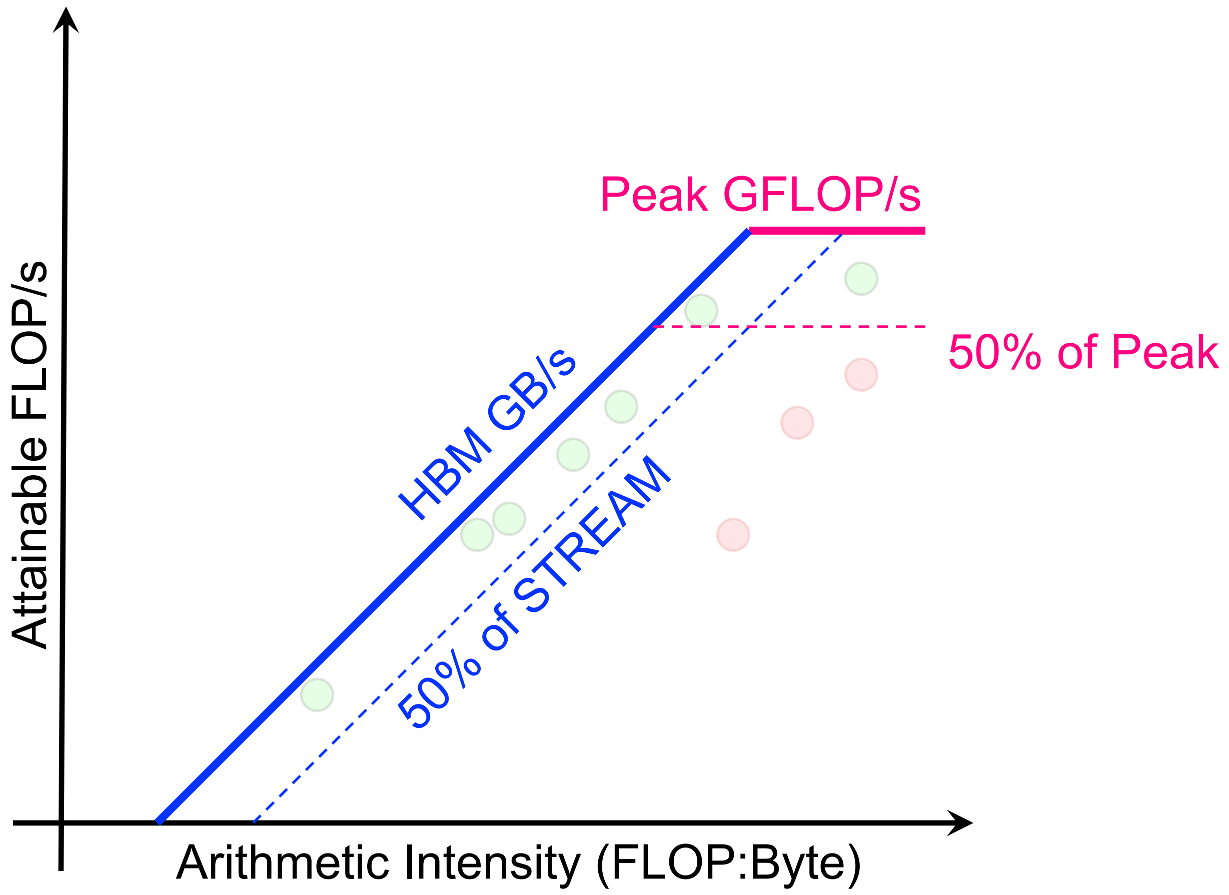
Are we getting good performance?

- Benchmarks near the roofline are making **good use** of computational resources
  - benchmarks can have **low performance** (GFLOP/s), but make **good use** (%STREAM) of a machine
  - benchmarks can have **high performance** (GFLOP/s), but still make **poor use** of a machine (%peak)



Recap: Roofline is made of two components

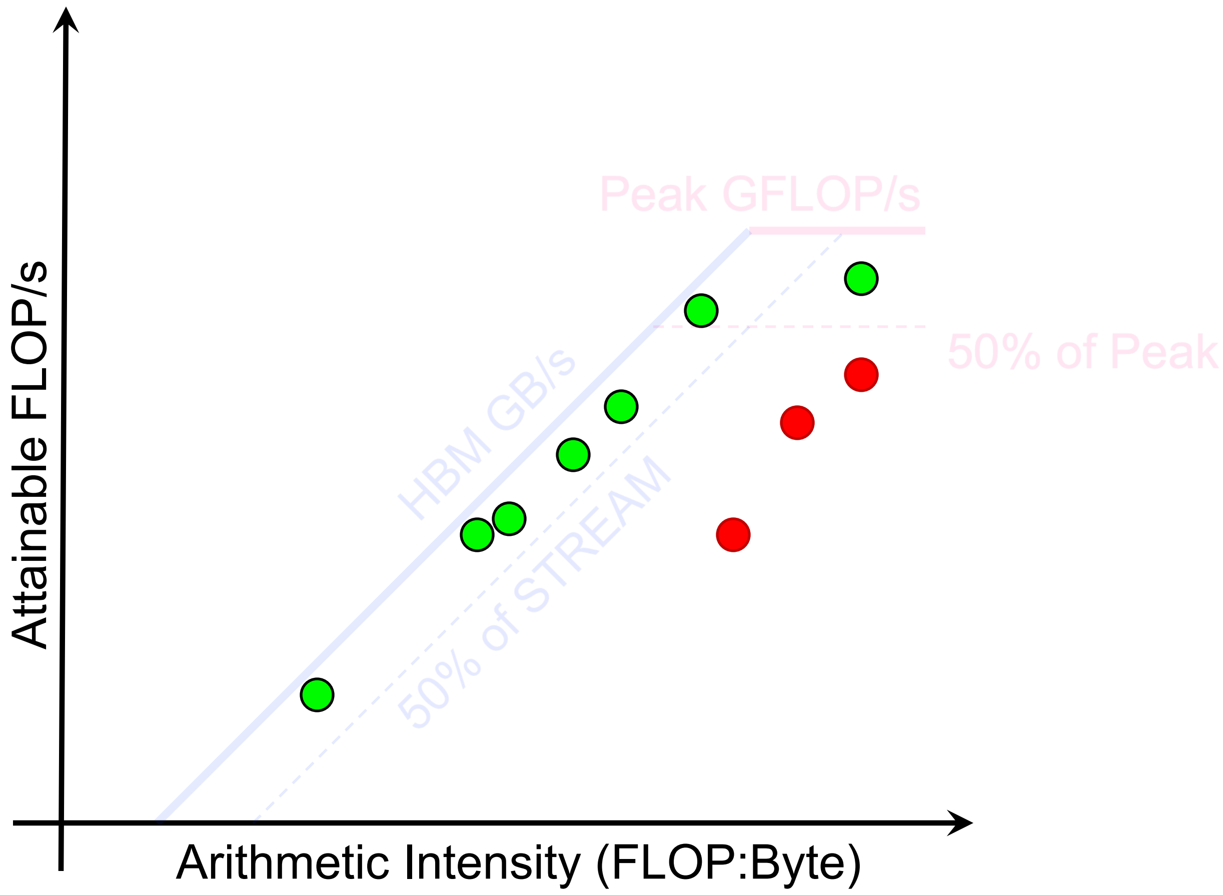
- Machine Model
  - Lines defined by peak GB/s and GF/s (**Benchmarking**)
  - Unique to each architecture
  - Common to all apps on that architecture



Recap: Roofline is made of two components

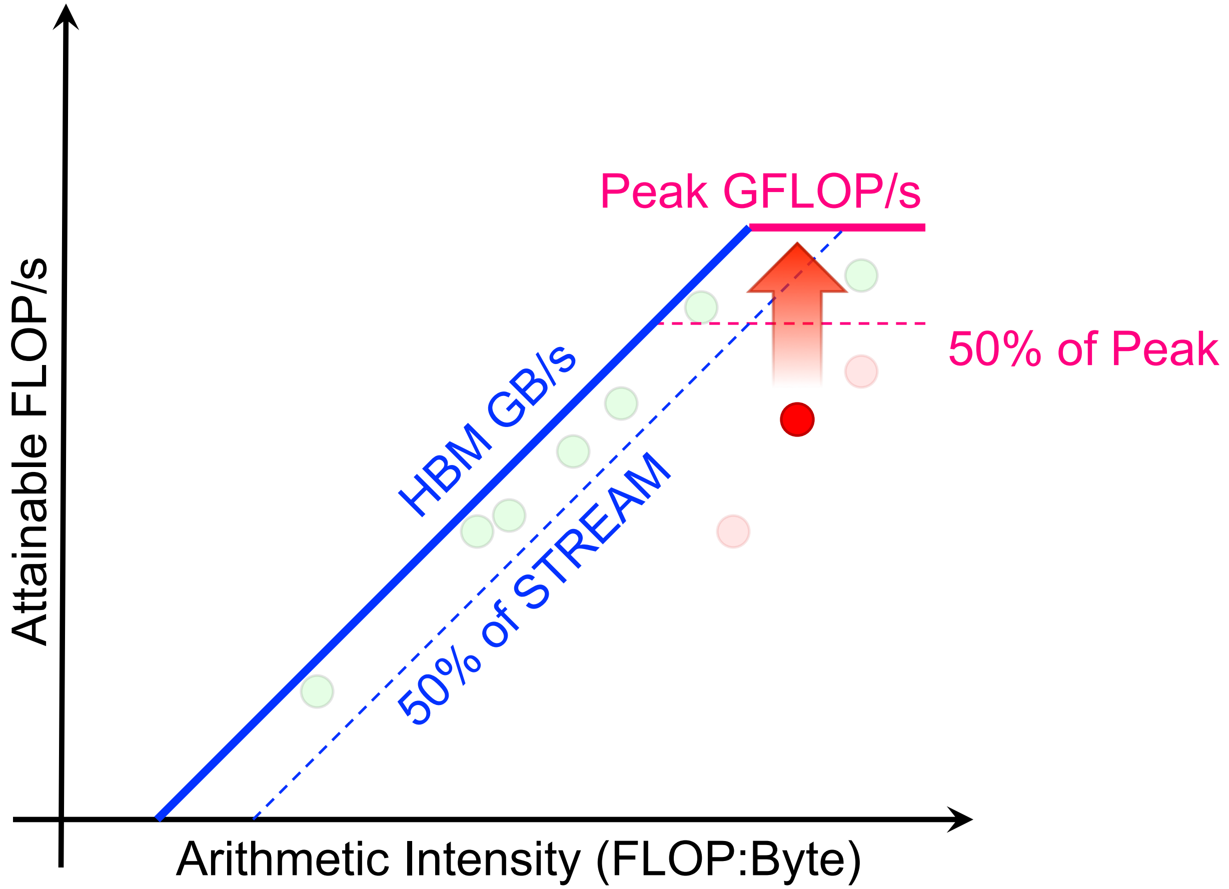
- Machine Model
  - Lines defined by peak GB/s and GF/s (**Benchmarking**)
  - Unique to each architecture
  - Common to all apps on that architecture
- Application Characteristics
  - Dots defined by application GFLOPs, GBs, and run time

    (**Application Instrumentation**)
  - Unique to each application
  - Unique to each architecture



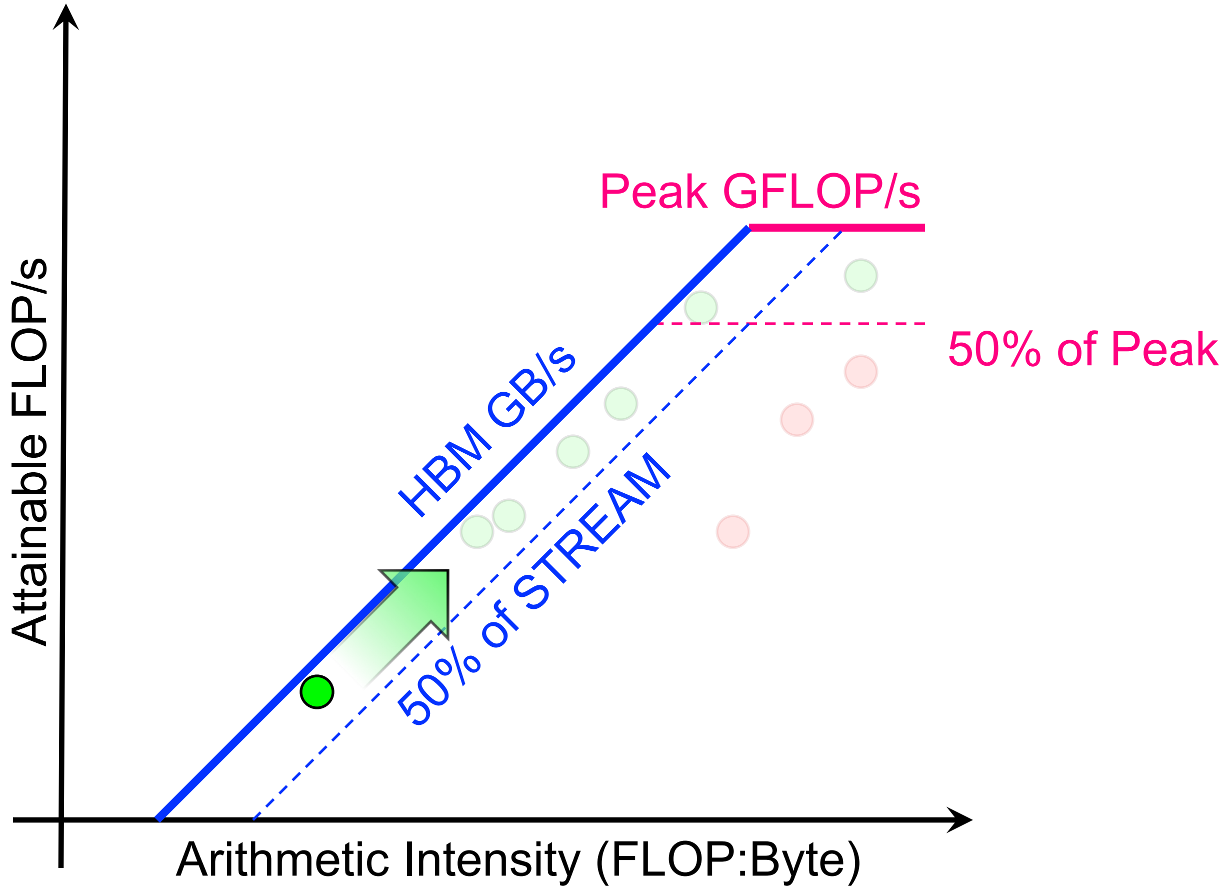
Recap: Optimization Strategy

1. Get to the Roofline



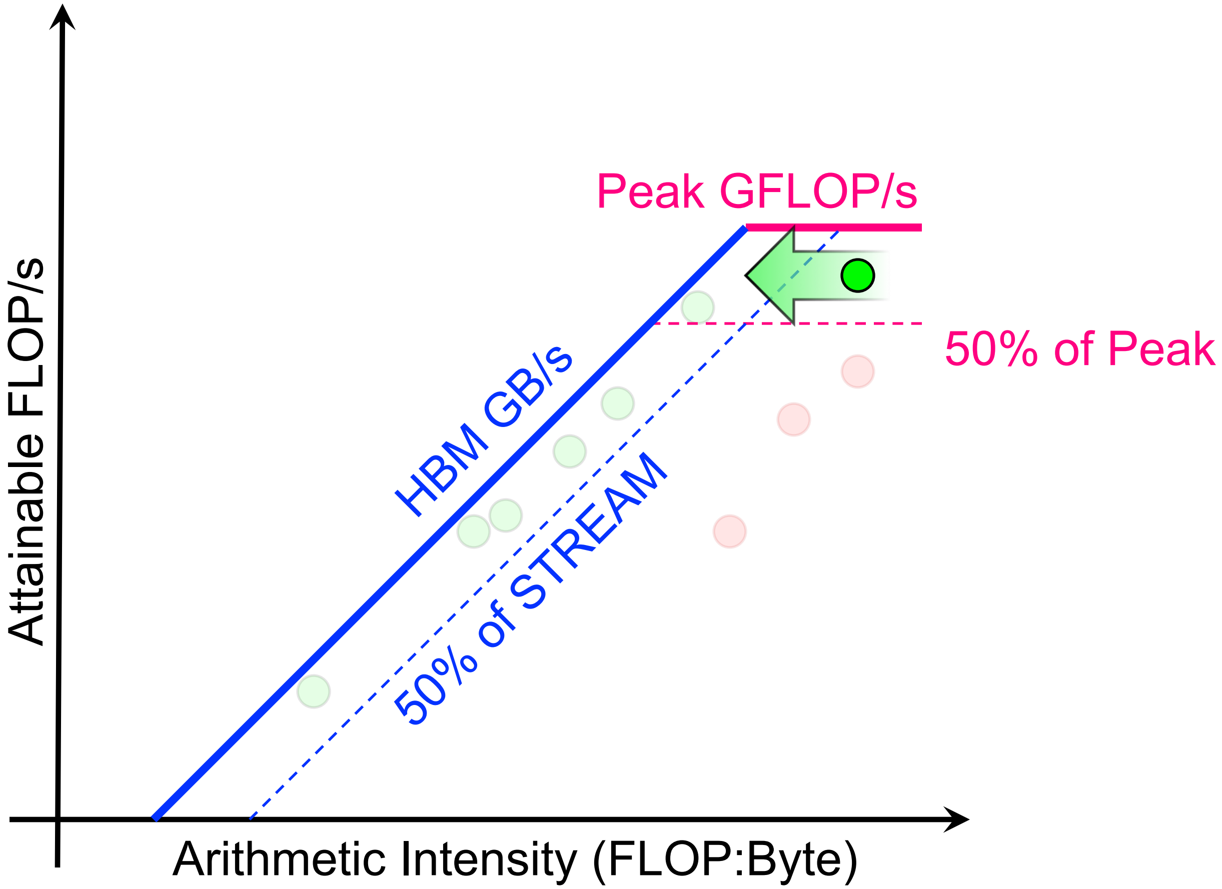
Recap: Optimization Strategy

1. Get to the Roofline
2. Reduce Data movement when bandwidth-limited
   - Bandwidth-bound implies run time is tied to data movement and peak GB/s.
   - ***Optimizations that reduce data movement will improve performance***



Recap: Optimization Strategy

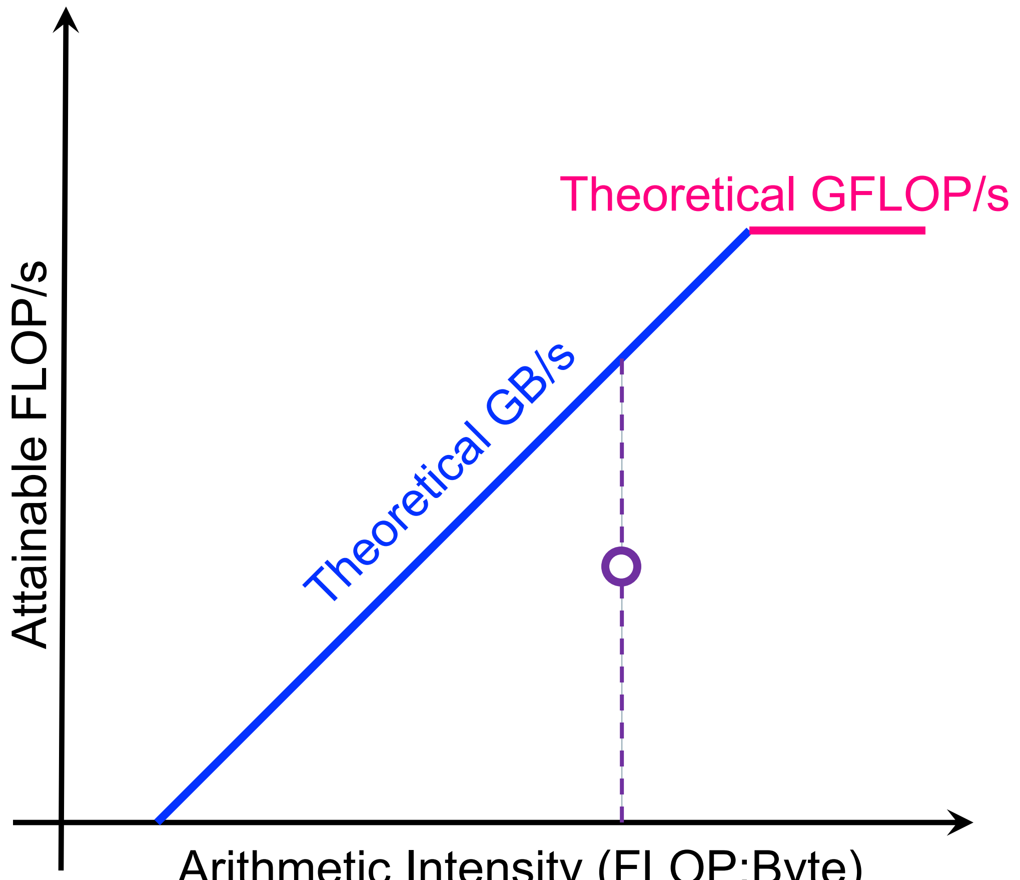
1. Get to the Roofline
2. Reduce Data movement when bandwidth-limited
   - Bandwidth-bound implies run time is tied to data movement and peak GB/s.
   - Optimizations that reduce data movement will improve performance
3. Reduce the number of #FLOPs when compute-bound
   - Compute bound implies run time is tied to #FLOPs and peak GFLOP/s
   - ***Optimizations that eliminate FLOPs will improve time-to-solution (but may reduce GFLOP/s)***
   - Subtlety, this will reduce AI, but increase performance



How can performance ever be below the Roofline?

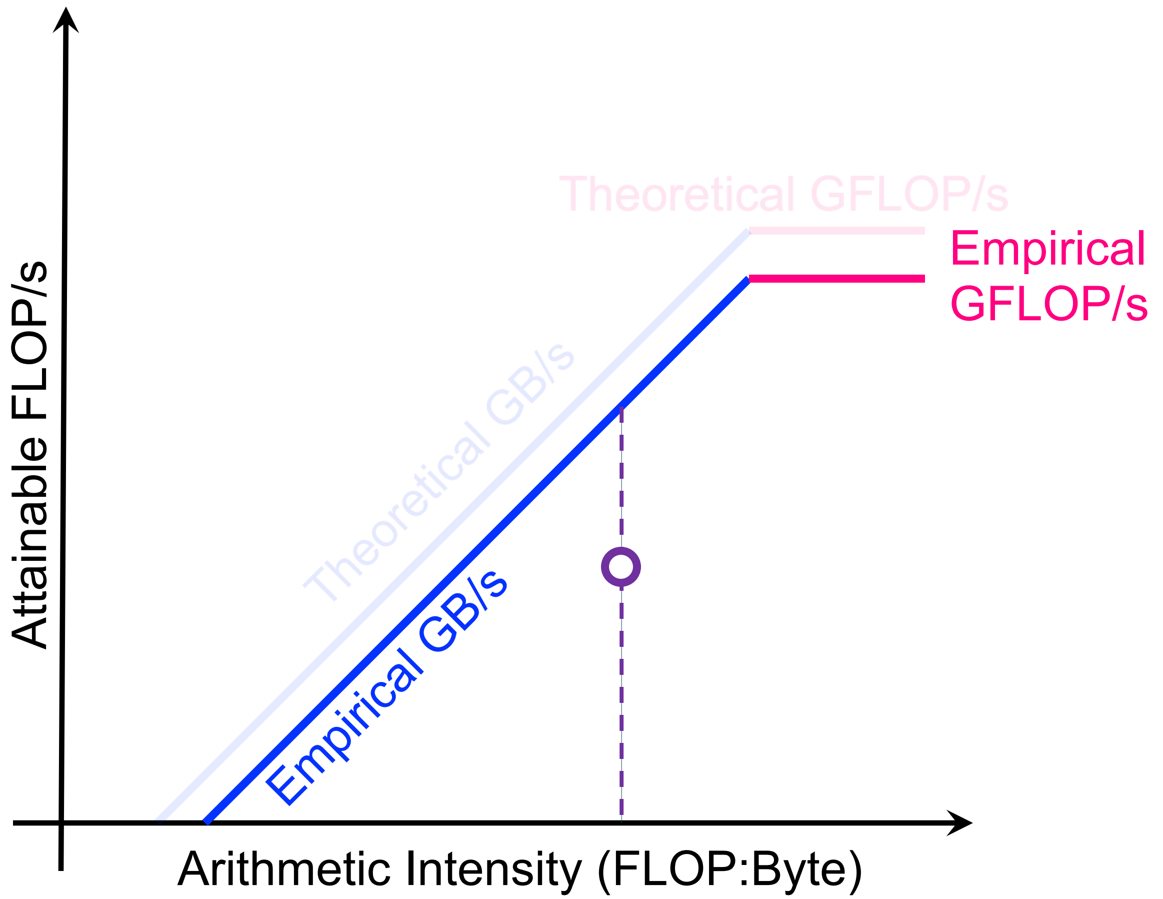
Theoretical vs. Empirical

- Theoretical Roofline:
  - Pin bandwidth == bits * GHz
  - Peak FLOPs == FPUs * GHz
  - 1 C++ FLOP = 1 ISA FLOP
  - Data movement = Compulsory Misses



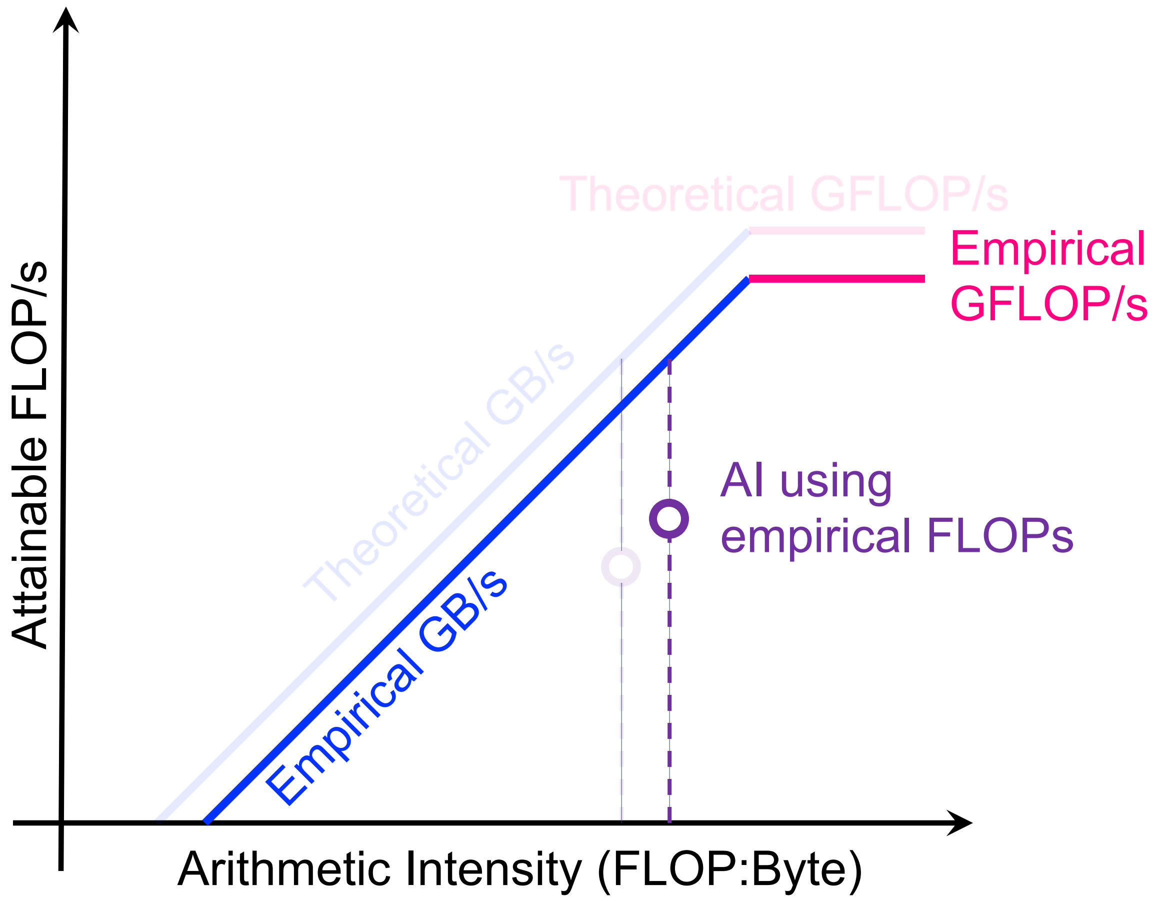
Theoretical vs. Empirical / Benchmarking

- Theoretical Roofline:
  - Pin bandwidth == bits * GHz
  - Peak FLOPs == FPUs * GHz
  - 1 C++ FLOP = 1 ISA FLOP
  - Data movement = Compulsory Misses
- Empirical Roofline:
  - Empirical bandwidth (STREAM) <= theoretical
  - Empirical peak FLOP/s <= theoretical



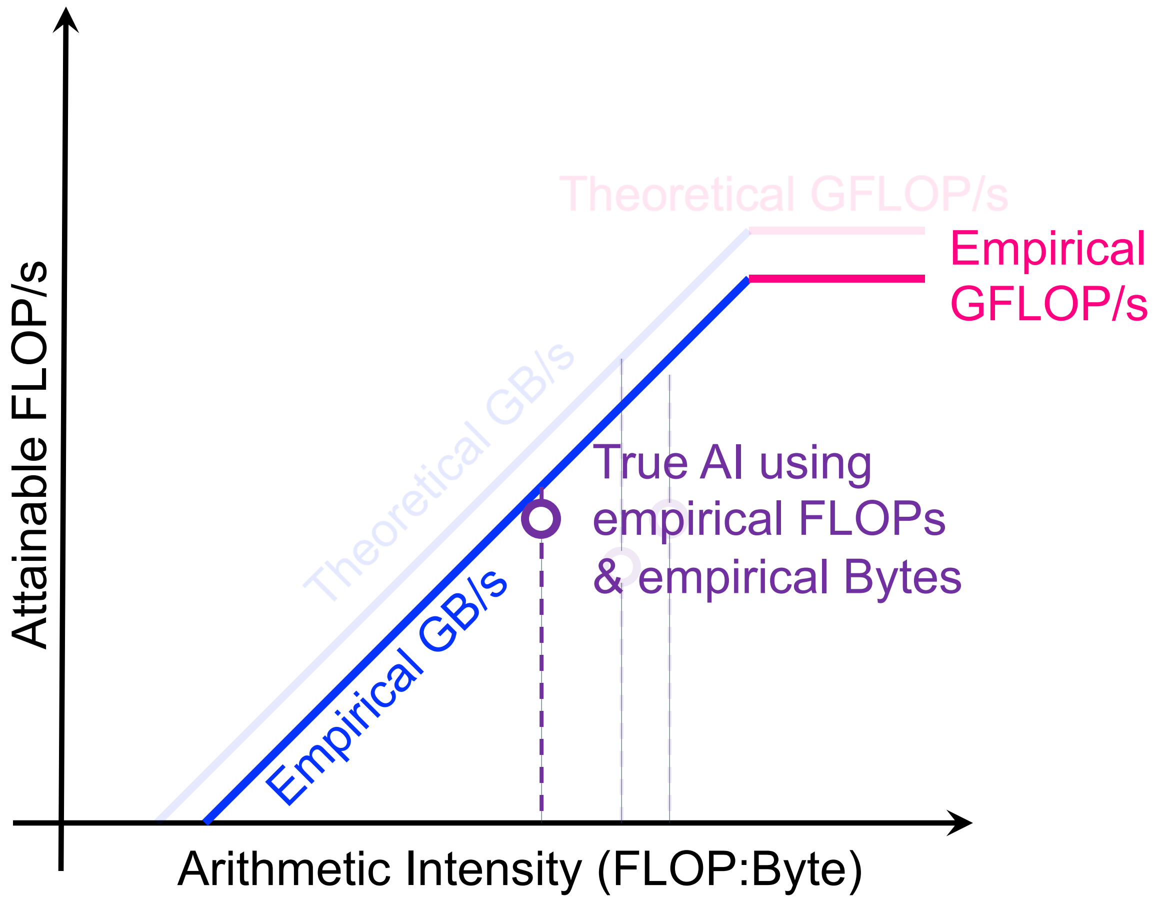
Theoretical vs. Empirical / FLOPs

- Theoretical Roofline:
  - Pin bandwidth == bits * GHz
  - Peak FLOPs == FPUs * GHz
  - 1 C++ FLOP = 1 ISA FLOP
  - Data movement = Compulsory Misses
- Empirical Roofline:
  - Empirical bandwidth (STREAM) <= theoretical
  - Empirical peak FLOP/s <= theoretical
  - 1 C++ FLOP >= 1 ISA FLOP (e.g. divide)



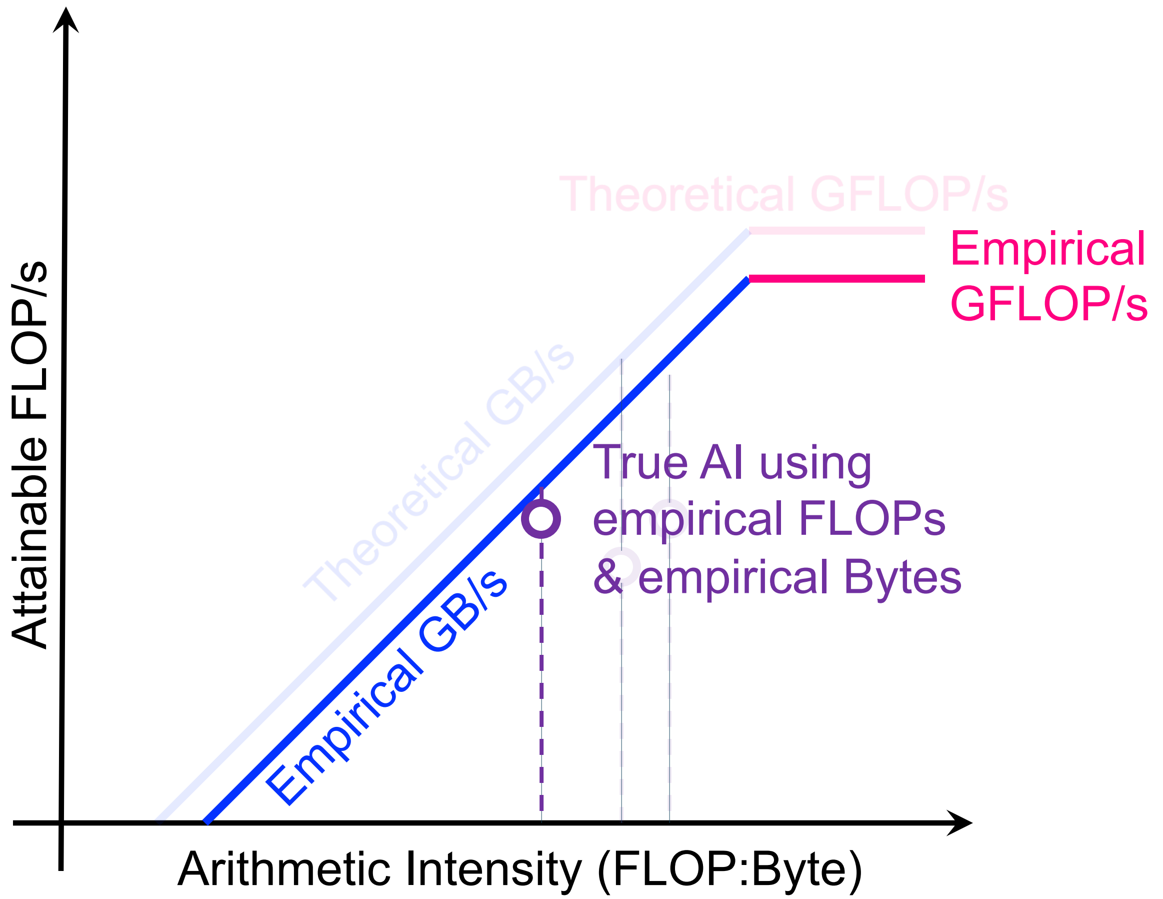
Theoretical vs. Empirical / Bytes

- Theoretical Roofline:
  - Pin bandwidth == bits * GHz
  - Peak FLOPs == FPUs * GHz
  - 1 C++ FLOP = 1 ISA FLOP
  - Data movement = Compulsory Misses
- Empirical Roofline:
  - Empirical bandwidth (STREAM) <= theoretical
  - Empirical peak FLOP/s <= theoretical
  - 1 C++ FLOP >= 1 ISA FLOP (e.g. divide)
  - Data movement >> Compulsory Misses



Theoretical vs. Empirical / Bytes

- Theoretical Roofline:
  - Pin bandwidth == bits * GHz
  - Peak FLOPs == FPUs * GHz
  - 1 C++ FLOP = 1 ISA FLOP
  - Data movement = Compulsory Misses
- Empirical Roofline:
  - Empirical bandwidth (STREAM) <= theoretical
  - Empirical peak FLOP/s <= theoretical
  - 1 C++ FLOP >= 1 ISA FLOP (e.g. divide)
  - Data movement >> Compulsory Misses



- **Use benchmarking tools to construct the Roofline model (ceilings)**
- **Use Profiling tools to populate the Roofline model (dots)**

How else can performance be below the Roofline?

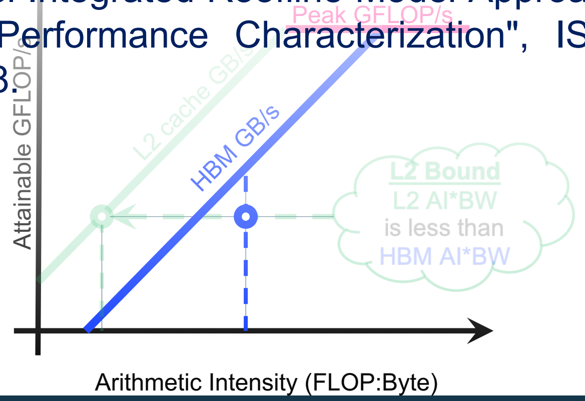
Simple DRAM model can be insufficient for a variety of reasons…

DRAM’s not the bottleneck…

- Cache bandwidth and cache locality
- PCIe bandwidth

…The Hierarchical Roofline Model

T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, S. Williams, "A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization", ISC, 2018.

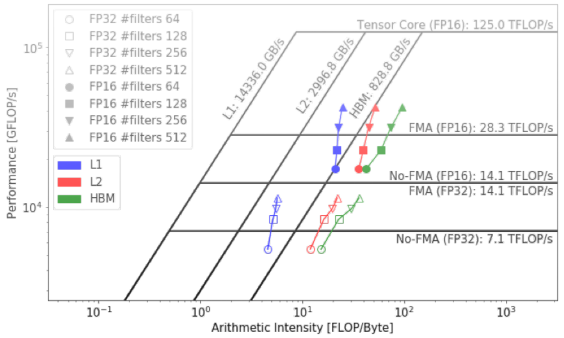


Not enough of Vector/Tensor instr.

- No FMA
- Mixed Precision
- No Tensor Core OPs

… Additional Ceilings

C. Yang, T. Kurth, S. Williams, "Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system", CCPE, 2019.

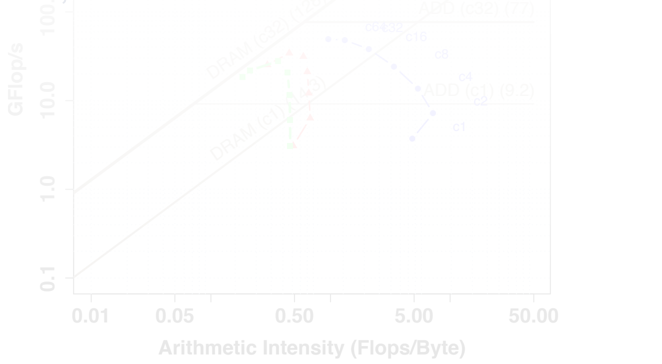


Lack of Parallelism…

- Idle Cores/SMs
- Insufficient ILP/TLP
- Divergence and Predication

… Roofline Scaling Trajectories

K. Ibrahim, S. Williams, L. Oliker, "Performance Analysis of GPU Programming Models using the Roofline Scaling Trajectories", BEST PAPER, Bench, 2019.

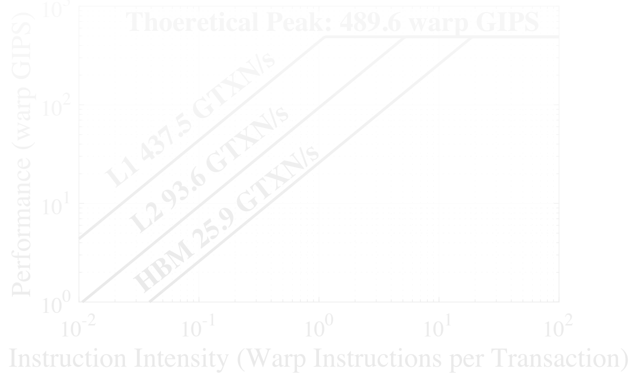


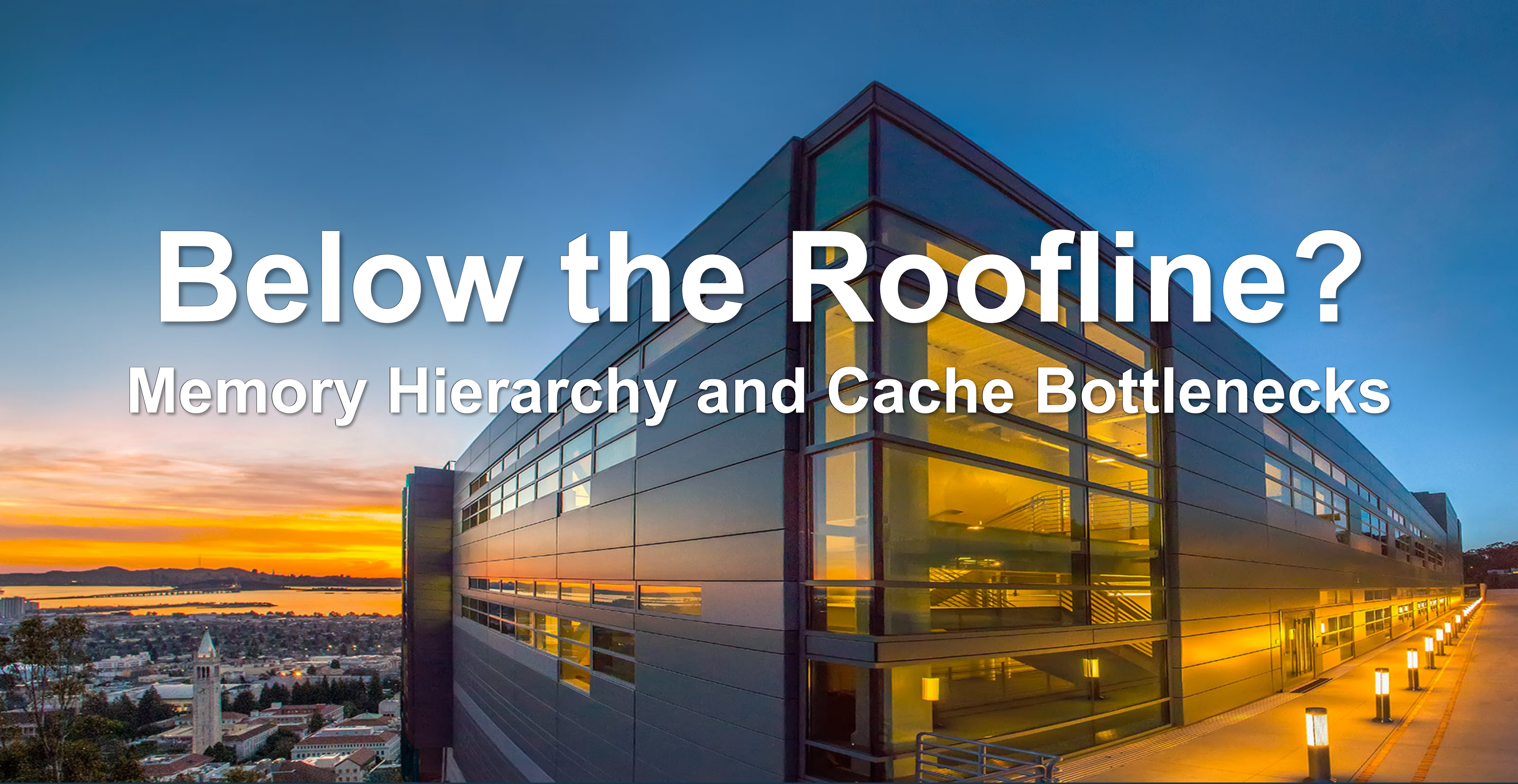
Integer-heavy Codes…

- Non-FP inst. impede FLOPs
- No FP instructions

… The Instruction Roofline Model

N. Ding, S. Williams, "An Instruction Roofline Model for GPUs", BEST PAPER, PMBS, 2019.



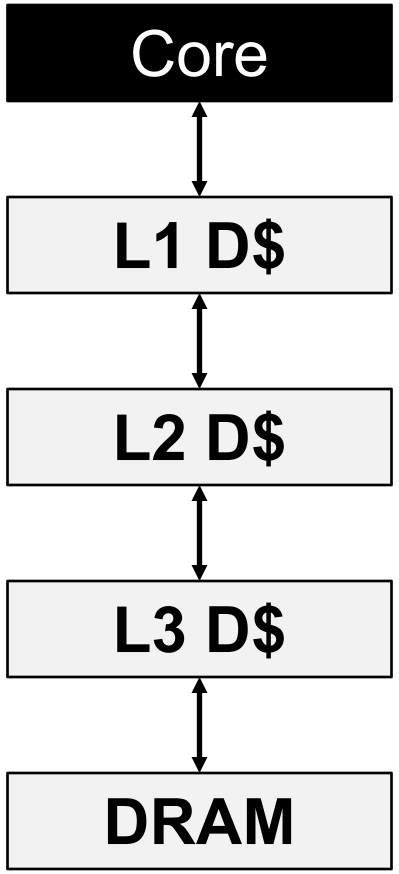


Below the Roofline?

Memory Hierarchy and Cache Bottlenecks

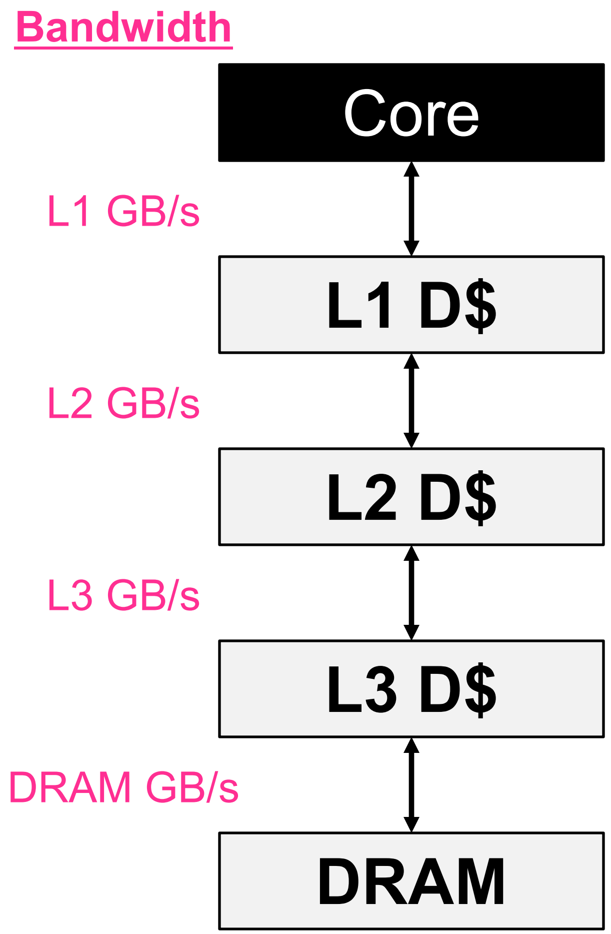
Memory Hierarchy

- CPUs/GPUs have multiple levels of memory/cache
  - Registers
  - L1, L2, L3 cache
  - HBM/HBM (KNL/GPU device memory)
  - DDR (main memory)
  - NVRAM (non-volatile memory)



Memory Hierarchy

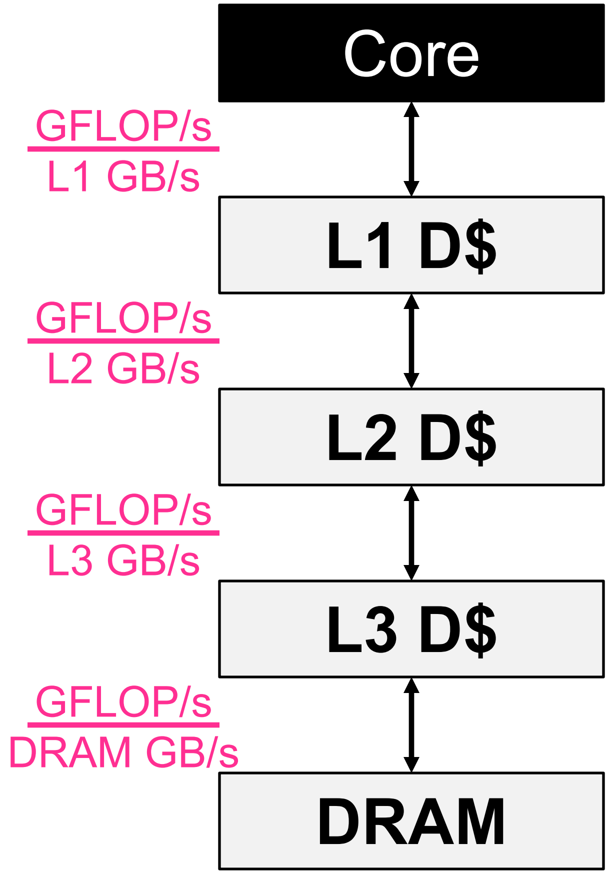
- CPUs/GPUs have different bandwidths for each level



Memory Hierarchy

- CPUs/GPUs have different bandwidths for each level
  - different **machine balances** for each level

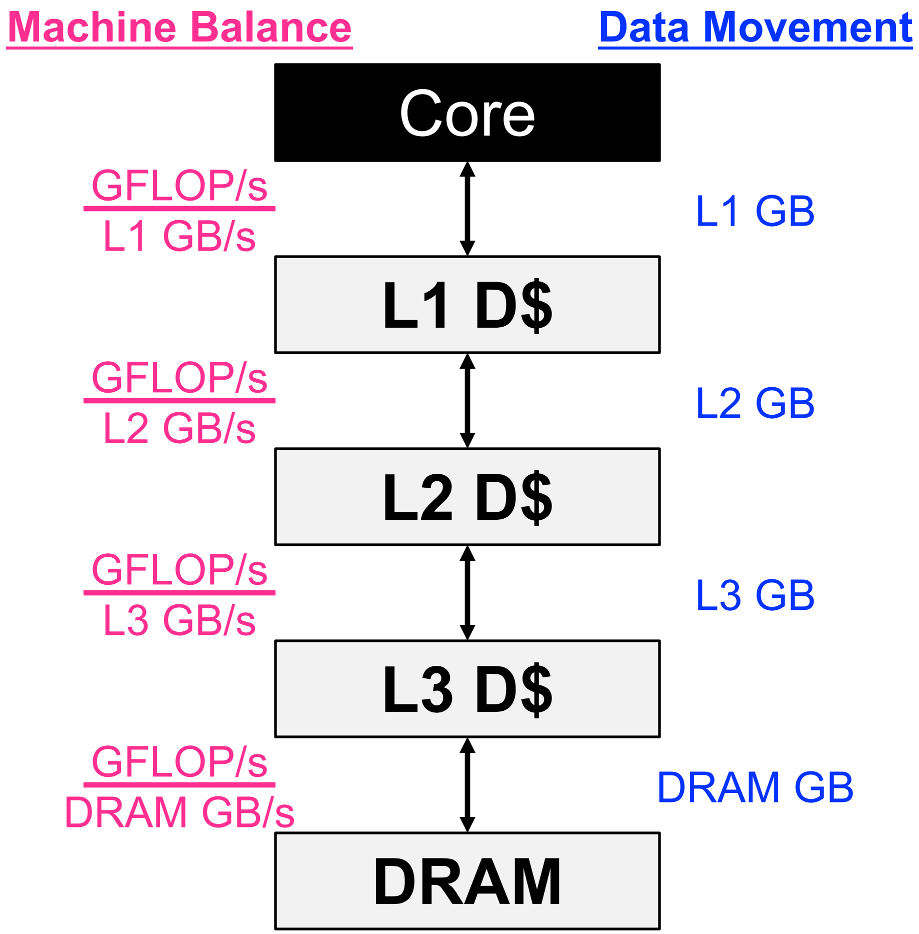
Machine Balance



Memory Hierarchy

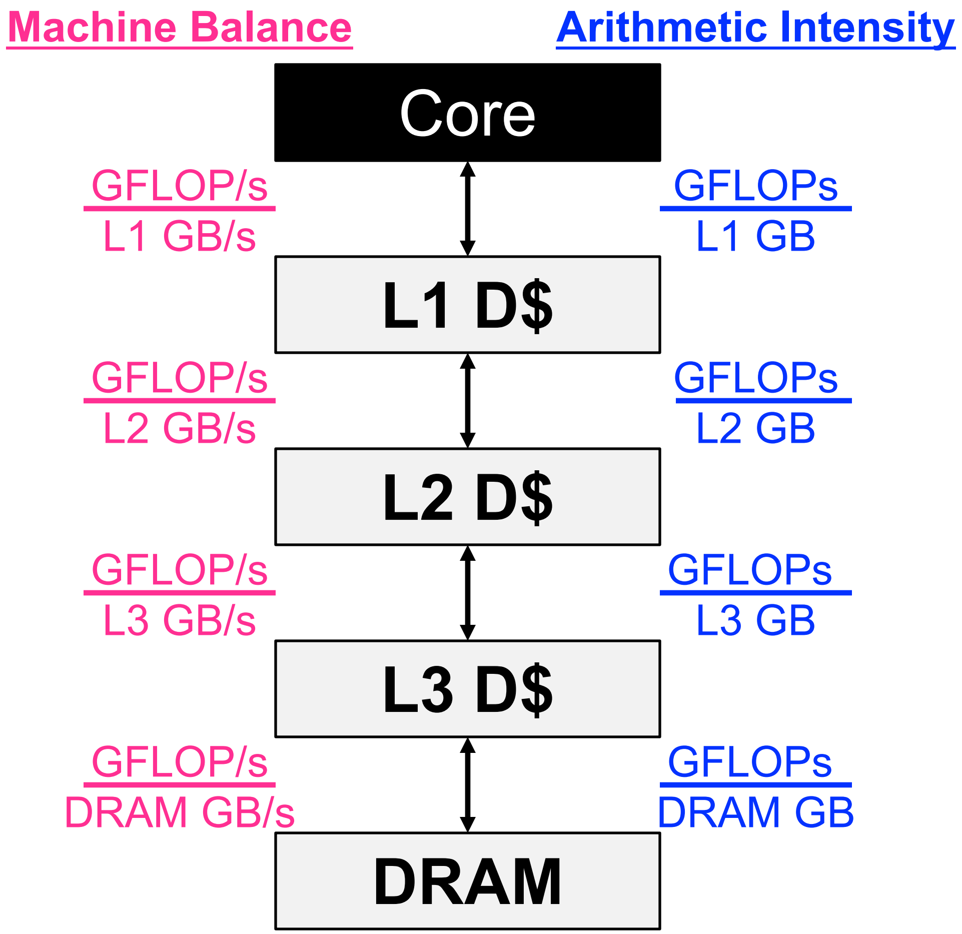
- CPUs/GPUs have different bandwidths for each level
  - different machine balances for each level

- Applications have locality in each level
  - different **data movements** for each level



Memory Hierarchy

- CPUs/GPUs have different bandwidths for each level
  - different machine balances for each level
- Applications have locality in each level
  - different data movements for each level
  - different **arithmetic intensity** for each level



Cache Bottlenecks

- For each additional level of the memory hierarchy, we can add another term to our model…

$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI}_{\text{DRAM}} * \text{DRAM GB/s} \end{cases}$$

AI_x (Arithmetic Intensity at level “x”) = FLOPs / Bytes (moved to/from level “x”)

Cache Bottlenecks

- For each additional level of the memory hierarchy, we can add another term to our model…

$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI}_{\text{DRAM}} * \text{DRAM GB/s} \\ \text{AI}_{\text{L2}} * \text{L2 GB/s} \end{cases}$$

AI_{x} (Arithmetic Intensity at level “x”) = FLOPs / Bytes (moved to/from level “x”)

Cache Bottlenecks

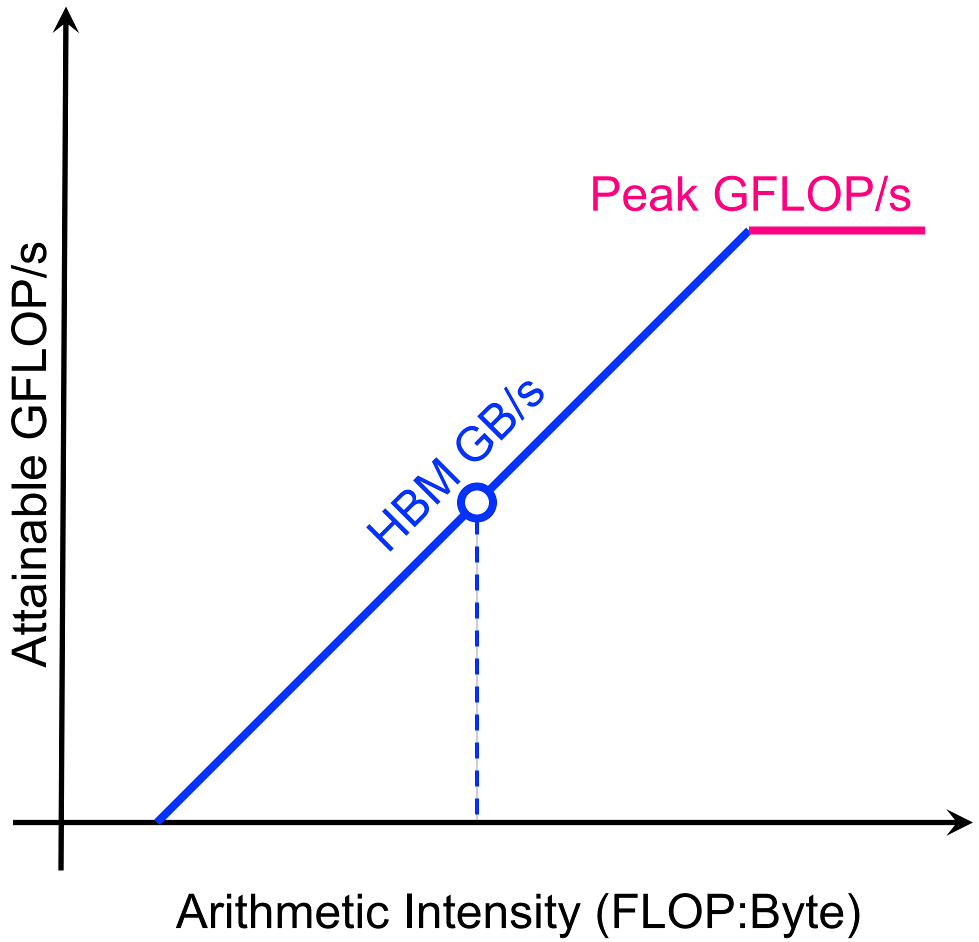
- For each additional level of the memory hierarchy, we can add another term to our model…

$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI}_{\text{DRAM}} * \text{DRAM GB/s} \\ \text{AI}_{\text{L2}} * \text{L2 GB/s} \\ \text{AI}_{\text{L1}} * \text{L1 GB/s} \end{cases}$$

AI_x (Arithmetic Intensity at level “x”) = FLOPs / Bytes (moved to/from level “x”)

Cache Bottlenecks

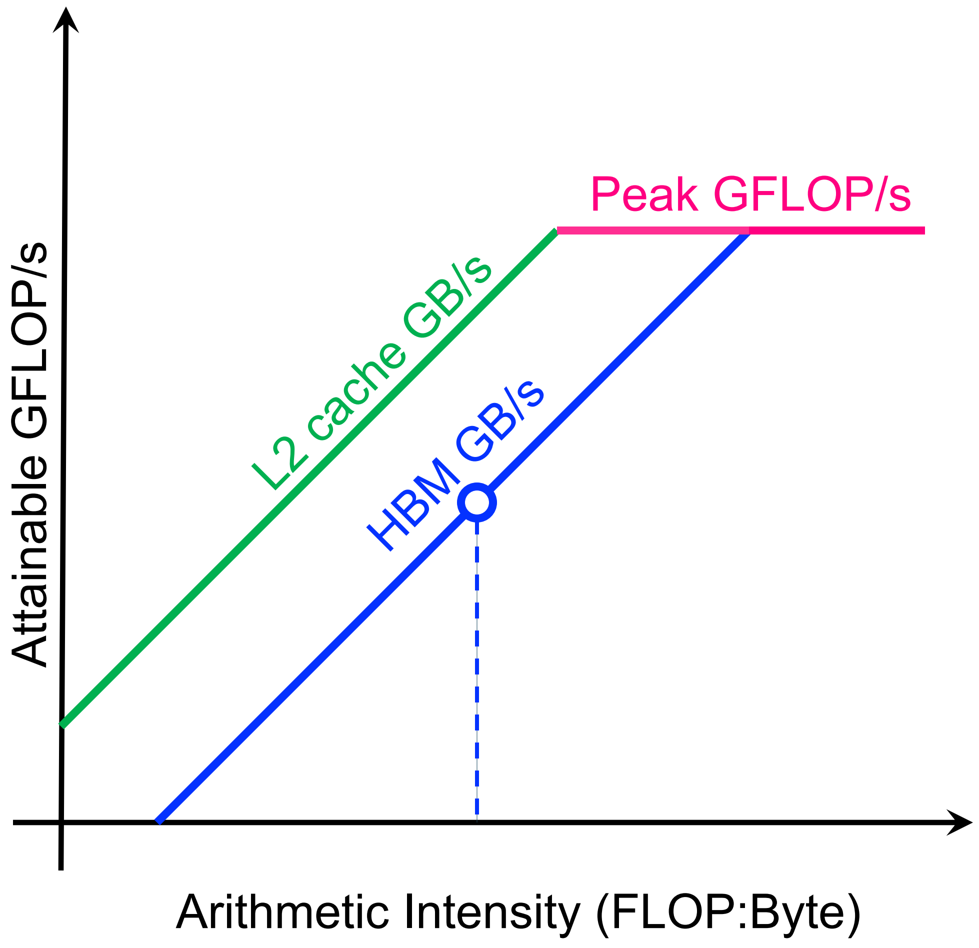
- Plot equation in a single figure…
  - “**Hierarchical Roofline**” Model



T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, S. Williams, "A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization", ISC, 2018.

Cache Bottlenecks

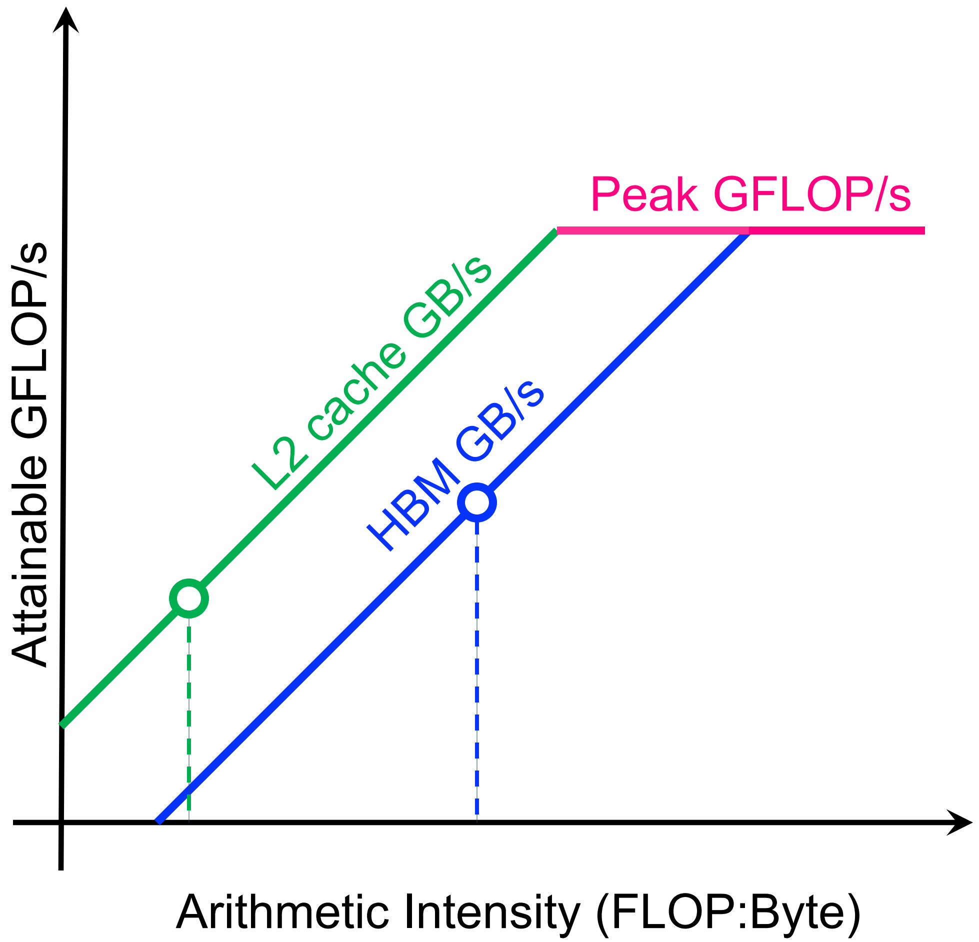
- Plot equation in a single figure…
  - “**Hierarchical Roofline**” Model
  - Bandwidth ceiling (diagonal line) for each level of memory



T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, S. Williams, "A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization", ISC, 2018.

Cache Bottlenecks

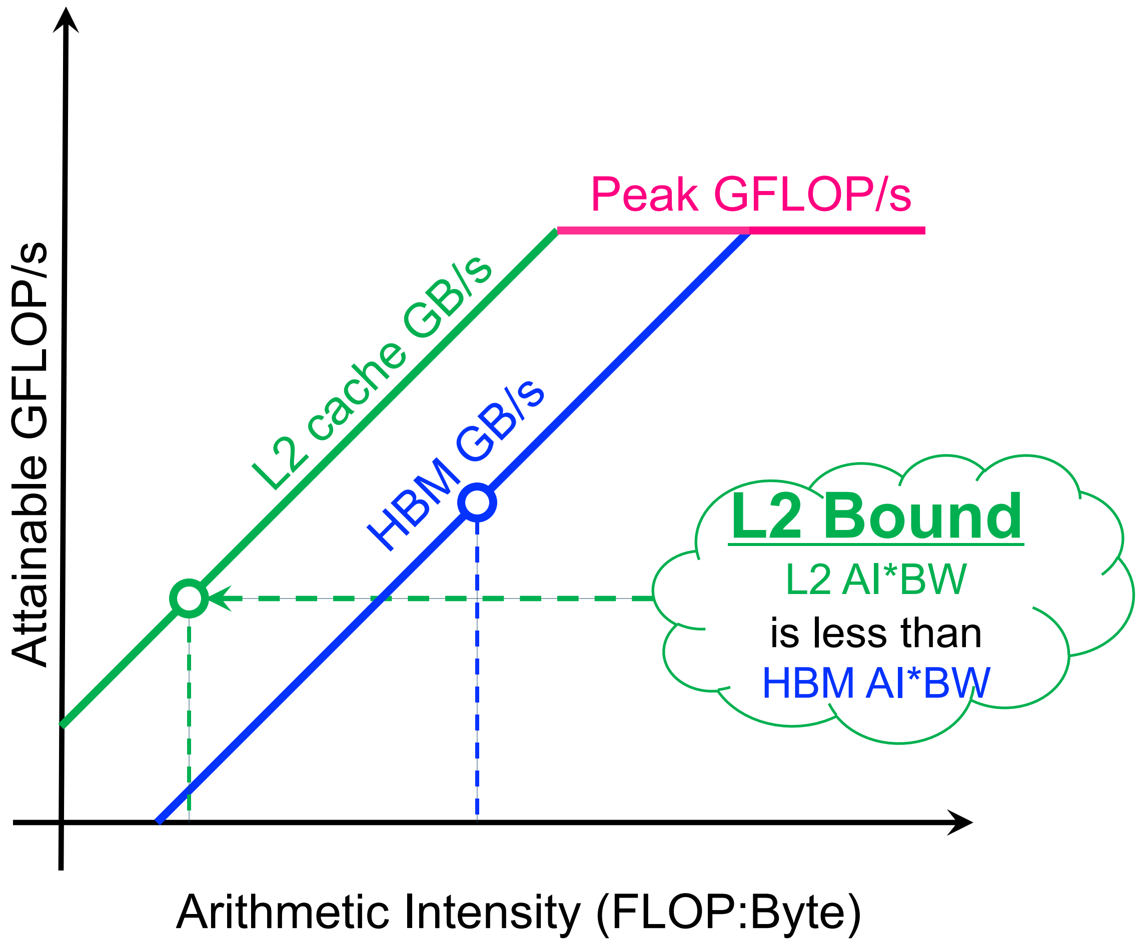
- Plot equation in a single figure…
  - “**Hierarchical Roofline**” Model
  - Bandwidth ceiling (diagonal line) for each level of memory
  - Arithmetic Intensity (dot) for each level of memory



T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, S. Williams, "A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization", ISC, 2018.

Cache Bottlenecks

- Plot equation in a single figure…
  - “**Hierarchical Roofline**” Model
  - Bandwidth ceiling (diagonal line) for each level of memory
  - Arithmetic Intensity (dot) for each level of memory
  - **performance is ultimately the minimum of these bounds**



L2 Bound

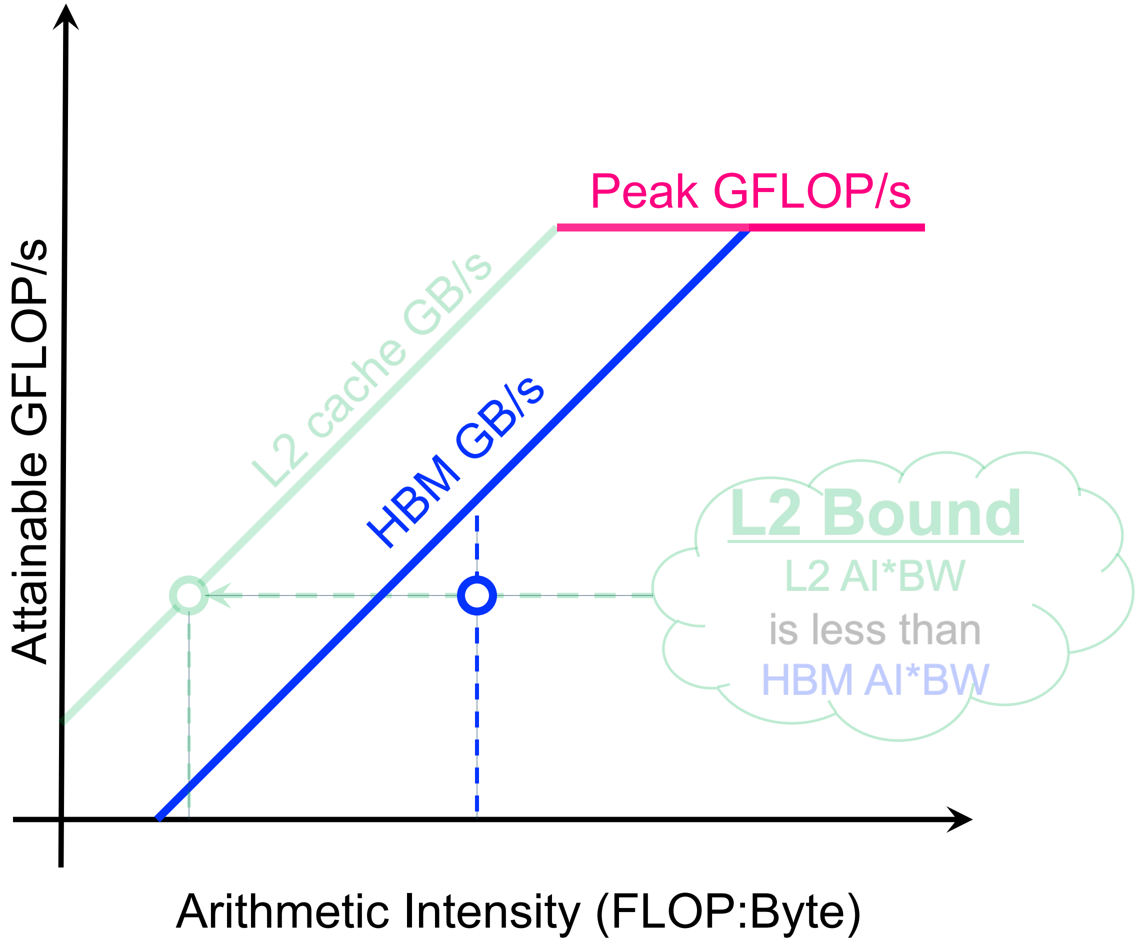
L2 AI*BW
is less than
HBM AI*BW

T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, S. Williams, "A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization", ISC, 2018.

Cache Bottlenecks

- Plot equation in a single figure…
  - "**Hierarchical Roofline**" Model
  - Bandwidth ceiling (diagonal line) for each level of memory
  - Arithmetic Intensity (dot) for each level of memory
  - **performance is ultimately the minimum of these bounds**

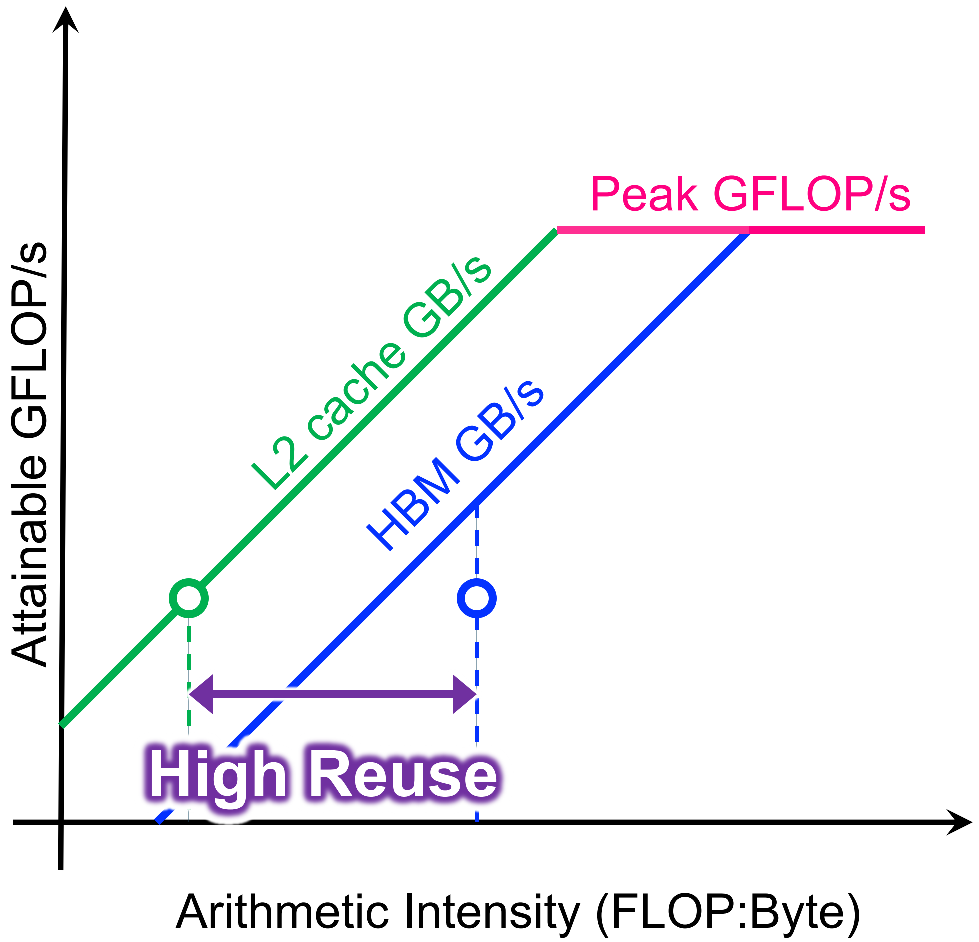
- **If L2 bound, we see DRAM dot well below DRAM ceiling**



T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, S. Williams, "A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization", ISC, 2018.

Cache Hit Rates

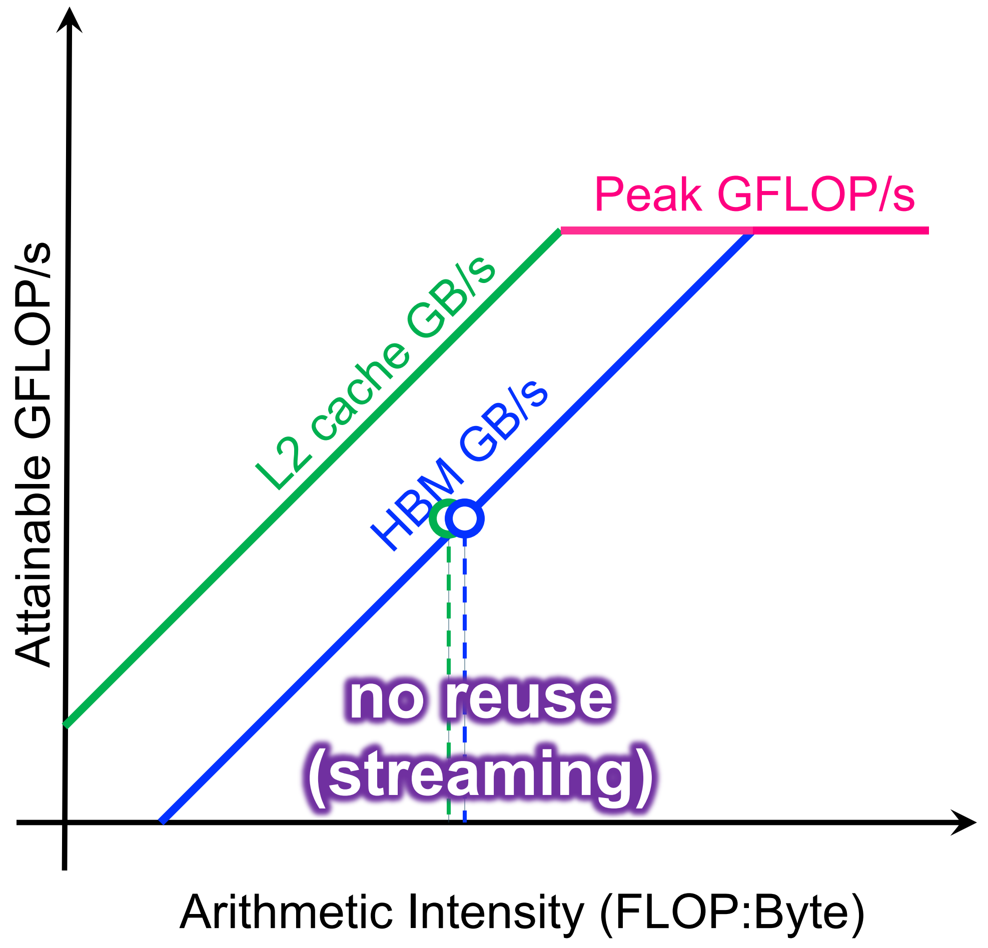
- Widely separated Arithmetic Intensities indicate high reuse in the (L2) cache



T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, S. Williams, "A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization", ISC, 2018.

Cache Hit Rates

- Widely separated Arithmetic Intensities indicate high reuse in the (L2) cache
- Similar Arithmetic Intensities indicate effectively no (L2) cache reuse (**== streaming**)



T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, S. Williams, "A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization", ISC, 2018.

Below the Roofline?

Return of CISC

Return of CISC

- Vectors have their limits (finite DLP, register file energy scales with VL, etc…)
- Death of Moore’s Law is reinvigorating Complex Instruction Set Computing (CISC)

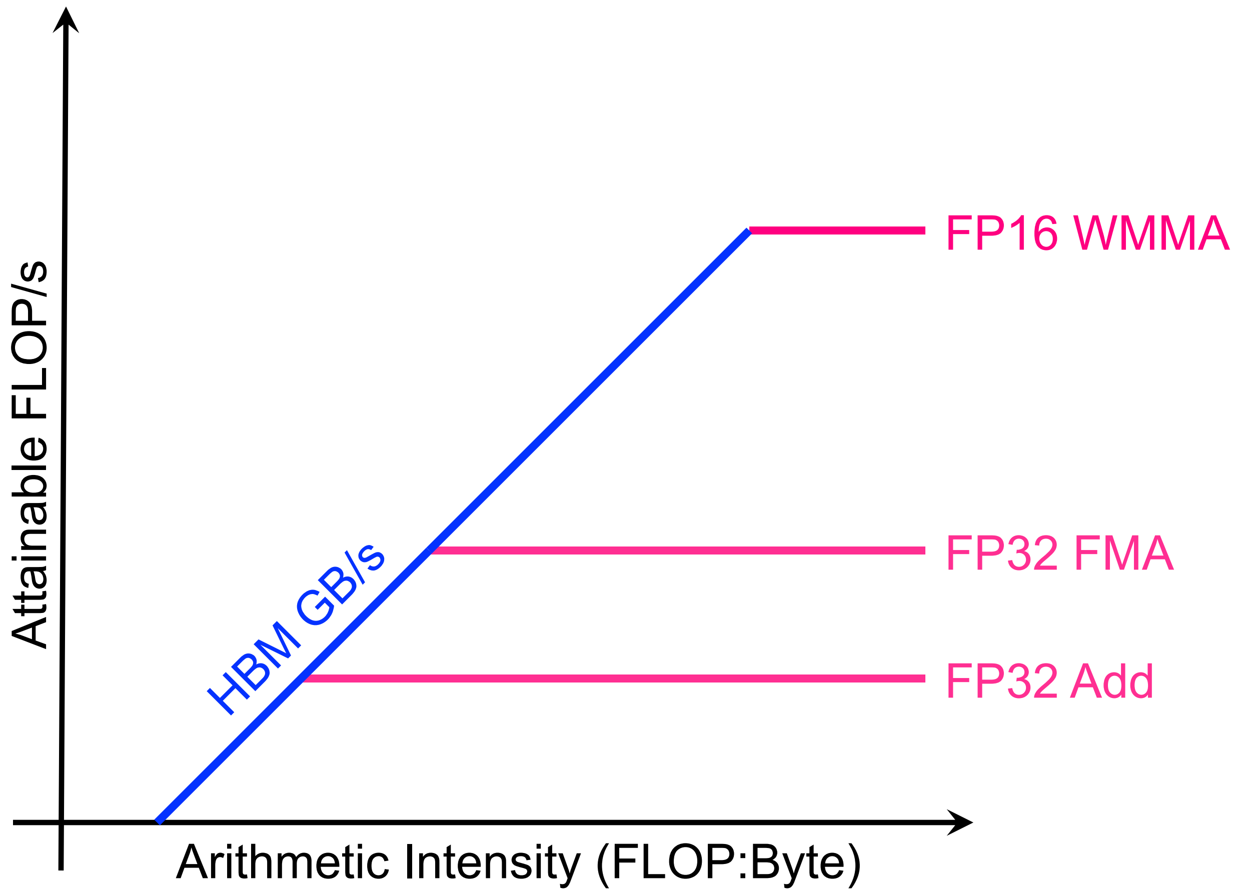
- Modern CPUs and GPUs are increasingly reliant on special (fused) instructions that perform multiple operations (fuse common instruction sequences)…

○ FMA (Fused Multiply Add):	z=a*x+y	*…z,x,y are vectors or scalars*
○ 4FMA (Quad FMA):	z=A*x+z	*…A is a FP32 matrix; x,z are vectors*
○ WMMA (Tensor Core):	Z=AB+C	*…A,B are FP16 matrices; Z,C are FP32*

➢ **Define a set of “ceilings” based on instruction type (all tensor, all FMA, or all FADD)**

Floating-Point and Mixed Precision Ceilings

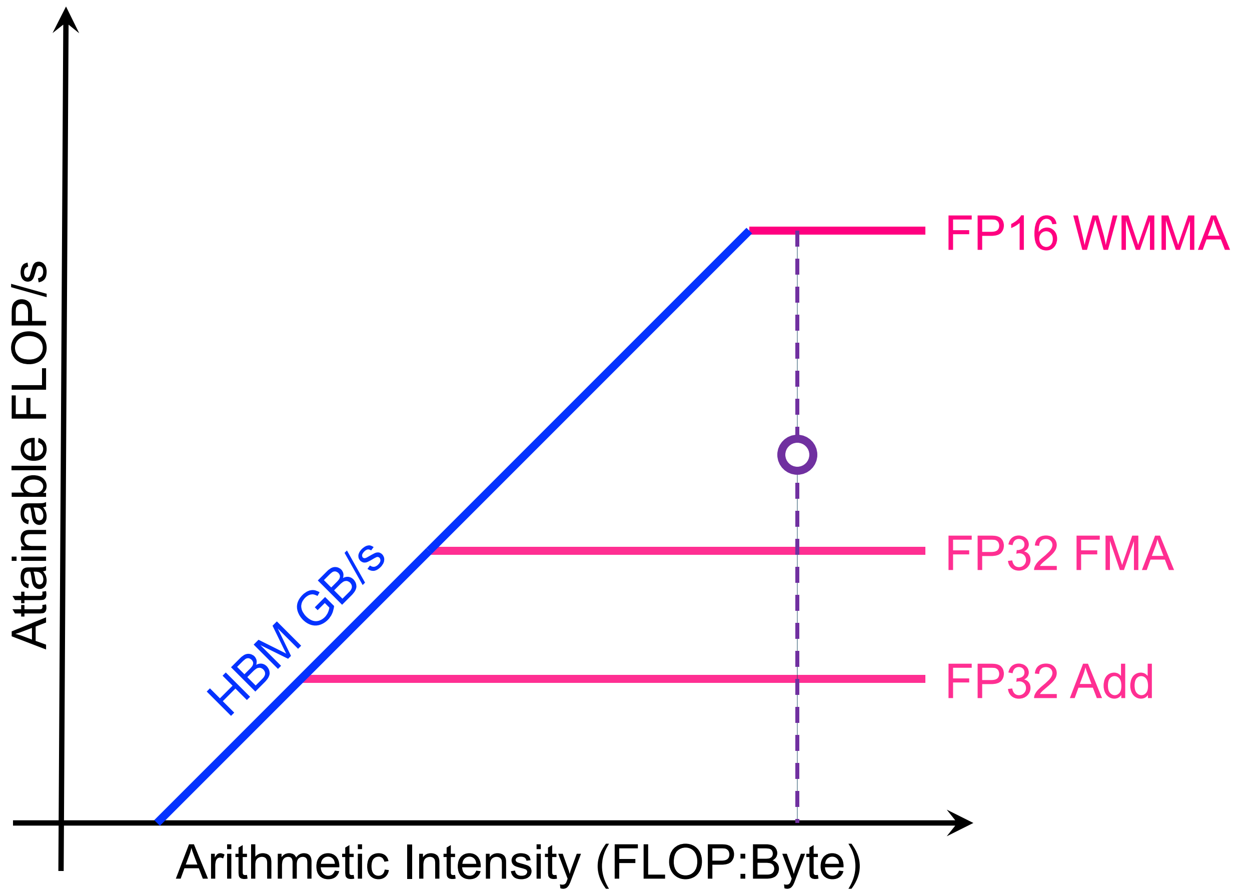
- Consider NVIDIA Volta GPU
- We may define 3 performance ceilings…
  - 15 TFLOPS for FP32 FMA
  - 7.5 TFLOPs for FP32 Add
  - ~100 TFLOPs for FP16 Tensor



Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System", Cray User Group (CUG), May 2019.

Floating-Point and Mixed Precision Ceilings

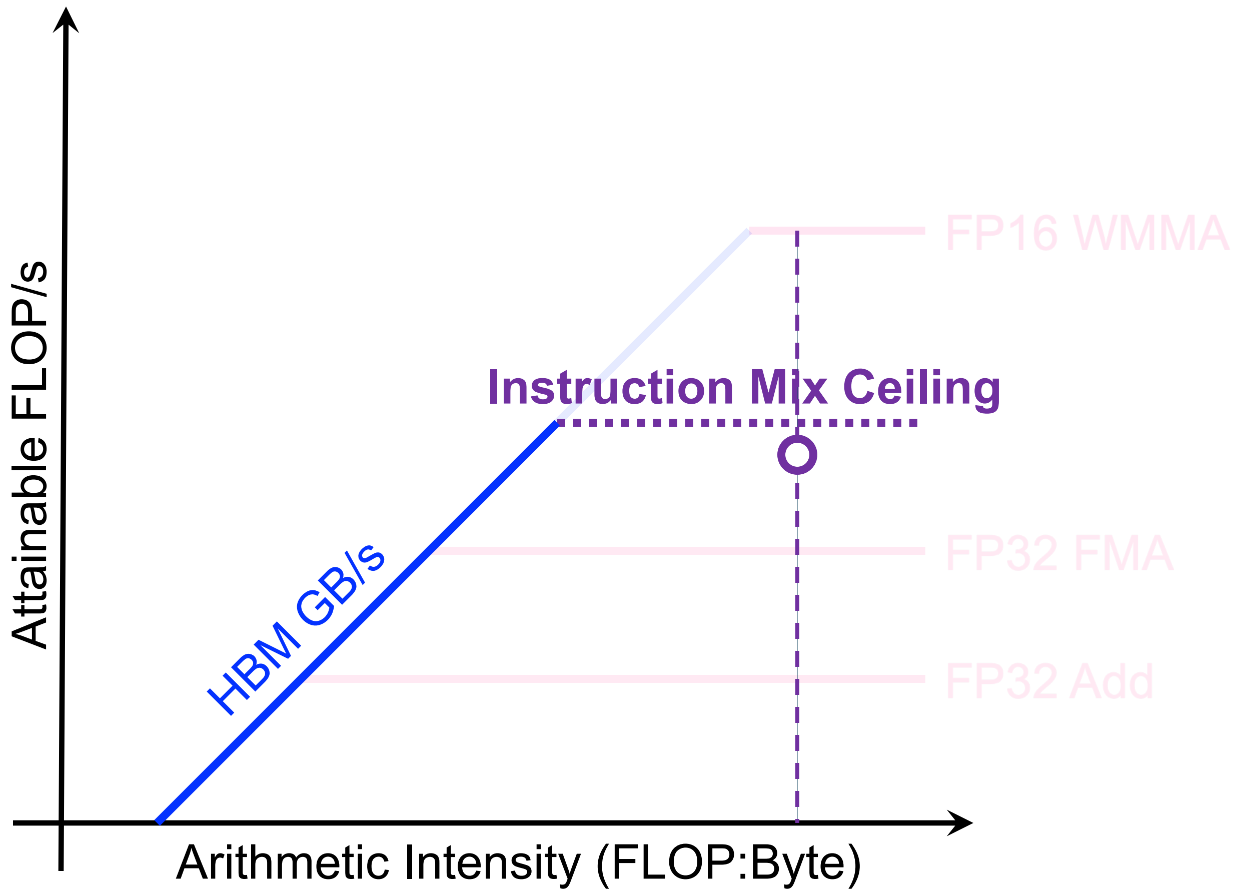
- When calculating (AI,GFLOP/s), count the total FLOPs from all types of instructions
- DL performance can often be well below nominal Tensor Core peak



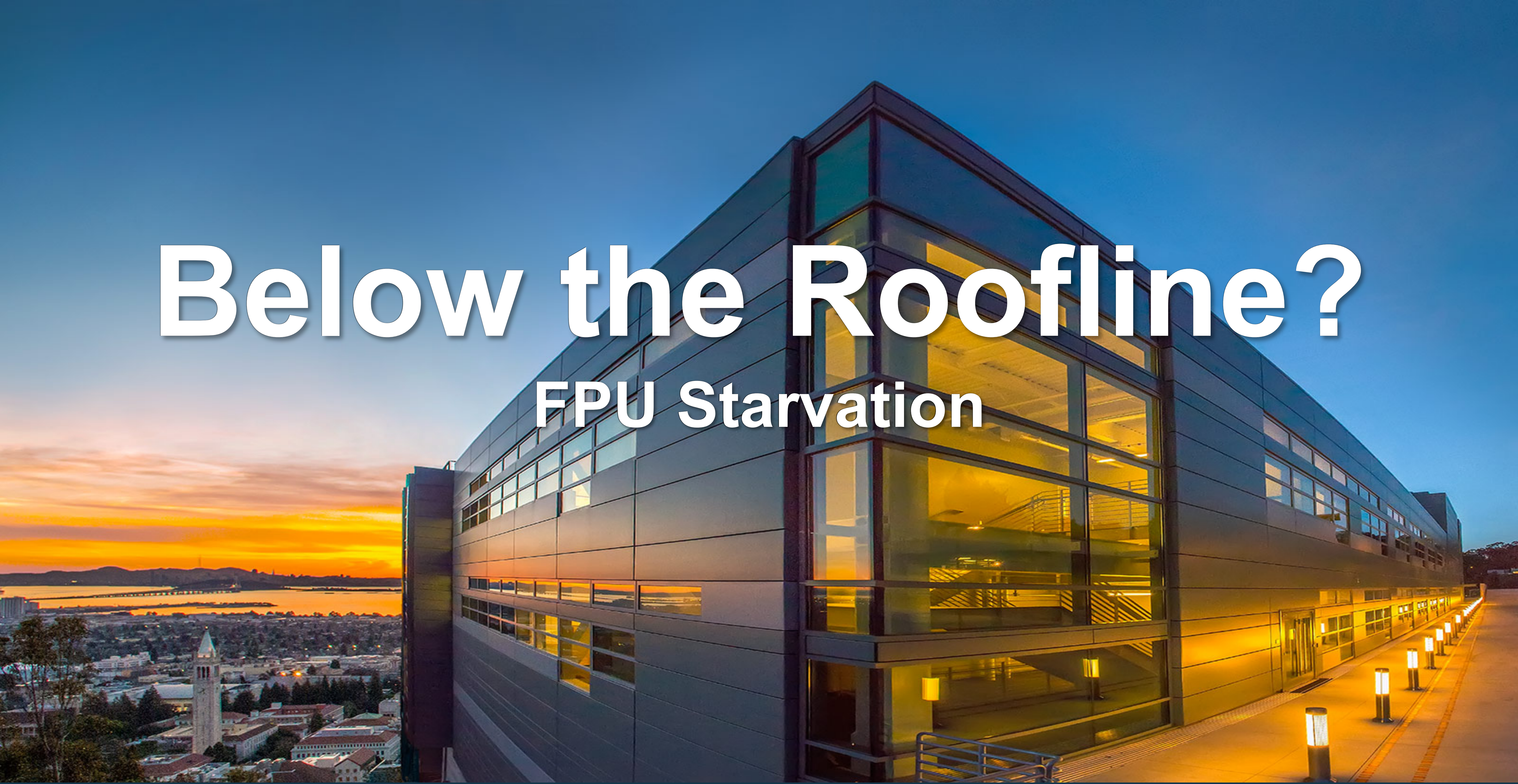
Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System", Cray User Group (CUG), May 2019.

Floating-Point and Mixed Precision Ceilings

- When calculating (AI,GFLOP/s), count the total FLOPs from all types of instructions
- DL performance can often be well below nominal Tensor Core peak
- DL applications are a mix Tensor, FP16, and FP32 instructions
- Thus, there is an **ceiling** on performance defined by the mix of instructions



Charlene Yang, Thorsten Kurth, Samuel Williams, "Hierarchical Roofline Analysis for GPUs: Accelerating Performance Optimization for the NERSC-9 Perlmutter System", Cray User Group (CUG), May 2019.



Below the Roofline?

FPU Starvation

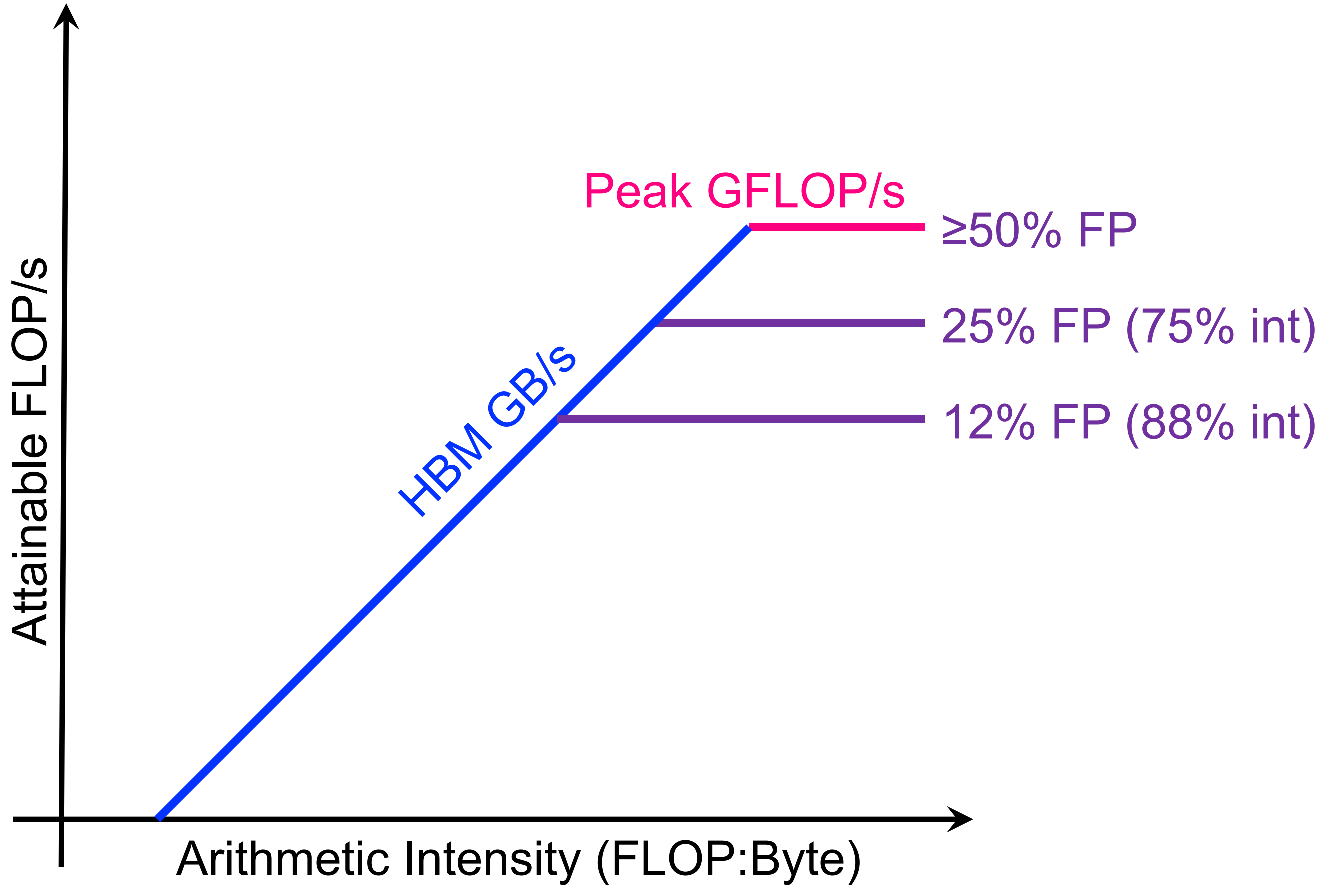
FPU Starvation

- CPUs and GPUs have finite instruction fetch/decode/issue bandwidth
- The number of FPUs dictates the FP issue rate required to hit peak

➢ **Ratio of these two rates is the minimum FP instruction fraction required to hit peak**

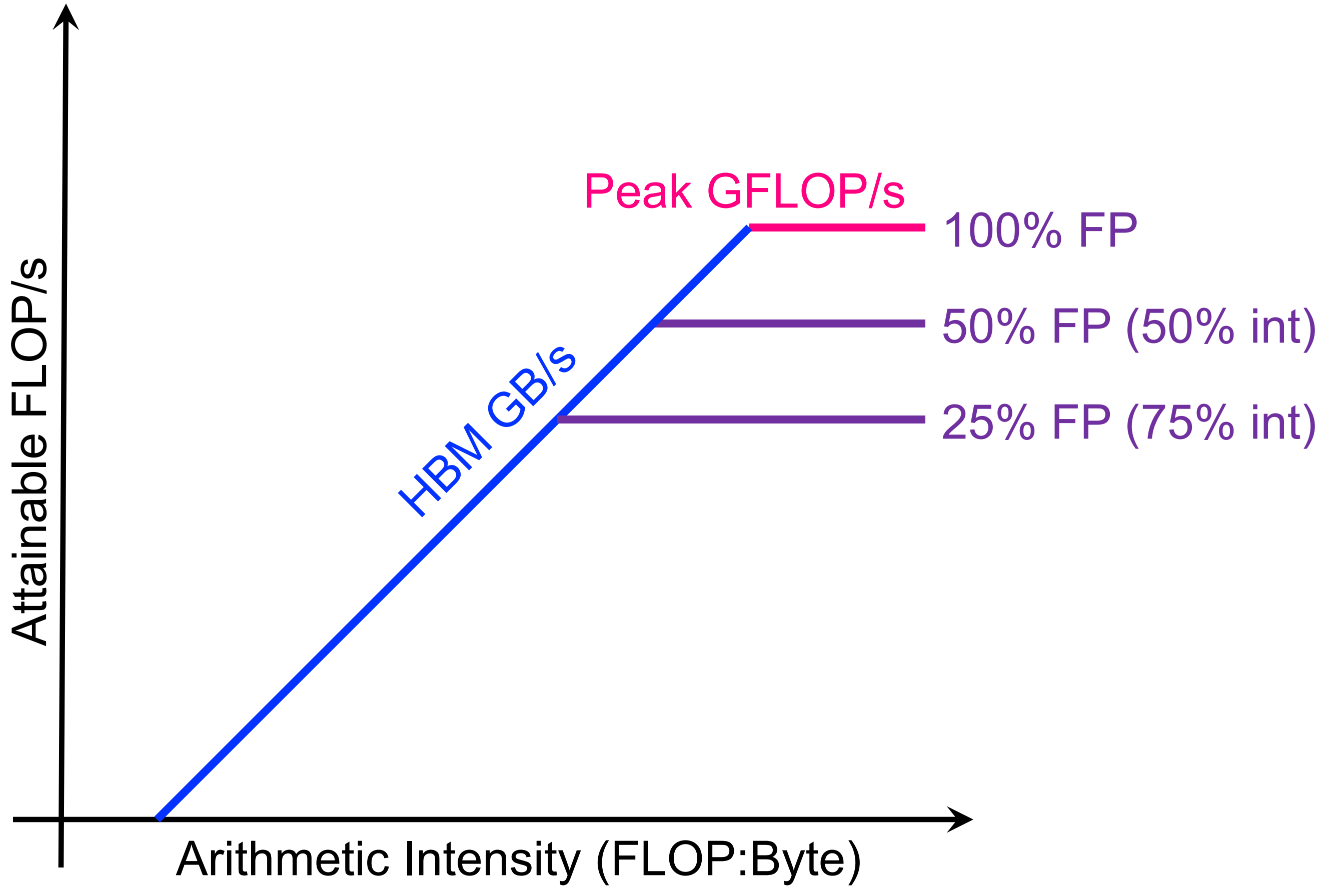
FPU Starvation

- Consider…
  - 4-issue CPU (or GPU)
  - 2 FP data paths
  - **>50% of the instructions** must be FP to have any chance at peak performance



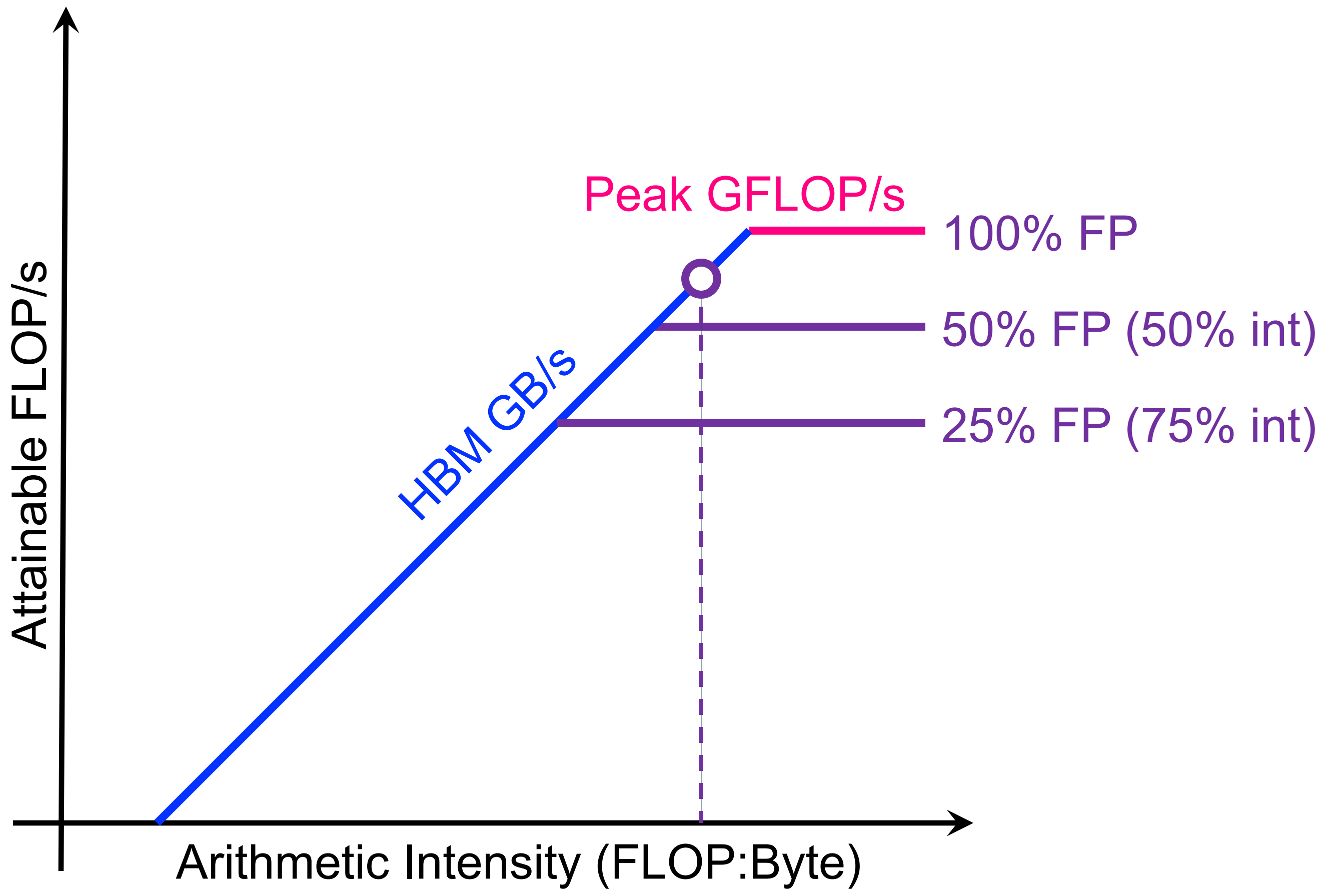
FPU Starvation

- Conversely,
  - Keeping 2 FP data paths,
  - but downscaling to 2-issue CPU (or GPU)
  - ➢ **100% of the instructions must be FP to get peak performance**



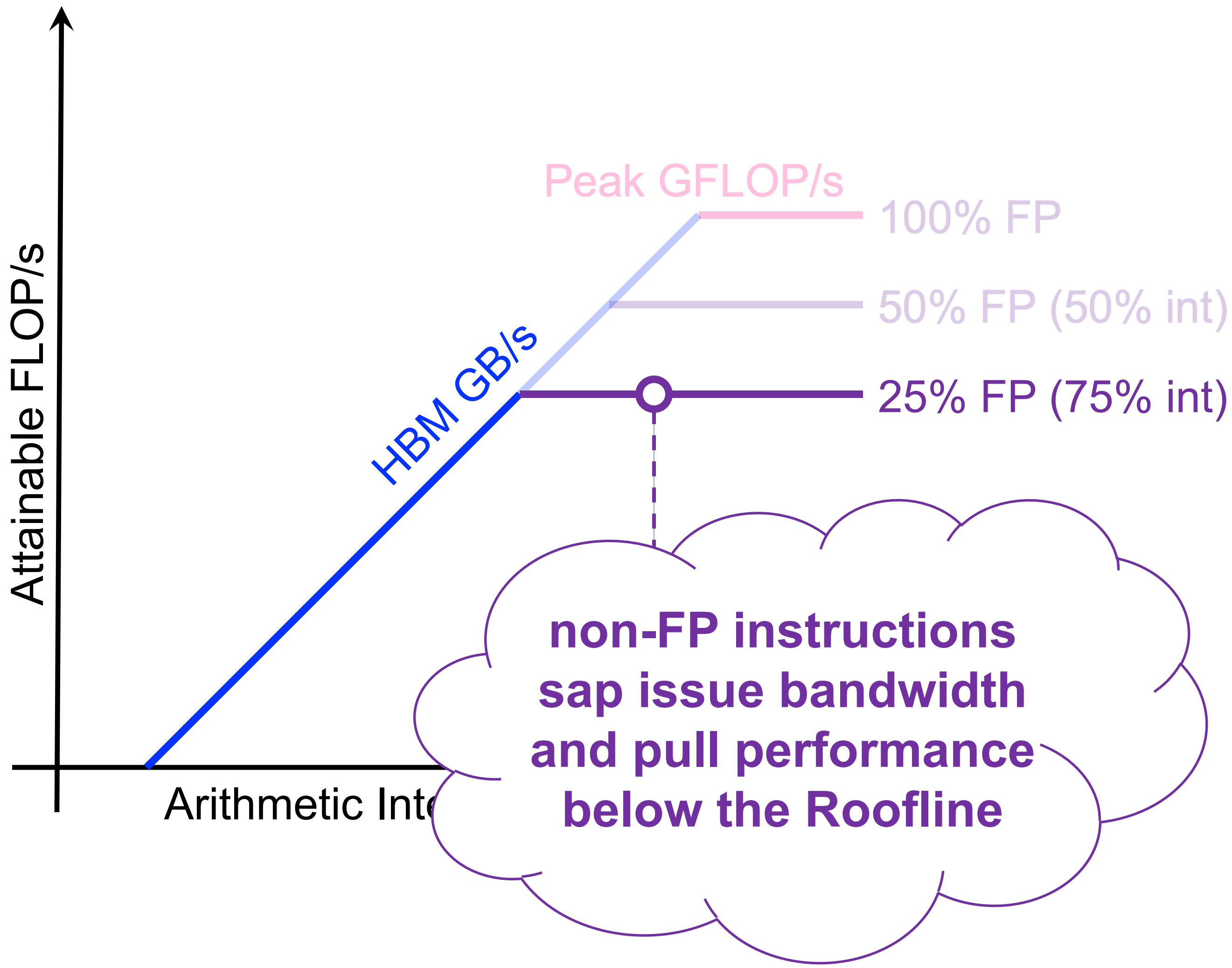
FPU Starvation

- Conversely,
  - Keeping 2 FP data paths,
  - but downscaling to 2-issue CPU (or GPU)
  - ➢ **100% of the instructions must be FP to get peak performance**



FPU Starvation

- Conversely,
  - Keeping 2 FP data paths,
  - but downscaling to 2-issue CPU (or GPU)
  - 100% of the instructions must be FP to get peak performance
  - **Codes that would have been memory-bound are now decode/issue-bound.**



non-FP instructions sap issue bandwidth and pull performance below the Roofline

Recap

Recap

- Roofline bounds performance as a function of Arithmetic Intensity
  - Horizontal Lines = Compute Ceilings
  - Diagonal Lines = Bandwidth Ceilings
  - Bandwidth ceilings are always parallel on log-log scale
  - **Collectively, define an upper limit on performance (speed-of-light)**
- Loop Arithmetic Intensity (for each level of memory)
  - **Total FLOPs / Total Data Movement** (for that level of memory)
  - Measure of a loop's temporal locality
  - Includes **all** cache effects
- Plotting loops on the (Hierarchical) Roofline
  - **Each loop has one dot per level of memory**
  - x-coordinate = arithmetic intensity at that level
  - y-coordinate = performance (e.g. GFLOP/s)
  - Proximity to associated ceiling is indicative of a performance bound
  - Proximity of dots to each other is indicative of **streaming** behavior (low cache hit rates)

Why would you use Roofline?

- Understand performance differences between Architectures, Programming Models, implementations, etc…
  - Why do some Architectures/Implementations move more data than others?
  - Why do some compilers outperform others?

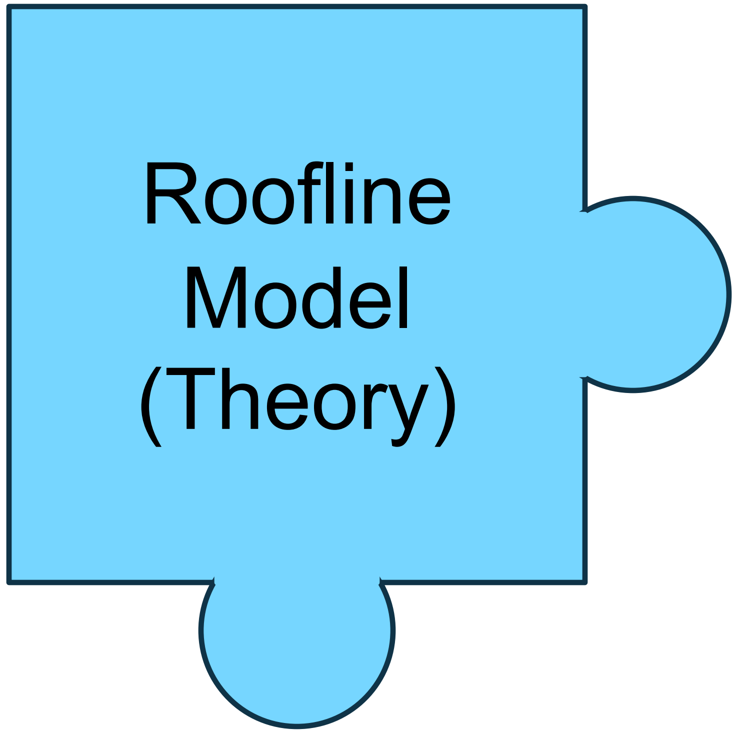
- Predict performance on future machines / architectures
  - Set realistic performance expectations
  - Drive for HW/SW Co-Design

- Identify performance bottlenecks & motivate software optimizations

- Determine when we're done optimizing code
  - Assess performance relative to machine capabilities
  - Track progress towards optimality
  - Motivate need for algorithmic changes

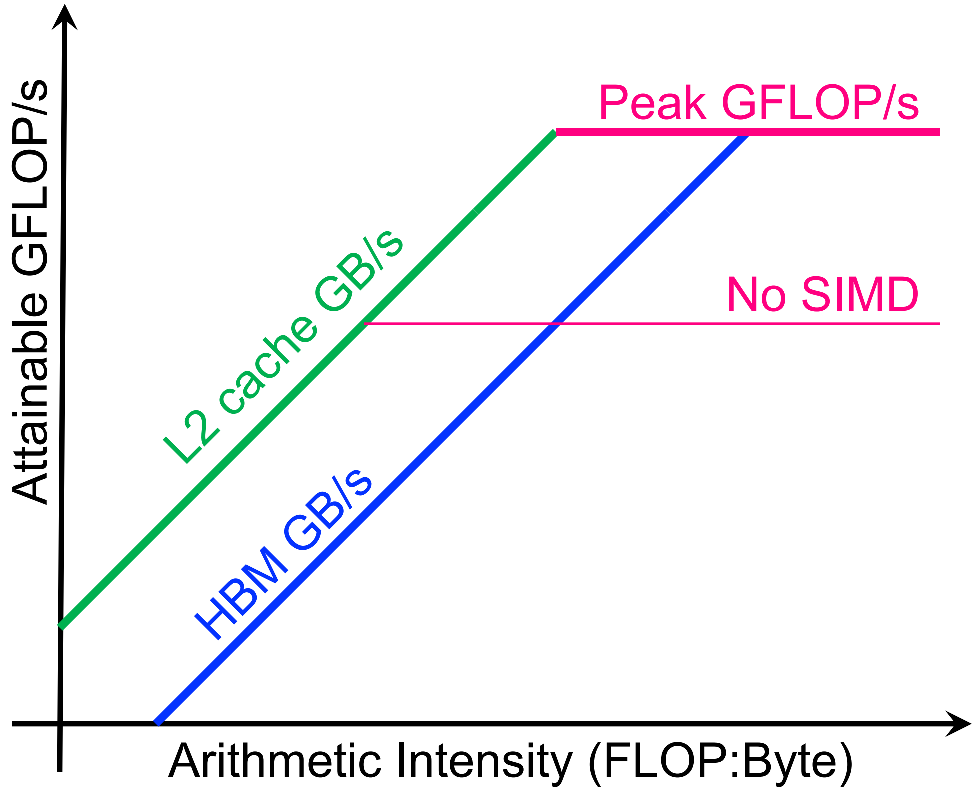
Model is just one piece of the puzzle…

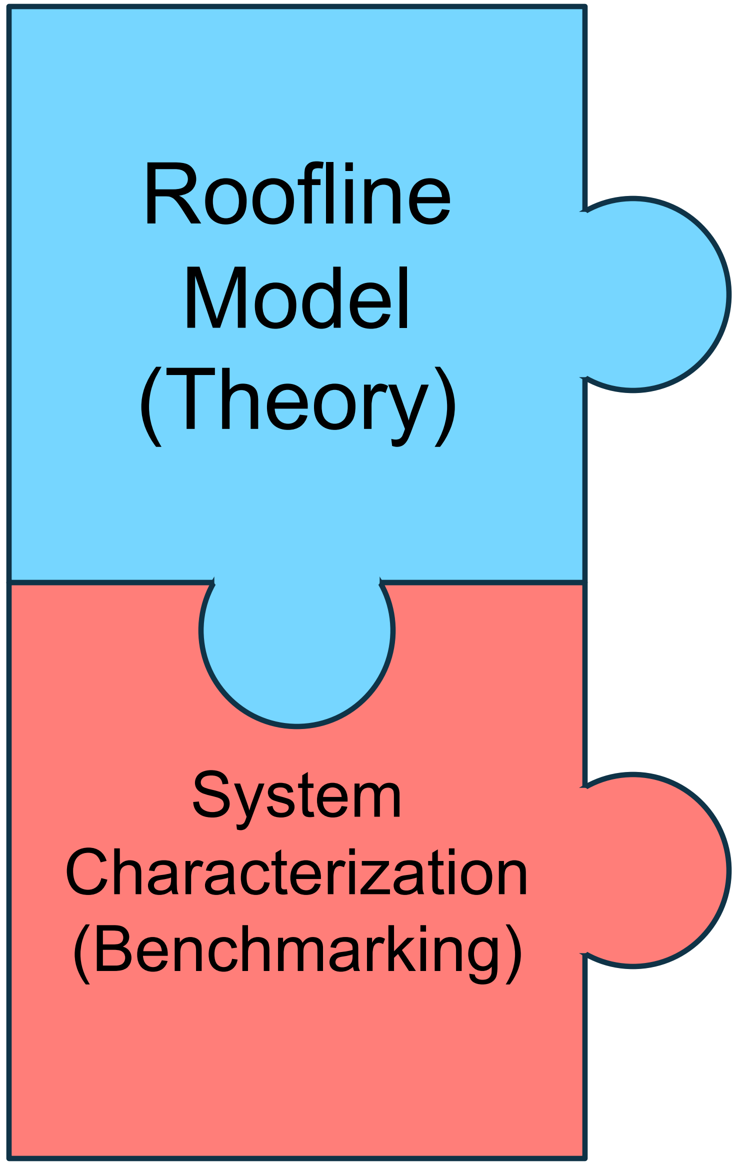
- Roofline Model defines the basic concepts and equations.



Model is just one piece of the puzzle…

- System Characterization defines the shape of the Roofline (peak bandwidths and FLOP/s)

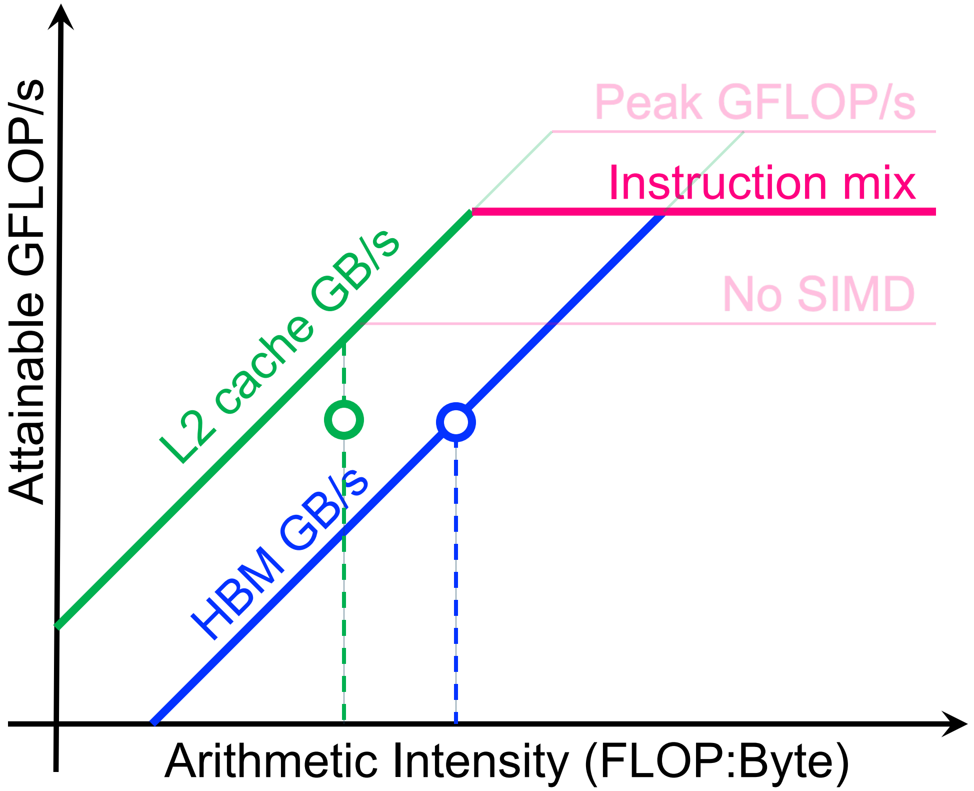


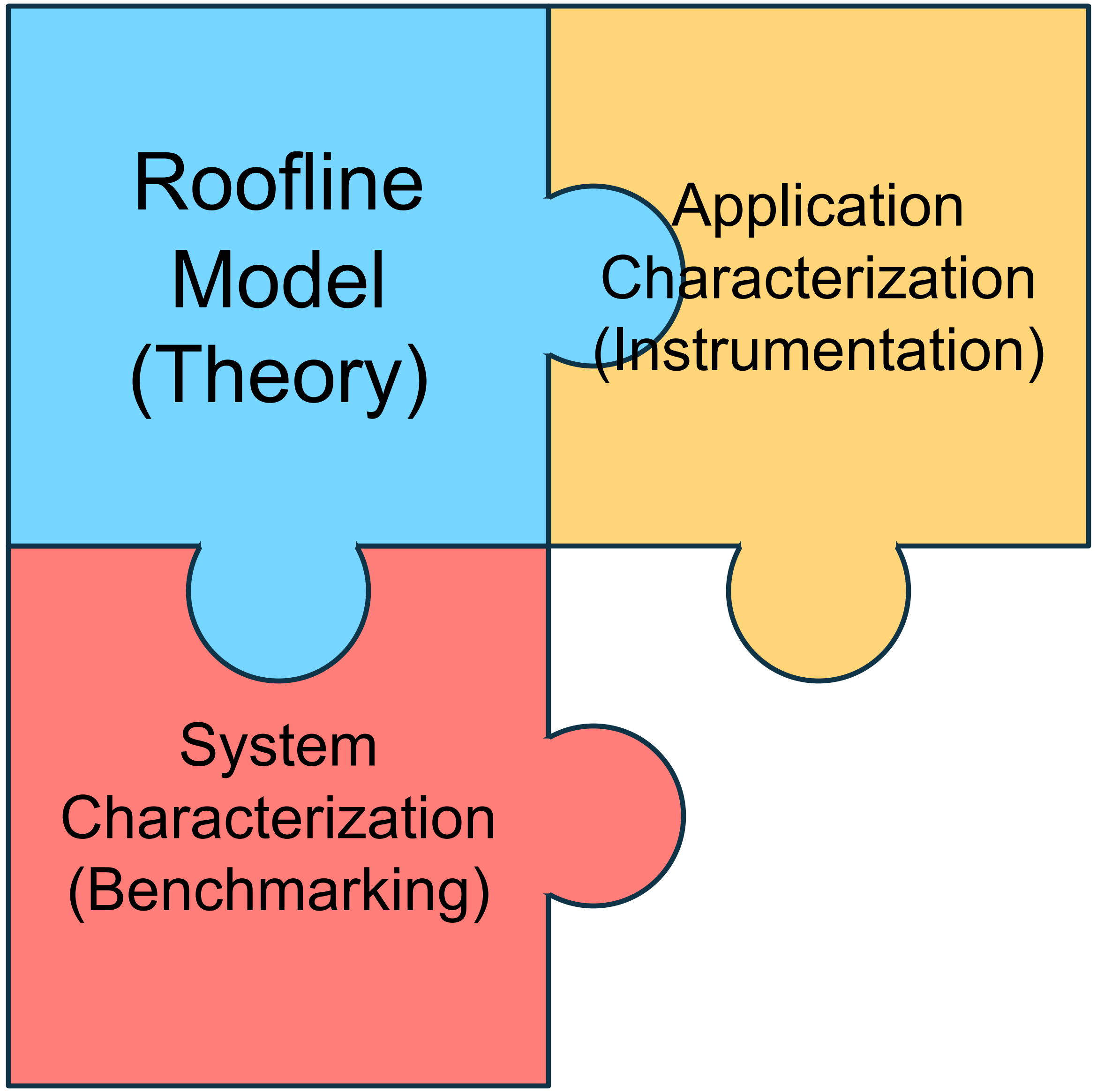

Roofline Model (Theory)

System Characterization (Benchmarking)

Model is just one piece of the puzzle…

- Application Characterization determines…
  - Intensity and Performance of each loop
  - Position of any implicit ceilings

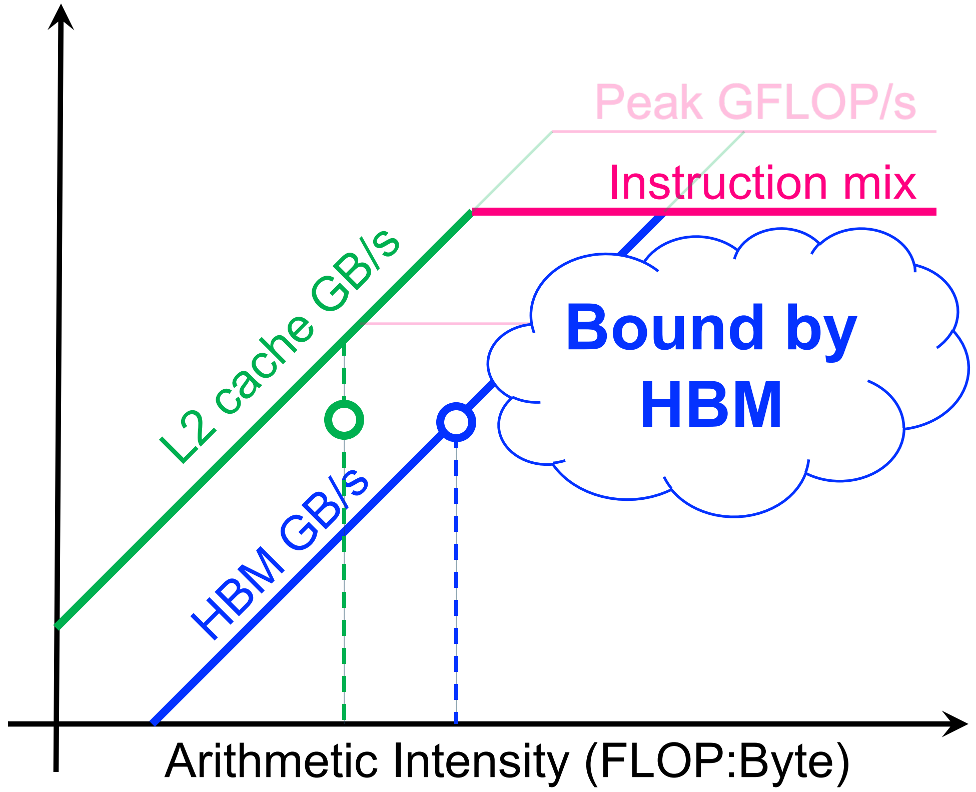


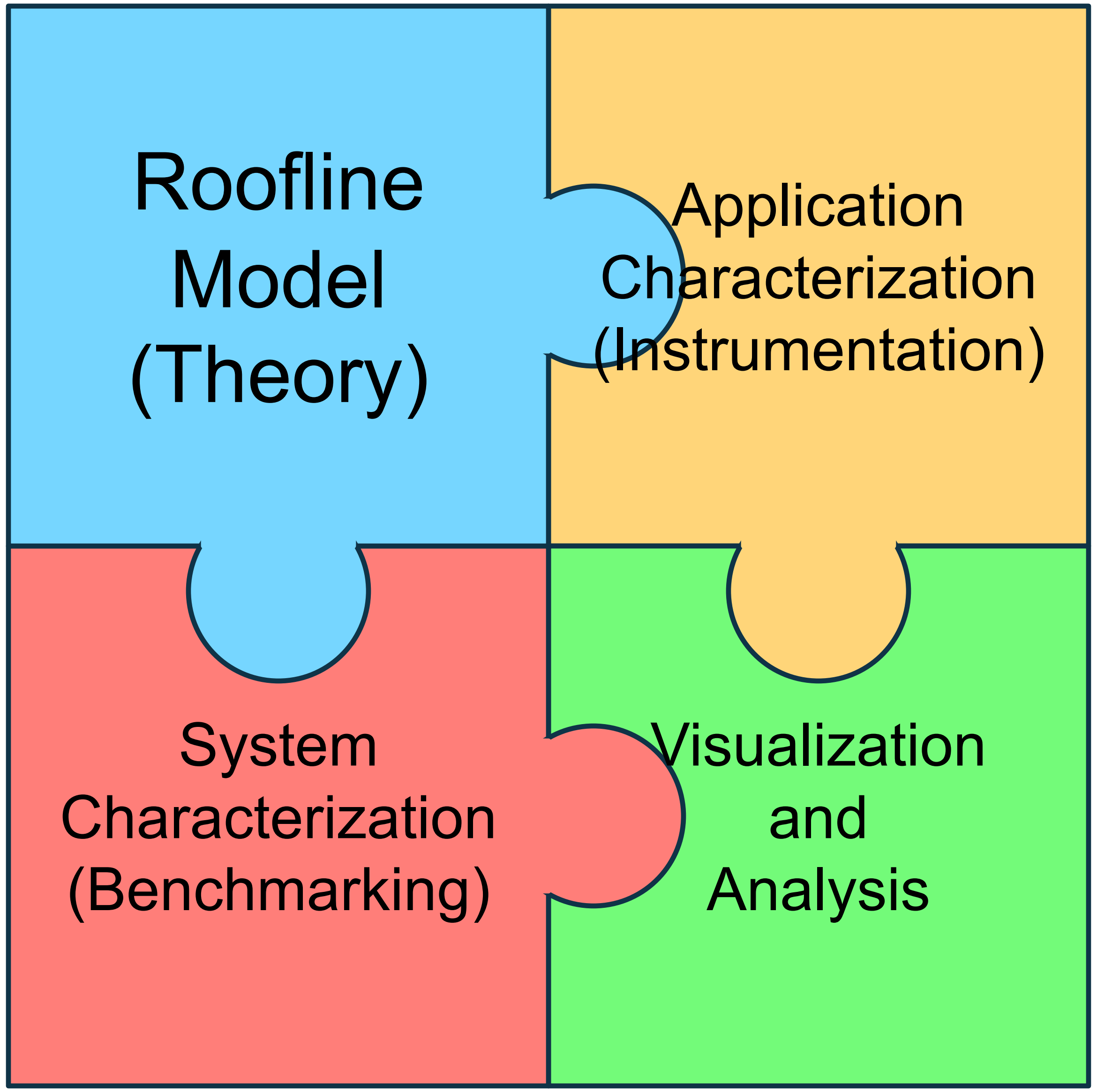


Roofline Model (Theory)
Application Characterization (Instrumentation)
System Characterization (Benchmarking)

Model is just one piece of the puzzle…

- Visualization tools combine all data together and provide analytical capability





Roofline Model (Theory)
Application Characterization (Instrumentation)
System Characterization (Benchmarking)
Visualization and Analysis

Rest of Tutorial

- Tools for Roofline analysis on NVIDIA GPUs and use cases at NERSC
- Tools for Roofline analysis on Intel GPUs/CPUs and use cases at ALCF
- Advanced Roofline topics
- Using TiMemory for portable application profiling

Questions?