Roofline Performance Modeling for HPC and Deep Learning Applications

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

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

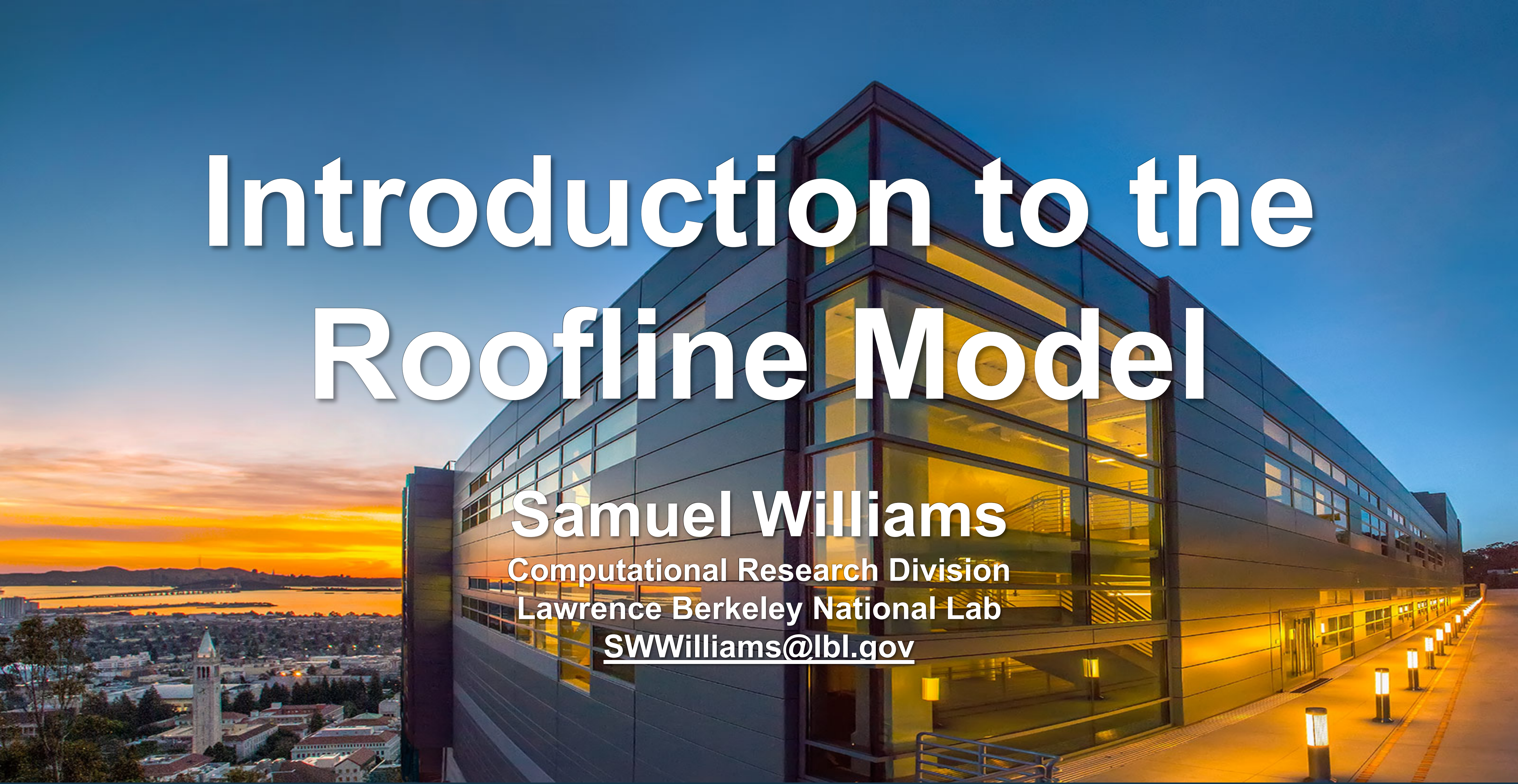
Yunsong Wang
NERSC
Lawrence Berkeley National Lab
yunsongwang@lbl.govv

Schedule

9:00am	Introduction to Roofline	Samuel Williams
9:30am	NVProf Methodology/demo	Yunsong Wang
10:00am	Roofline Use Cases	Charlene Yang

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.



Introduction to the Roofline Model

Samuel Williams

Computational Research Division

Lawrence Berkeley National Lab

SWWilliams@lbl.gov

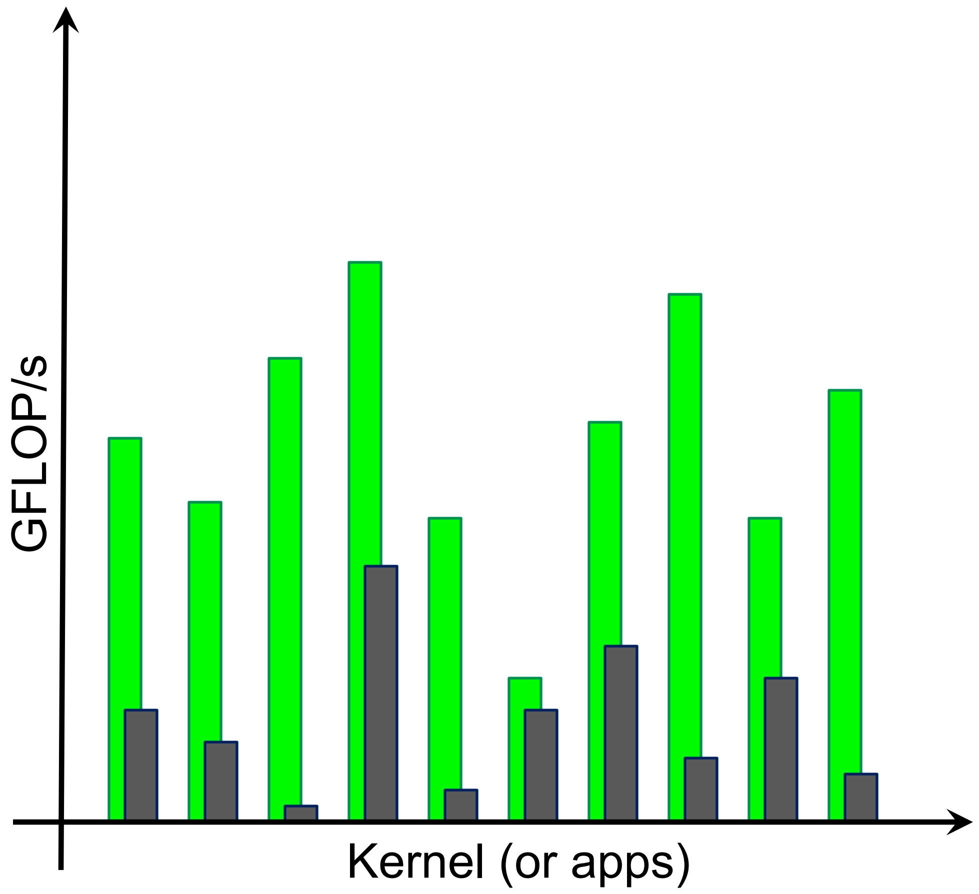
You just bought a $10,000 throughput-optimized GPU!

Are you making good use of your investment?

You could just run benchmarks

- Imagine running a mix of benchmarks or kernels…
- GFLOP/s alone may not be particularly insightful
- speedup relative to a Xeon may seem random

➢ **We need a quantitative model that defines Good Performance**



BERKELEY LAB

What is "Good" Performance?

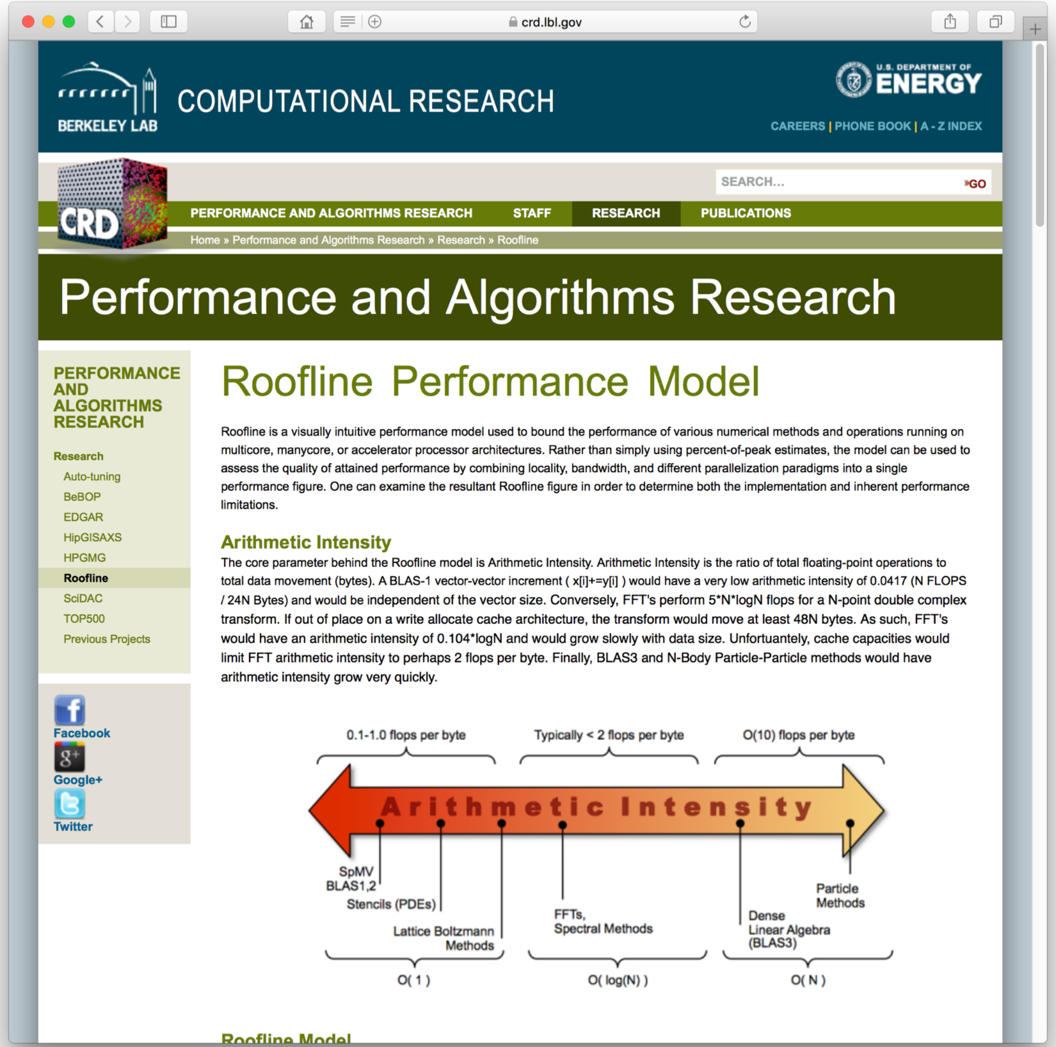
- Good Performance is tied to "Efficient" execution
- Two fundamental requirements …

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

2. Must attain high utilization of the GPU's **compute** and/or **bandwidth** capabilities

Roofline Model

- **Roofline Model** is a throughput-oriented performance model
- applies to x86, ARM, POWER CPUs, GPUs, Google TPUs[1], FPGAs, etc…
- Helps quantify **Good Performance**



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

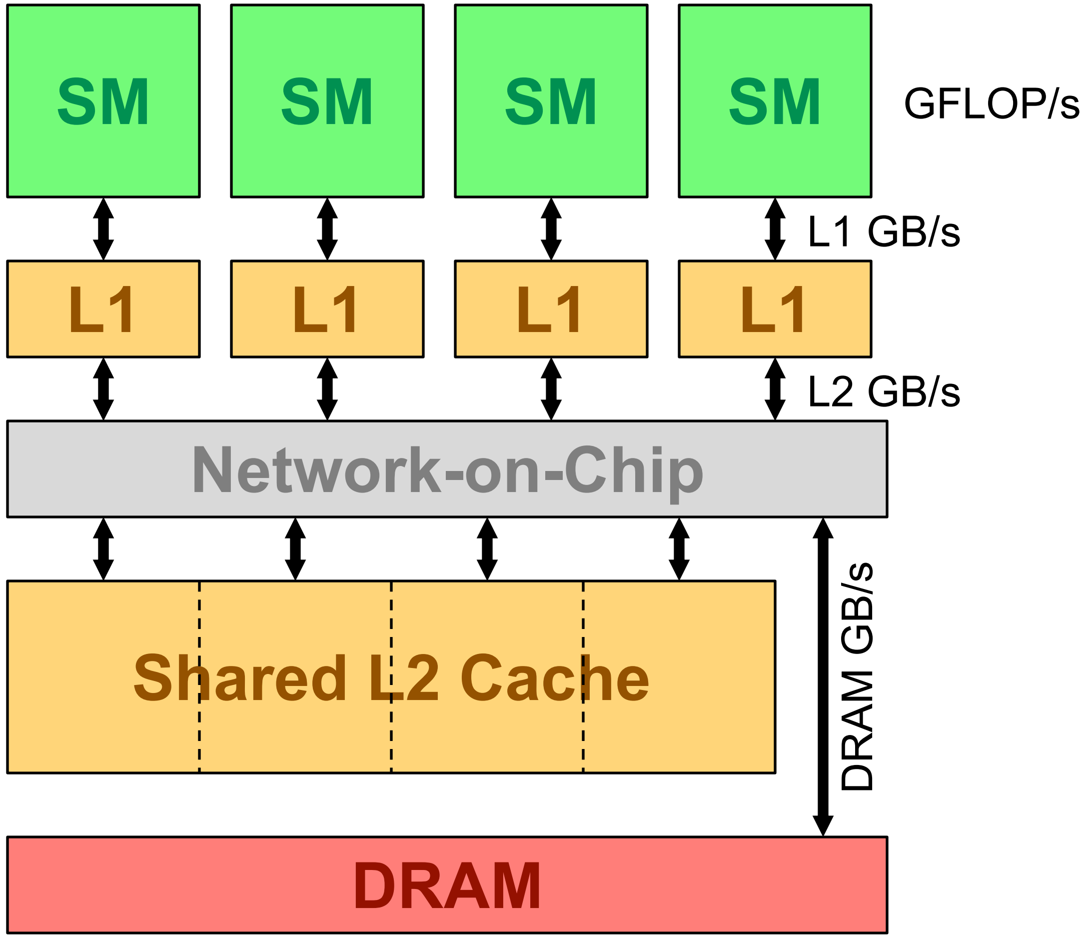
[1]Jouppi et al, "In-Datacenter Performance Analysis of a Tensor Processing Unit", ISCA, 2017.

Reduced Model

- GPU architectures can be complex
- Don’t model / simulate full architecture
- Make assumptions on performance and usage…

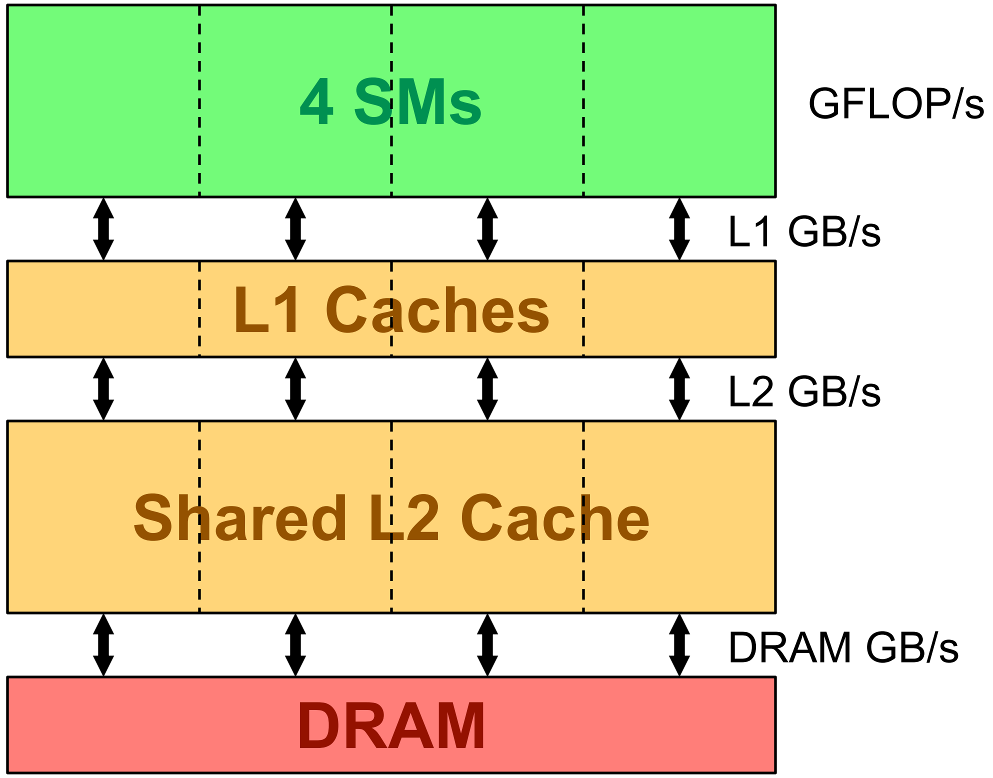
Reduced Model

- GPU architectures can be complex
- Don’t model / simulate full architecture
- Make assumptions on performance and usage…
  - Peak GFLOP/s on data in L1



Reduced Model

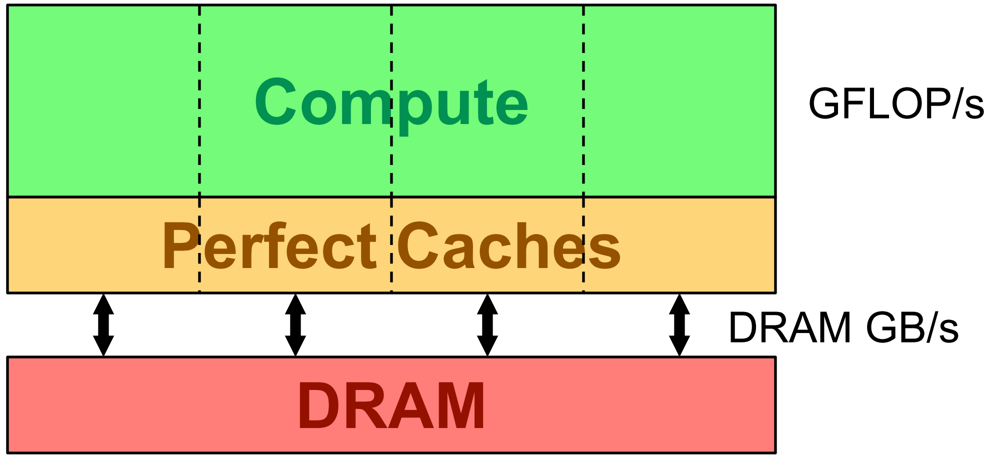
- GPU architectures can be complex
- Don't model / simulate full architecture
- Make assumptions on performance and usage…
  - Peak GFLOP/s on data in L1
  - Load-balanced SPMD code



BERKELEY LAB

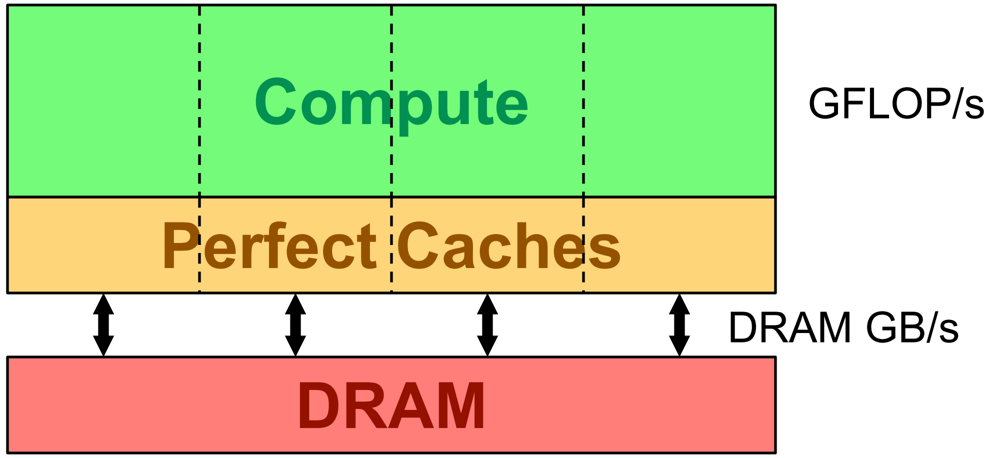
Reduced Model

- GPU architectures can be complex
- Don't model / simulate full architecture
- Make assumptions on performance and usage…
  - Peak GFLOP/s on data in L1
  - Load-balanced SPMD code
  - Sufficient cache bandwidth/capacity



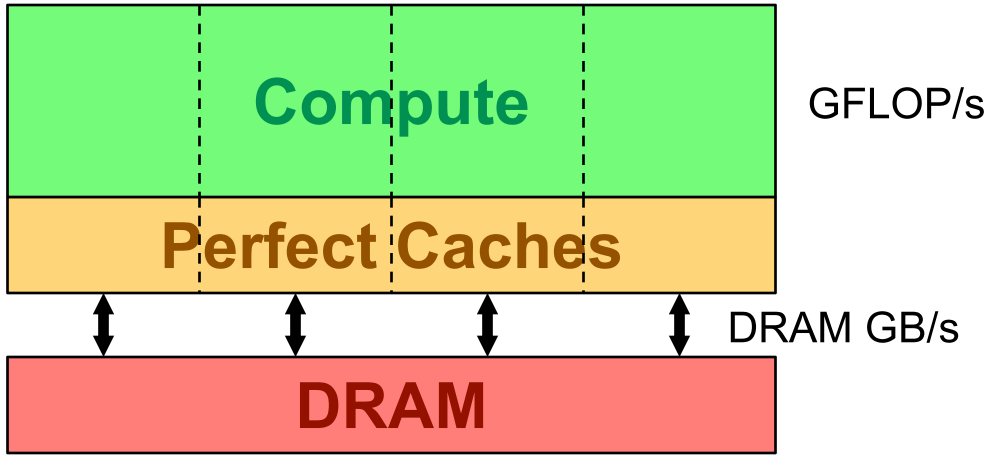
Reduced Model

- GPU architectures can be complex
- Don't model / simulate full architecture
- Make assumptions on performance and usage…
  - Peak GFLOP/s on data in L1
  - Load-balanced SPMD code
  - Sufficient cache bandwidth/capacity
  - ➢ **Basis for DRAM Roofline Model**



(DRAM) Roofline

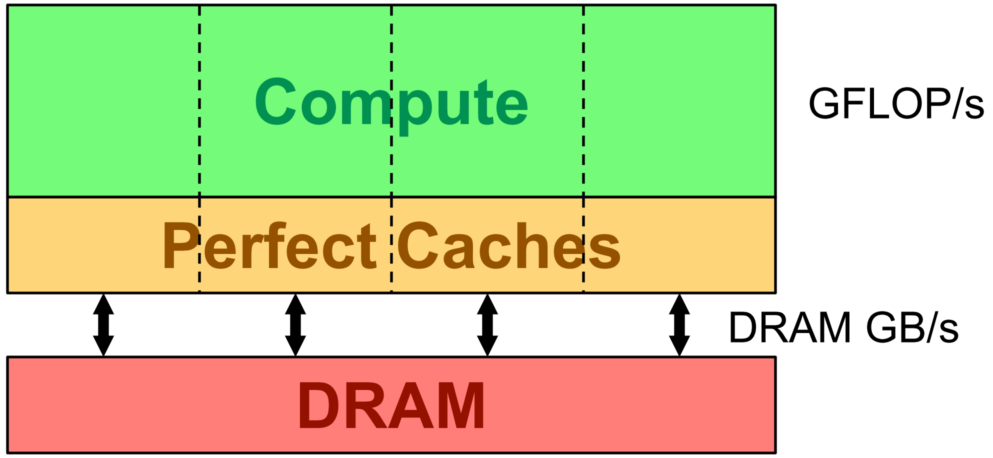
- Any given loop nest will perform:
  - Computation (e.g. FLOPs)
  - Communication (e.g. moving data to/from DRAM)
- With perfect overlap of communication and computation...
  - Run time is determined by whichever is greater



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

(DRAM) Roofline

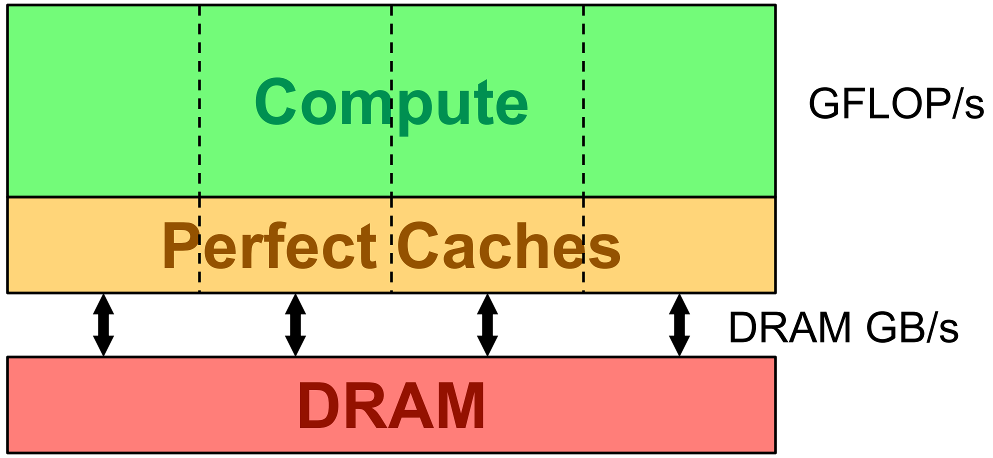
- Any given loop nest will perform:
  - Computation (e.g. FLOPs)
  - Communication (e.g. moving data to/from DRAM)
- With perfect overlap of communication and computation...
  - Run time is determined by whichever is greater



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

(DRAM) Roofline

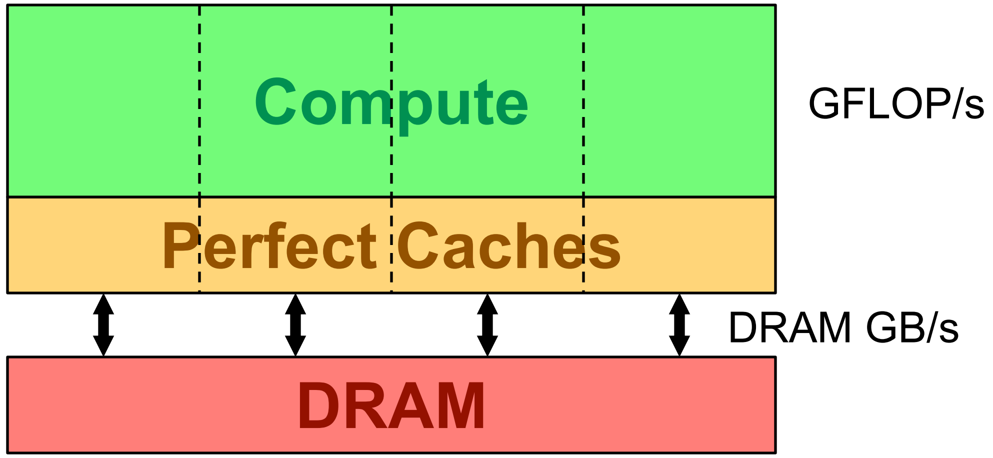
- Any given loop nest will perform:
  - Computation (e.g. FLOPs)
  - Communication (e.g. moving data to/from DRAM)
- With perfect overlap of communication and computation...
  - Run time is determined by whichever is greater



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

(DRAM) Roofline

- Any given loop nest will perform:
  - Computation (e.g. FLOPs)
  - Communication (e.g. moving data to/from DRAM)
- With perfect overlap of communication and computation...
  - Run time is determined by whichever is greater



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

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

Arithmetic Intensity

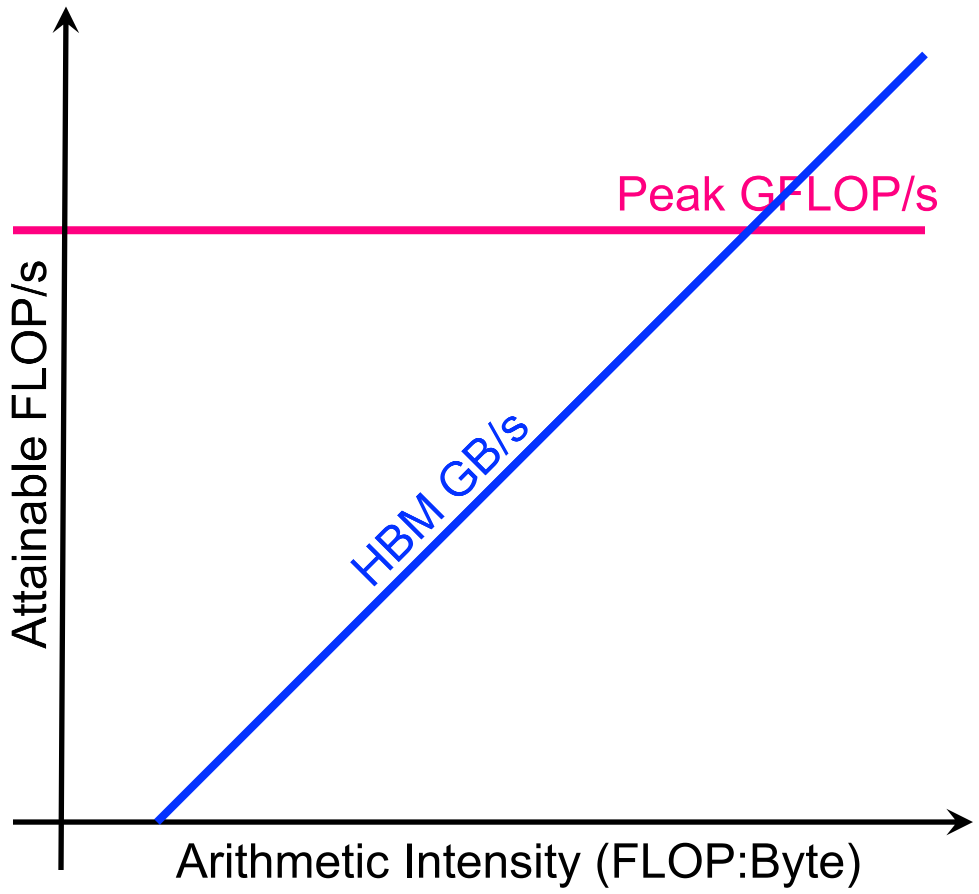
- Measure of data locality (data reuse)
- Ratio of **Total Flops** performed to **Total Bytes** moved
- For the DRAM Roofline…
  - Total Bytes to/from DRAM
  - Includes all cache and prefetcher effects
  - Can be very different from total loads/stores (bytes requested)
  - Equal to ratio of sustained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

(DRAM) Roofline Model

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

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…

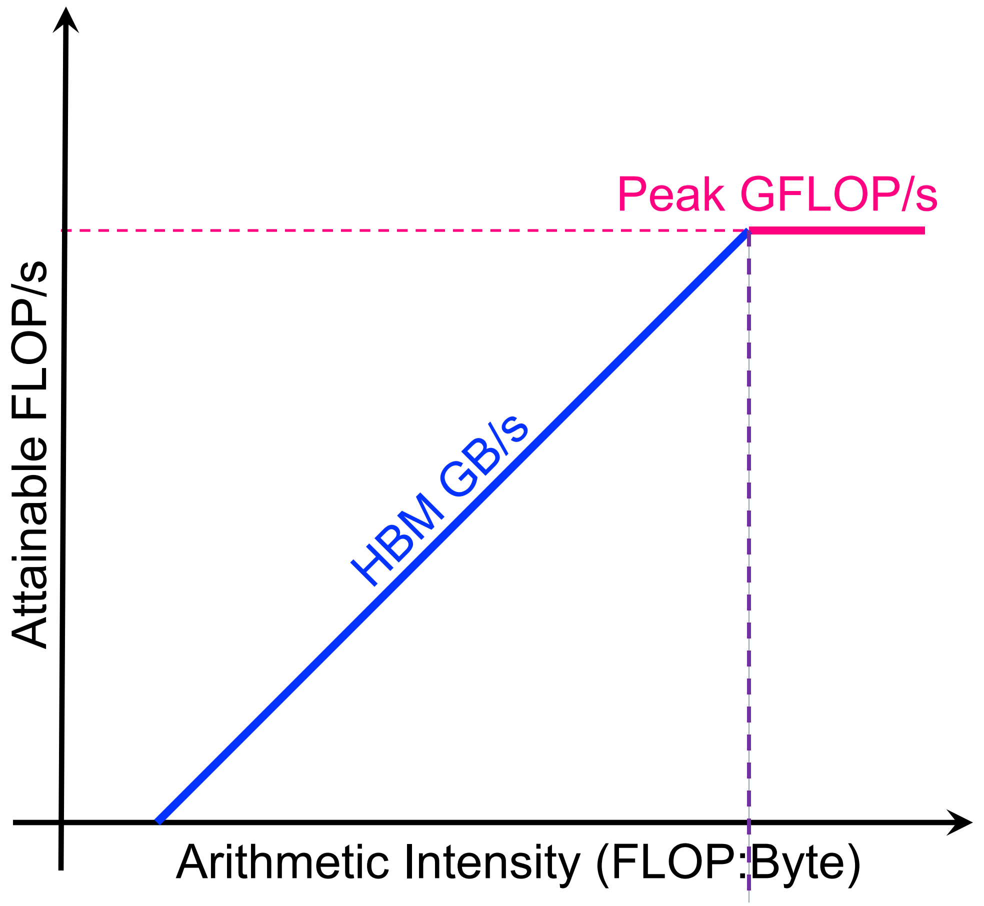


(DRAM) Roofline Model

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

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…



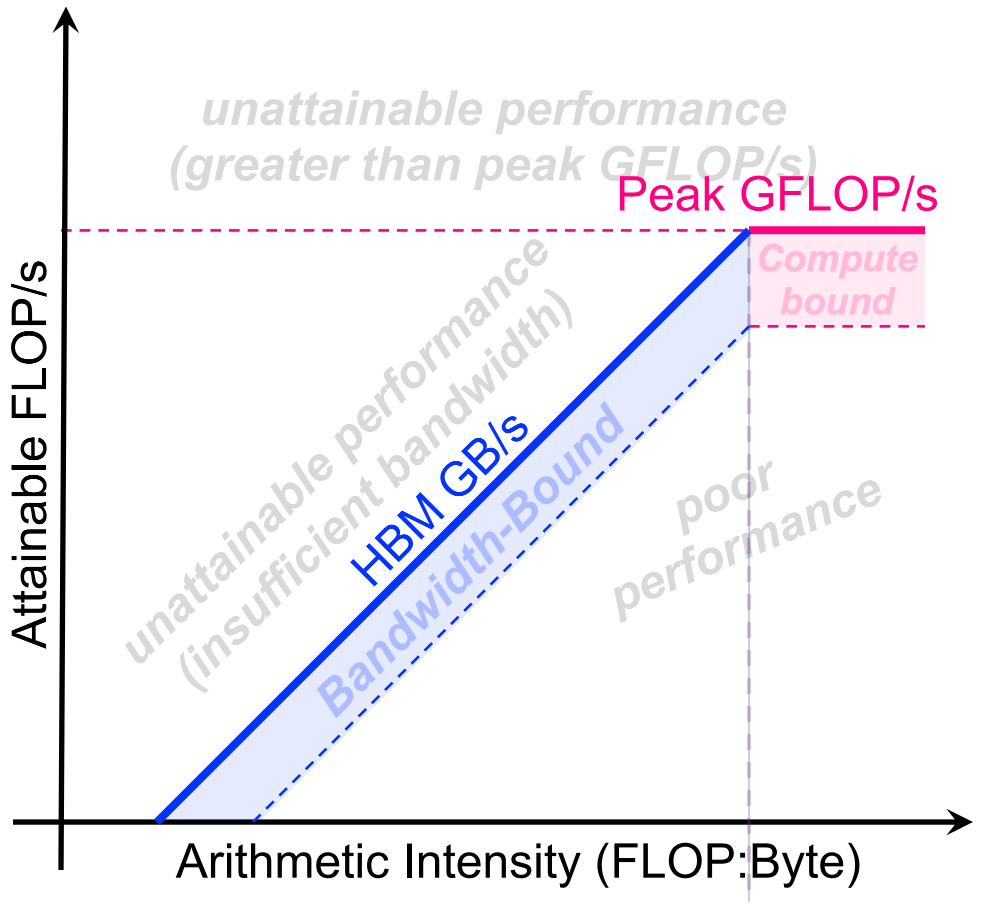
Transition @ AI ==
Peak GFLOP/s / Peak GB/s ==
'Machine Balance'

(DRAM) Roofline Model

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

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'

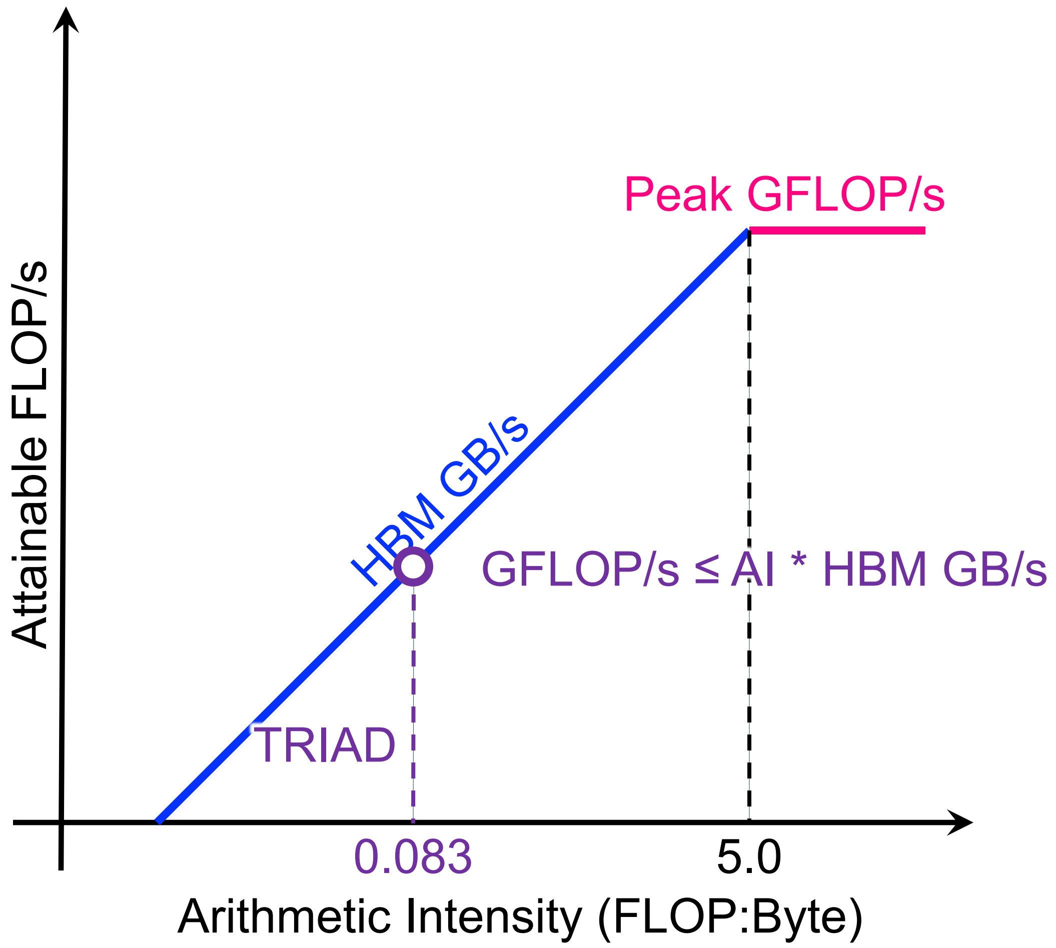
Roofline Example #1

- 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…

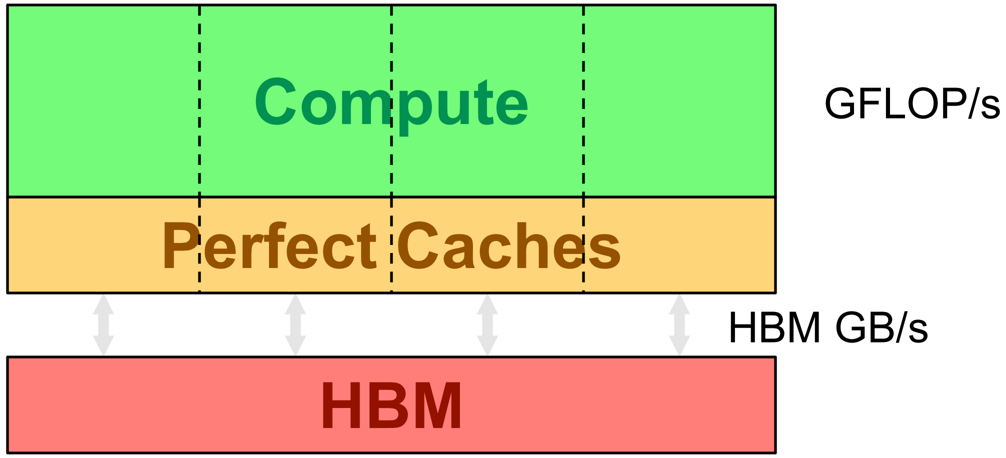
```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

  - 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

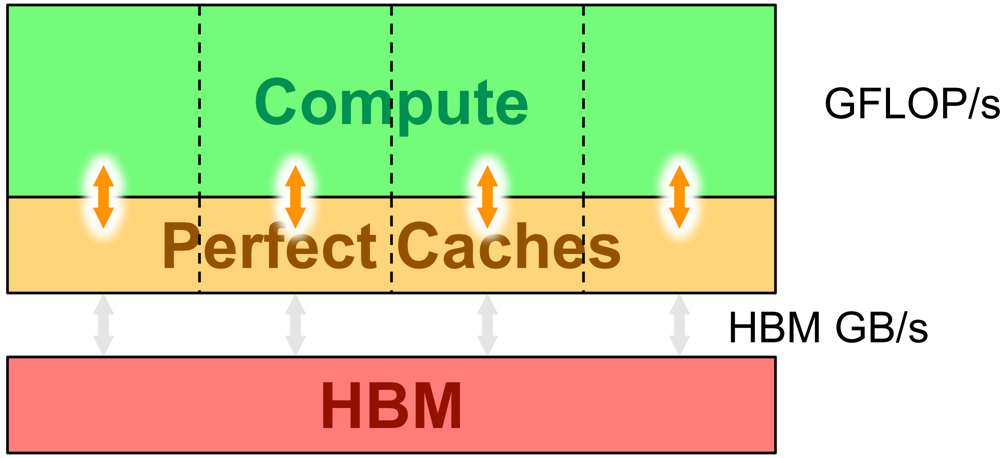
- Conversely, 7-point constant coefficient stencil…




```
#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)

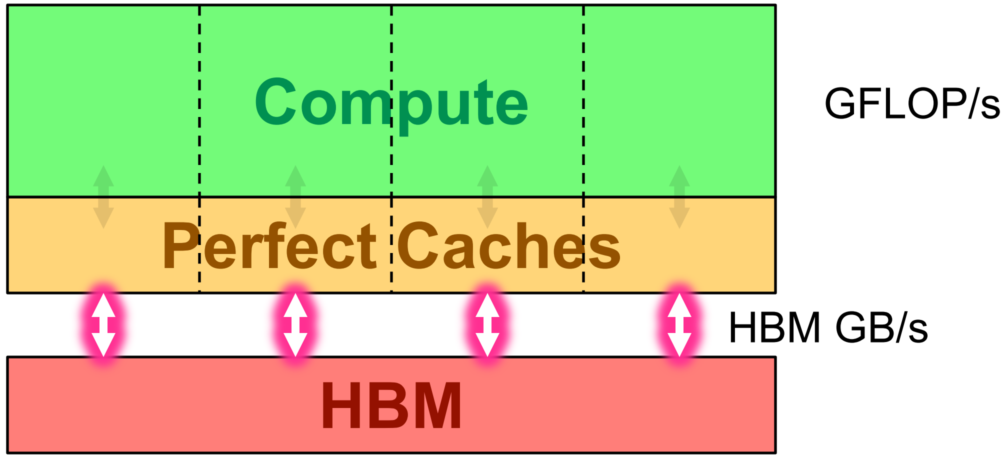



```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                    + old[k  ][j  ][i-1]
                    + old[k  ][j  ][i+1]
                    + old[k  ][j-1][i  ]
                    + old[k  ][j+1][i  ]
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


BERKELEY LAB

Roofline Example #2

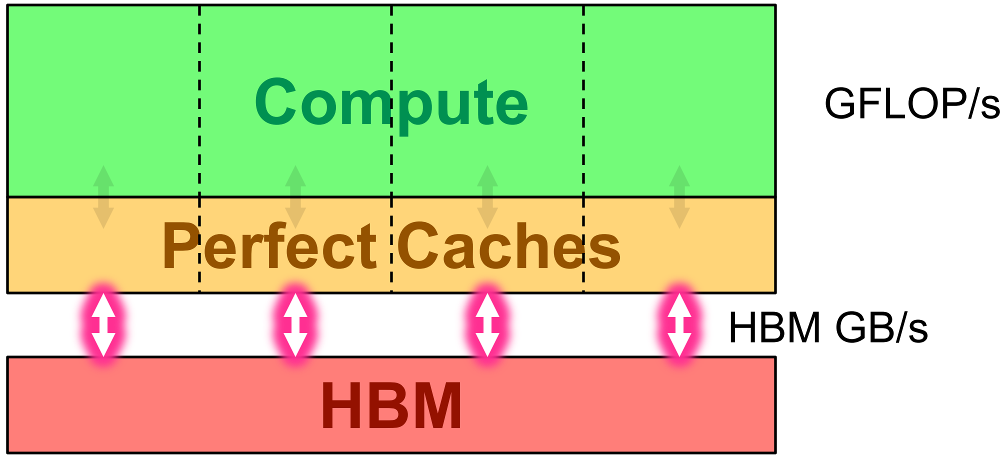
- Conversely, 7-point constant coefficient stencil…
  - 7 FLOPs
  - 8 memory references (7 reads, 1 store) per point
  - **Ideally, cache will filter all but 1 read and 1 write per point**




```
#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)**



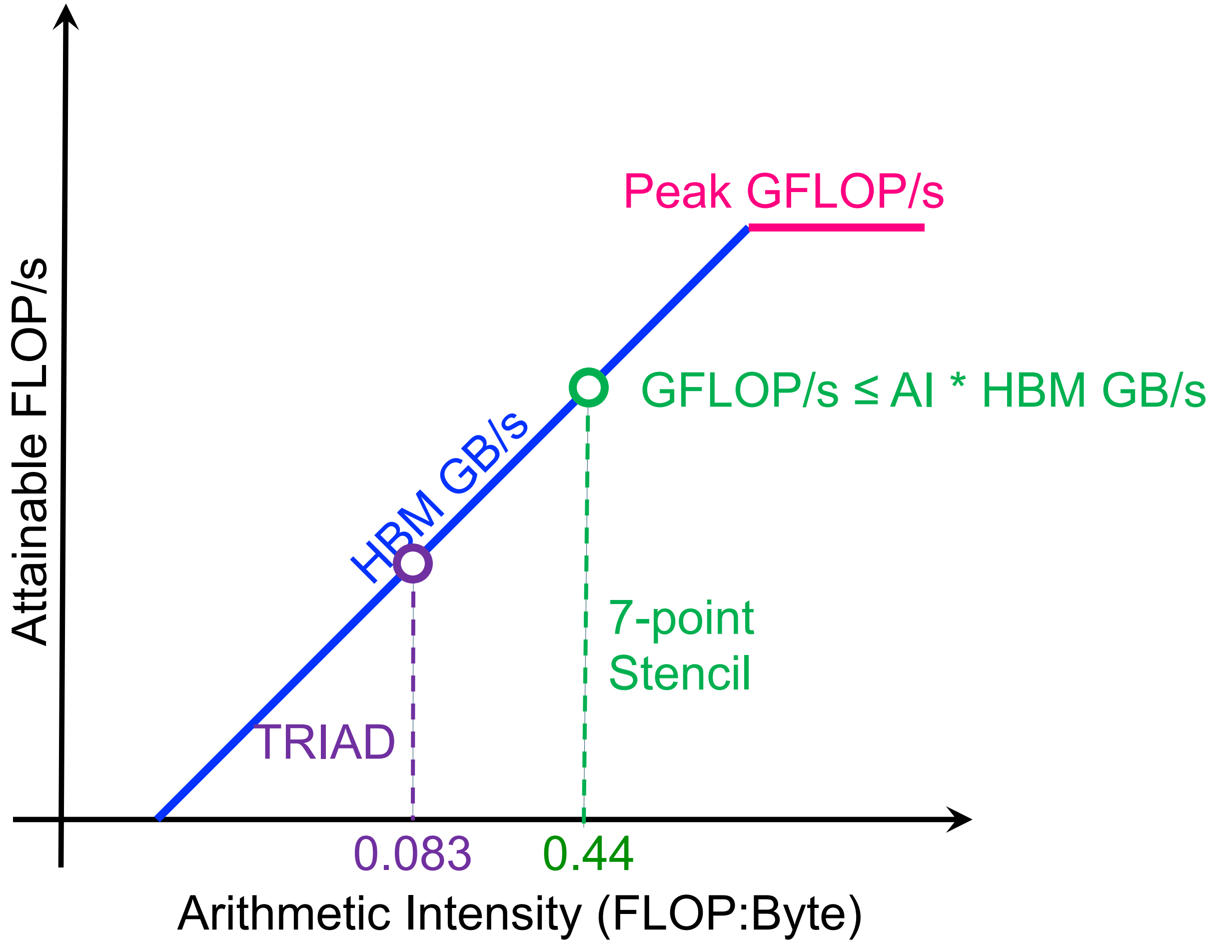

```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                    + old[k  ][j  ][i-1]
                    + old[k  ][j  ][i+1]
                    + old[k  ][j-1][i  ]
                    + old[k  ][j+1][i  ]
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

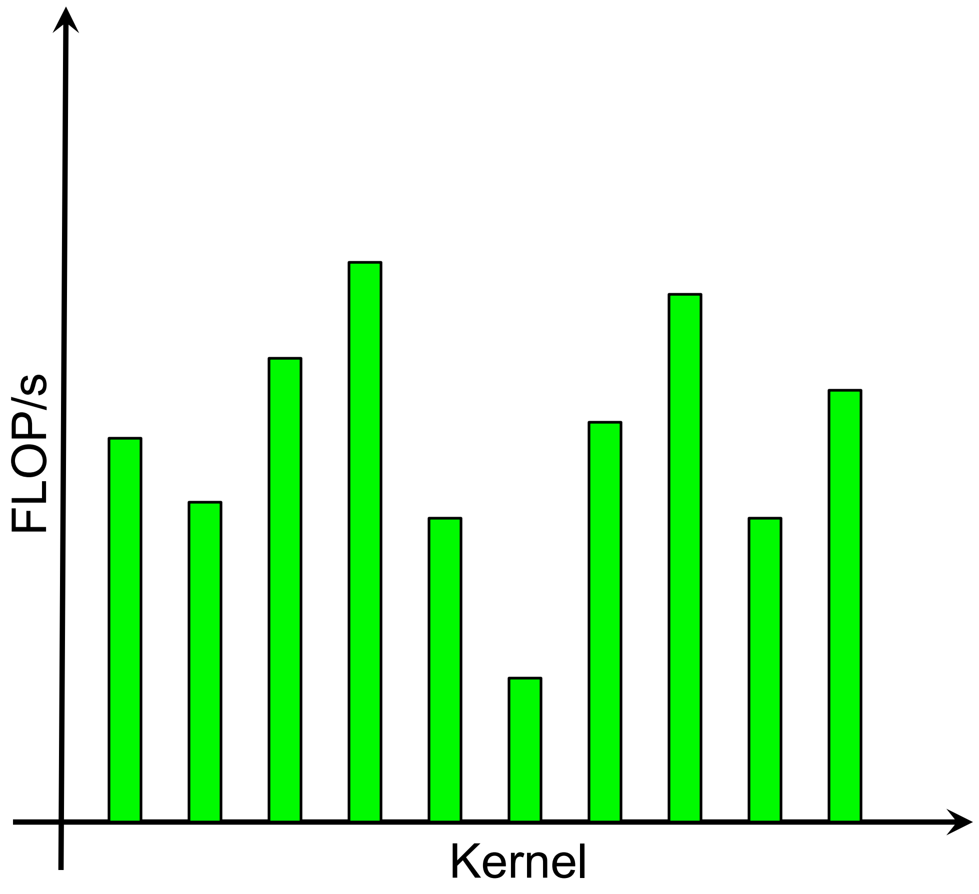
    == memory bound, but 5x the FLOP rate as TRIAD

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



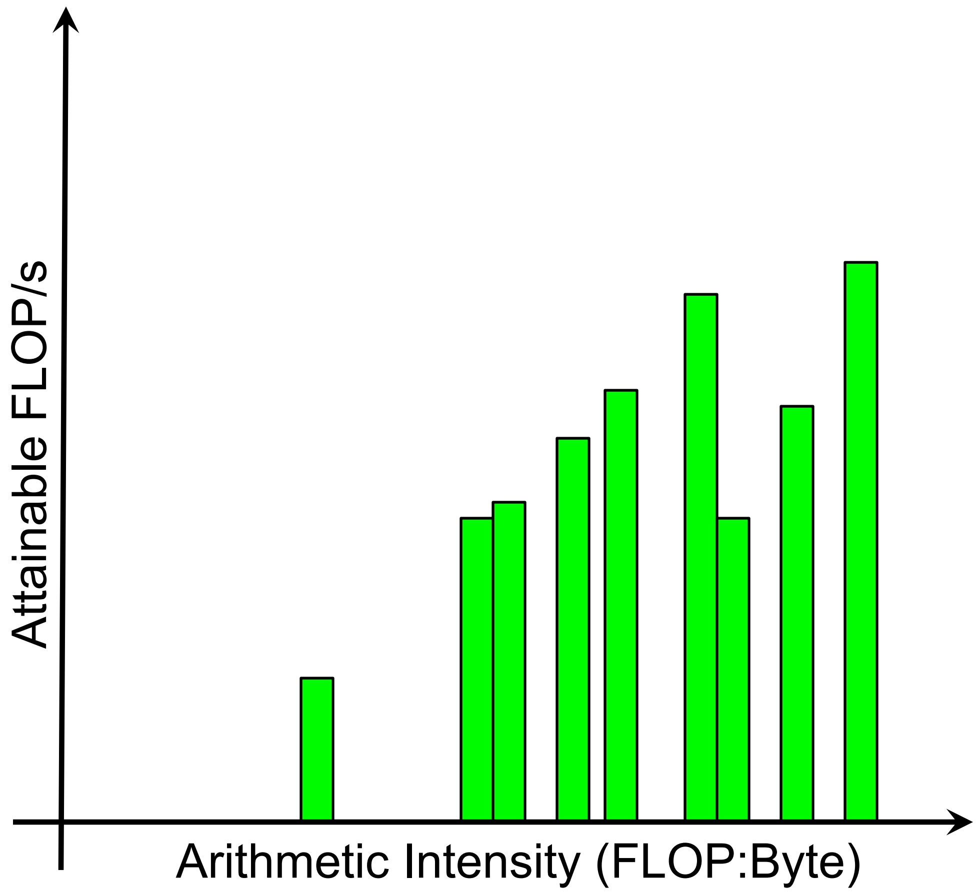
What is “Good” Performance?

- Think back to our mix of loop nests (benchmarks)…



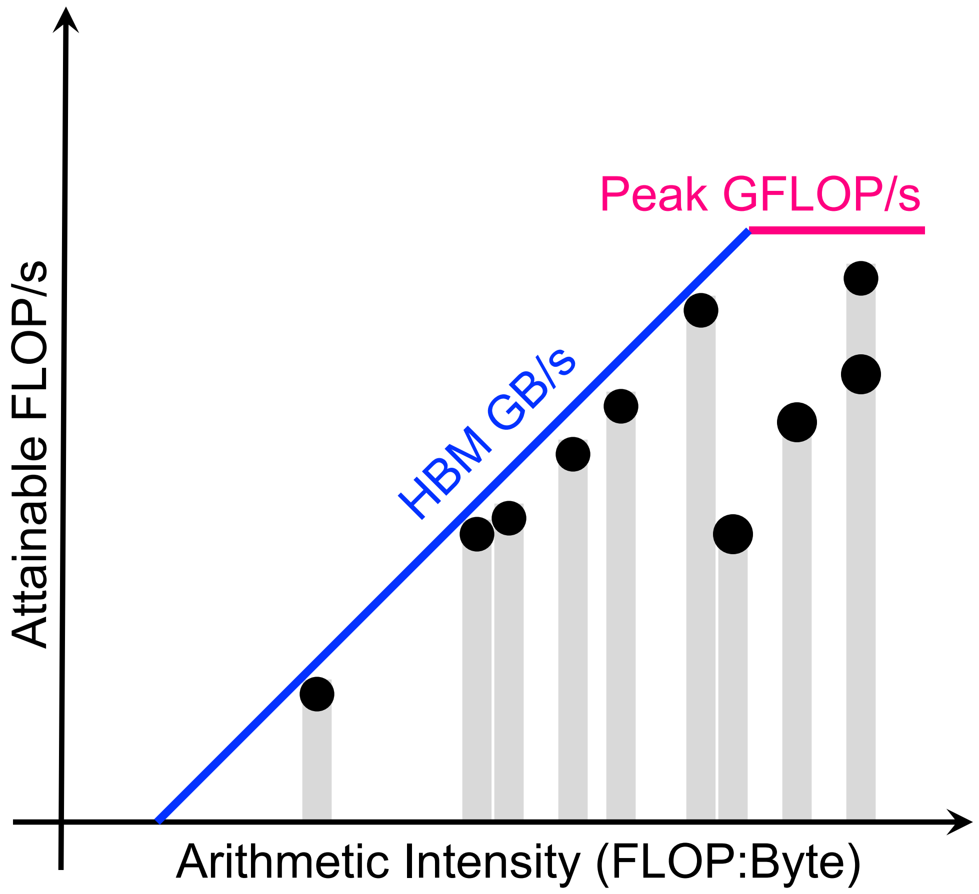
What is “Good” Performance?

- Think back to our mix of benchmarks (kernels)
- We can sort kernels by their arithmetic intensity…



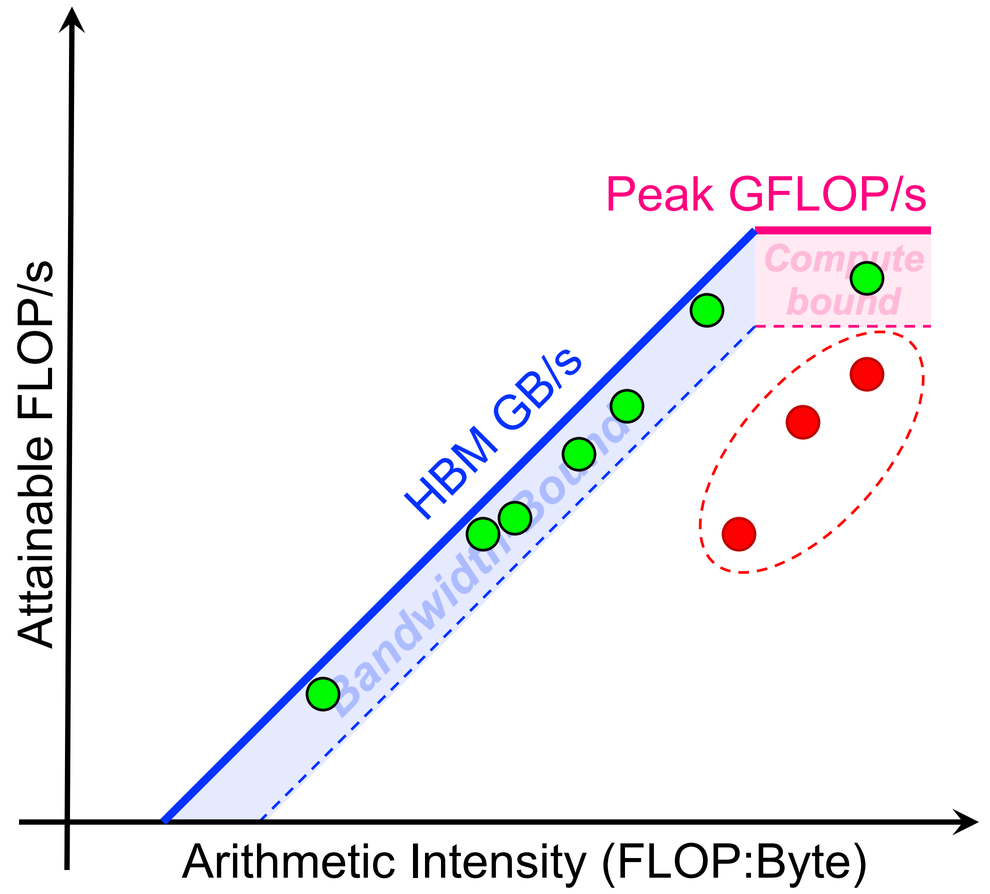
What is “Good” Performance?

- Think back to our mix of benchmarks (kernels)
- We can sort kernels by their arithmetic intensity…
- … and compare performance relative to machine capabilities



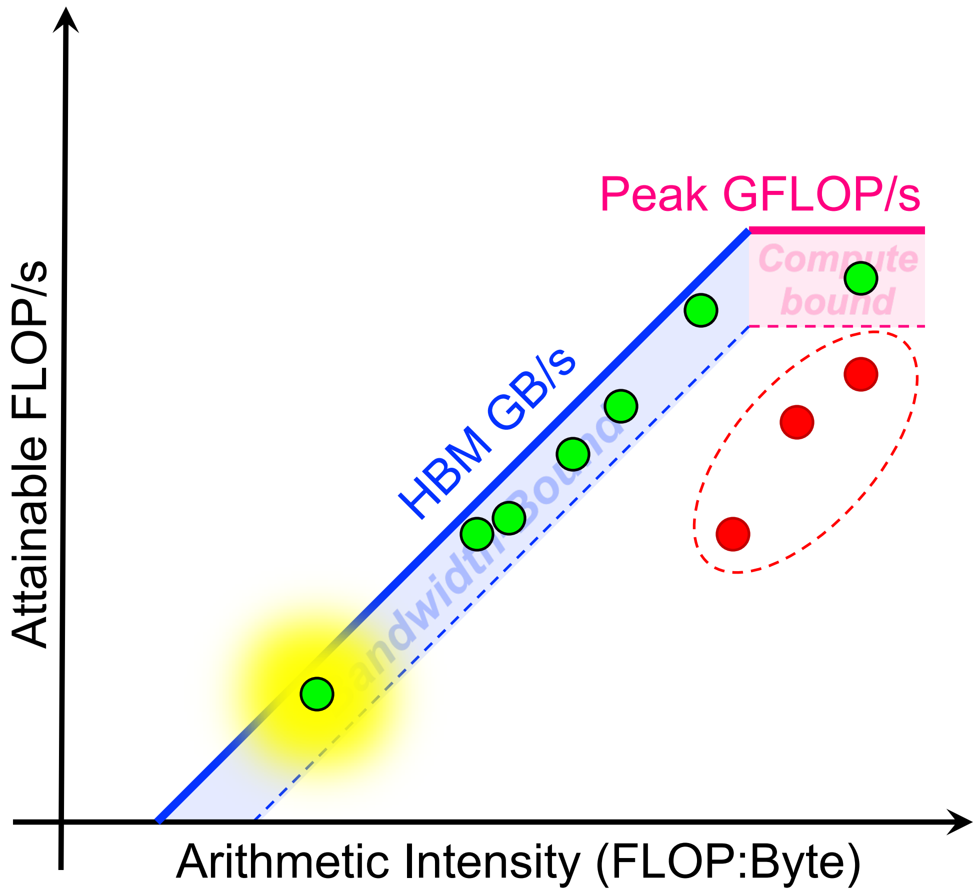
What is “Good” Performance?

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



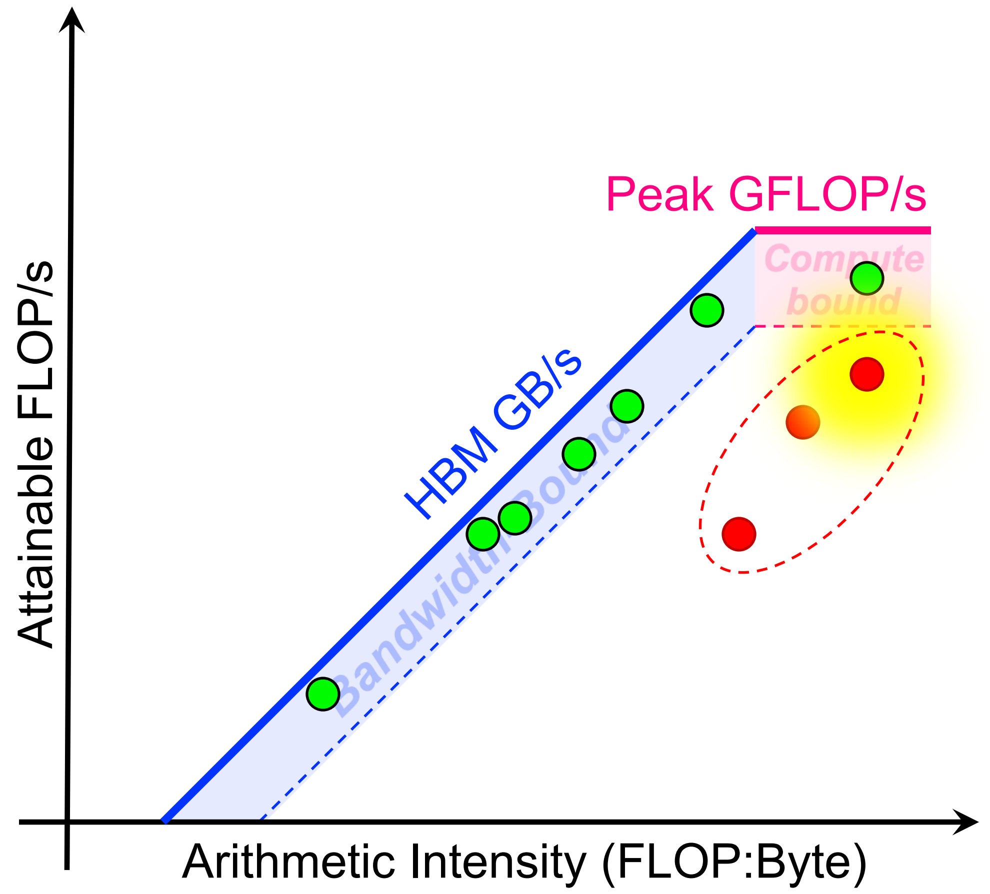
What is “Good” Performance?

- 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?

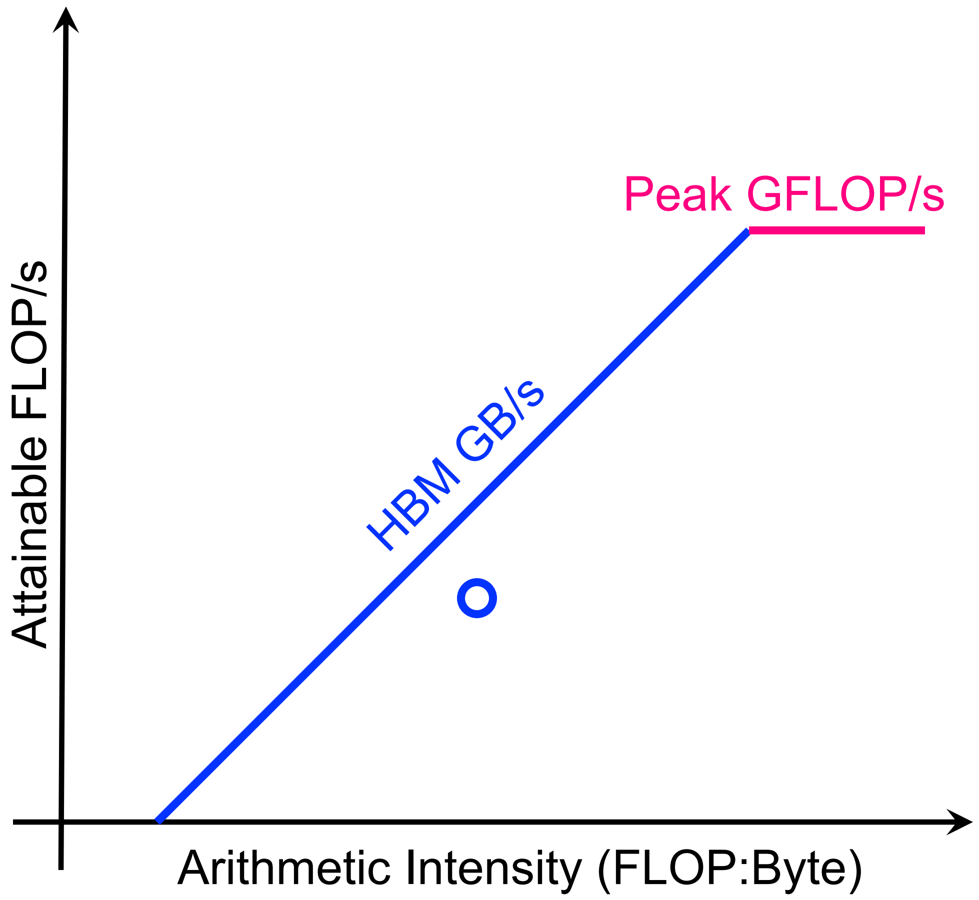
- Kernels near the roofline are making **good use** of computational resources
  - kernels can have low performance (GFLOP/s), but make good use (%STREAM) of a machine
  - kernels can have high performance (GFLOP/s), but still make poor use of a machine (%peak)

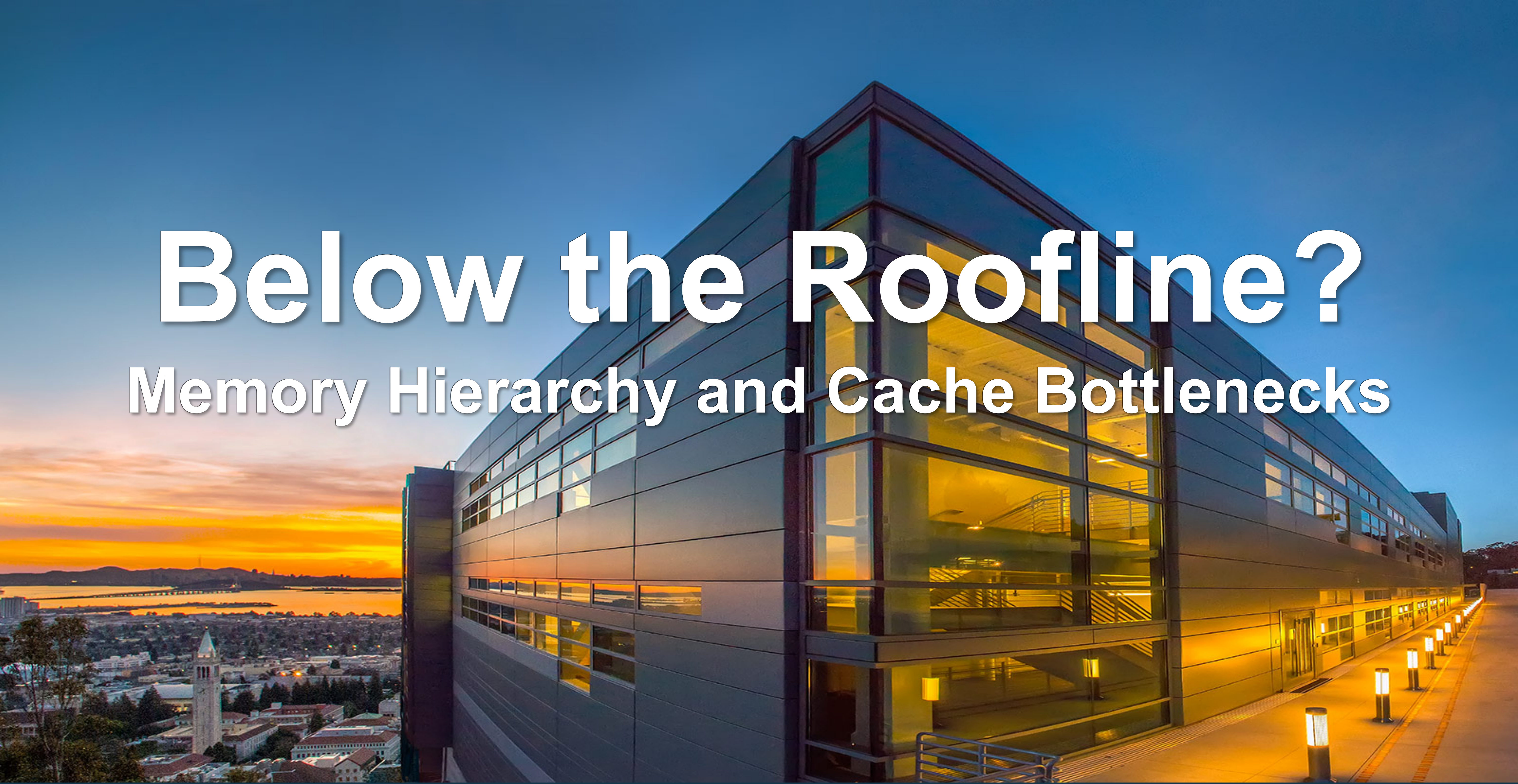


How can performance ever be below the Roofline?

How can performance be below the Roofline?

- Does one always attain either…
  - Peak DRAM Bandwidth
  - Peak FLOP/s
- Theoretical vs. Empirical
  - Use benchmarked GFLOP/s and GB/s
  - Application FLOPs can be underestimated (how many FLOPs is a divide?)
- Bottlenecks other than DRAM and FLOP/s…
  - Insufficient cache bandwidth + locality
  - Didn’t use FMA / Vectors / Tensors / …
  - Too many non-FP instructions
  - etc…



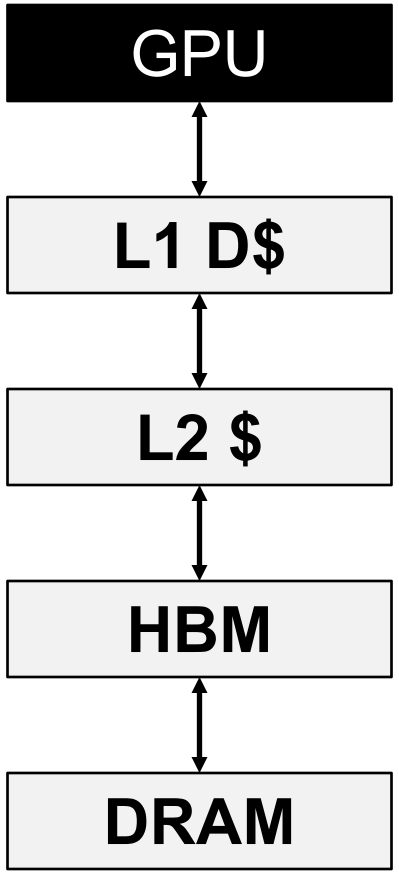


Below the Roofline?

Memory Hierarchy and Cache Bottlenecks

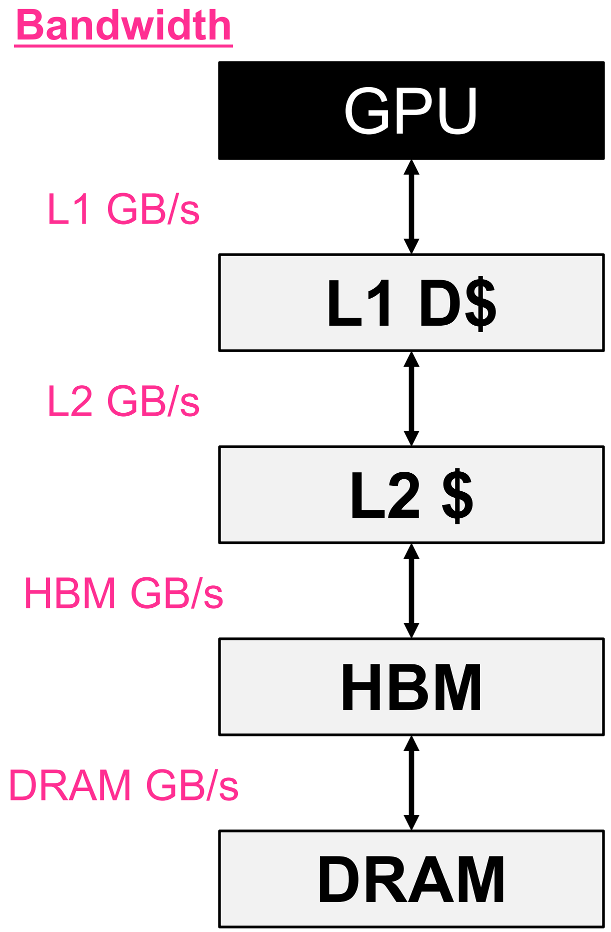
Memory Hierarchy

- GPUs have multiple levels of memory/cache
  - Registers
  - L1, L2, cache
  - HBM (GPU device memory)
  - DDR (host memory)



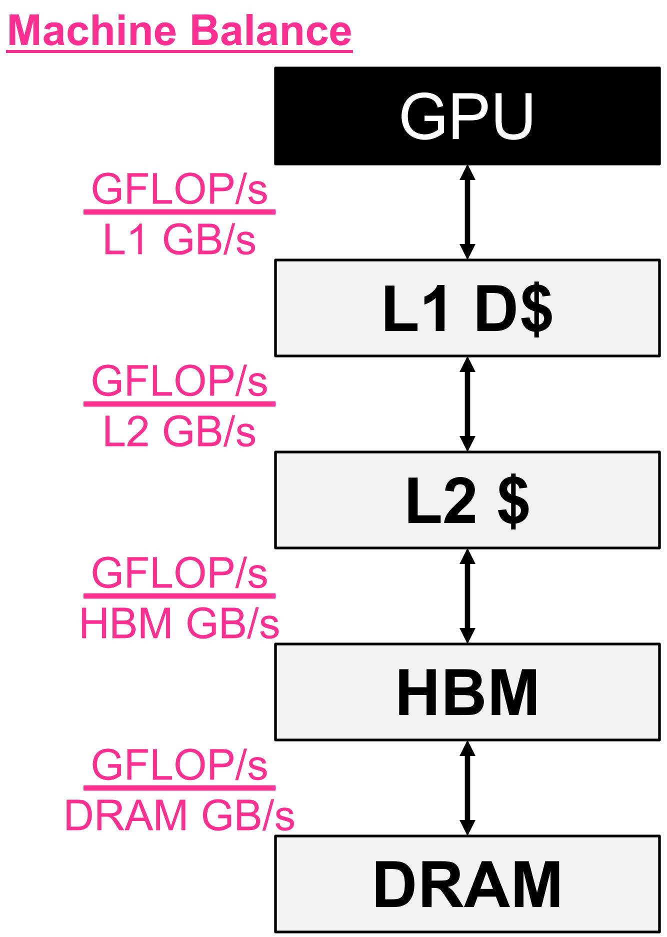
Memory Hierarchy

- GPUs have different bandwidths for each level



Memory Hierarchy

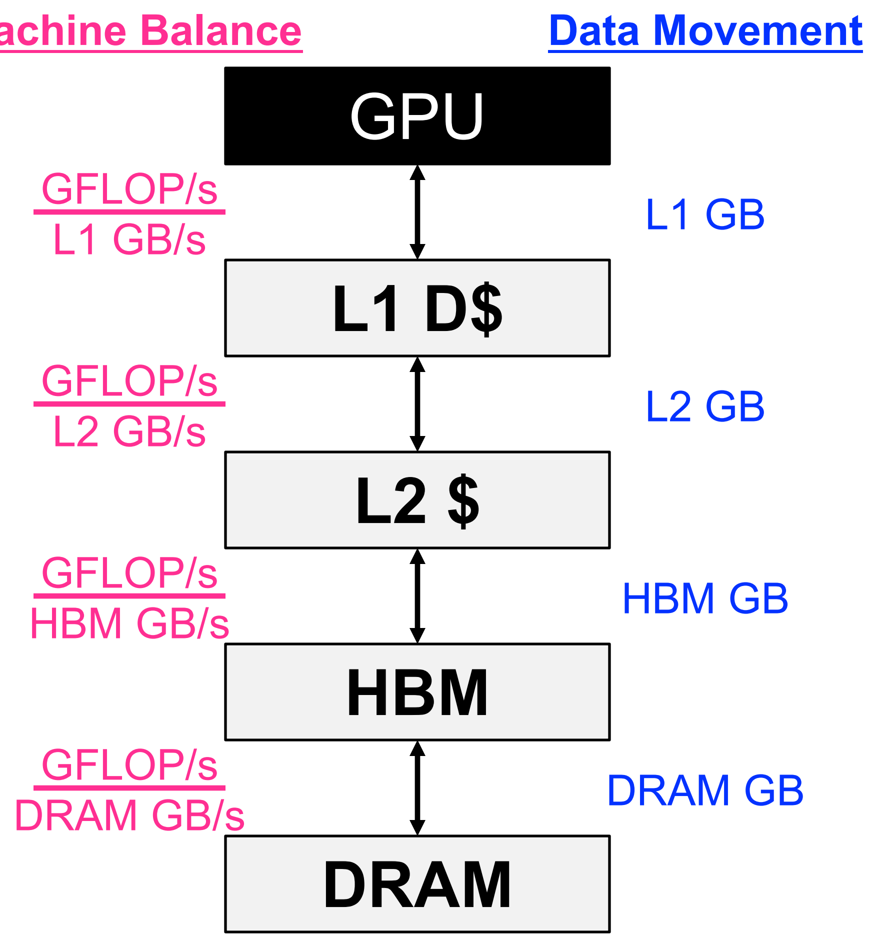
- GPUs have different bandwidths for each level
  - different **machine balances** for each level



Memory Hierarchy

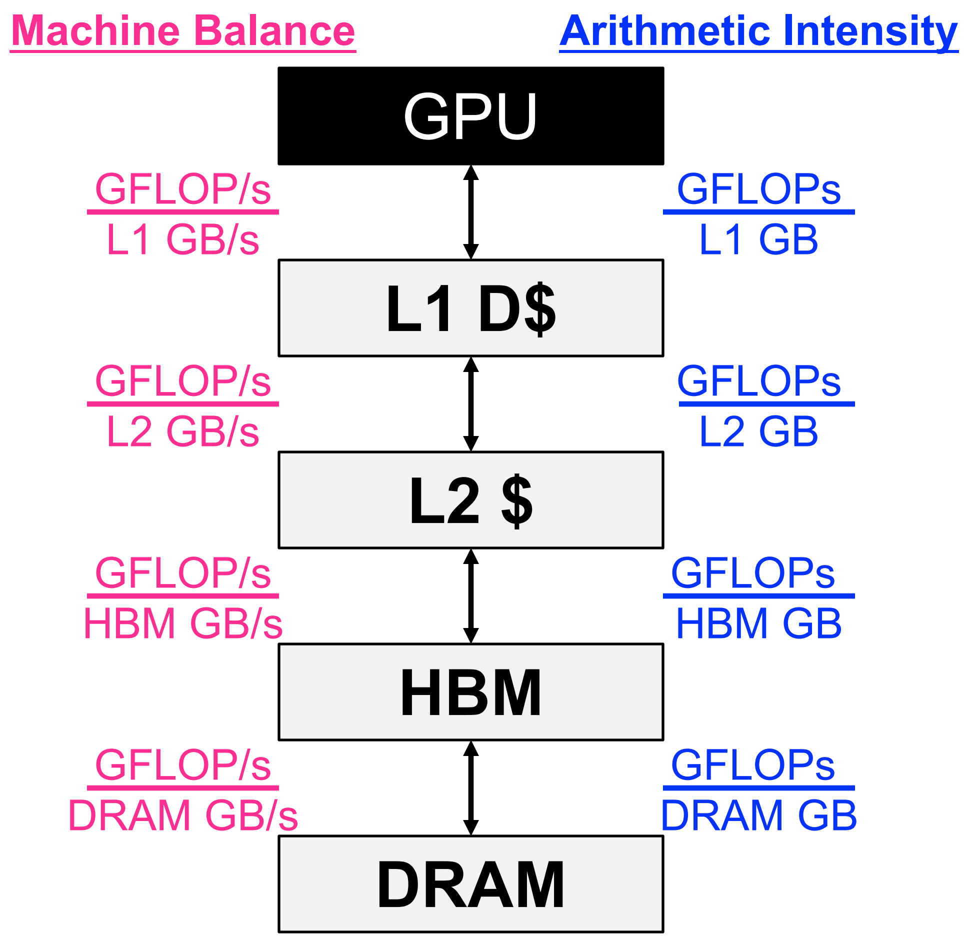
- GPUs have different bandwidths for each level
  - different machine balances for each level

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



Memory Hierarchy

- 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…

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

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…

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

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

Cache Bottlenecks

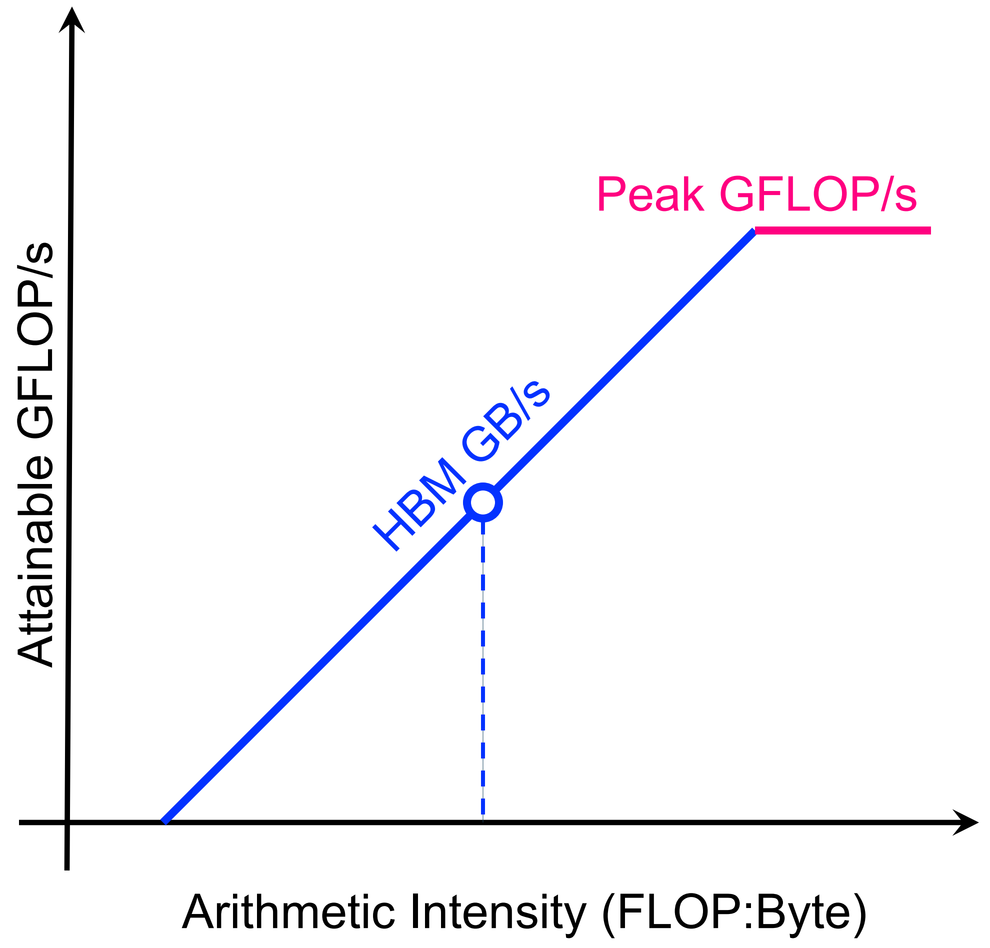
- For each additional level of the memory hierarchy, we can add another term to our model…

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

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

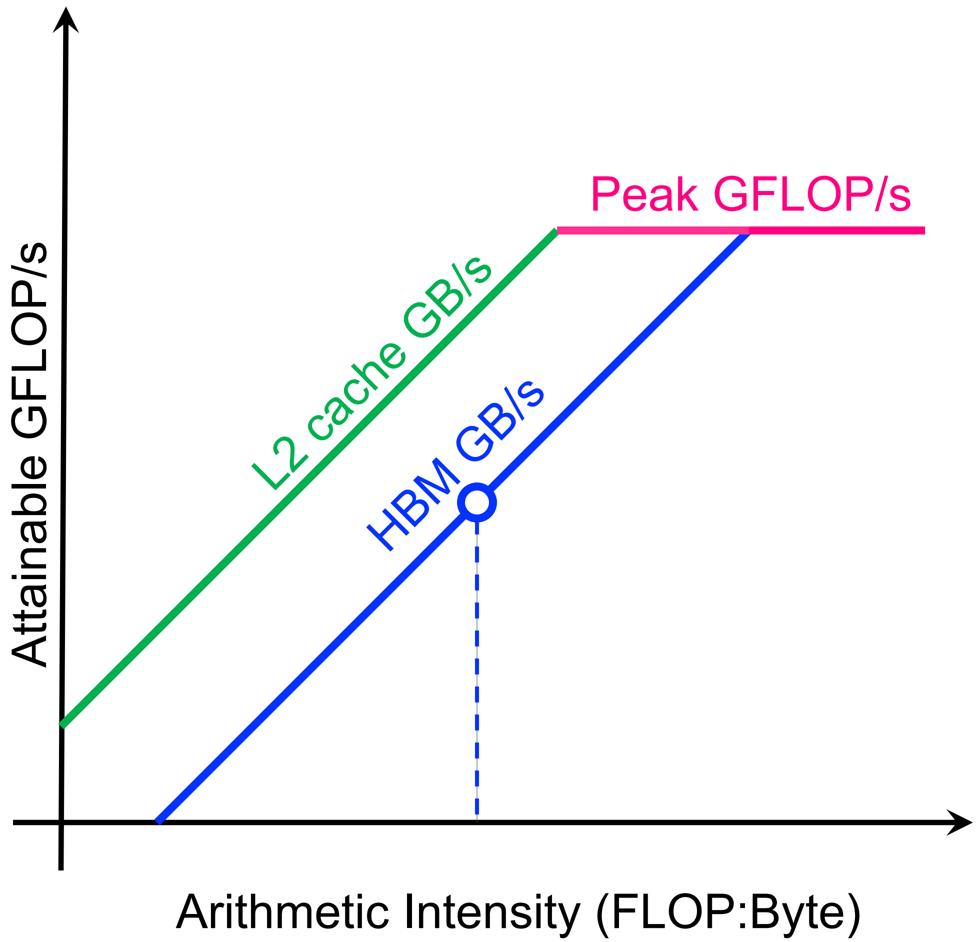
Cache Bottlenecks

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



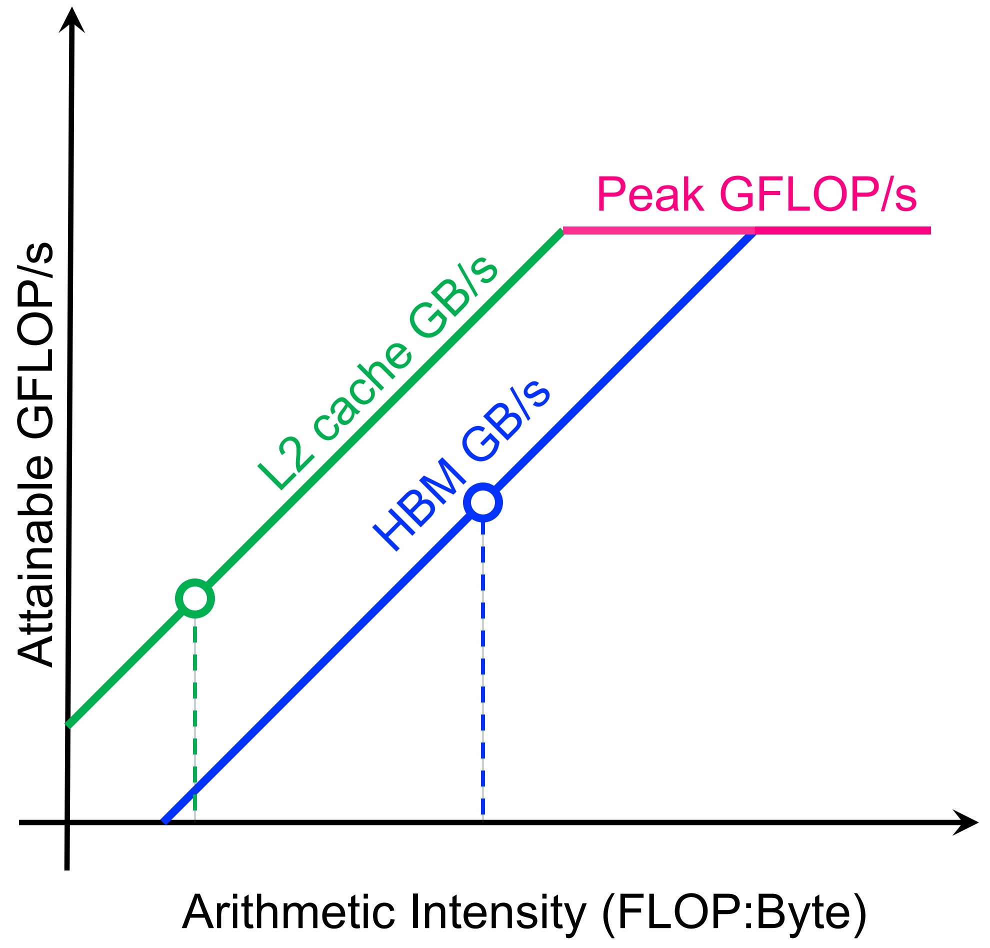
Cache Bottlenecks

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



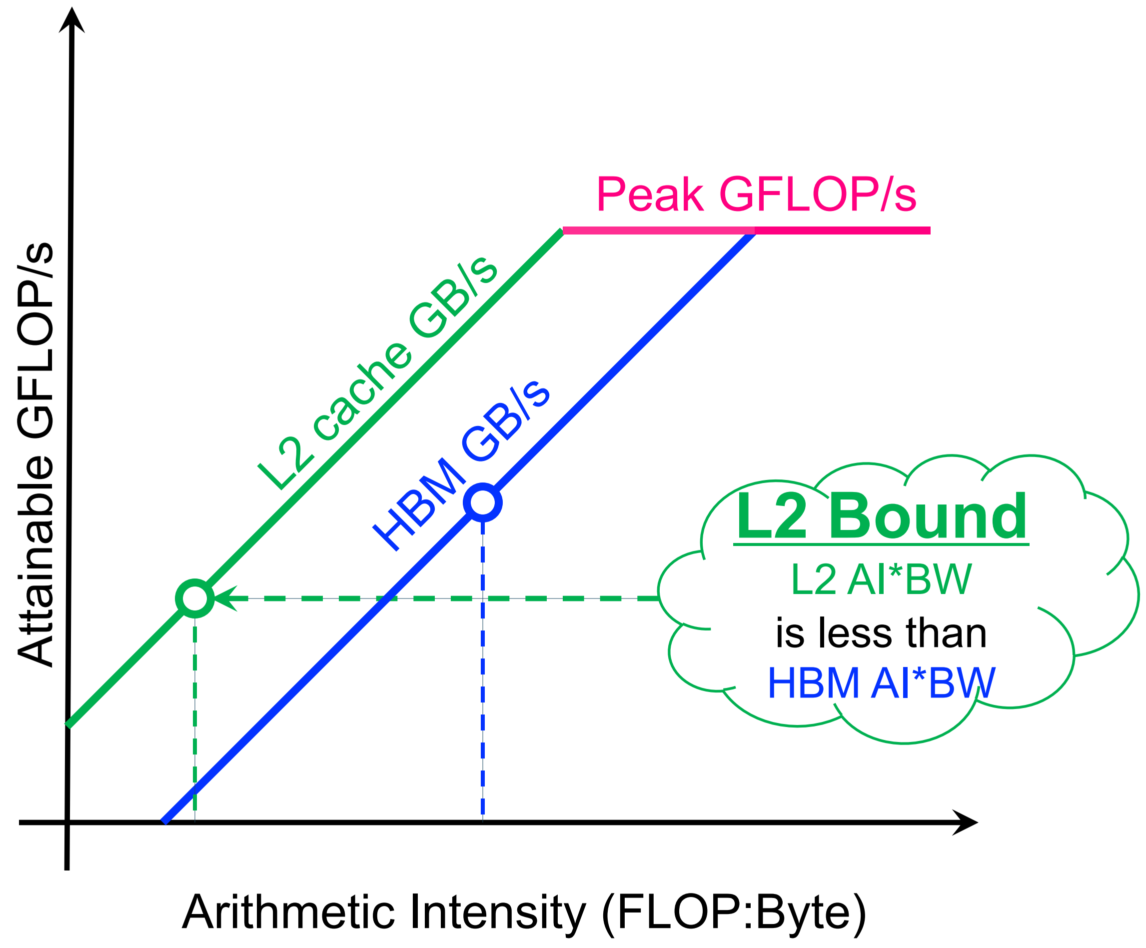
Cache Bottlenecks

- 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



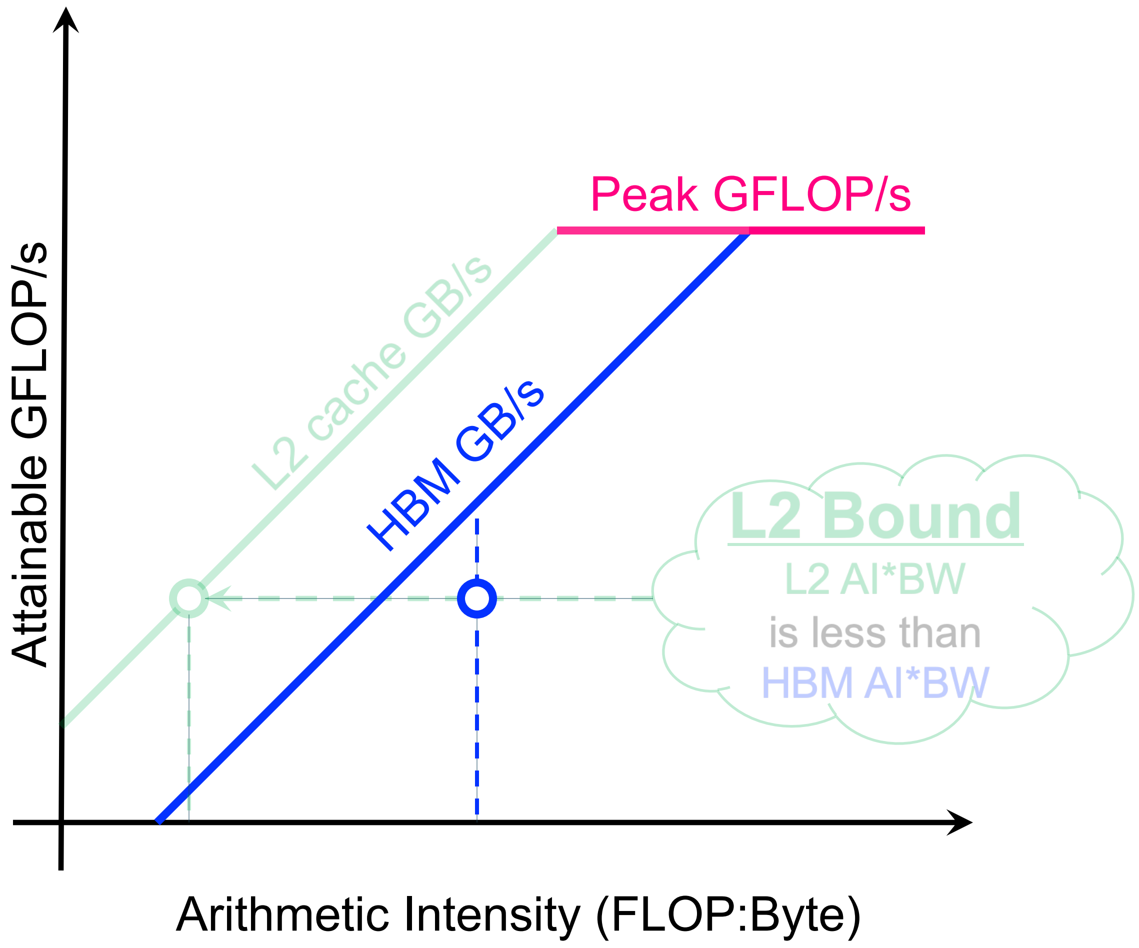
Cache Bottlenecks

- 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**



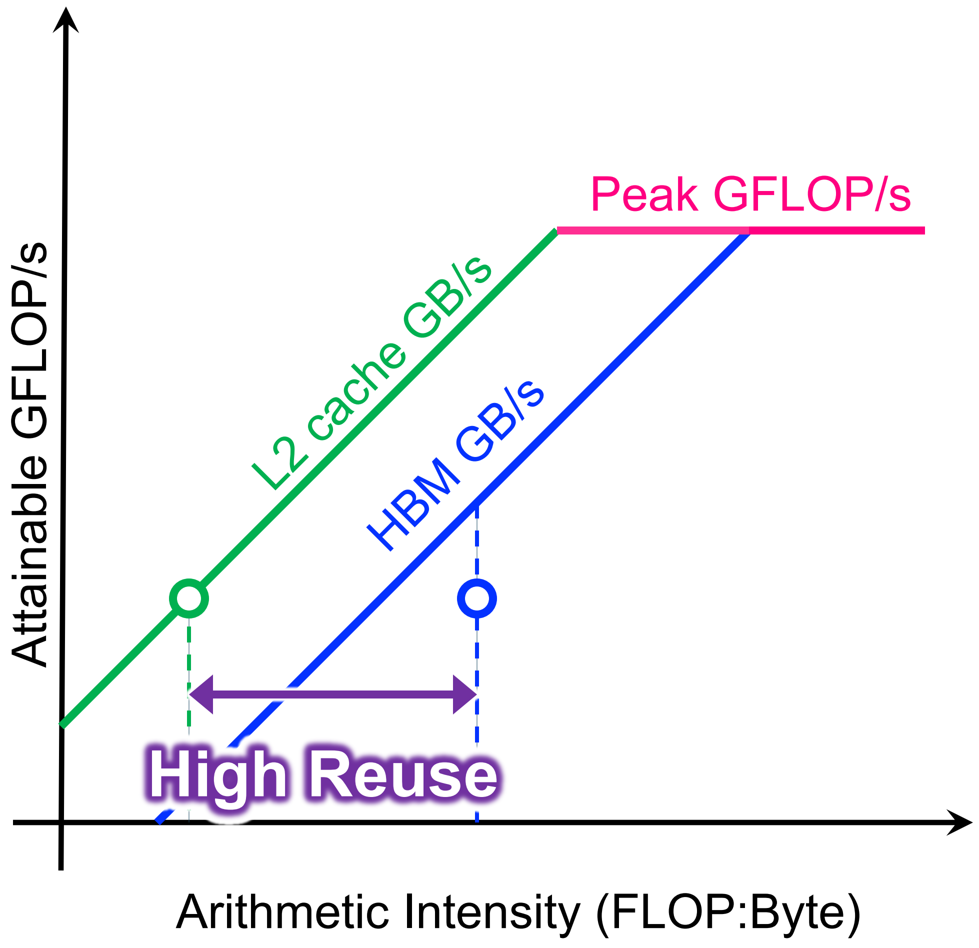
Cache Bottlenecks

- 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 HBM dot well below HBM ceiling**



Cache Bottlenecks

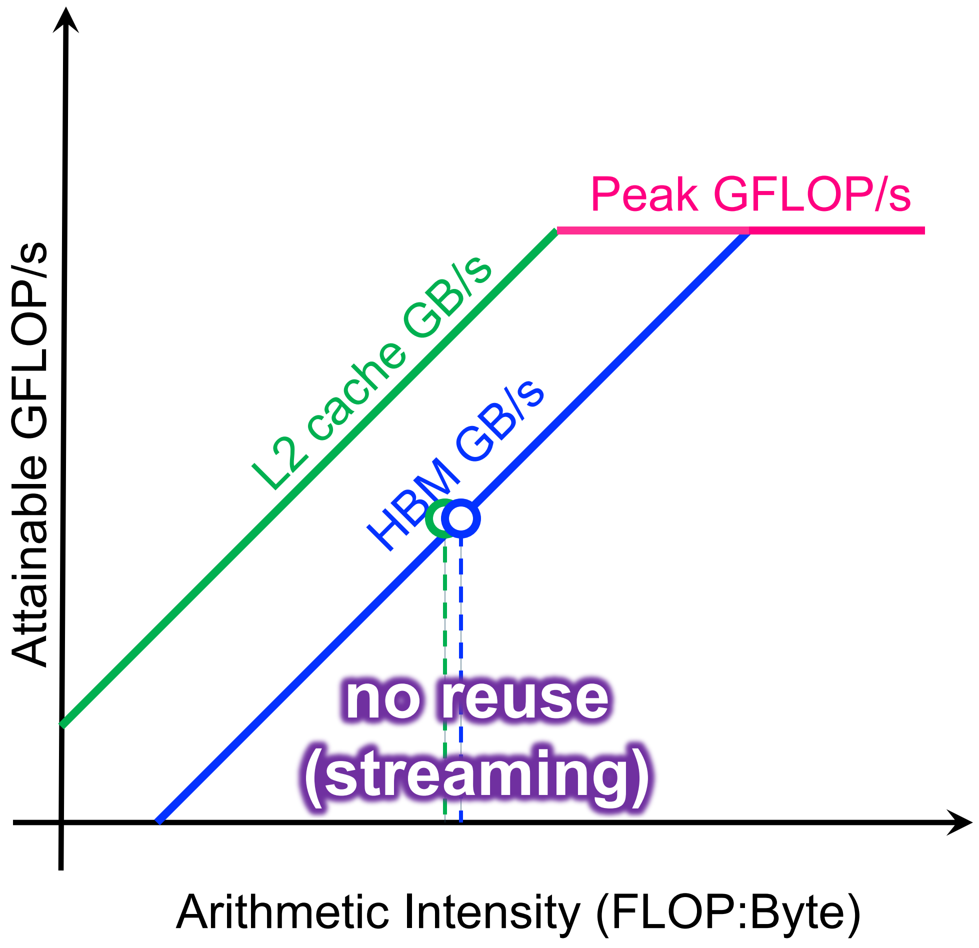
- Widely separated Arithmetic Intensities indicate high reuse in the cache



BERKELEY LAB

Cache Bottlenecks

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



Below the Roofline?

FMA, Reduced Precision, Tensor Cores

Return of CISC

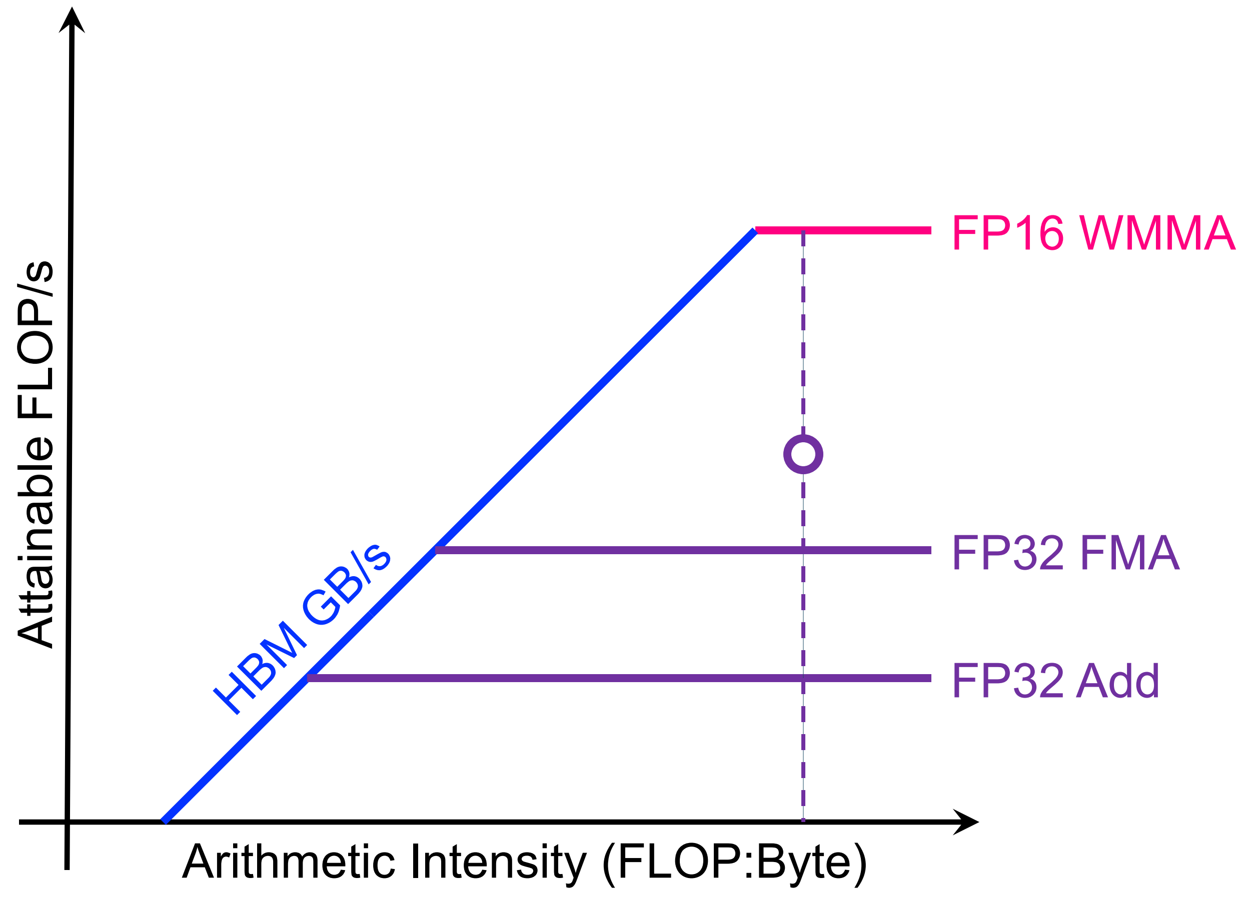
- Vectors have their limits (finite DLP, register file energy scales with VL, etc…)
- Death of Moore's Law is reinvigorating Complex Instruction Set Computing (CISC)

- 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.**

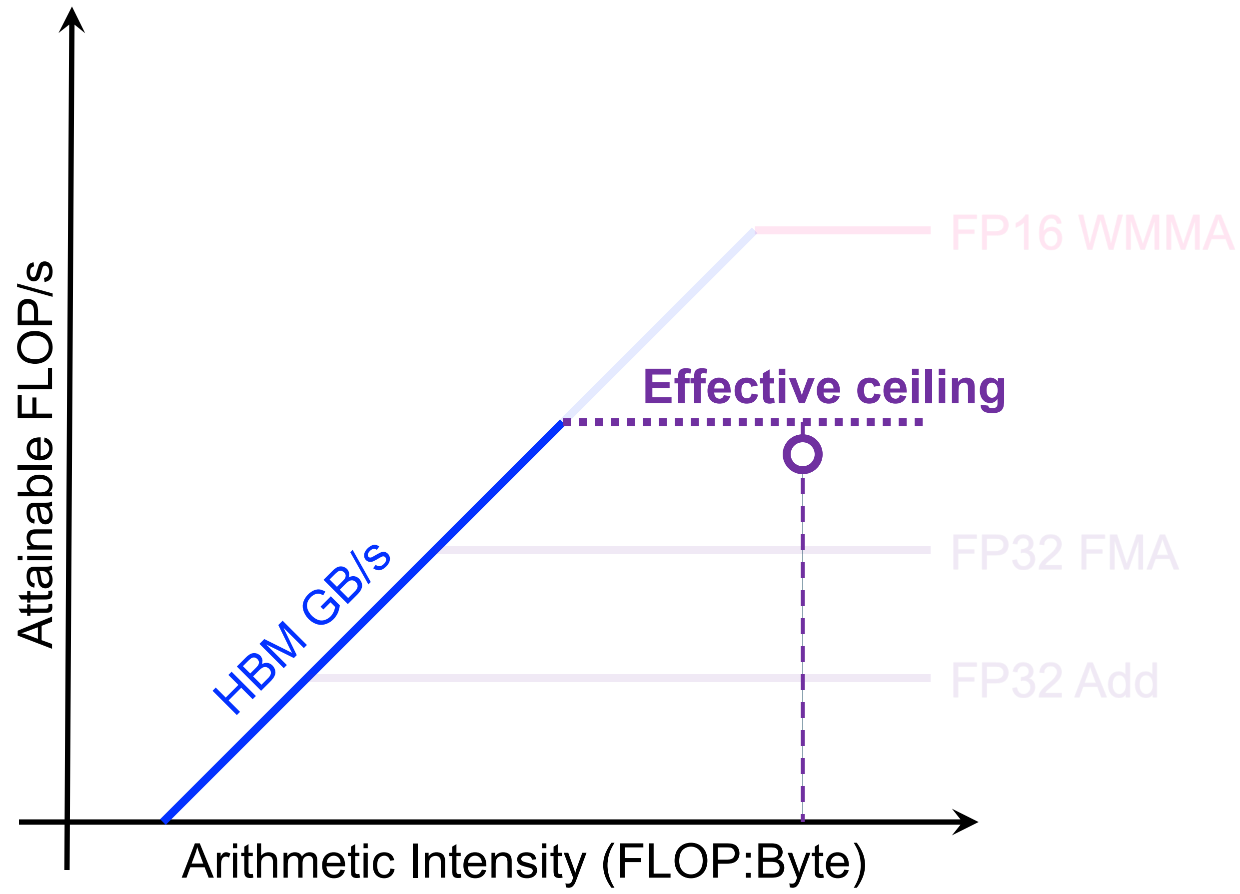
Return of CISC

- 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…



BERKELEY LAB

Below the Roofline?

FPU Starvation and Instruction Roofline

Think about classifying apps by instruction mix…

- Heavy floating-point (traditional FLOP Roofline)
- Mix of integer and floating-point (FPU starvation)
- Integer-only (e.g. bioinformatics, graphs, etc…)
- Mixed precision (e.g. deep learning)

} ***Instruction Roofline Model***

NVIDIA GPU Instruction Roofline

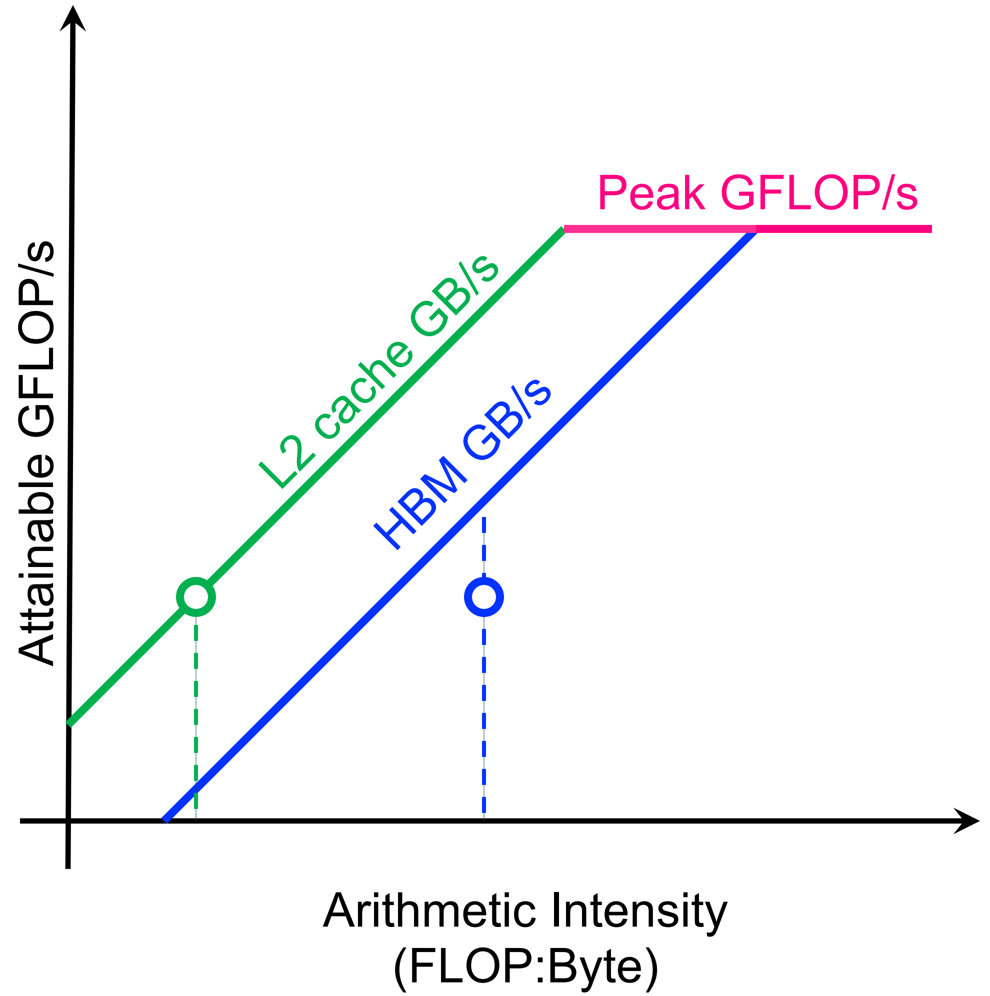
- Instructions per second? Instructions per Byte?

- 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)**

- Naively, one would think instruction intensity should use 'bytes'
- 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

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

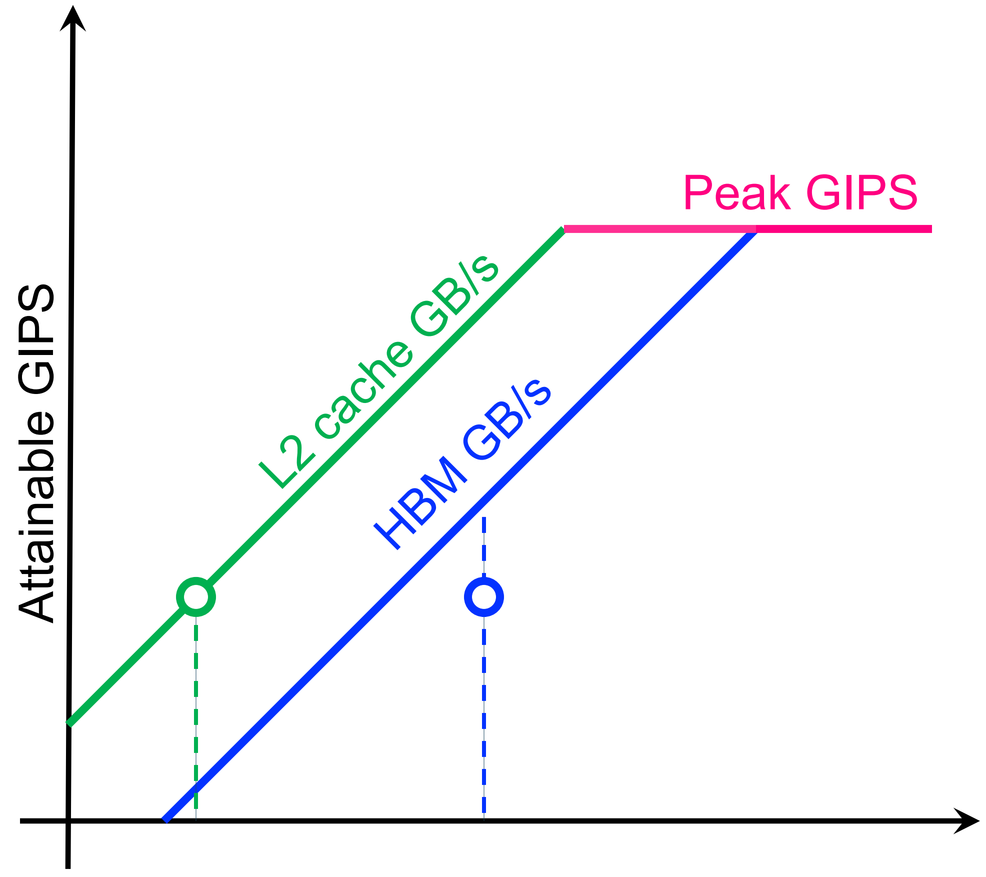


Instruction Roofline

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

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

Instructions per Byte



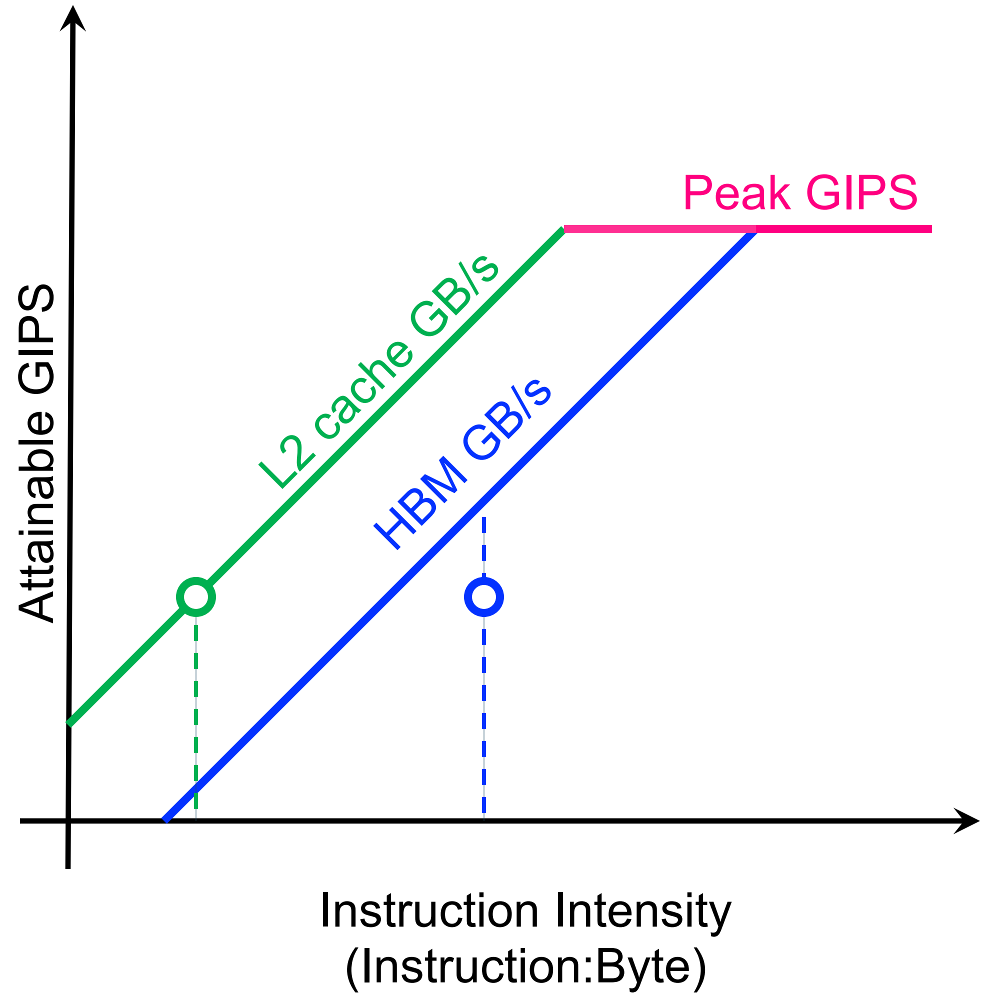
Instruction Roofline on GPUs

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

Warp Instructions

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

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



Instruction Roofline on GPUs

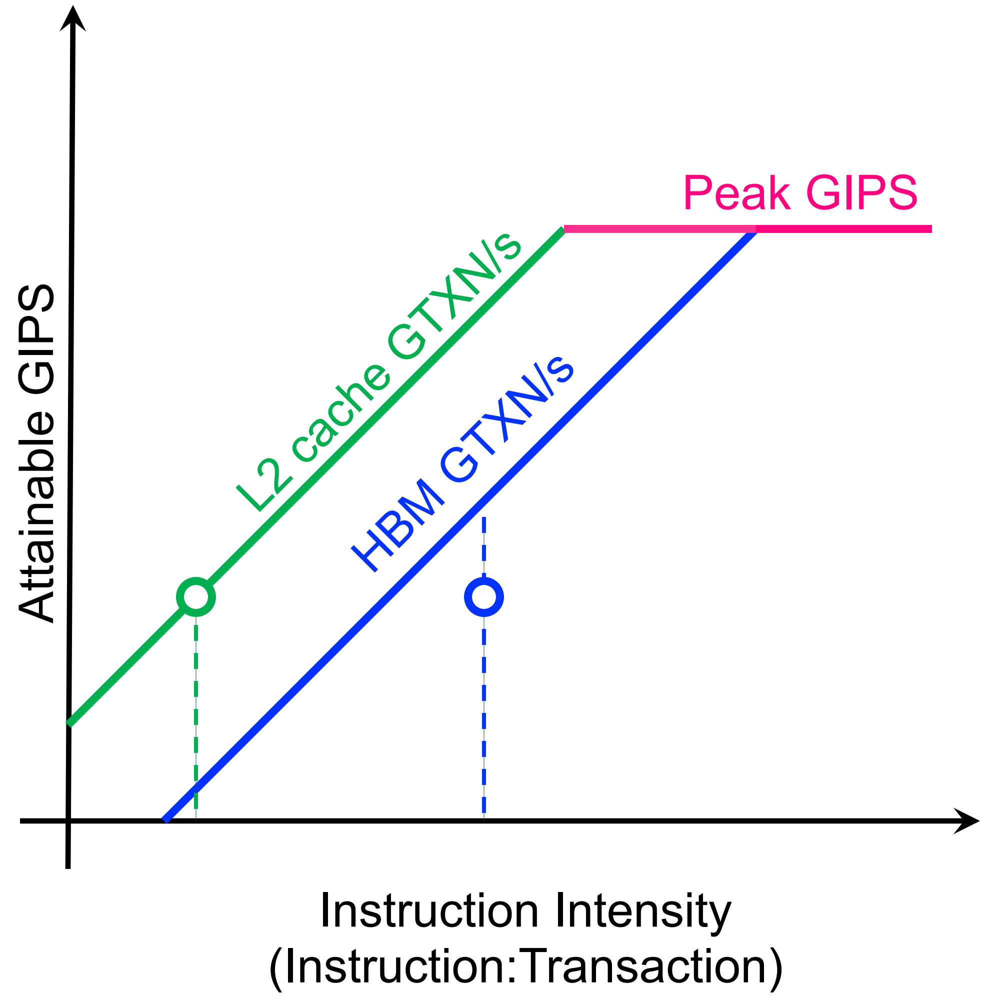
$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ AI_{DRAM} * \text{DRAM GB/s} \end{cases}$$

⬇

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

⬇

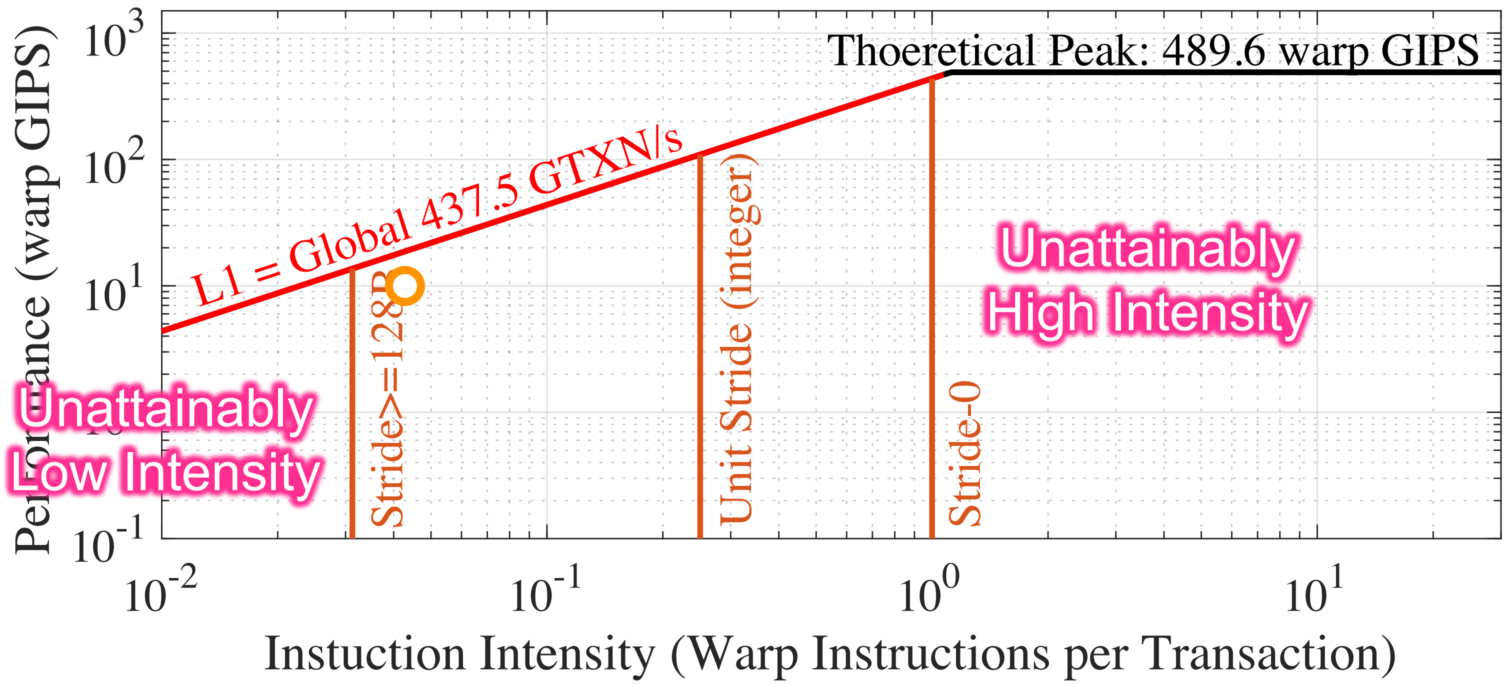
$$\text{GIPS} = \min \begin{cases} \text{Peak GIPS} \\ II_{DRAM} * \text{DRAM GTXN/s} \end{cases}$$



II_x (Instruction Intensity at level "x") =
Instructions / Transactions (to/from level "x")

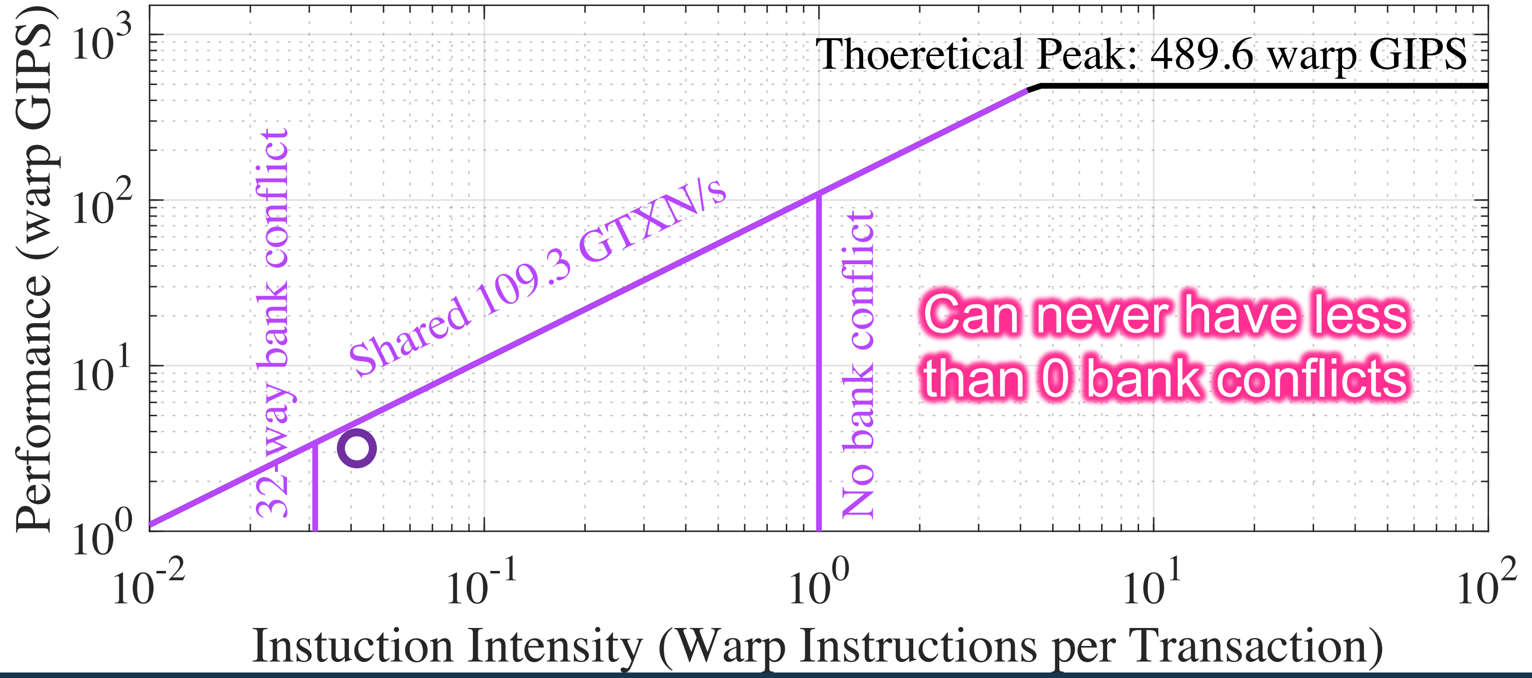
Efficiency of Global Memory Access

- (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)

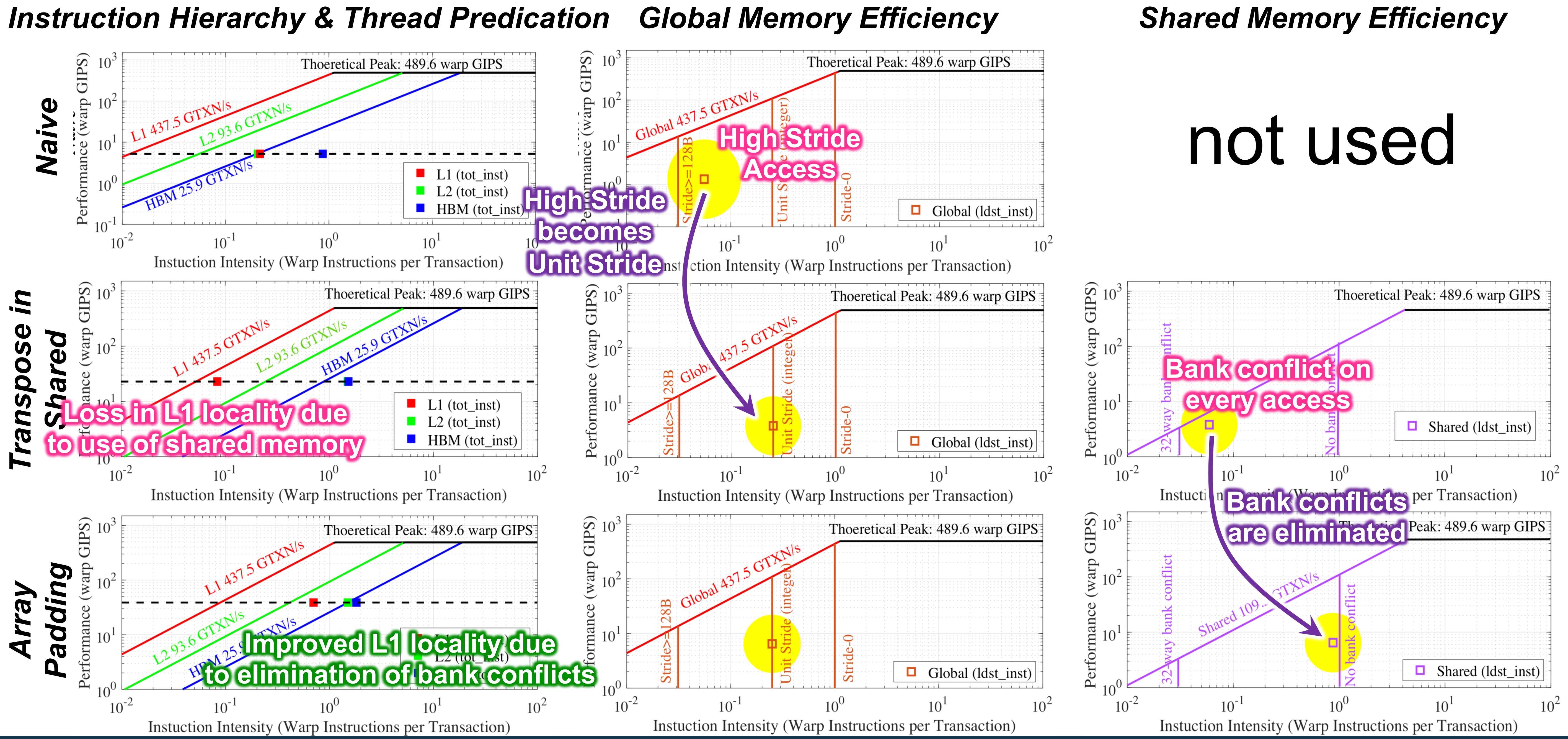


Efficiency of Shared Memory Access

- (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 Matrix Transpose



Recap

Roofline Recap

Bounds performance

- 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

- Different intensity for each level of memory
- Total FLOPs / Total Data Movement
- Includes **all** cache effects

> **Measure of a loop's temporal locality**

Plotting loops…

- Each loop has one dot **per level** of memory
- x-coordinate =
  AI at that level
- y-coordinate =
  GFLOP/s

> **Proximity to associated ceiling is indicative of a performance bound**

> **Position of dots relative to each other is indicative of cache locality**

Instruction Roofline Takeaway

Traditional Roofline

- **Tells us about performance**
  (floating-point)
- Intensity based on data locality (FLOPs / Bytes)
- Use of FMA, SIMD, vectors, tensors has no affect on intensity
- Presence of integer instructions has no affect on intensity.
- Reducing precision (64b, 32b, 16b) increases arithmetic intensity

Instruction Roofline

- **Tells us about bottlenecks**
  (issue and memory)
- Intensity based on **total** instructions and transactions
- Use of FMA, SIMD, vectors, tensors decreases intensity.
- Presence of integer instructions increases intensity.
- Reducing precision has no affect on intensity

Memory Walls

- **Tells us about efficiency**
  (memory access)
- Intensity based on LDST instructions and transactions
- Reducing precision shifts intensity and the unit-stride wall

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

Questions?

BACKUP