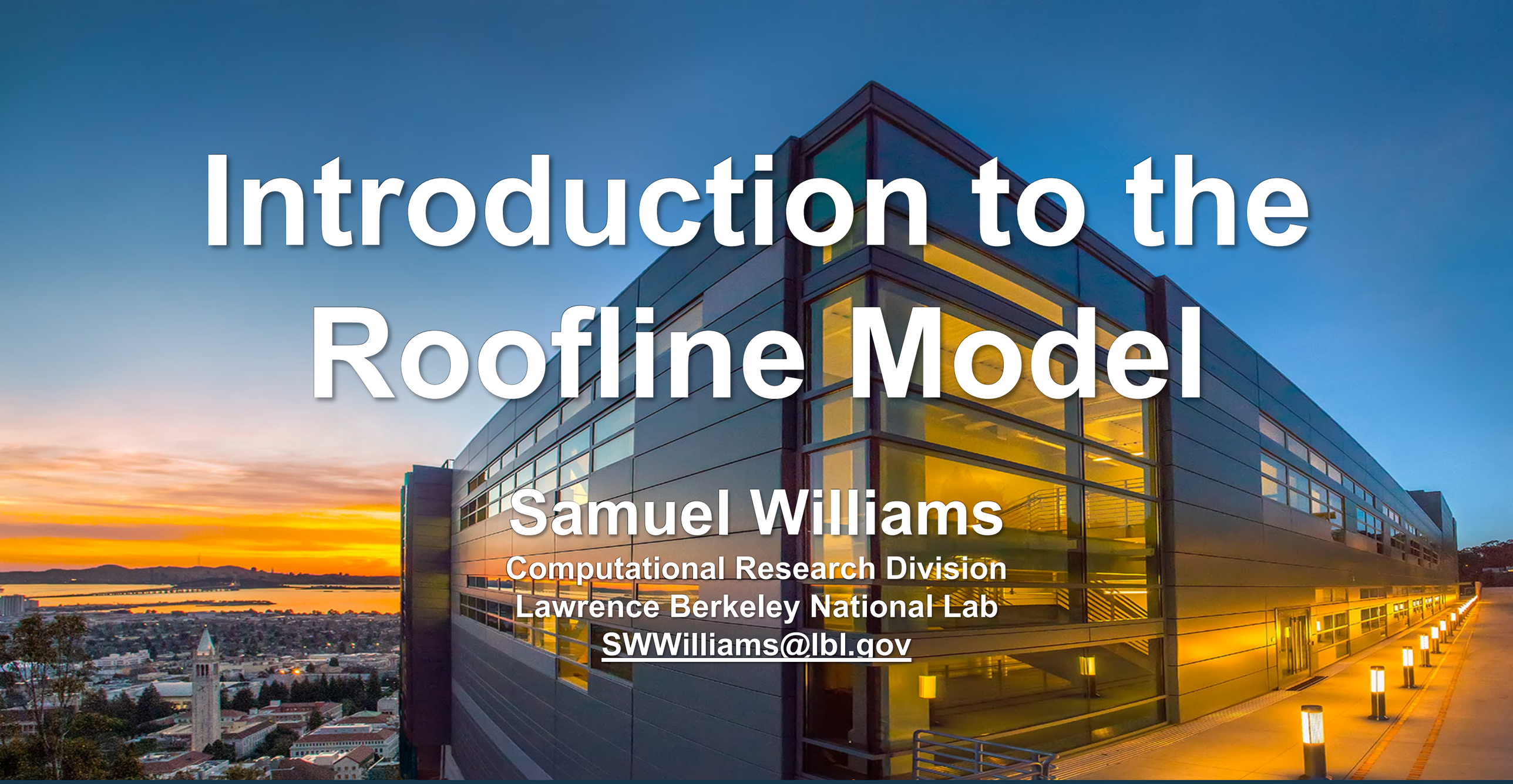


Introduction to the Roofline Model

Samuel Williams

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

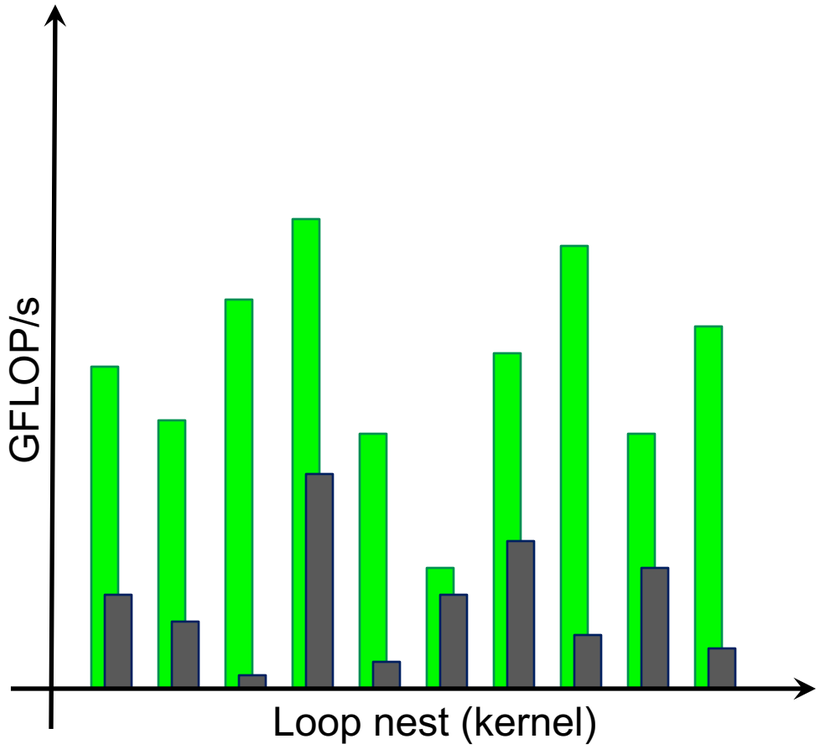
Acknowledgements

- 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.


You just spent 6 months porting your application to GPUs
Are you done?

What is “Good” Performance?

- Imagine profiling the mix of loop nests in an application when running on the GPU
  - GFLOP/s alone may not be particularly insightful
  - speedup relative to a Xeon may seem random



What is “Good” Performance?

- Two fundamental aspects to “Good” performance…

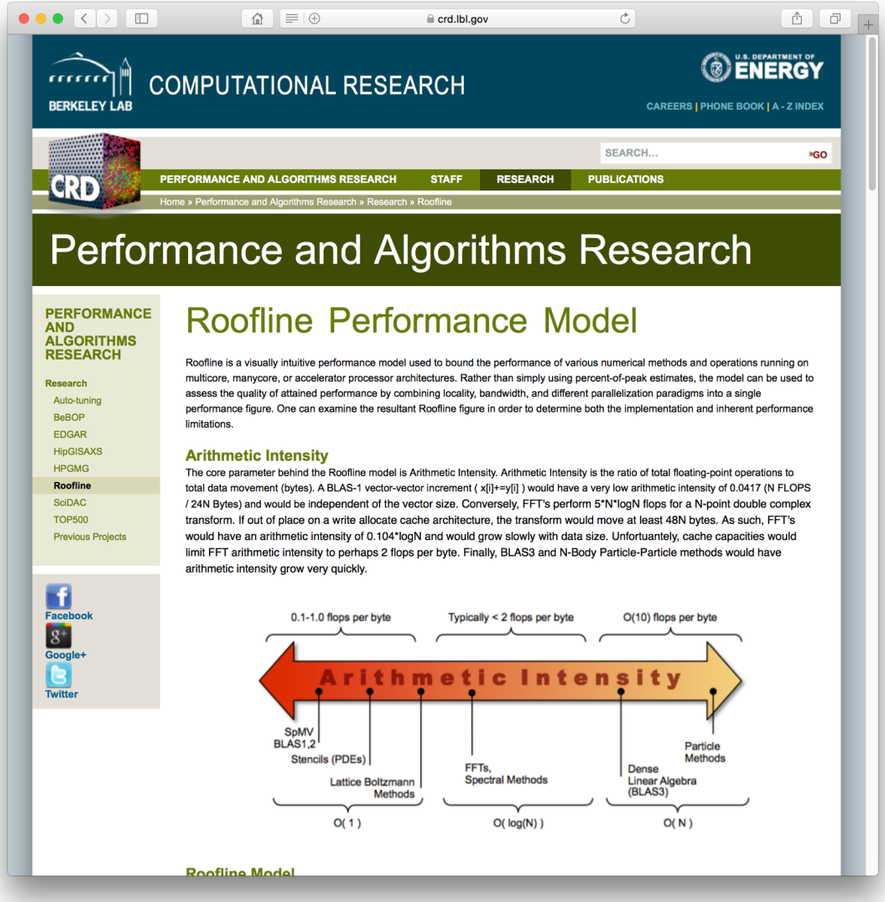
1. Operating in the throughput-limited regime
   not sensitive to Amdahl effects, D2H/H2D transfers, launch overheads, etc…

2. making good use of the GPU’s **compute** and/or **bandwidth** capabilities

➢ **Ultimately, we need a quantitative model rather than qualitative statements like “good”**

Roofline Model

- **Roofline Model** is a throughput-oriented performance model
- Tracks rates not times
- Independent of ISA and architecture
- applies to CPUs, GPUs, Google TPUs[1], FPGAs, etc…
- Helps quantify **Good Performance**

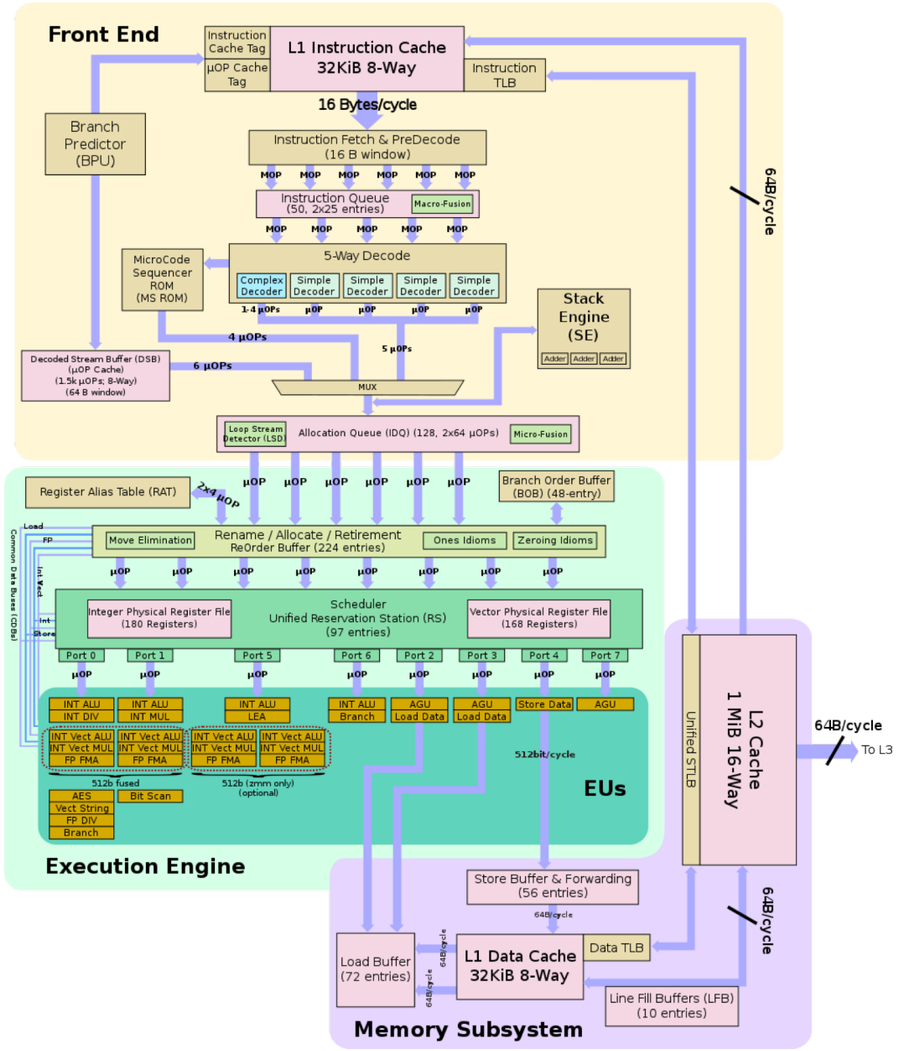


https://crd.lbl.gov/departments/computer-science/PAR/research/roofline

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

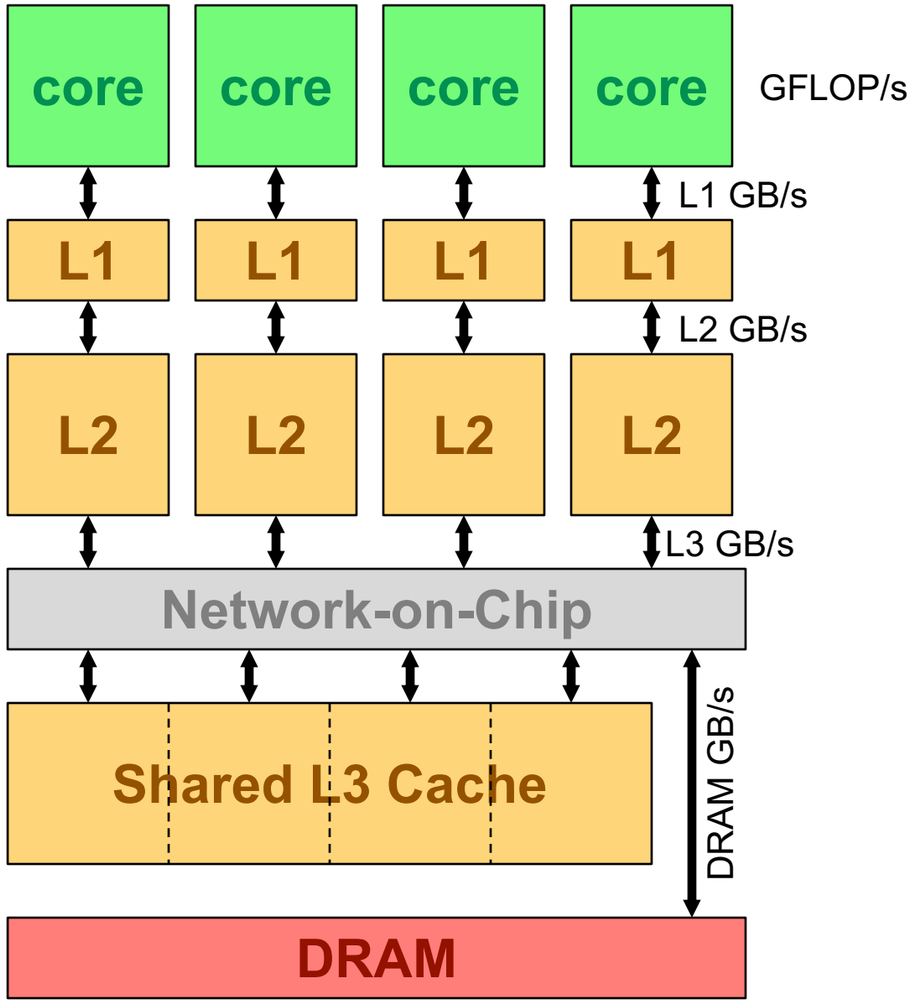
Simulation → Modeling

- Superscalar architectures can be complex
- Don’t model / simulate full architecture
- Created simplified processor architecture



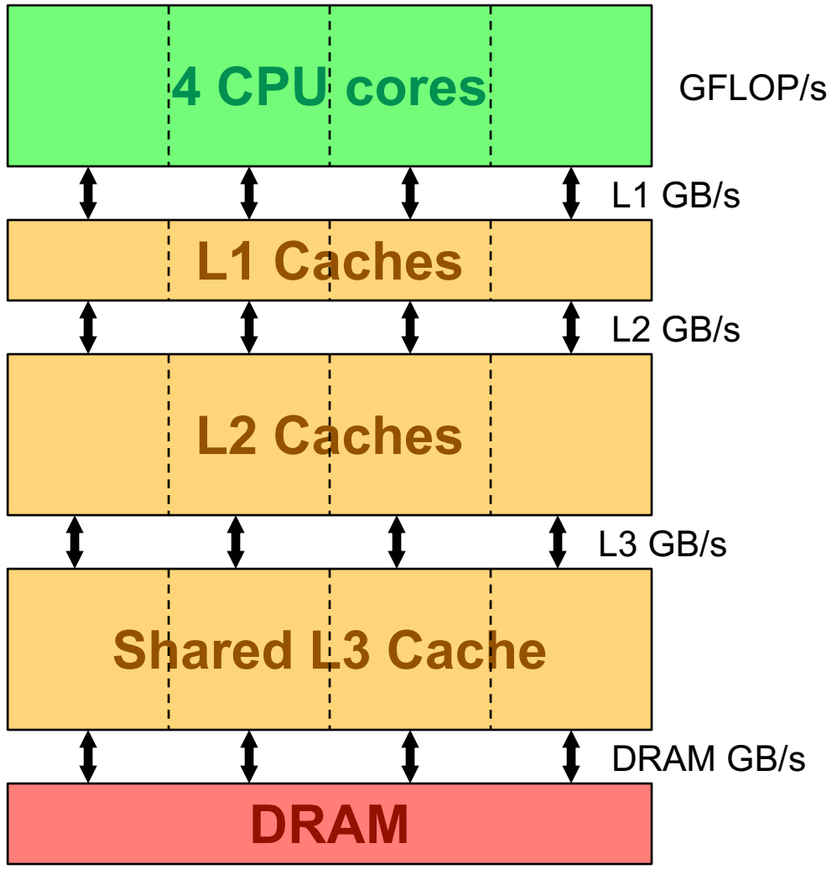
Reduced Model

- Superscalar architectures can be complex
- Don’t model / simulate full architecture
- Created simplified processor architecture
  - Cores can attain peak GFLOP/s on local data



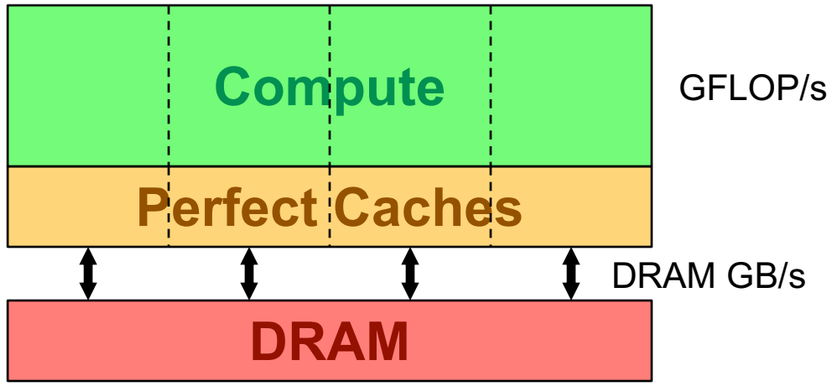
Reduced Model

- Superscalar architectures can be complex
- Don’t model / simulate full architecture
- Created simplified processor architecture
  - Cores can attain peak GFLOP/s on local data
  - Cores execute load-balanced SPMD code



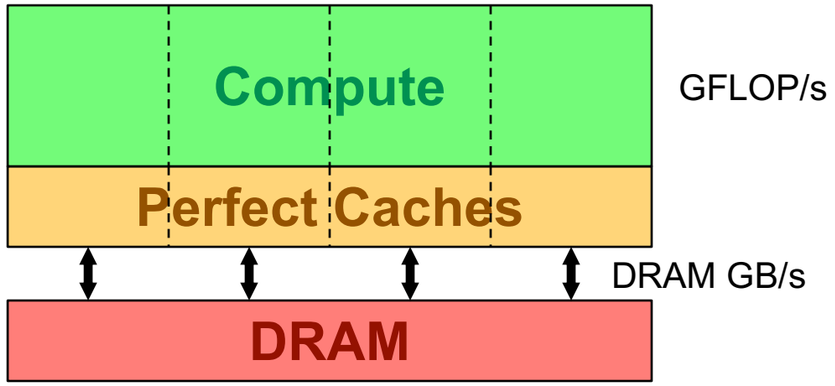
Reduced Model

- Superscalar architectures can be complex
- Don’t model / simulate full architecture
- Created simplified processor architecture
  - Cores can attain peak GFLOP/s on local data
  - Cores execute load-balanced SPMD code
  - There is sufficient cache bandwidth and capacity such that they do not affect performance



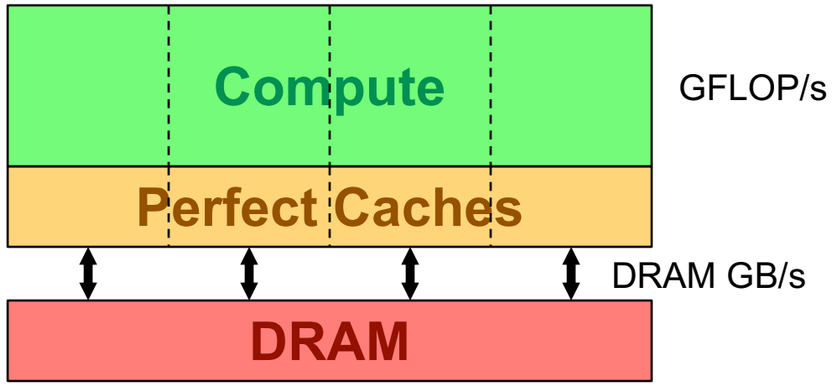
Reduced Model

- Superscalar architectures can be complex
- Don’t model / simulate full architecture
- Created simplified processor architecture
  - Cores can attain peak GFLOP/s on local data
  - Cores execute load-balanced SPMD code
  - There is sufficient cache bandwidth and capacity such that they do not affect performance
  - ➢ **Basis for DRAM Roofline Model**



Data Movement or Compute?

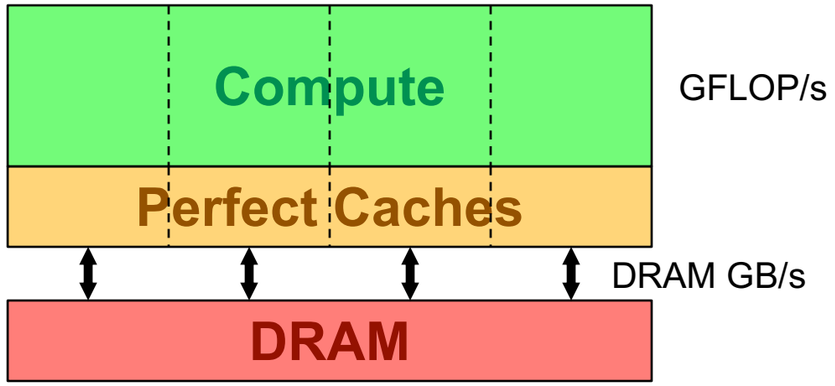
- Which takes longer?
  - Data Movement
  - Compute?



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

Data Movement or Compute?

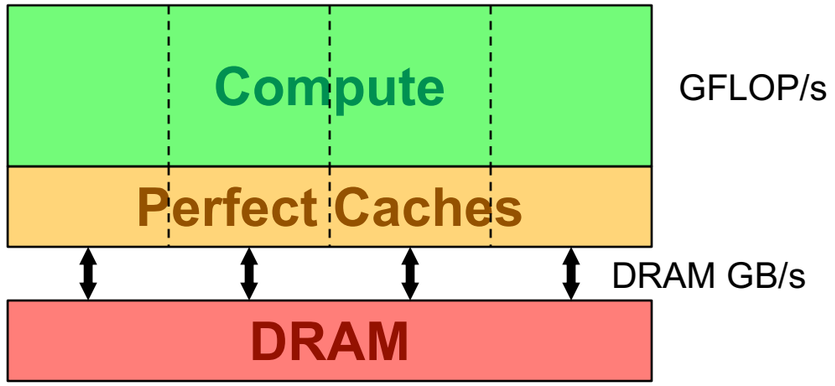
- Which takes longer?
  - Data Movement
  - Compute?
- Is performance limited by compute or data movement?



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

Data Movement or Compute?

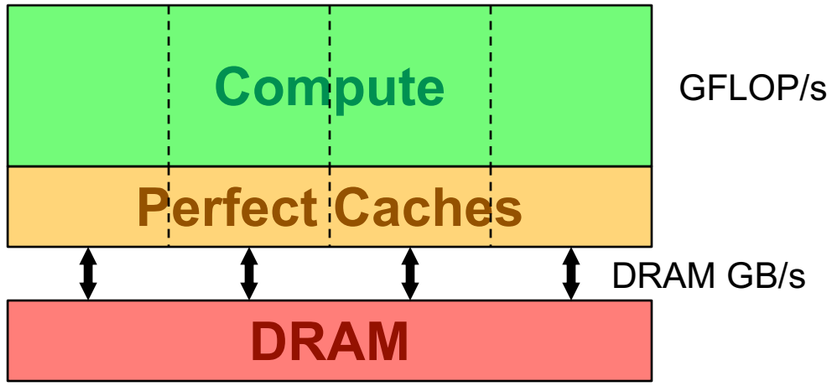
- Which takes longer?
  - Data Movement
  - Compute?
- Is performance limited by compute or data movement?



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

Data Movement or Compute?

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



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

AI (Arithmetic Intensity) = FLOPs / Bytes (as presented to DRAM)

Arithmetic Intensity

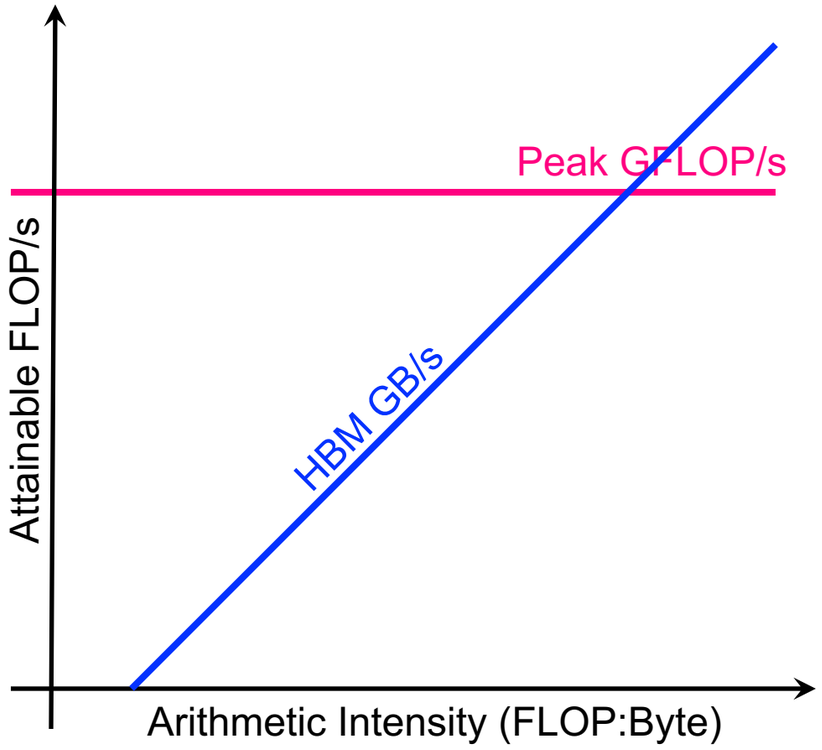
- Measure of data locality (data reuse)
- Ratio of **Total Flops** performed to **Total Bytes** moved
- For the DRAM Roofline…
  - Total Bytes to/from DRAM
  - Includes all cache and prefetcher effects
  - Can be very different from total loads/stores (bytes requested)
  - Equal to ratio of sustained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

(DRAM) Roofline Model

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

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

- Plot Roofline bound using Arithmetic Intensity as the x-axis
- **Log-log scale** makes it easy to doodle, extrapolate performance along Moore's Law, etc…

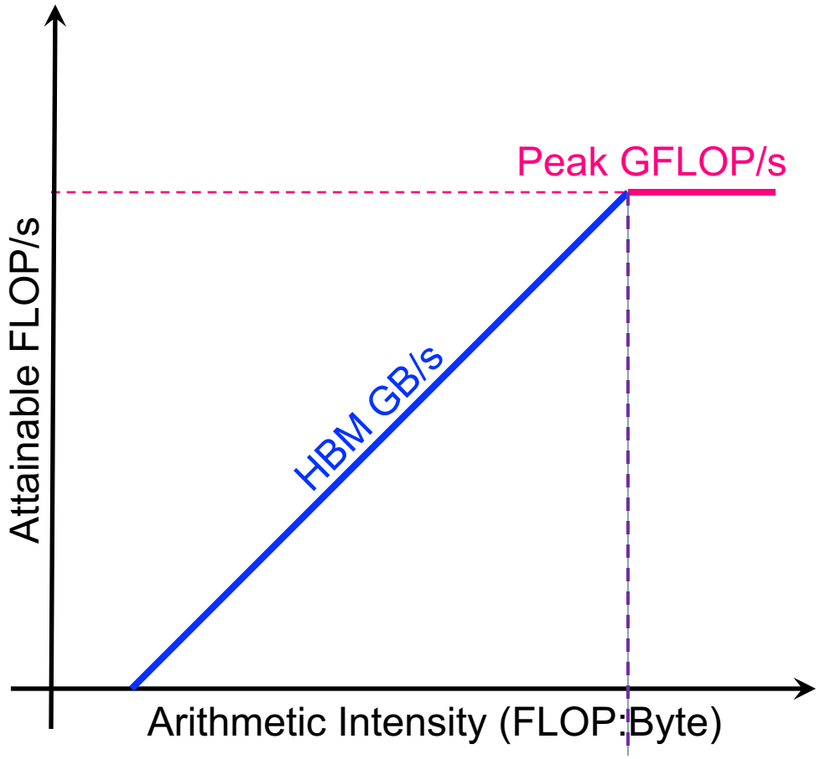


(DRAM) Roofline Model

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

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

- Plot Roofline bound using Arithmetic Intensity as the x-axis
- **Log-log scale** makes it easy to doodle, extrapolate performance along Moore's Law, etc…



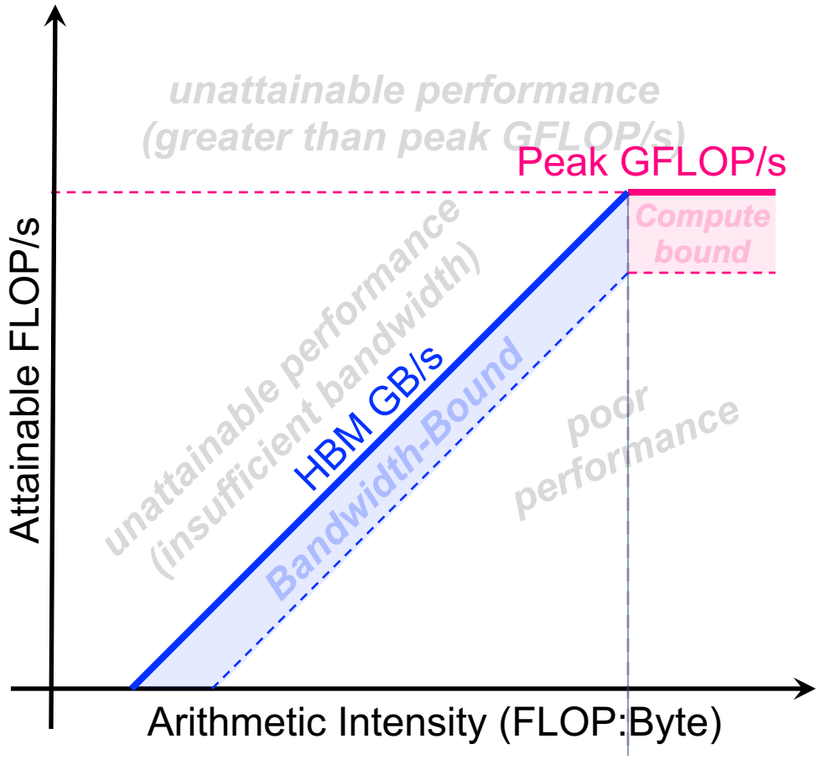
Transition @ AI == Peak GFLOP/s / Peak GB/s == 'Machine Balance'

(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)

- Roofline tessellates this 2D view of performance into 5 regions…



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

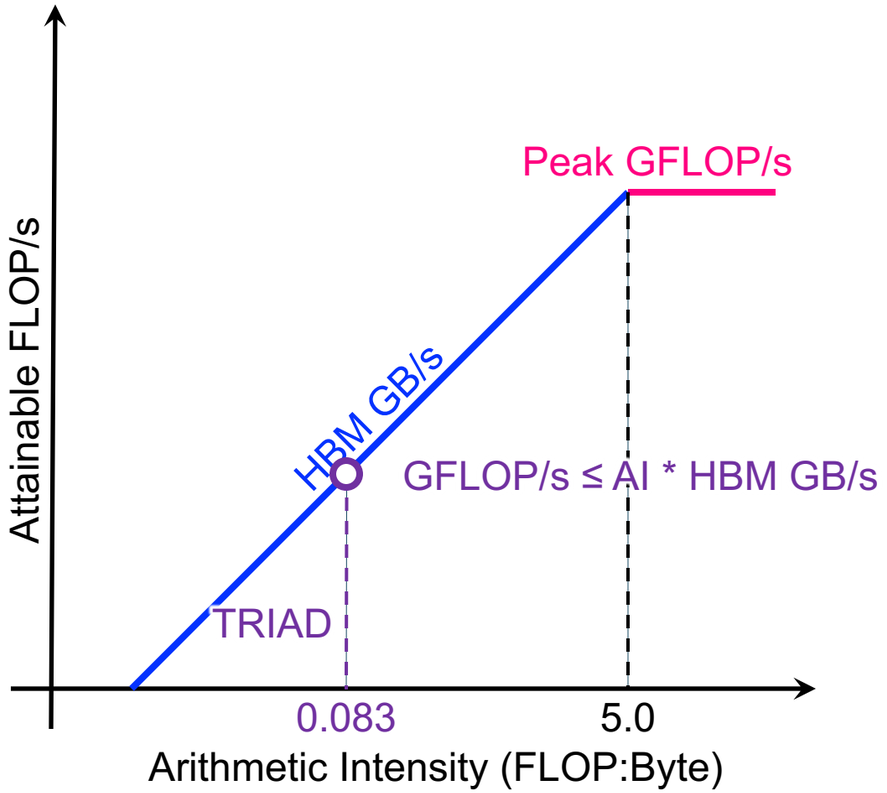
Roofline Example #1

- 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…

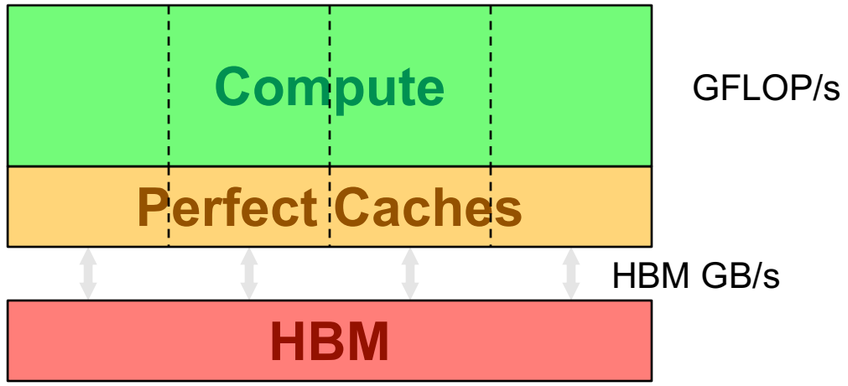
```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

  - 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…

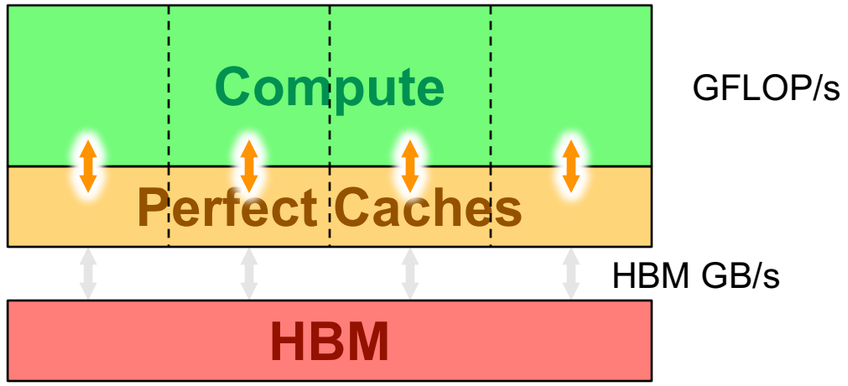



```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                 + old[k  ][j  ][i-1]
                 + old[k  ][j  ][i+1]
                 + old[k  ][j-1][i  ]
                 + old[k  ][j+1][i  ]
            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

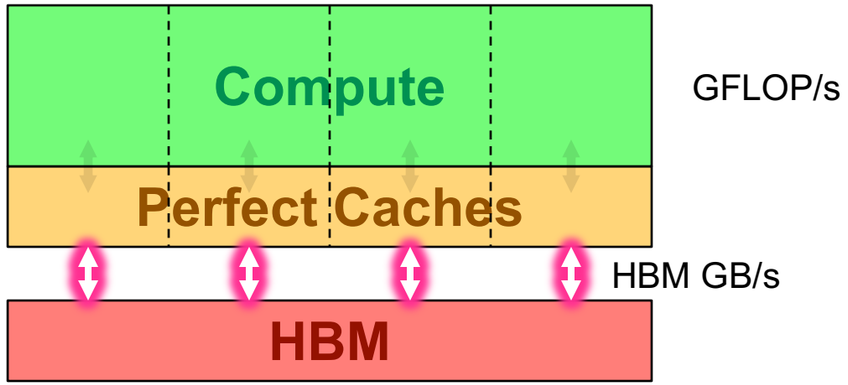
    (measured at the L1)




```
#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

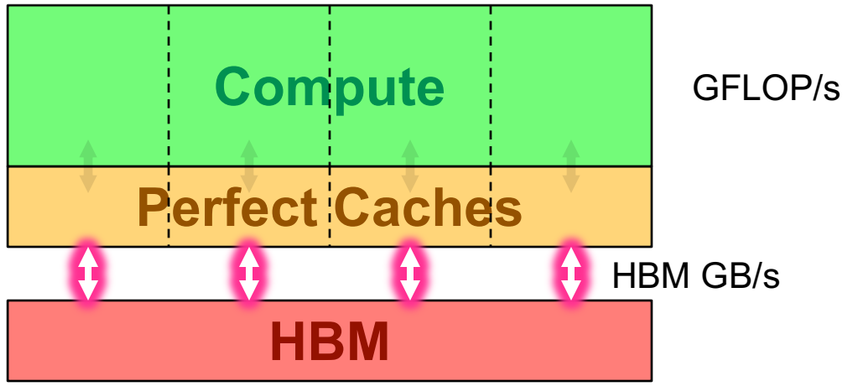
- Conversely, 7-point constant coefficient stencil…
  - 7 FLOPs
  - 8 memory references (7 reads, 1 store) per point
  - **Ideally, cache will filter all but 1 read and 1 write per point**




```
#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)**



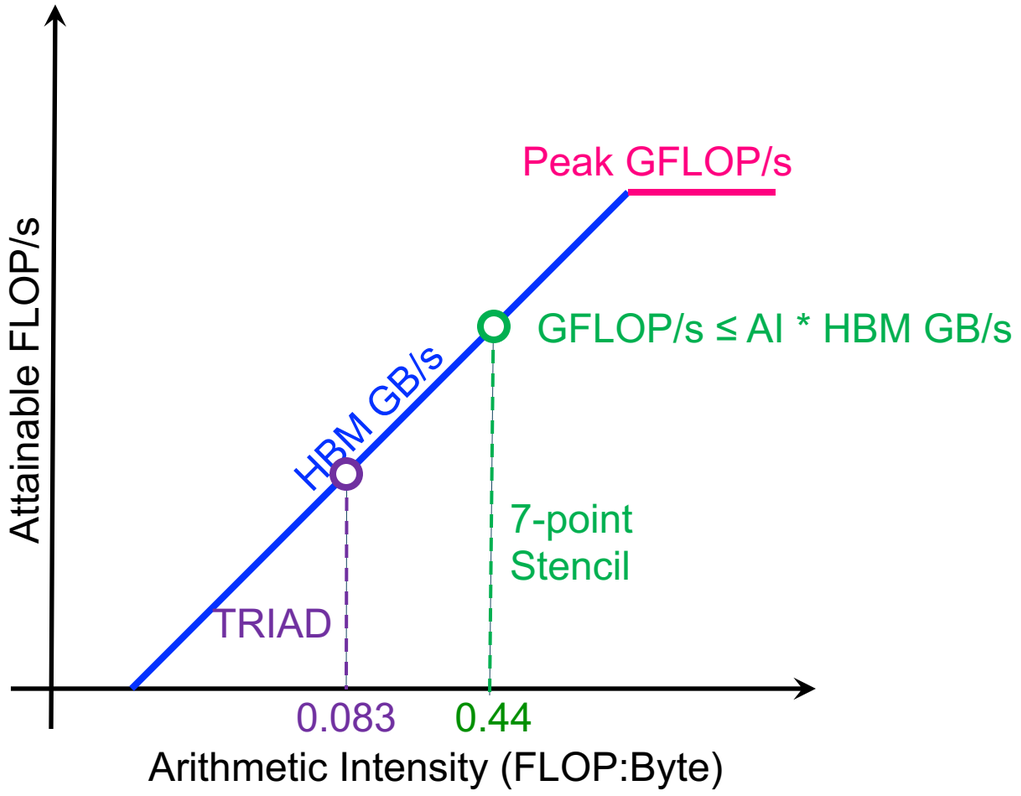

```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                 + old[k  ][j  ][i-1]
                 + old[k  ][j  ][i+1]
                 + old[k  ][j-1][i  ]
                 + old[k  ][j+1][i  ]
            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

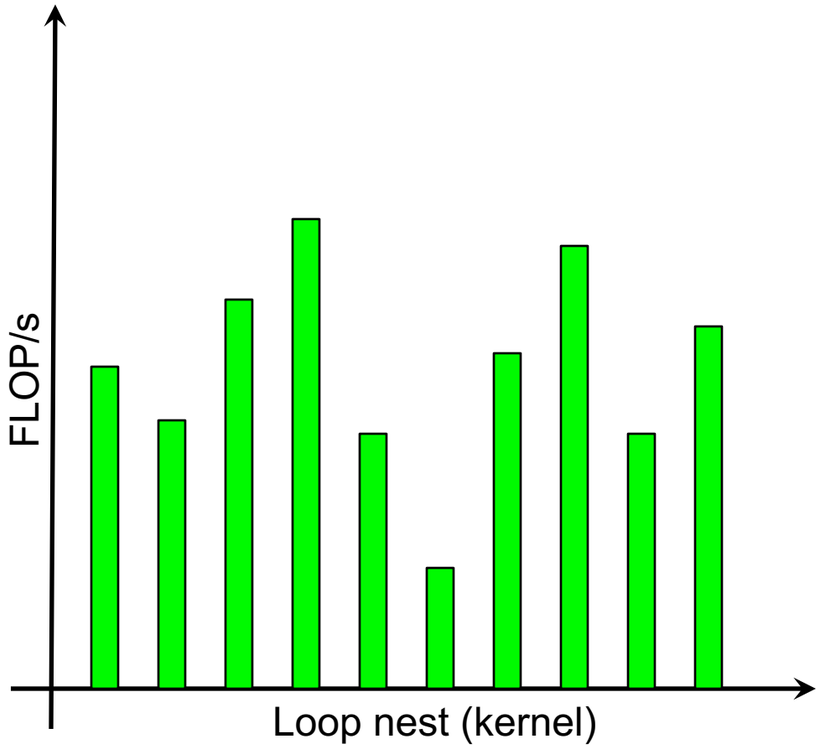
    == memory bound, but 5x the FLOP rate as TRIAD

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



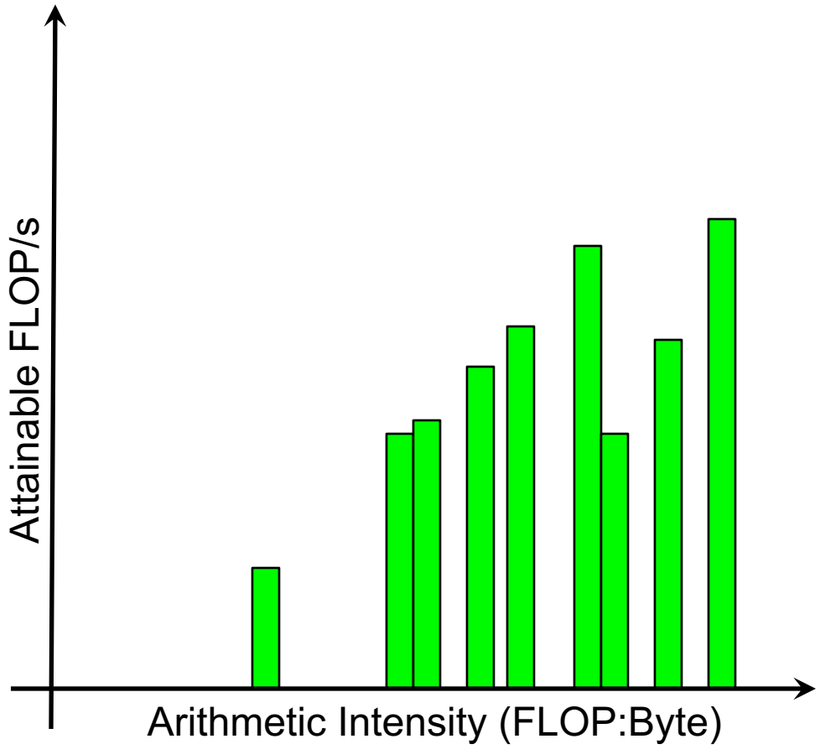
What is “Good” Performance?

- Think back to our mix of loop nests…



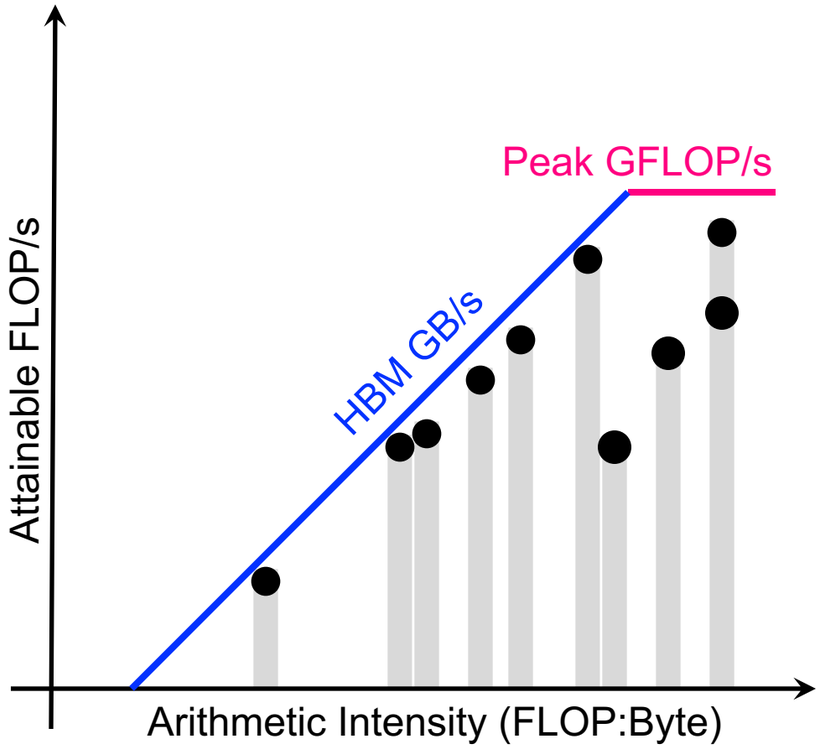
What is “Good” Performance?

- We can sort kernels by arithmetic intensity…



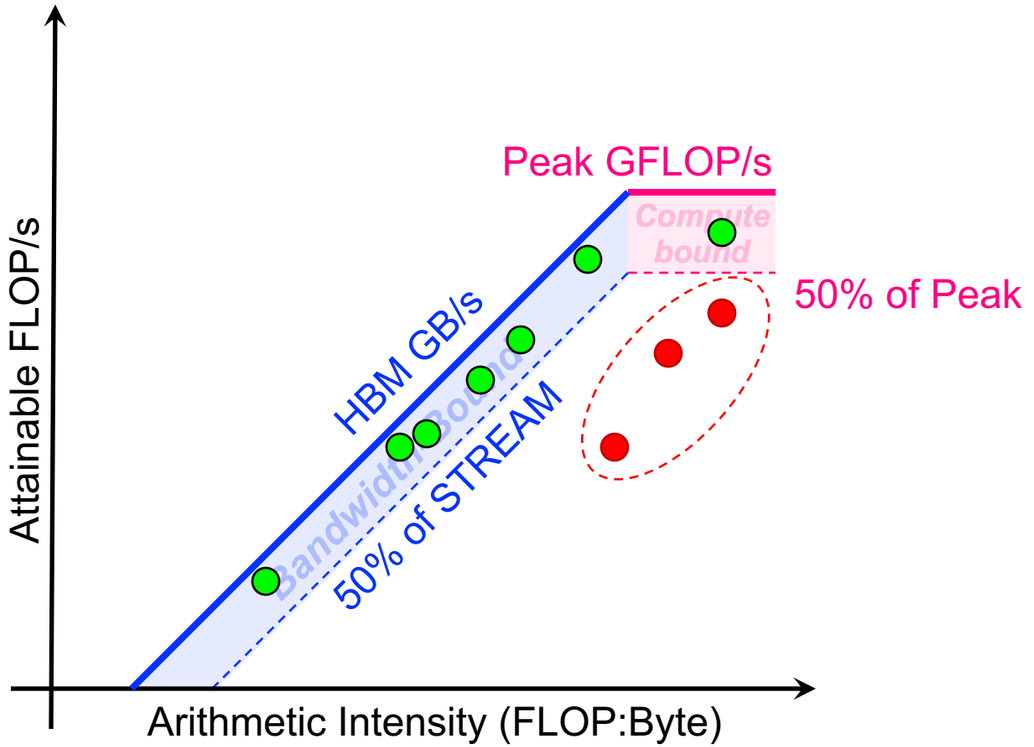
What is “Good” Performance?

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



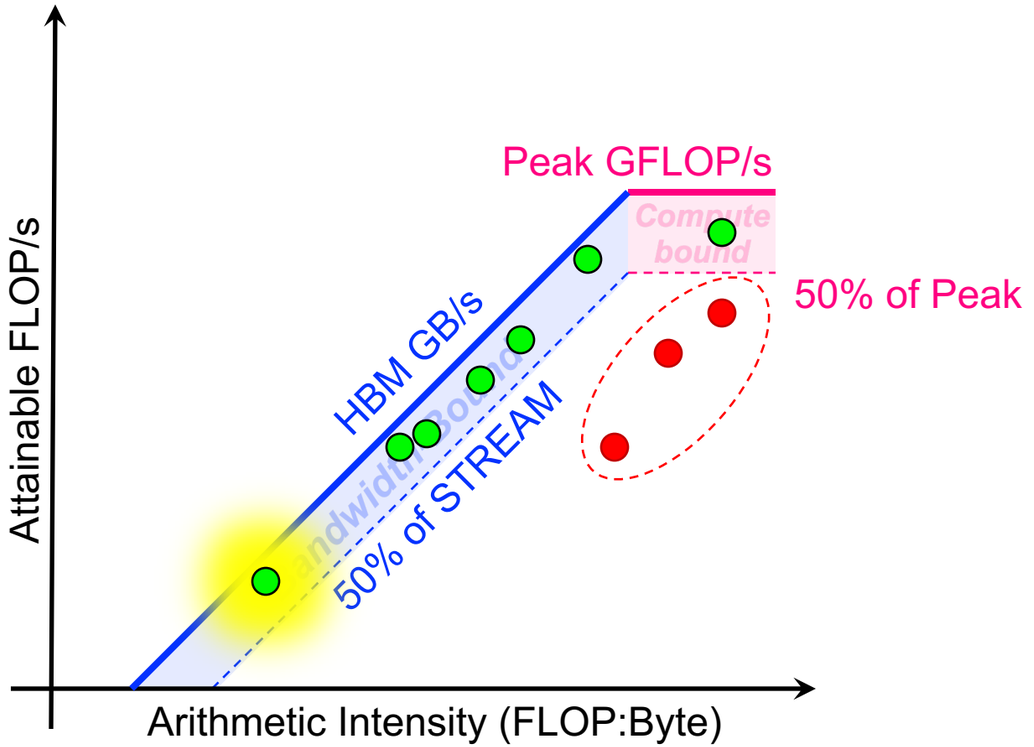
What is “Good” Performance?

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



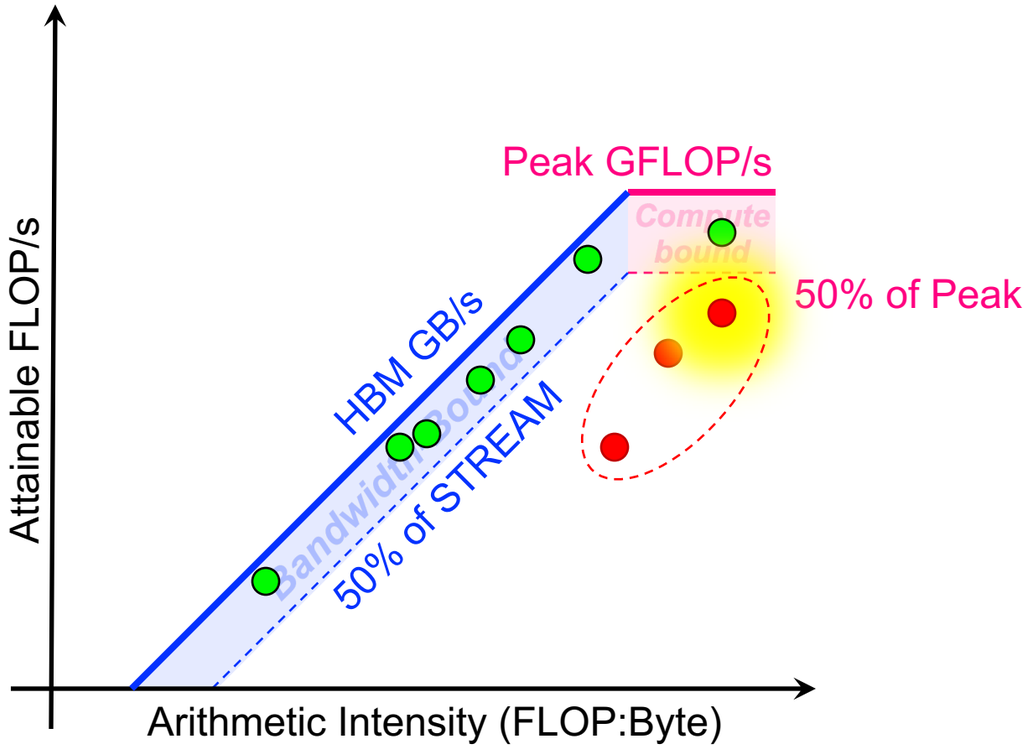
What is “Good” Performance?

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



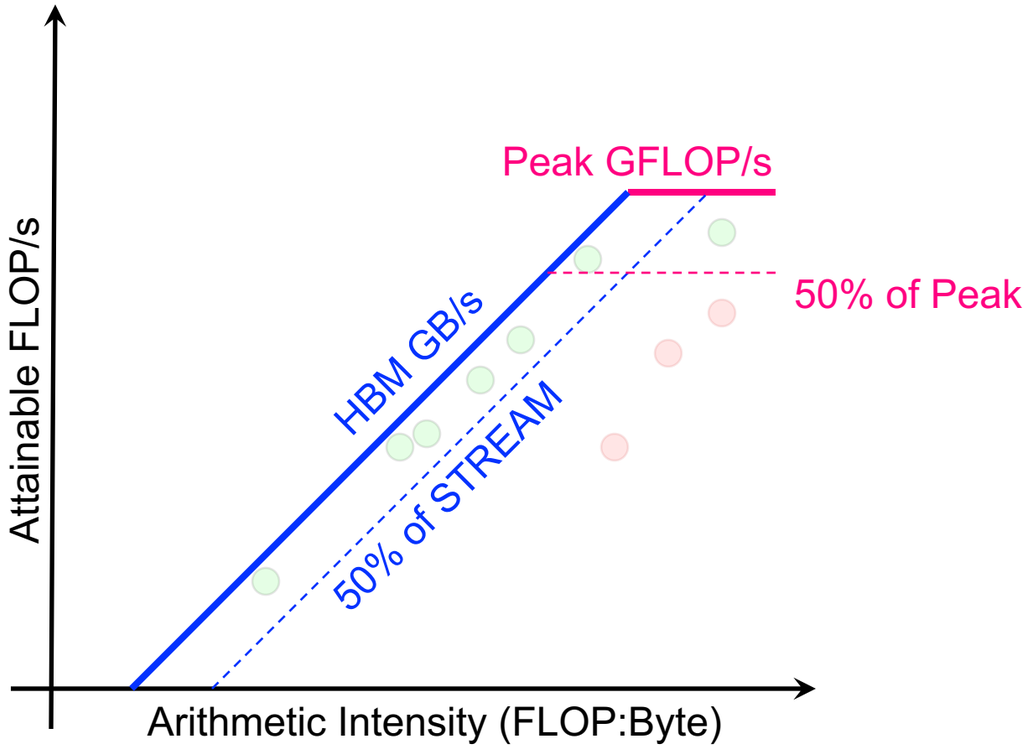
What is “Good” Performance?

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



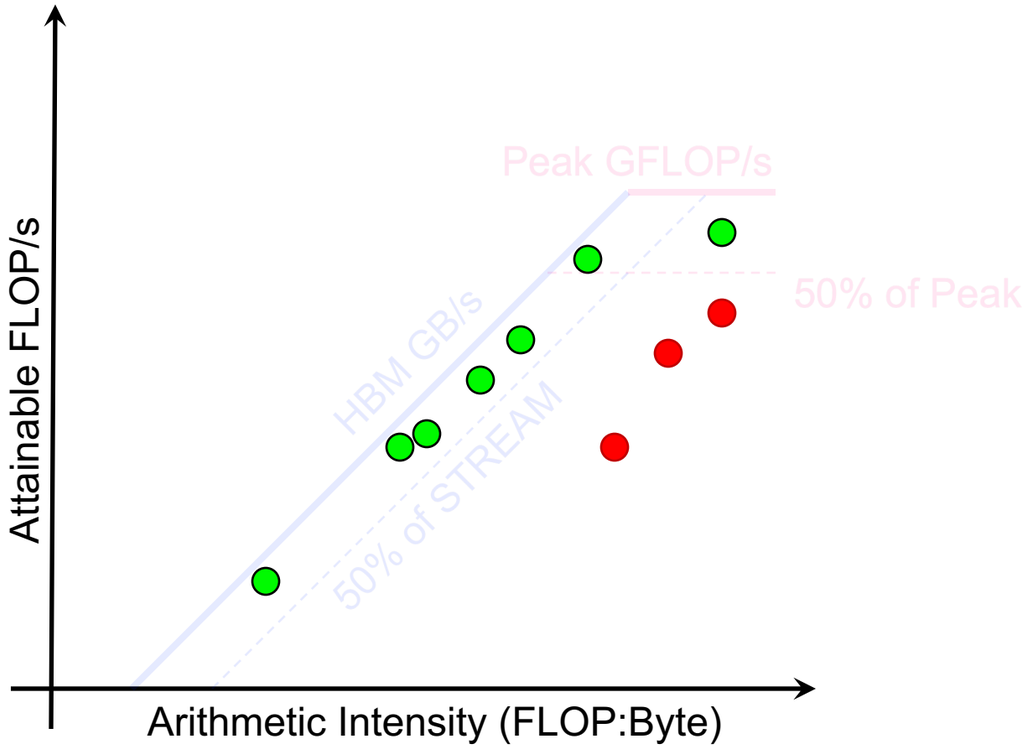
Roofline is made of two components

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



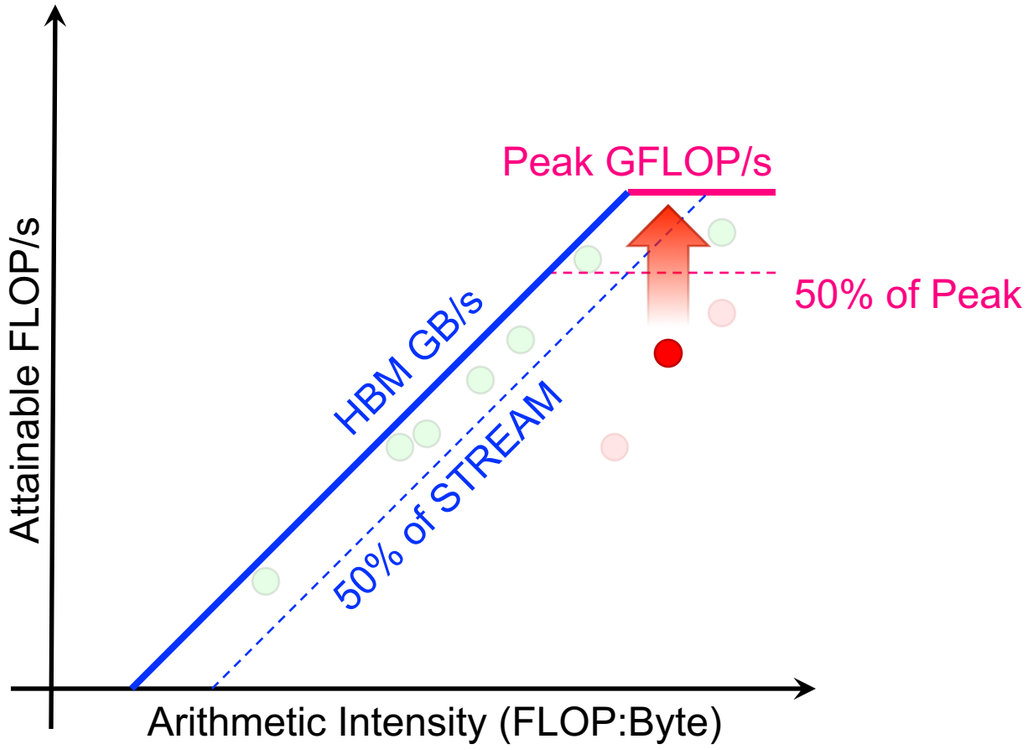
Roofline is made of two components

- 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



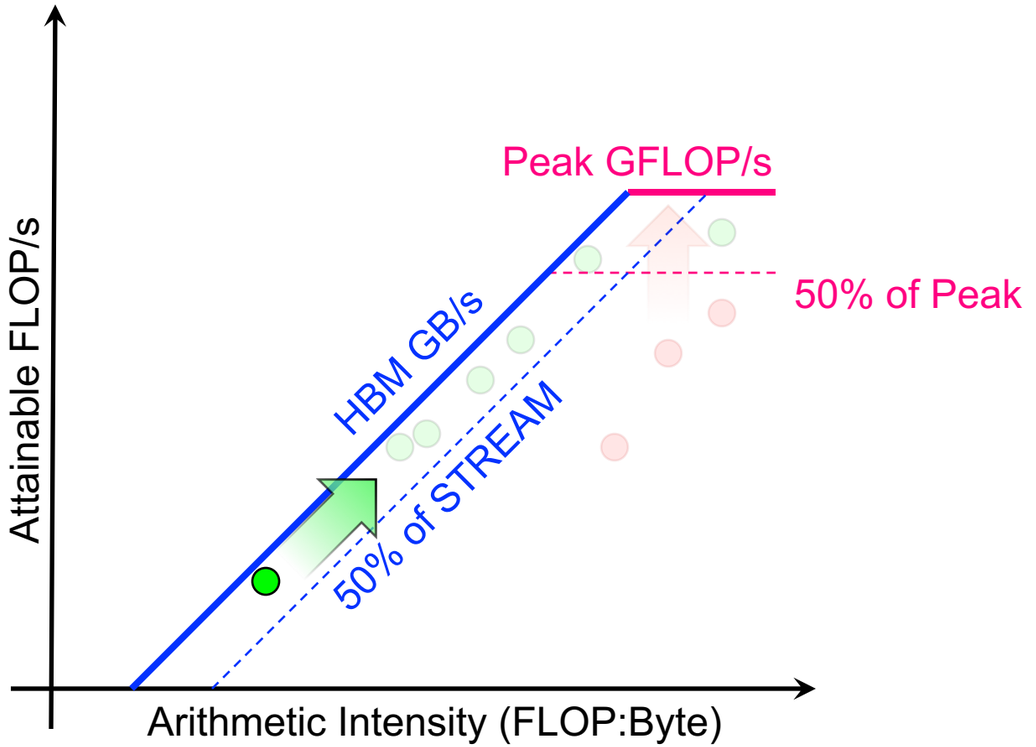
General Performance Optimization Strategy

- Get to the Roofline



General Performance Optimization Strategy

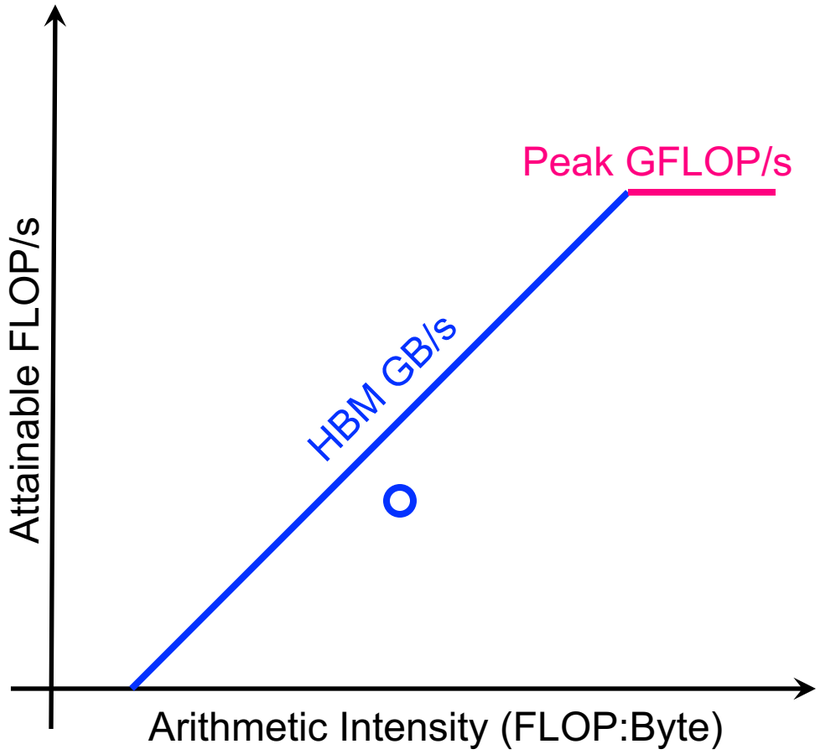
- Get to the Roofline
- Increase Arithmetic Intensity when bandwidth-limited
  - Reducing data movement (denominator) increases AI
  - Spatial locality, cache blocking, data structures, data types, etc…




How can performance ever be below the Roofline?

How can performance be below the Roofline?

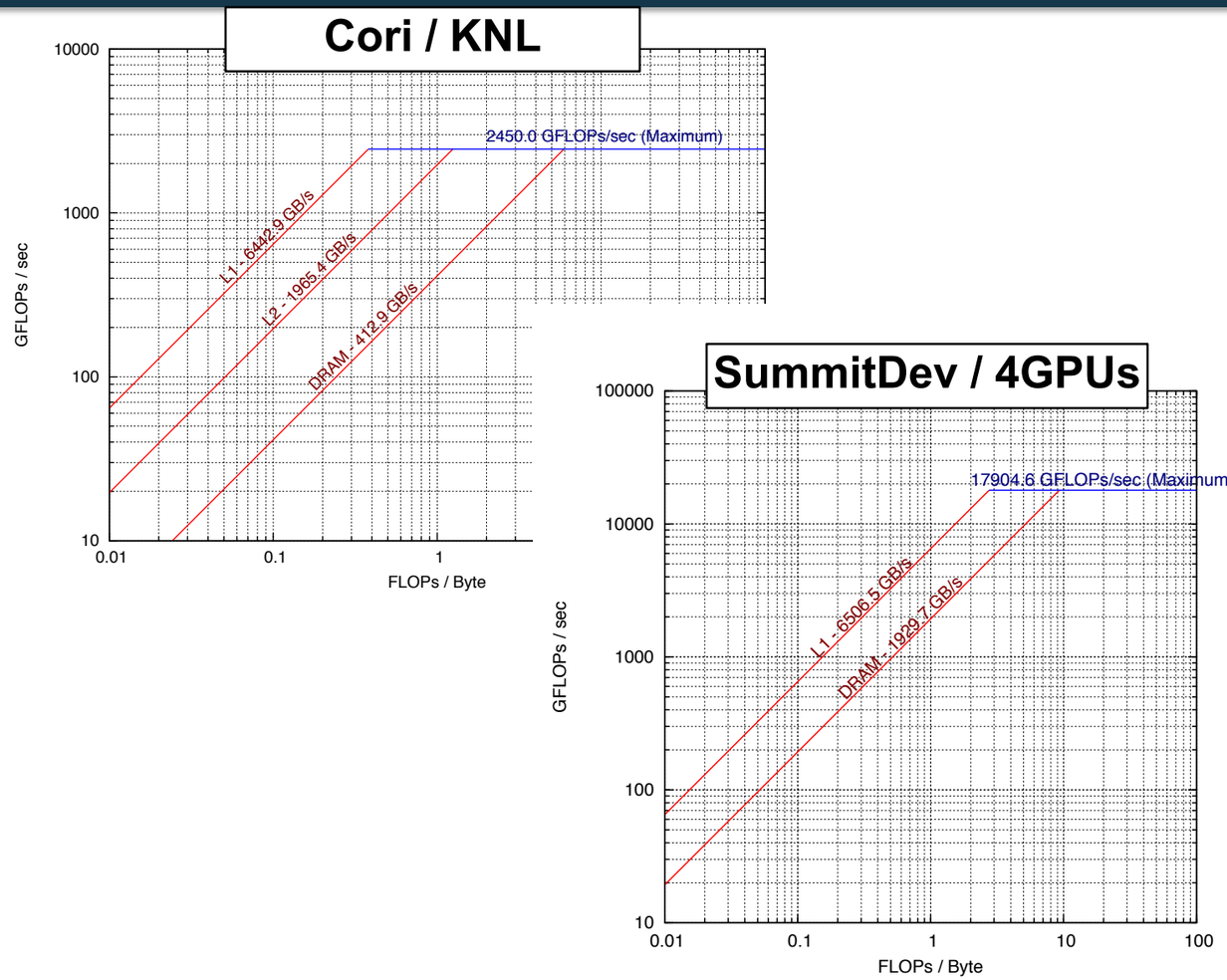
- Kernels (dots) are misplaced…
  Wrong #FLOPs or wrong #Bytes
  - Broken HW/SW performance counters
- Lines are misplaced…
  Peak HBM and FLOP/s are wrong
  - Use empirical approaches to model construction
  - Assumptions on load balance
- Missing lines…
  There are bounds other than DRAM and FLOPs
  - Insufficient cache bandwidth + locality
  - Didn't use FMA / Vectors / Tensors / …
  - Too many non-FP instructions
  - etc…




Below the Roofline?
Model or Application Instrumentation

Machine Characterization

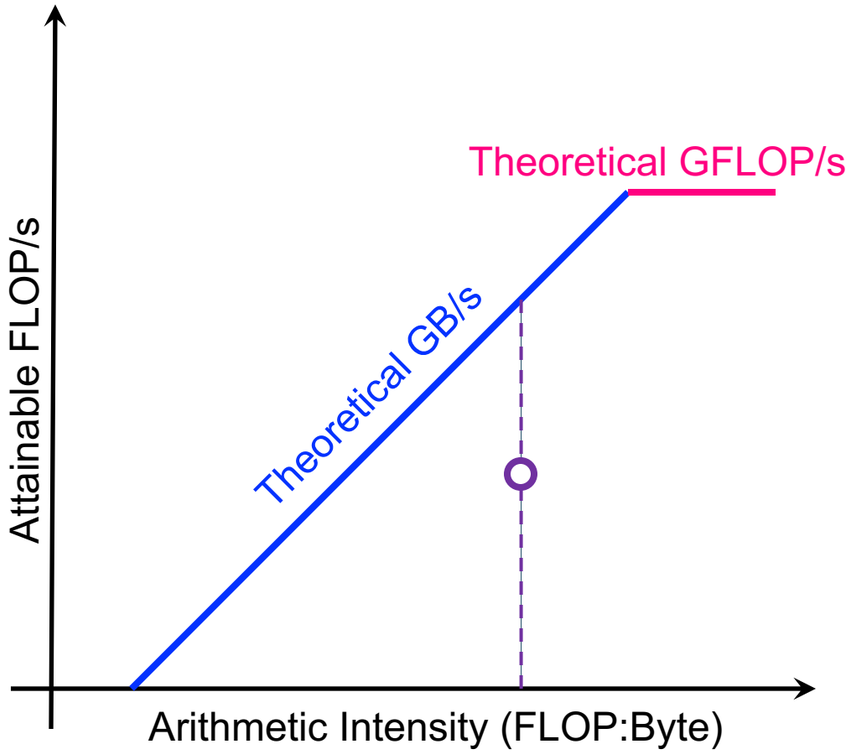
- Theoretical performance (specs) can be highly optimistic…
  - DRAM pin bandwidth vs. sustained
  - TurboMode / Underclocking
  - compiler failing on high-AI loops.
- Need empirical performance data
- LBL developed the Empirical Roofline Toolkit (ERT)…
  - Characterize CPU/GPU systems
  - Peak Flop 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

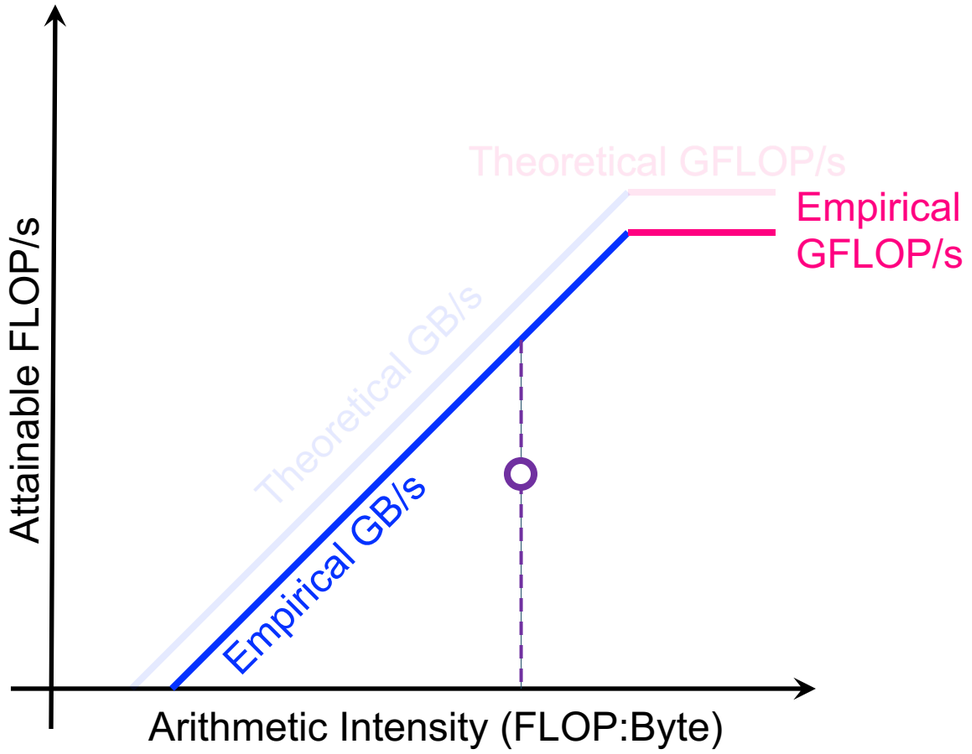
Theoretical vs. Empirical

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



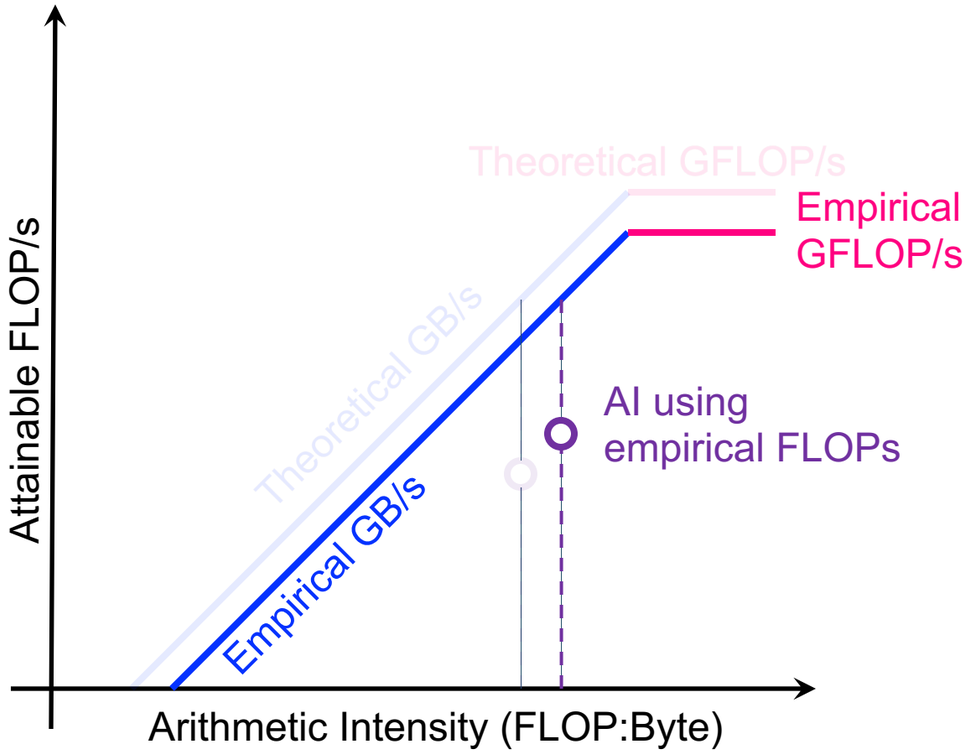
Theoretical vs. Empirical (Machine)

- Theoretical Roofline:
  - Pin bandwidth
  - FPUs * GHz
  - 1 C++ FLOP = 1 ISA FLOP
  - Data movement = Compulsory Misses
- Empirical Roofline:
  - Measured bandwidth
  - Measured Peak FLOP/s



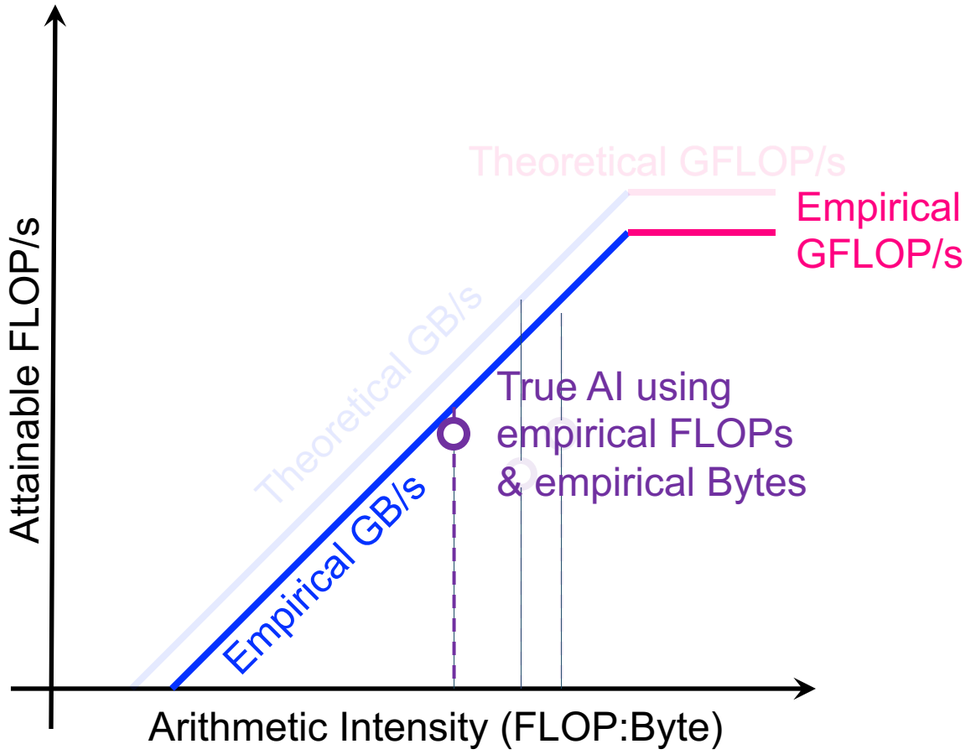
Theoretical vs. Empirical (Application FLOPs)

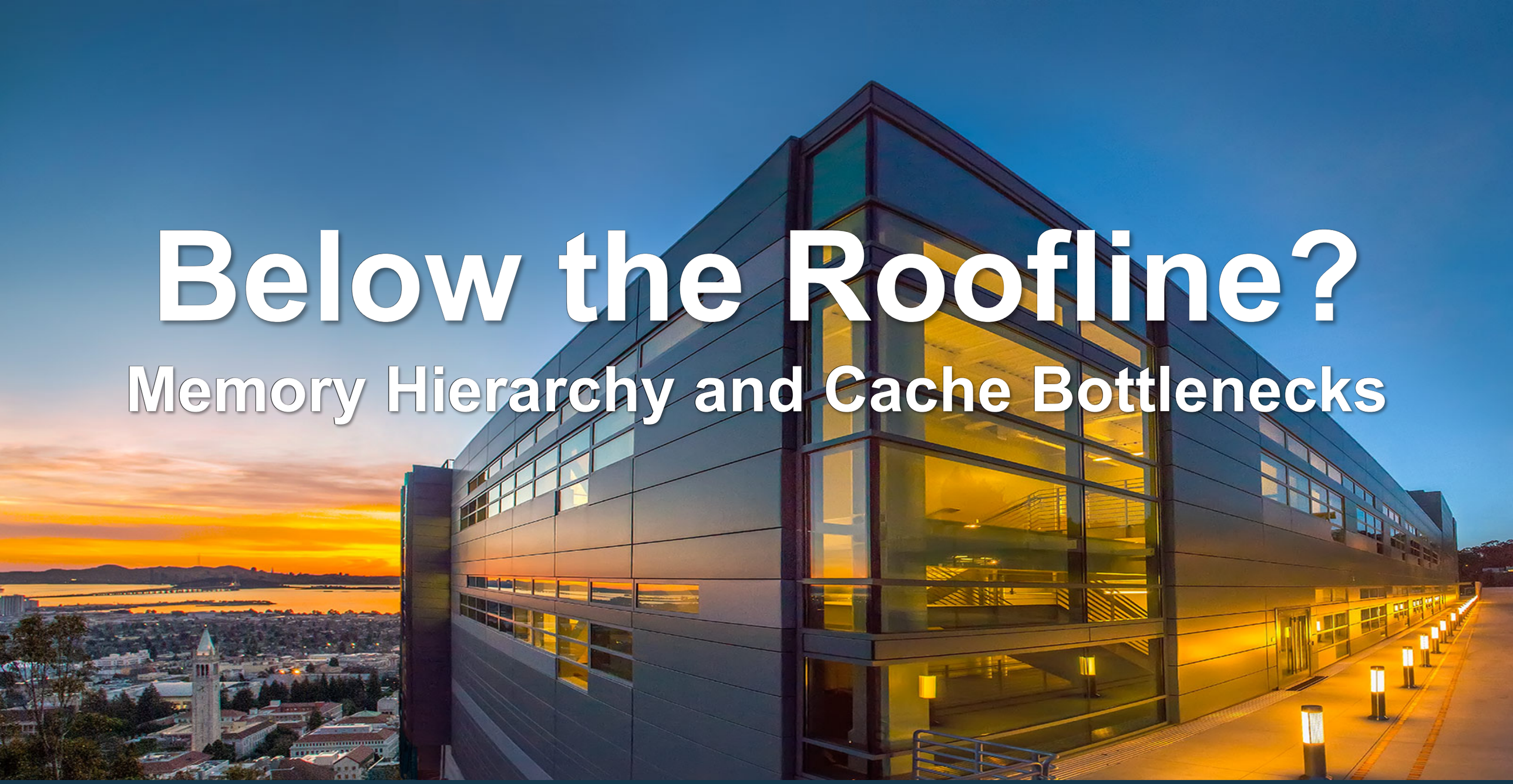
- Theoretical Roofline:
  - Pin bandwidth
  - FPUs * GHz
  - 1 C++ FLOP = 1 ISA FLOP
  - Data movement = Compulsory Misses
- Empirical Roofline:
  - Measured bandwidth
  - Measured Peak FLOP/s
  - 1 C++ FLOP >= 1 ISA FLOP (e.g. divide)



Theoretical vs. Empirical (Application Bytes)

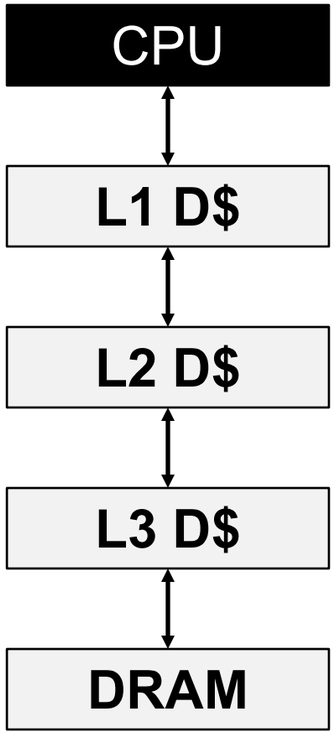
- Theoretical Roofline:
  - Pin bandwidth
  - FPUs * GHz
  - 1 C++ FLOP = 1 ISA FLOP
  - Data movement = Compulsory Misses
- Empirical Roofline:
  - Measured bandwidth
  - Measured Peak FLOP/s
  - 1 C++ FLOP >= 1 ISA FLOP (e.g. divide)
  - Measured data movement (cache effects)
  - ➢ **True Arithmetic Intensity can be higher or lower than expected**




Below the Roofline?
Memory Hierarchy and Cache Bottlenecks

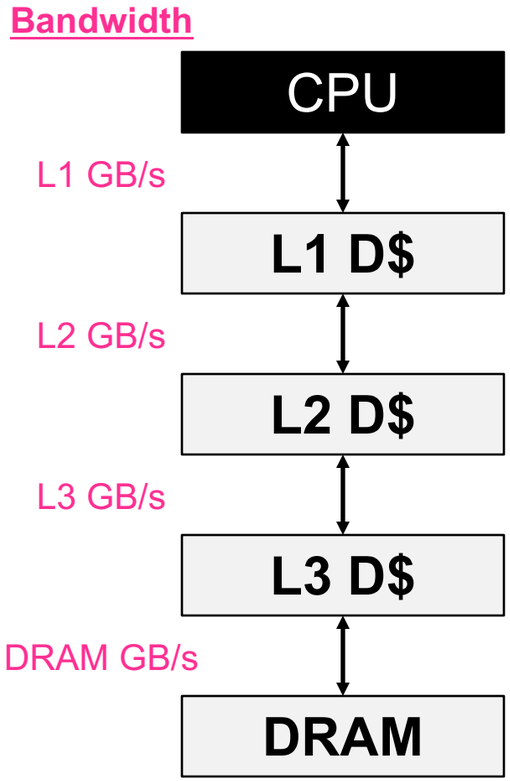
Memory Hierarchy

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



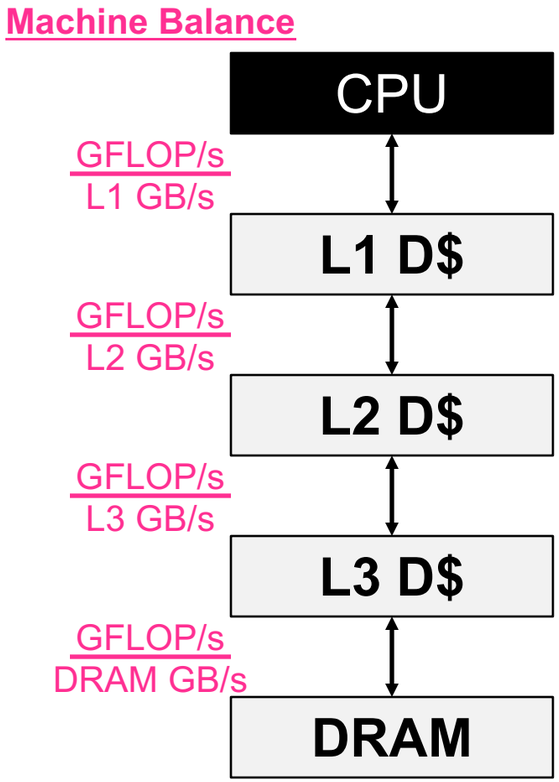
Memory Hierarchy

- Processors have different bandwidths for each level



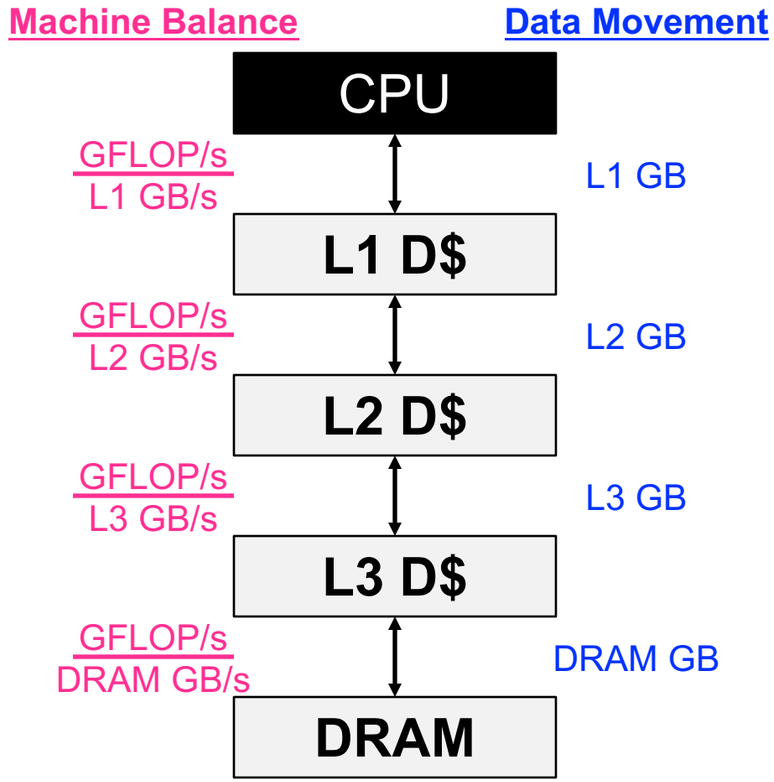
Memory Hierarchy

- Processors have different bandwidths for each level
  - different **machine balances** for each level



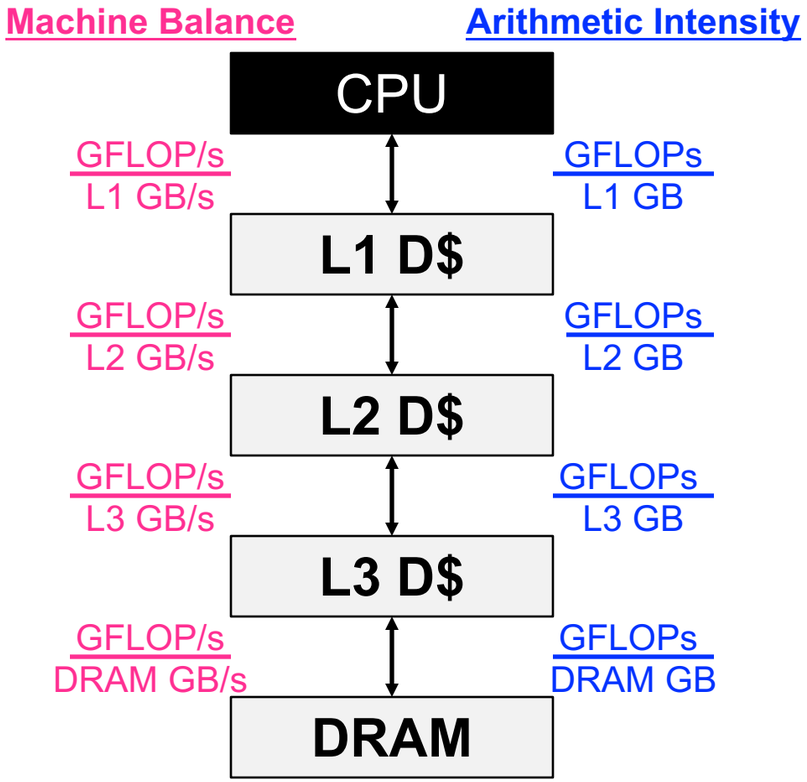
Memory Hierarchy

- Processors 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

- Processors 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

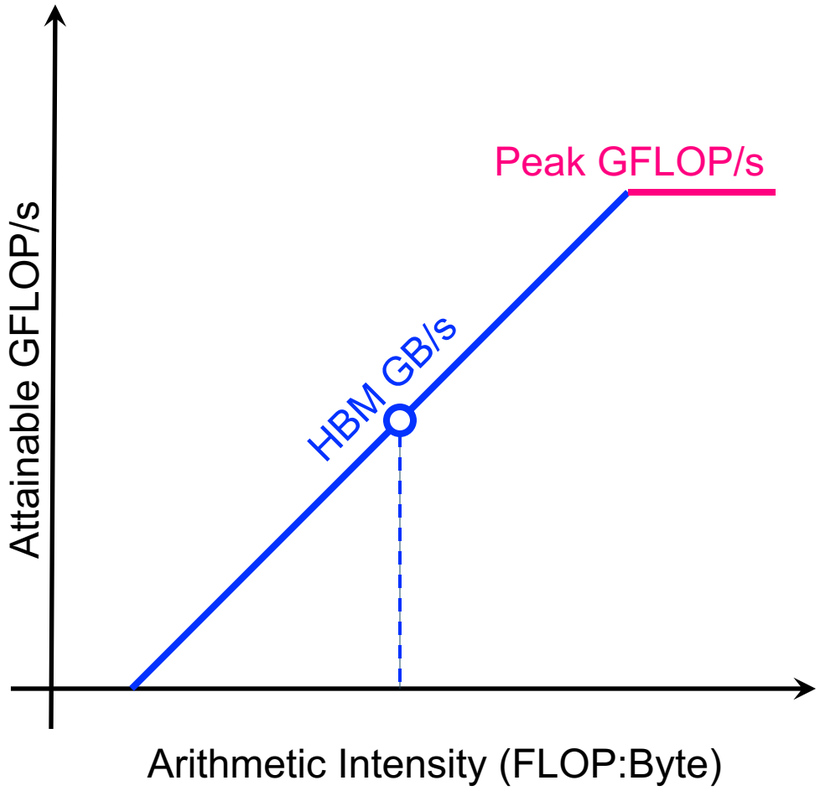
- For each additional level of the memory hierarchy, we can add another term to our model…

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

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

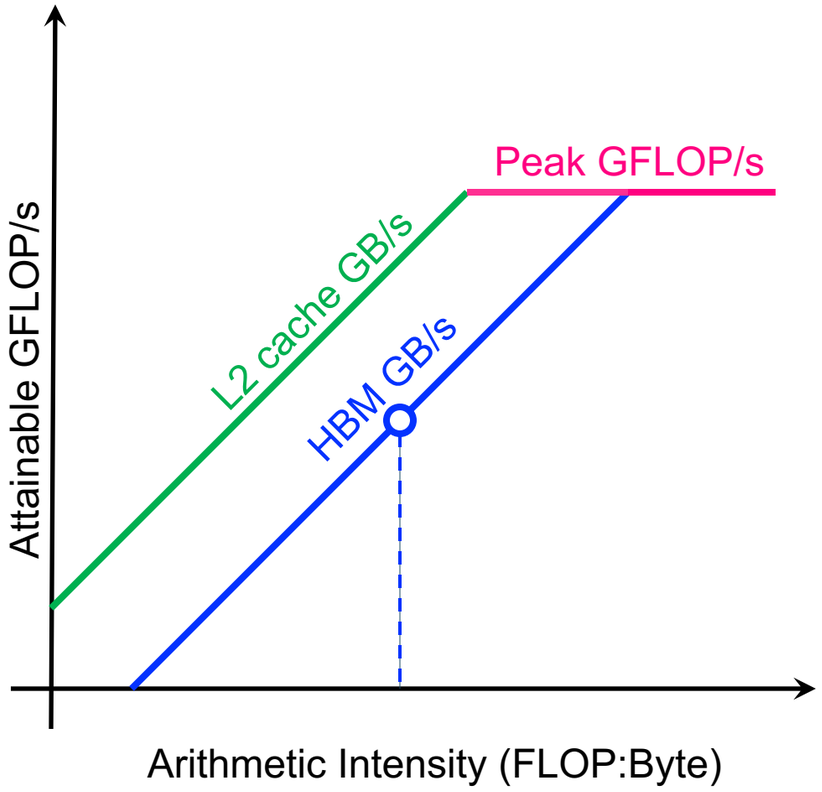
Cache Bottlenecks

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



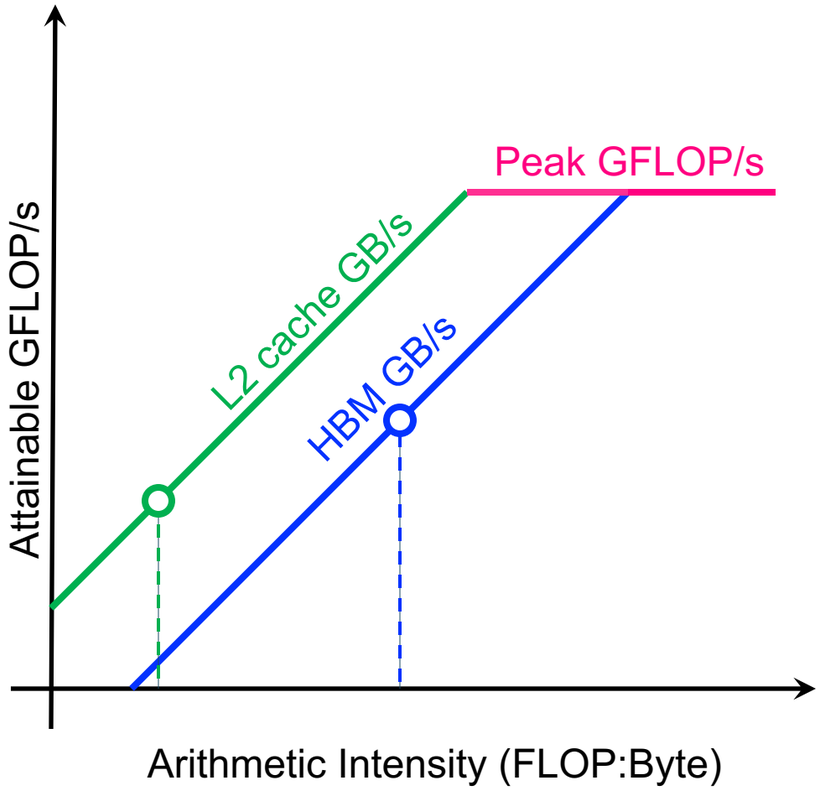
Cache Bottlenecks

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



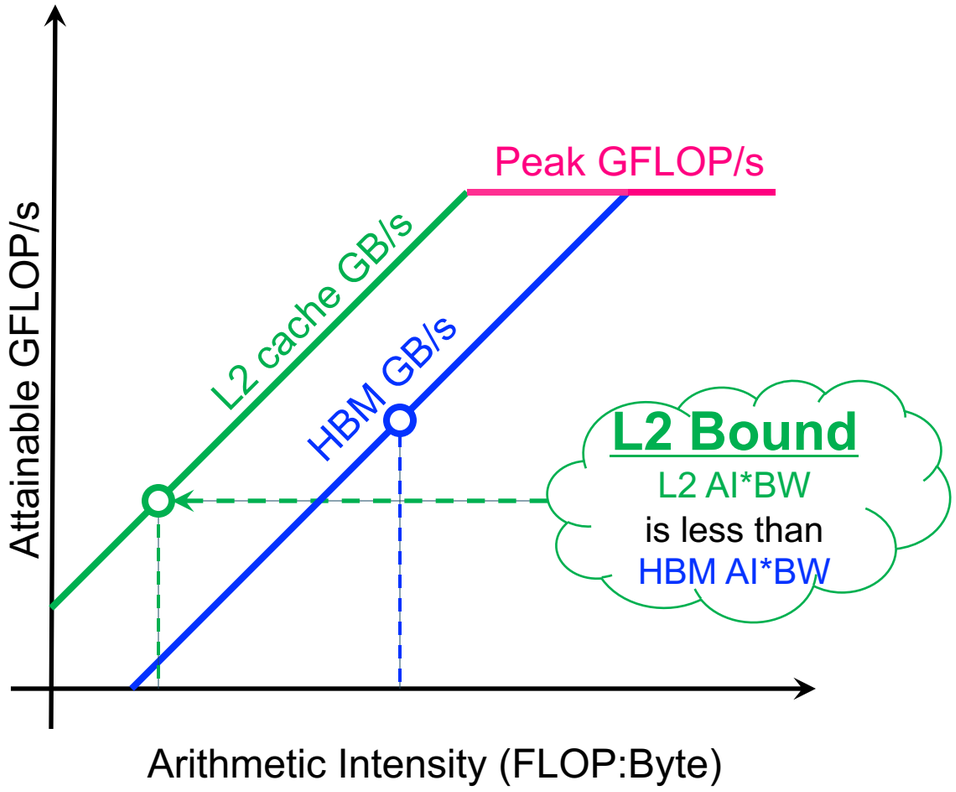
Cache Bottlenecks

- 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



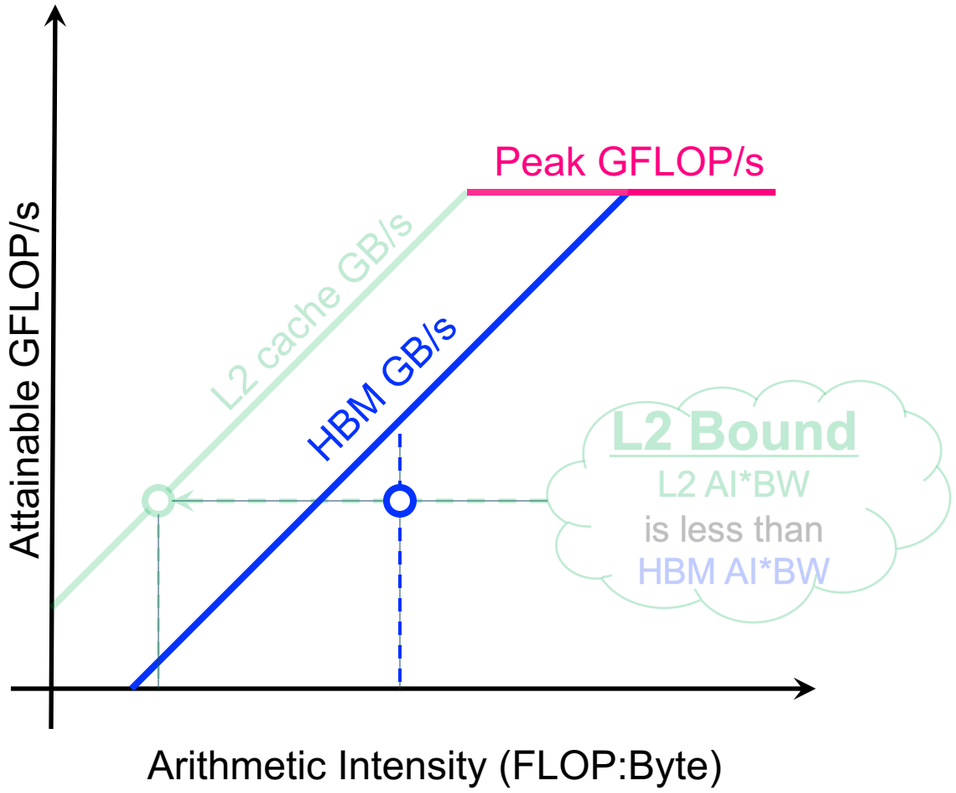
Cache Bottlenecks

- 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**



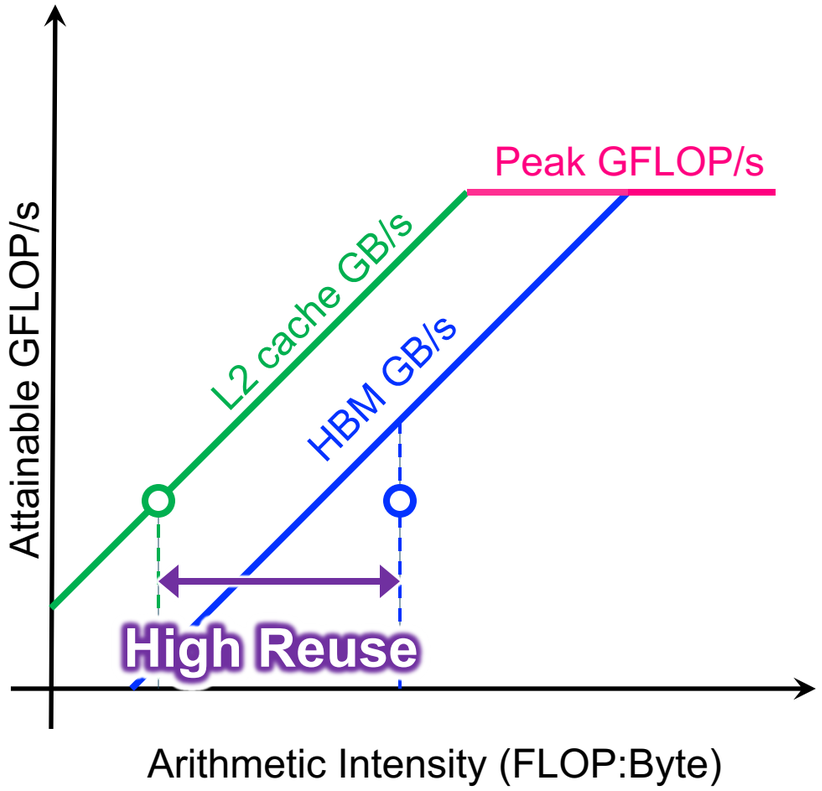
Cache Bottlenecks

- 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 Bottlenecks

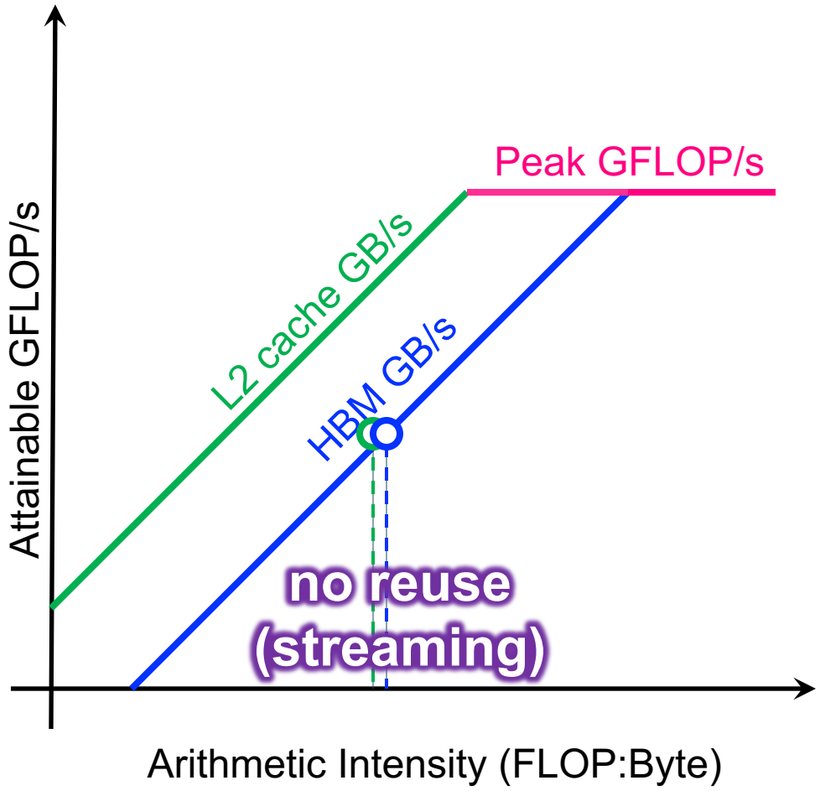
- Widely separated Arithmetic Intensities indicate high reuse in the cache



Cache Bottlenecks

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

- ***Same concepts on GPUs***




Below the Roofline?
FMA, Vectorization, Tensor Cores
BERKELEY LAB
LAWRENCE BERKELEY NATIONAL LABORATORY
BERKELEY LAB
U.S. DEPARTMENT OF
ENERGY

Return of CISC

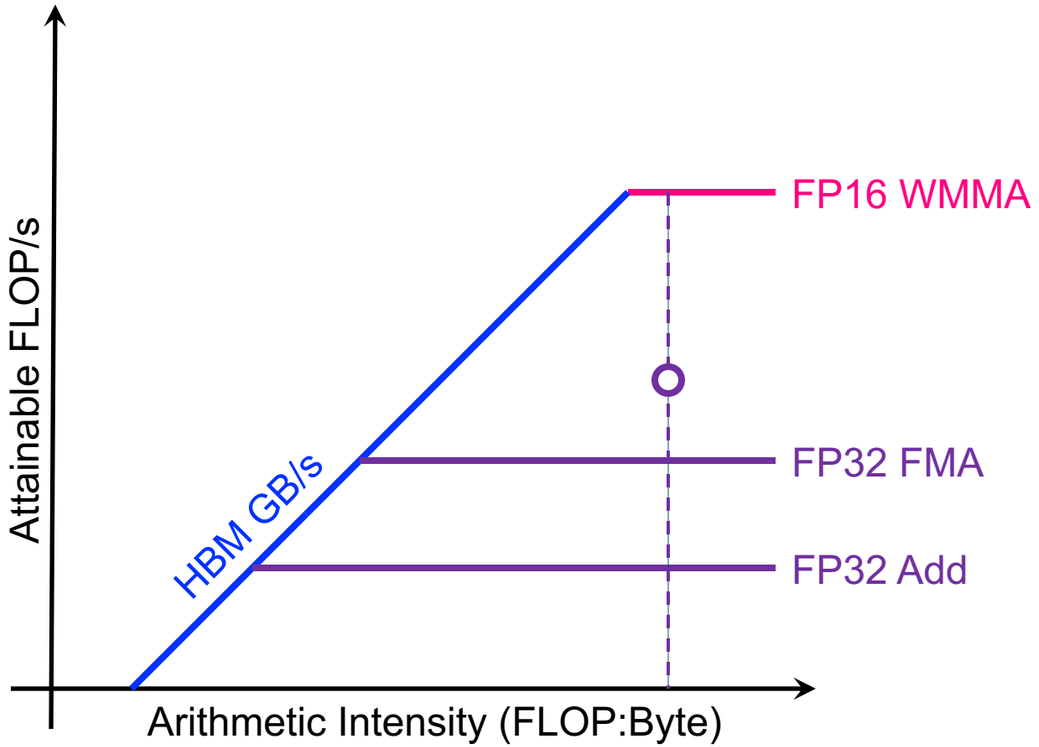
- Vectors have their limits (finite DLP, register file energy scales with VL, etc…)
- Death of Moore's Law is reinvigorating Complex Instruction Set Computing (CISC)

- 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*

➢ **If instructions are a mix or scalar (predicated), vector, and matrix operations, performance is now a weighted average of them.**

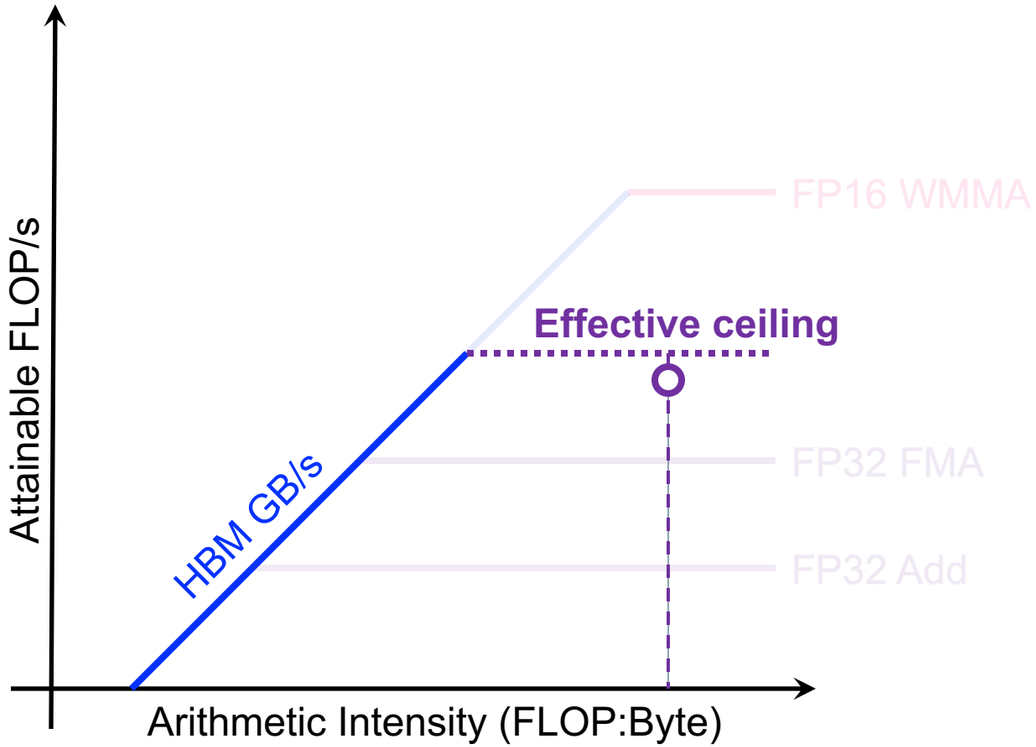
Return of CISC

- Consider NVIDIA Volta GPU…
  - ~100 TFLOPs for FP16 Tensor
  - 15 TFLOPS for FP32 FMA
  - 7.5 TFLOPs for FP32 Add
- DL applications mix Tensor, FP16, and FP32
- DL performance may be well below nominal Tensor Core peak



Return of CISC

- Consider NVIDIA Volta GPU…
  - ~100 TFLOPs for FP16 Tensor
  - 15 TFLOPS for FP32 FMA
  - 7.5 TFLOPs for FP32 Add
- DL applications mix Tensor, FP16, and FP32
- DL performance may be well below nominal Tensor Core peak
- The actual mix of instructions introduces an **effective ceiling** on performance…




Below the Roofline?
FPU Starvation

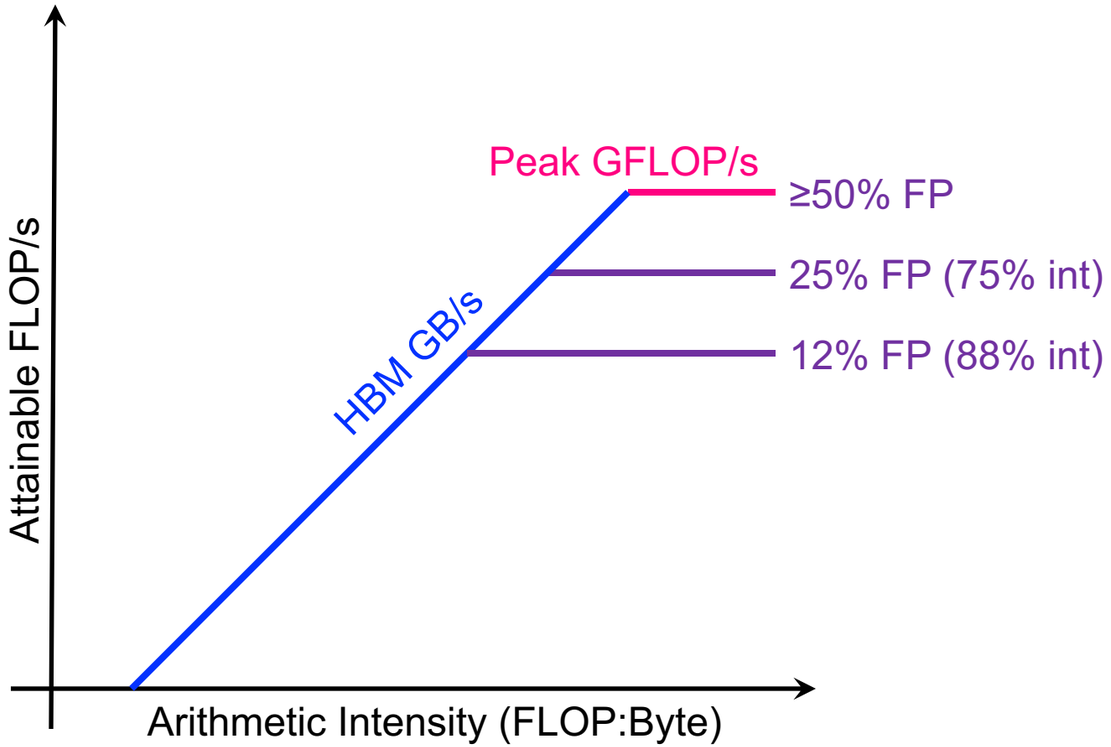
FPU Starvation

- Processors have finite instruction fetch/decode/issue bandwidth
- Moreover, the number of FP units dictates the FP issue rate required to hit peak

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

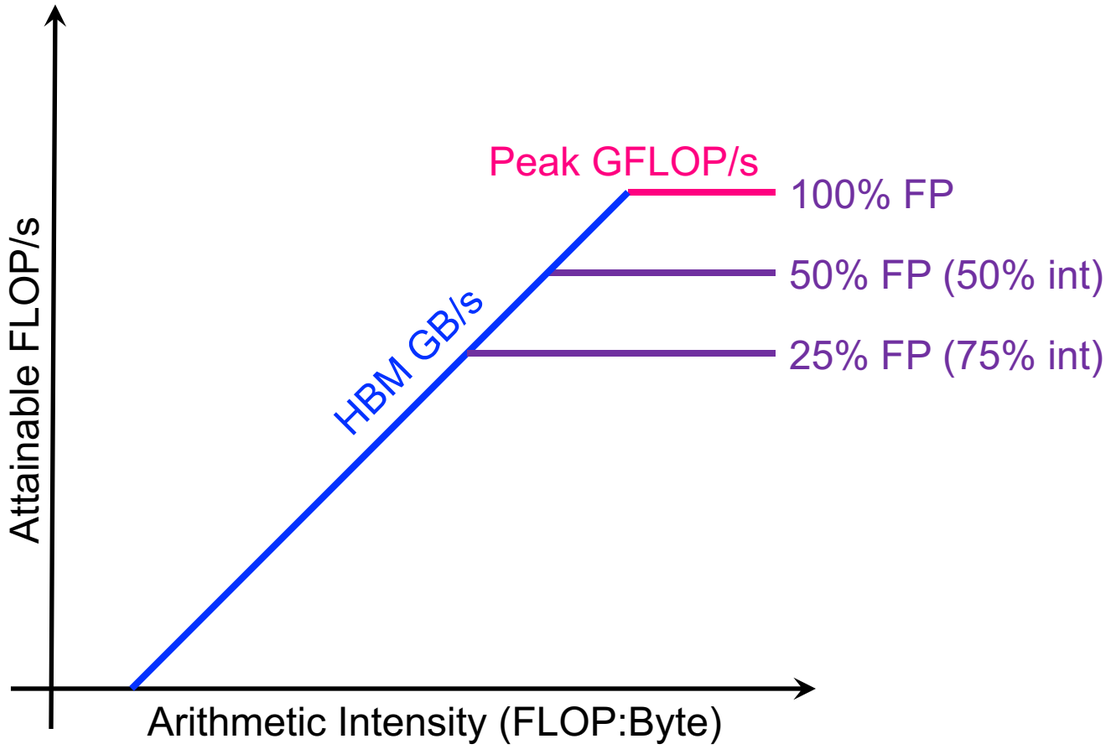
FPU Starvation

- Consider…
  - 4-issue superscalar
  - 2 FP data paths
  - ➢ **>50% of the instructions** must be FP to have any chance at peak performance



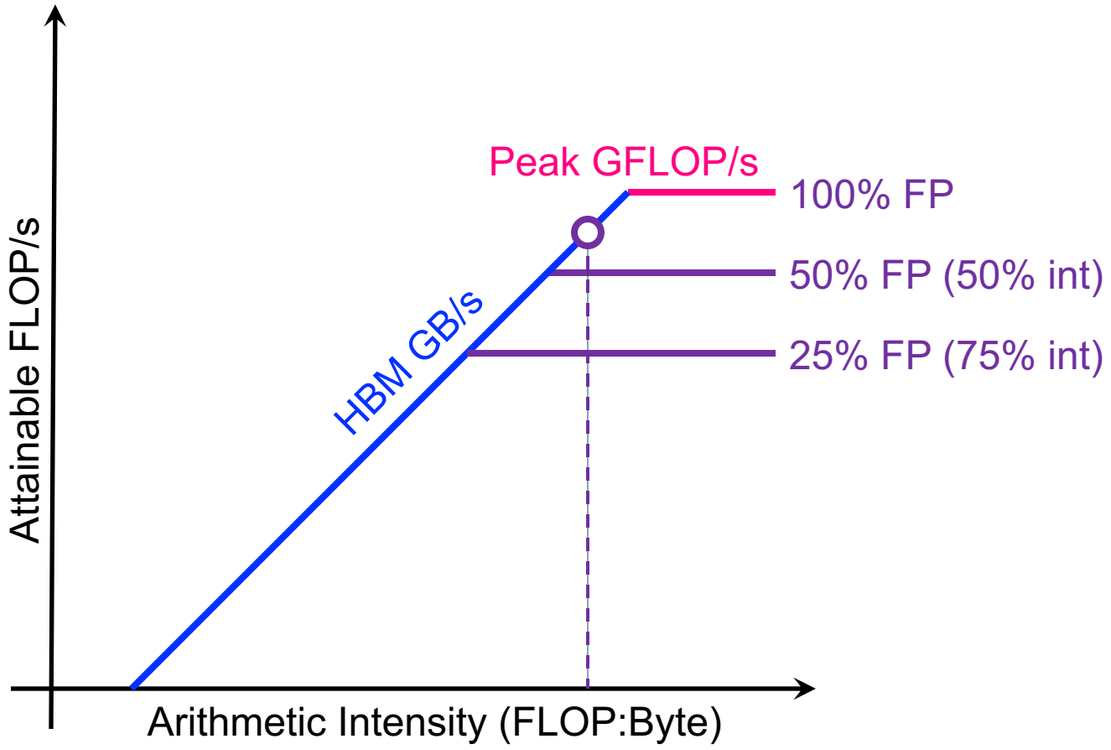
FPU Starvation

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



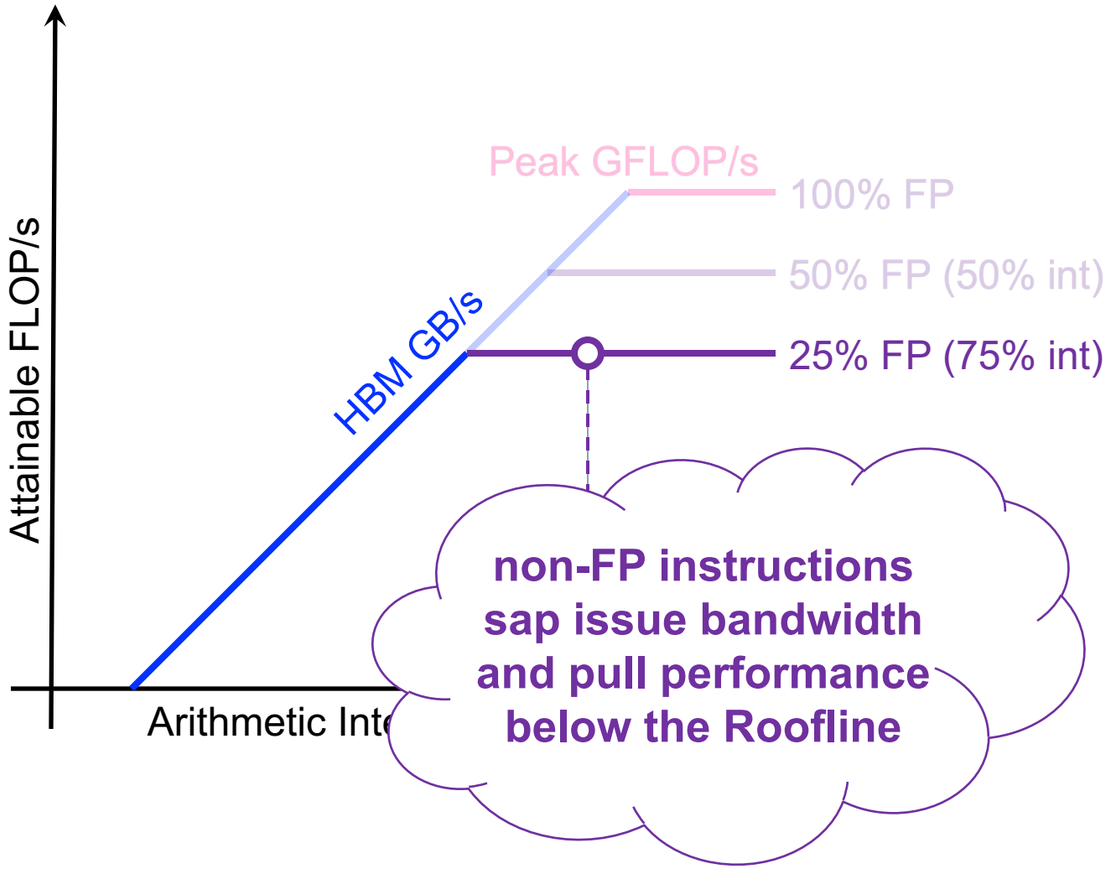
FPU Starvation

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



FPU Starvation

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



Instruction Roofline Model

When FLOP/s aren't what's important

Nan Ding, Samuel Williams, "An Instruction Roofline Model for GPUs", Performance Modeling, Benchmarking, and Simulation (PMBS), November, 2019.

How do we go beyond the FLOP Roofline?

- Think about classifying applications by instruction mix…
  - Heavy floating-point (rare in DOE)
  - Mix of integer and floating-point
  - Integer-only (e.g. bioinformatics, graphs, etc…)
  - Mixed precision
- FLOP/s → IntOP/s → FLOP/s+IntOP/s
  - Adopted by Intel Advisor
  - Useful when wanting to understand ‘performance’ rather than bottlenecks
  - **Instruction Fetch/Decode/Issue bottlenecks?**
  - **Functional Unit Bottlenecks?**
  - ➢ **Need instruction Roofline Model**

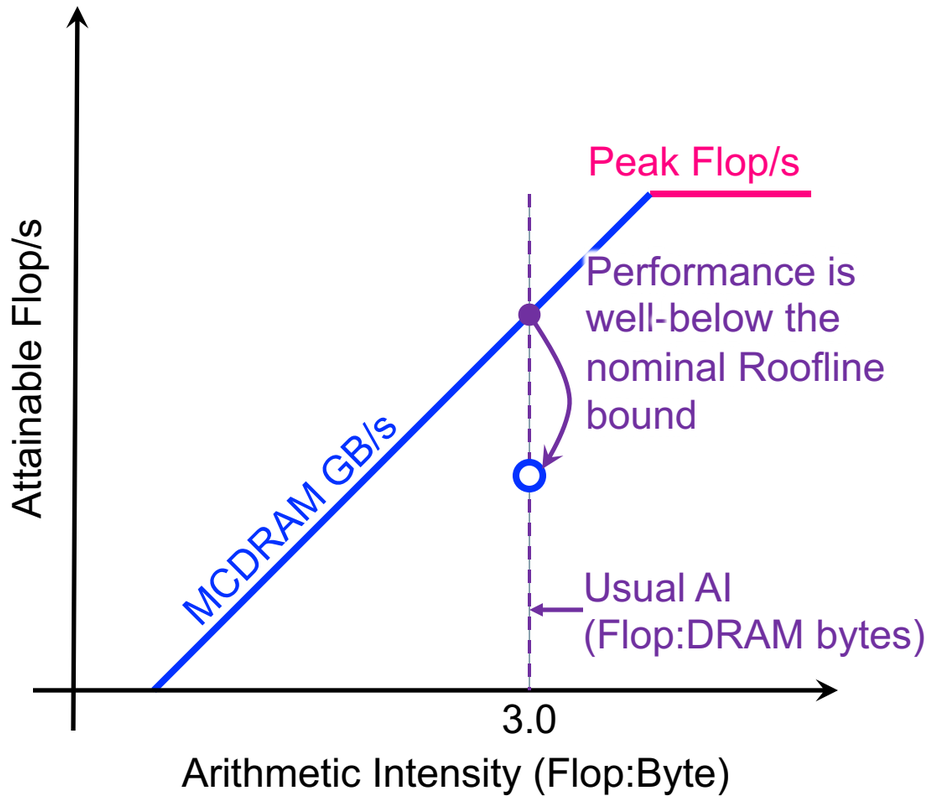
FLOP Roofline → VUOP Roofline

- On SIMD machines, one might consider vuop/s instead of flop/s…

✓ vuop/s (scalar + vector) can easily be mapped to vector unit utilization
✓ 100% vector unit utilization can bottleneck performance
✓ Performance counters give vuop/s and not flop/s
✗ 100% vector unit utilization does not imply 100% of peak (FMA, scalar vs. vector)

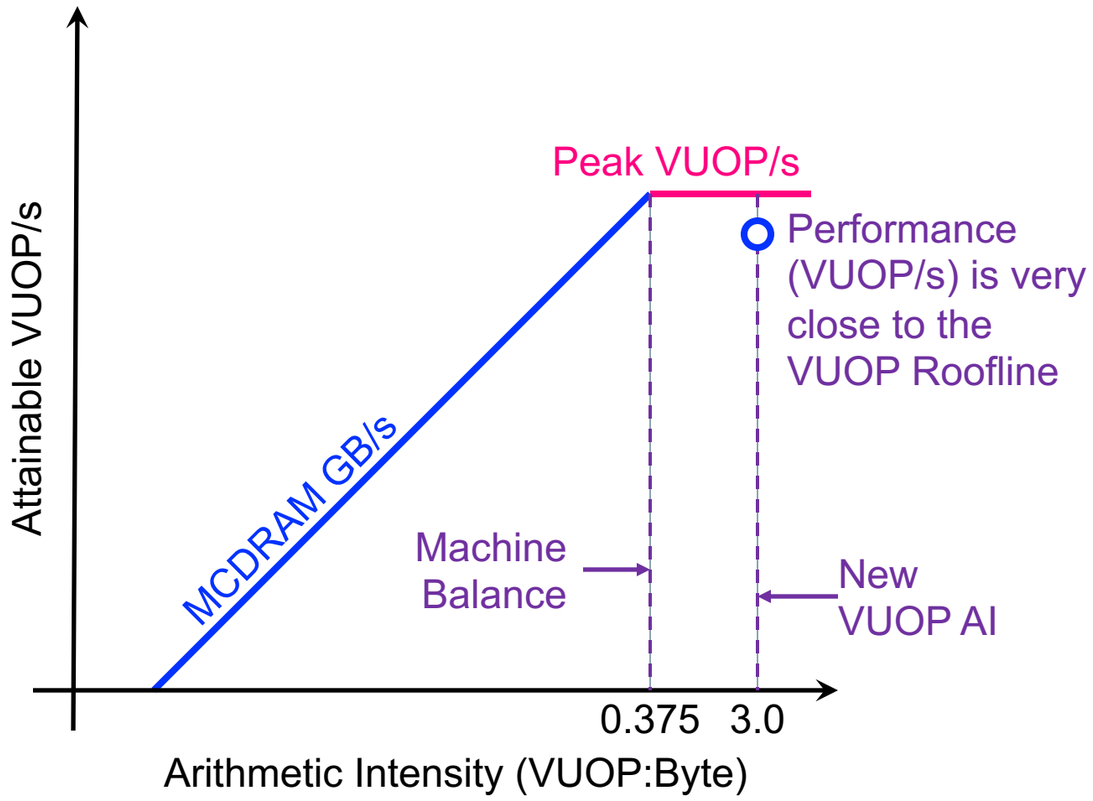
FLOP Roofline

- With performance counters alone, its hard to deduce why performance is well-below the FLOP Roofline.
  - VL?
  - Precision?
  - FMA?
  - Masks?
  - Non-FP vector instructions
- Moreover, one might conclude a code is memory bound when in reality is compute-bound



VUOP Roofline

- In a VUOP Roofline
  - machine peak (VUOP/s) is lower
  - machine balance (VUOP:Byte) is lower
- FLOP/s can be low, but VUOP/s can be high (and in the compute-bound regime)
- Could be used to understand FMA, vectorization, and mixed precision
- ➢ **Requires both performance counters (instructions, FLOPs by precision,) as well as dynamic code analysis (masks, VL, etc…)**

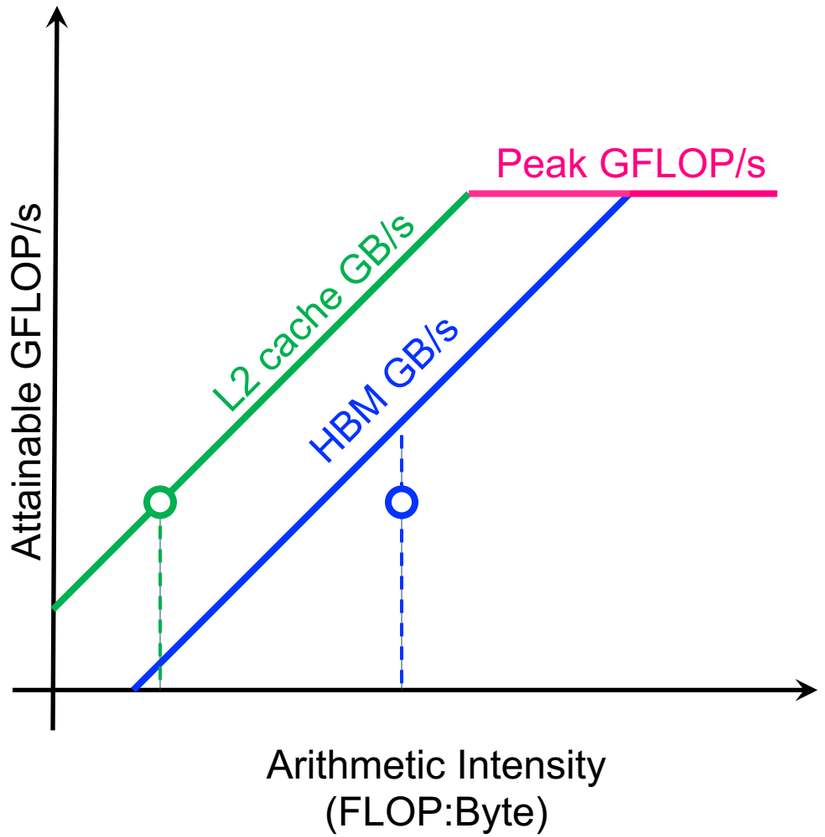


NVIDIA GPU Instruction Roofline

- Instructions/second? Instructions per Byte?
- What is an ‘Instruction’ on a GPU?
  - Thread-level hides issue limits?
  - Warp-level hides predication effects?
  - **Scale non-predicated threads down by the warp size (divide by 32)**
  - **Show warp instructions per second**
  - Break instructions into subclasses (integer, FP32, FP64, LDST, WMMA)
- Naively, one would think instruction intensity should use ‘bytes’
  - Matches well to existing Roofline; works with well-known bandwidths
- GPUs access memory using ‘transactions’
  - 32B for global/local/L2/HBM
  - 128B for shared memory
  - ➢ **“Instructions/Transaction” preserves traditional Roofline, but enables a new way of understanding memory access**

Instruction Roofline

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

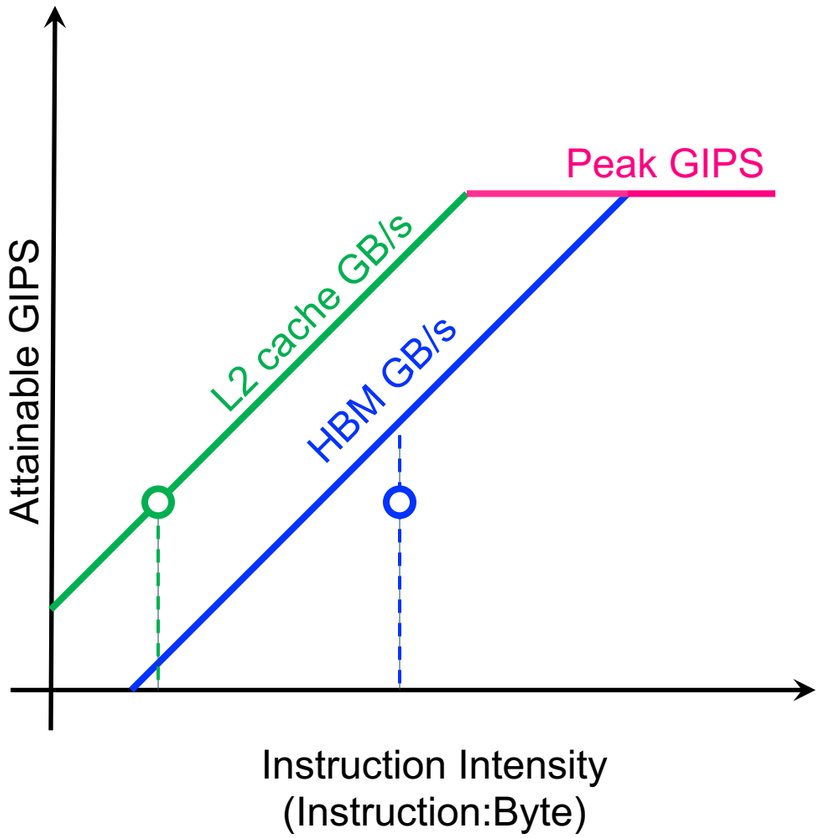


Instruction Roofline

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

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

Instructions per Byte



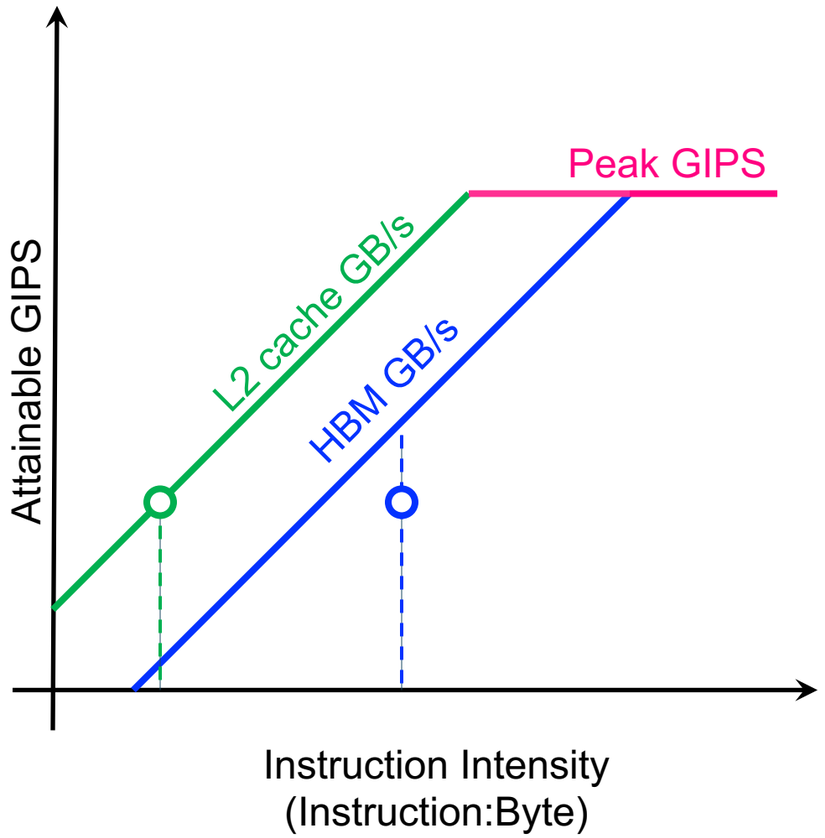
Instruction Roofline on GPUs

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

Warp Instructions

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

As the natural quanta for GPU memory access is a "transaction"…

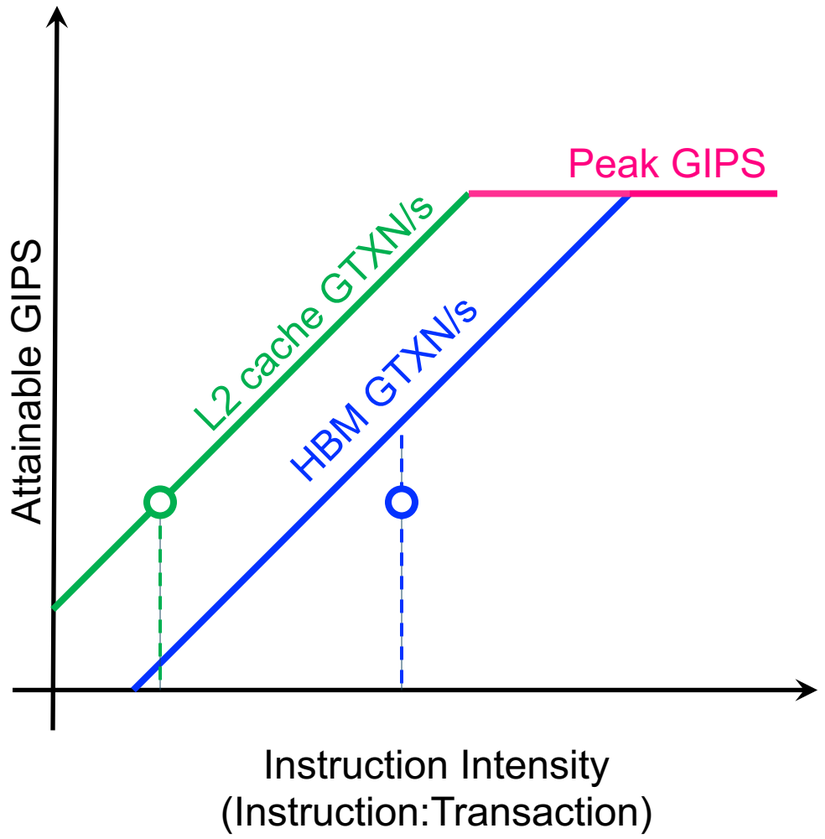


Instruction Roofline on GPUs

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

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

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



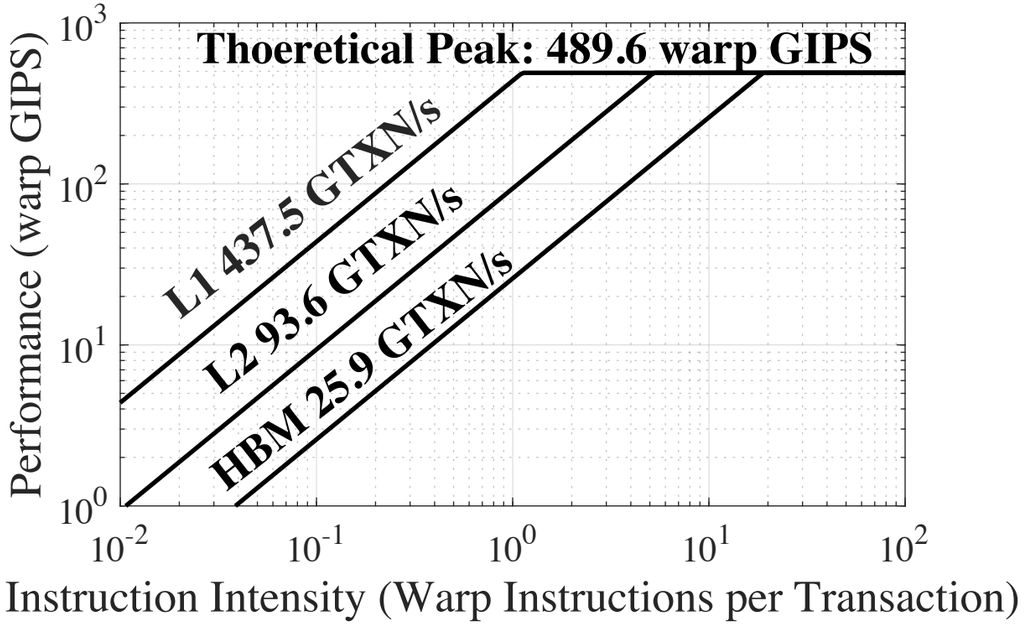
II_x (Instruction Intensity at level "x") =

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

Instruction Roofline on NVIDIA GPUs

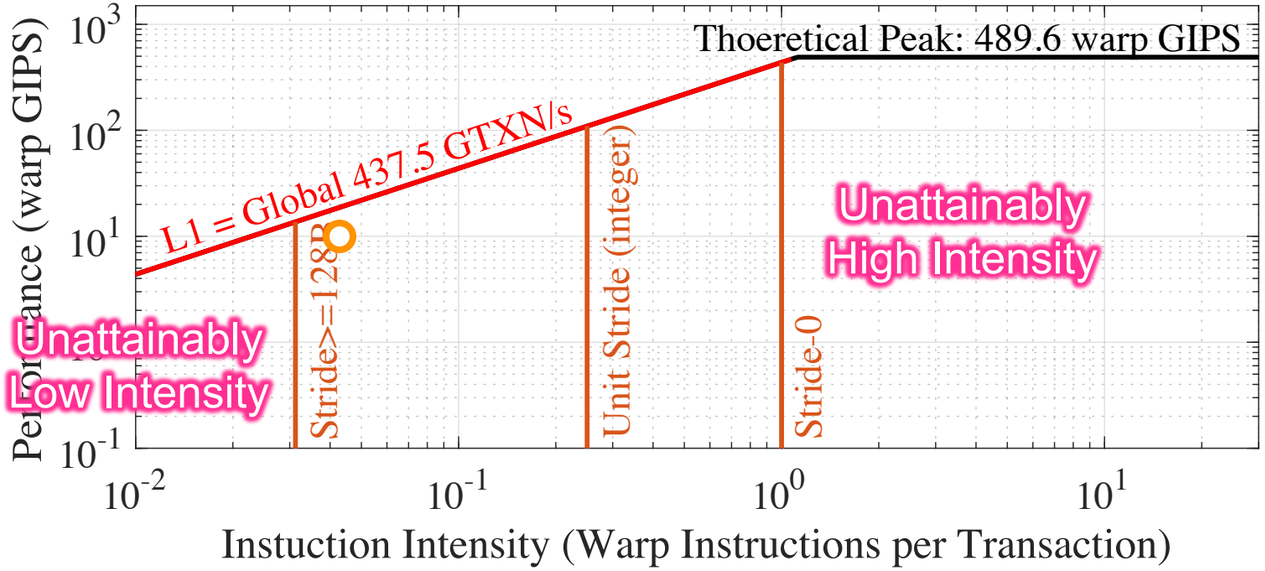
- Instruction Intensity (II)
  - (Warp or equivalent) Instructions / Transaction
  - Refine into L1 (global+local+shared), L2, HBM Instruction Intensities
  - Further refine based on instruction type (**LDST instructions / global transaction**)

- Peak Performance and Peak Bandwidths
  - Instruction:
    - 80 SMs * 4 warps * 1.53GHz ~ 490 GIPS (warp-level)
  - Use ERT for memory (convert from GB/s)
    - L1: 80 SMs * 4 transactions/cycle * 1.53 GHz ~ 490 GTXN/s
    - L2: 94 GTXN/s (empirical)
    - HBM: 26 GTXN/s (empirical)



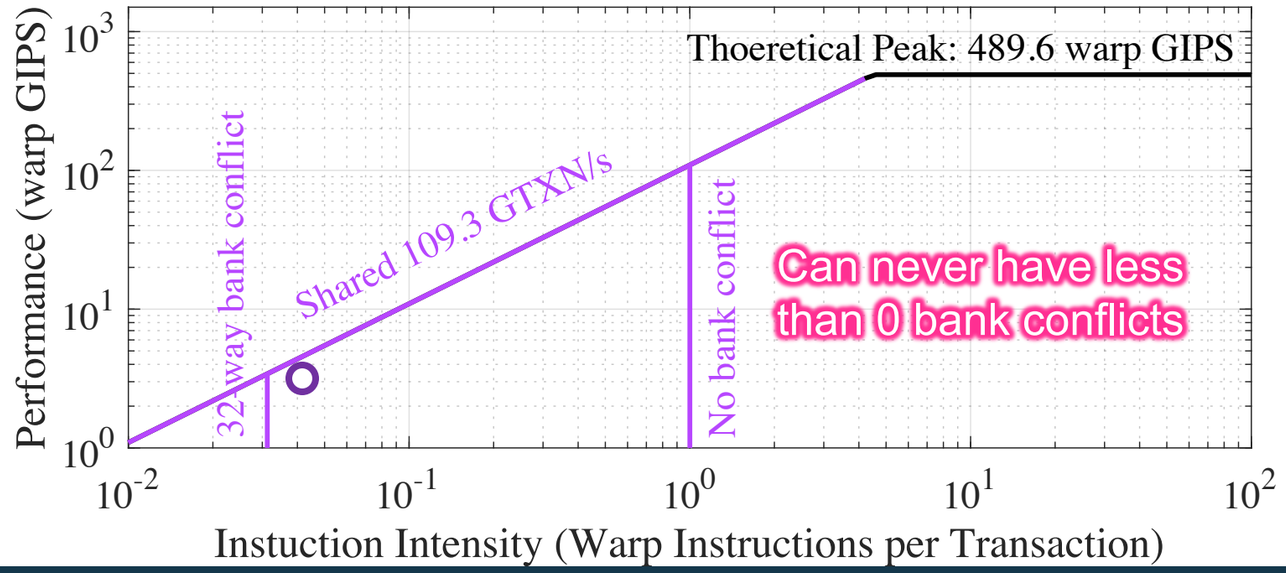
Efficiency of Global Memory Access

- (Global)LDST Instruction Intensity has a special meaning / use…
  - Global LDST instructions / Global transactions
  - Numerator lower than nominal II
  - Denominator can be lower than nominal L1 II (no local or shared transactions)
- Denotes efficiency of memory access
- 3 "**Walls**" of interest:
  - ≥1 transaction per LDST instruction (all threads access same location)
  - ≤32 transactions per LDST instruction (gather/scatter or stride>=128B)
  - Unit Stride: 1 LDST per 8 transactions (double precision)



Efficiency of Shared Memory Access

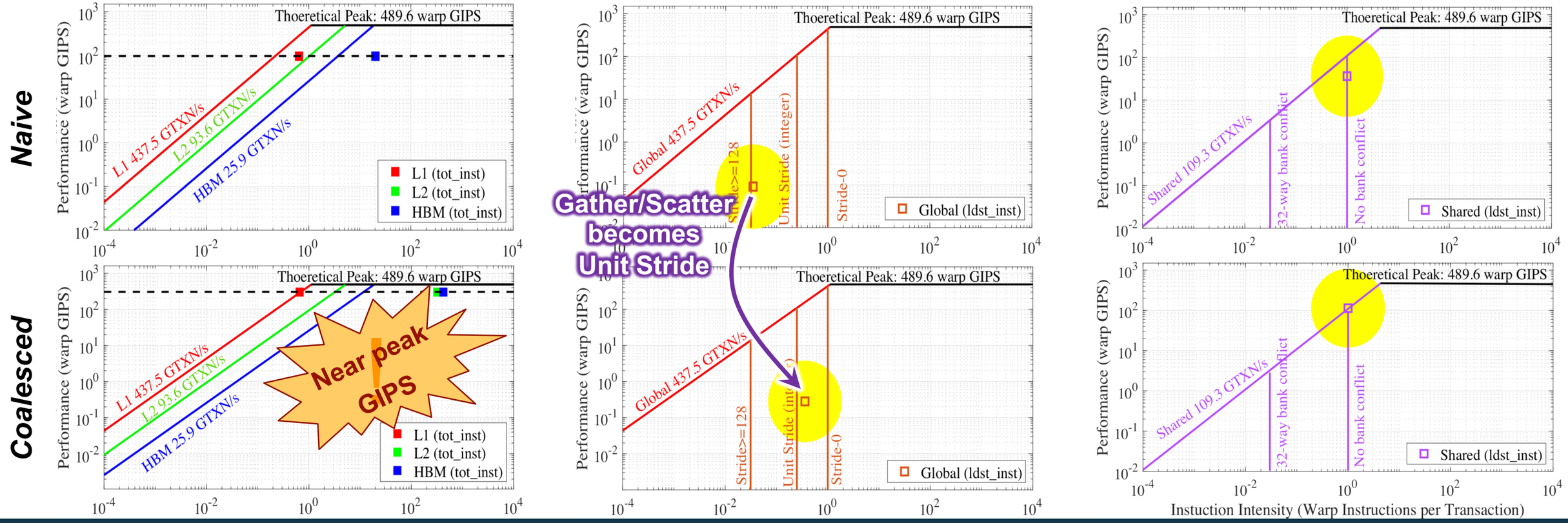
- (Shared)LDST Instruction Intensity also has a special meaning / use
  - Shared LDST instructions / Shared transactions
  - II is similarly loosely related to nominal II
- Can be used to infer the number of bank conflicts
- 2 "**Walls**" of interest:
  - Minimum of 1 transaction per shared LDST instruction (***no bank conflicts***)
  - Maximum of 32 transactions per shared LDST instruction (***all threads access different lines in the same bank***)



Instruction Roofline for Smith-Waterman

- Integer-only Alignment code on NVIDIA GPU
- No predication effects, but inefficient global memory access

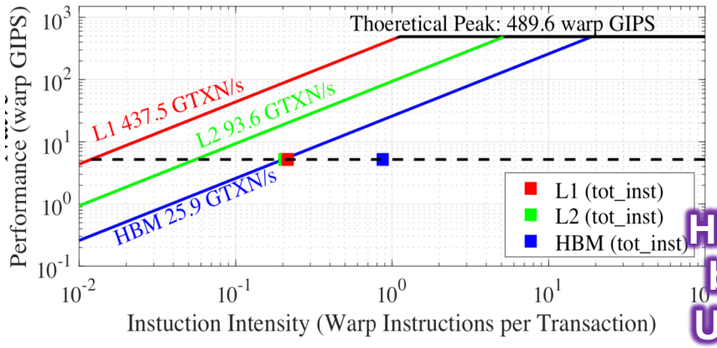
Instruction Hierarchy & Thread Predication ***Global Memory Efficiency*** ***Shared Memory Efficiency***

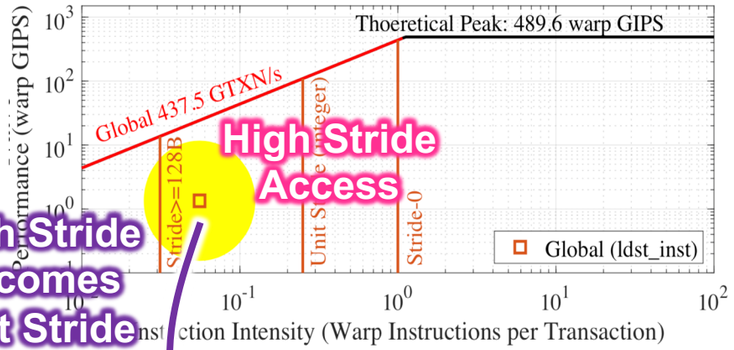


Instruction Roofline for Matrix Transpose

Instruction Hierarchy & Thread Predication ***Global Memory Efficiency*** ***Shared Memory Efficiency***

Naive



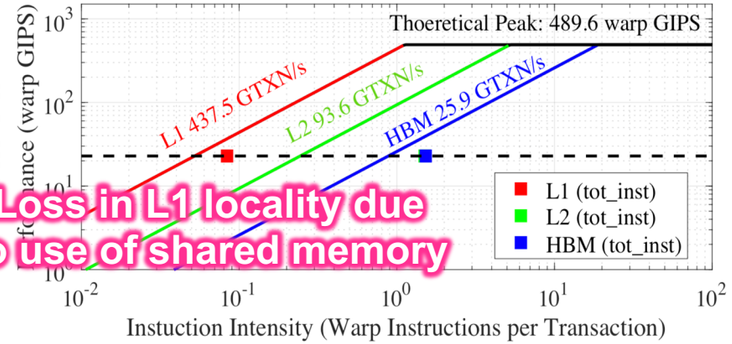


High Stride Access

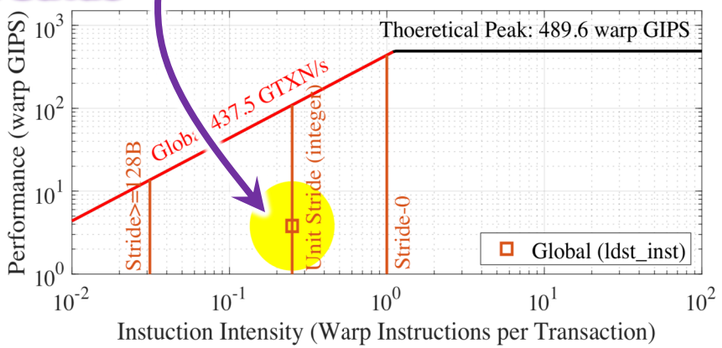
High Stride becomes Unit Stride

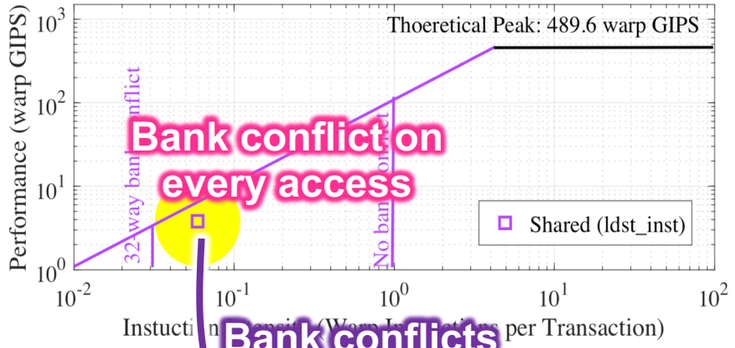
not used

Transpose in Shared



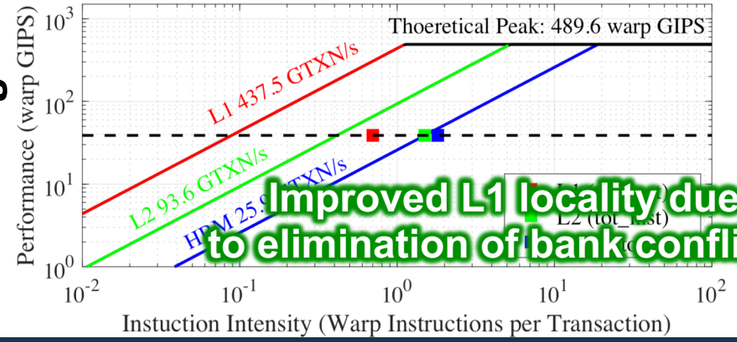
Loss in L1 locality due to use of shared memory



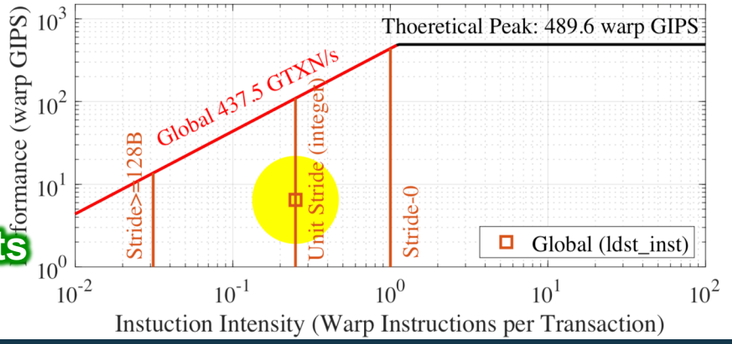


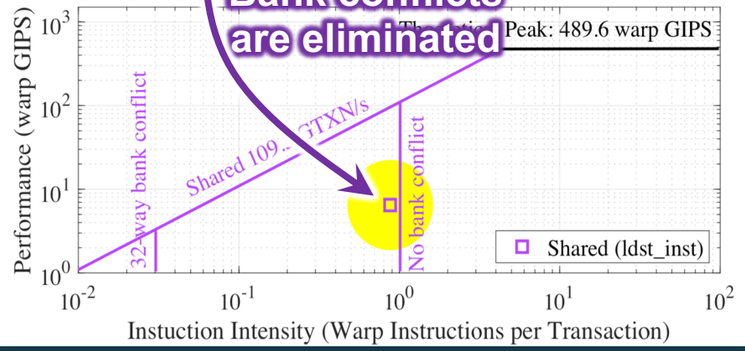
Bank conflict on every access

Array Padding



Improved L1 locality due to elimination of bank conflicts





Bank conflicts are eliminated

Other Uses of Instruction Roofline

- Predication
  - Individual threads can mask out execution when in branch-not-taken
  - 16 FLOPs/SM/cycle… 1 FP warp every 2 cycles
    –or –
    1 FP warp every cycle with half threads predicated
  - Use performance metrics to plot both **warp GIPS** and **non-predicated threads** (scaled by 32)

- FMA, Tensor Cores, Mixed Precision, …
  - Rather than counting FLOPs, count **instructions**
  - Can differentiate total instruction issue bandwidth from functional unit utilization (FP32, FP64)
  - n.b., some GIPS should be summed (FP16+FP32) while others are have dedicated pipelines (FP64, TC)

Instruction Roofline Takeaway

Traditional Roofline

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

Instruction Roofline

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

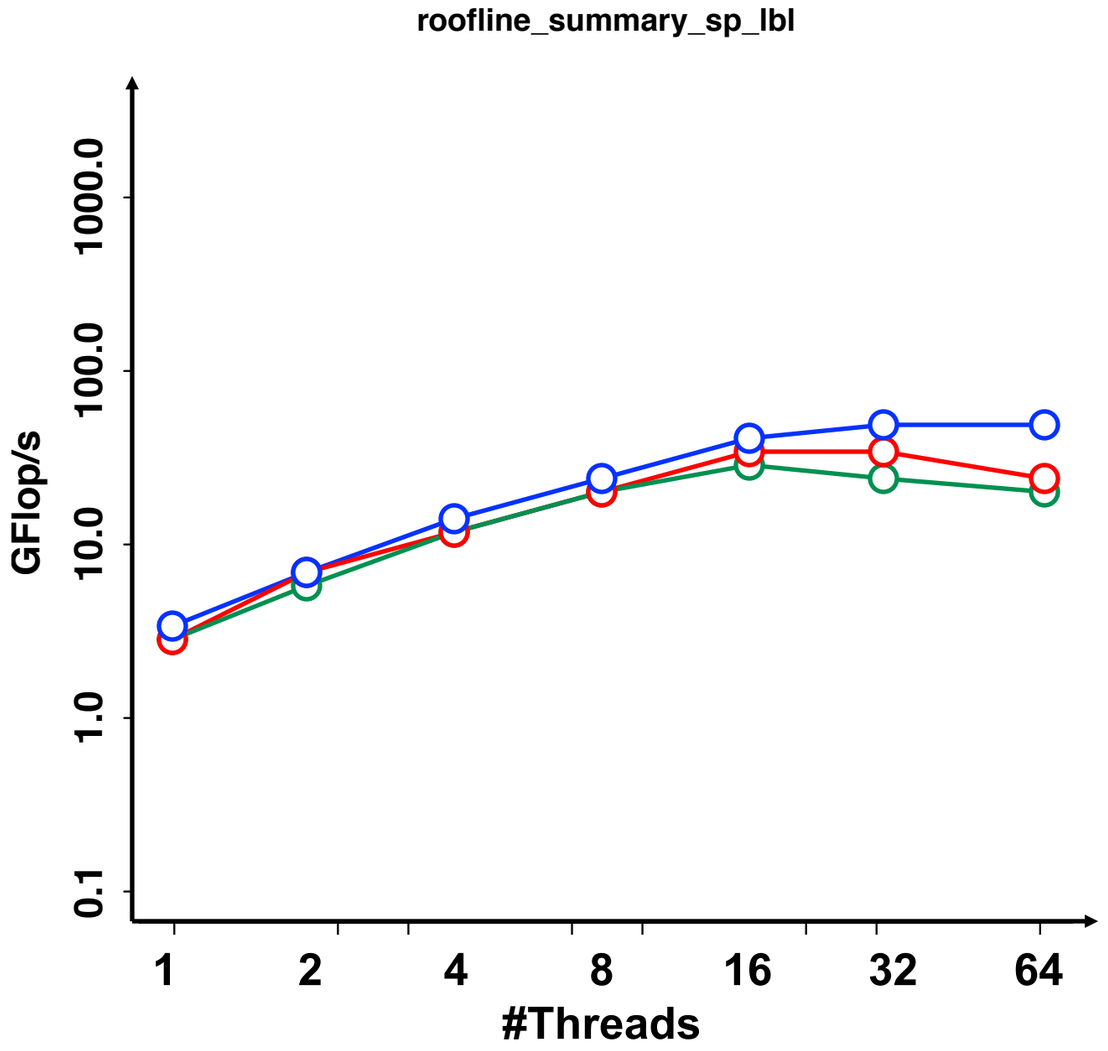
Memory Walls

- **Tells us about efficiency** *(memory access)*
- Intensity based on LDST instructions and transactions
- Predication could affect intensity (could have zero transactions for a LDST instruction, but not all LDST instructions)
- Reducing precision shifts intensity, and the unit-stride wall


Roofline Scaling
Trajectories

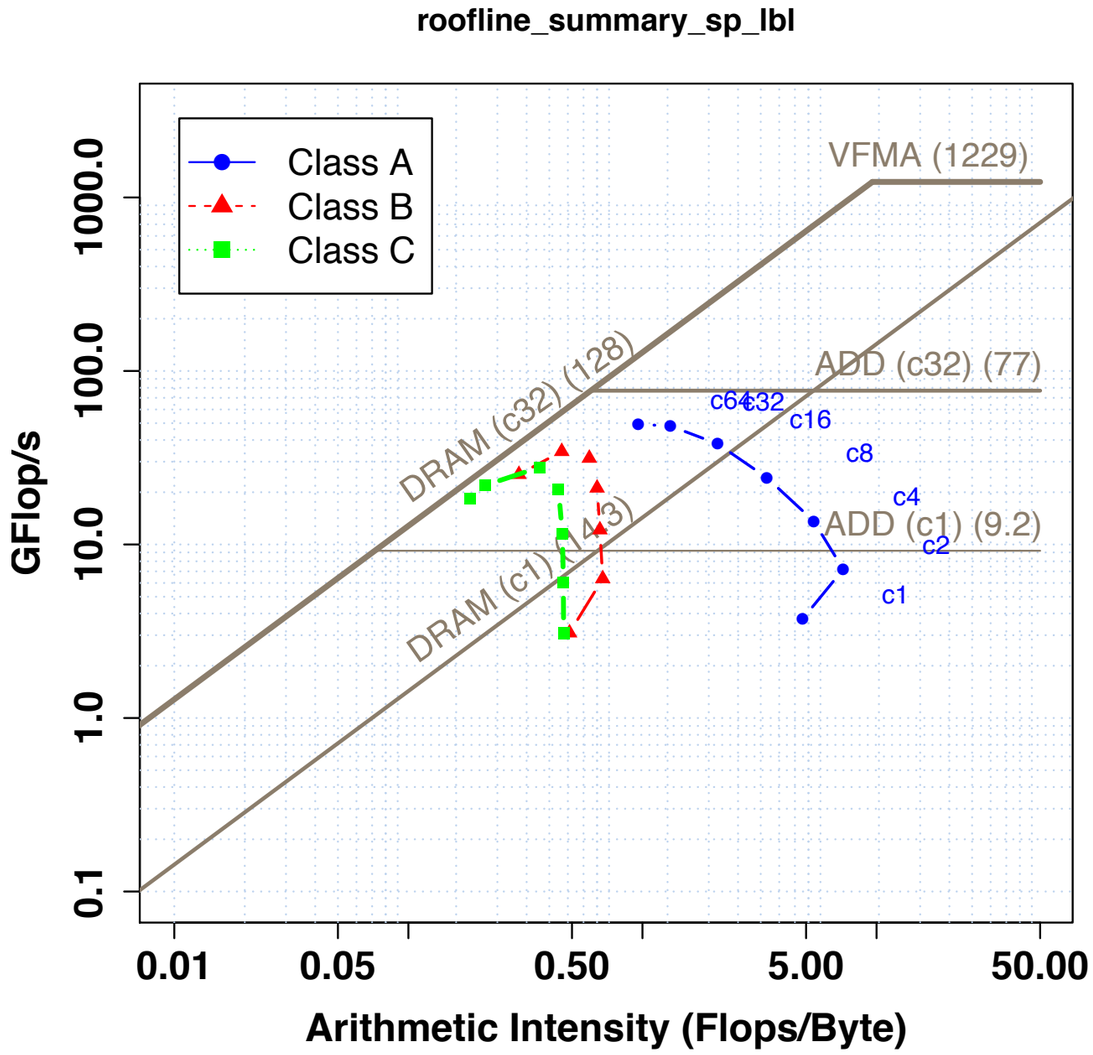
Roofline Scaling Trajectories

- We often plot performance as a function of thread concurrency
  - Carries no insight or analysis
  - Provides no actionable information.



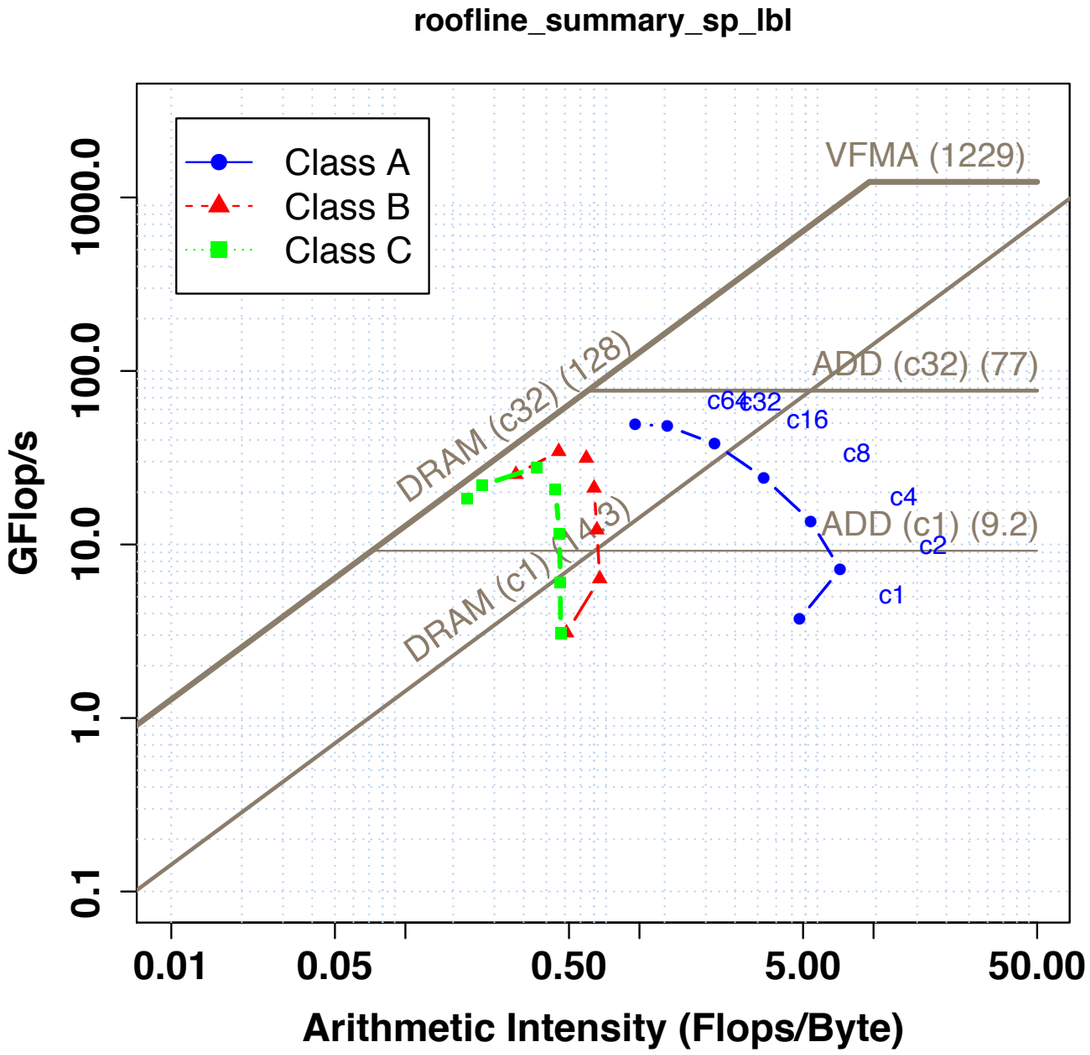
Roofline Scaling Trajectories

- We often plot performance as a function of thread concurrency
  - Carries no insight or analysis
  - Provides no actionable information.
- Use Roofline to analyze thread (or process) scalability
  - 2D scatter plot of performance as a function of intensity and concurrency
  - Identify loss in performance due to increased cache pressure (data movement)



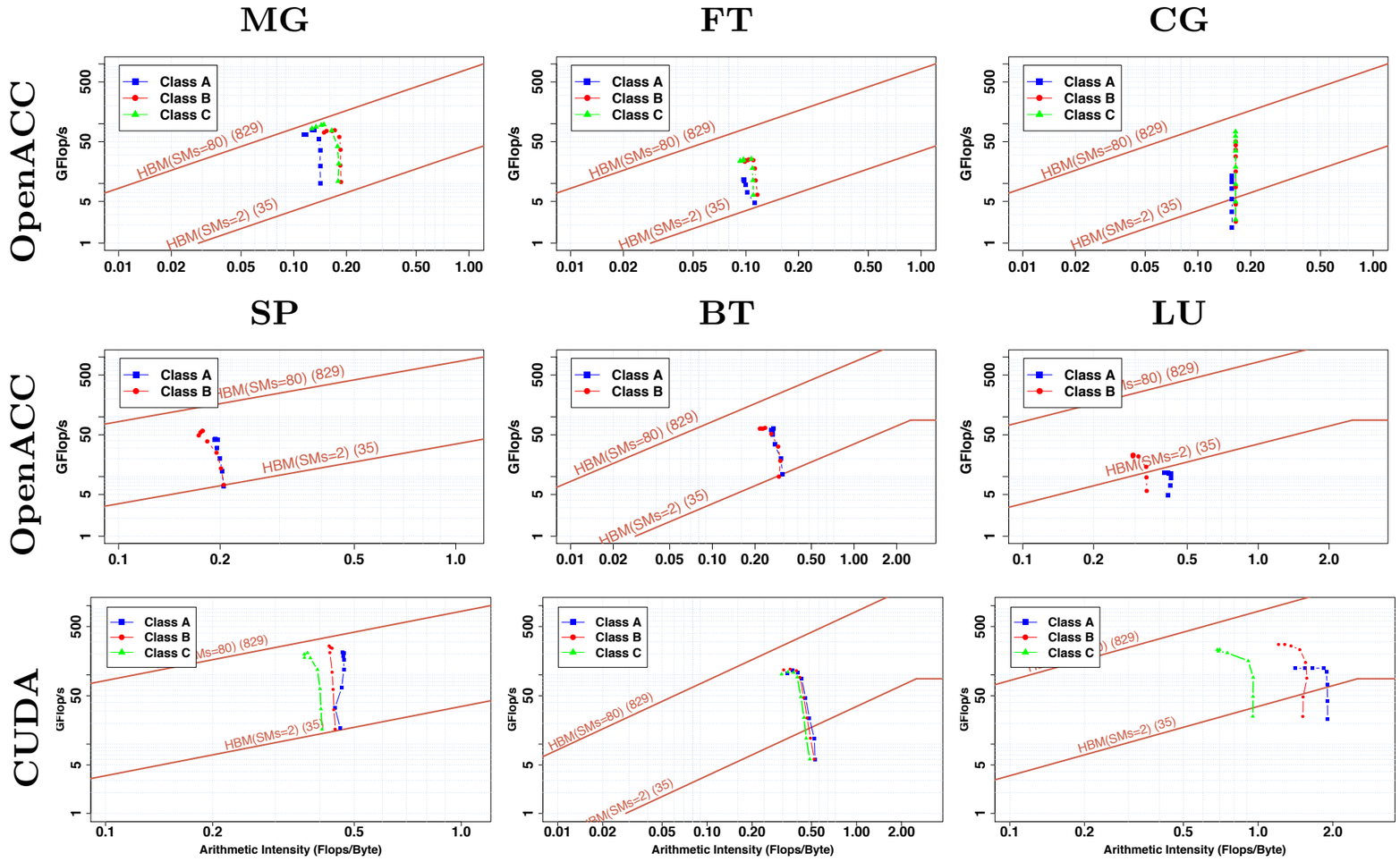
Roofline Scaling Trajectories

- Insights from NPB
  - Intensity (data movement) varies with concurrency and problem size
  - Large problems (green and red) move more data per thread, and exhaust cache capacity
  - Falling Intensity → hit the bandwidth ceiling quickly and degrade.
  - ➢ **Useful for understanding locality/BW contention induced scaling bottlenecks**



Roofline Scaling Trajectories on GPUs

- More Recently applied to GPUs (SM scaling)
- Allows comparisons of programming models (OpenACC vs. CUDA)
- Strong differences in scalability and performance per thread(block).



Khaled Ibrahim, Samuel Williams, Leonid Oliker, ”Performance Analysis of GPU Programming Models Using the Roofline Scaling Trajectories", (BEST PAPER) Bench, 2019.


Recap

Roofline Recap

Roofline bounds performance as a function of Arithmetic Intensity

- Horizontal Lines = Compute Ceilings
- Diagonal Lines = Bandwidth Ceilings
- Bandwidth ceilings are parallel on log-log scale
- ➢ **Collectively, ceilings define an upper limit on performance**

Arithmetic Intensity

- Unique for each loop nest
- Unique for each level of memory
- Total FLOPs / Total Bytes
- Includes **all** cache effects
- Different on every architecture
- ➢ **Measure of a loop's temporal locality**

Plotting loops on the 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**
- ➢ **Position of dots relative to each other is indicative of cache locality**

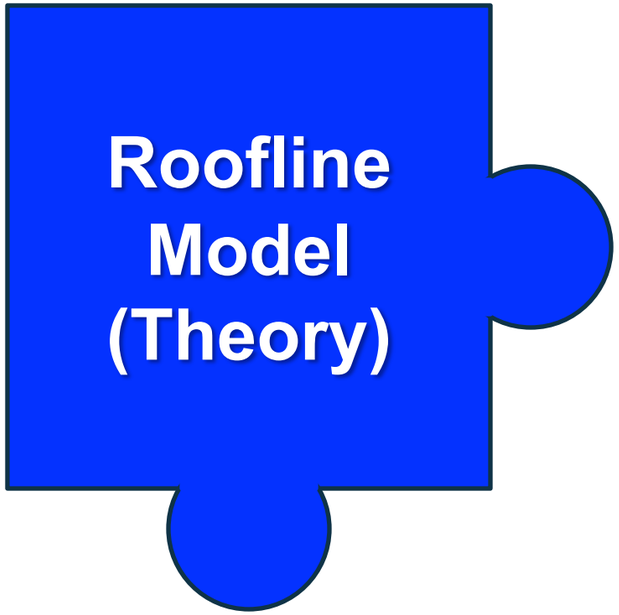
Applies equally to GPUs and other accelerators

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

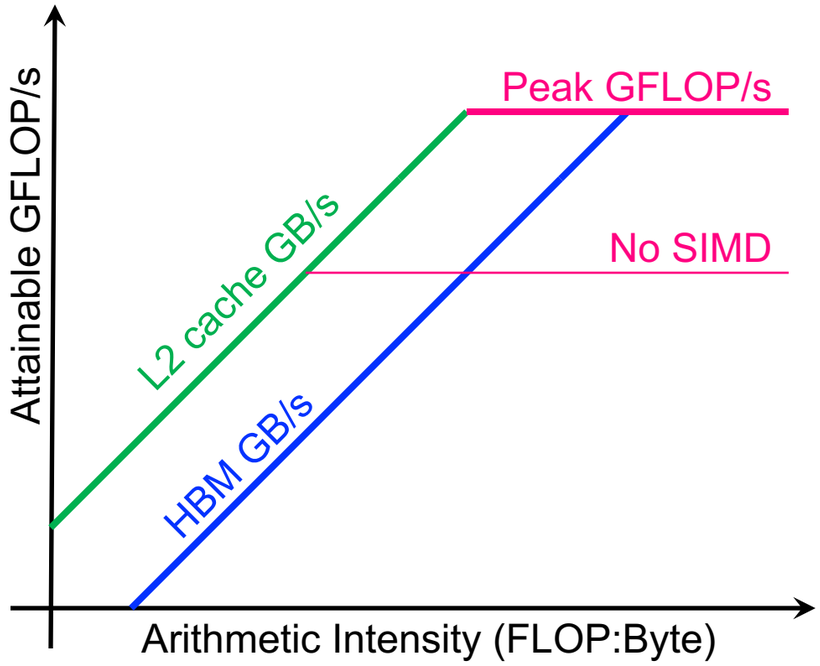
Model is just one piece of the puzzle…

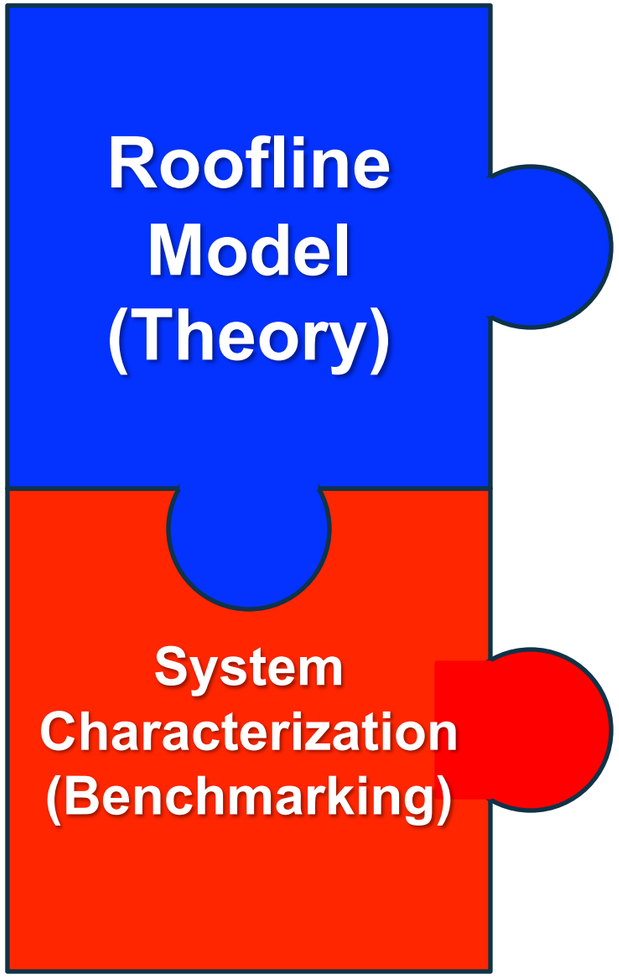
- Roofline Model defines the basic concepts and equations.



Model is just one piece of the puzzle…

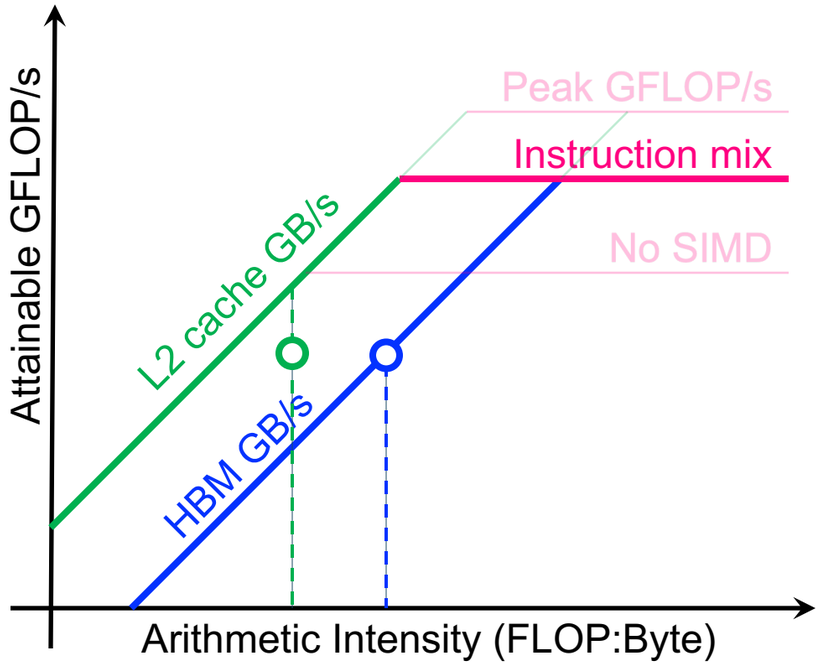
- System Characterization defines the shape of the Roofline (peak bandwidths and FLOP/s)

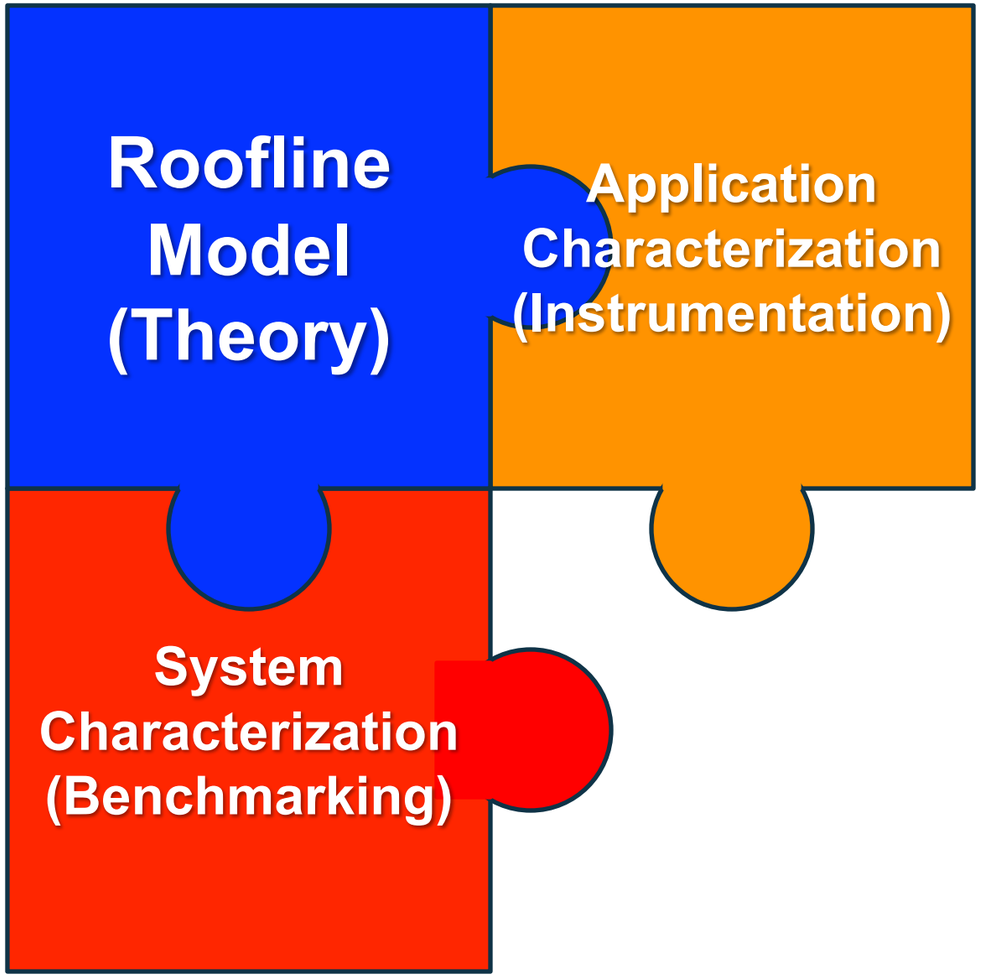




Model is just one piece of the puzzle…

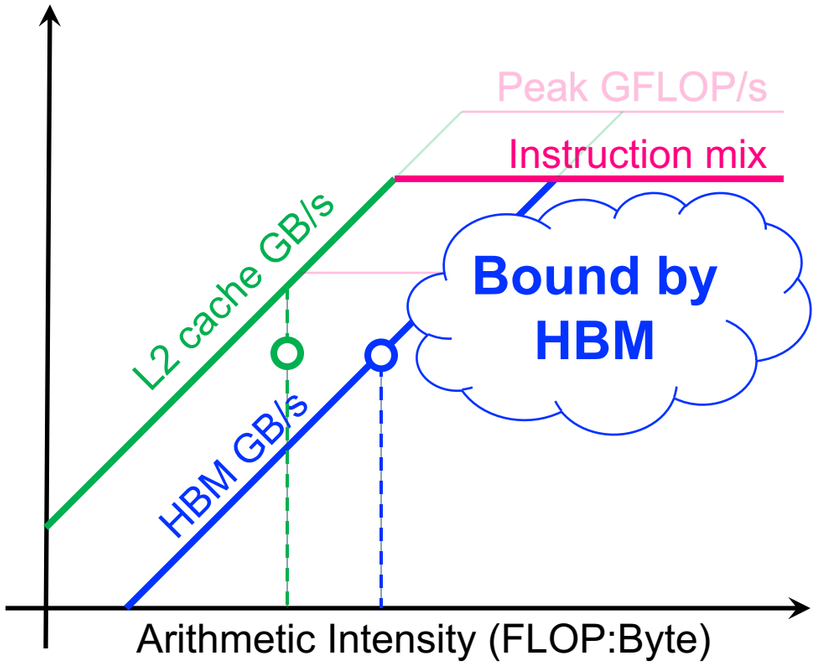
- Application Characterization determines…
  - Intensity and Performance of each loop
  - Position of any implicit ceilings

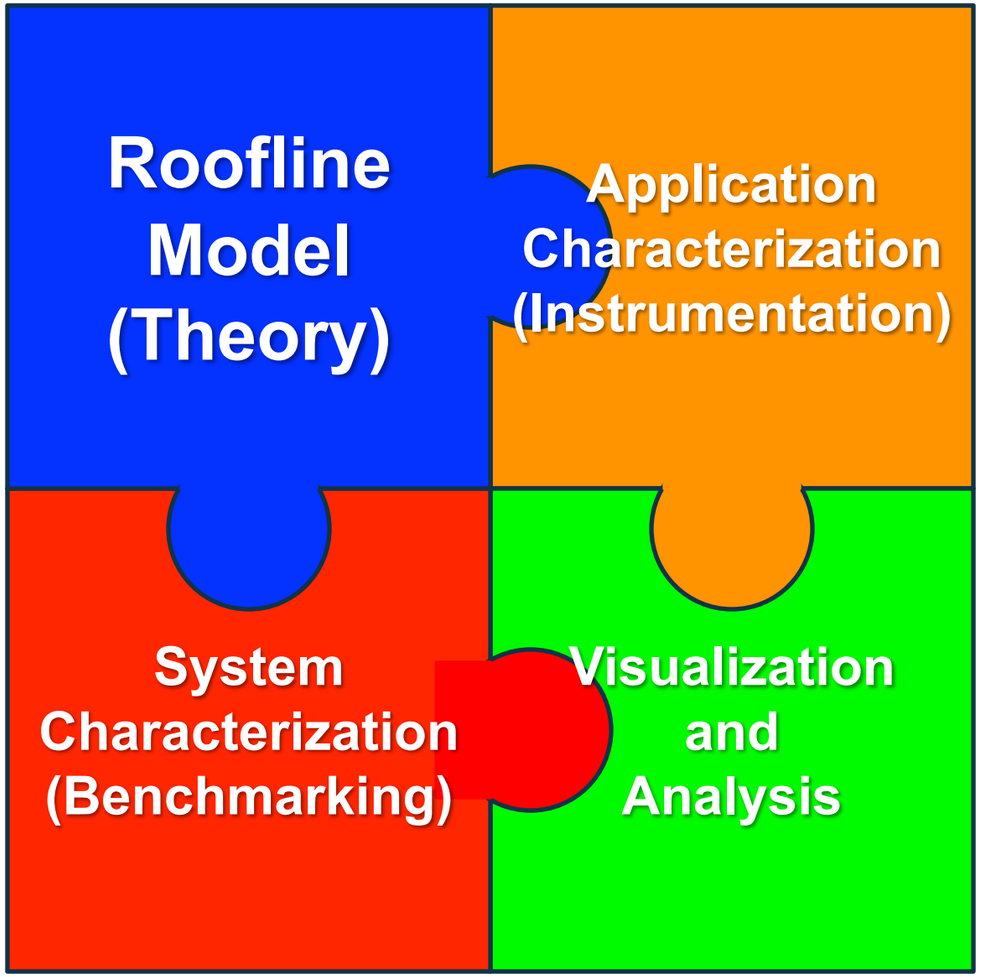




Model is just one piece of the puzzle…

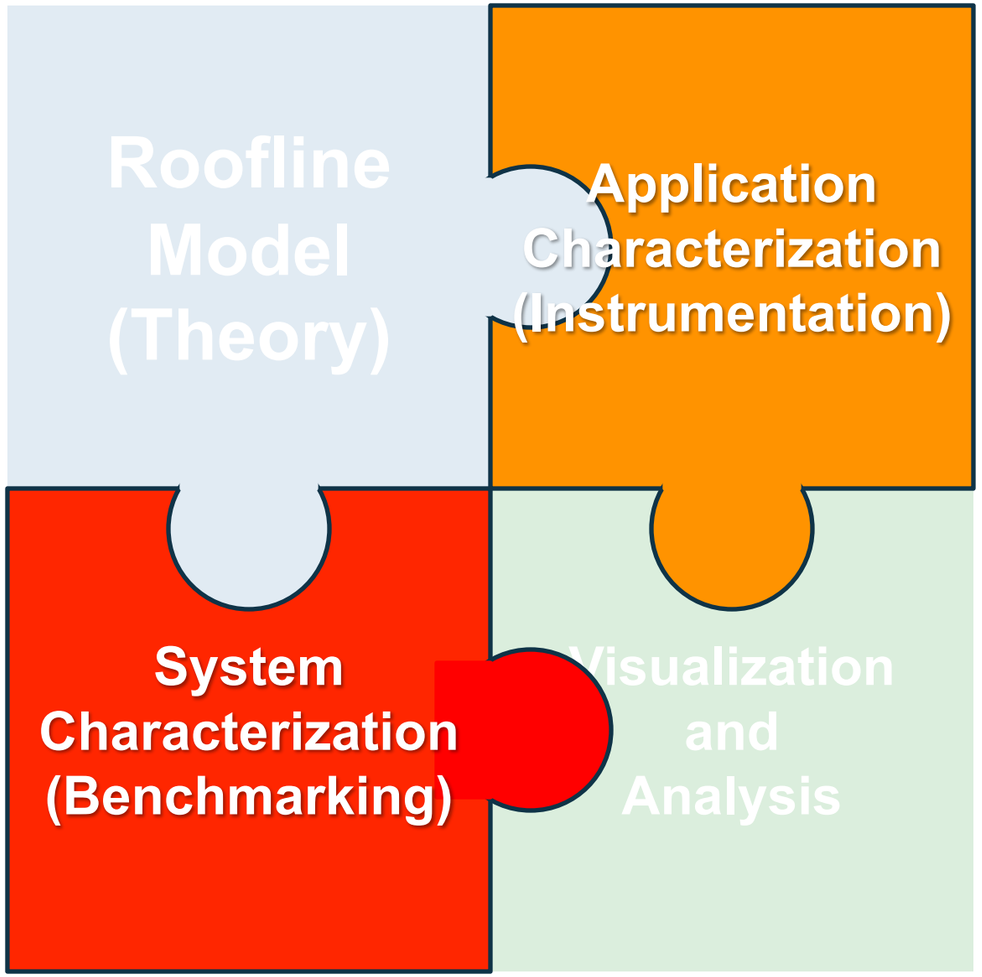
- Visualization tools combine all data together and provide analytical capability





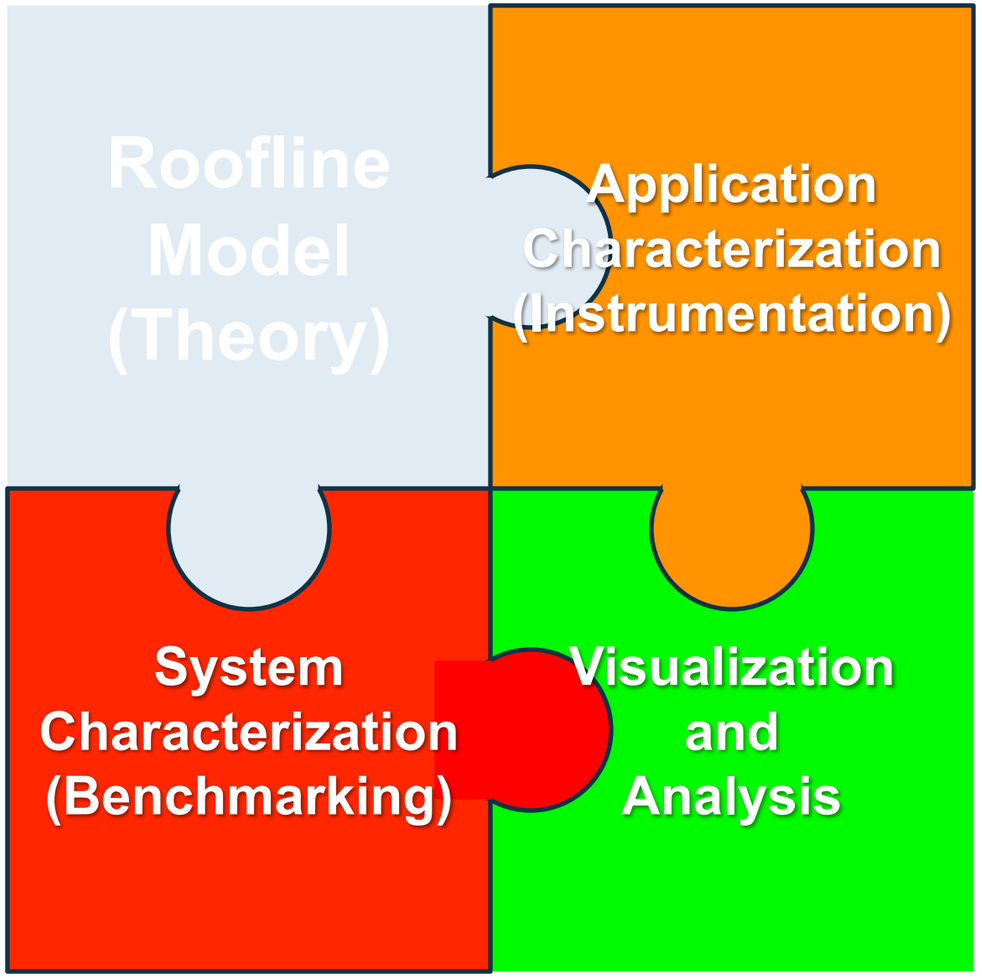
Rest of Tutorial…

- Charlene will demonstrate how to construct and use Roofline model on an NVIDIA GPU
  - GPU benchmarking
  - application characterization



Rest of Tutorial…

- Charlene will demonstrate how to construct and use Roofline model on an NVIDIA GPU
  - GPU benchmarking
  - application characterization
- Max will demonstrate how Nsight Compute now automates Roofline
  - GPU benchmarking
  - application characterization
  - Visualization
- You will use Roofline in Nsight Compute to analyze your apps.



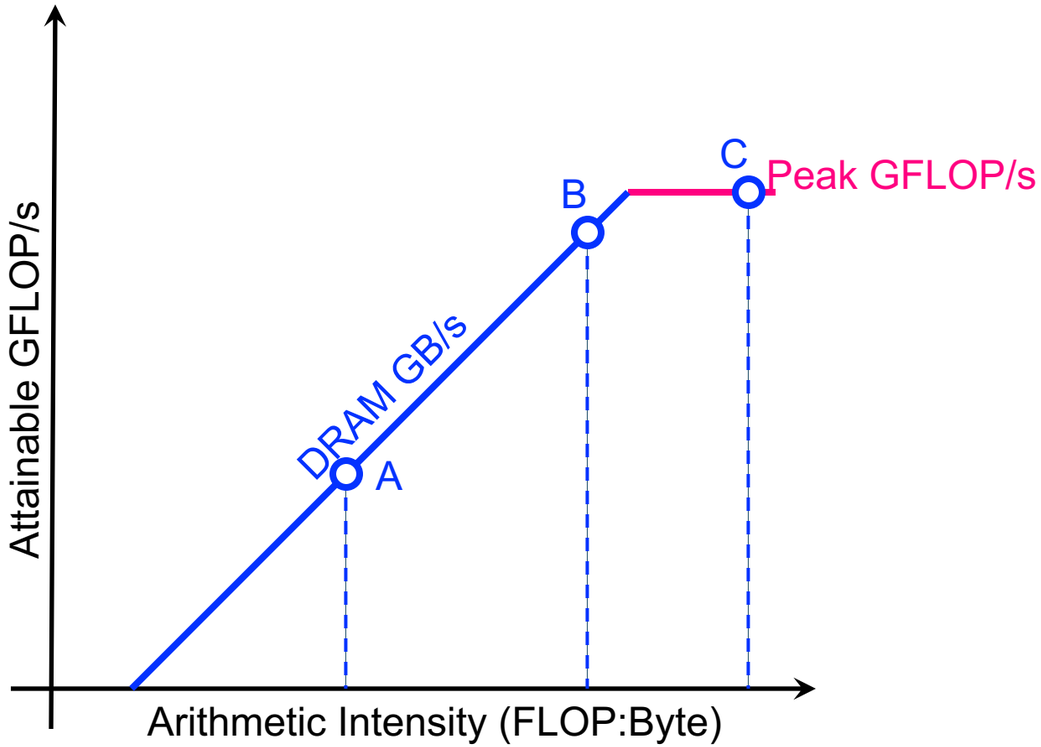

Questions?


BACKUP


Performance
Extrapolations

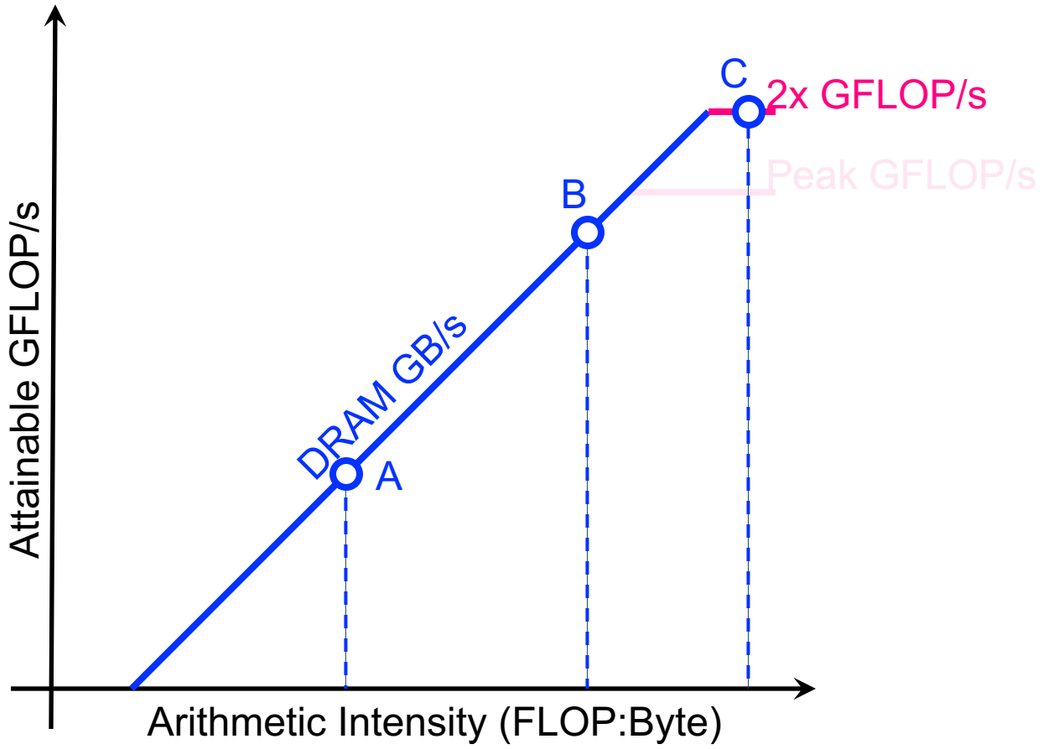
Setting Realistic Expectations…

- Consider 3 kernels (A,B,C)
  - **kernels A and B are bound by memory bandwidth**
  - **kernel C is bound by peak FLOP/s**



Setting Realistic Expectations…

- Imagine you want to run on a machine with twice the peak FLOPs…
  - ➢ **kernel C's performance could double**
  - ✗ **kernels A and B will be no faster**



Setting Realistic Expectations…

- What if that machine also doubled memory bandwidth…
  - **kernel A and B's performance could also double**