Roofline Performance Model

JaeHyuk Kwack
Argonne National Laboratory

Outline

Introduction to the roofline model

Hands-on example on Aurora

Introduction to the Roofline Model

This is a short version of SC Tutorial Slides generated by Samuel Williams, LBNL
(swwilliams@lbl.gov)

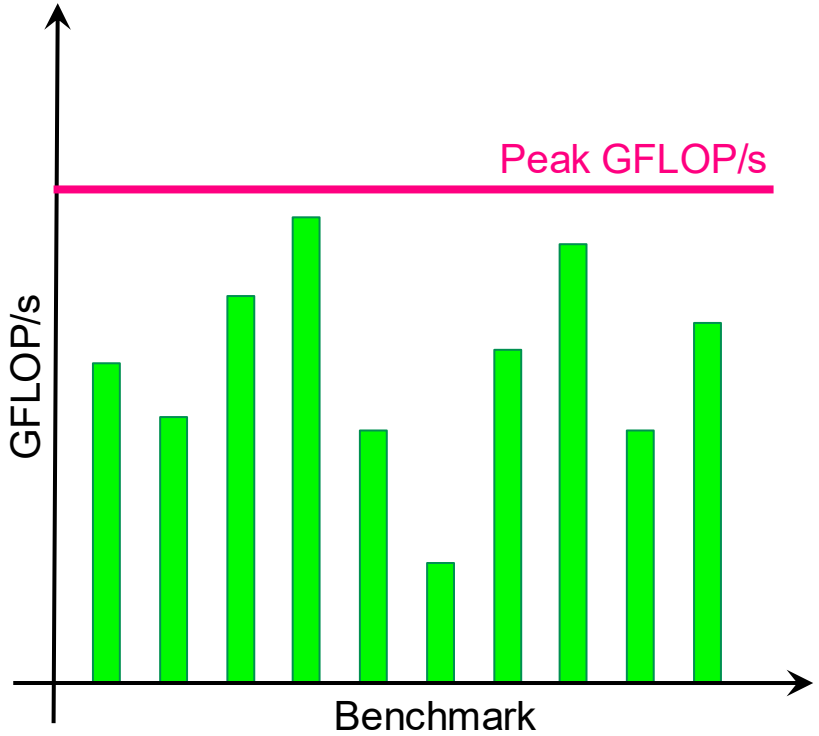
Please join SC26 tutorial for Roofline model for more details and practices

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?

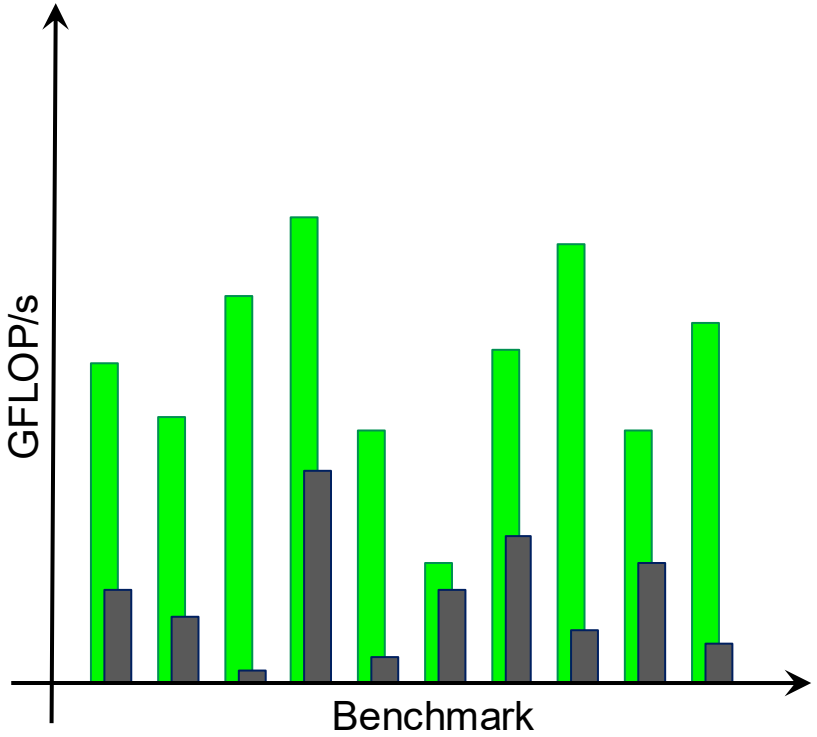
- Really a question of getting good performance on application benchmarks
- Imagine profiling a mix of GPU-accelerated benchmarks …
- Performance (GFLOP/s) alone may not be particularly insightful



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.**



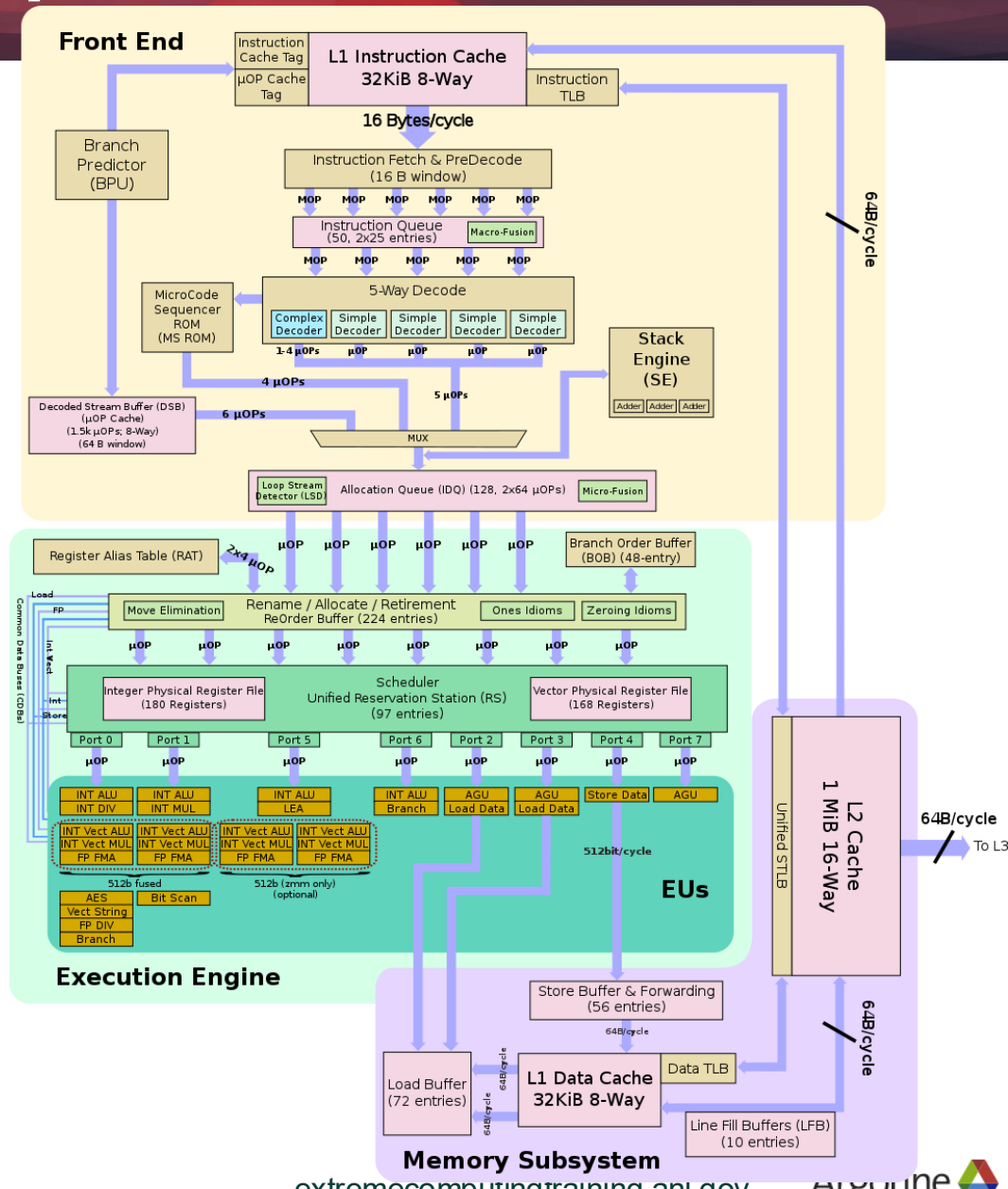
Are we getting good performance?

- Instead of speedup, we could take a CS approach and look at performance counters…
  - Record microarchitectural events on CPUs/GPUs
  - Use 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?

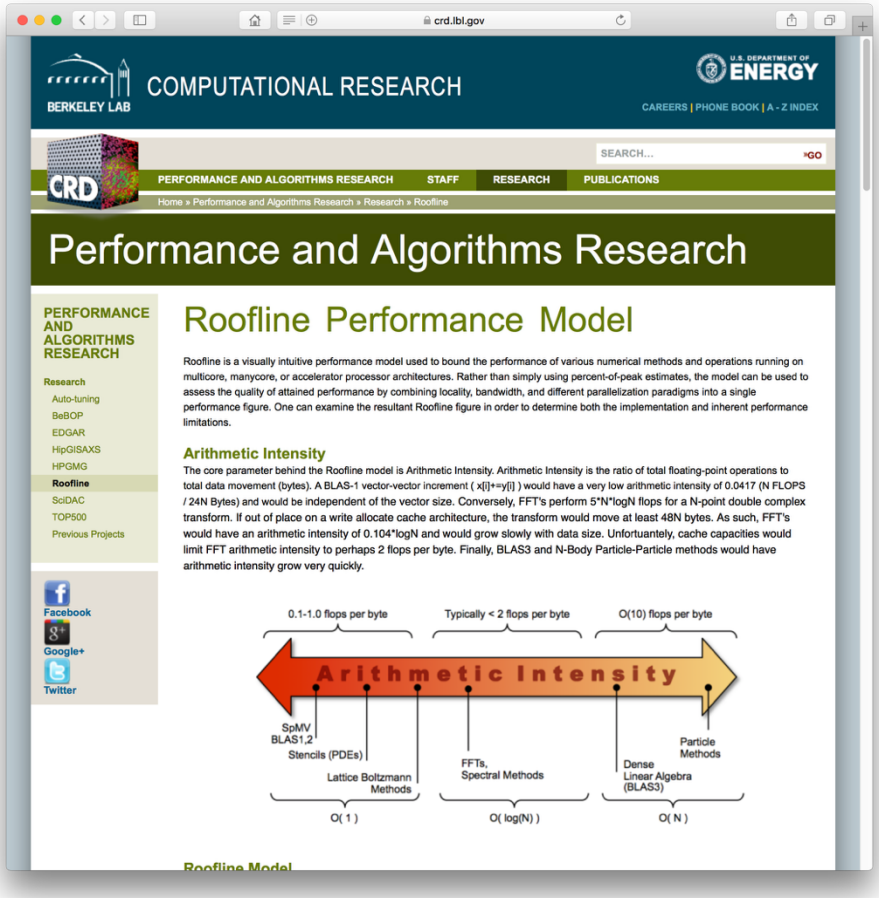
- We could take the computer architect's approach and build a simulator to understand performance nuances…
  - Modern architectures are incredibly complex
  - Simulators may perfectly reproduce performance
  - Lots of information interpretable only by computer architects
  - worse, might incur 10^6x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



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

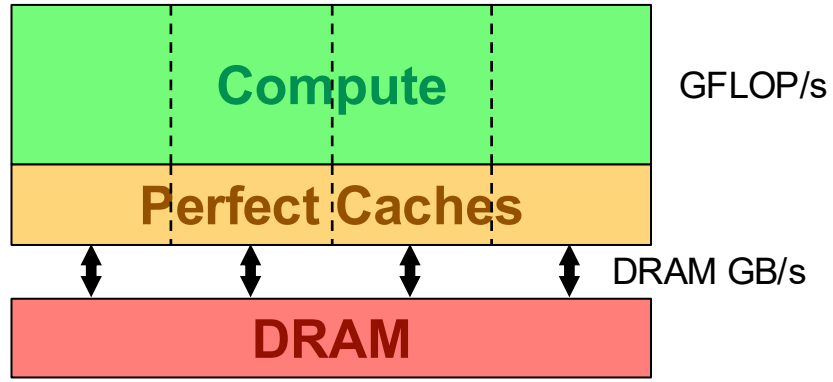
➢ **Roofline is just such a model**



https://crd.lbl.gov/roofline

Data Movement or Compute?

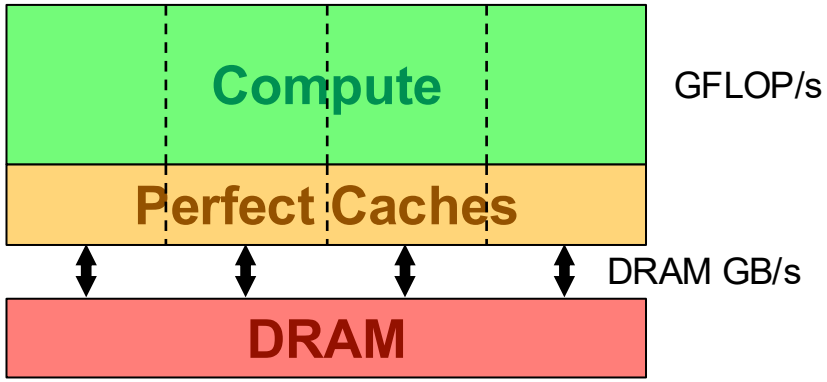
- Assume HW/SW can perfectly overlap communication and computation
- Which takes longer?
  - Data Movement
  - Computation



Time = max { #FP ops / Peak GFLOP/s ; #Bytes / Peak GB/s }

Data Movement or Compute?

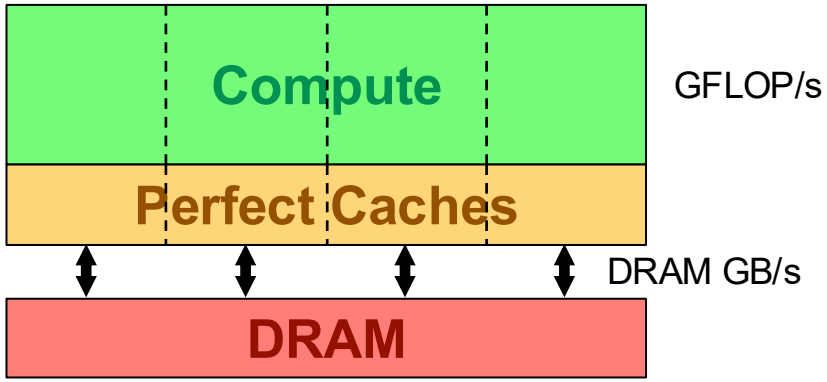
- Assume HW/SW can perfectly overlap communication and computation
- Which takes longer?
  - Data Movement
  - Computation
- Is performance limited by compute or data movement?



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

Data Movement or Compute?

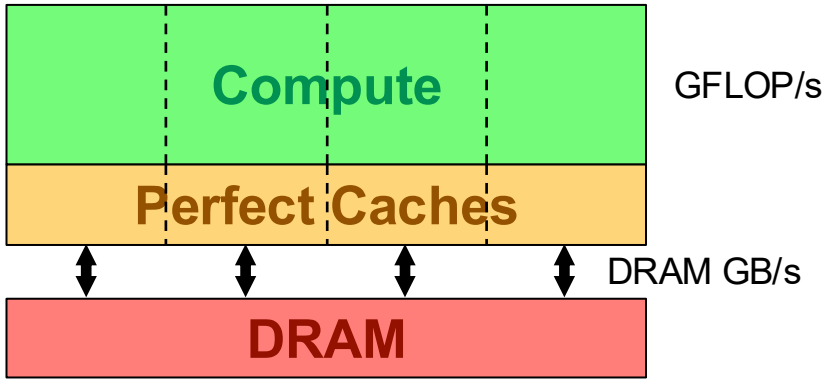
- Assume HW/SW can perfectly overlap communication and computation
- Which takes longer?
  - Data Movement
  - Computation
- Is performance limited by compute or data movement?



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

Data Movement or Compute?

- Assume HW/SW can perfectly overlap communication and computation
- Which takes longer?
  - Data Movement
  - Computation
- Is performance limited by compute or data movement?



GFLOP/s = min { Peak GFLOP/s ; AI * Peak GB/s }

Arithmetic Intensity (AI) = measure of data locality

Arithmetic Intensity

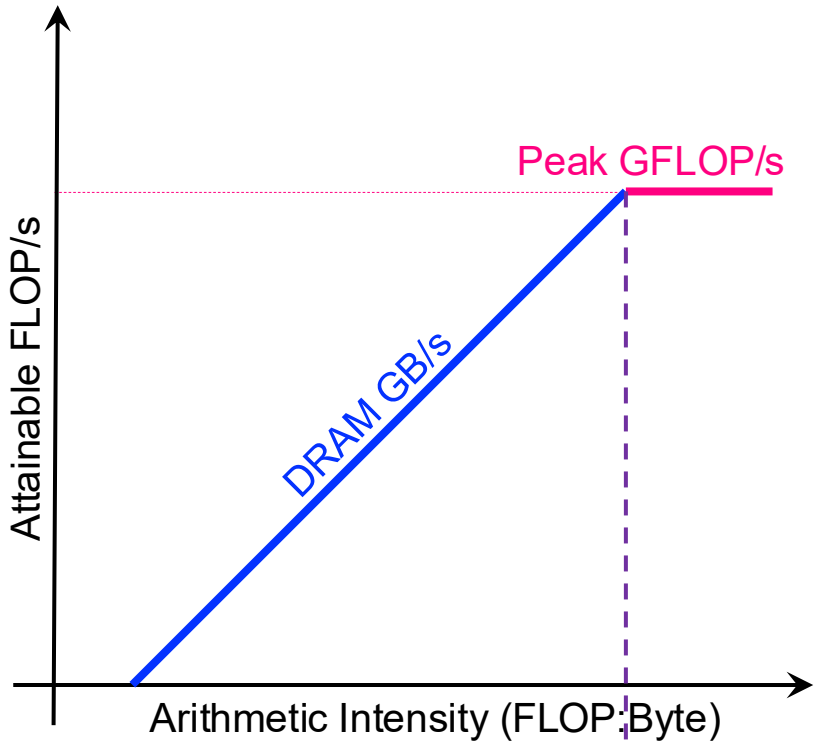
- Measure of data locality (data reuse)
- Ratio of **Total Flops** performed to **Total Bytes** moved
- For the DRAM Roofline…
  - Total Bytes to/from DRAM
  - Includes all cache and prefetcher effects
  - Can be very different from total loads/stores (bytes requested)
  - Equal to ratio of sustained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

(DRAM) Roofline Model

GFLOP/s = min { **Peak GFLOP/s**, **AI * Peak GB/s** }

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

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



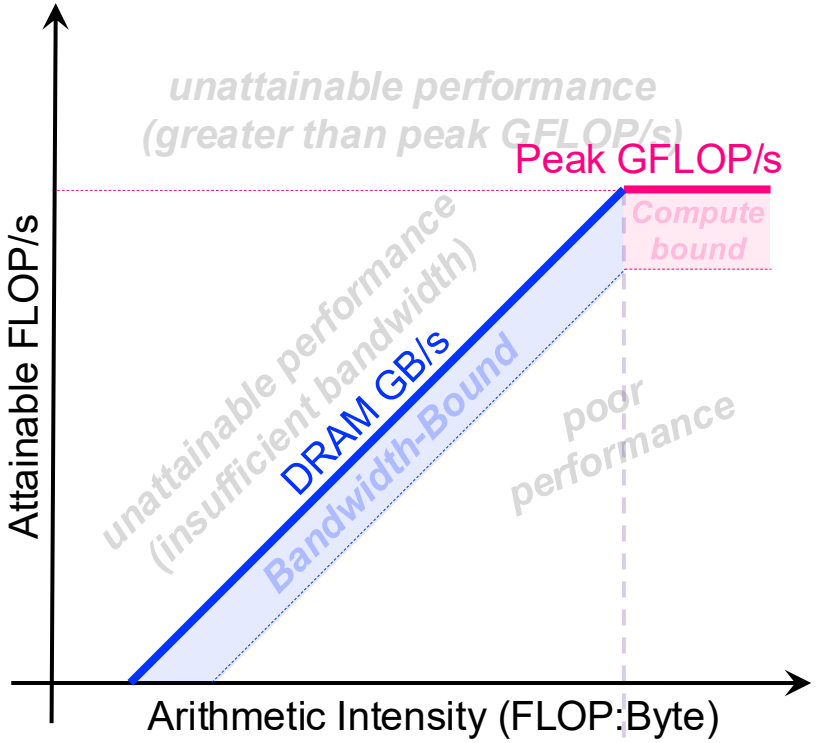
Transition @ AI == Peak GFLOP/s / Peak GB/s == 'Machine Balance'

(DRAM) Roofline Model

GFLOP/s = min { Peak GFLOP/s ; AI * Peak GB/s }

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…



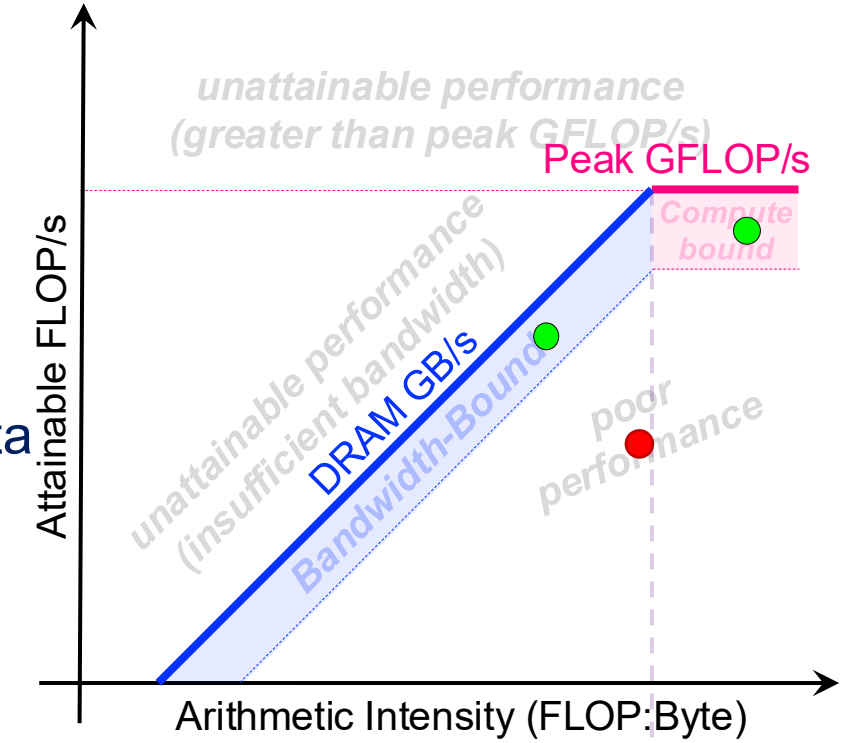
Transition @ AI == Peak GFLOP/s / Peak GB/s == 'Machine Balance'

(DRAM) Roofline Model

GFLOP/s = min { Peak GFLOP/s ; AI * Peak GB/s }

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

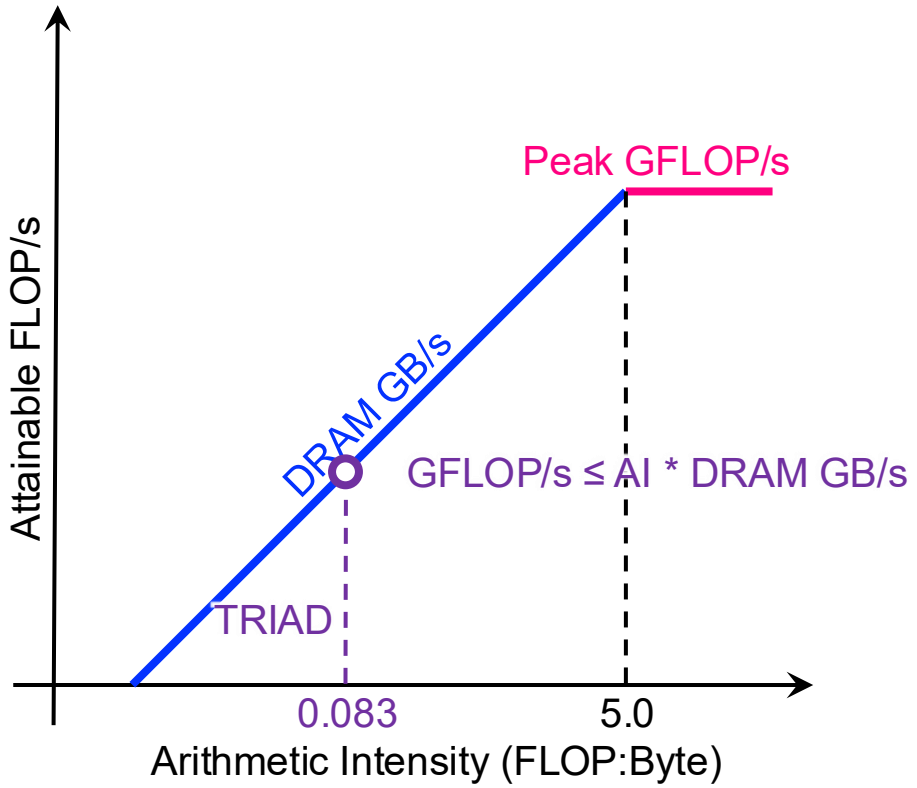
Roofline Example #1

- 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…

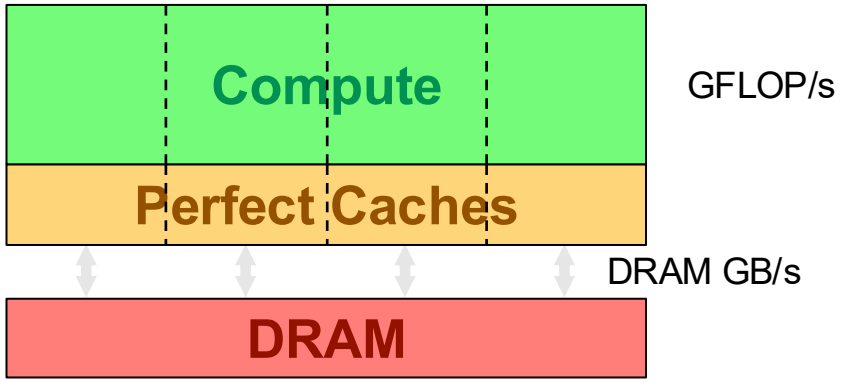
```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

  - 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

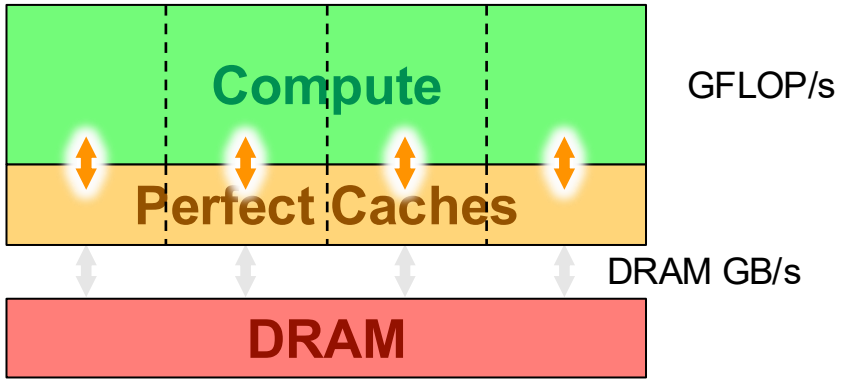
- Conversely, 7-point constant coefficient stencil…




```
#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

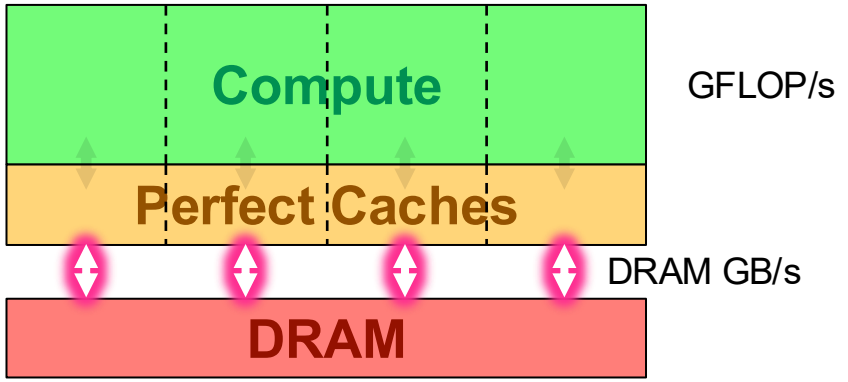
- Conversely, 7-point constant coefficient stencil…
  - 7 FLOPs
  - 8 memory references (7 reads, 1 store) per point
  - **AI = 7 / (8*8) = 0.11 FLOPs per byte (measured at the L1)**




```
#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

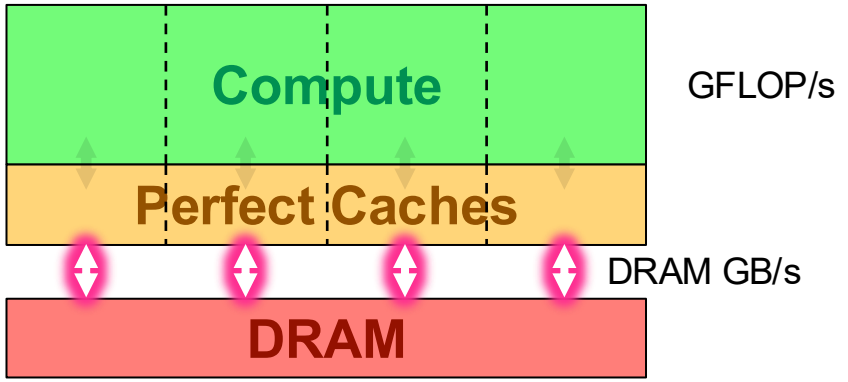
- Conversely, 7-point constant coefficient stencil…
  - 7 FLOPs
  - 8 memory references (7 reads, 1 store) per point
  - **Ideally, cache will filter all but 1 read and 1 write per point**




```
#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)**



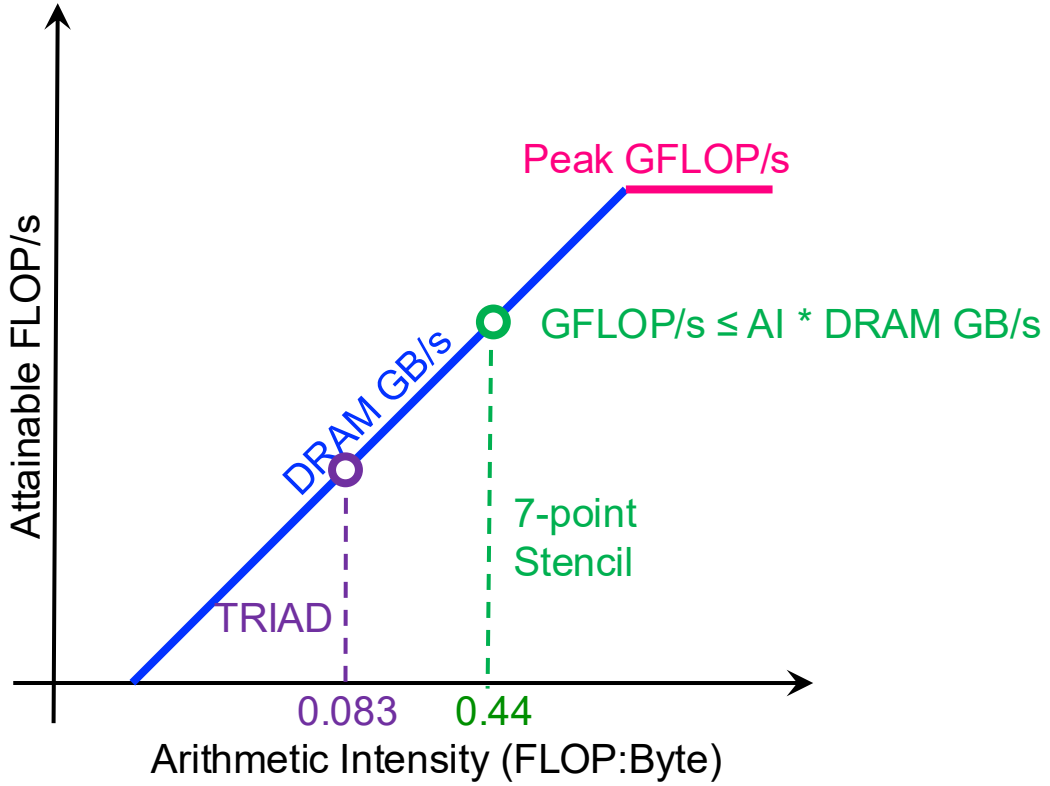

```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                 + old[k  ][j  ][i-1]
                 + old[k  ][j  ][i+1]
                 + old[k  ][j-1][i  ]
                 + old[k  ][j+1][i  ]
            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

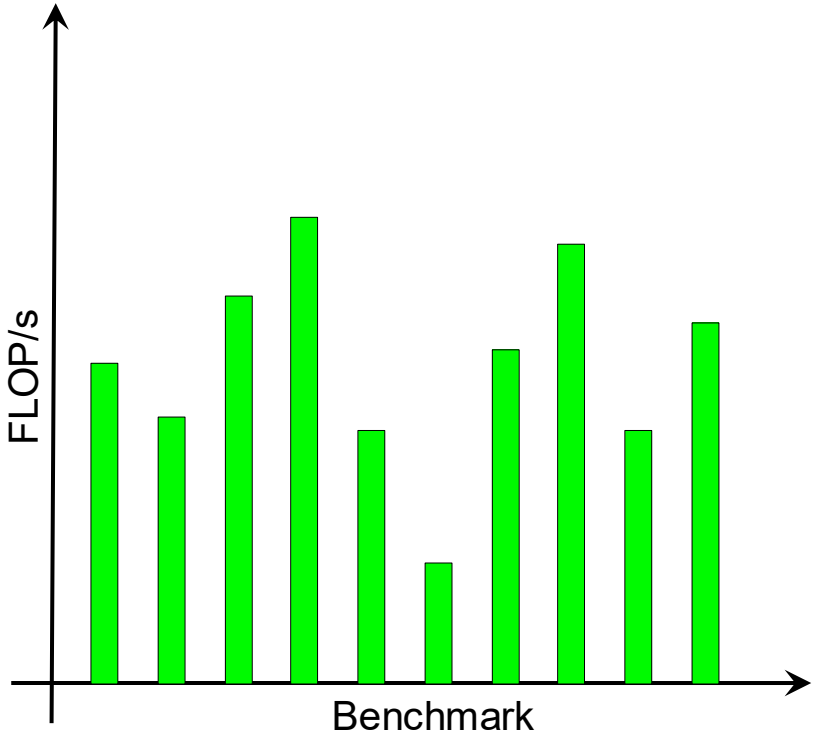
== memory bound, but 5x the FLOP rate as TRIAD

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



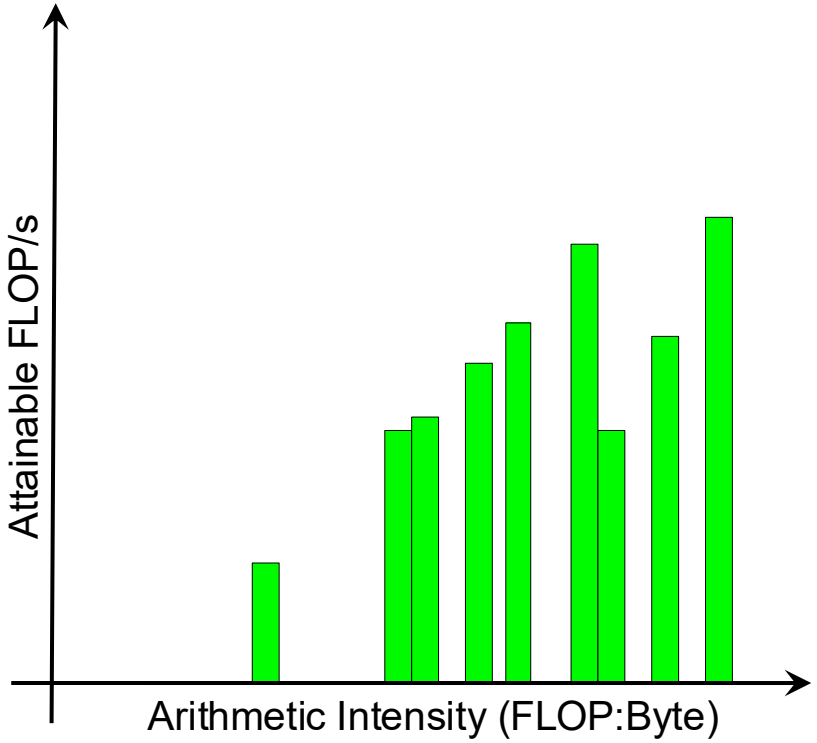
Are we getting good performance?

- Think back to our mix of benchmarks…



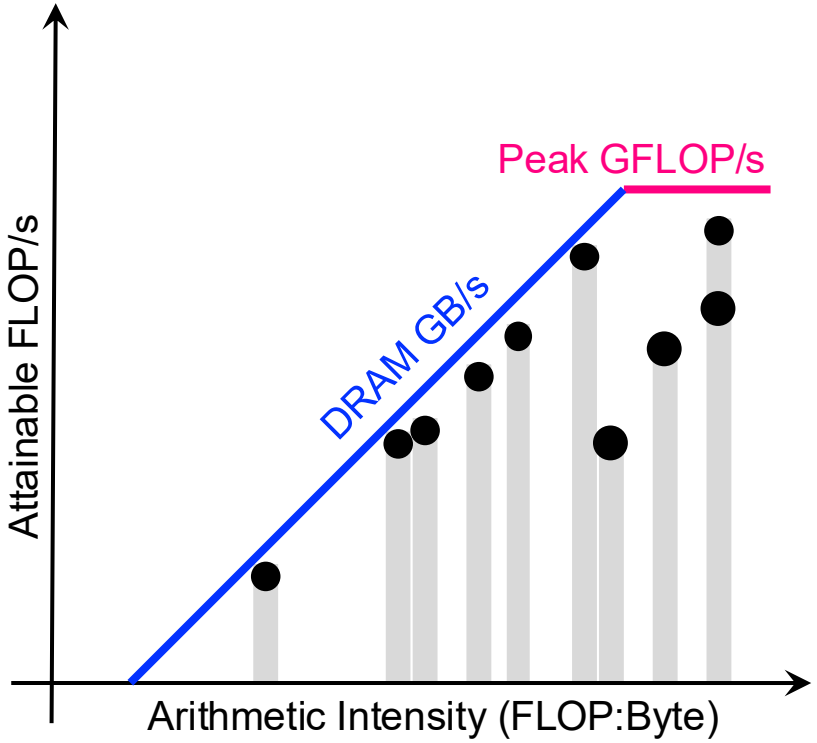
Are we getting good performance?

- We can sort benchmarks by arithmetic intensity…



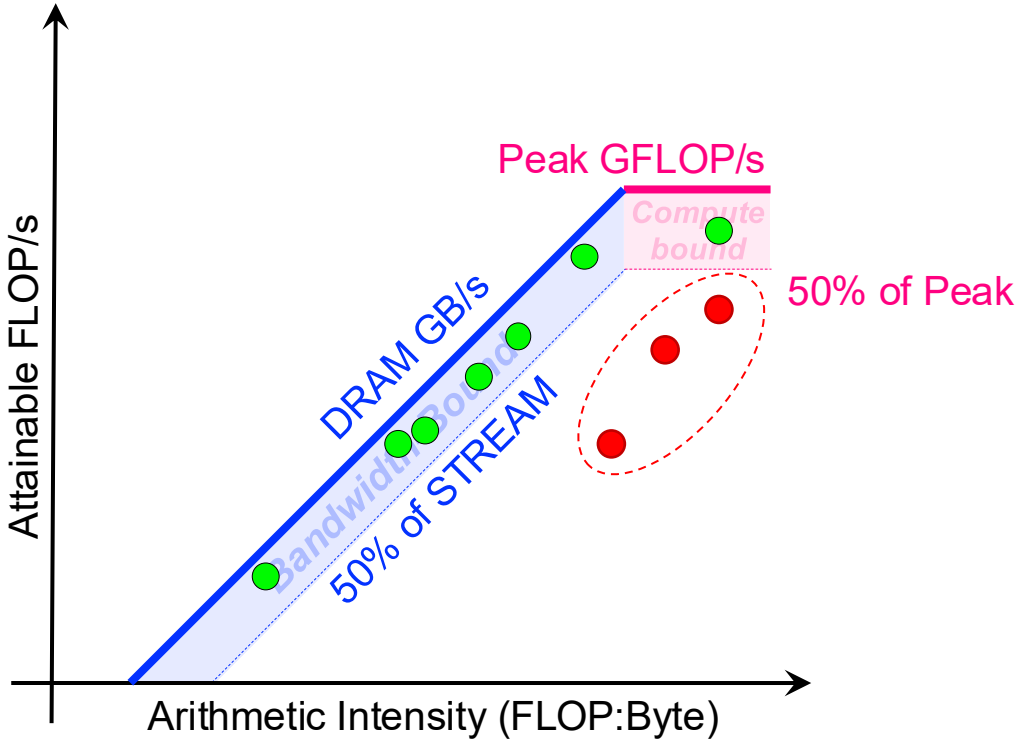
Are we getting good performance?

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



Are we getting good performance?

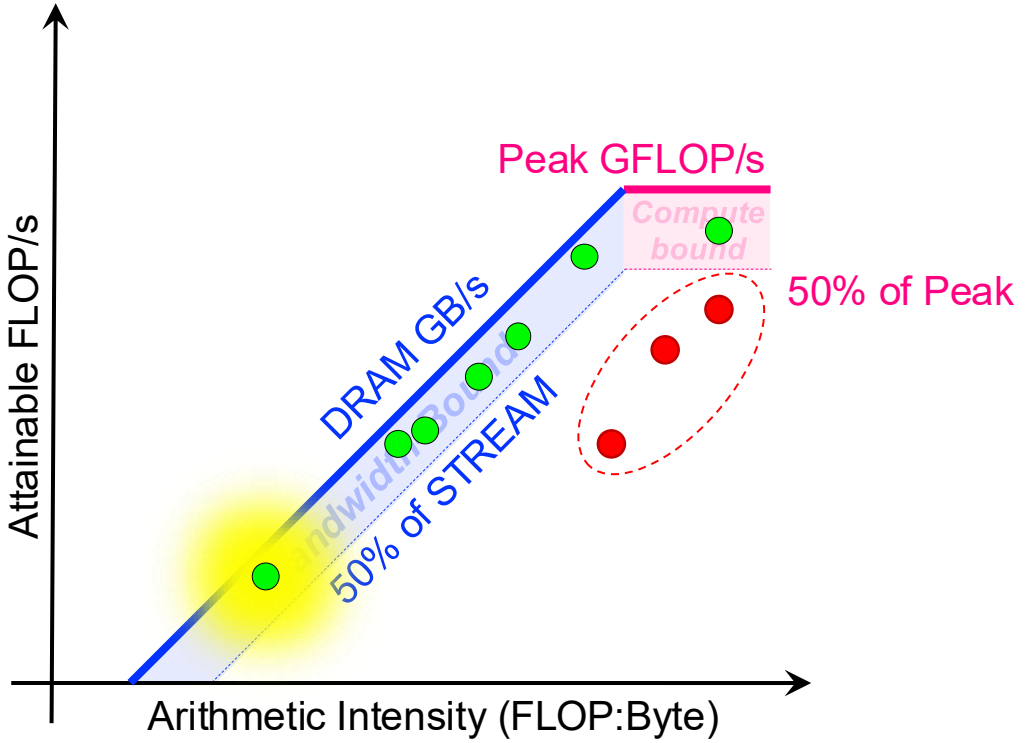
- Benchmarks near the roofline are making **good use** of computational resources



Are we getting good performance?

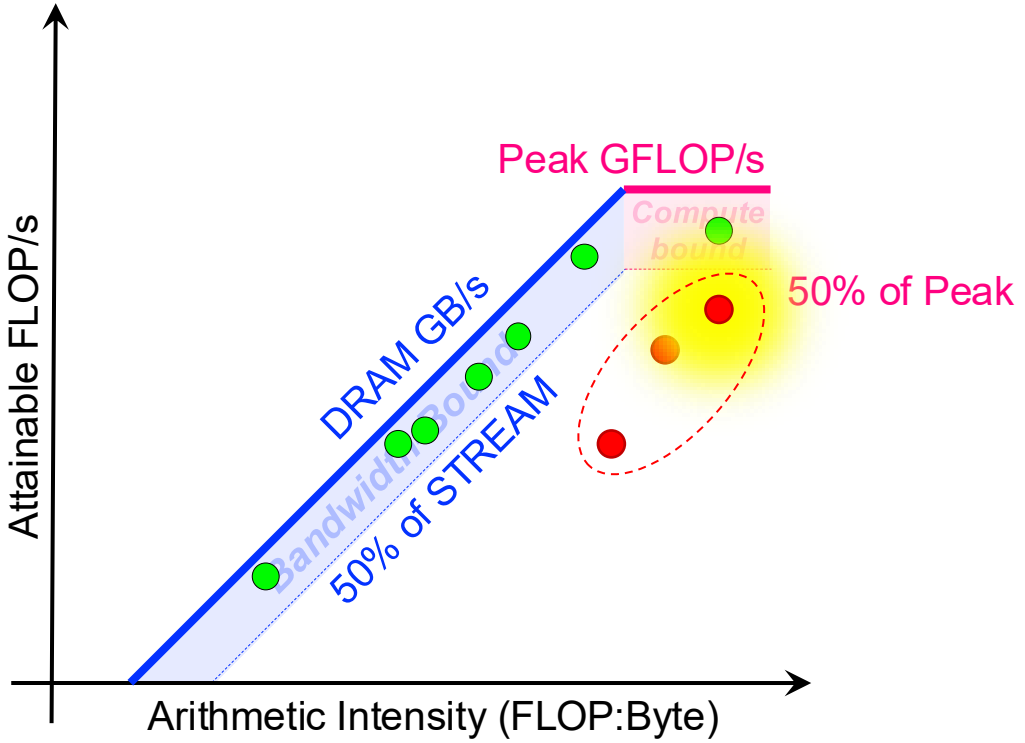
- Benchmarks near the roofline are making **good use** of computational resources

  - benchmarks can have **low performance** (GFLOP/s), but make **good use** (%STREAM) of a machine



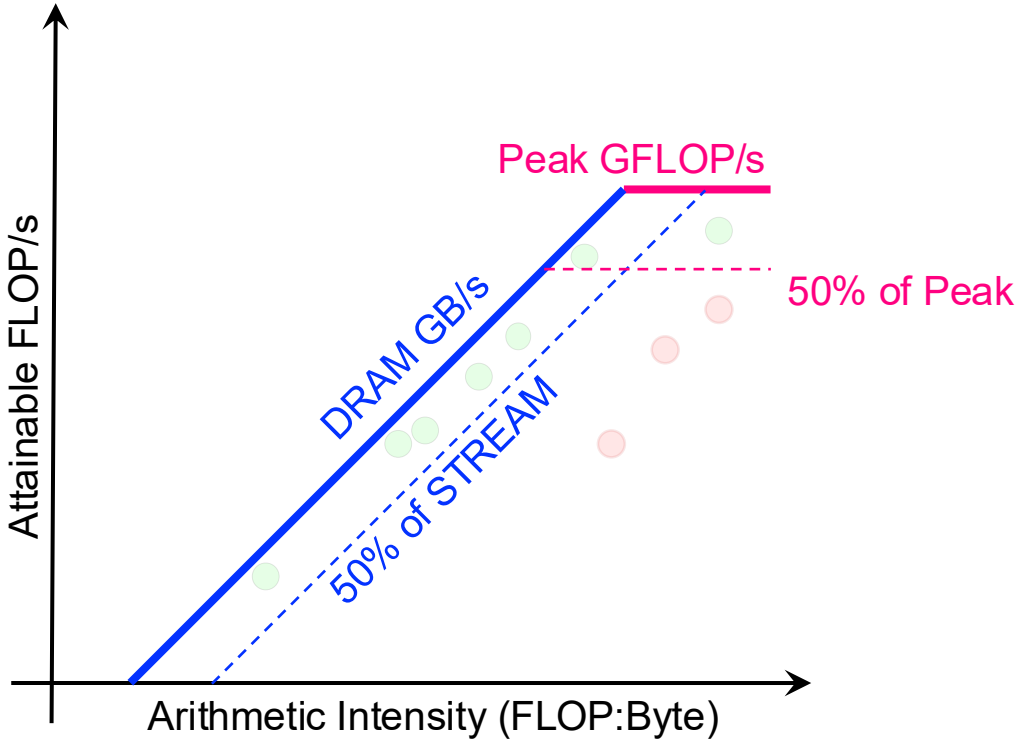
Are we getting good performance?

- 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)



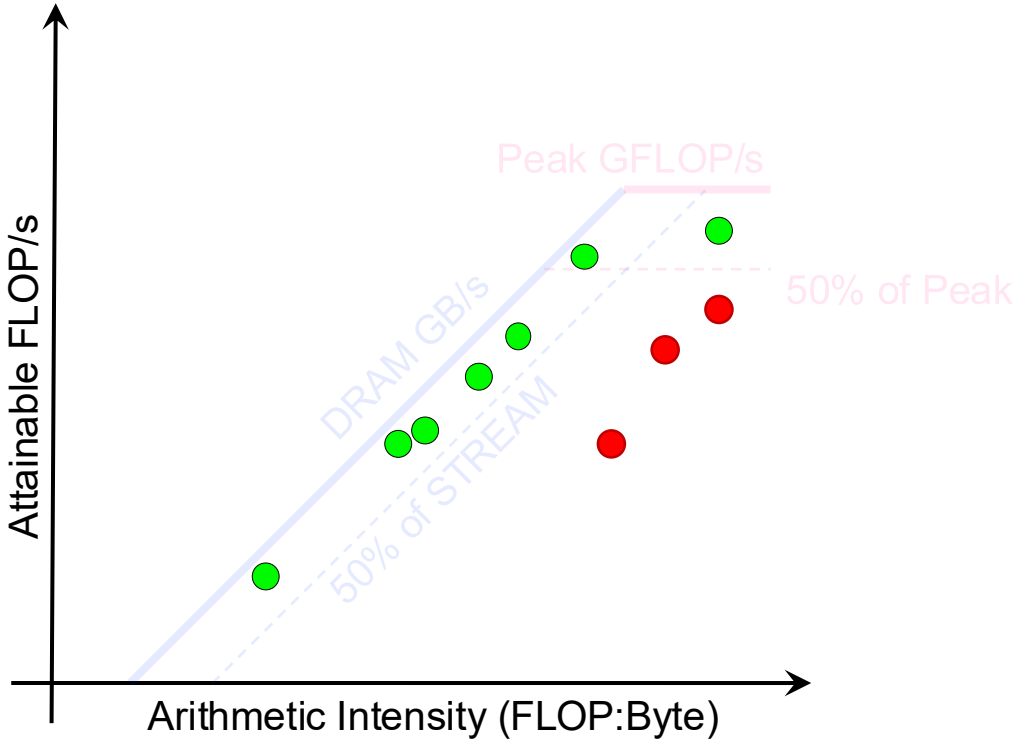
Recap: Roofline is made of two components

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



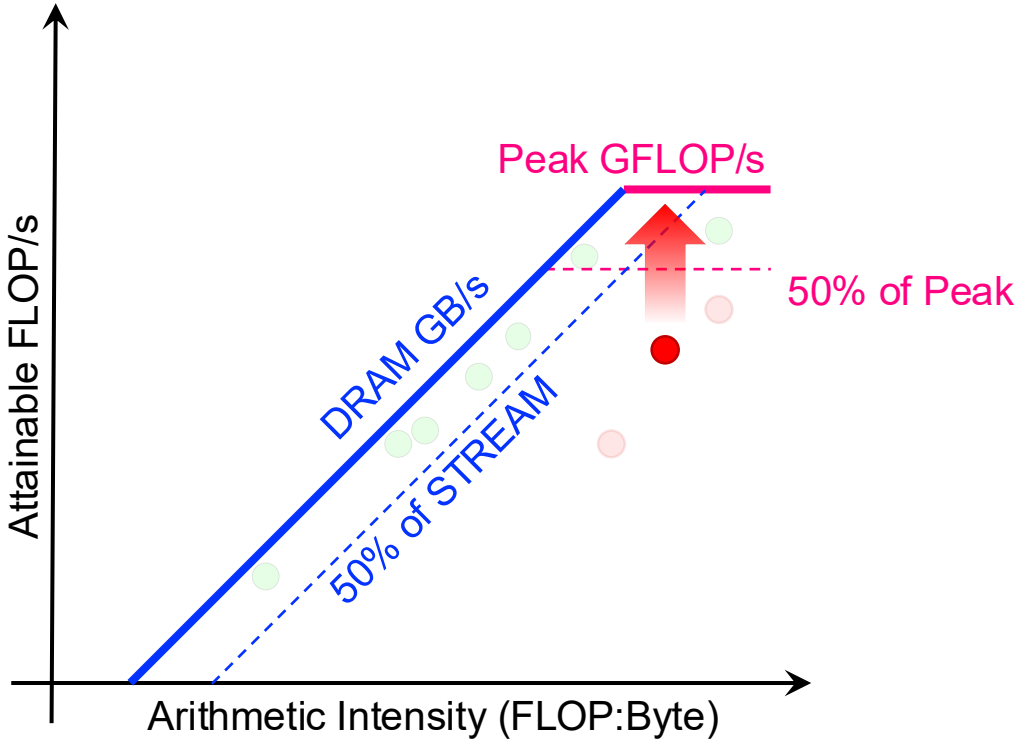
Recap: Roofline is made of two components

- 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



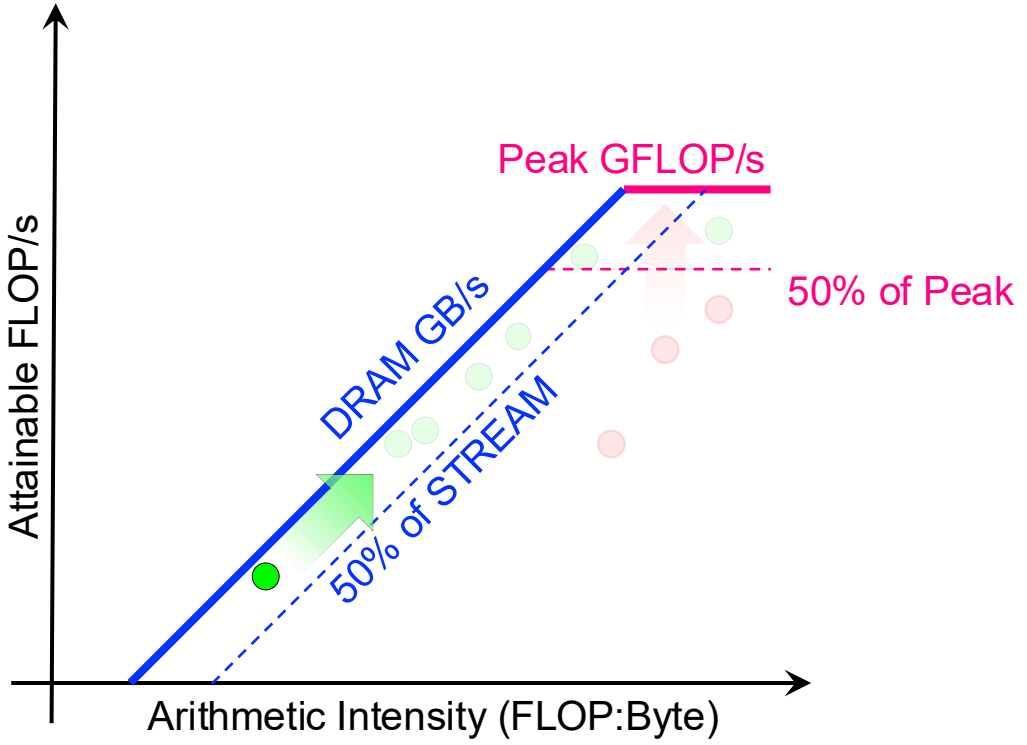
Recap: Optimization Strategy

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



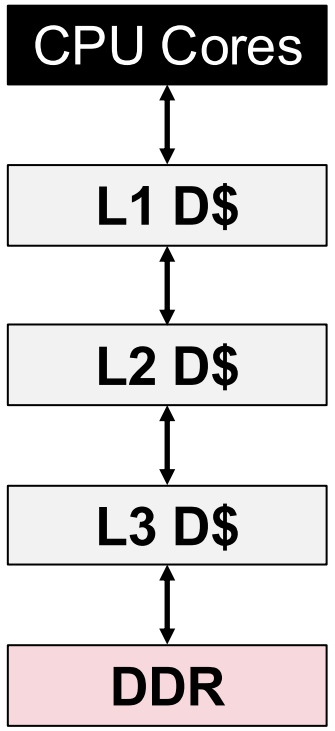
How can performance ever be below the Roofline?

Below the Roofline?

Memory Hierarchy and Cache Bottlenecks

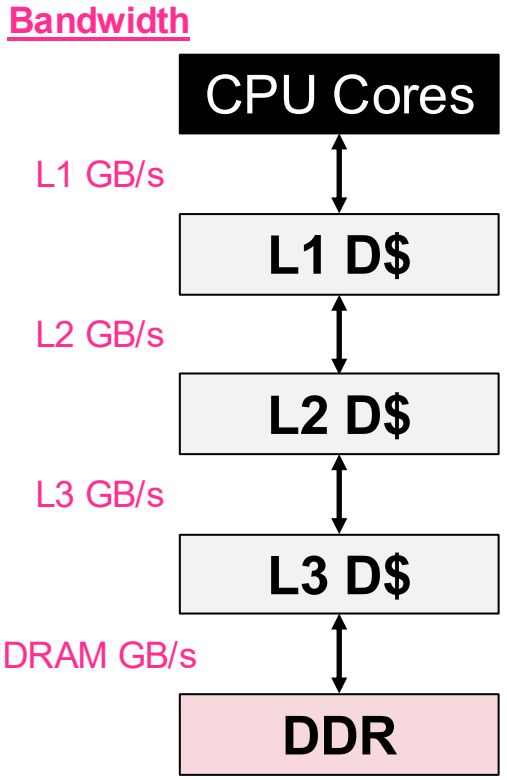
Memory Hierarchy

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



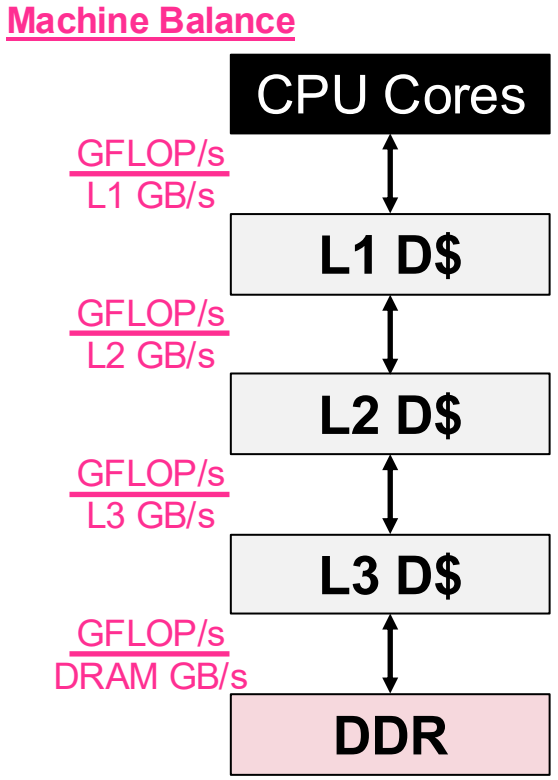
Memory Hierarchy

- CPUs/GPUs have different bandwidths for each level



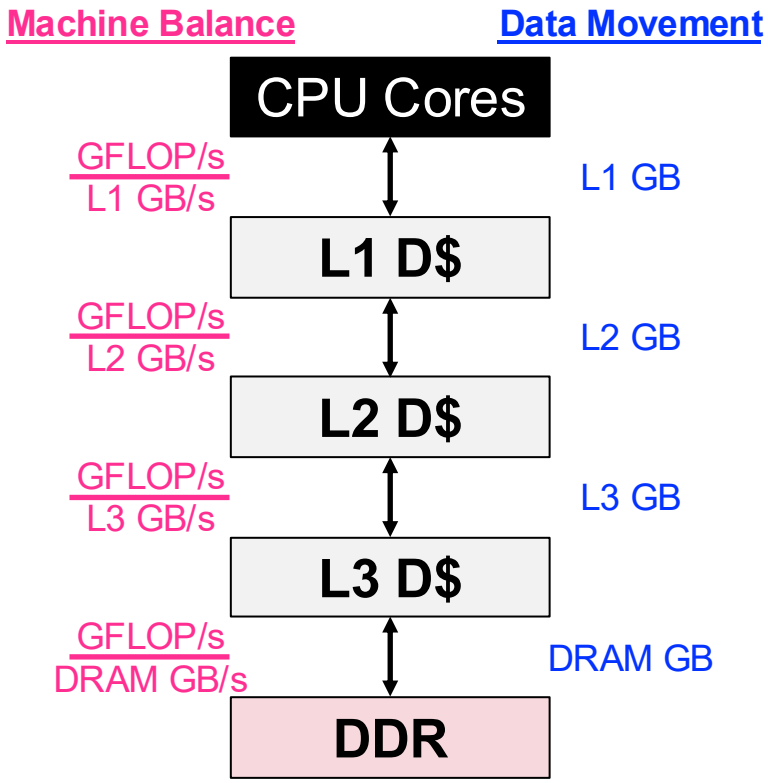
Memory Hierarchy

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



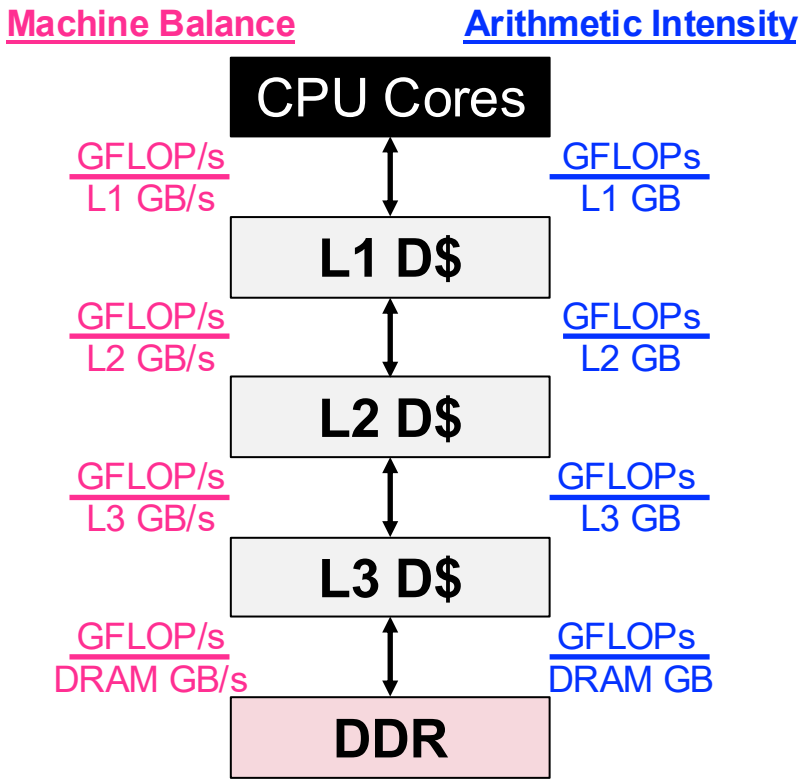
Memory Hierarchy

- 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



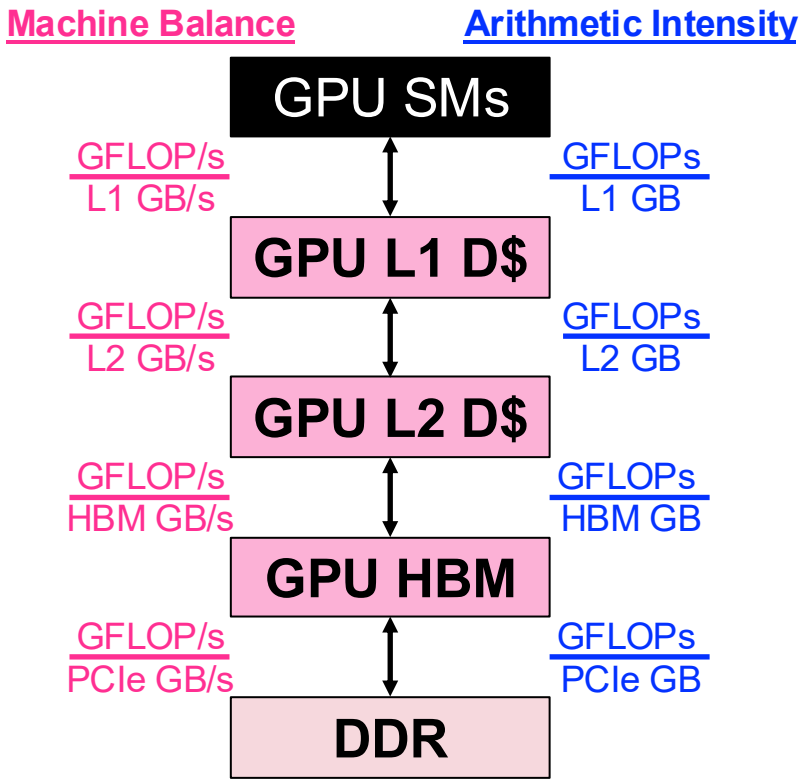
Memory Hierarchy

- 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



Memory Hierarchy (Discrete GPU)

- 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
- Same concept applies to GPUs and disaggregated memory
  - DDR is accessed via **PCIe, or CXL**



Cache Bottlenecks

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

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

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

Cache Bottlenecks

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

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

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

Cache Bottlenecks

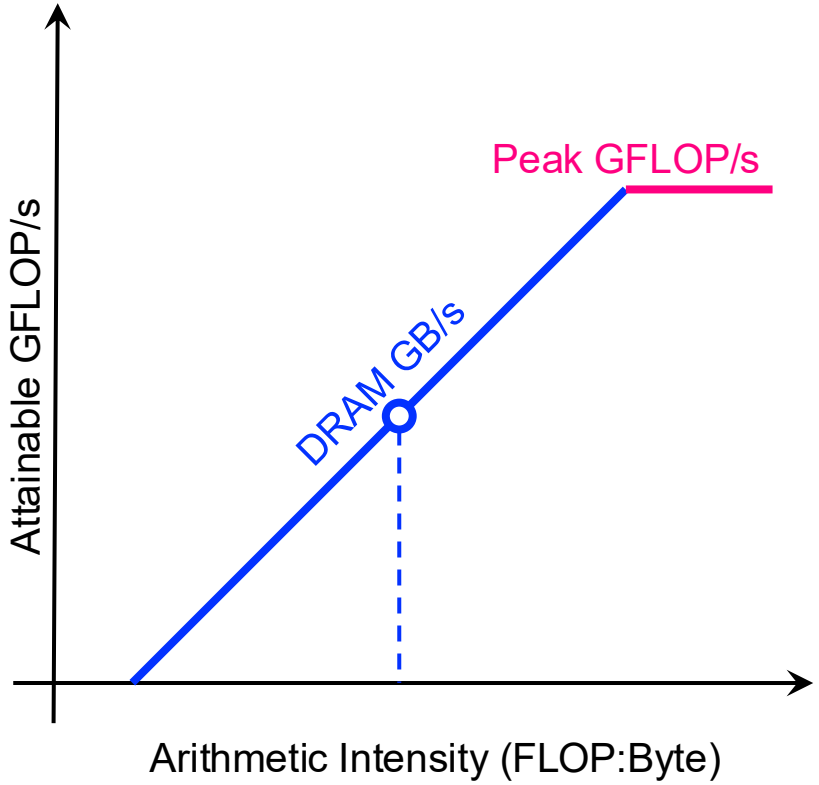
- For each additional level of the memory hierarchy, we can add another term to our model…

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

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

Cache Bottlenecks

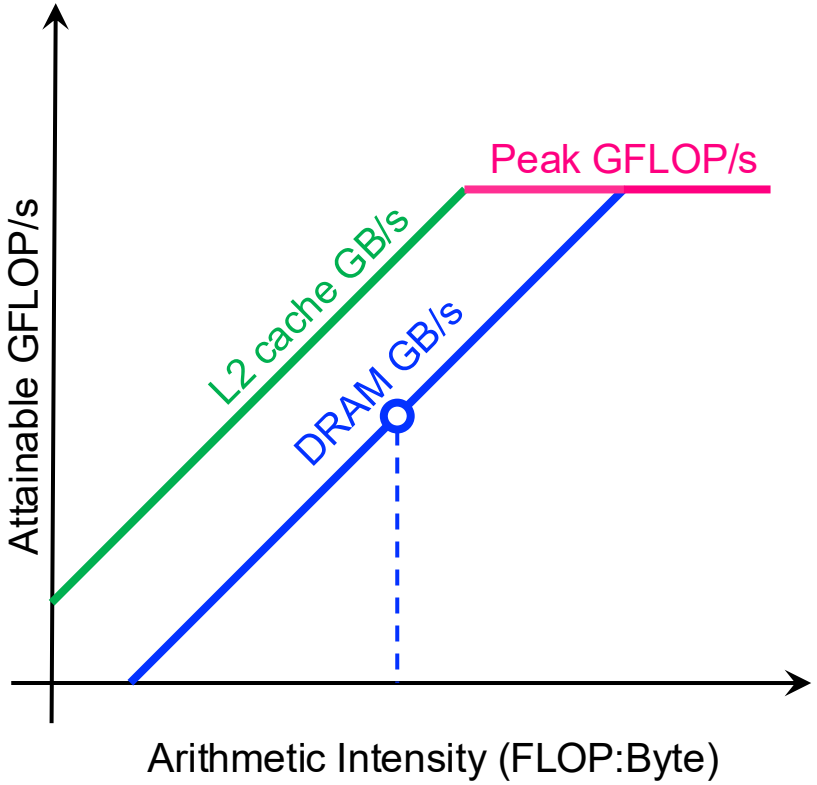
- Plot equation in a single figure…
  - "**Hierarchical Roofline**" Model



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

Cache Bottlenecks

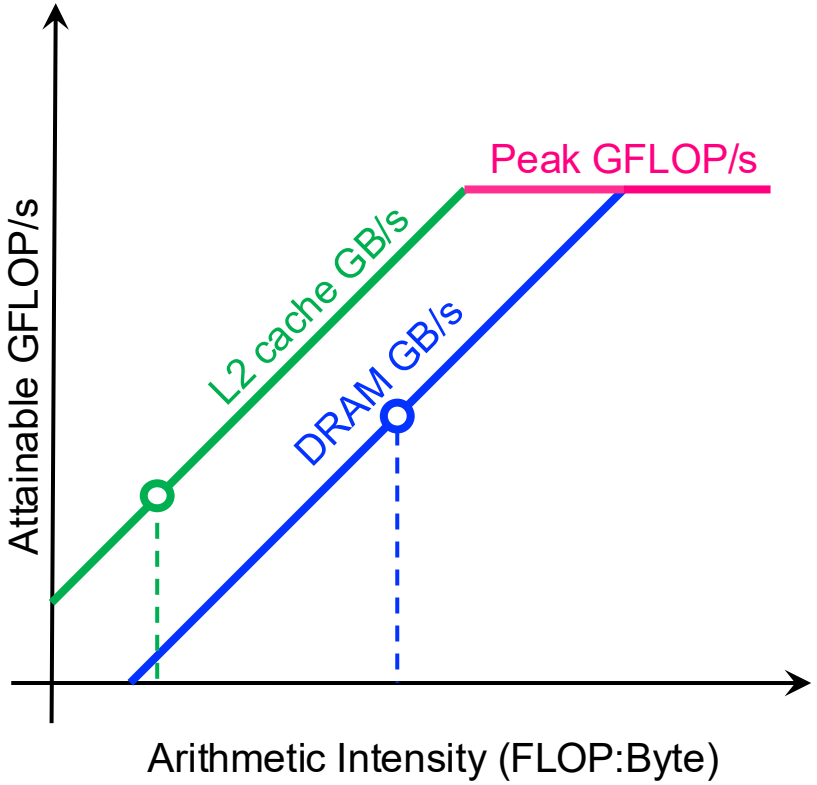
- Plot equation in a single figure…
  - "**Hierarchical Roofline**" Model
  - Bandwidth ceiling (diagonal line) for each level of memory



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

Cache Bottlenecks

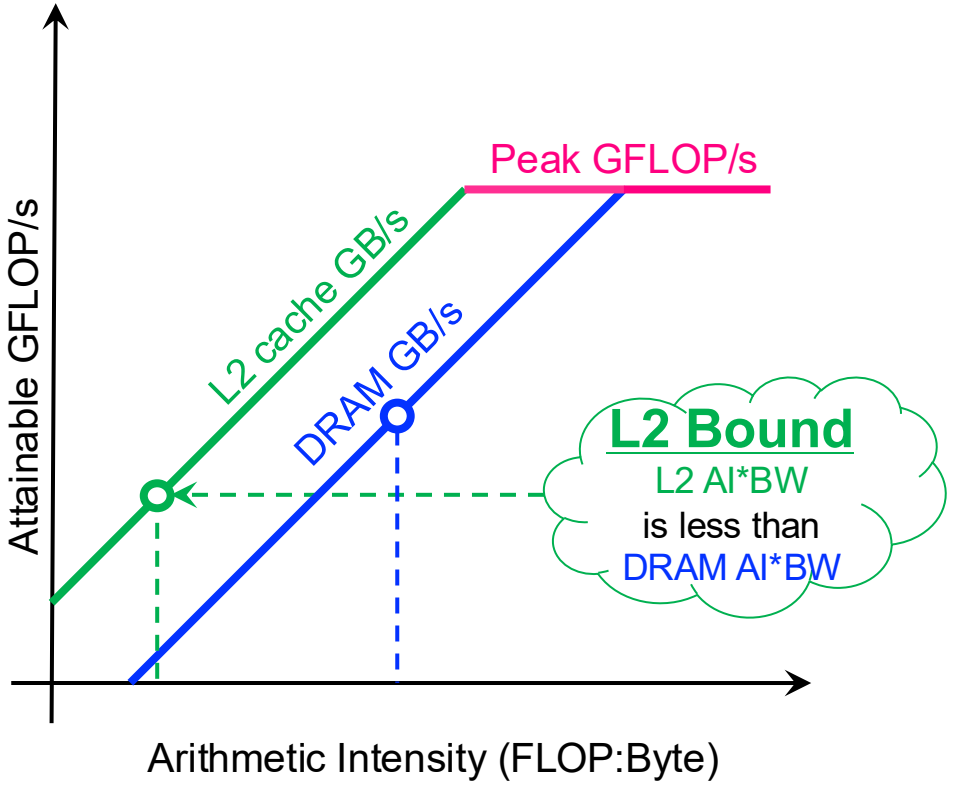
- Plot equation in a single figure…
  - "**Hierarchical Roofline**" Model
  - Bandwidth ceiling (diagonal line) for each level of memory
  - Arithmetic Intensity (dot) for each level of memory



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

Cache Bottlenecks

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

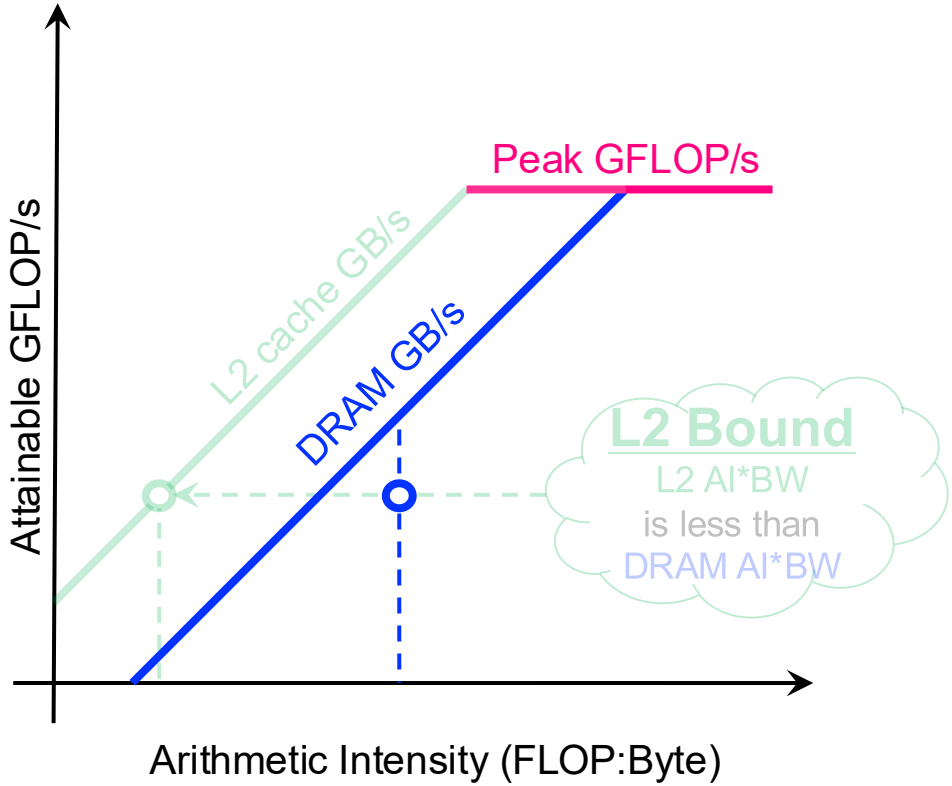


T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, S. Williams, " Novel Multi-Level Integrated Roofline Model Approach for Performance CharacterizatiAon", ISC, 2018.

Cache Bottlenecks

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

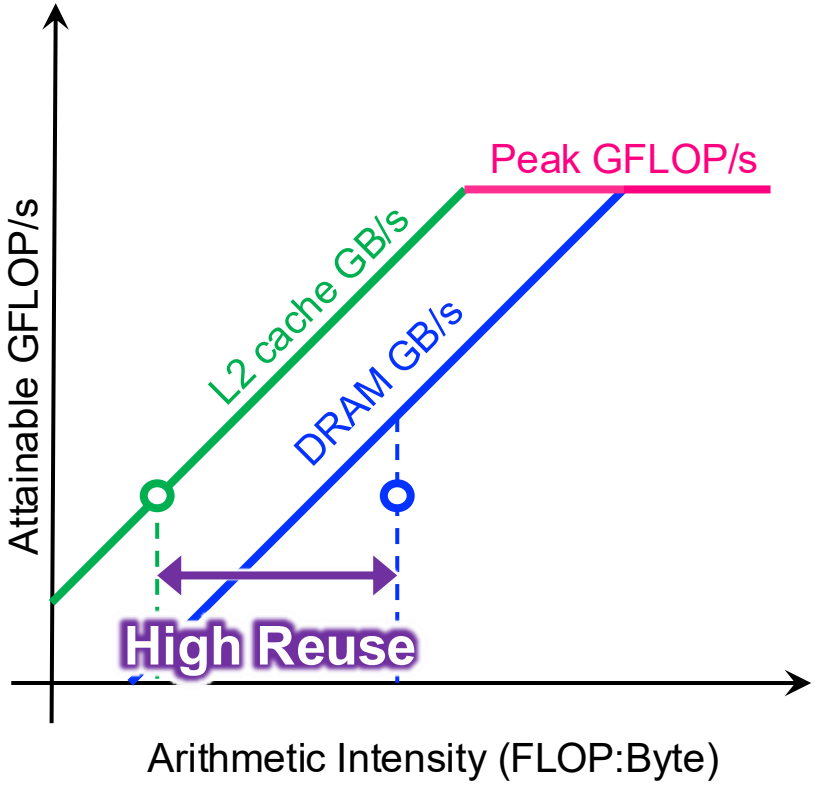
- **If L2 bound, we see DRAM dot well below DRAM ceiling**



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

Cache Hit Rates

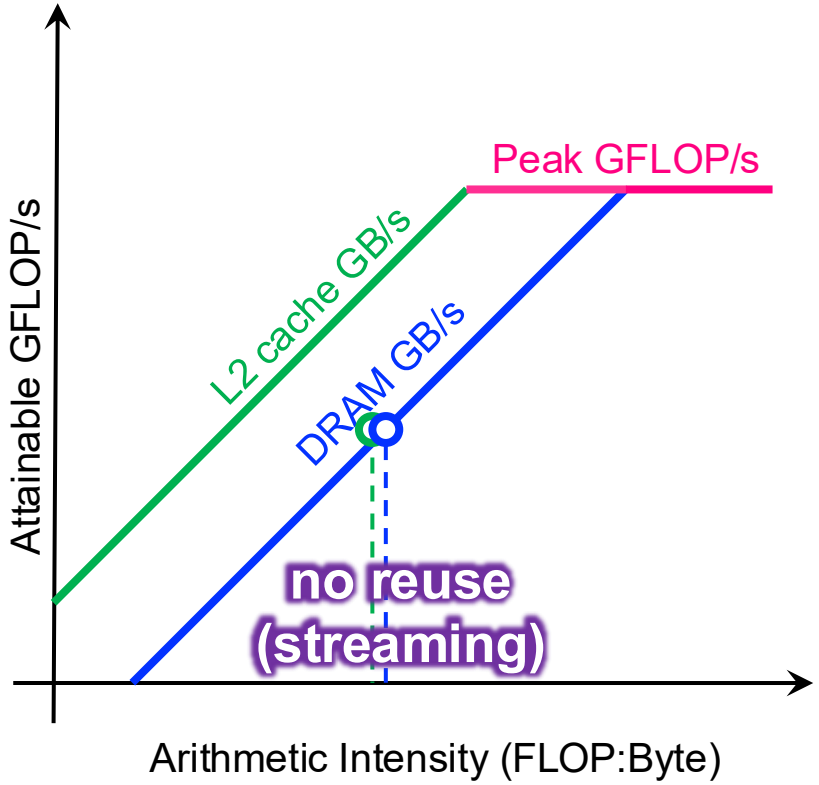
- Widely separated Arithmetic Intensities indicate high reuse in the (L2) cache



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

Cache Hit Rates

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



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

Below the Roofline?

Fused Operations and Accelerators

Fused Operations and Accelerators

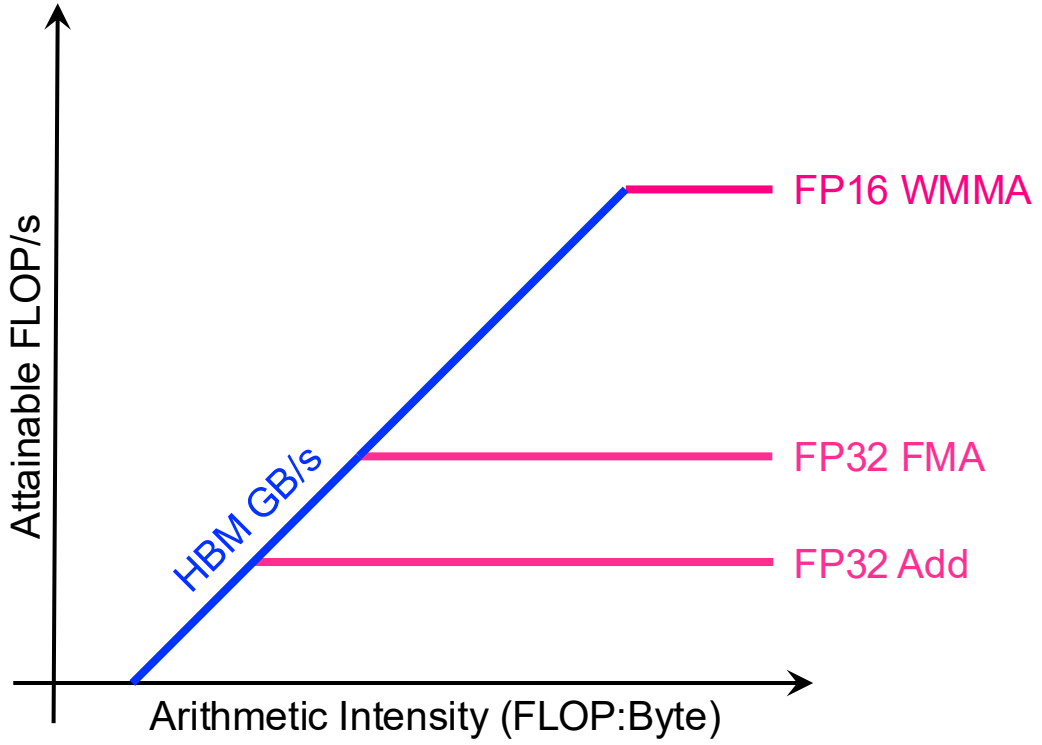
- Vectors have their limits (finite DLP, register file energy scales with VL, etc…)
- Death of Moore's Law is incentivizing operator fusion (e.g. FMA) and compute accelerators (matrix multipliers)

- Modern CPUs and GPUs are increasingly reliant on special (fused) instructions that perform multiple operations (fuse common instruction sequences)…
  - FMA (Fused Multiply Add): z=a*x+y …*z,x,y are vectors or scalars*
  - 4FMA (Quad FMA): z=A*x+z …*A is a FP32 matrix; x,z are vectors*
  - WMMA (Tensor Core): Z=AB+C …*A,B are FP16 matrices; Z,C are FP32*

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

Floating-Point and Mixed Precision Ceilings

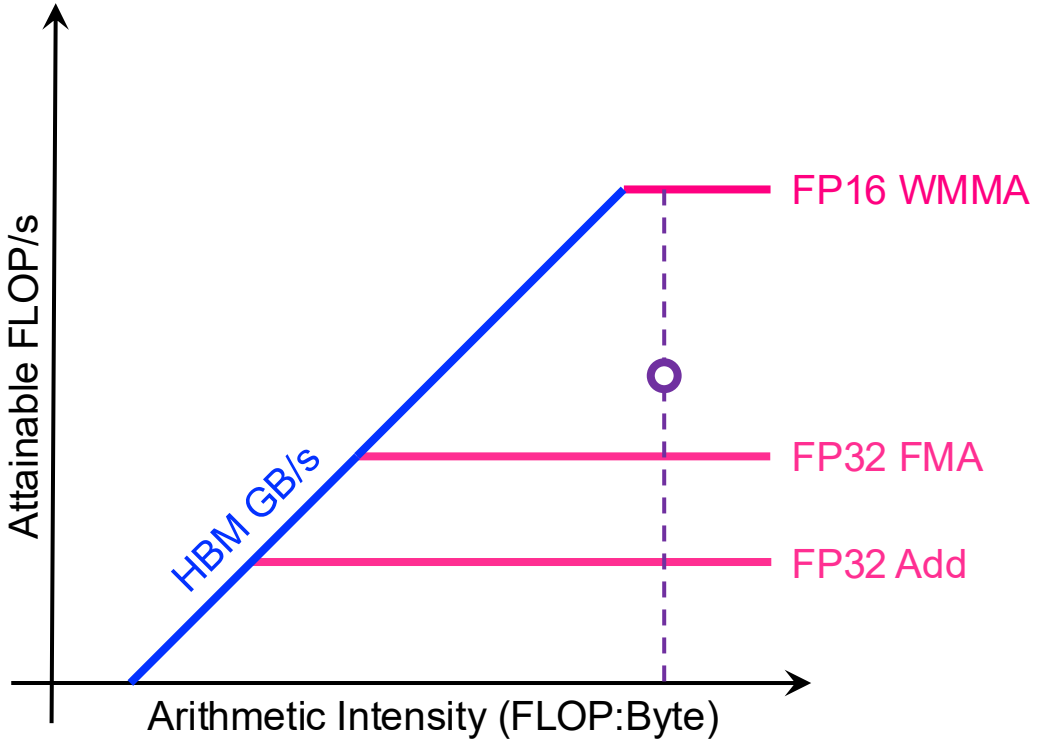
- Consider NVIDIA Volta GPU
- We may define 3 performance ceilings…
  - 15 TFLOPS for FP32 FMA
  - 7.5 TFLOPs for FP32 Add
  - ~100 TFLOPs for FP16 Tensor



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

Floating-Point and Mixed Precision Ceilings

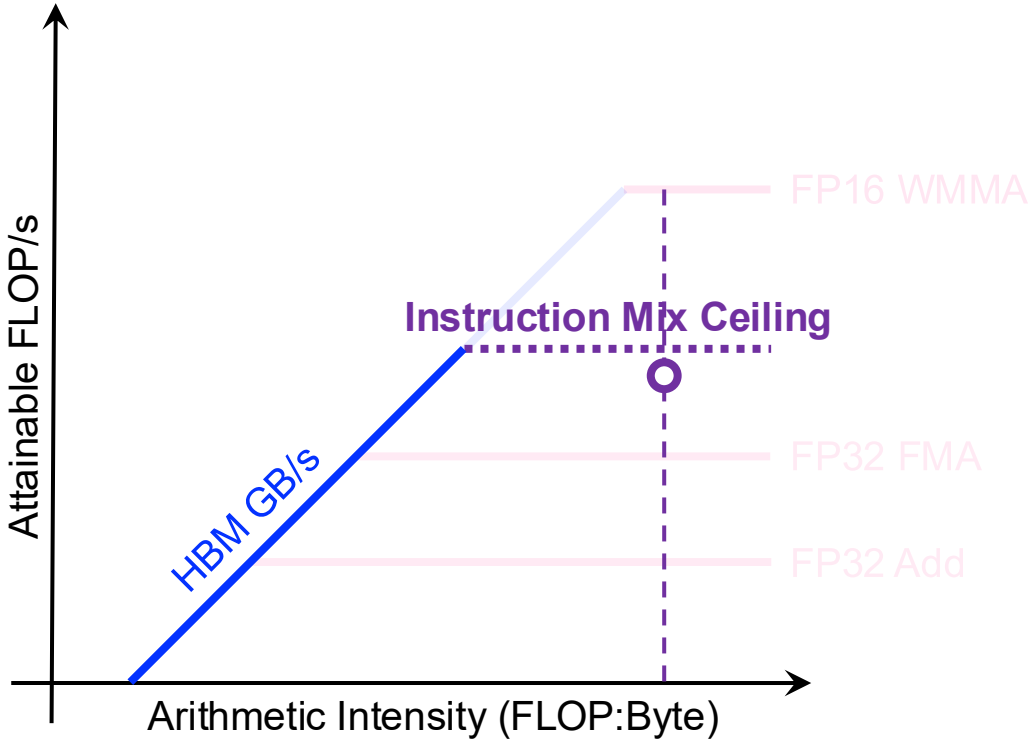
- When calculating (AI,GFLOP/s), count the total FLOPs from all types of instructions
- DL performance can often be well below nominal Tensor Core peak



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

Floating-Point and Mixed Precision Ceilings

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



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

Below the Roofline?

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

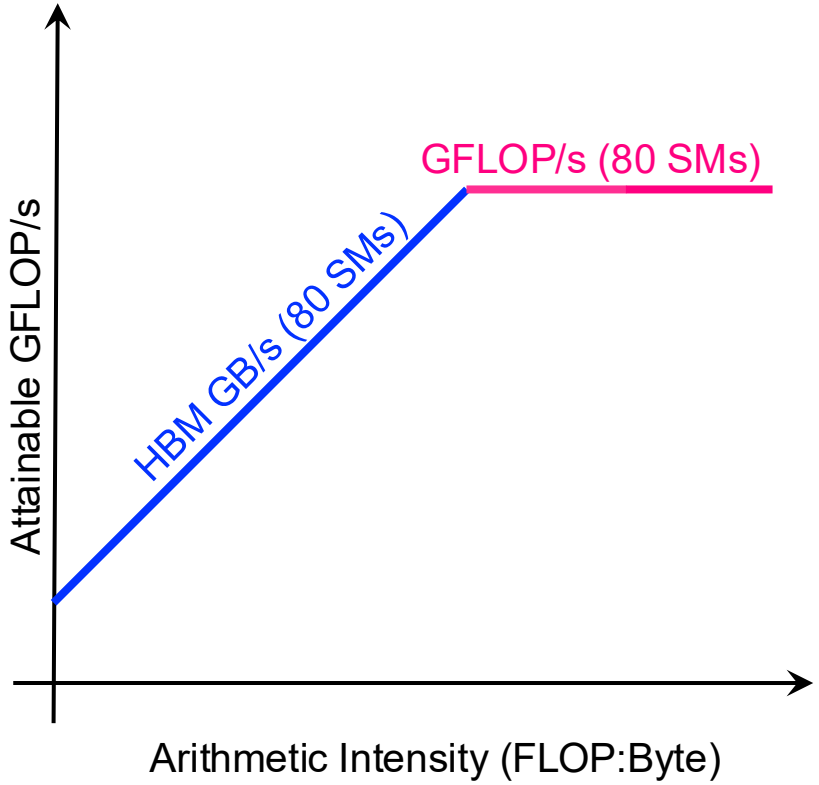
- But all CPUs and GPUs are highly parallel architectures
- GFLOP/s and GB/s are a function of how much parallelism we utilize…

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

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

Roofline and Parallelism

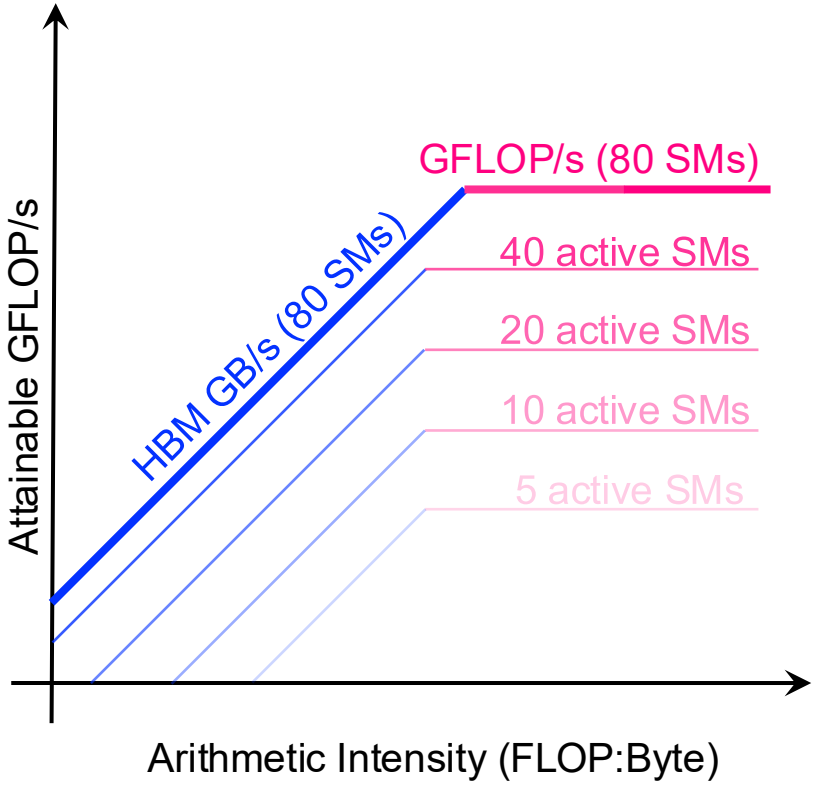
- How do we visualize parallelism in the Roofline?
  - Naively, GFLOP/s(P) and GB/s(P) are proportional to parallelism P
  - SMs are capable of pulling more than their fair share of HBM
  - DVFS implies not true for GFLOP/s



Roofline and Parallelism

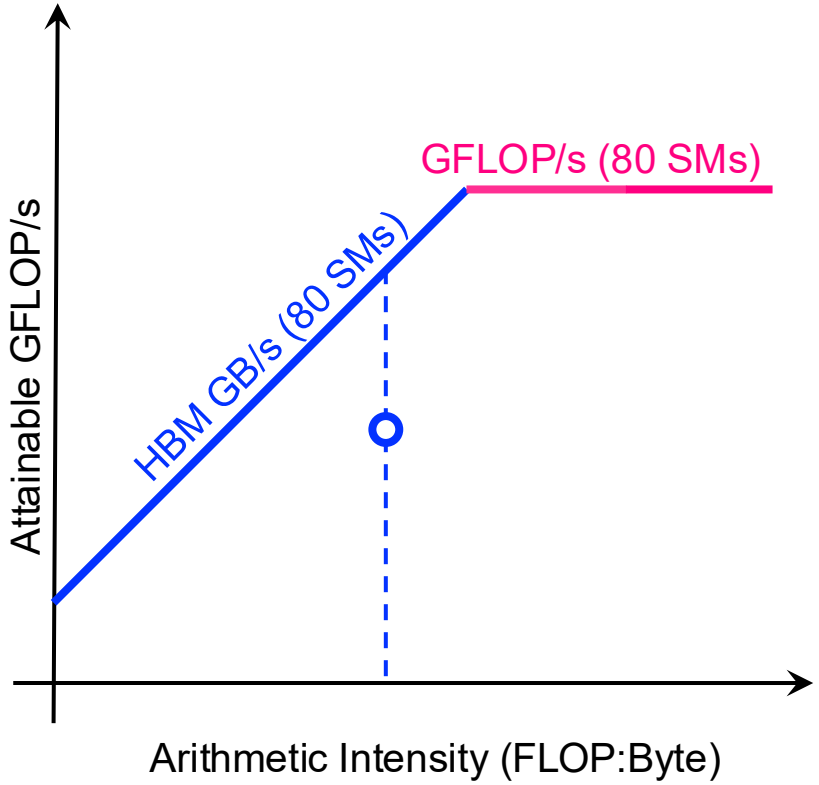
- How do we visualize parallelism in the Roofline?
  - Naively, GFLOP/s(P) and GB/s(P) are proportional to parallelism P
  - SMs are capable of pulling more than their fair share of HBM
  - DVFS implies not true for GFLOP/s

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



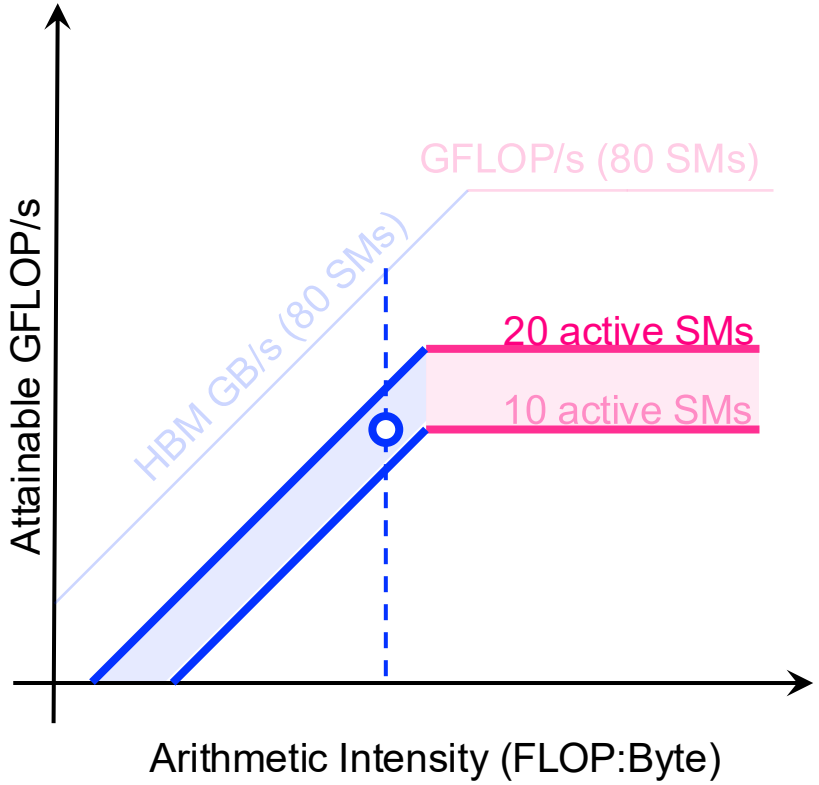
Roofline and Parallelism

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



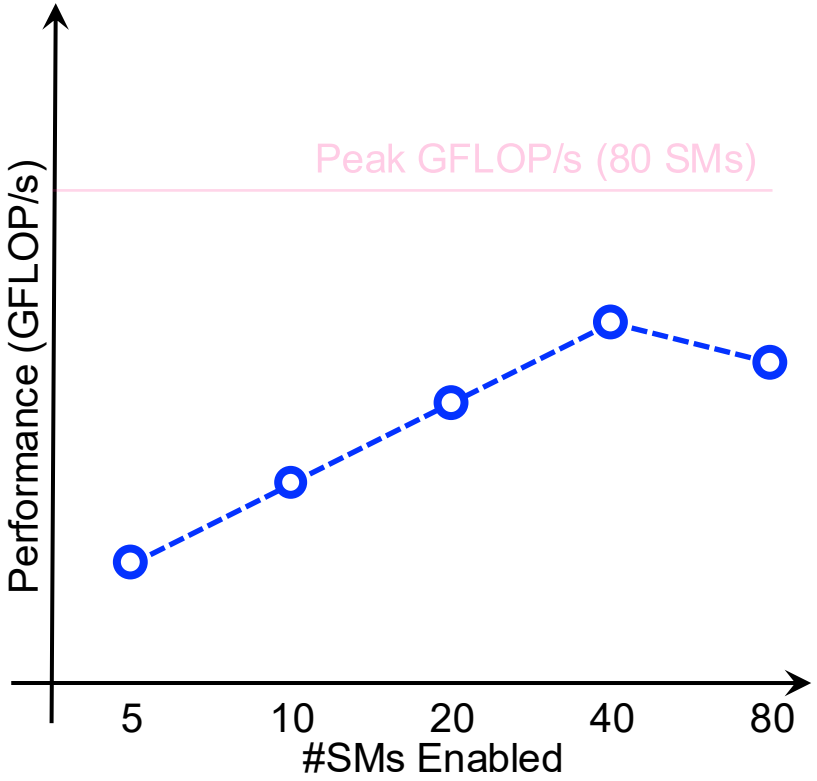
Roofline and Parallelism

- 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?

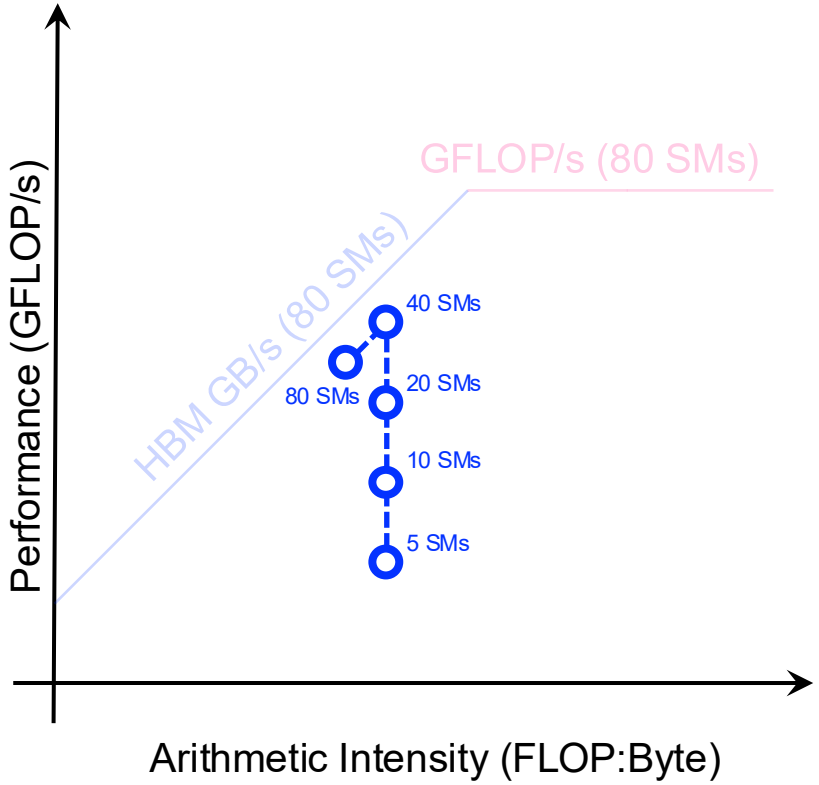


Roofline Scaling Trajectories

- 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

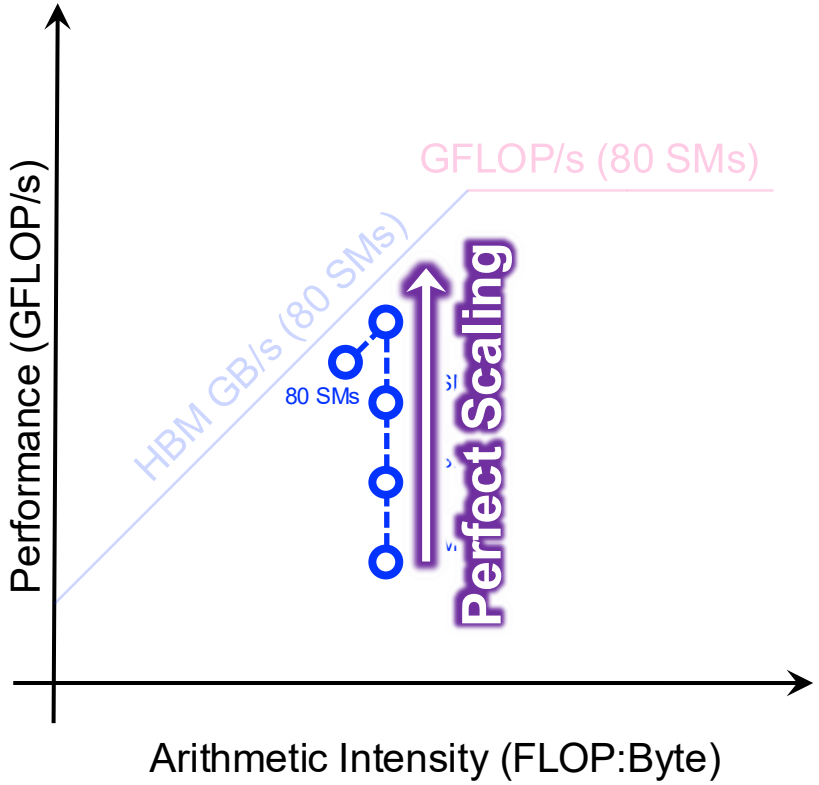


Roofline Scaling Trajectories

- 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**

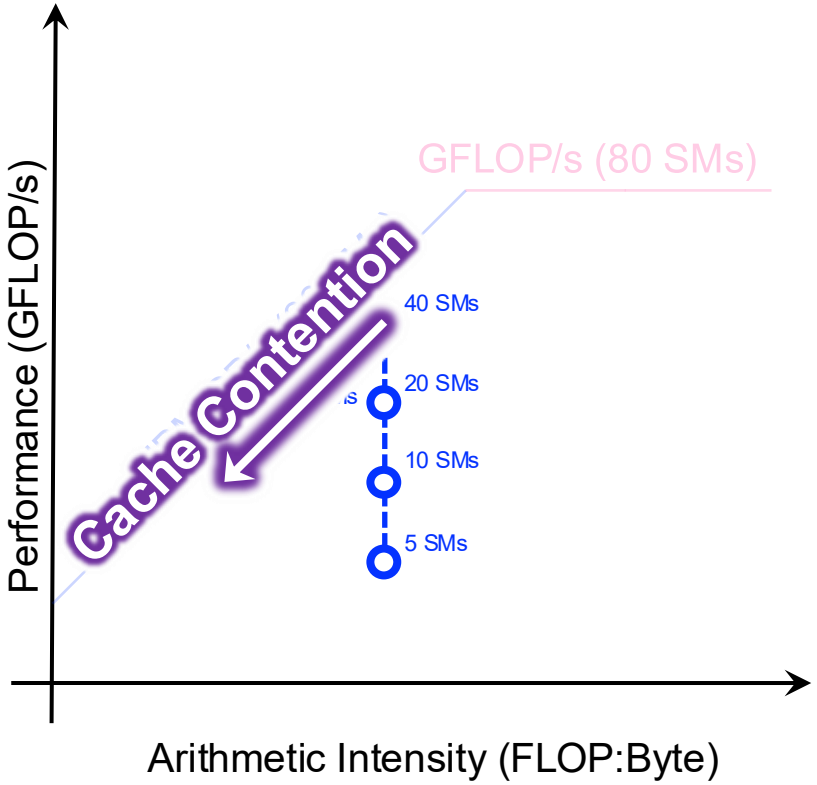


Roofline Scaling Trajectories

- 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)**



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

Hands-on example

Vendor tools for Roofline analysis

- Intel
  - Intel Advisor
    - https://www.intel.com/content/www/us/en/developer/tools/oneapi/advisor.html
  - Aurora Advisor user-guide
    - https://docs.alcf.anl.gov/aurora/performance-tools/advisor/

- NVIDIA
  - NVIDIA Nsight Compute
    - https://developer.nvidia.com/nsight-compute
    - https://docs.nvidia.com/nsight-compute/NsightCompute/index.html#details-page

- AMD
  - AMD ROCm Compute Profiler (Omniperf, previously)
    - https://rocm.docs.amd.com/projects/rocprofiler-compute/en/latest/what-is-rocprof-compute.html

ISO3DFD Code on Aurora

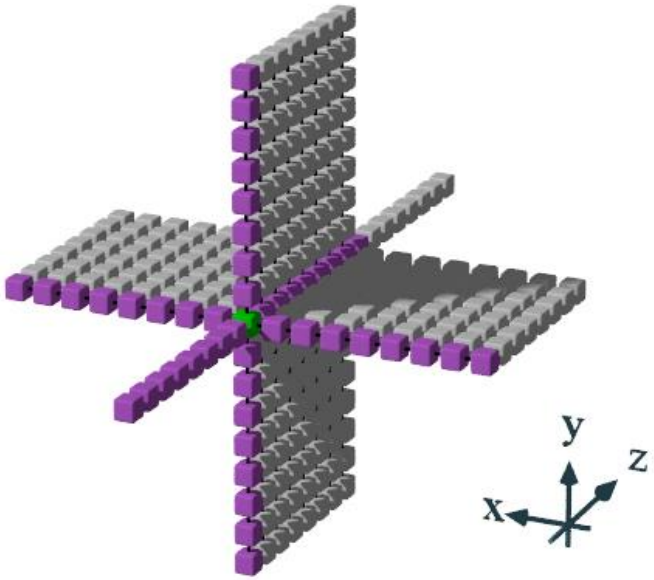
: A 16th order Finite-Difference Stencil for the 3D Isotropic Wave Equation

ISO3dfd code

- A Finite Difference stencil kernel for solving the 3D acoustic isotropic wave equation
  - A proxy for propagating a seismic wave
  - 16th order in space, with symmetric coefficients
  - 2nd order in time scheme without boundary conditions.
- Problem Statement
  - Partial Differential Equation (PDE) for wave propagation

$$\frac{d^2p}{dt^2} = v^2\left(\frac{d^2p}{dx^2} + \frac{d^2p}{dy^2} + \frac{d^2p}{dz^2}\right)$$

  - , where p is pressure, v is velocity, and t is time.
  - For a 16th order finite difference stencil, we use the adjacent 8 values in the x, y, and z axis
  - The expanded equation looks like this, where the array C holds the coefficients for the changes in x, y and z.

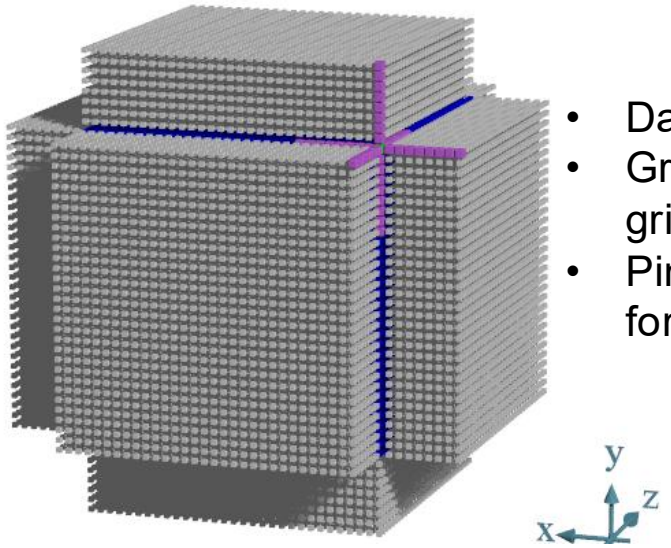


$$p^{n+1}_{i,j,k} = C[0]p^n_{i,j,k} + C[1]\left(p^n_{i+1,j,k} + p^n_{i-1,j,k} + p^n_{i,j+1,k} + p^n_{i,j-1,k} + p^n_{i,j,k+1} + p^n_{i,j,k-1}\right) + \cdots$$
$$+ C[8]\left(p^n_{i+8,j,k} + p^n_{i-8,j,k} + p^n_{i,j+8,k} + p^n_{i,j-8,k} + p^n_{i,j,k+8} + p^n_{i,j,k-8}\right)$$

https://www.intel.com/content/www/us/en/developer/articles/technical/iso3dfd-code-walkthrough.html

Code walk through

- Variables
  - n1 n2 n3 : Grid dimensions for the stencil
  - Iterations : No. of timesteps.
  - n1, n2, and n3 has the addition of 2*kHalfLength to represent the entire block including the halo region.



- Dark blue: grid
- Gray: halo of the grid
- Pink: points needed for calculation

```
…
 try {
    // Parse command line arguments and increase them by HALO
    n1 = std::stoi(argv[1]) + (2 * kHalfLength);
    n2 = std::stoi(argv[2]) + (2 * kHalfLength);
    n3 = std::stoi(argv[3]) + (2 * kHalfLength);
    num_iterations = std::stoi(argv[4]);
  } catch (...) {
    Usage(argv[0]);
    return 1;
  }
…
 // Compute the total size of grid
  size_t nsize = n1 * n2 * n3;
…

  // Apply the DX, DY and DZ to coefficients
  coeff[0] = (3.0f * coeff[0]) / (dxyz * dxyz);
  for (auto i = 1; i <= kHalfLength; i++) {
    coeff[i] = coeff[i] / (dxyz * dxyz);
  }
```

- In this hands-on, we use one ISO3DFD variant: **3_GPU_linear.cpp** (reduced index calculation)

https://github.com/oneapi-src/oneAPI-samples/tree/master/DirectProgramming/C%2B%2BSYCL/StructuredGrids/guided_iso3dfd_GPUOptimization

Build the code

```
$ qsub -I -l select=1 -l walltime=1:00:00 -l filesystems=home:flare  -A ATPESC2026 -q ATPESC

$ cd /flare/ATPESC2026/usr/$USER

$ git clone https://github.com/oneapi-src/oneAPI-samples.git
$ cd oneAPI-samples/DirectProgramming/C++SYCL/StructuredGrids/guided_iso3dfd_GPUOptimization/
or
$ cp -r /flare/ATPESC2026/EXAMPLES/track-5-HPC-tools-and-debug/roofline/oneAPI-
samples/DirectProgramming/C++SYCL/StructuredGrids/guided_iso3dfd_GPUOptimization/ .
$ cd guided_iso3dfd_GPUOptimization

$ mkdir build
$ cd build

$ module load cmake
$ cmake ..
$ make
```

3_GPU_linear: using linearized index to reduce index calculation

```
// Send a SYCL kernel(lambda) to the device for parallel execution
     // Each kernel runs single cell
     h.parallel_for(kernel_range, [=](id<3> nidx) {
       // Start of device code
       // Add offsets to indices to exclude HALO
       int n2n3 = n2 * n3;
       int i = nidx[0] + kHalfLength;
       int j = nidx[1] + kHalfLength;
       int k = nidx[2] + kHalfLength;

       // Calculate linear index for each cell
       int idx = i * n2n3 + j * n3 + k;

       // Calculate values for each cell
       float value = prev_acc[idx] * coeff_acc[0];
#pragma unroll(8)
       for (int x = 1; x <= kHalfLength; x++) {
         value +=
             coeff_acc[x] * (prev_acc[idx + x]        + prev_acc[idx - x] +
                             prev_acc[idx + x * n3]   + prev_acc[idx - x * n3] +
                             prev_acc[idx + x * n2n3] + prev_acc[idx - x * n2n3]);
       }
       next_acc[idx] = 2.0f * prev_acc[idx] - next_acc[idx] +
                            value * vel_acc[idx];
       // End of device code
     });
```

Linearized index

Computing values for each cell

SYCL kernel to the device for parallel execution over x, y, and z

Run 3_GPU_linear

$ export ZE_AFFINITY_MASK=0.0
$ src/3_GPU_linear 512 512 512 100

```
 Running linear indexed GPU version
 Running on Intel(R) Data Center GPU Max 1550
 The Device Max Work Group Size is : 1024
 The Device Max EUCount is : 448
-------------------------------------
time          : 0.854 secs
throughput    : 15716.4 Mpts/s
flops         : 958.698 GFlops
bytes         : 188.596 GBytes/s
-------------------------------------
```

$ advisor -collect roofline --profile-gpu --project-dir ADV_03_512 -- ./src/3_GPU_linear 512 512 512 100

or

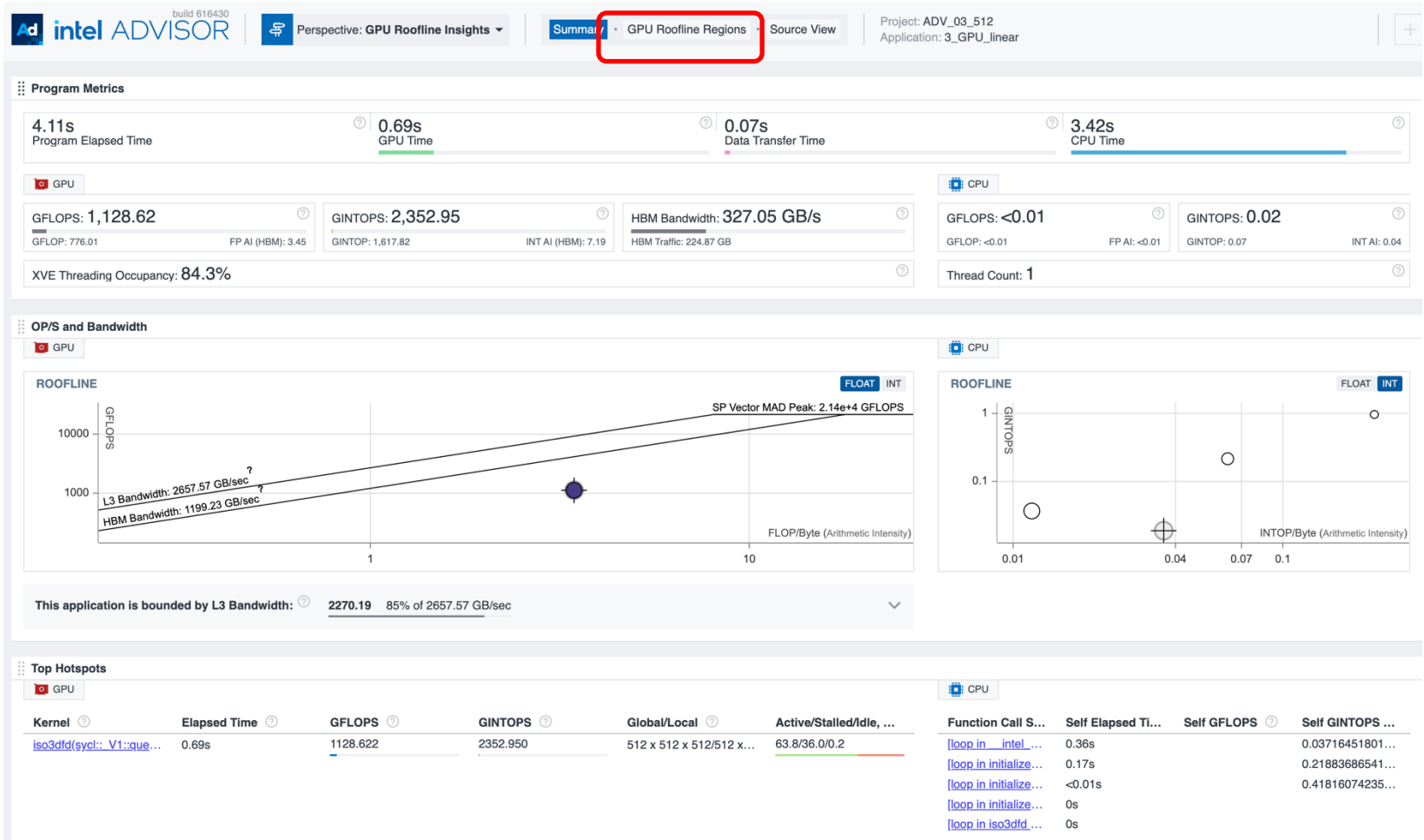
$ cp /flare/ATPESC2026/EXAMPLES/track-5-HPC-tools-and-debug/roofline/ADV_results/ADV_03_512 .

 # Download advisor-report.html from /ADV_03_512/e000/report

Orient yourself in GPU+CPU Roofline!

- **Quiz**: What are GFLOPS of CPU and GPU?
- **Quiz**: What is a cumulative application bottleneck (Bounded by)?
- **Quiz**: Do you see iso3dfd kernel?

- Now, let's switch to the **main page ("GPU Roofline Regions")**

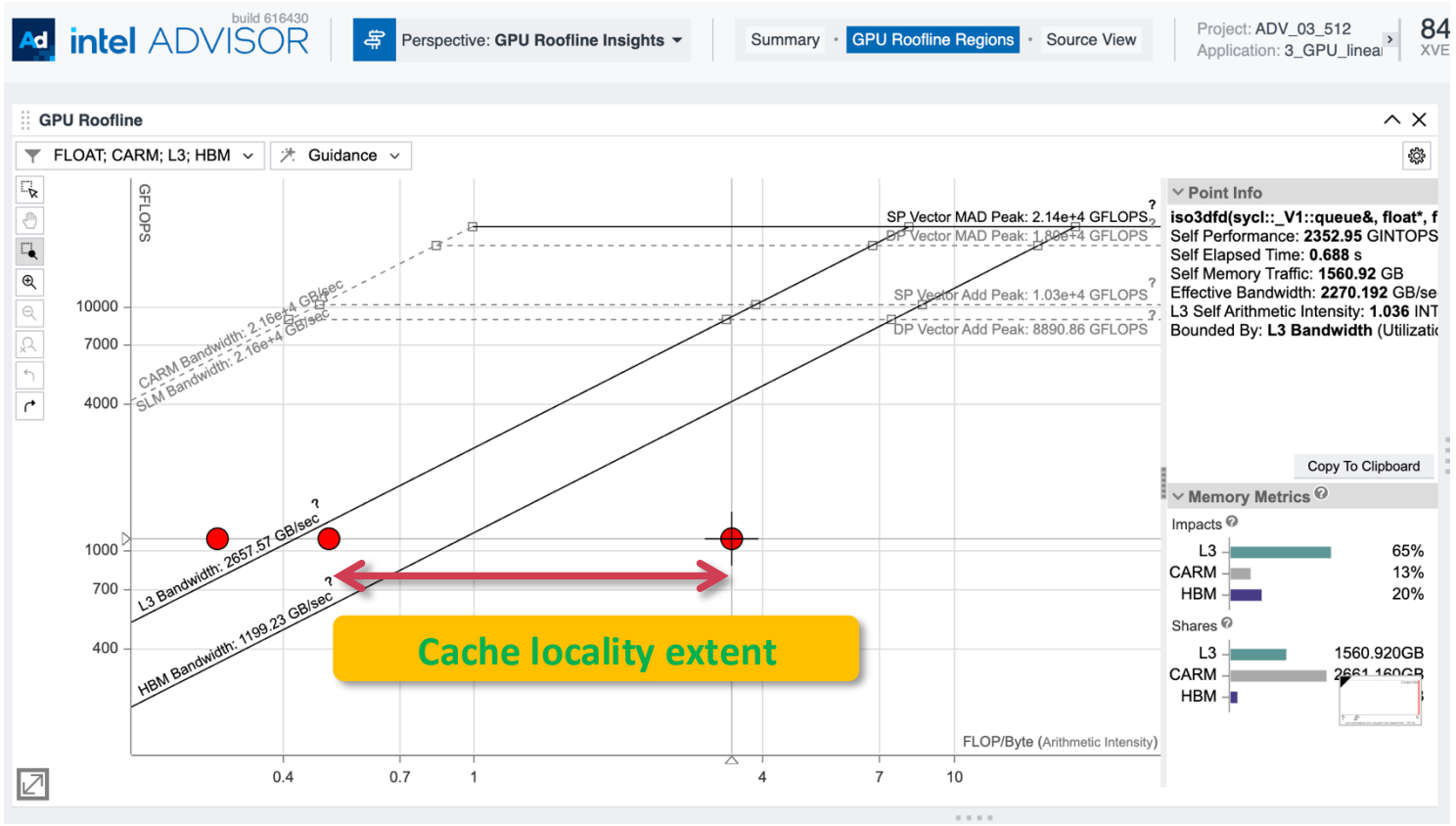


GPU MLR Roofline

Note some difference, “HBM” by default

1. Enable **Memory Metrics and Point info**
2. Look into **GPU Details** tab and find **INDIVIDUAL ROOFLINE** chart with small Guidance, Hints and BoundBy
   - **Quiz: what is a main bottleneck (“Bound By”) for the iso3dfd kernel?**
   - **Quiz: what are FP AI and INT AI?**
   - **Quiz: what are Instruction Mix Details?**
3. Go back to **Main** Roofline Chart and **double-click** on the circle to get the same guidance on the large chart

Kernel (loop) locality is proportional to the width of the “bound by” line (ratio of DRAM to Lx bytes)



3_GPU_linear

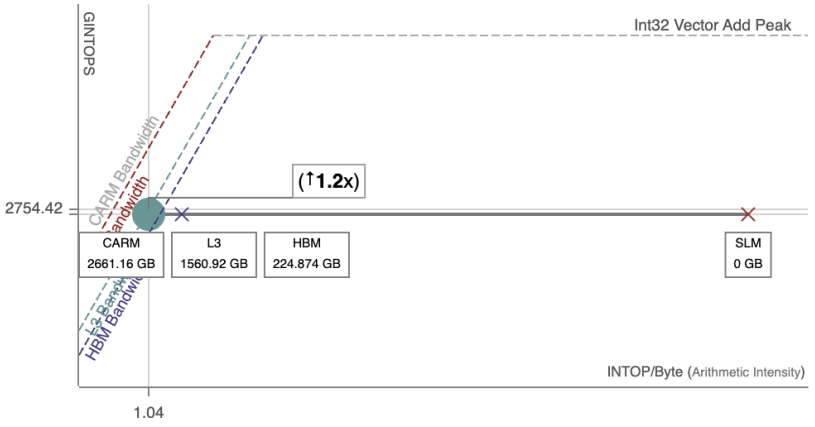
iso3dfd(sycl::_V1::queue&, float*, float*, float*, float*, unsigned long, unsigned long, unsi…

SUMMARY

Elapsed Time	**0.688s**
GINTOPS	**2,352.950**
GFLOPS	**1,128.622**
Global	**512 x 512 x 512**
Local	**512 x 2 x 1**

ROOFLINE GUIDANCE

This kernel is bounded by L3 Bandwidth.



OP/S AND BANDWIDTH

Compute (GINTOPS):	2352.95	0% of 966000.00 GINTOPS Int32 Vecto
Compute (GFLOPS):	1128.62	5% of 21400.00 GFLOPS SP Vector M
L3 Bandwidth:	2270.19	85% of 2657.57 GB/sec
Traffic:	1560.92	
FP AI:	0.50	
INT AI:	1.04	
SLM Bandwidth:	0	0% of 21600.00 GB/sec
CARM Bandwidth:	3870.38	17% of 21600.00 GB/sec
HBM Bandwidth:	327.05	27% of 1199.23 GB/sec

INSTRUCTION MIX DETAILS

Compute — 128.77 — 60%

Instruction	Data Type	Count
BASIC	SP	17.20
BASIC	INT32	25.59
BASIC	INT64	46.98
MAD	SP	4.19
MAD	INT32	16.78
BIT	INT32	1.68
BIT	INT64	14.26
VECTOR	INT32	2.10

Memory — 18.04 — 8%

Instruction	Count
STORE	0.42
LOAD	17.62

Other — 66.69 — 31%

Instruction	Count
MOVE	49.07
CONTROL FLOW	0.42
SYNC	17.20

MEMORY METRICS

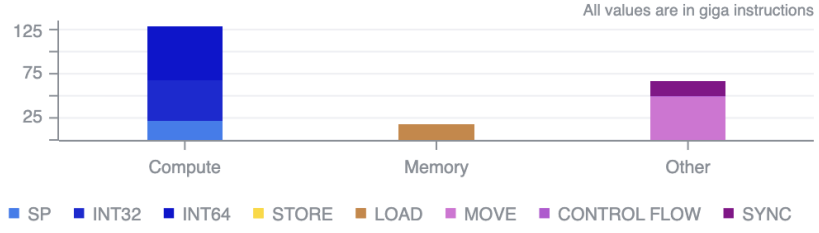
Impacts

L3	65%
SLM	0%
CARM	14%
HBM	21%

Shares

L3	1560.920GB
SLM	0GB
CARM	2661.160GB
HBM	224.874GB

INSTRUCTION MIX



PERFORMANCE CHARACTERISTICS

Active:	63.8%
Stalled:	36.0%
Idle:	0.2%
XVE Threading Occupancy:	84.3%
SIMD Width:	32

Thank you!

ARGONNE TRAINING PROGRAM ON EXTREME-SCALE COMPUTING

Produced by Argonne National Laboratory, a U.S. Department of Energy Laboratory managed by UChicagoArgonne, LLC under contract DE-AC02-06CH11357.

Special thanks to the National Energy Research Scientific Computing Center (NERSC) and Oak Ridge Leadership Computing Facility (OLCF) for the use of their resources during the training event.