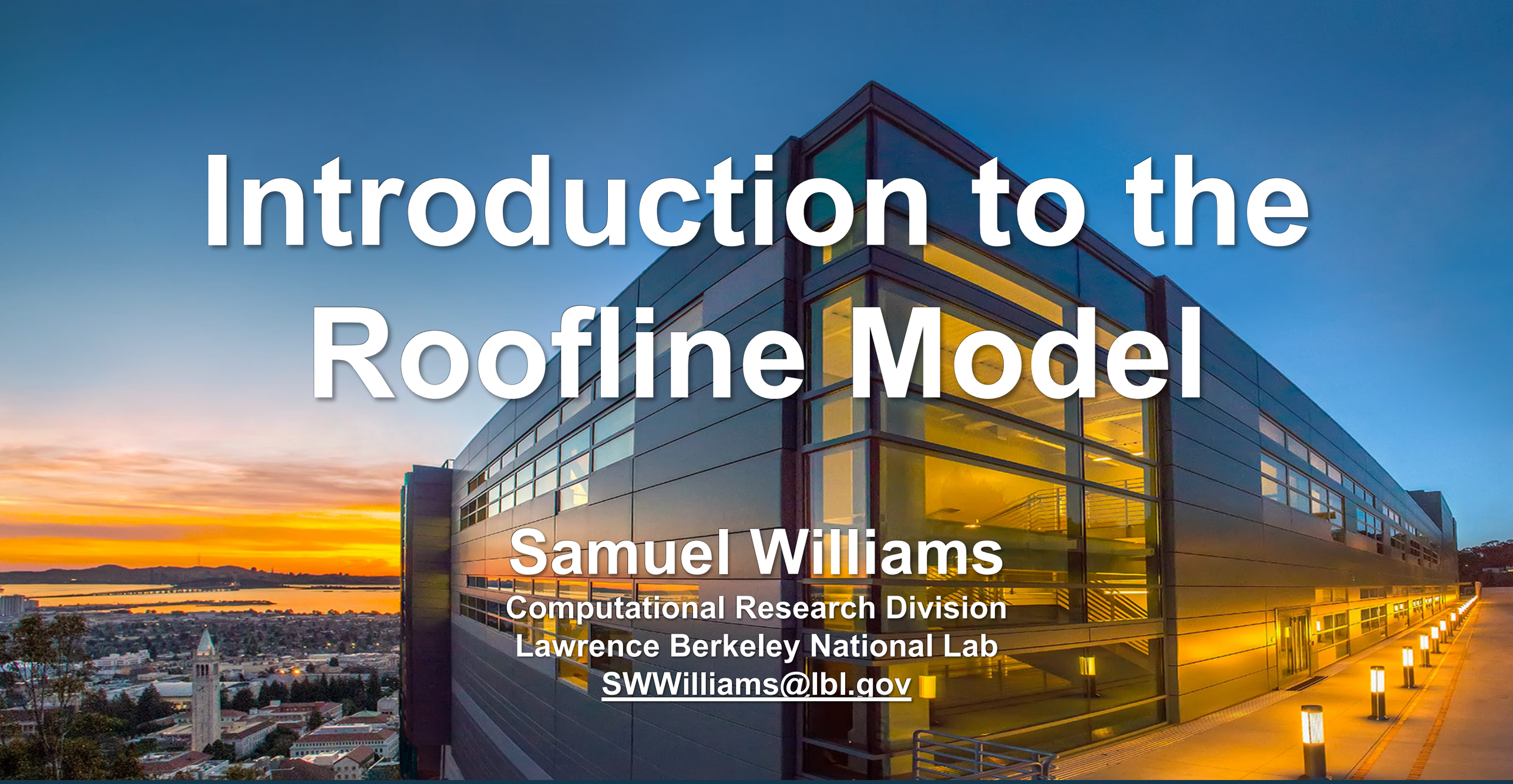


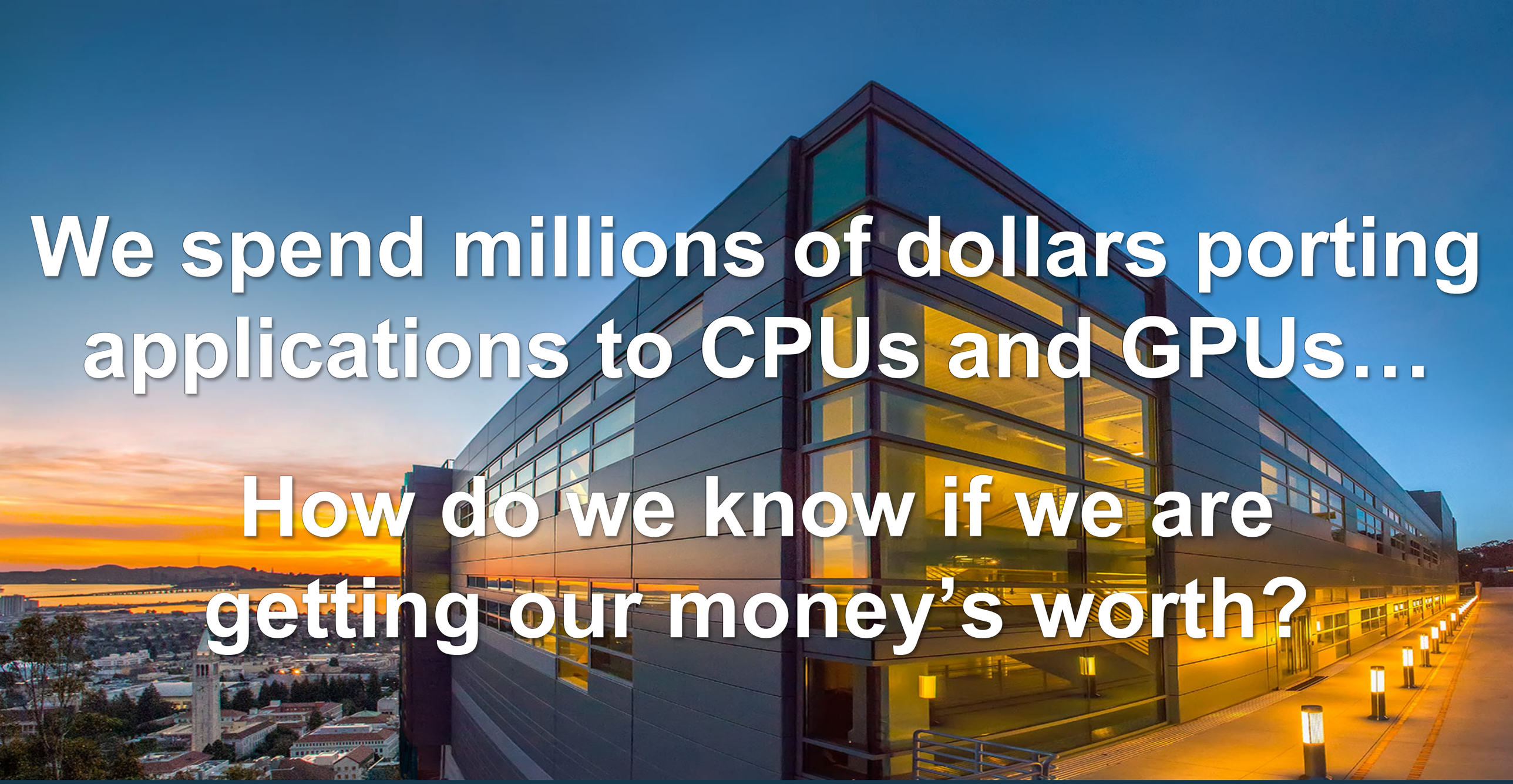
Introduction to the Roofline Model

Samuel Williams

Computational Research Division

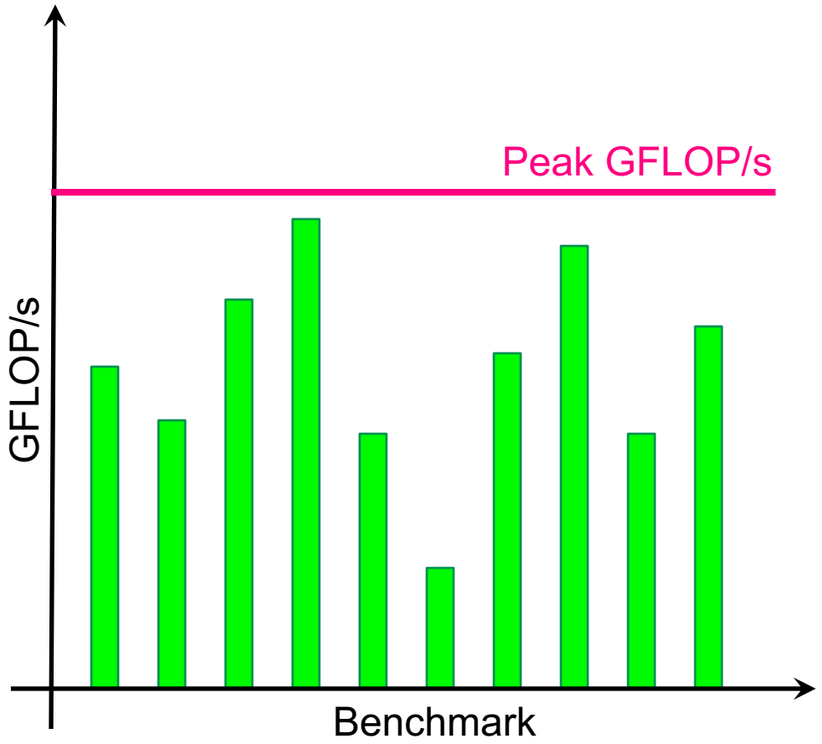
Lawrence Berkeley National Lab

SWWilliams@lbl.gov


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?

- Really a question of good performance on application benchmarks
- Imagine profiling a mix of GPU-accelerated benchmarks …
- GFLOP/s alone may not be particularly insightful

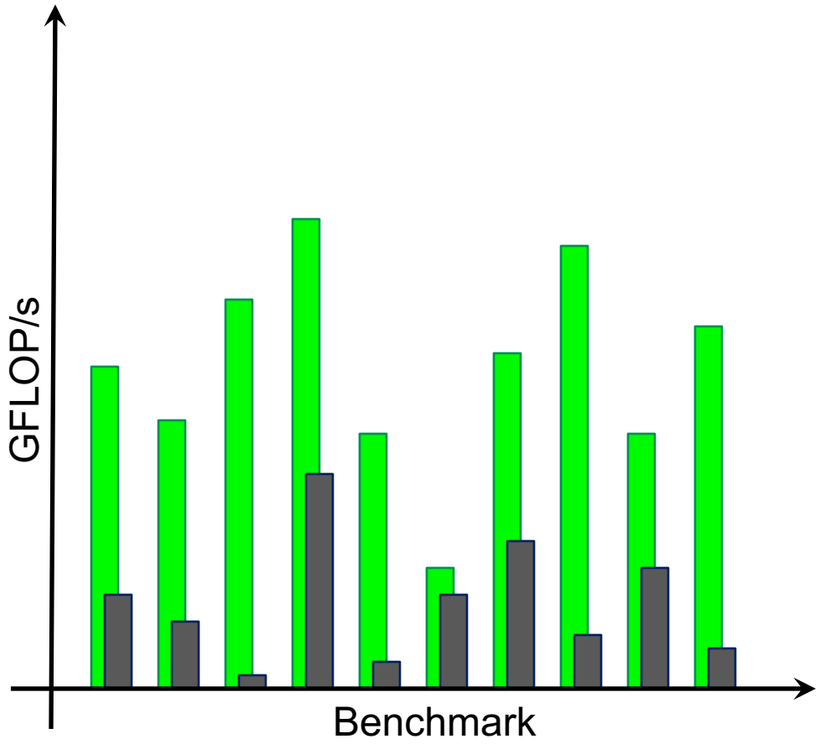


Are we getting good performance?

- 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.**

➢ **'Speedup' provides no guidance to CS, AM, applications, procurement, or vendors.**



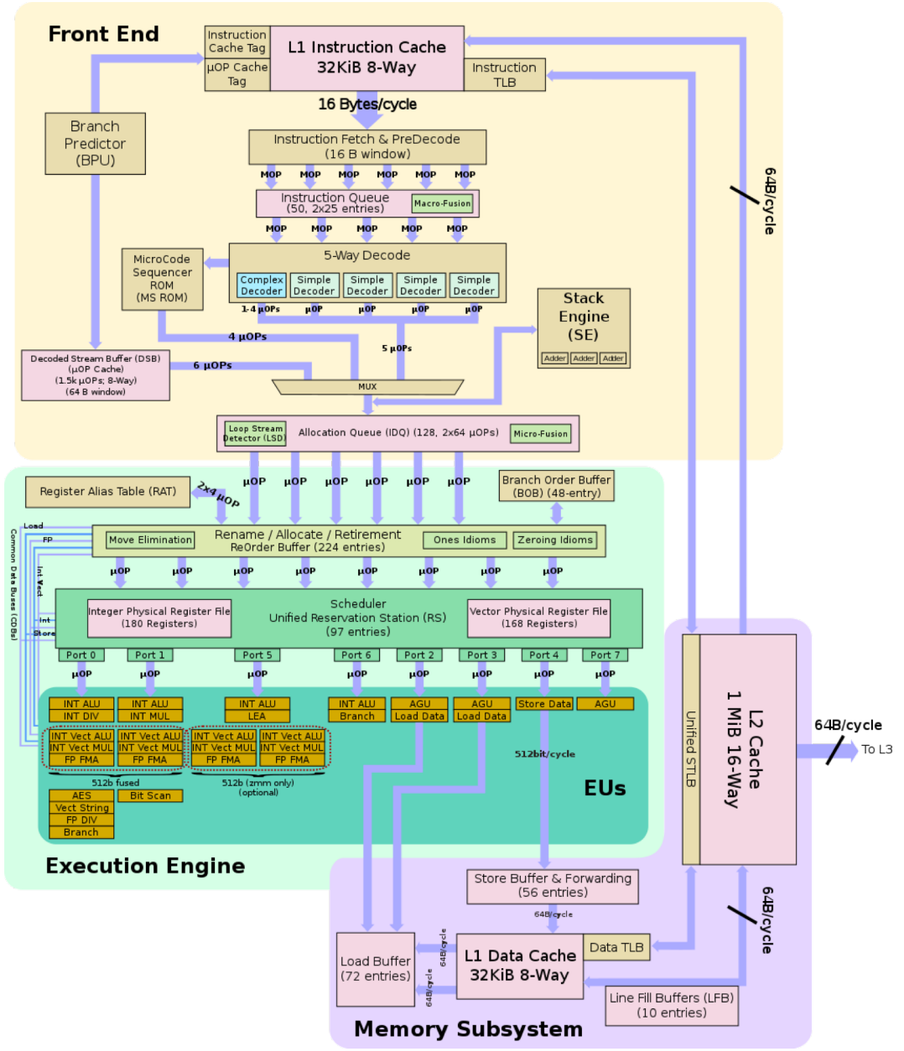
Are we getting good performance?

- 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?

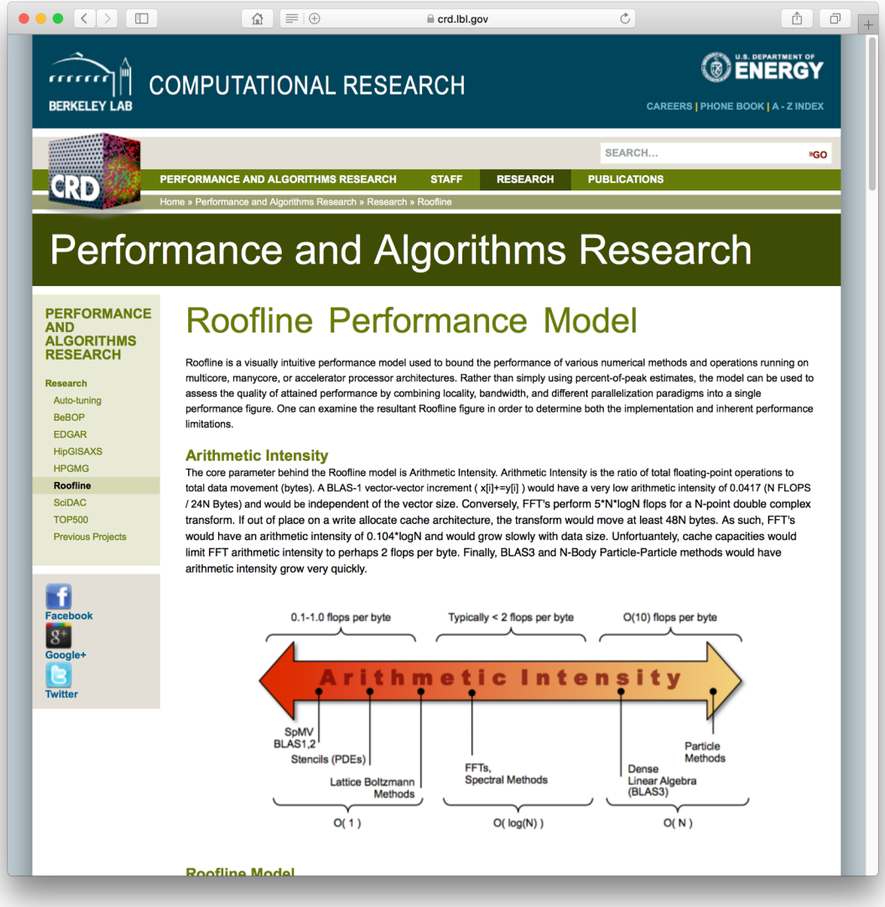
- We could take the computer architect's approach and build a simulator to understand performance nuances…
  - Modern architectures are incredibly complex
  - Simulators may perfectly reproduce performance
  - Deluge of information interpretable only by computer architects
  - worse, might incur 10^6x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



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

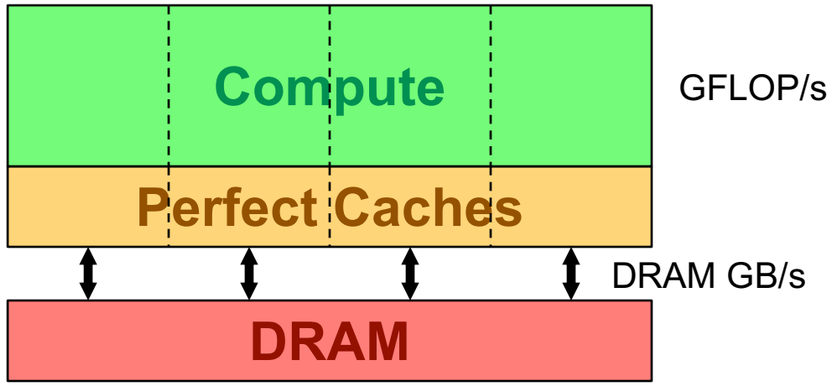
➢ **Roofline is just such a model**



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

Data Movement or Compute?

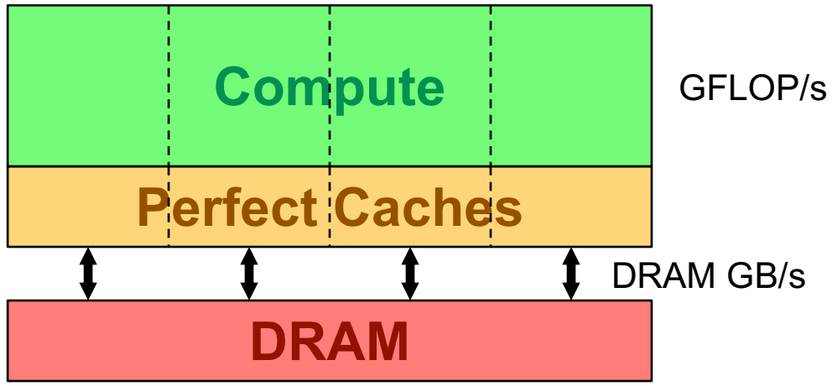
- Which takes longer?
  - Data Movement
  - Compute?



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

Data Movement or Compute?

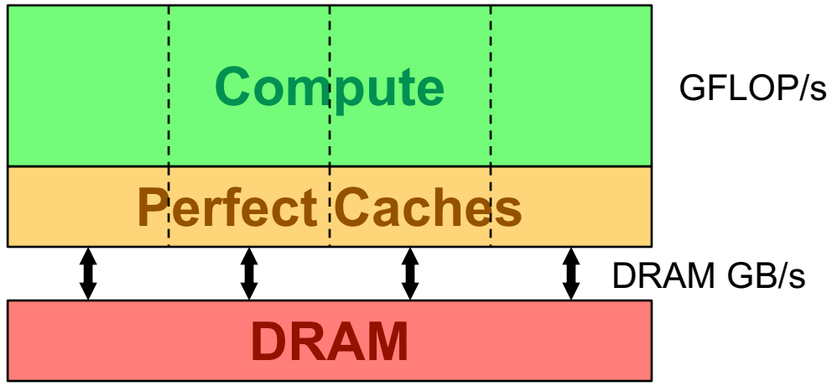
- Which takes longer?
  - Data Movement
  - Compute?
- Is performance limited by compute or data movement?



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

Data Movement or Compute?

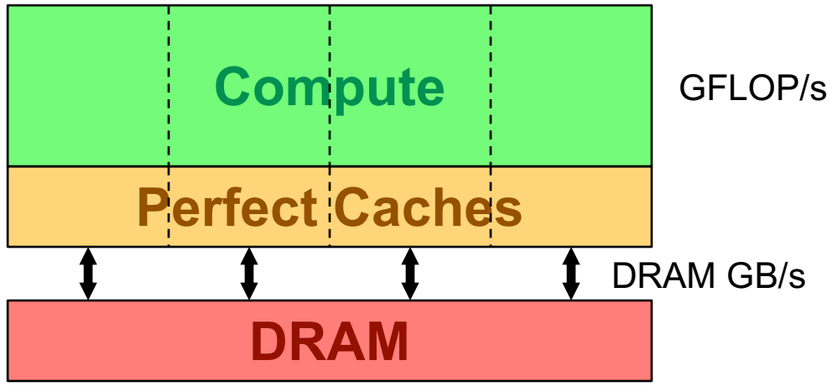
- Which takes longer?
  - Data Movement
  - Compute?
- Is performance limited by compute or data movement?



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

Data Movement or Compute?

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



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

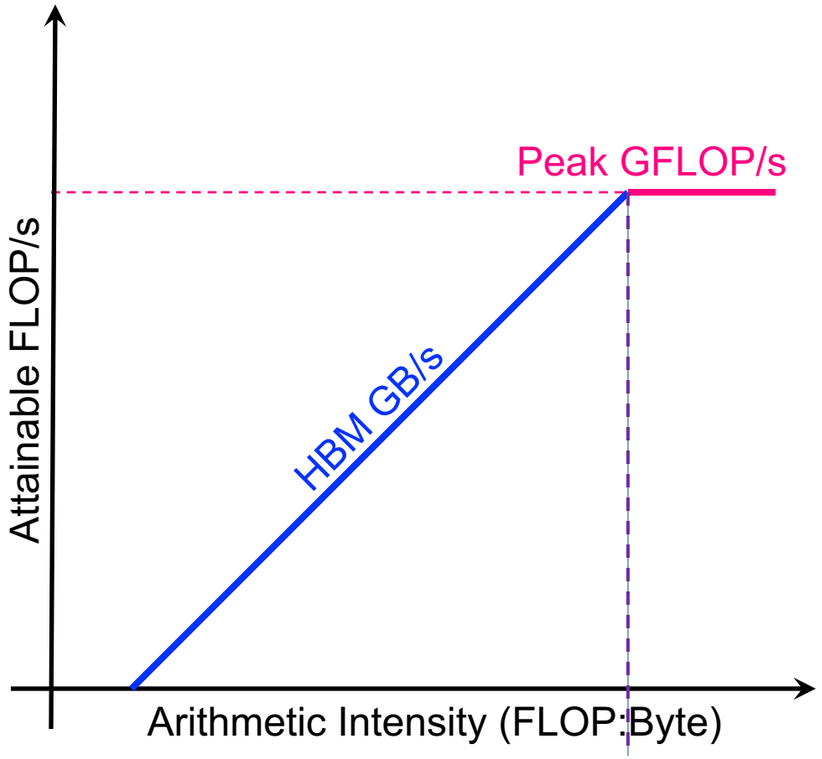
Arithmetic Intensity (AI) = measure of data locality

(DRAM) Roofline Model

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

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

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



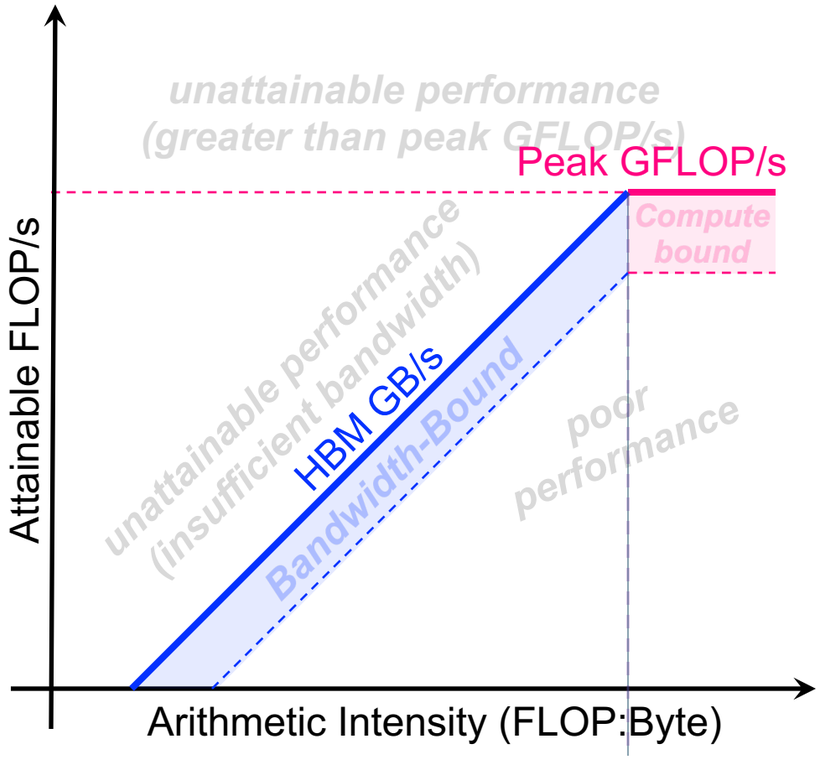
Transition @ AI == Peak GFLOP/s / Peak GB/s == 'Machine Balance'

(DRAM) Roofline Model

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

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…



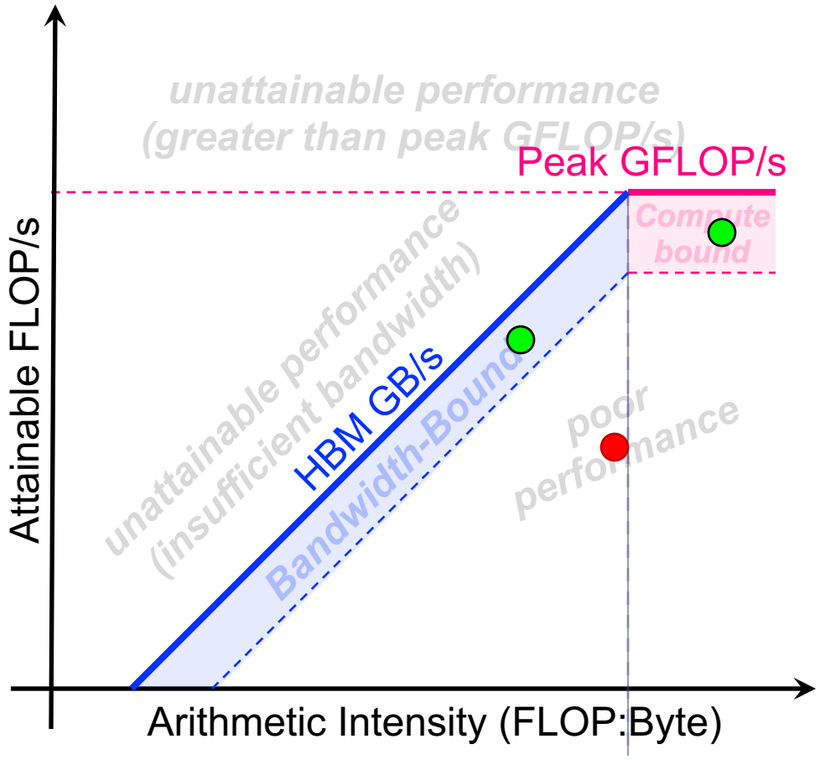
Transition @ AI ==
Peak GFLOP/s / Peak GB/s ==
'Machine Balance'

(DRAM) Roofline Model

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

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

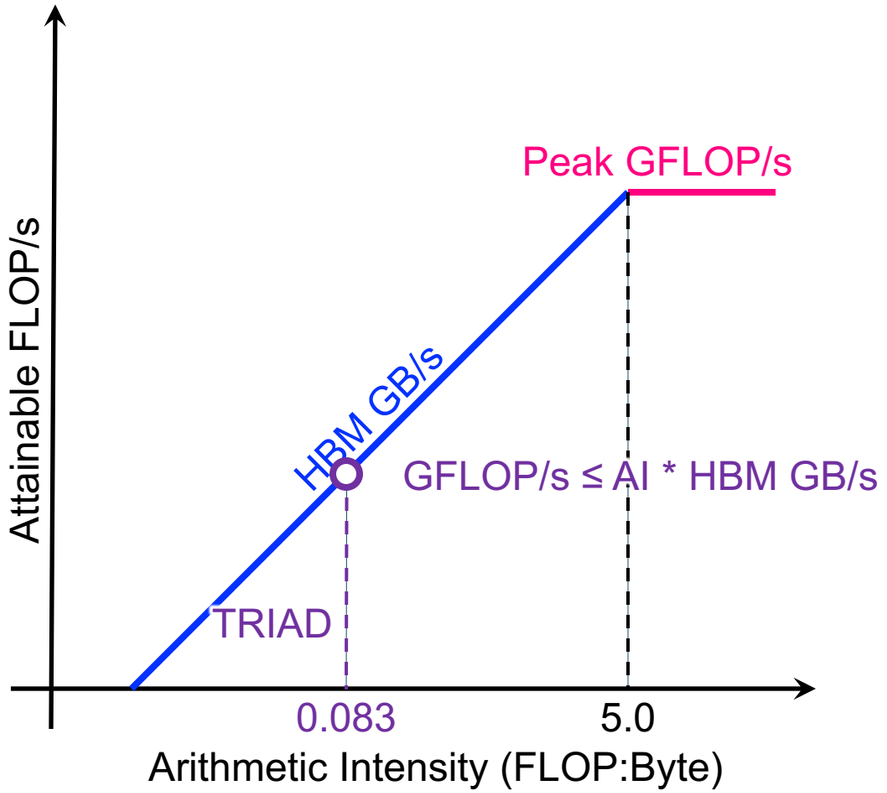
Roofline Example #1

- 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…

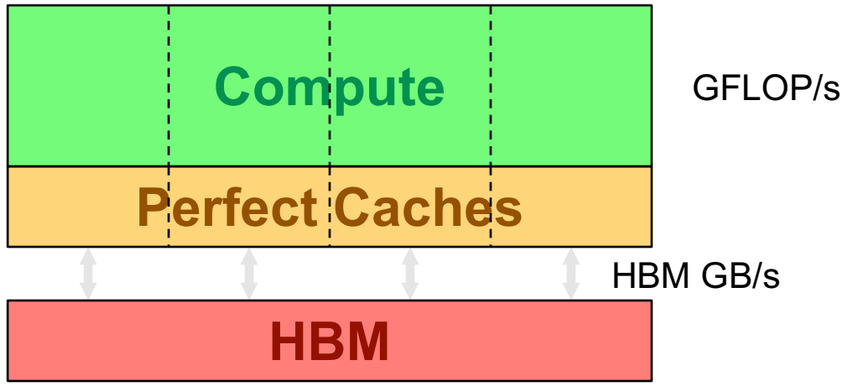
```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

  - 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…

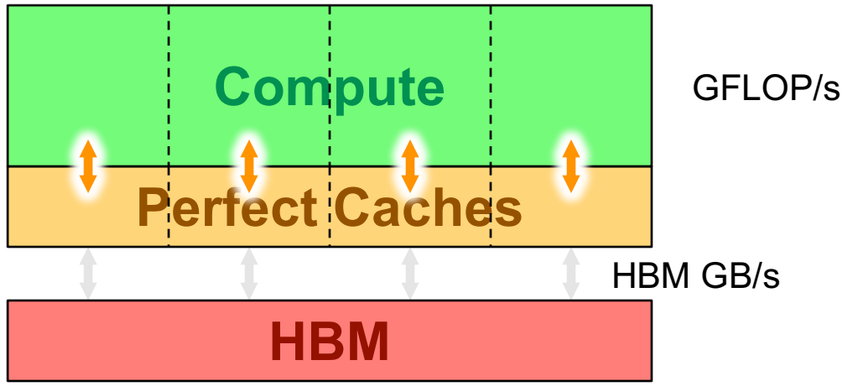



```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                 + old[k  ][j  ][i-1]
                 + old[k  ][j  ][i+1]
                 + old[k  ][j-1][i  ]
                 + old[k  ][j+1][i  ]
            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

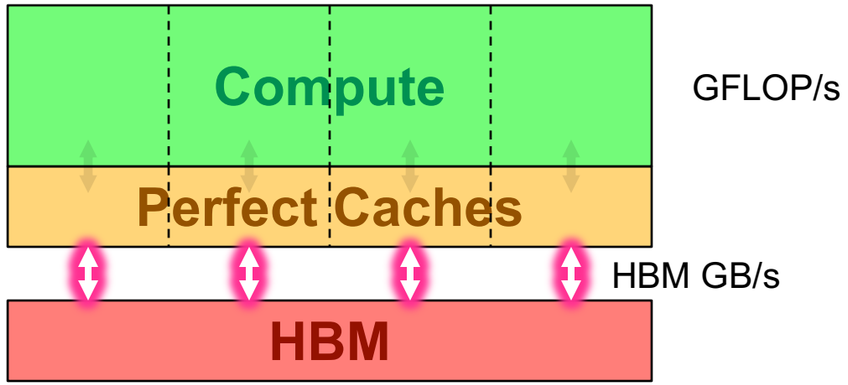
    (measured at the L1)




```
#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

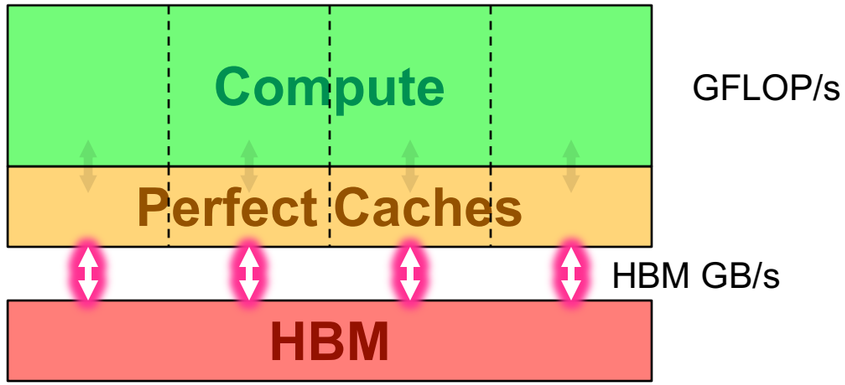
- Conversely, 7-point constant coefficient stencil…
  - 7 FLOPs
  - 8 memory references (7 reads, 1 store) per point
  - **Ideally, cache will filter all but 1 read and 1 write per point**




```
#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)**



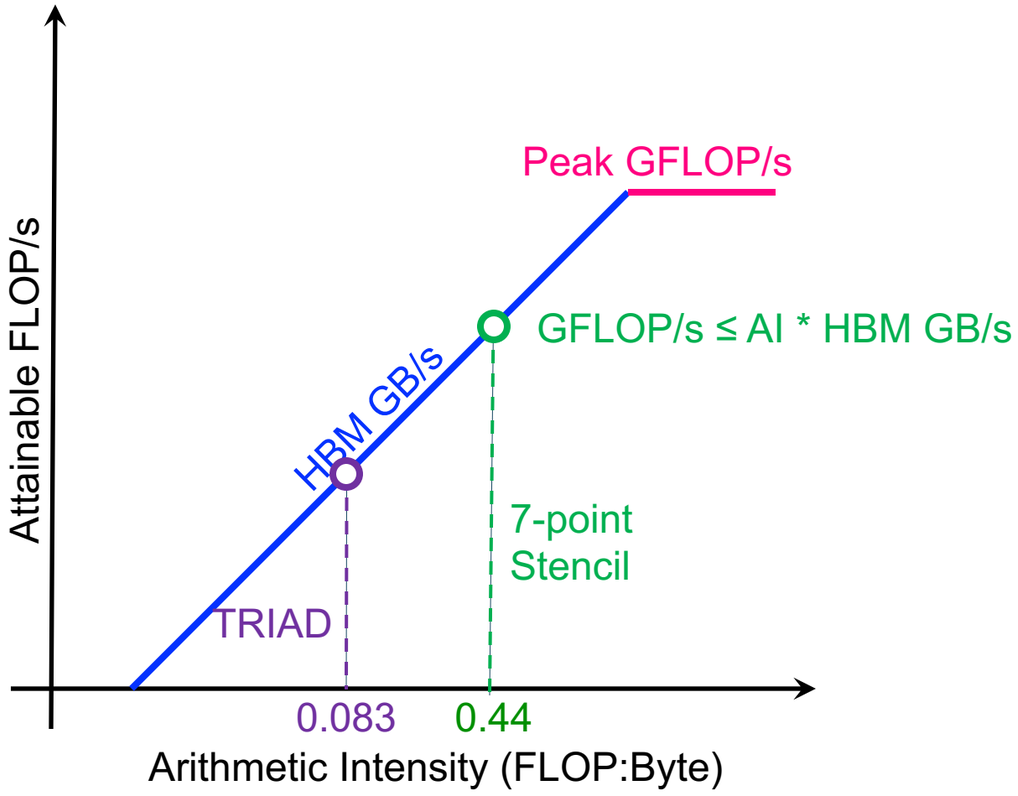

```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                 + old[k  ][j  ][i-1]
                 + old[k  ][j  ][i+1]
                 + old[k  ][j-1][i  ]
                 + old[k  ][j+1][i  ]
            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

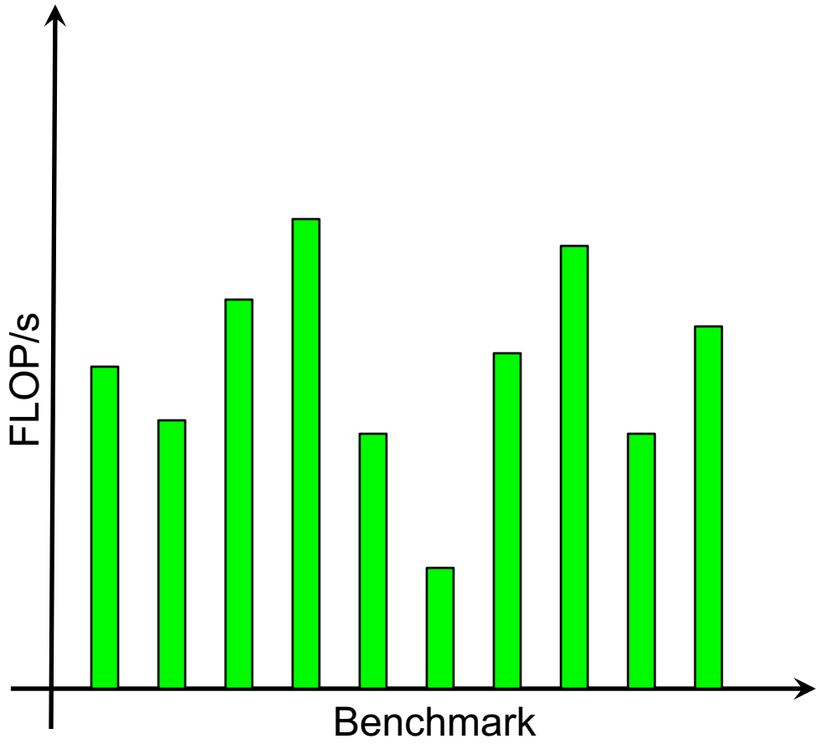
    == memory bound, but 5x the FLOP rate as TRIAD

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



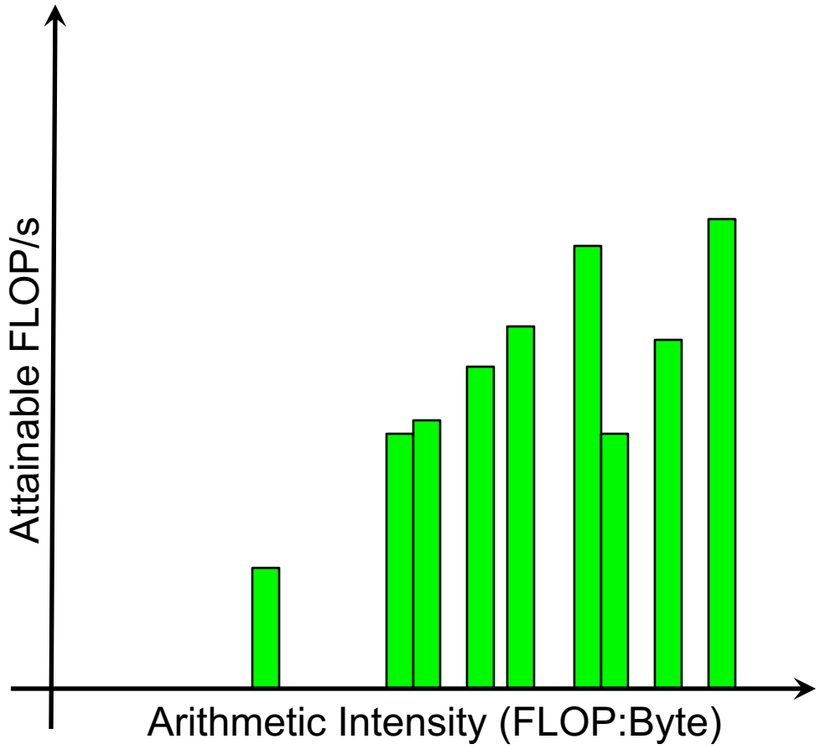
Are we getting good performance?

- Think back to our mix of benchmarks…



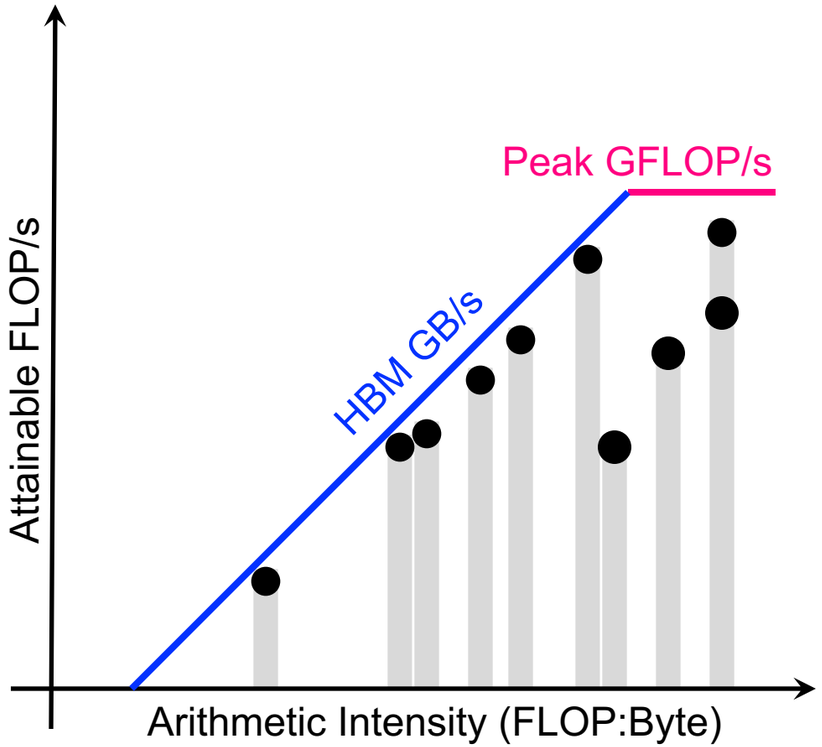
Are we getting good performance?

- We can sort benchmarks by arithmetic intensity…



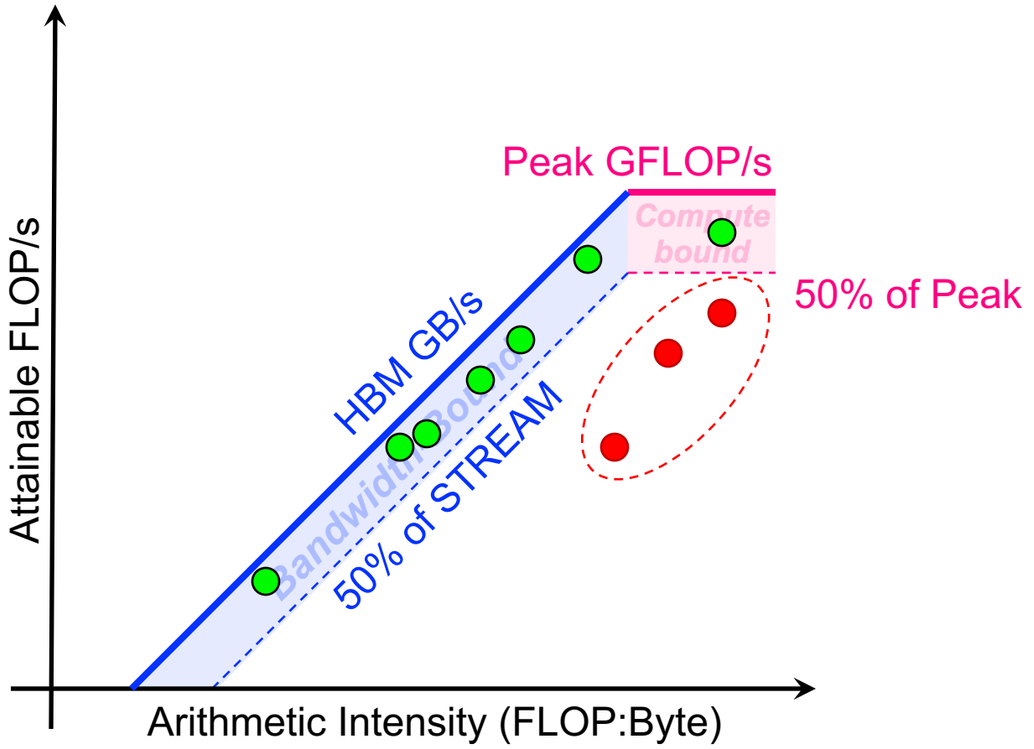
Are we getting good performance?

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



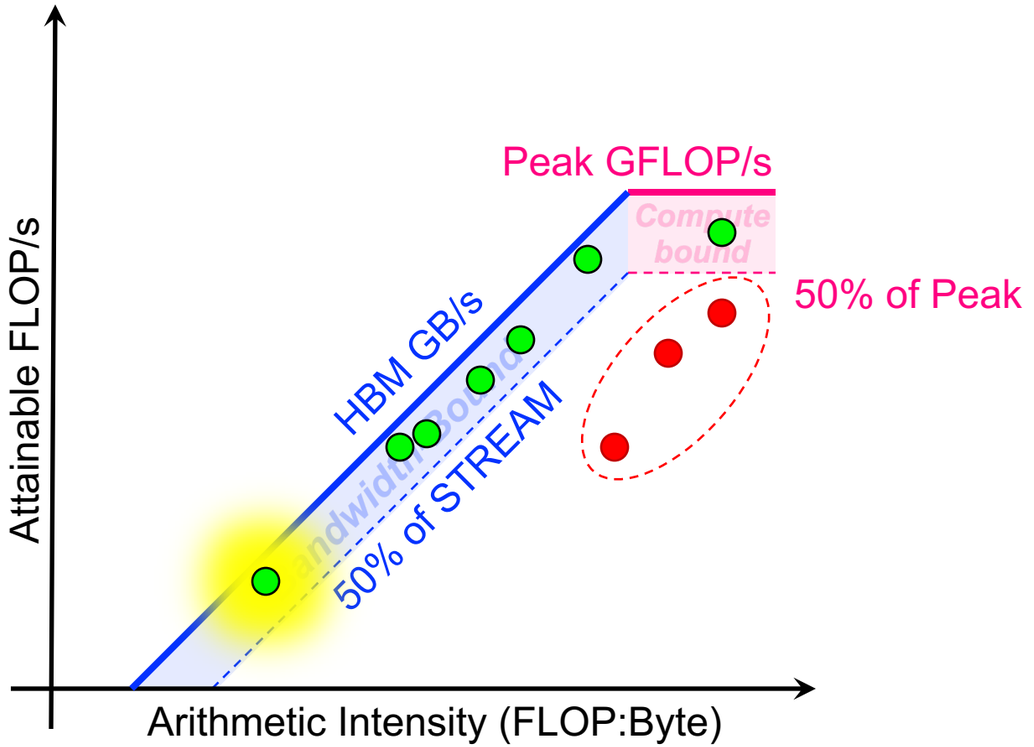
Are we getting good performance?

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



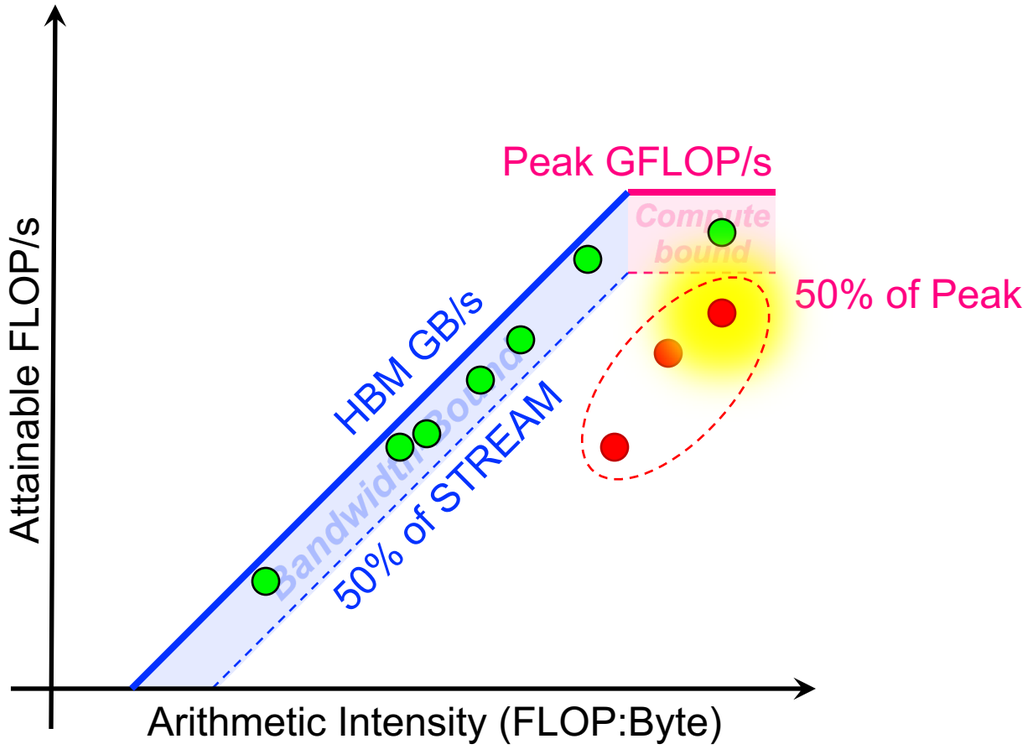
Are we getting good performance?

- 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



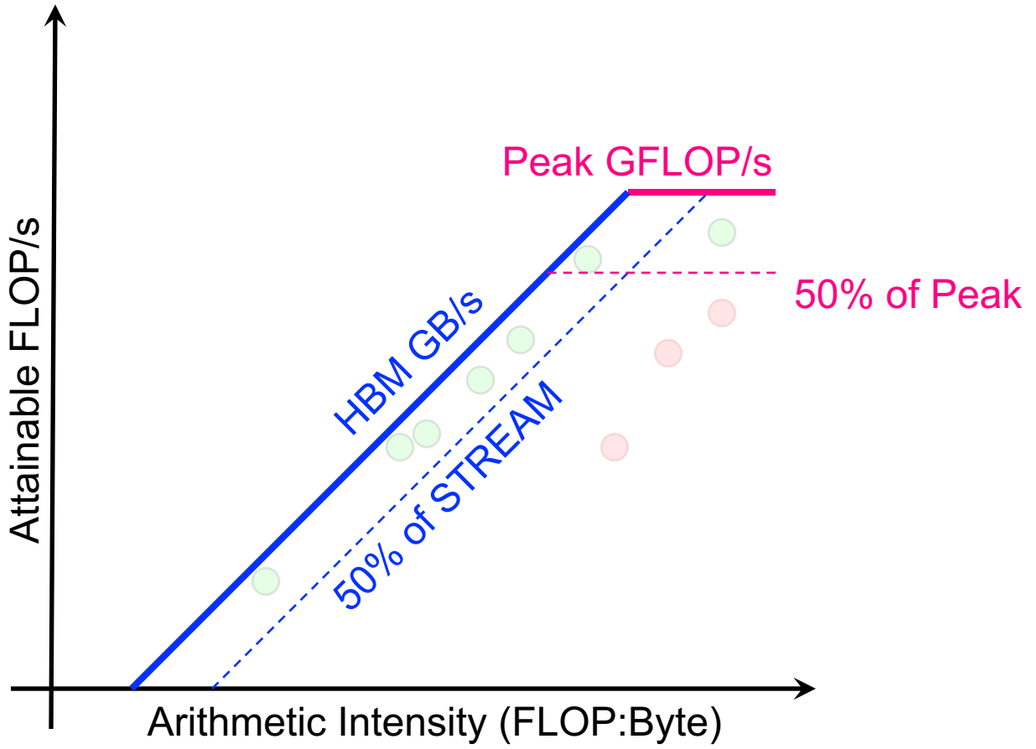
Are we getting good performance?

- 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)



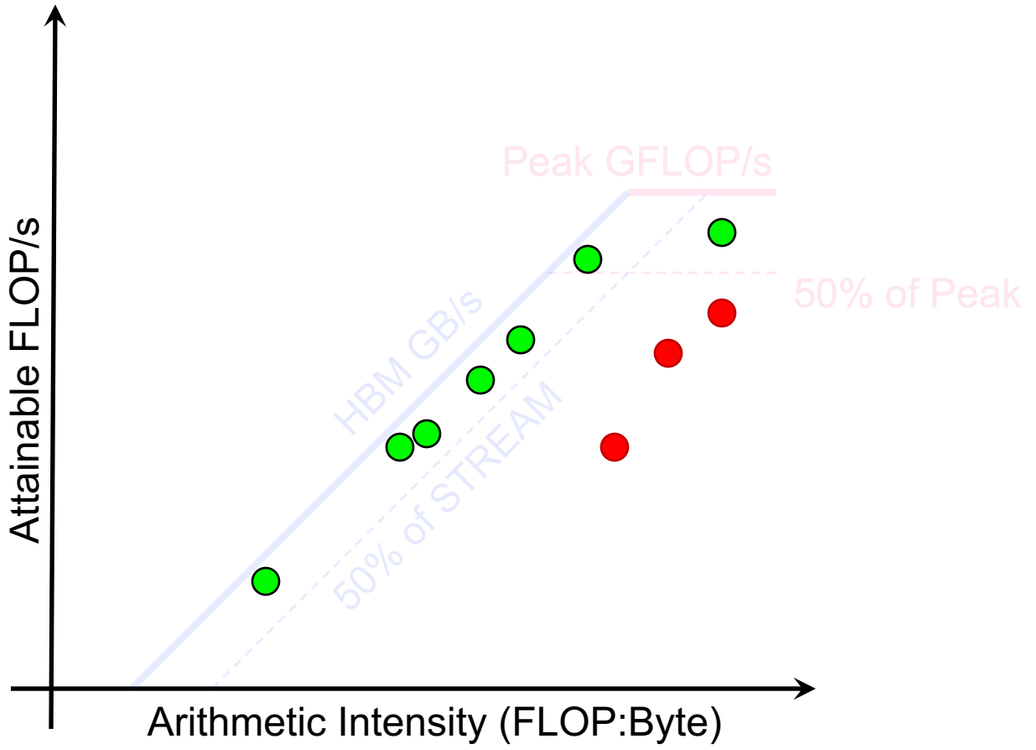
Recap: Roofline is made of two components

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



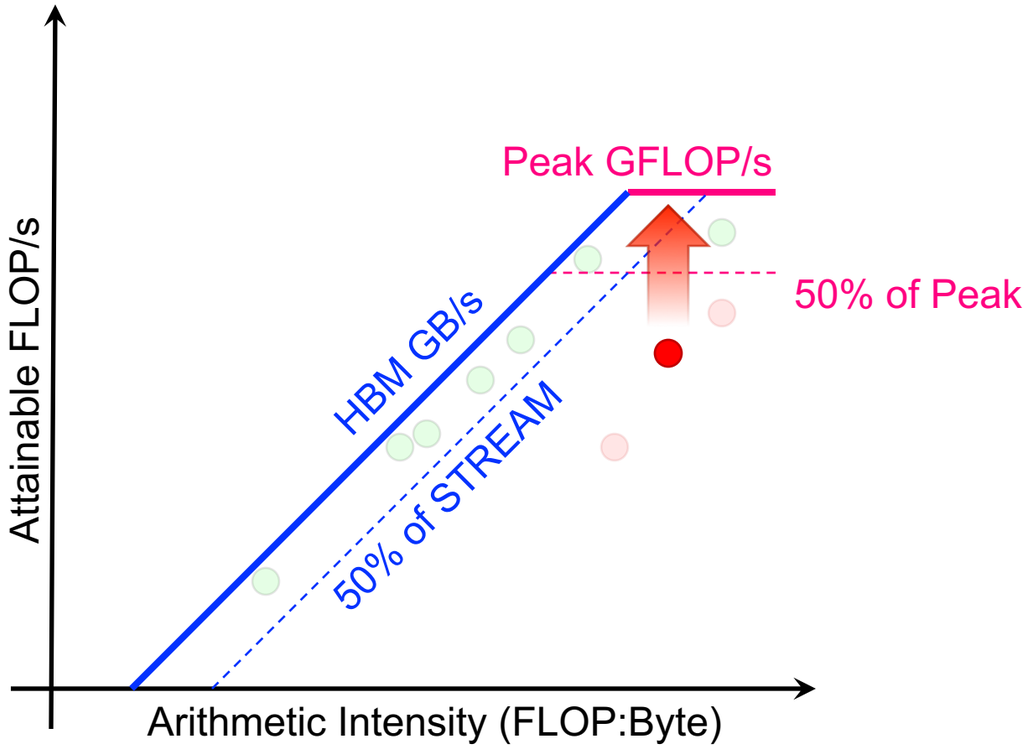
Recap: Roofline is made of two components

- 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 GFLOP's and GB's (**Application Instrumentation**)
  - Unique to each application
  - Unique to each architecture



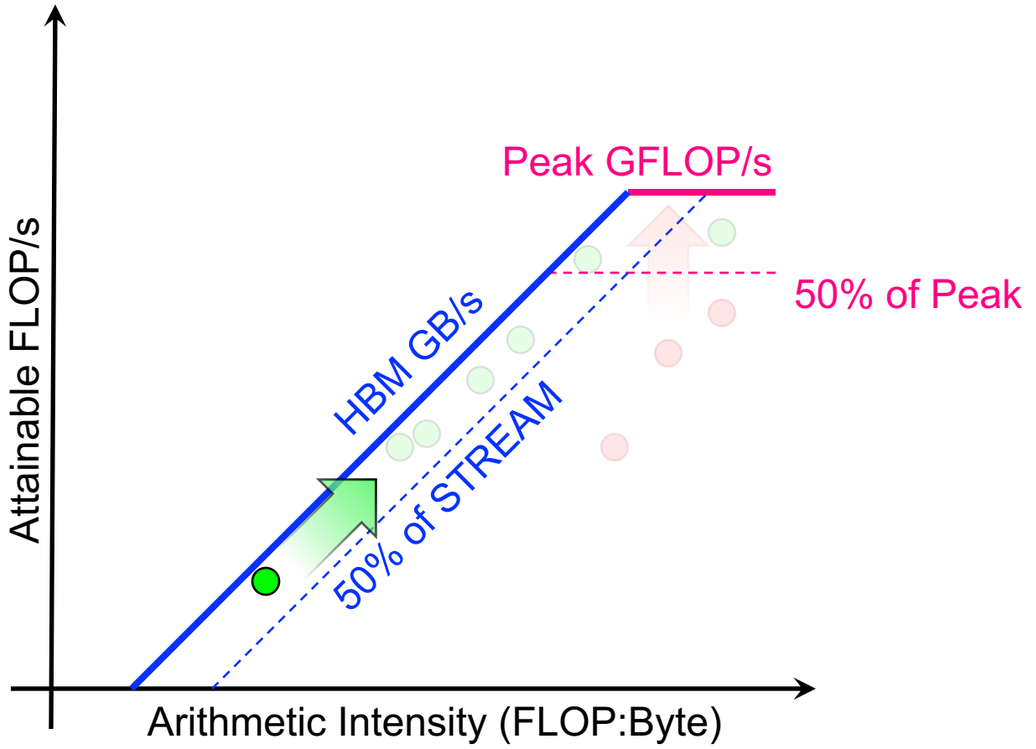
Recap: Optimization Strategy

1. Get to the Roofline



Recap: Optimization Strategy

1. Get to the Roofline
2. Increase Arithmetic Intensity when bandwidth-limited
   - Reducing data movement increases AI
   - Increasing AI increases performance when bandwidth-bound




How can performance ever be below the Roofline?

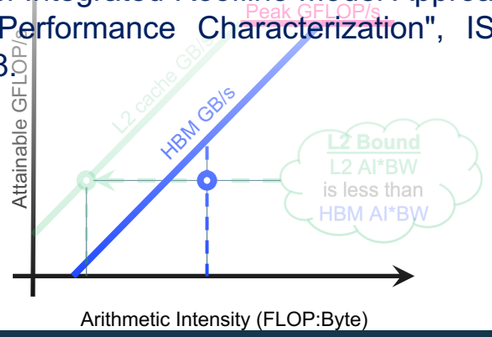
How can performance be below the Roofline?

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

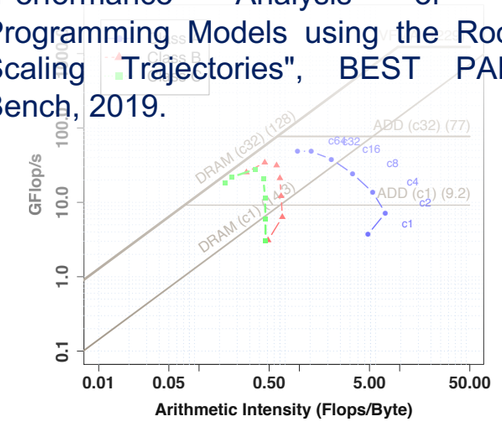
T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, S. Williams, "A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization", ISC, 2018.



Lack of Parallelism…
- Idle Cores/SMs
- Insufficient ILP/TLP
- Divergence and Predication

… Roofline Scaling Trajectories

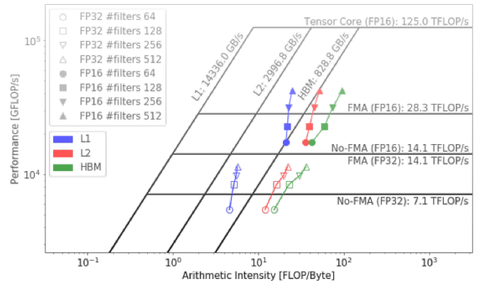
K. Ibrahim, S. Williams, L. Oliker, "Performance Analysis of GPU Programming Models using the Roofline Scaling Trajectories", BEST PAPER, Bench, 2019.



Not enough of Vector/Tensor instr.
- No FMA
- Mixed Precision
- No Tensor Core OPs

… Additional Ceilings

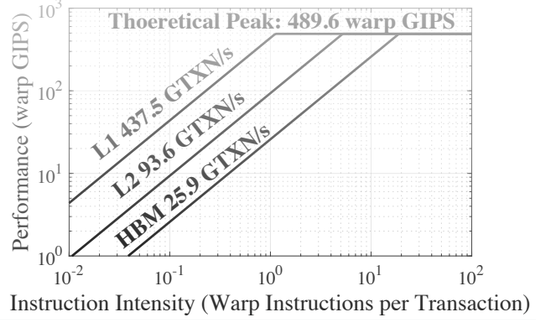
C. Yang, T. Kurth, S. Williams, "Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system", CCPE, 2019.

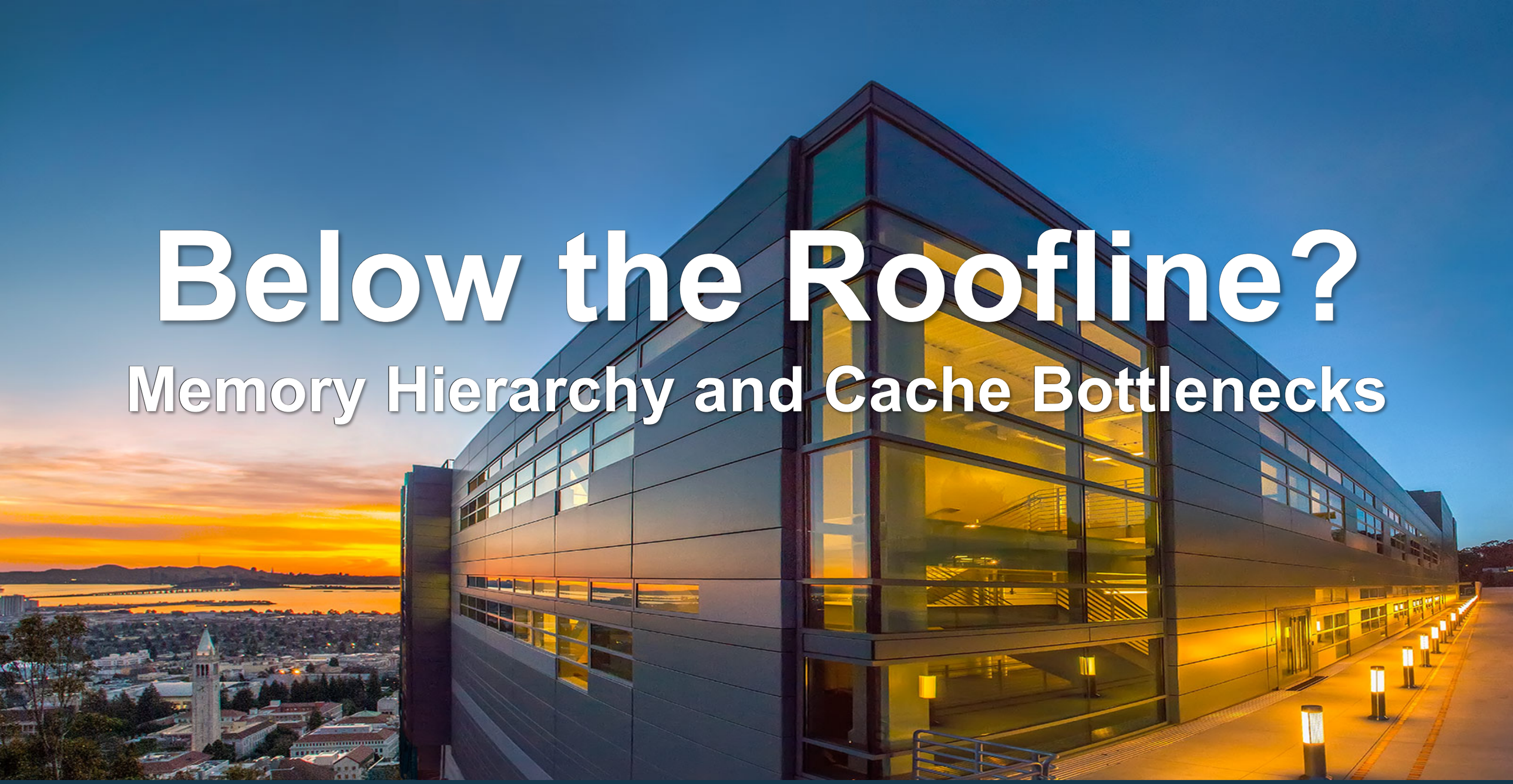


Integer-heavy Codes…
- Non-FP inst. impede FLOPs
- No FP instructions

… The Instruction Roofline Model

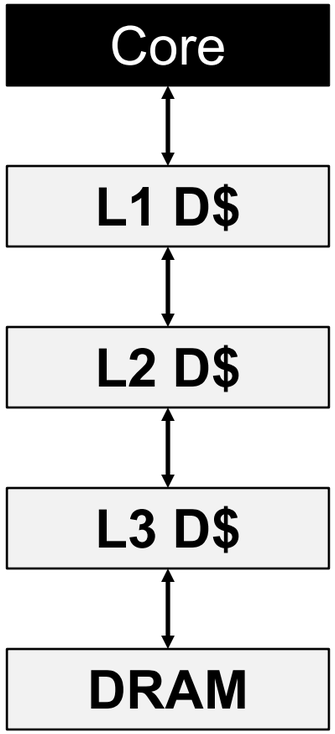
N. Ding, S. Williams, "An Instruction Roofline Model for GPUs", BEST PAPER, PMBS, 2019.




Below the Roofline?
Memory Hierarchy and Cache Bottlenecks

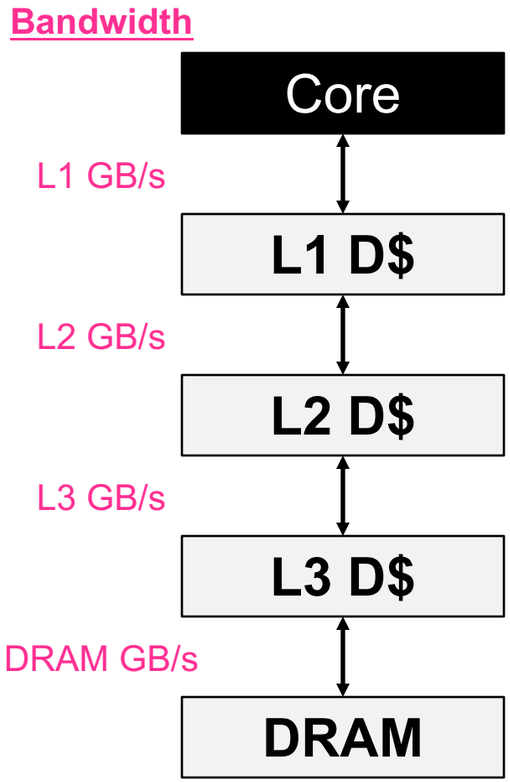
Memory Hierarchy

- 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)



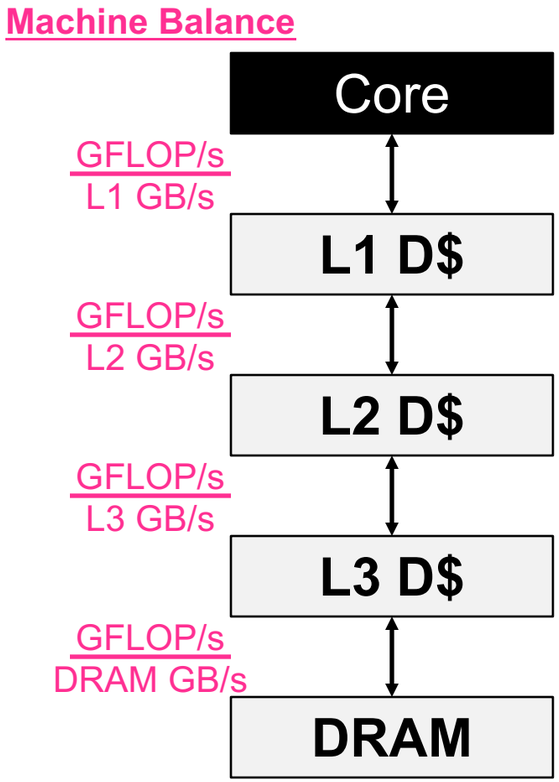
Memory Hierarchy

- CPUs/GPUs have different bandwidths for each level



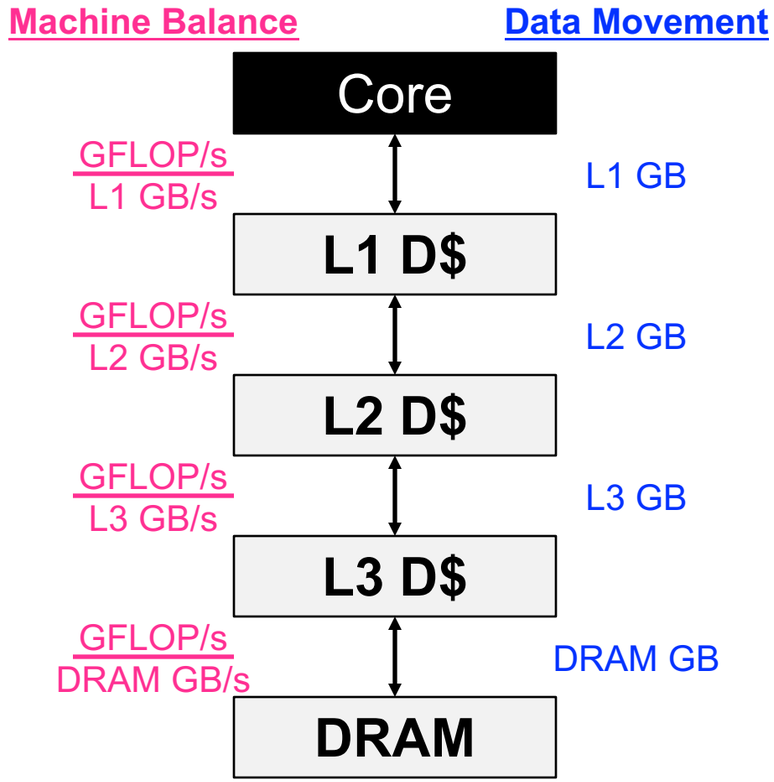
Memory Hierarchy

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



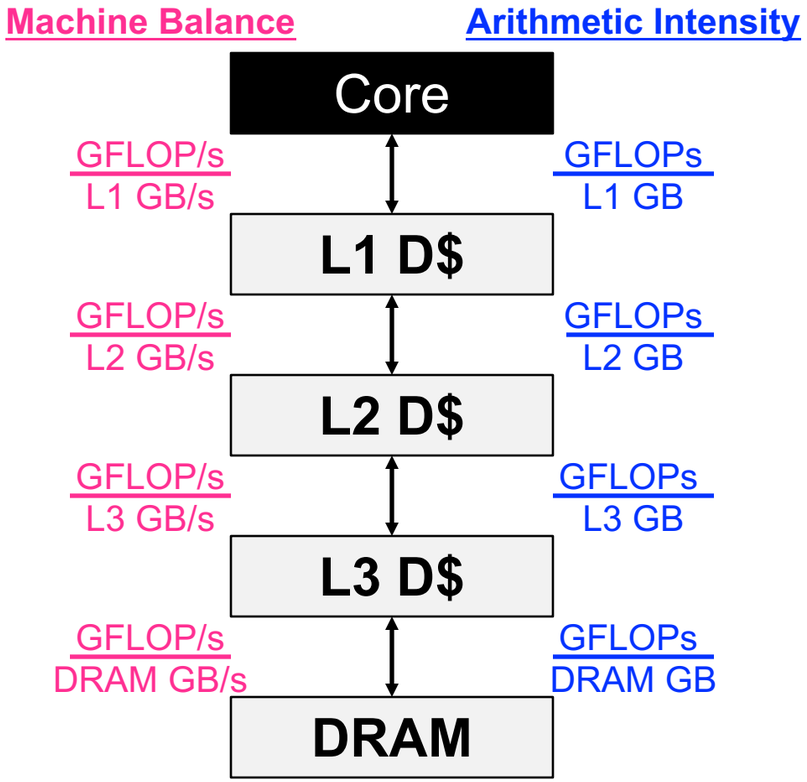
Memory Hierarchy

- 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

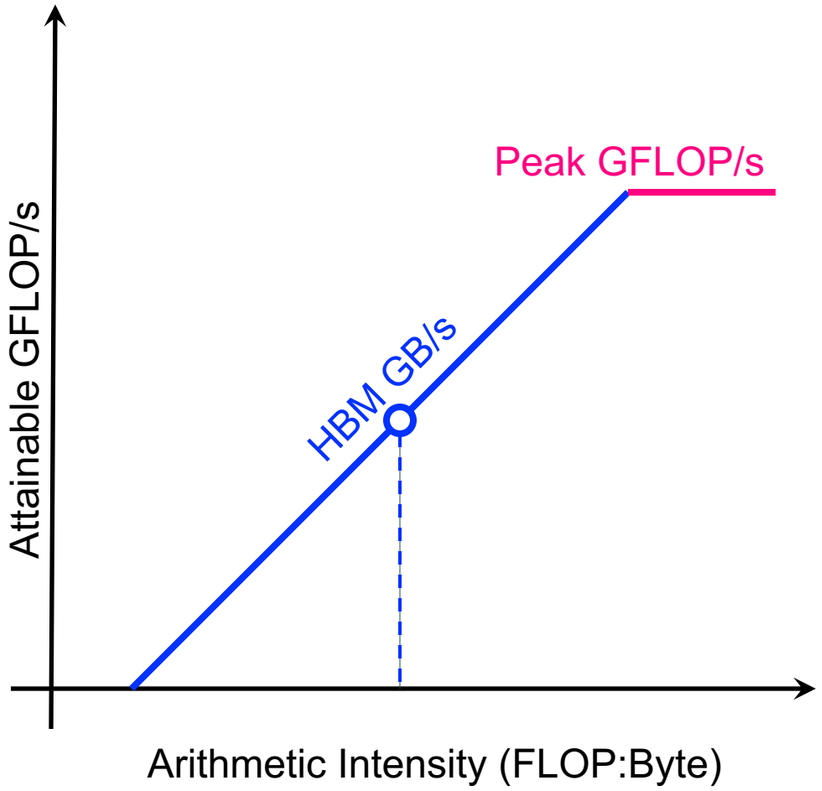
- For each additional level of the memory hierarchy, we can add another term to our model…

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

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

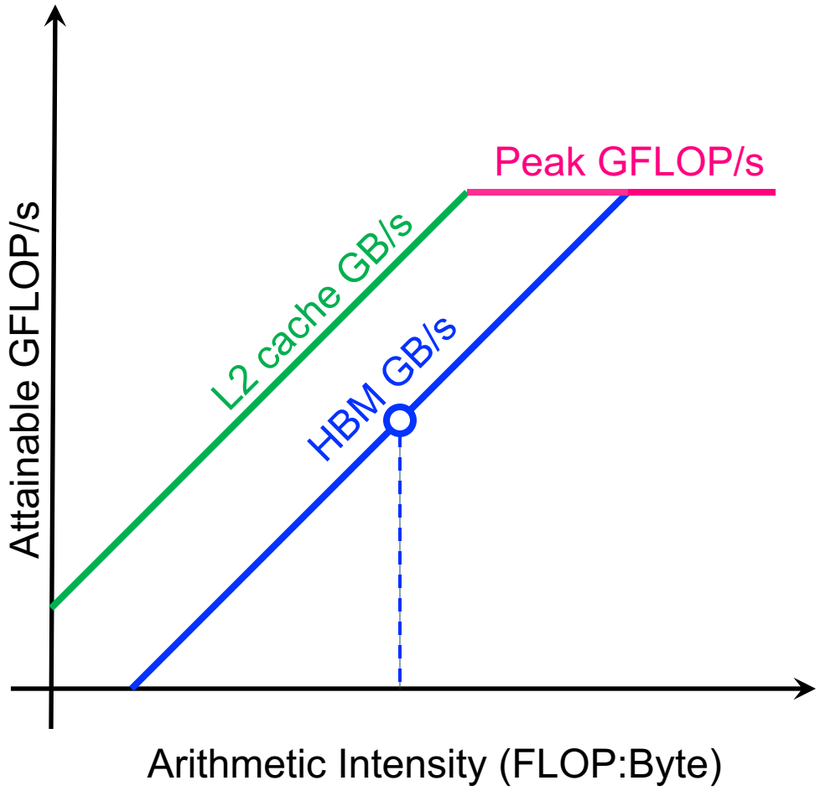
Cache Bottlenecks

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



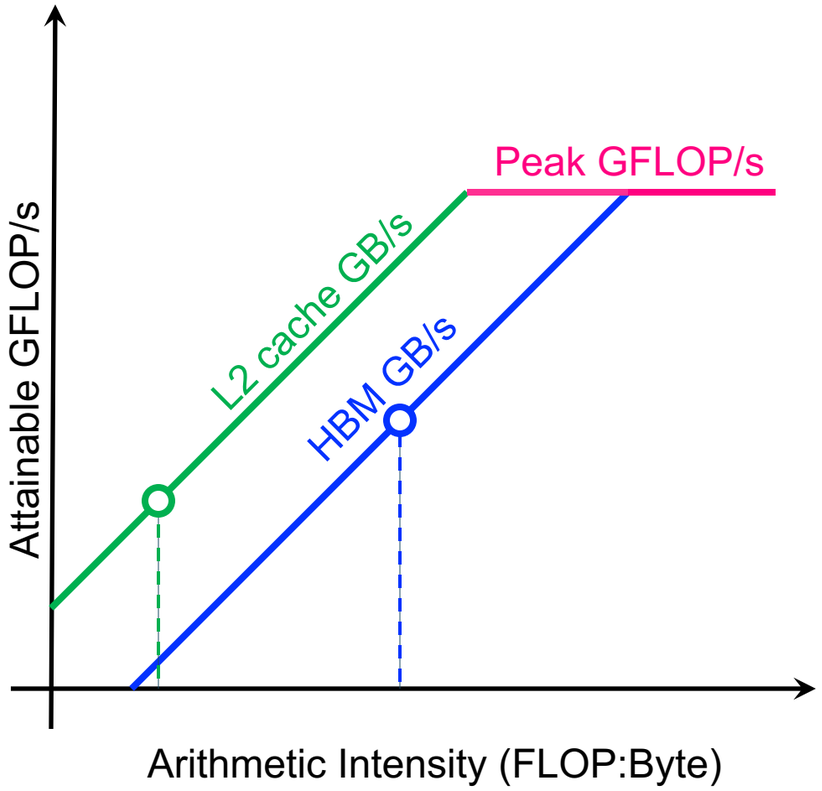
Cache Bottlenecks

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



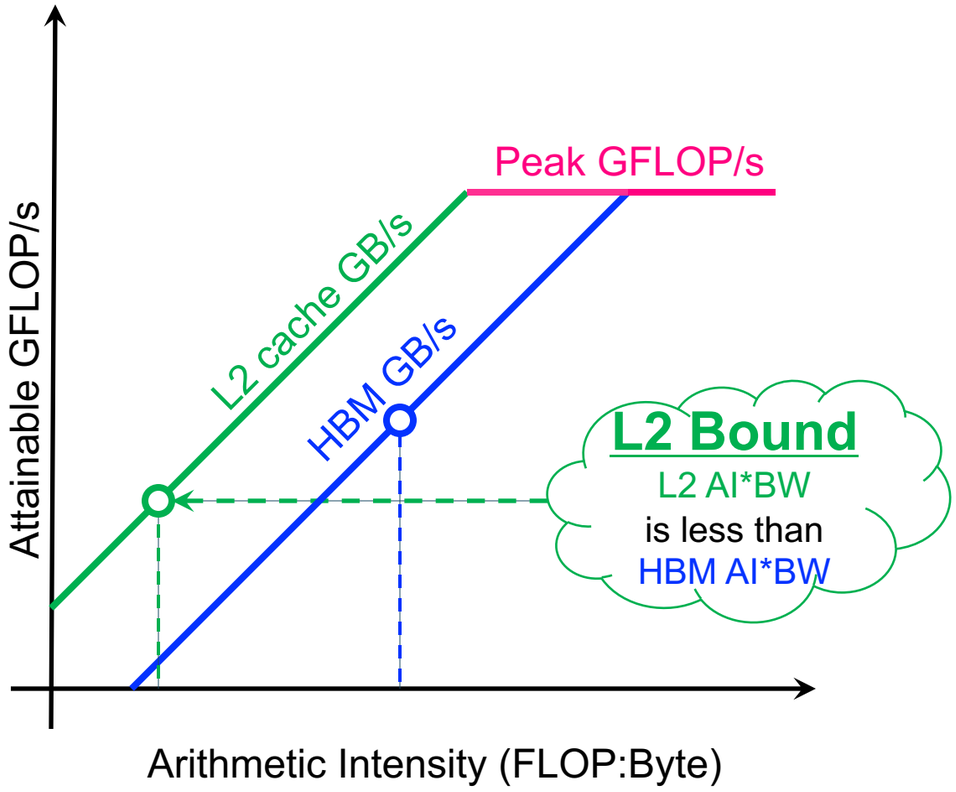
Cache Bottlenecks

- 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



Cache Bottlenecks

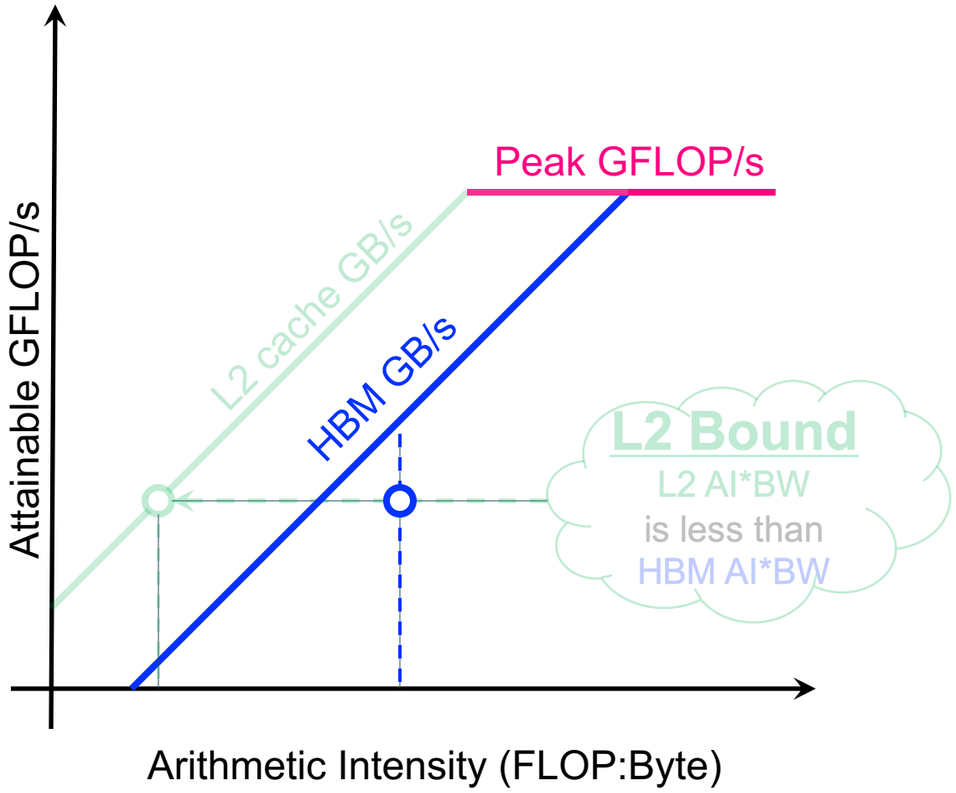
- Plot equation in a single figure…
  - "**Hierarchical Roofline**" Model
  - Bandwidth ceiling (diagonal line) for each level of memory
  - Arithmetic Intensity (dot) for each level of memory
  - ➢ **performance is ultimately the minimum of these bounds**



Cache Bottlenecks

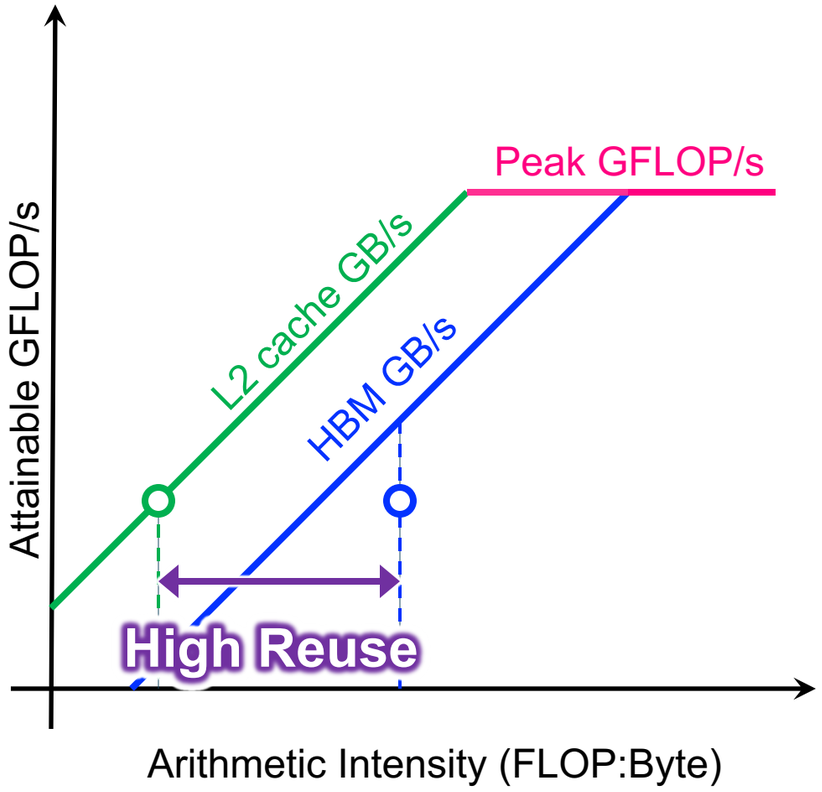
- Plot equation in a single figure…
  - "**Hierarchical Roofline**" Model
  - Bandwidth ceiling (diagonal line) for each level of memory
  - Arithmetic Intensity (dot) for each level of memory
  - ➢ **performance is ultimately the minimum of these bounds**

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



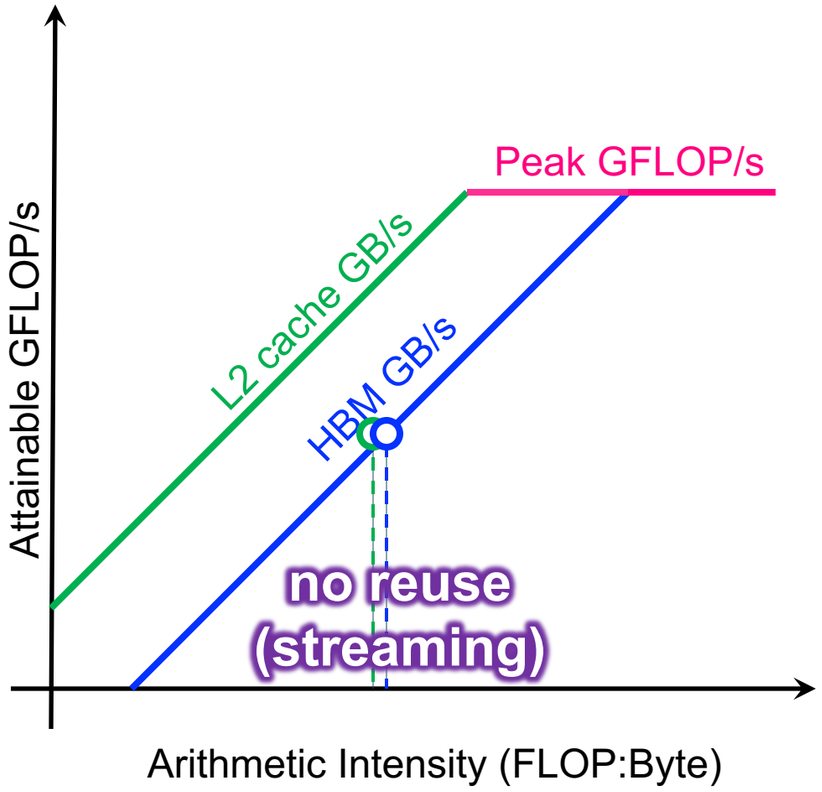
Cache Hit Rates

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



Cache Hit Rates

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




Below the Roofline?
Lack of Parallelism

Roofline and Parallelism

- We've assumed we can always hit either peak GFLOP/s or peak GB/s

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

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

Roofline and Parallelism

- We've assumed we can always hit either peak GFLOP/s or peak GB/s

- But all CPUs and GPUs are highly parallel architectures
- GFLOP/s and GB/s are a function of how much parallelism we utilize…

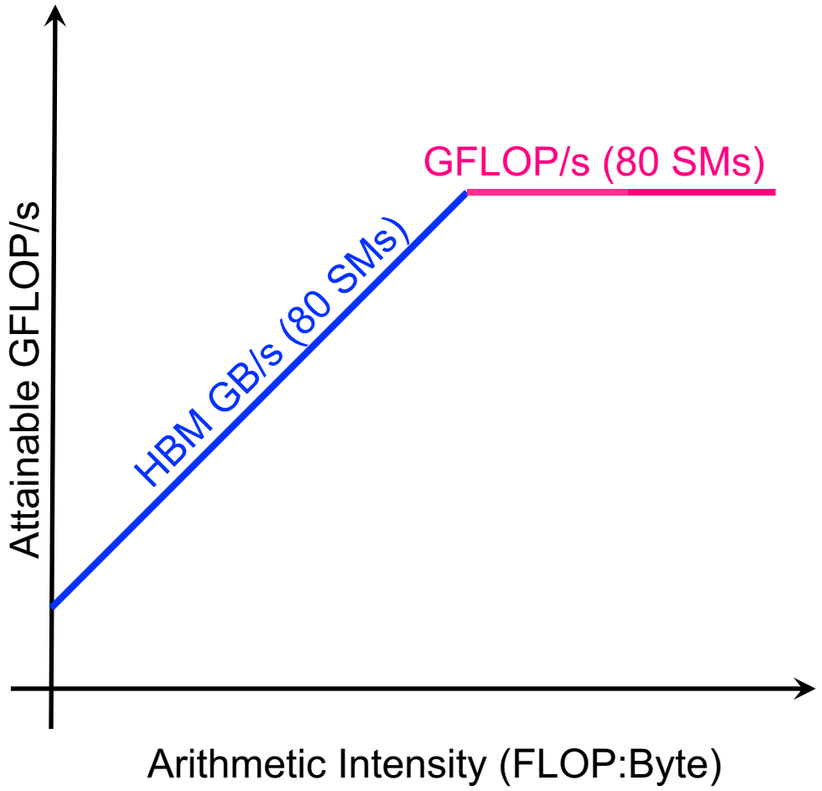
$$\text{GFLOP/s}(P) = \min \begin{cases} \text{GFLOP/s}_{\text{Peak}}(P) \\ \text{AI}_{\text{DRAM}}(P) * \text{GB/s}_{\text{DRAM}}(P) \end{cases}$$

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

AI_{DRAM} is a function of parallelism because cache contention can generate superfluous LLC capacity misses (==DRAM data movement)

Roofline and Parallelism

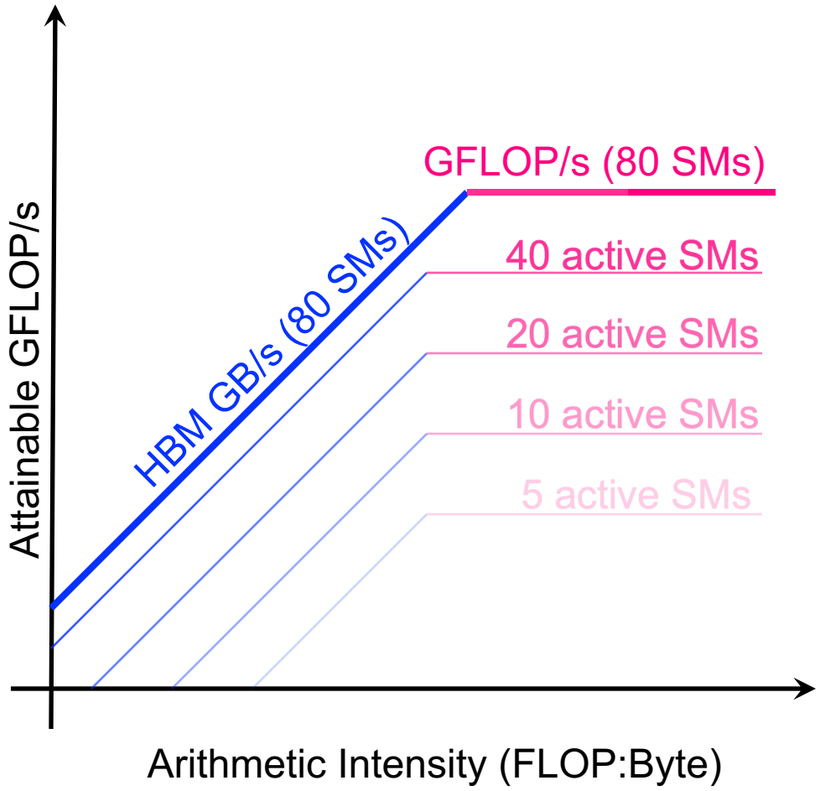
- How do we visualize parallelism in the Roofline?
  - Naively, GFLOP/s(P) and GB/s(P) are proportional to parallelism P
  - SMs are capable of pulling more than their fair share of HBM
  - DVFS implies not true for GFLOP/s



Roofline and Parallelism

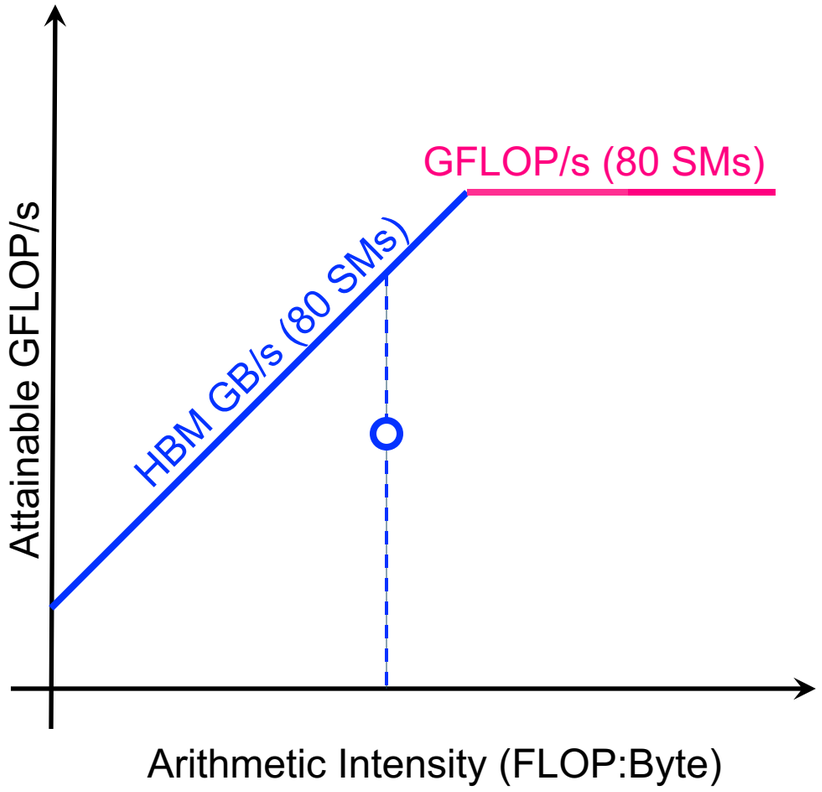
- How do we visualize parallelism in the Roofline?
  - Naively, GFLOP/s(P) and GB/s(P) are proportional to parallelism P
  - SMs are capable of pulling more than their fair share of HBM
  - DVFS implies not true for GFLOP/s

➢ **one must benchmark GFLOP/s and GB/s at each concurrency**



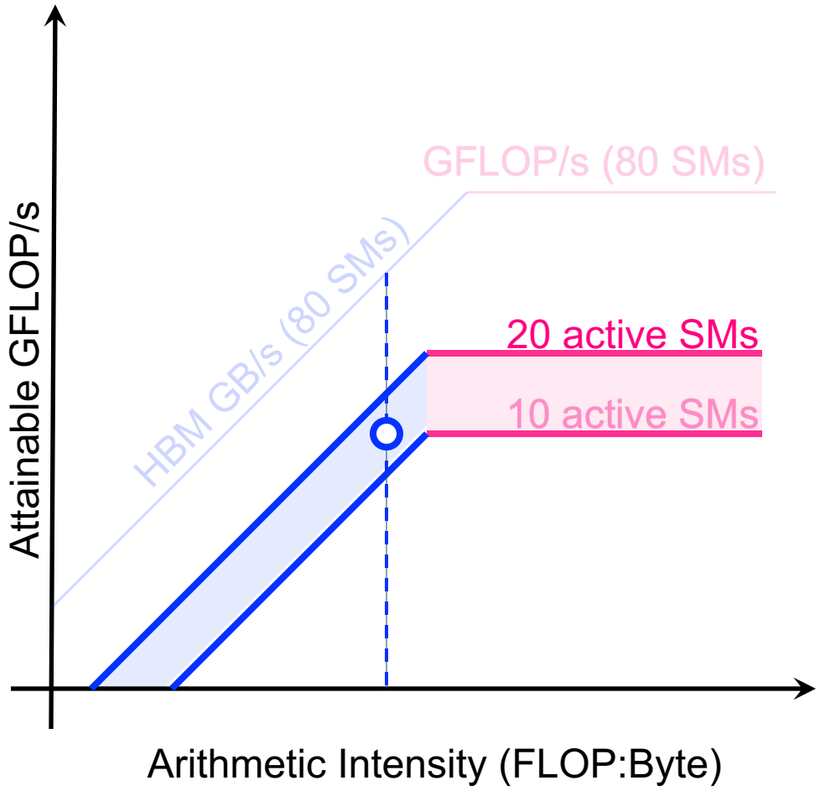
Roofline and Parallelism

- Consider CUDA kernel optimized for Fermi (16 SMs) running on Volta (80 SMs)
  - Performance looks very poor



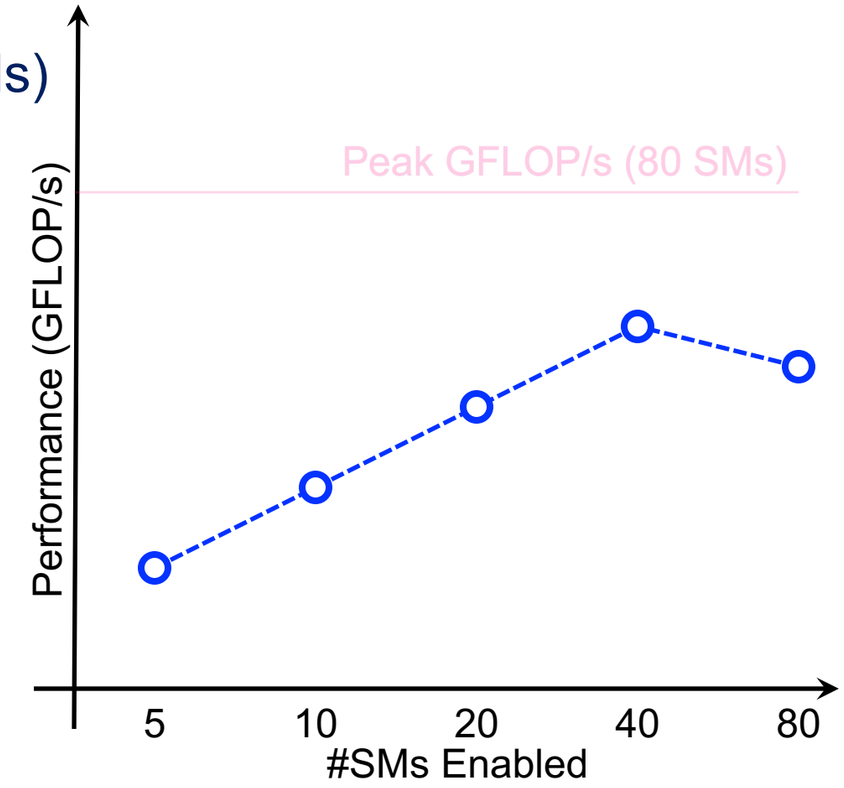
Roofline and Parallelism

- Consider CUDA kernel optimized for Fermi (16 SMs) running on Volta (80 SMs)
  - Performance looks very poor
  - Kernels using only 16 SMs underutilize the V100 architecture.
  - Roofline highlights the fact that performance is constrained by a lack of software parallelism



Roofline Scaling Trajectories

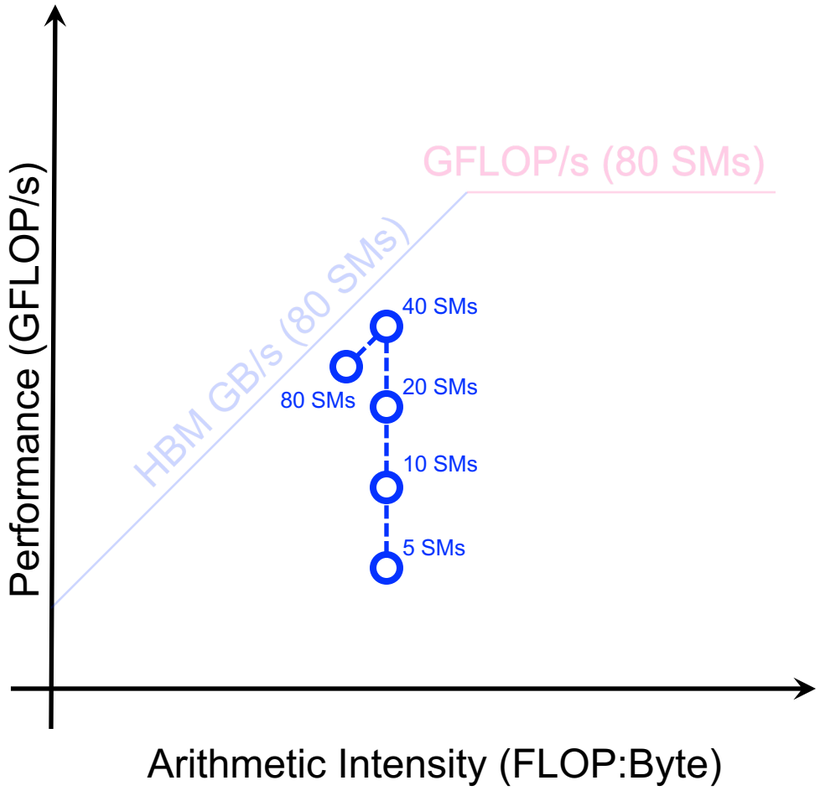
- Traditional Scalability:
  - Plot performance vs. concurrency (#cores or #SMs)
  - Observation without much insight
  - Why does performance decrease?



Khaled Ibrahim, Samuel Williams, Leonid Oliker, "Performance Analysis of GPU Programming Models using the Roofline Scaling Trajectories", Bench, November, 2019.

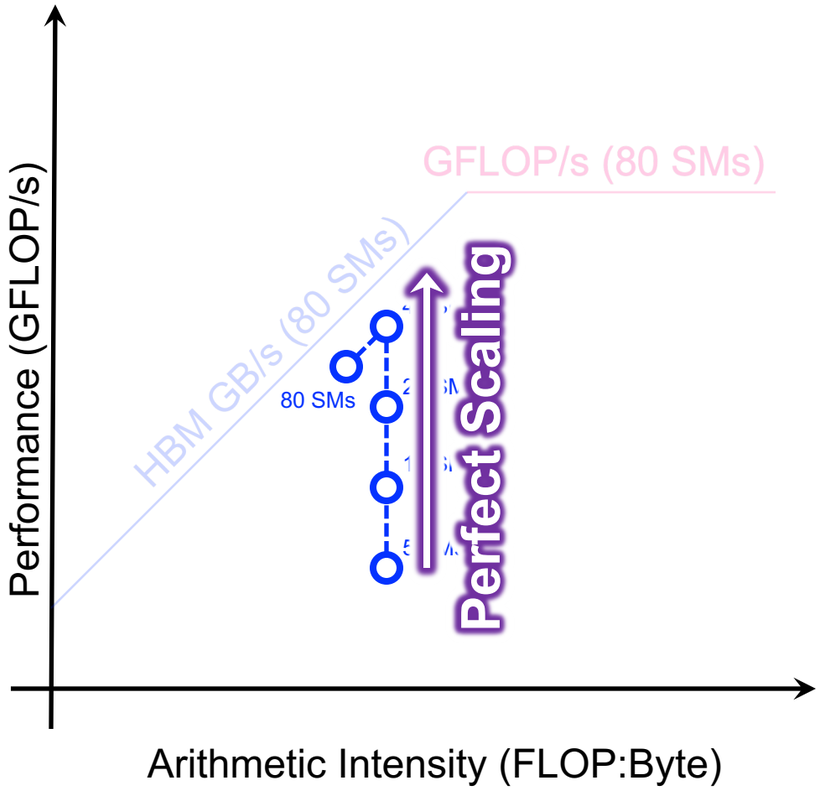
Roofline Scaling Trajectories

- Khaled Ibrahim leveraged Roofline to understand the interplay between concurrency, data locality, and performance
- **Roofline Scaling Trajectories**
  - Measure (AI,GFLOP/s) for each concurrency
  - Plot as a trendline on Roofline



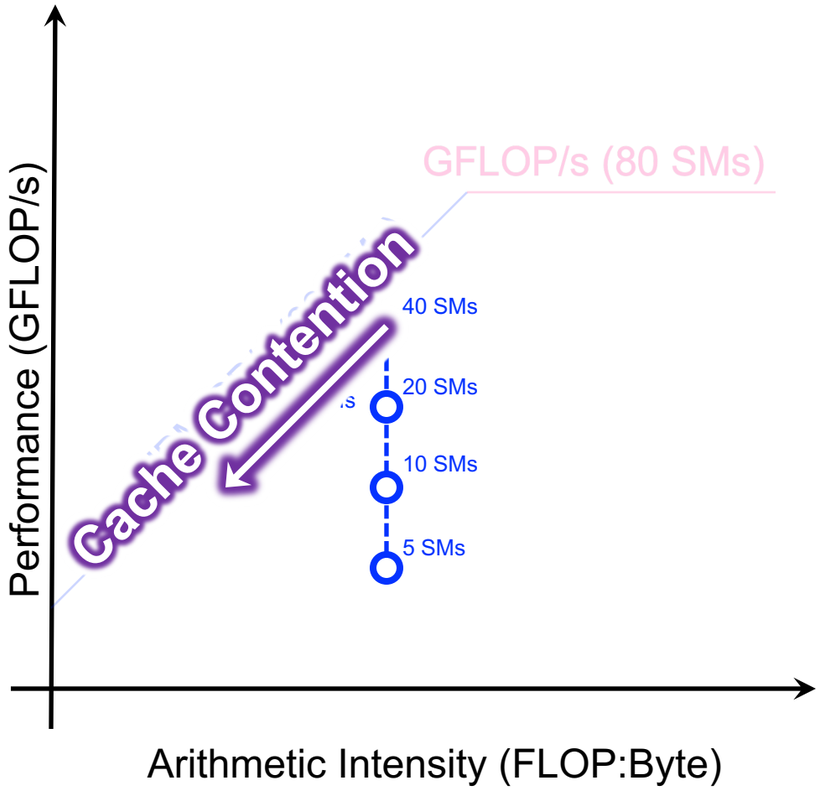
Roofline Scaling Trajectories

- Khaled Ibrahim leveraged Roofline to understand the interplay between concurrency, data locality, and performance
- **Roofline Scaling Trajectories**
  - Measure (AI,GFLOP/s) for each concurrency
  - Plot as a trendline on Roofline
  - **Perfect scaling is a vertical line**



Roofline Scaling Trajectories

- Khaled Ibrahim leveraged Roofline to understand the interplay between concurrency, data locality, and performance
- **Roofline Scaling Trajectories**
  - Measure (AI,GFLOP/s) for each concurrency
  - Plot as a trendline on Roofline
  - Perfect scaling is a vertical line
  - **Turnover in AI indicates cache capacity exhaustion (extra L2 misses drives down AI)**




Below the Roofline?
Return of CISC

Return of CISC

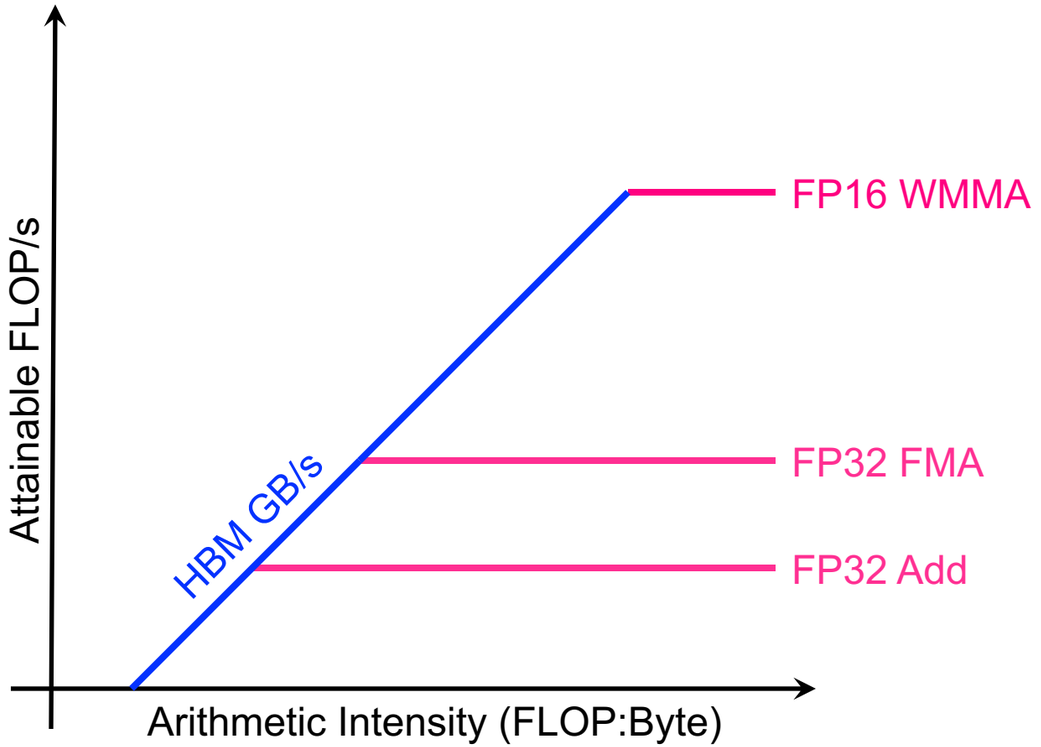
- Vectors have their limits (finite DLP, register file energy scales with VL, etc…)
- Death of Moore's Law is reinvigorating Complex Instruction Set Computing (CISC)

- 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)**

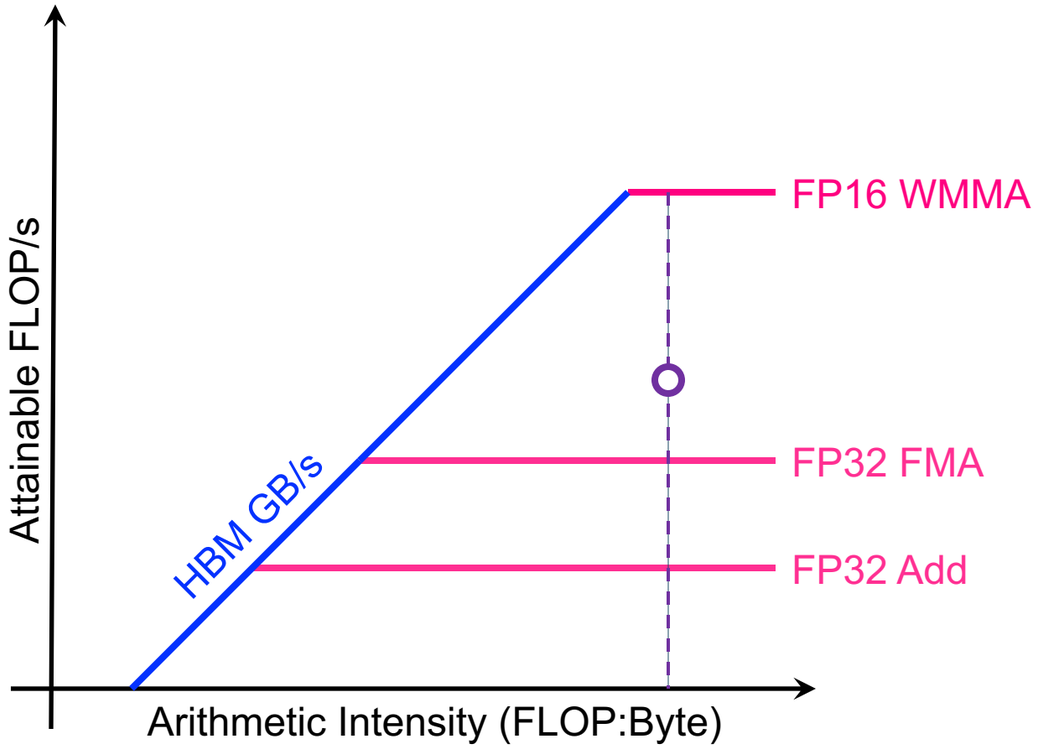
Floating-Point and Mixed Precision Ceilings

- 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



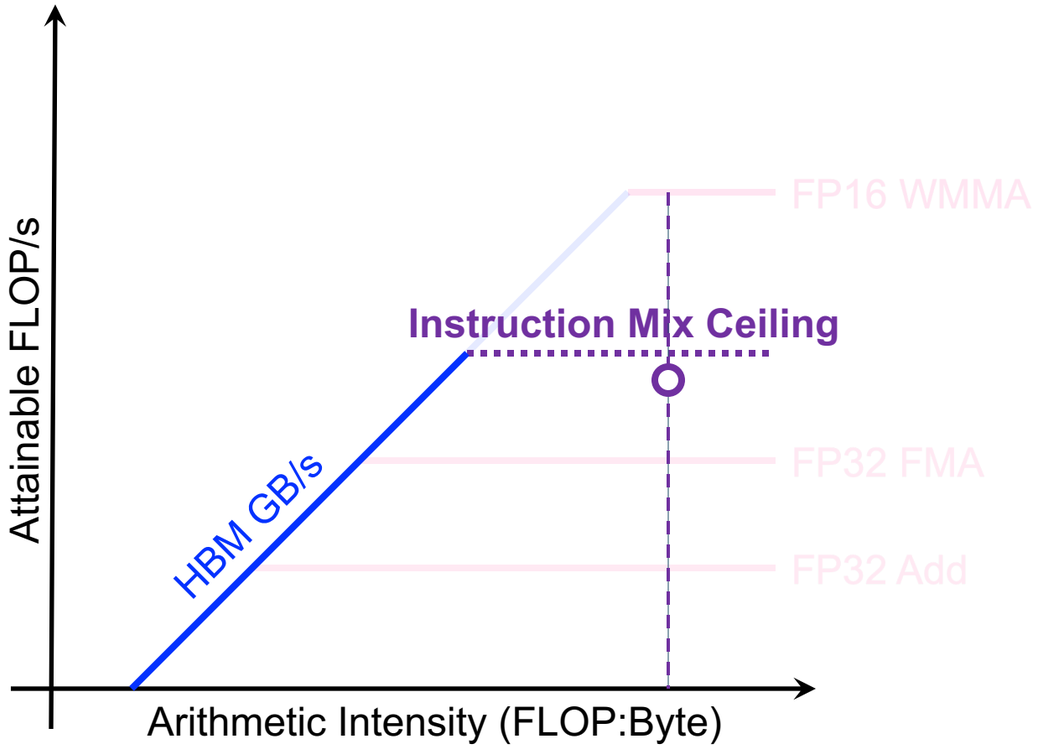
Floating-Point and Mixed Precision Ceilings

- Charlene Yang: 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



Floating-Point and Mixed Precision Ceilings

- Charlene Yang: 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




Below the Roofline?
FPU Starvation

How do we go beyond the FLOP Roofline?

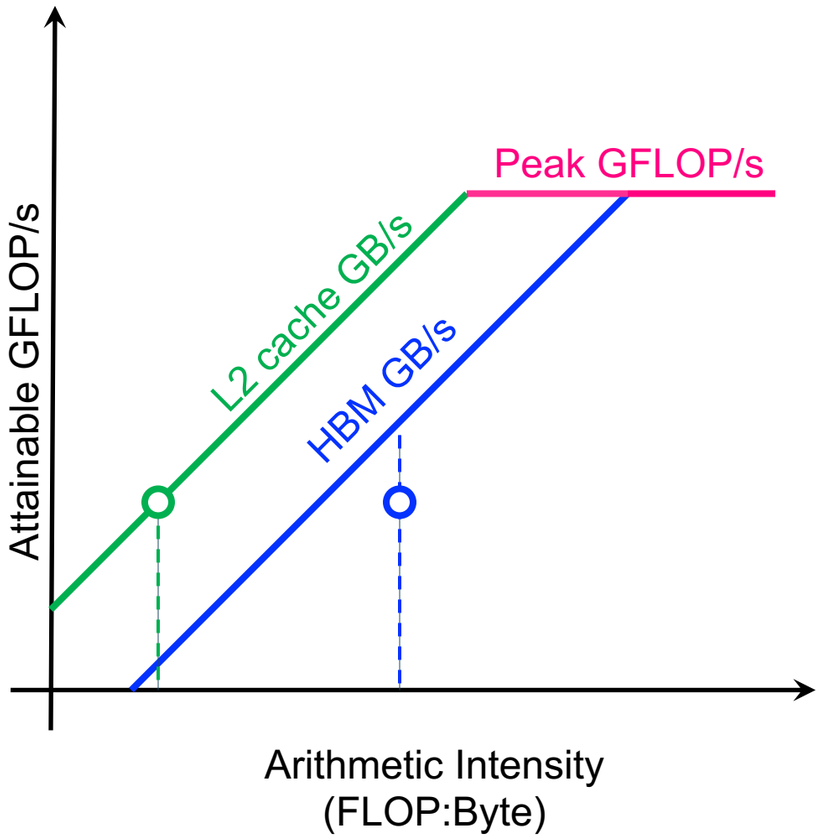
- Think about classifying applications by instruction mix…
  - Heavy floating-point (rare in DOE)
  - Mixed precision
  - Mix of integer and floating-point
  - Integer-only (e.g. bioinformatics, graphs, etc…)

- We've shown the tradition Roofline can address the first two cases, but what about the other two?

- Two options:
  - Redefine FLOPs as (FP+Integer) Ops
  - Count instructions rather than FLOPs

Instruction Roofline

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

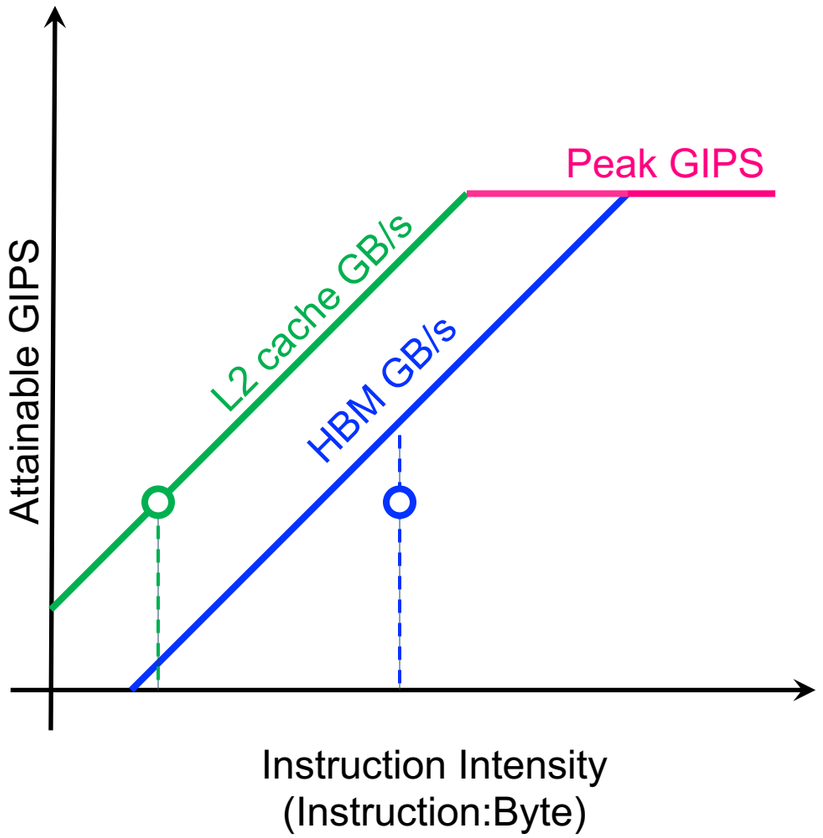


Instruction Roofline

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

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

Instructions per Byte



Instruction Roofline on GPUs

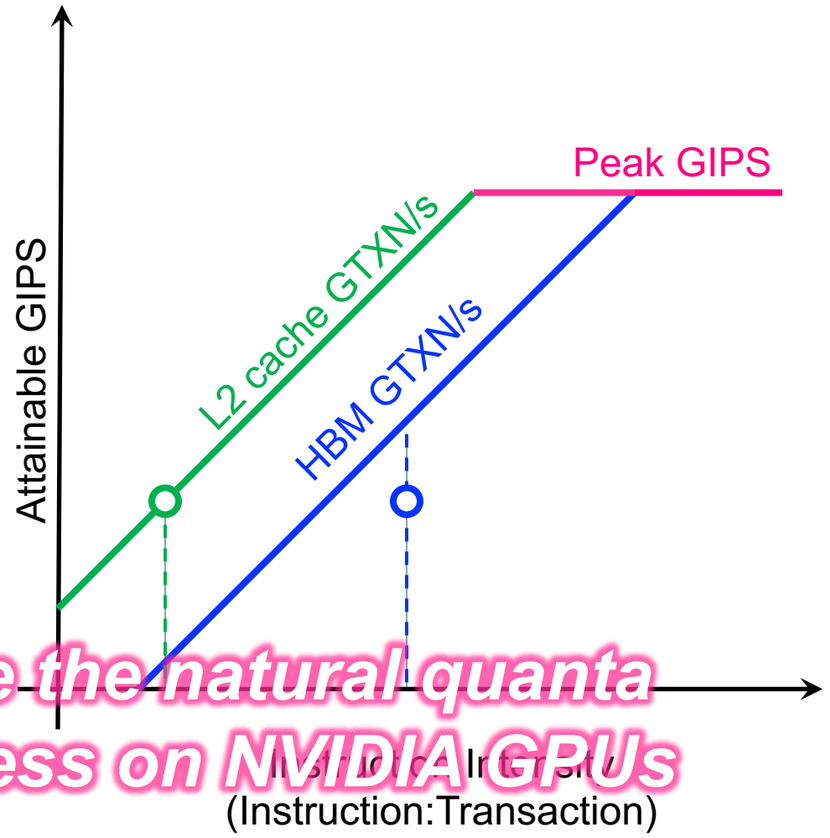
GFLOP/s = min
- **Peak GFLOP/s**
- **AI_{DRAM} * DRAM GB/s**

GIPS = min
- **Peak GIPS**
- **II_{DRAM} * DRAM GB/s**

Use warp Instructions

GIPS = min
- **Peak GIPS**
- **II_{DRAM} * DRAM GTXN/s**

Transactions are the natural quanta for memory access on NVIDIA GPUs



II_x (Instruction Intensity at level "x") =

Instructions / Transactions (to/from level "x")

Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", PMBS, November, 2019.

Instruction Roofline Takeaway

Traditional Roofline

- **Tells us about performance** *(GFLOP/s)*
- Presence of integer instructions has no affect on intensity, but may decrease performance
- Use of FMA, SIMD, vectors, tensors has no affect on intensity, but may increase performance…
- Reducing precision (64b, 32b, 16b) increases arithmetic intensity

Instruction Roofline

- **Tells us about bottlenecks** *(issue and memory)*
- Presence of integer instructions increases intensity and might highlight a bottleneck.
- Use of FMA, SIMD, vectors, tensors decreases intensity and may decrease performance
- Reducing precision has no affect on intensity


Recap

Recap

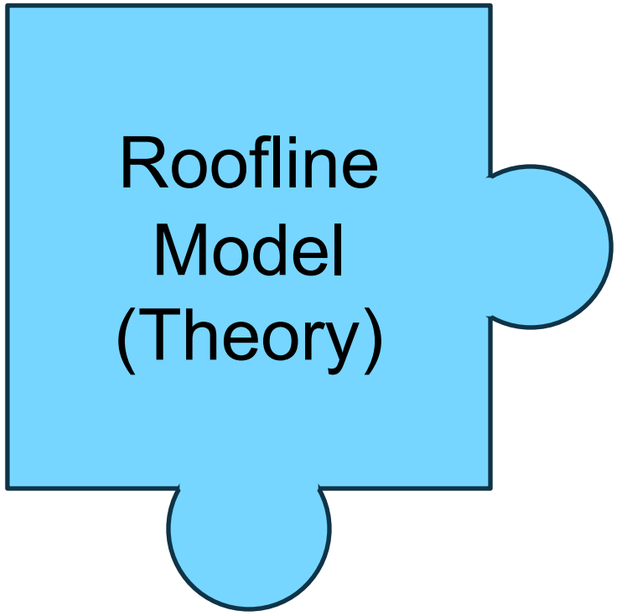
- Roofline bounds performance as a function of Arithmetic Intensity
  - Horizontal Lines = Compute Ceilings
  - Diagonal Lines = Bandwidth Ceilings
  - Bandwidth ceilings are always parallel on log-log scale
  - **Collectively, define an upper limit on performance (speed-of-light)**
- Loop Arithmetic 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 rate)

What is Roofline used for?

- 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

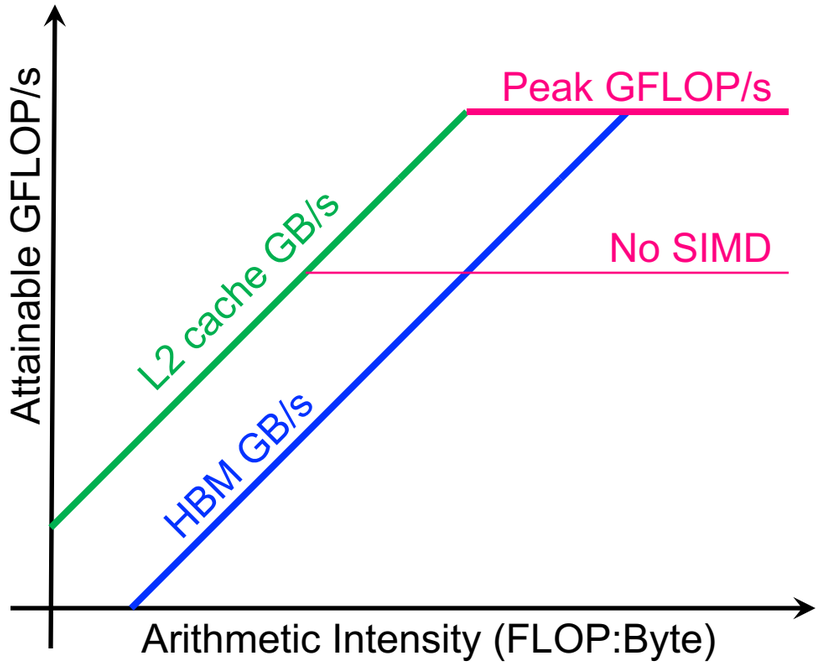
Model is just one piece of the puzzle…

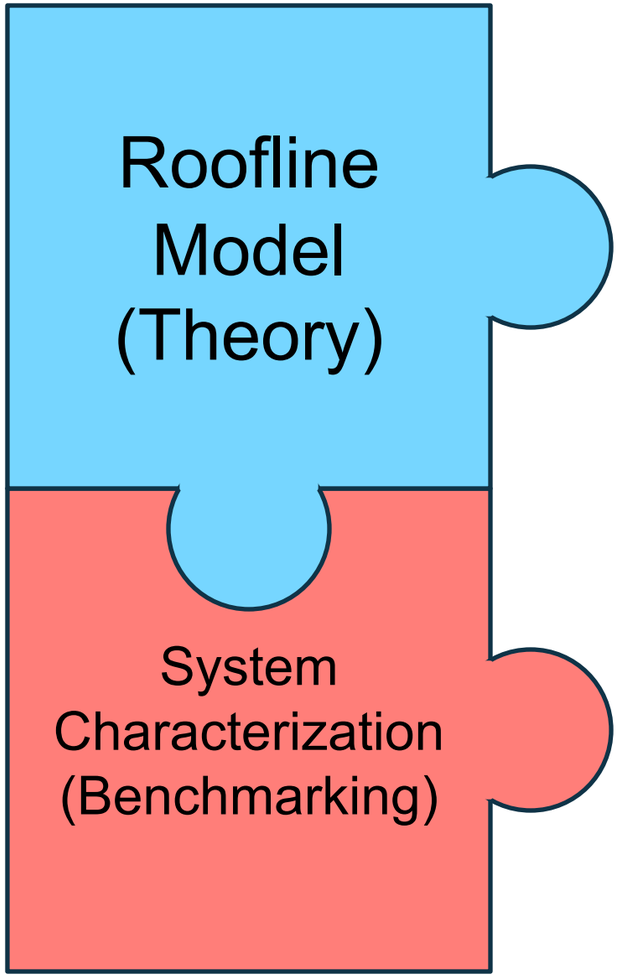
- Roofline Model defines the basic concepts and equations.



Model is just one piece of the puzzle…

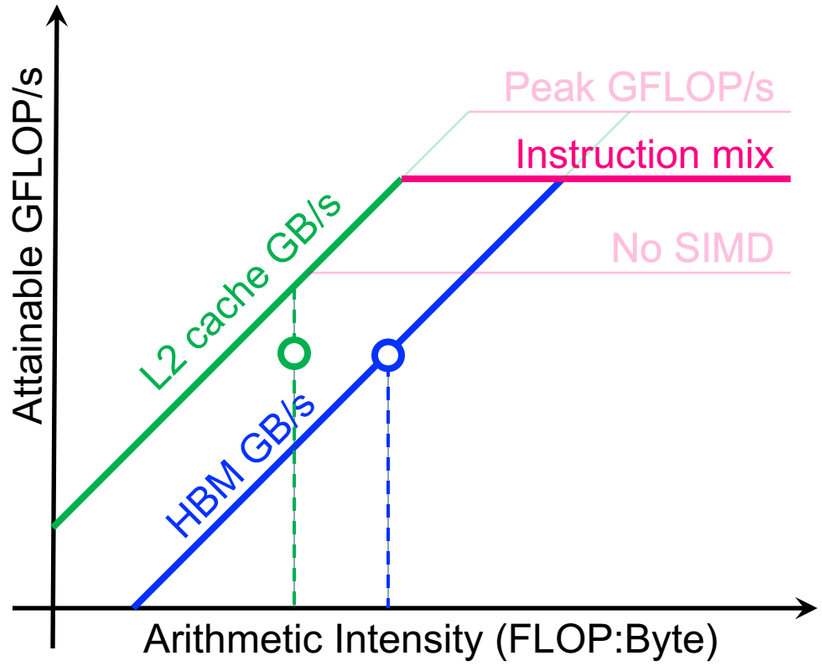
- System Characterization defines the shape of the Roofline (peak bandwidths and FLOP/s)

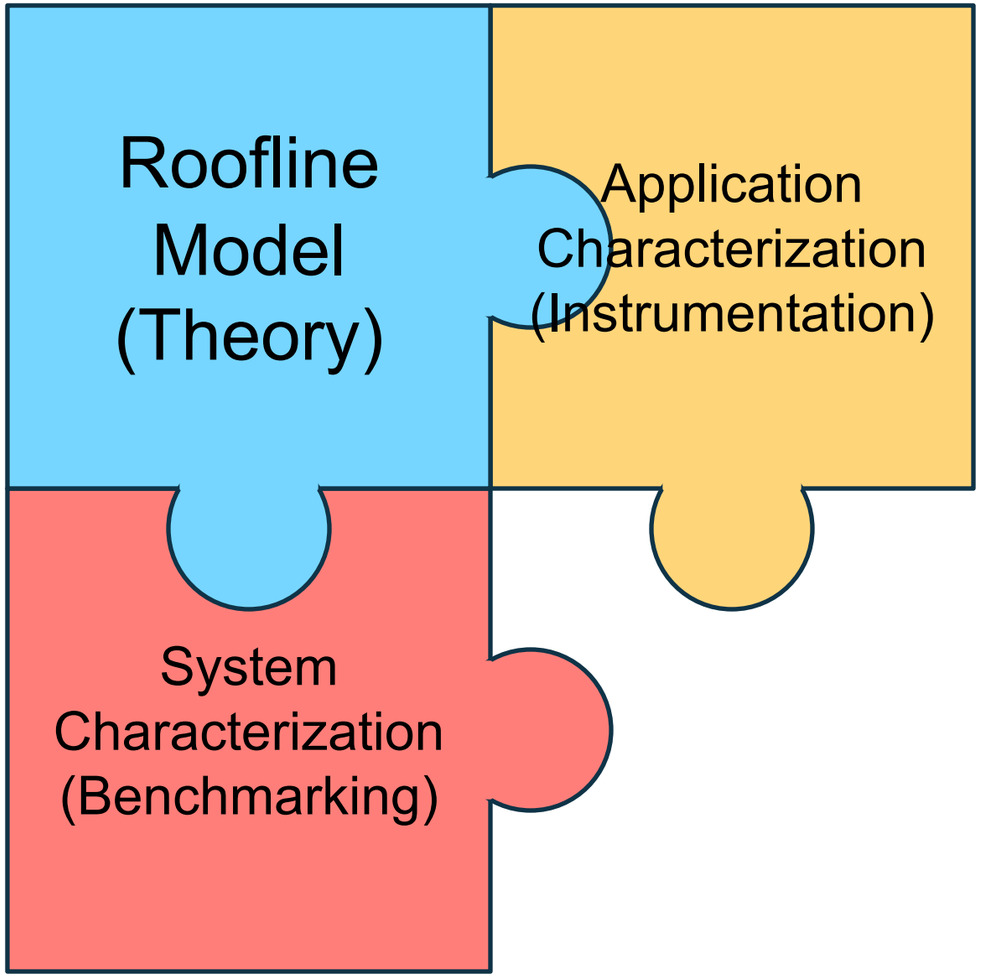




Model is just one piece of the puzzle…

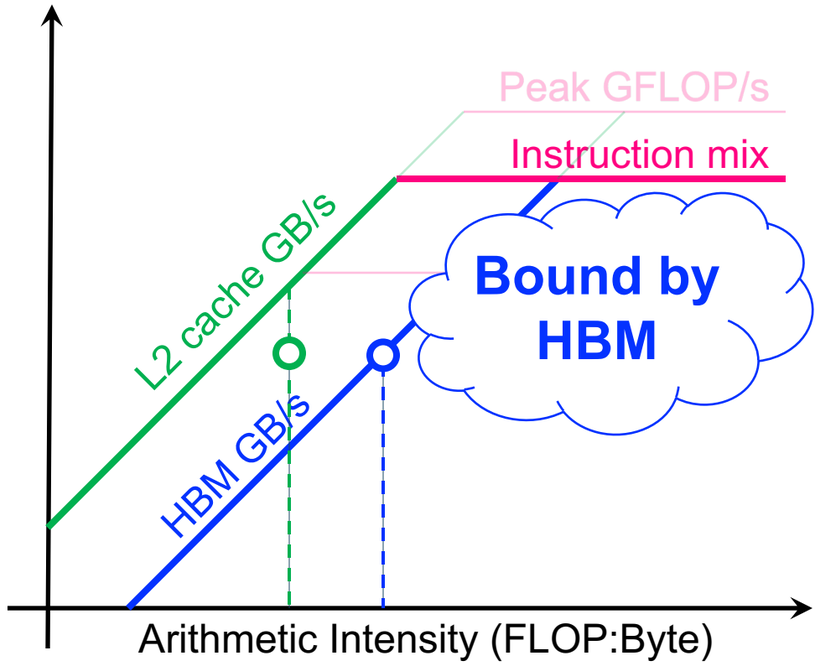
- Application Characterization determines…
  - Intensity and Performance of each loop
  - Position of any implicit ceilings

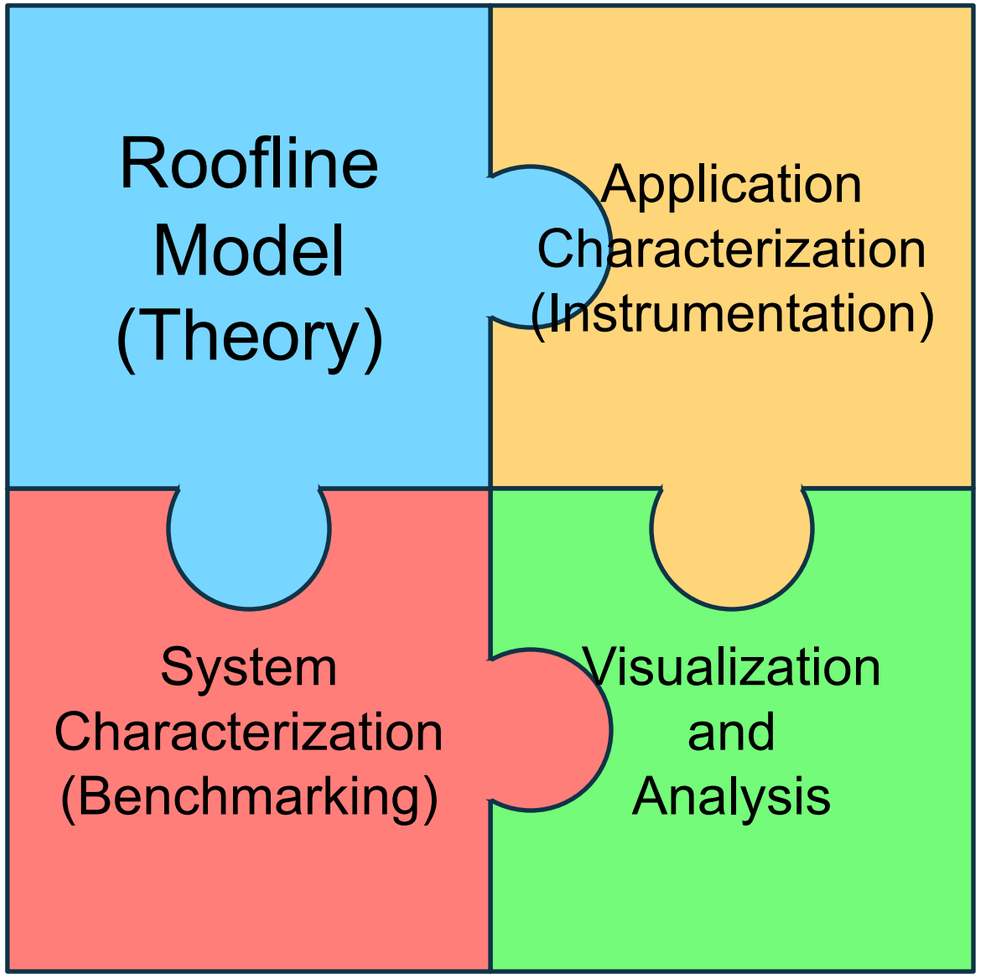




Model is just one piece of the puzzle…

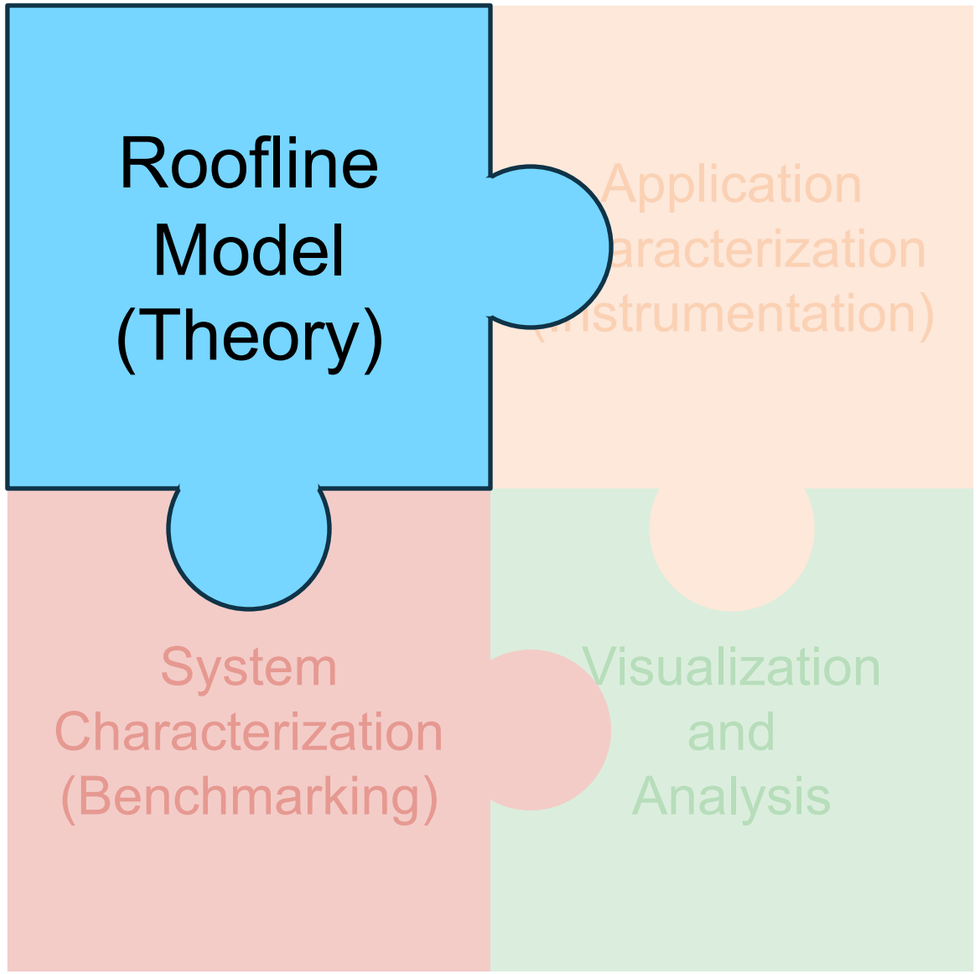
- Visualization tools combine all data together and provide analytical capability





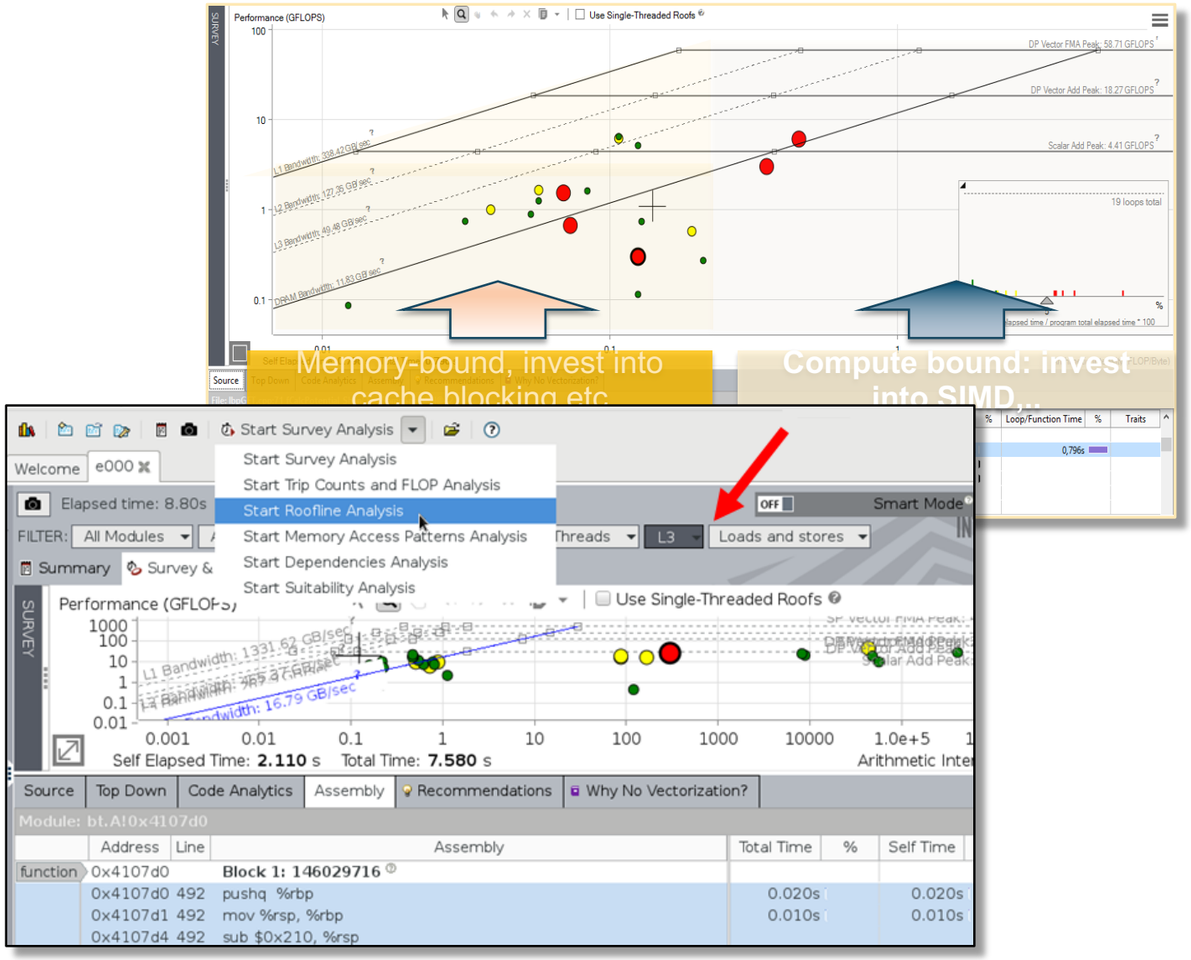
Rest of Tutorial…

- Aleks will introduce CARM and energy Rooflines
  - Extends Roofline formalism to present average memory bandwidth (vs. bandwidth at each level)
  - Provides visualization of power, energy, and energy efficiency.



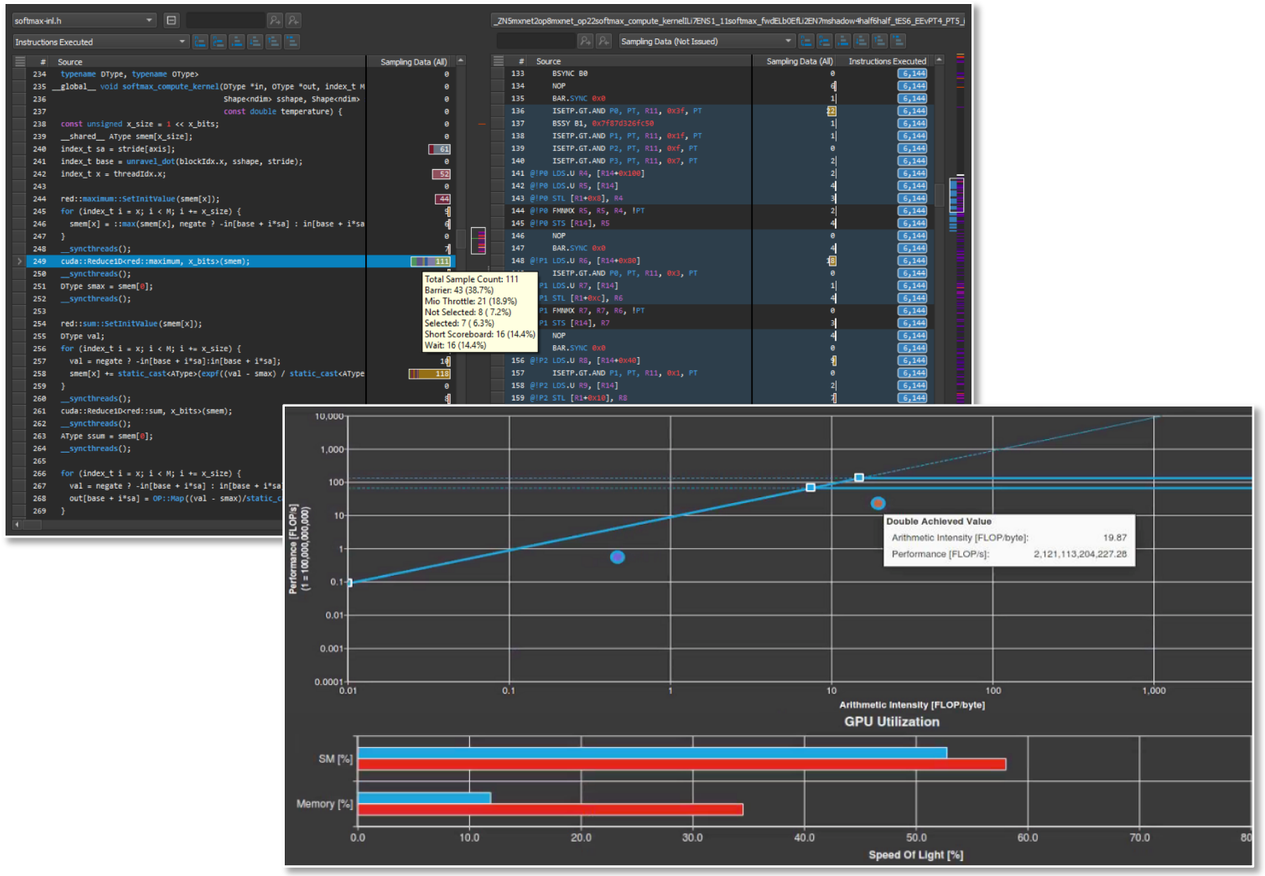
Rest of Tutorial…

- Today, Zakhar will run a hands-on using **Intel® Advisor**
  - ✓ Automatically instruments applications (one dot per loop nest/function)
  - ✓ Computes FLOPS and AI for each function (**CARM**)
  - ✓ **Integrated Cache Simulator (hierarchical roofline / multiple AI's)**
  - ✓ AVX-512 support that incorporates masks
  - ✓ Automatically benchmarks target system (calculates ceilings)
  - ✓ Full integration with existing Advisor capabilities



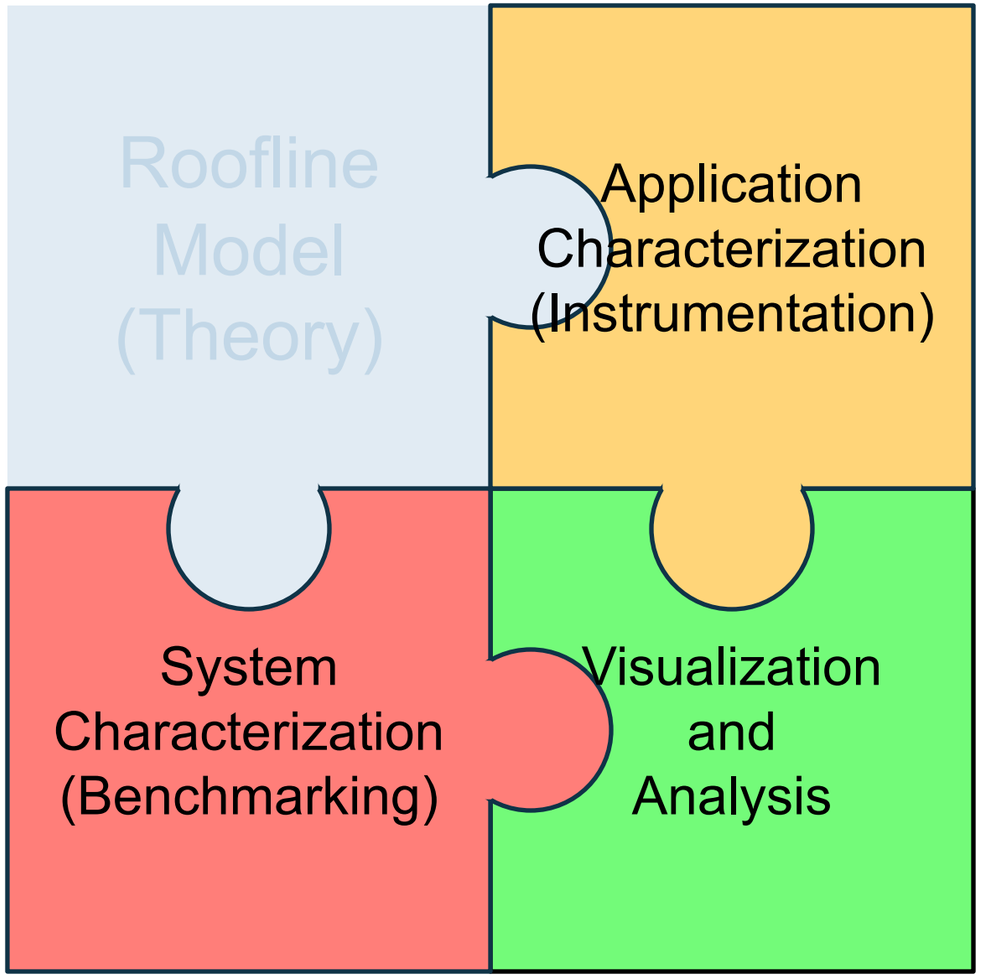
Rest of Tutorial…

- Tomorrow, Max will run a hands-on using **NVIDIA Nsight Compute**
  - ✓ Computes FLOPS and AI for each kernel (**DRAM Roofline**)
  - ✓ **Extensible Roofline Infrastructure (custom hierarchical or DL Rooflines)**
  - ✓ Automatically benchmarks target GPU (calculates ceilings)
  - ✓ Full integration with existing Nsight capabilities



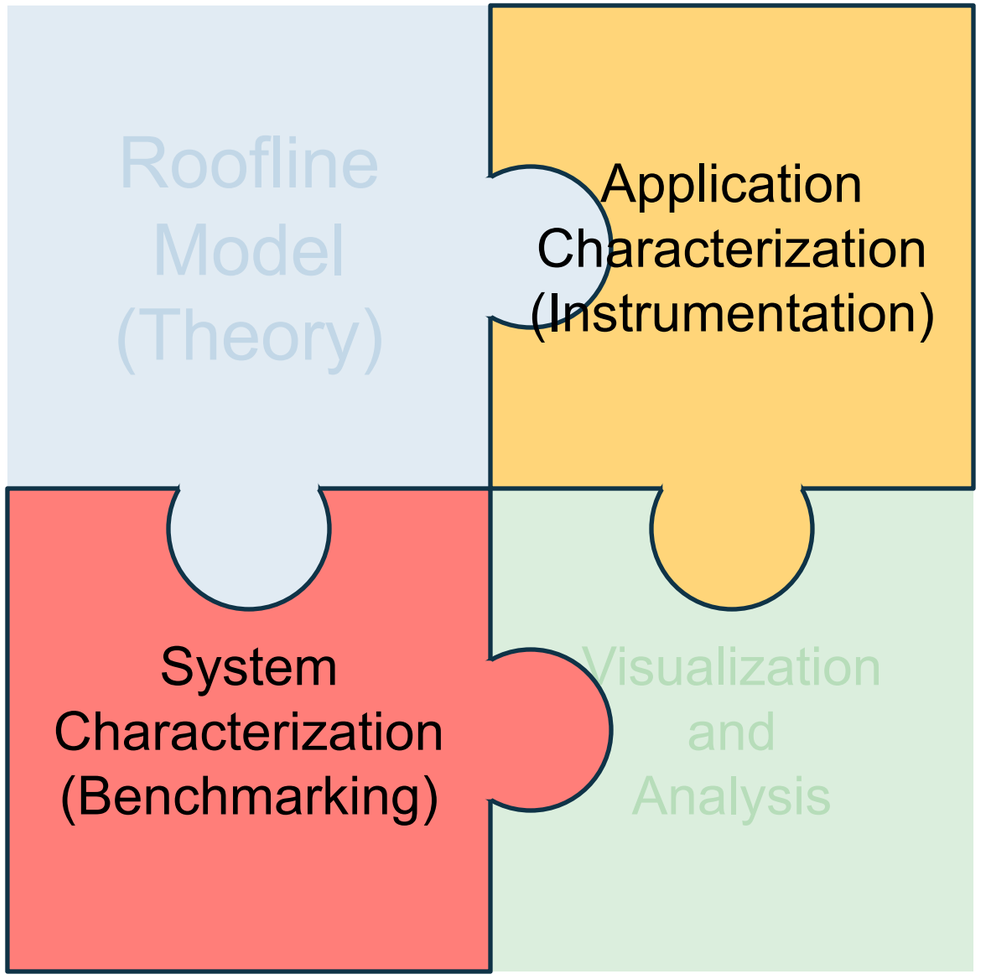
Rest of Tutorial…

- Intel® Advisor and NVIDIA Nsight Compute provide:
  - Integrated benchmarking
  - Application instrumentation and characterization
  - Integrated visualization and analysis
  - Robust, production-quality toolsuite



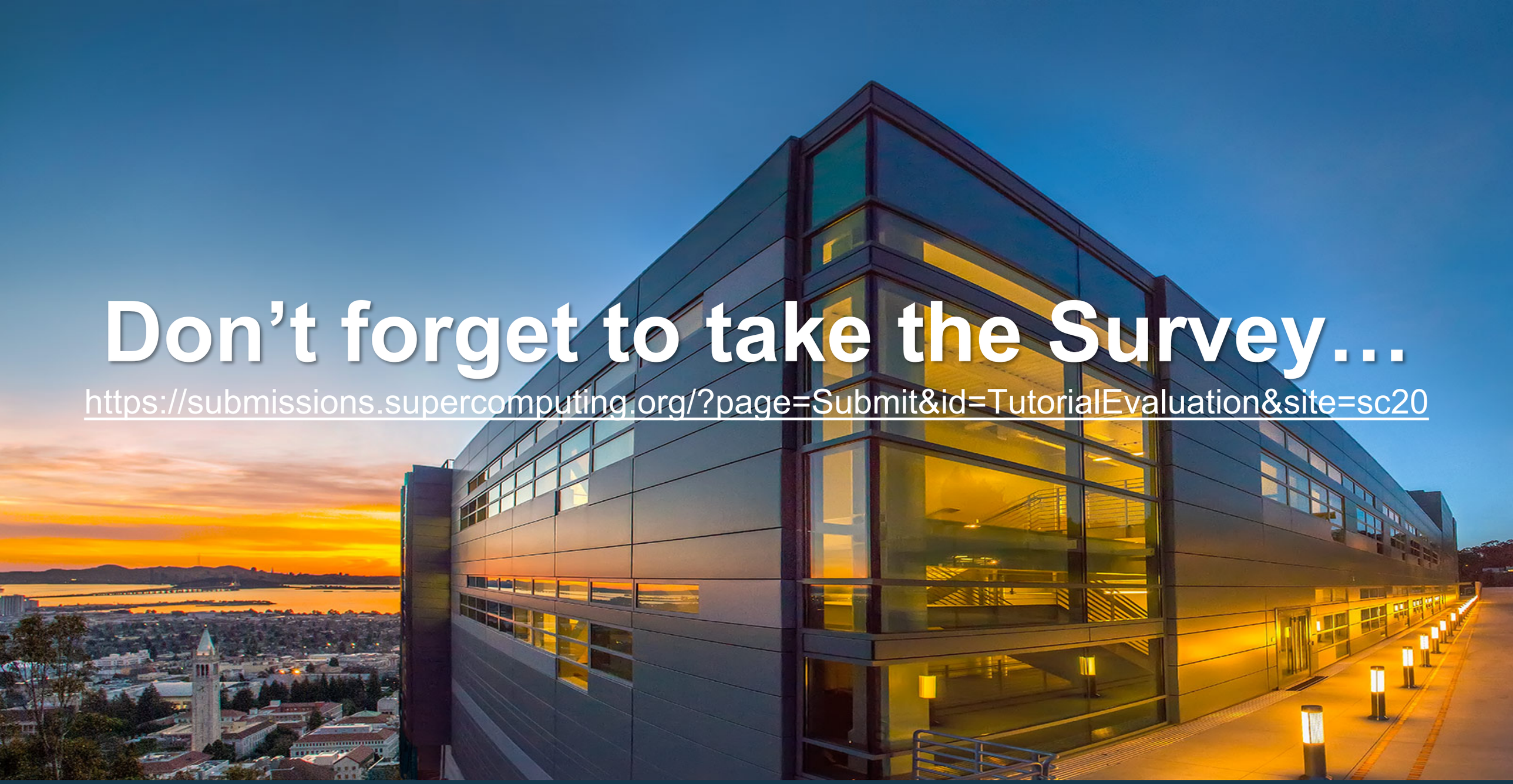
Rest of Tutorial…

- Tomorrow, we will see how Roofline is used to analyze HPC applications
  - Aleksandar Ilic (INESC)
  - JaeHyuk Kwack (DOE/ALCF)
  - Charlene Yang (DOE/NERSC)

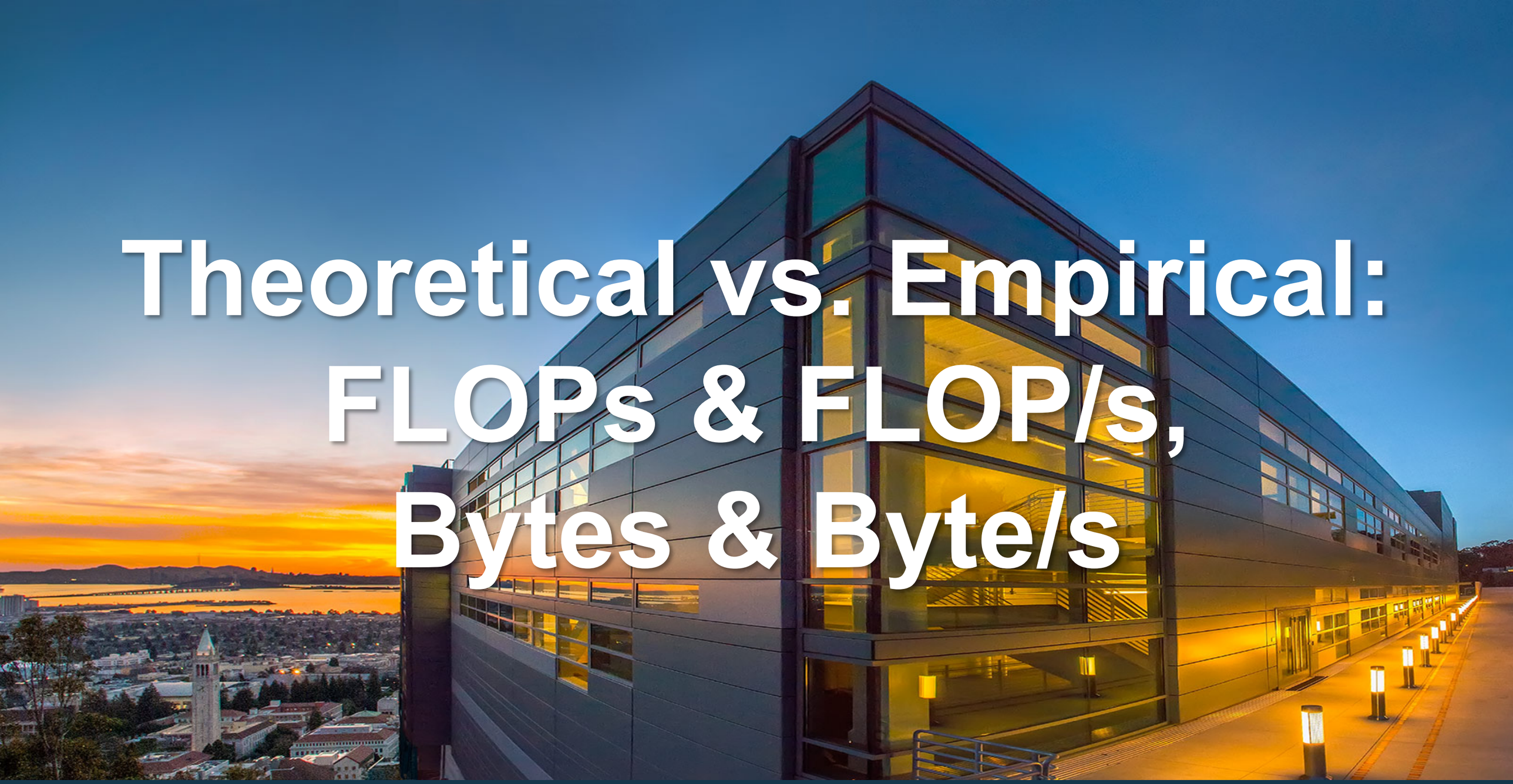


Acknowledgements

- 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.

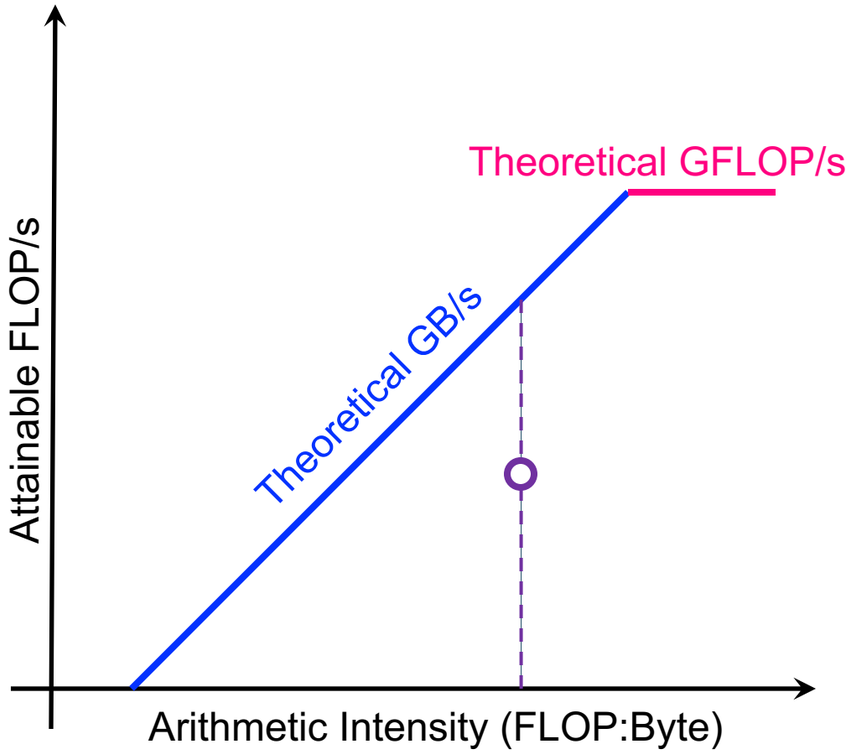

Don’t forget to take the Survey…
https://submissions.supercomputing.org/?page=Submit&id=TutorialEvaluation&site=sc20


BACKUP


Theoretical vs. Empirical:
FLOPs & FLOP/s,
Bytes & Byte/s

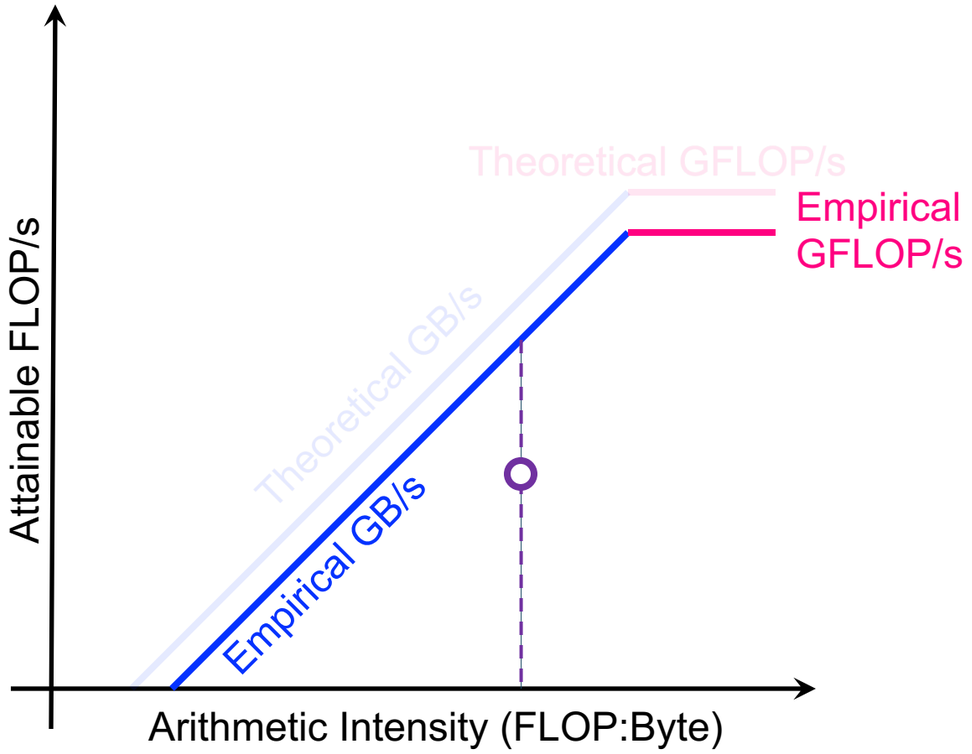
(1) Theoretical vs. Empirical

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



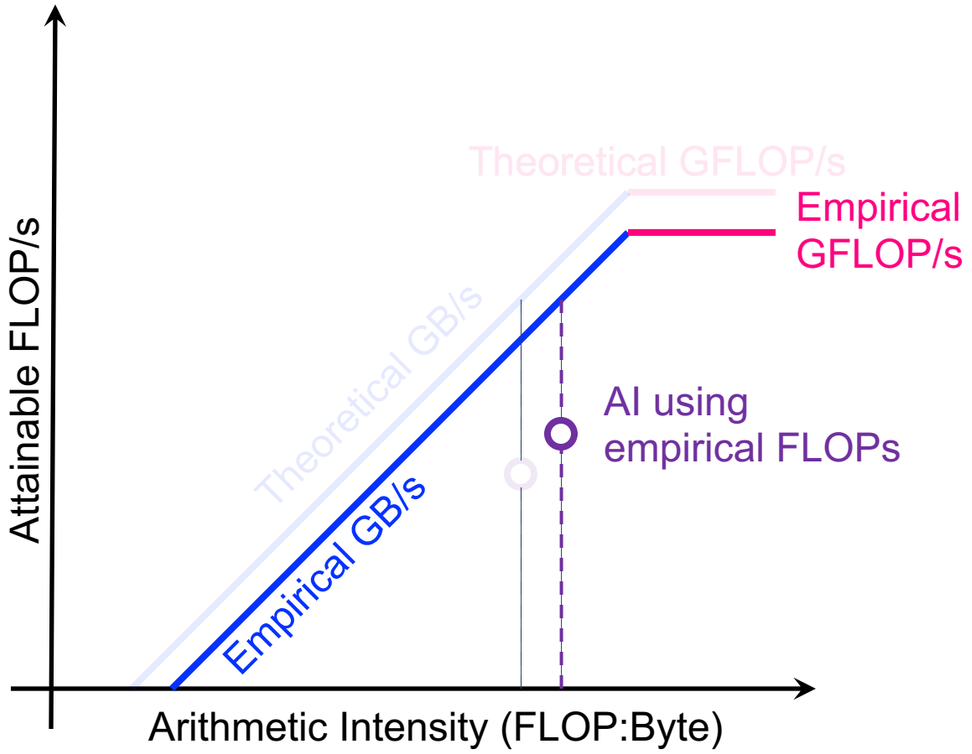
(1) Theoretical vs. Empirical / Benchmarking

- Theoretical Roofline:
  - Pin bandwidth
  - FPUs * GHz
  - 1 C++ FLOP = 1 ISA FLOP
  - Data movement = Compulsory Misses
- Empirical Roofline:
  - Realistic measured bandwidth
  - (e.g. STREAM)
  - Measured Peak FLOP/s



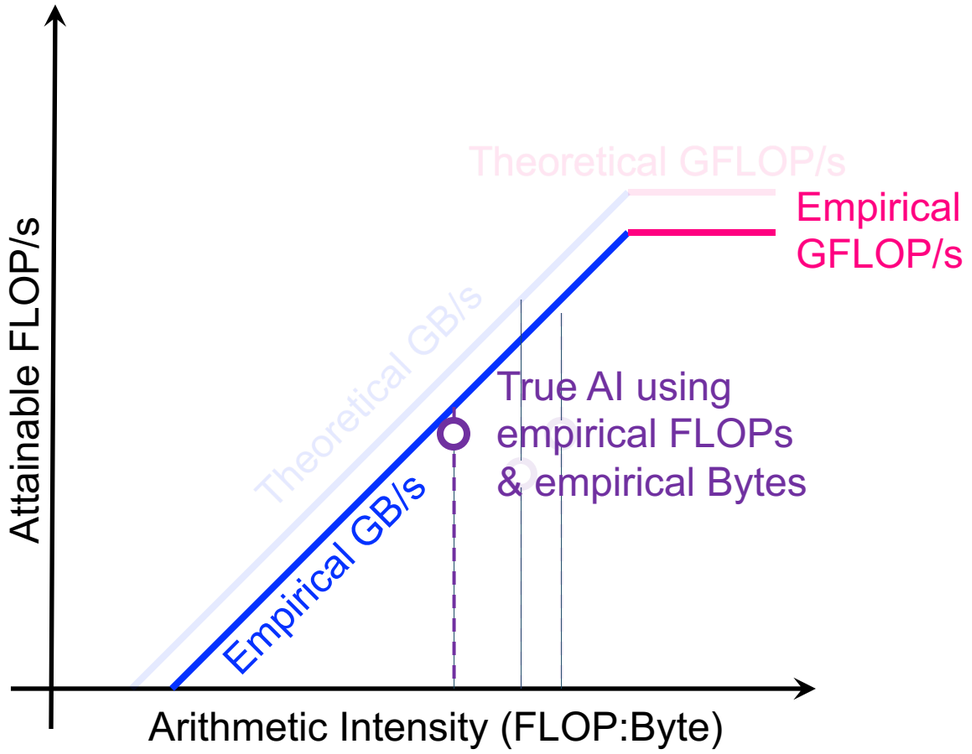
(1) Theoretical vs. Empirical / FLOPs

- Theoretical Roofline:
  - Pin bandwidth
  - FPUs * GHz
  - 1 C++ FLOP = 1 ISA FLOP
  - Data movement = Compulsory Misses
- Empirical Roofline:
  - Realistic measured bandwidth
  - (e.g. STREAM)
  - Measured Peak FLOP/s
  - 1 C++ FLOP >= 1 ISA FLOP (e.g. divide)



(1) Theoretical vs. Empirical / Bytes

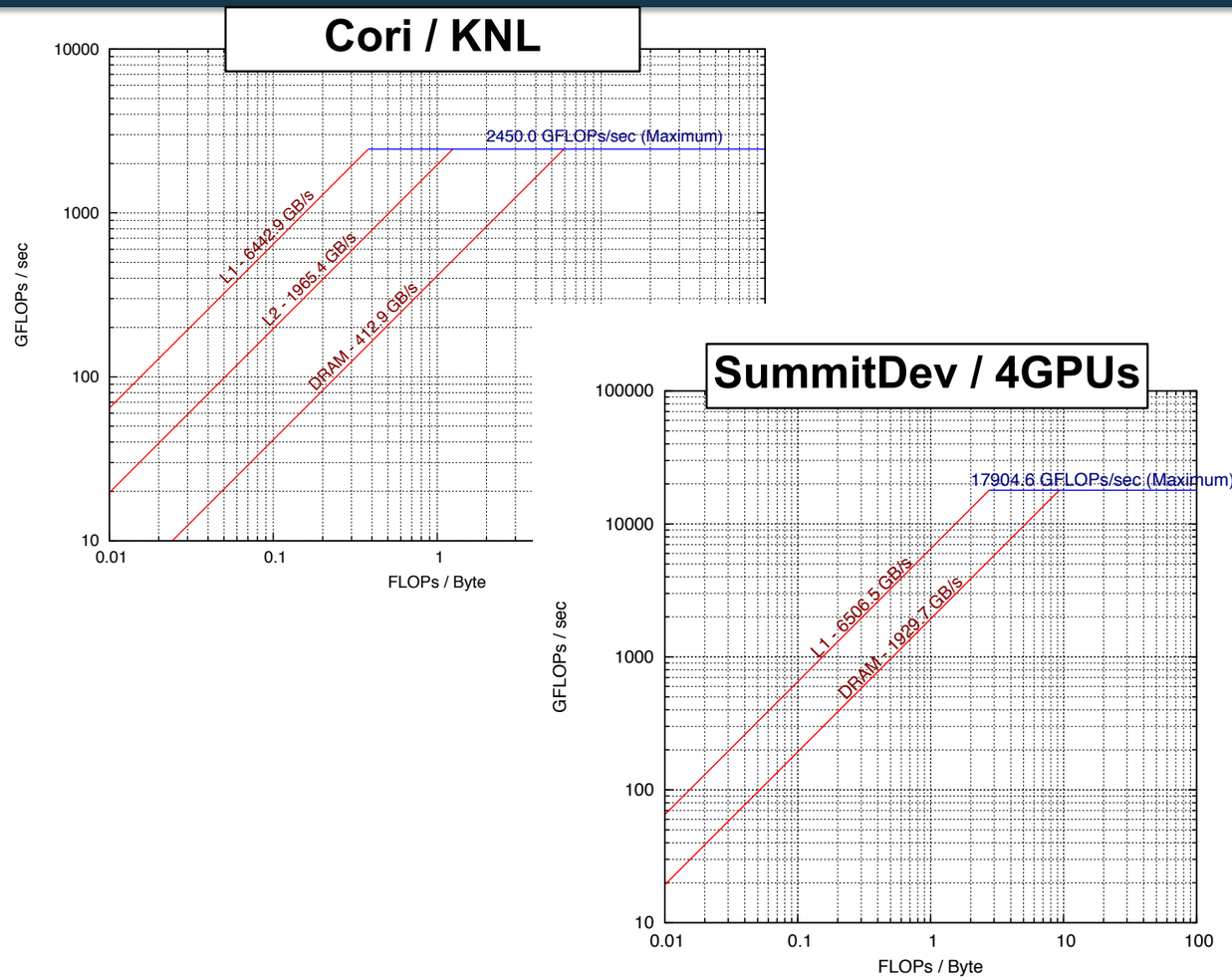
- Theoretical Roofline:
  - Pin bandwidth
  - FPUs * GHz
  - 1 C++ FLOP = 1 ISA FLOP
  - Data movement = Compulsory Misses
- Empirical Roofline:
  - Realistic measured bandwidth
  - (e.g. STREAM)
  - Measured Peak FLOP/s
  - 1 C++ FLOP >= 1 ISA FLOP (e.g. divide)
  - Data movement >> Compulsory Misses
  - **Intensity can be higher or lower**



Machine Characterization

- **"Theoretical Performance"** numbers can be highly optimistic…
  - Pin BW vs. sustained 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://bitbucket.org/berkeleylab/cs-roofline-toolkit
https://github.com/cyanguwa/nersc-roofline
https://crd.lbl.gov/departments/computer-science/PAR/research/roofline