ARGONNE
ATPESC2025
EXTREME-SCALE COMPUTING

# Roofline Performance Model

**JaeHyuk Kwack**
Argonne National Laboratory

# Outline

Introduction to the roofline model

Hands-on example on Aurora

# Introduction to the Roofline Model

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

Please join SC25 tutorial for Roofline model for more details and practices

https://sc25.conference-program.com/presentation/?id=tut115&sess=sess253

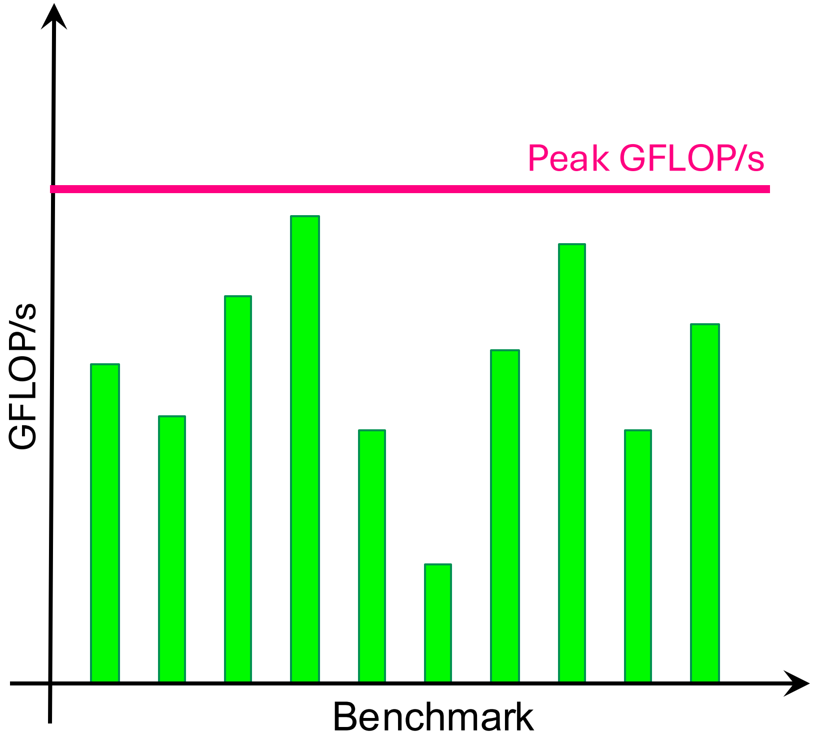
**We spend millions of dollars porting applications to CPUs and GPUs…**

***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

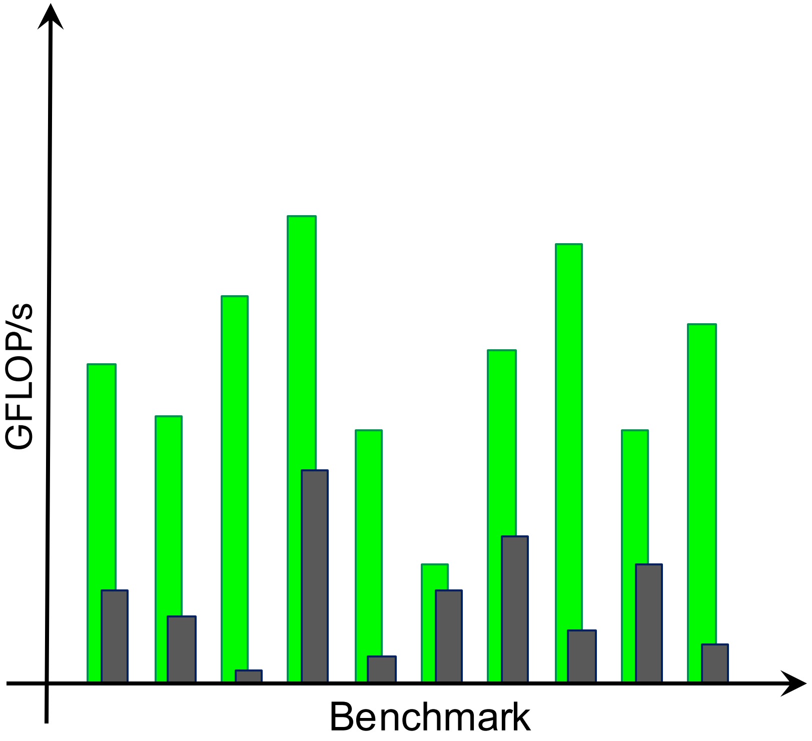


ATPESC2025

# 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.**



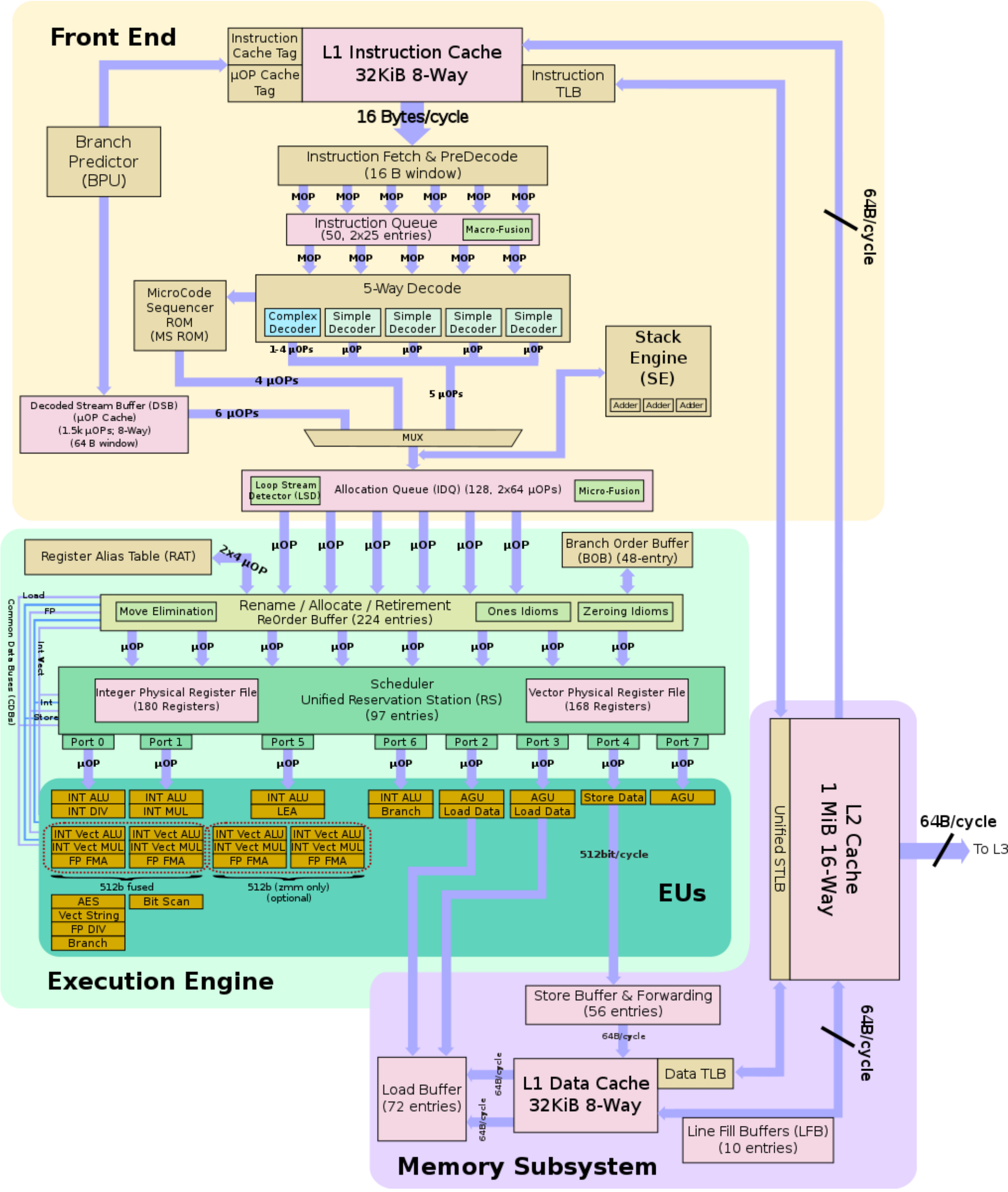
# Are we getting good performance?

- 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 $10^6$x slowdowns
  - **Provide no insights into quality or limits of algorithm or implementation.**
  - **Provide no guidance to CS, AM, application developers.**

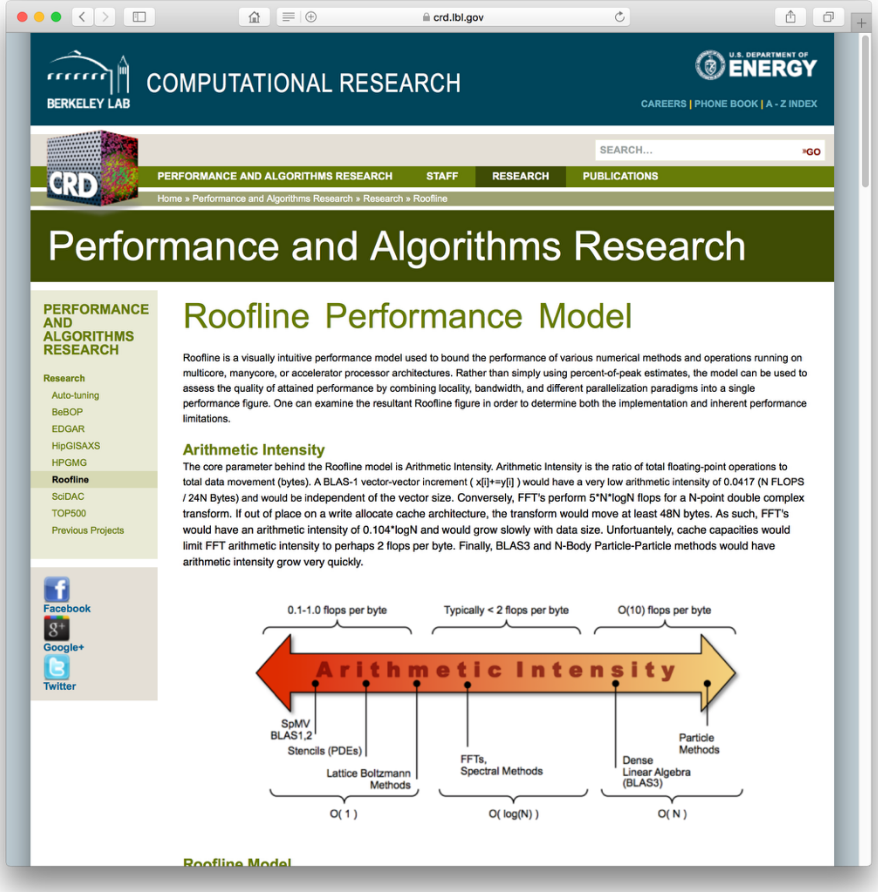


Front End
Instruction Cache Tag
µOP Cache Tag
L1 Instruction Cache 32KiB 8-Way
Instruction TLB
16 Bytes/cycle
Branch Predictor (BPU)
Instruction Fetch & PreDecode (16 B window)
MOP
Instruction Queue (50, 2x25 entries)
Macro-Fusion
MicroCode Sequencer ROM (MS ROM)
5-Way Decode
Complex Decoder
Simple Decoder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

# 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**



Performance and Algorithms Research

Roofline Performance Model

Roofline is a visually intuitive performance model used to bound the performance of various numerical methods and operations running on multicore, manycore, or accelerator processor architectures. Rather than simply using percent-of-peak estimates, the model can be used to assess the quality of attained performance by combining locality, bandwidth, and different parallelization paradigms into a single performance figure. One can examine the resultant Roofline figure in order to determine both the implementation and inherent performance limitations.

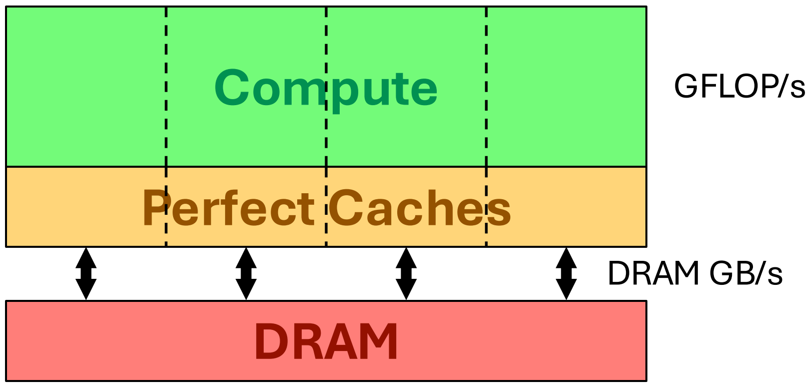
Arithmetic Intensity

The core parameter behind the Roofline model is Arithmetic Intensity. Arithmetic Intensity is the ratio of total floating-point operations to total data movement (bytes). A BLAS-1 vector-vector increment ( x[i]+=y[i] ) would have a very low arithmetic intensity of 0.0417 (N FLOPS / 24N Bytes) and would be independent of the vector size. Conversely, FFT's perform 5*N*logN flops for a N-point double complex transform. If out of place on a write allocate cache architecture, the transform would move at least 48N bytes. As such, FFT's would have an arithmetic intensity of 0.104*logN and would grow slowly with data size. Unfortuantely, cache capacities would limit FFT arithmetic intensity to perhaps 2 flops per byte. Finally, BLAS3 and N-Body Particle-Particle methods would have arithmetic intensity grow very quickly.

Roofline Model

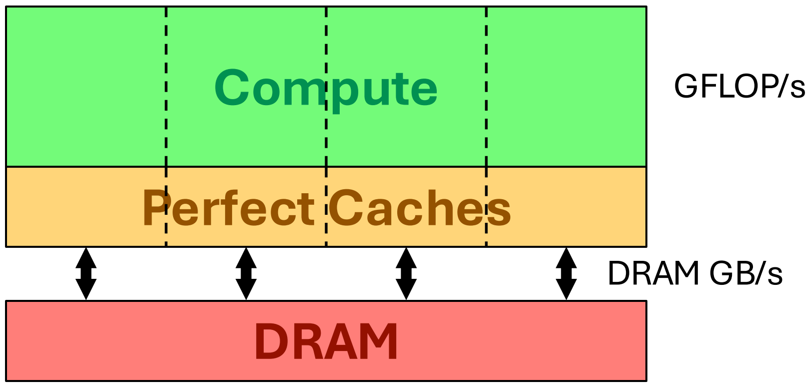
https://crd.lbl.gov/roofline

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



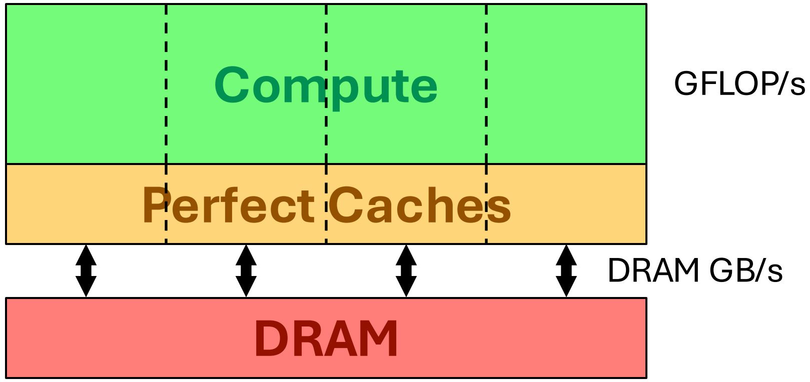
$$\text{Time} = \max \begin{cases} \text{\#FP ops / Peak GFLOP/s} \\ \text{\#Bytes / Peak GB/s} \end{cases}$$

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



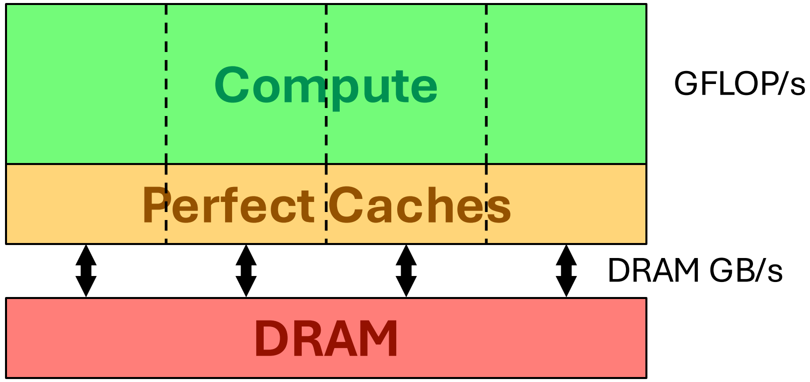
$$\frac{\text{Time}}{\text{\#FP ops}} = \max\begin{cases}\text{1 / Peak GFLOP/s} \\ \\ \text{\#Bytes / \#FP ops / Peak GB/s}\end{cases}$$

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



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

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



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

***Arithmetic Intensity (AI) = measure of data locality***

# Arithmetic Intensity

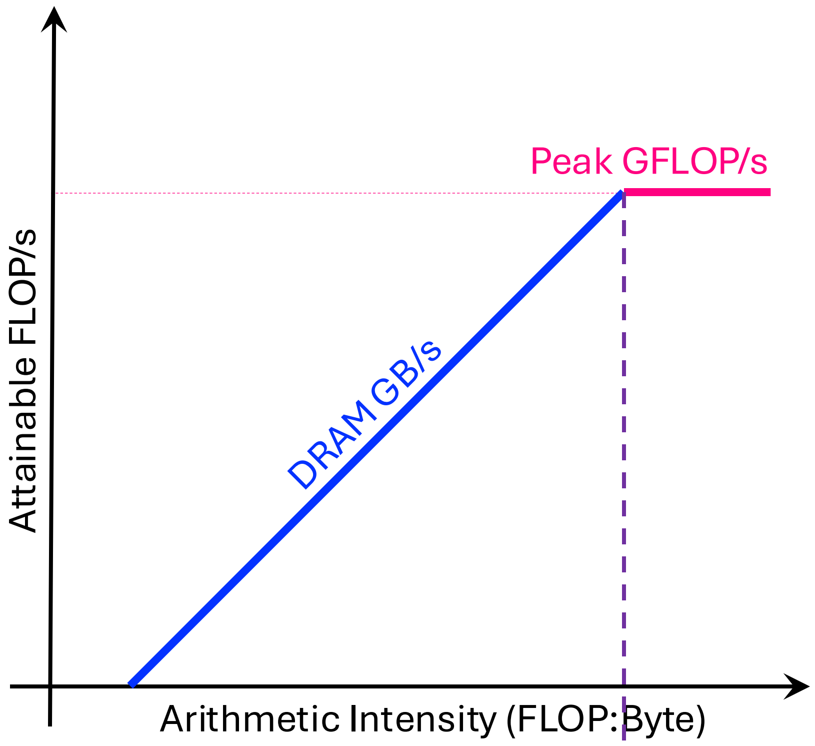
- Measure of data locality (data reuse)
- Ratio of **Total Flops** performed to **Total Bytes** moved
- For the DRAM Roofline…
  - Total Bytes to/from DRAM
  - Includes all cache and prefetcher effects
  - Can be very different from total loads/stores (bytes requested)
  - Equal to ratio of sustained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

# (DRAM) Roofline Model

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

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

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



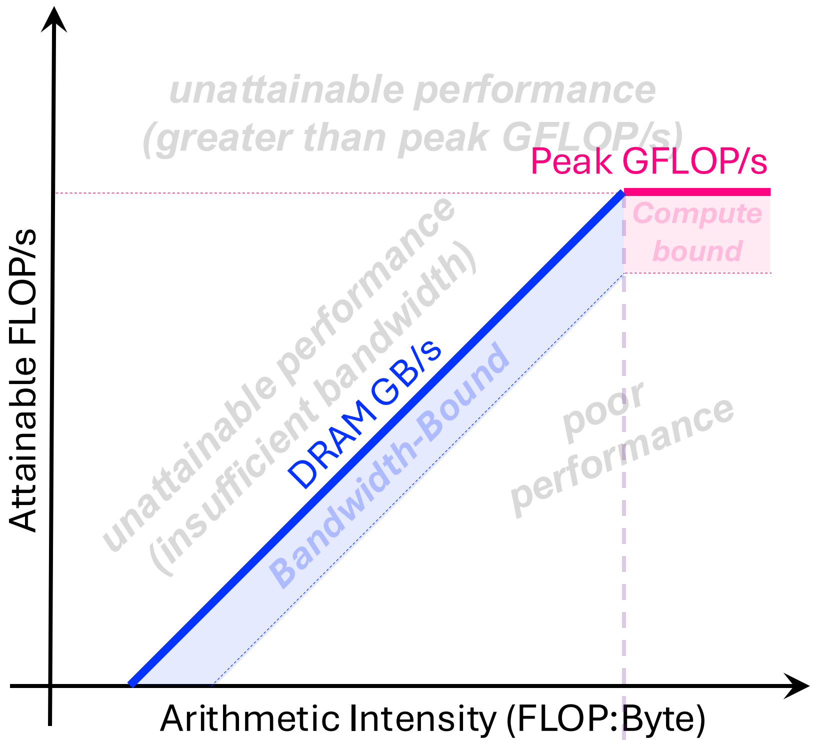
***Transition @ AI ==***
***Peak GFLOP/s / Peak GB/s ==***
***'Machine Balance'***

# (DRAM) Roofline Model

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

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

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



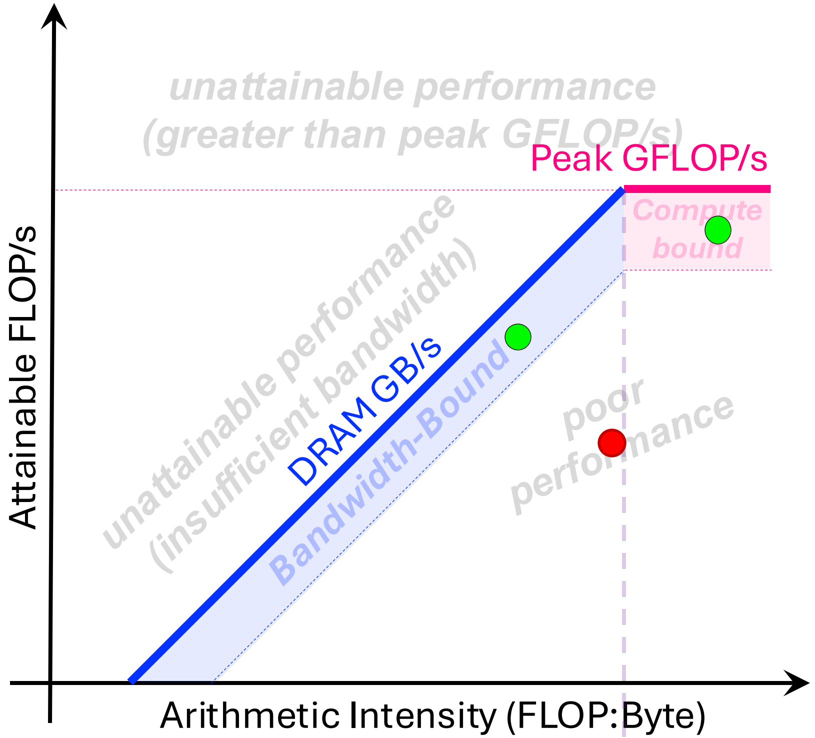
*Transition @ AI ==*
*Peak GFLOP/s / Peak GB/s ==*
*‘Machine Balance’*

# (DRAM) Roofline Model

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

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

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

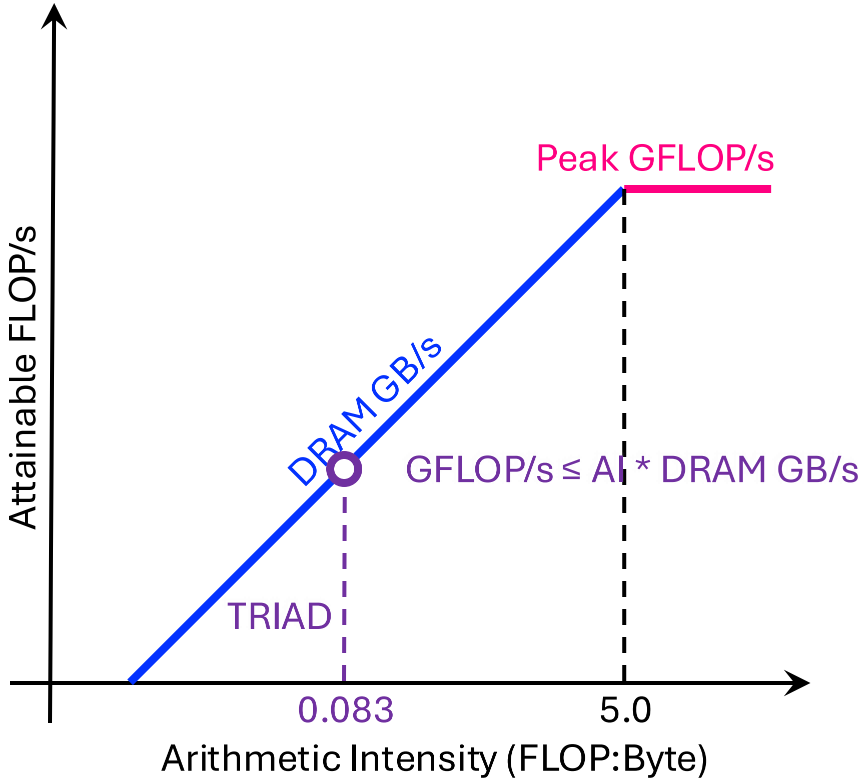
# Roofline Example #1

- 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…

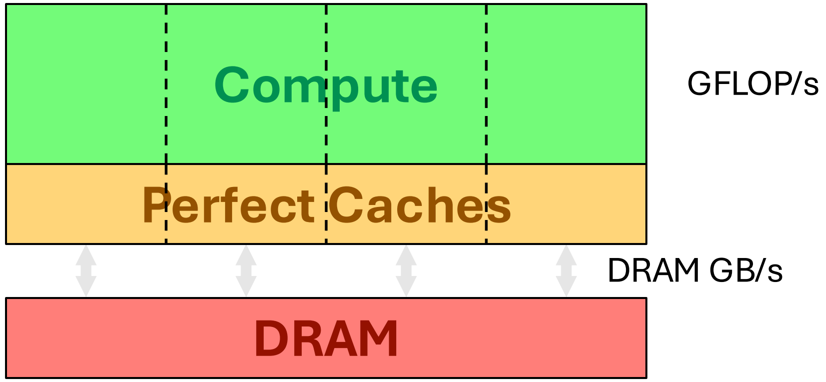
```
#pragma omp parallel for
for(i=0;i<N;i++){
  Z[i] = X[i] + alpha*Y[i];
}
```

  - 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…

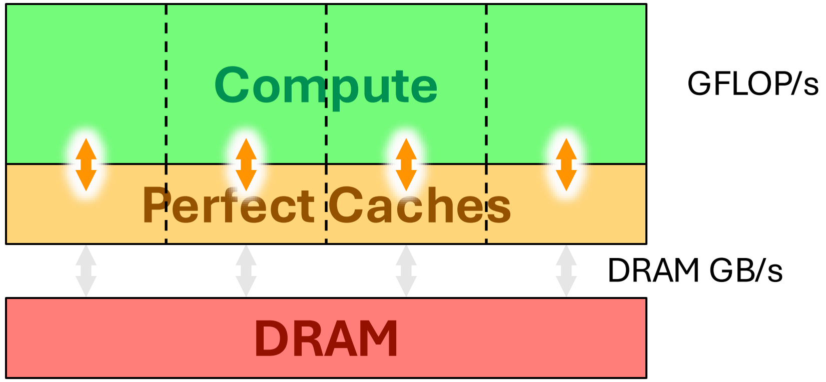


```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                    + old[k  ][j  ][i-1]
                    + old[k  ][j  ][i+1]
                    + old[k  ][j-1][i  ]
                    + old[k  ][j+1][i  ]
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

# Roofline Example #2

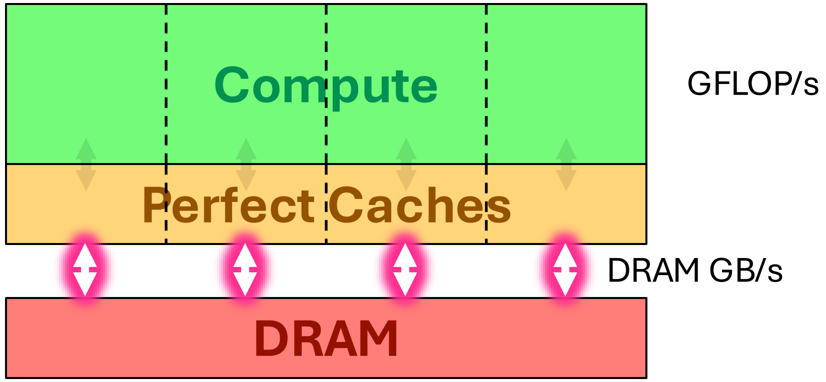
- Conversely, 7-point constant coefficient stencil…
  - ○ 7 FLOPs
  - ○ 8 memory references (7 reads, 1 store) per point
  - ○ **AI = 7 / (8*8) = 0.11 FLOPs per byte**
    **(measured at the L1)**

```
#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

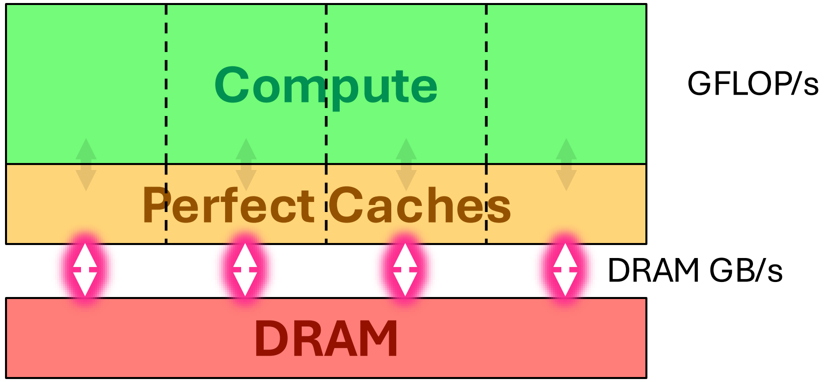
- Conversely, 7-point constant coefficient stencil…
  - 7 FLOPs
  - 8 memory references (7 reads, 1 store) per point
  - **Ideally, cache will filter all but 1 read and 1 write per point**



```
#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)**



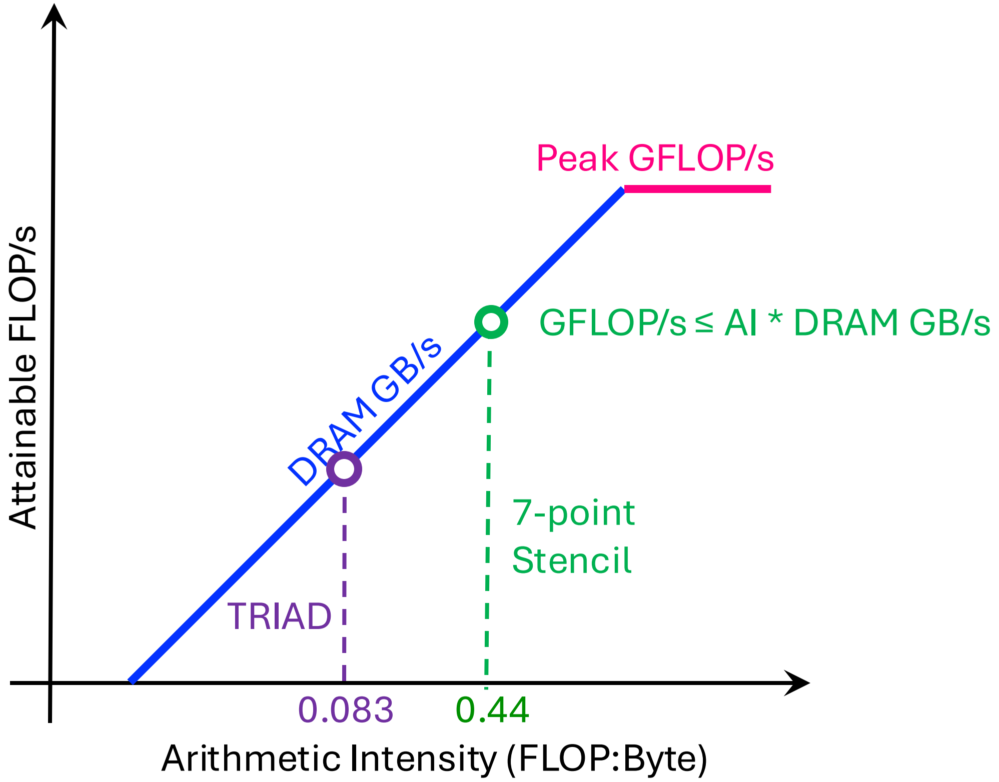
```
#pragma omp parallel for
for(k=1;k<dim+1;k++){
for(j=1;j<dim+1;j++){
for(i=1;i<dim+1;i++){
  new[k][j][i] = -6.0*old[k  ][j  ][i  ]
                    + old[k  ][j  ][i-1]
                    + old[k  ][j  ][i+1]
                    + old[k  ][j-1][i  ]
                    + old[k  ][j+1][i  ]
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



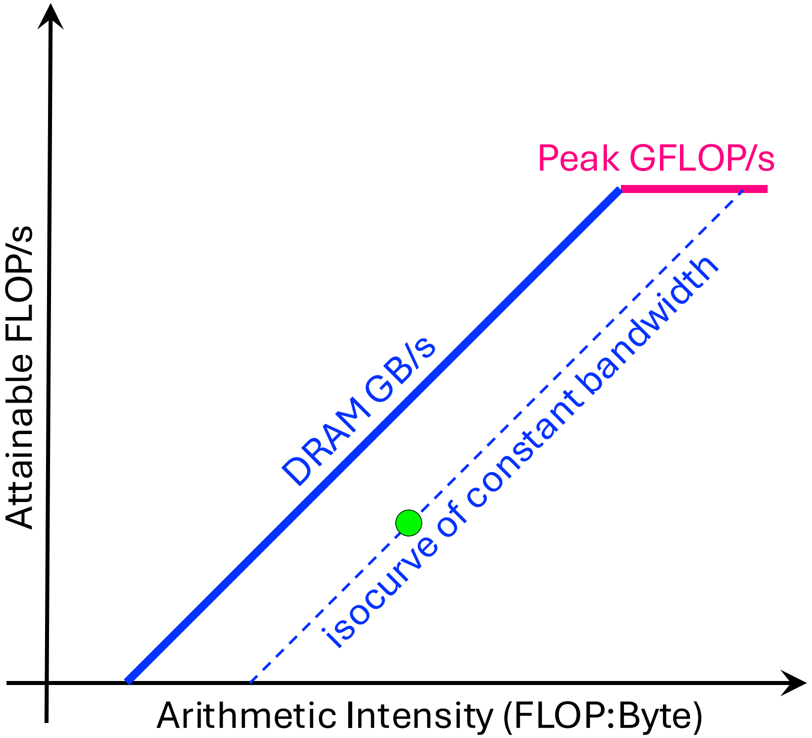
# Roofline Example #3

- Roofline makes it obvious what the bound on FLOP rate is, but let's ask the reverse…

- *Given a low FLOP rate and AI, what DRAM bandwidth are we attaining?*

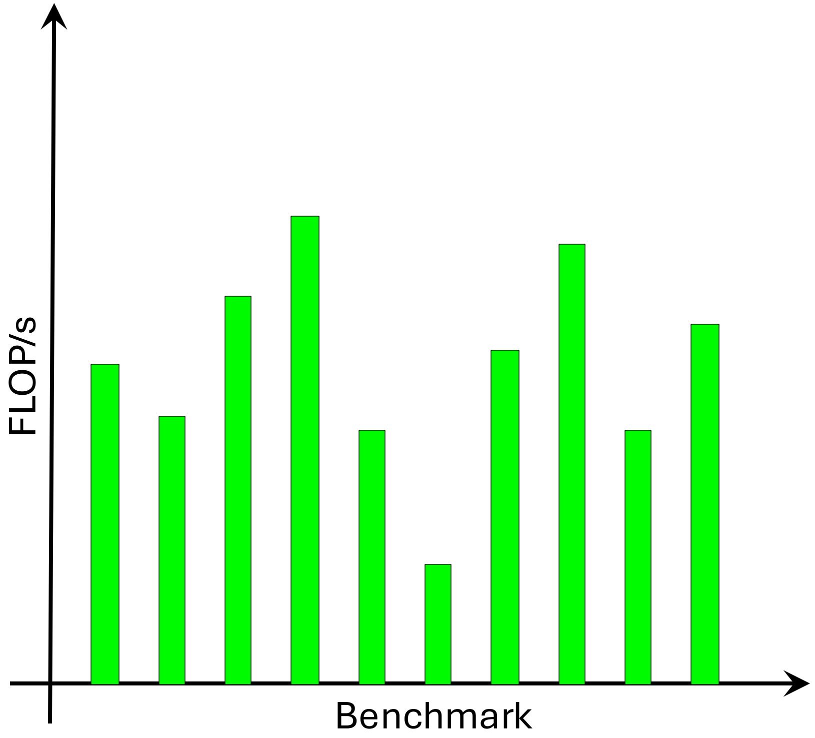
$$\text{average GB/s} = \frac{\text{GFLOP/s}}{\text{AI}_{\text{DRAM}}}$$

- This is just a slope (y/x)
- Thus we can define an isocurve of constant bandwidth


Peak GFLOP/s
DRAM GB/s
isocurve of constant bandwidth
Attainable FLOP/s
Arithmetic Intensity (FLOP:Byte)

# Are we getting good performance?

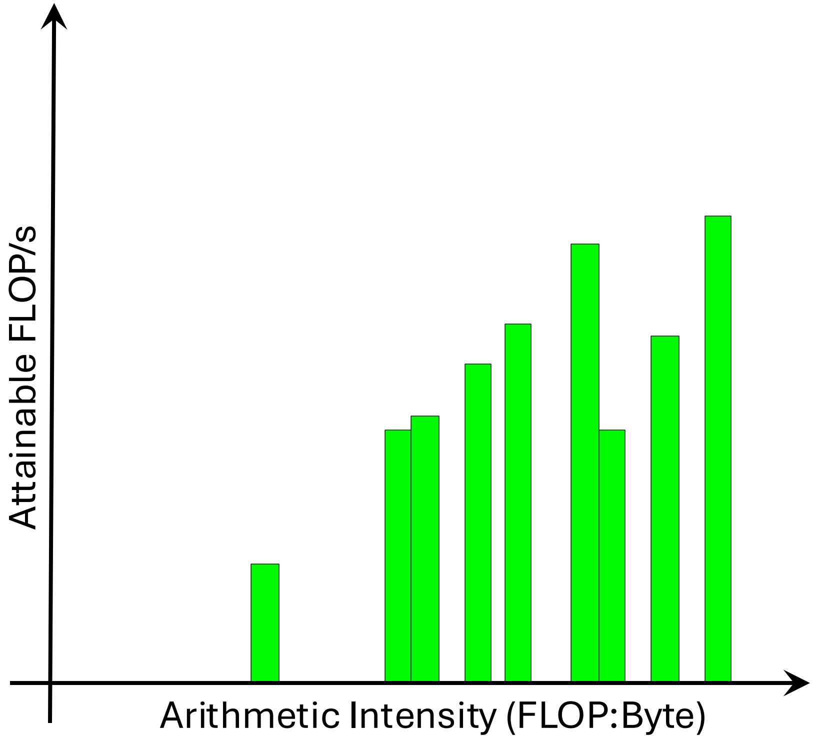
- Think back to our mix of benchmarks…



ATPESC2025

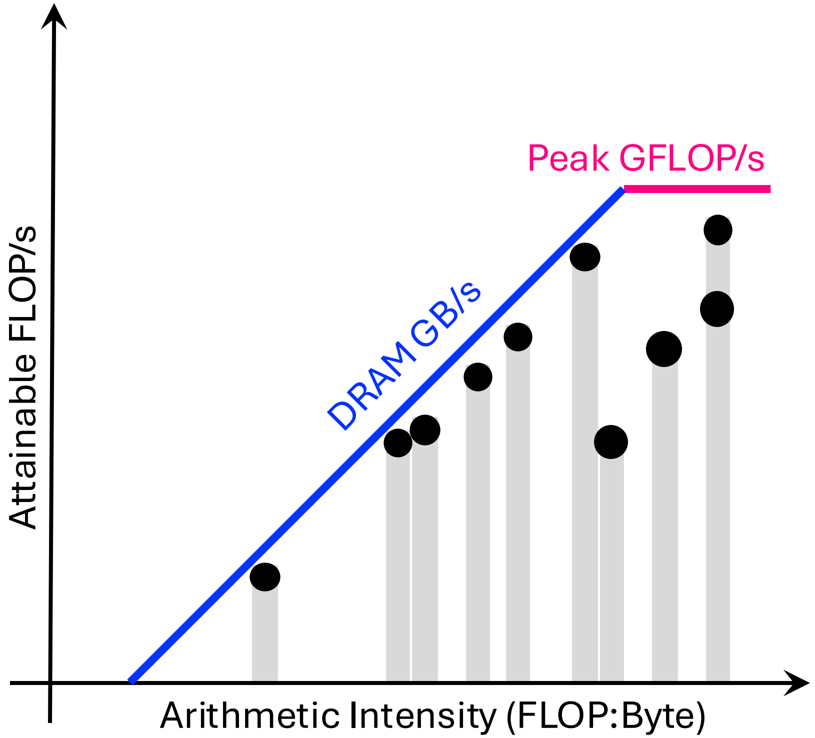
# Are we getting good performance?

- We can sort benchmarks by arithmetic intensity…



# Are we getting good performance?

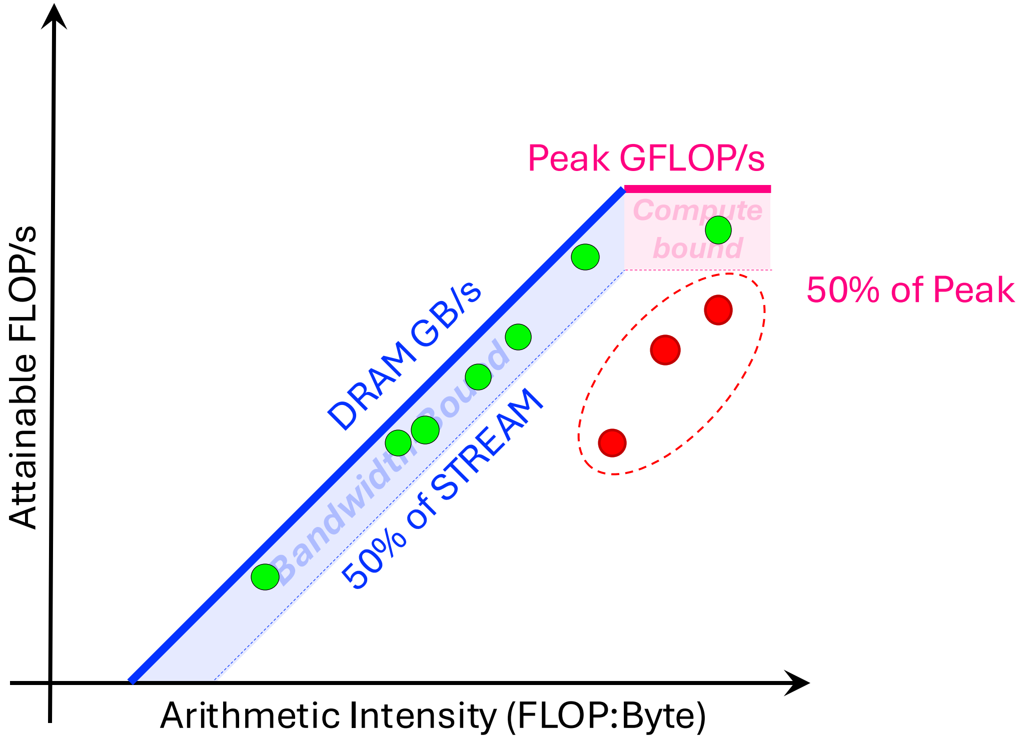
- We can sort benchmarks by arithmetic intensity…
- … and compare performance relative to machine capabilities



ATPESC2025

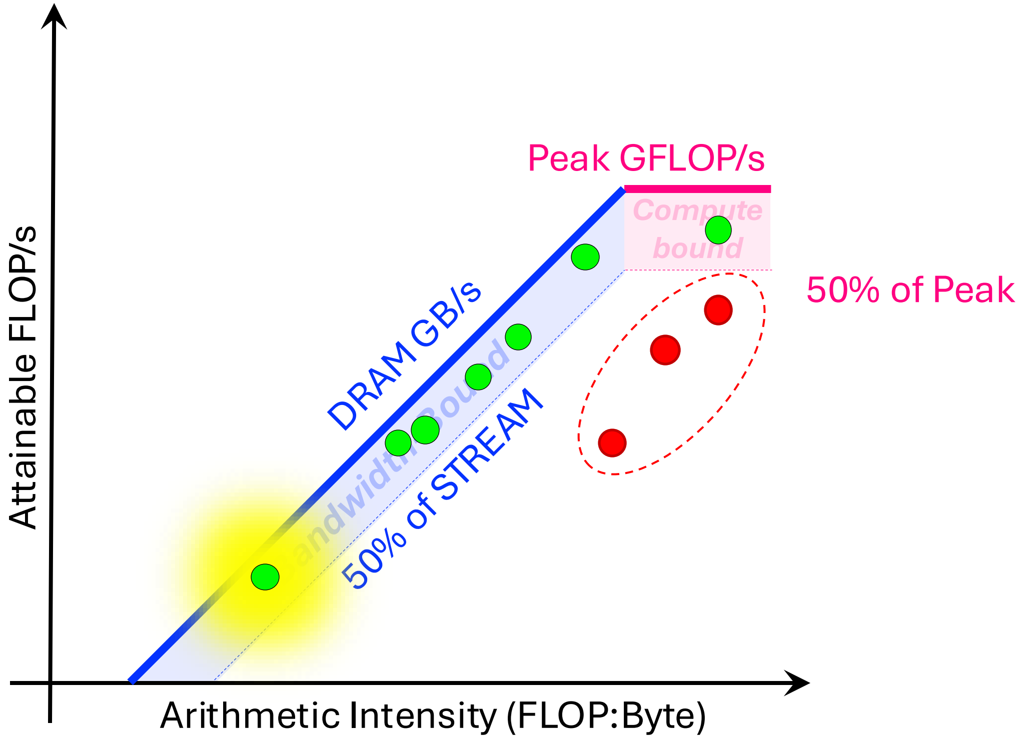
# Are we getting good performance?

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



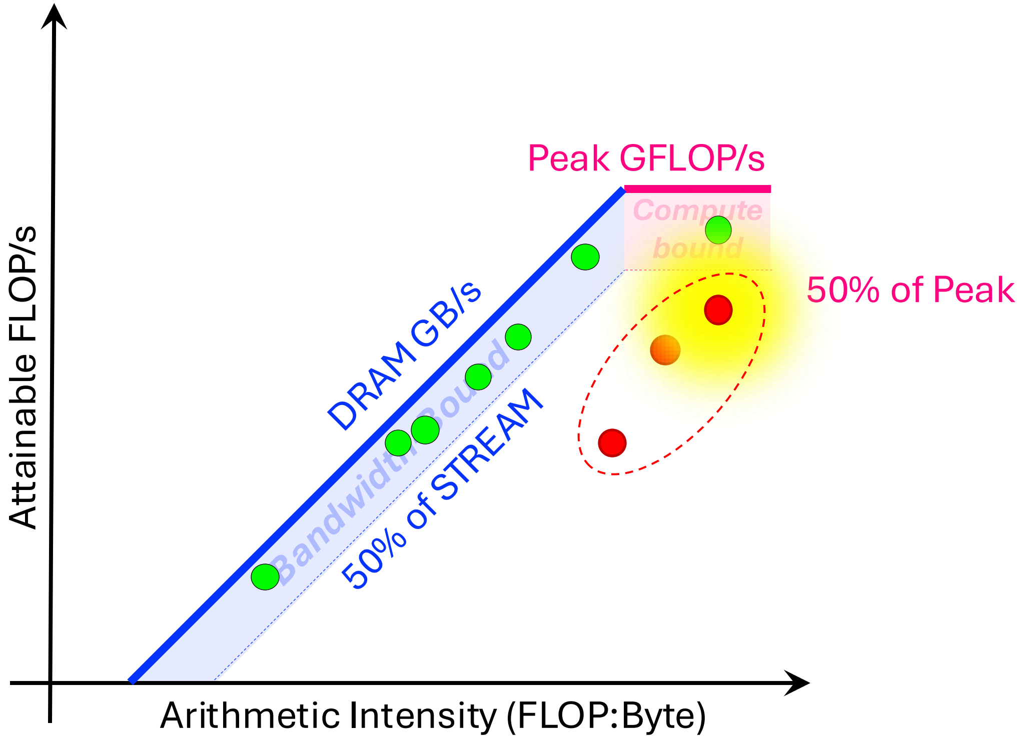
# Are we getting good performance?

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



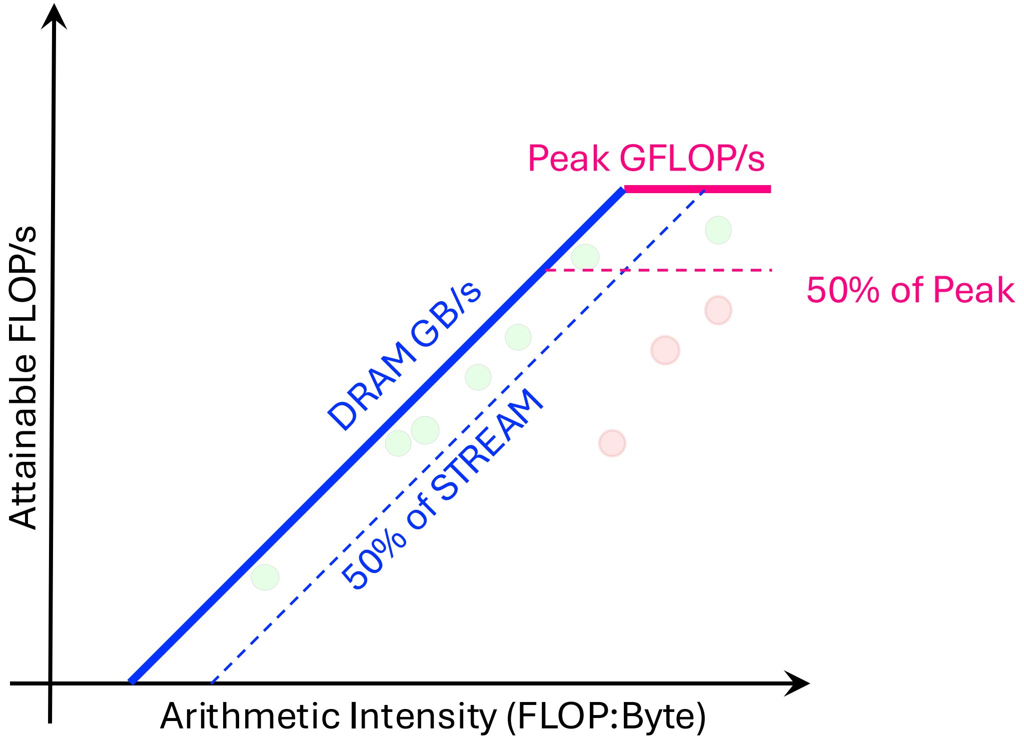
# Are we getting good performance?

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



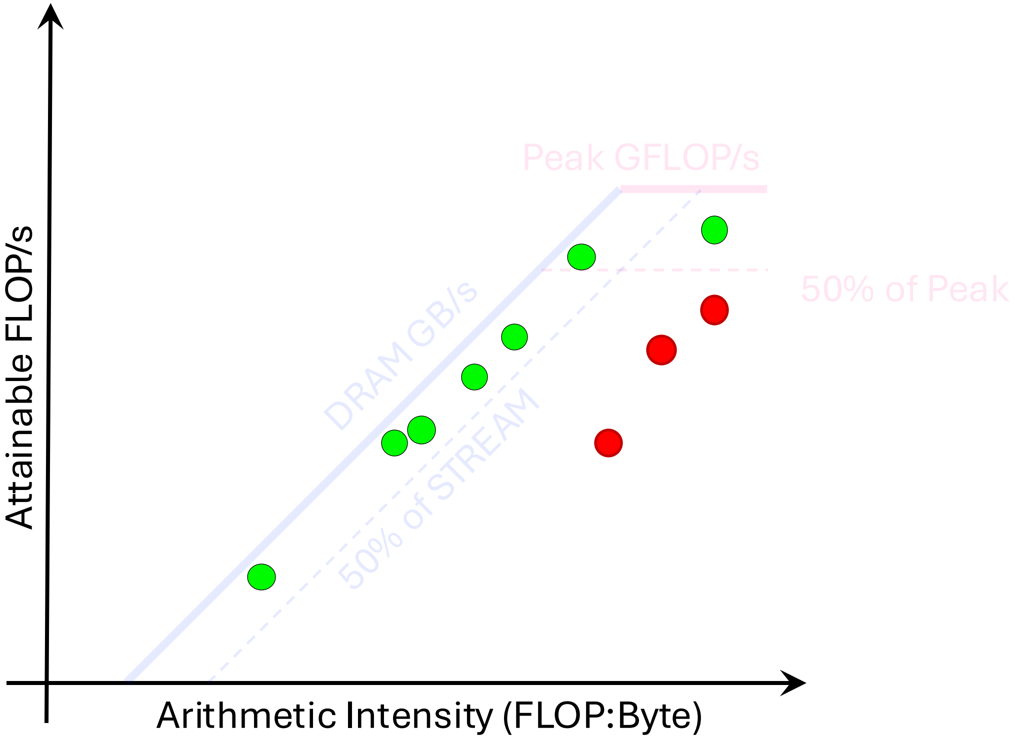
# Recap: Roofline is made of two components

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



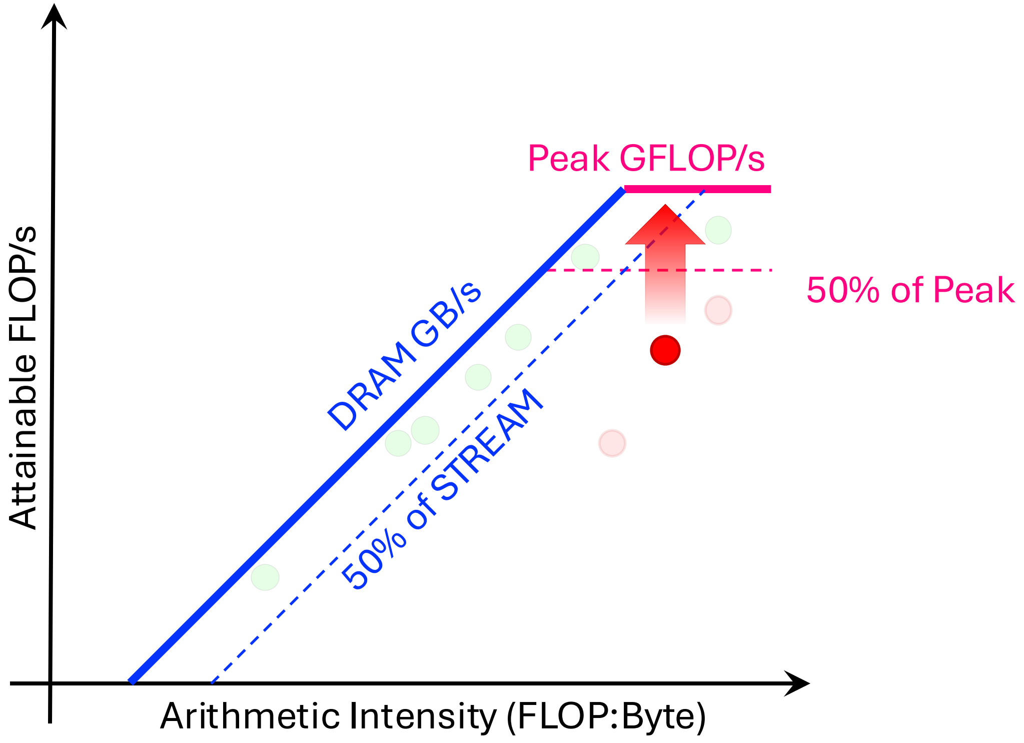
# Recap: Roofline is made of two components

- Machine Model
  - Lines defined by peak GB/s and GF/s (**Benchmarking**)
  - Unique to each architecture
  - Common to all apps on that architecture
- Application Characteristics
  - Dots defined by application GFLOP's and GB's (**Application Instrumentation**)
  - Unique to each application
  - Unique to each architecture



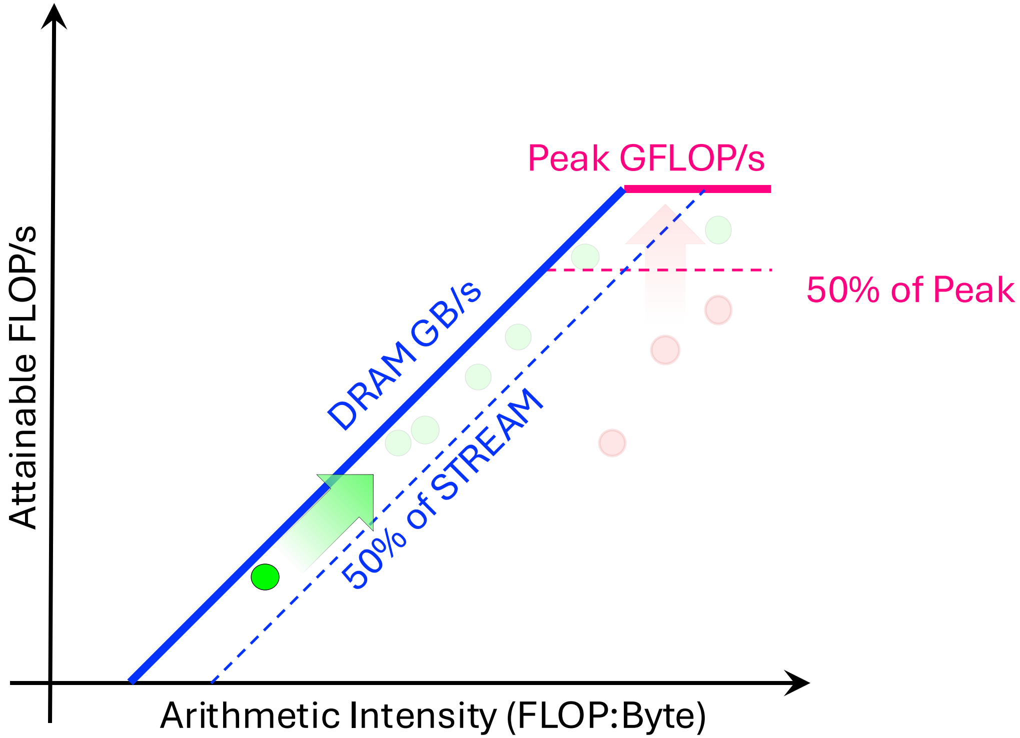
# Recap: Optimization Strategy

1. Get to the Roofline



# Recap: Optimization Strategy

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



# How can performance ever be below the Roofline?

# How can performance be below the Roofline?

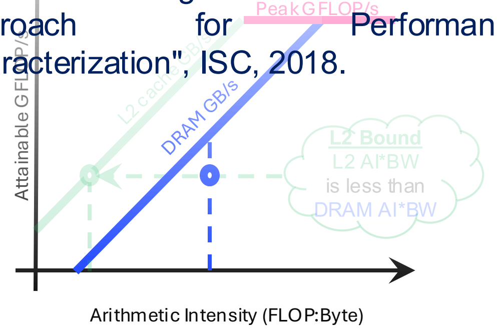
## *Simple DRAM model can be insufficient for a variety of reasons…*

### DRAM’s not the bottleneck…

- ○ Cache bandwidth and cache locality
- ○ PCIe bandwidth

### *…The Hierarchical Roofline Model*

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

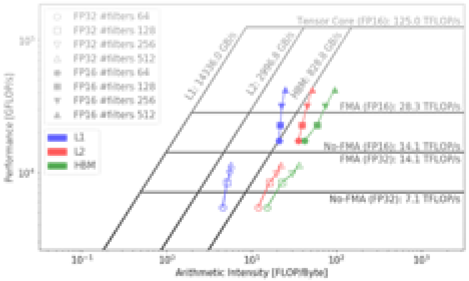


### Not enough of Vector/Tensor instr.

- ○ No FMA
- ○ Mixed Precision
- ○ No Tensor Core OPs

### *… Additional Ceilings*

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

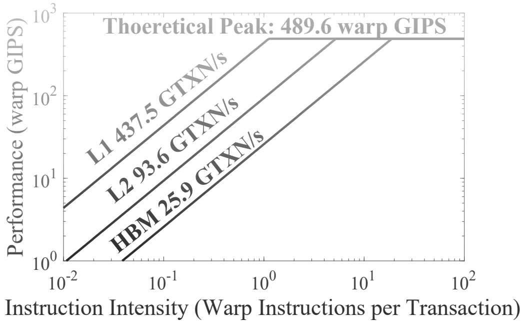


### Integer-heavy Codes…

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

### *… The Instruction Roofline Model*

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

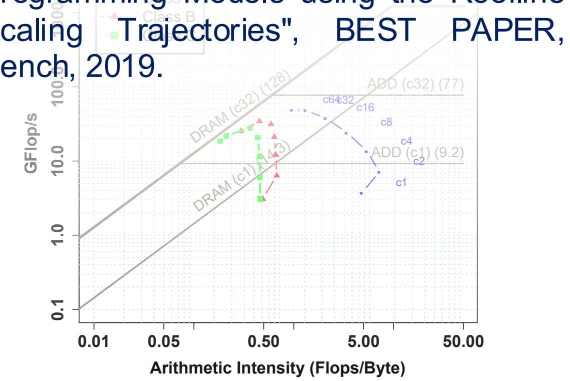


### Lack of Parallelism…

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

### *… Roofline Scaling Trajectories*

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



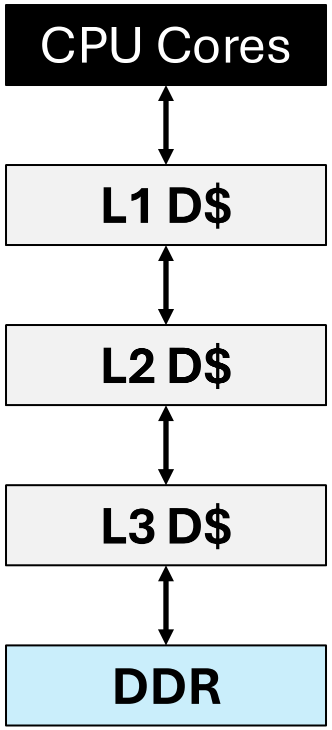
ATPESC2025

# Below the Roofline?

## Memory Hierarchy and Cache Bottlenecks

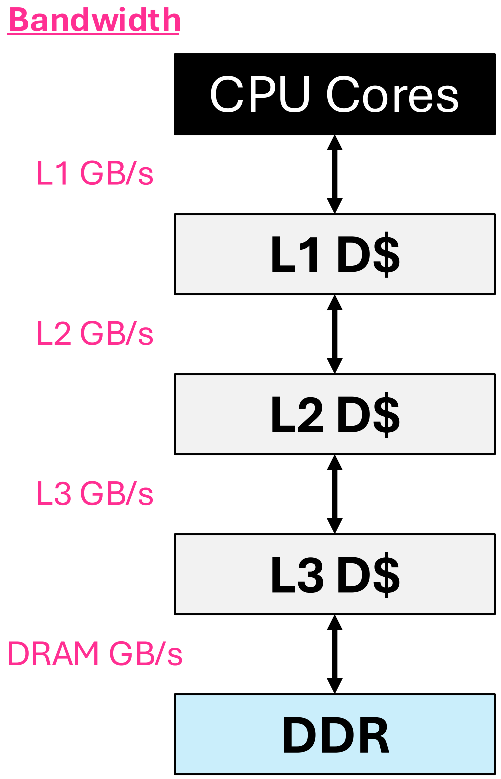
# Memory Hierarchy

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



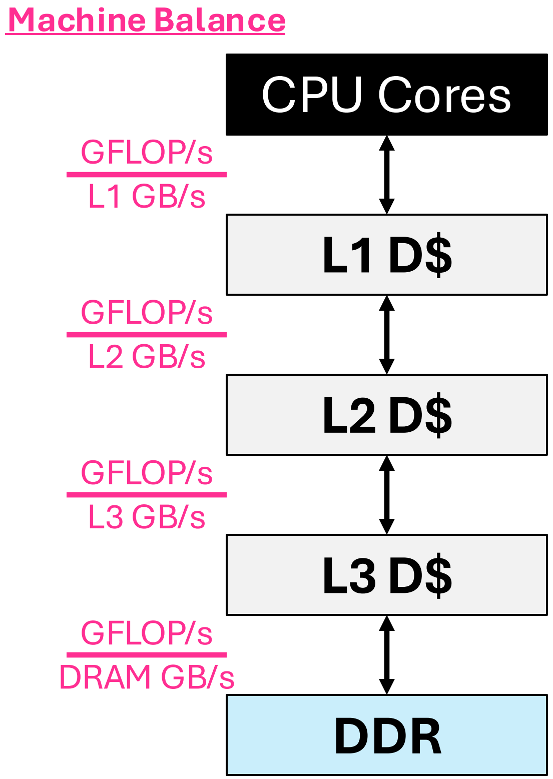
# Memory Hierarchy

- CPUs/GPUs have different bandwidths for each level



# Memory Hierarchy

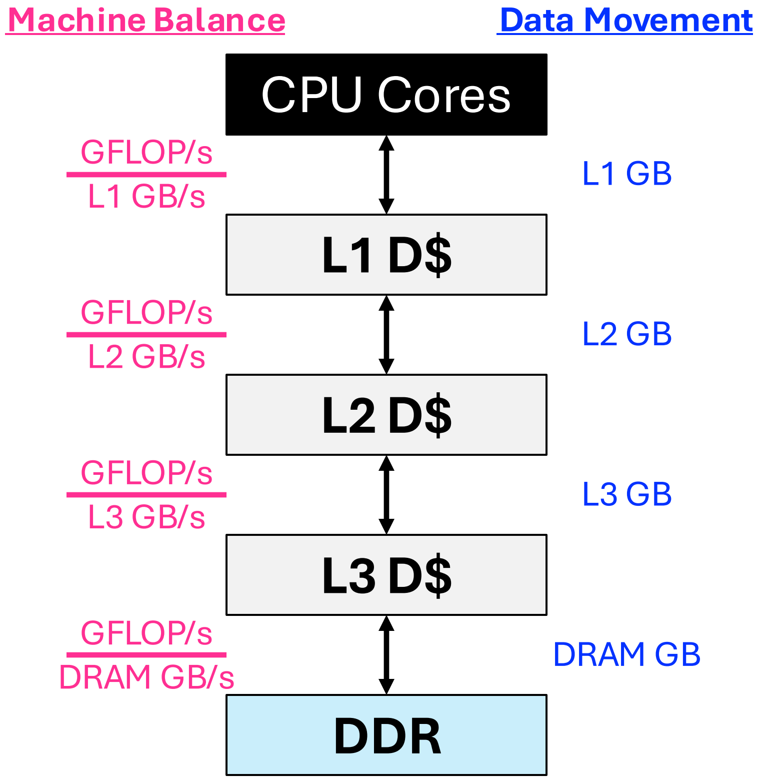
- CPUs/GPUs have different bandwidths for each level
    - different **machine balances** for each level



# Memory Hierarchy

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

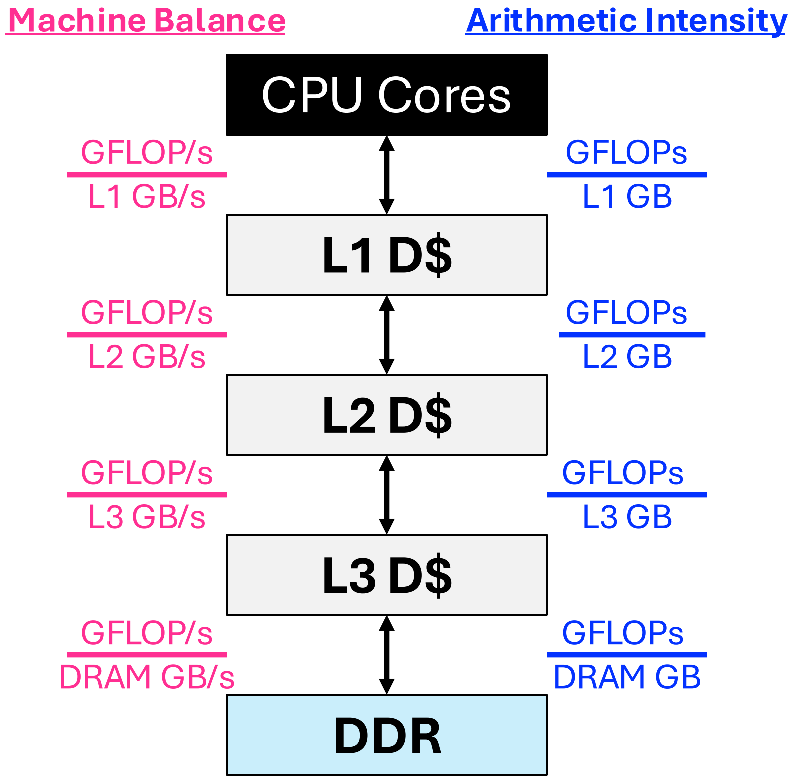
- Applications have locality in each level
  - different **data movements** for each level



# Memory Hierarchy

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

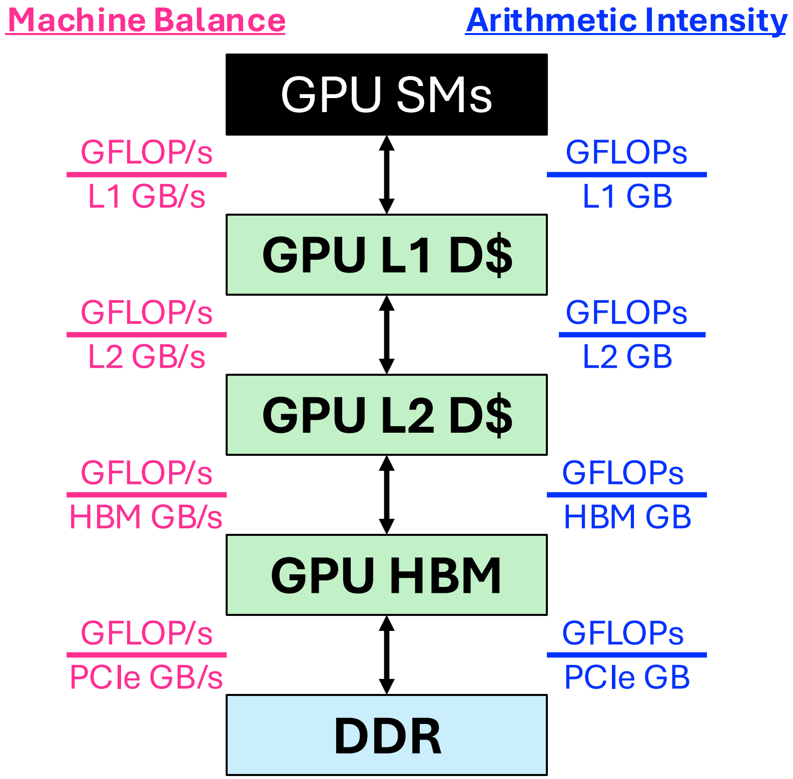
- Applications have locality in each level
  - different data movements for each level
  - different **arithmetic intensity** for each level



Machine Balance
Arithmetic Intensity
CPU Cores
$\frac{\text{GFLOP/s}}{\text{L1 GB/s}}$
$\frac{\text{GFLOPs}}{\text{L1 GB}}$
L1 D$
$\frac{\text{GFLOP/s}}{\text{L2 GB/s}}$
$\frac{\text{GFLOPs}}{\text{L2 GB}}$
L2 D$
$\frac{\text{GFLOP/s}}{\text{L3 GB/s}}$
$\frac{\text{GFLOPs}}{\text{L3 GB}}$
L3 D$
$\frac{\text{GFLOP/s}}{\text{DRAM GB/s}}$
$\frac{\text{GFLOPs}}{\text{DRAM GB}}$
DDR

# 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, CXL, or NoC**



Machine Balance
Arithmetic Intensity
GPU SMs
$\frac{\text{GFLOP/s}}{\text{L1 GB/s}}$
$\frac{\text{GFLOPs}}{\text{L1 GB}}$
GPU L1 D$
$\frac{\text{GFLOP/s}}{\text{L2 GB/s}}$
$\frac{\text{GFLOPs}}{\text{L2 GB}}$
GPU L2 D$
$\frac{\text{GFLOP/s}}{\text{HBM GB/s}}$
$\frac{\text{GFLOPs}}{\text{HBM GB}}$
GPU HBM
$\frac{\text{GFLOP/s}}{\text{PCIe GB/s}}$
$\frac{\text{GFLOPs}}{\text{PCIe GB}}$
DDR

# Cache Bottlenecks

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

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

$AI_x$ (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...

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

$\text{AI}_x$ (Arithmetic Intensity at level “x”) = FLOPs / Bytes (moved to/from level “x” )

ATPESC2025

# Cache Bottlenecks

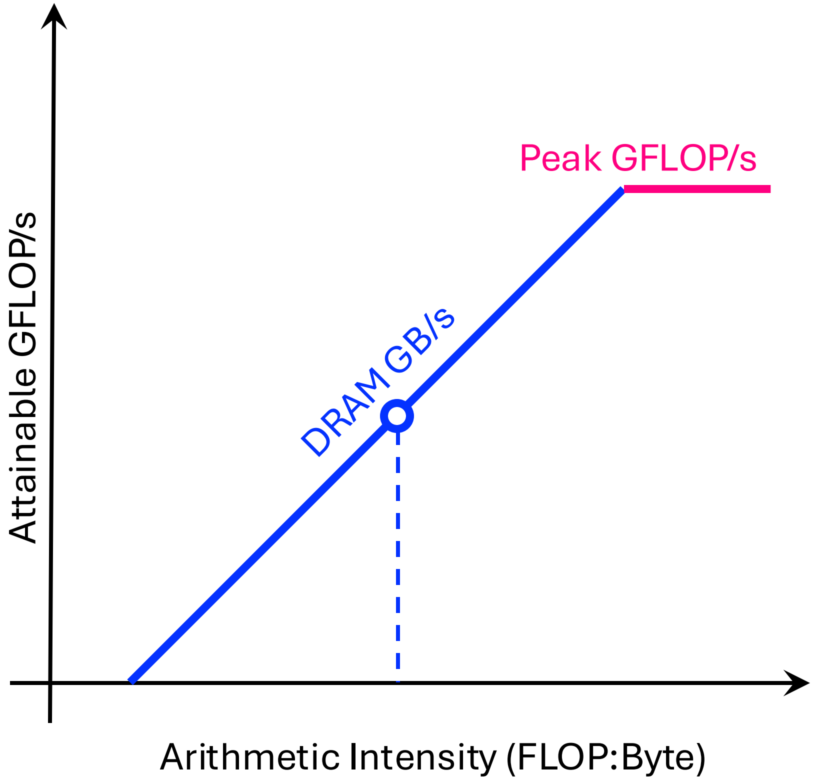
- For each additional level of the memory hierarchy, we can add another term to our model…

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

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

# Cache Bottlenecks

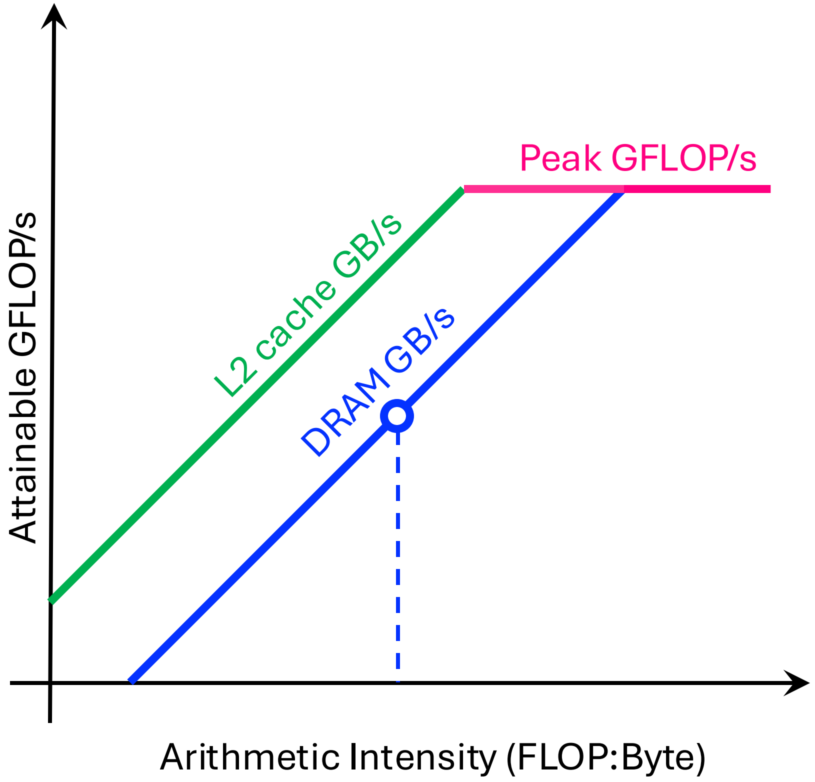
- Plot equation in a single figure…
  - ○ “**Hierarchical Roofline**” Model



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

# Cache Bottlenecks

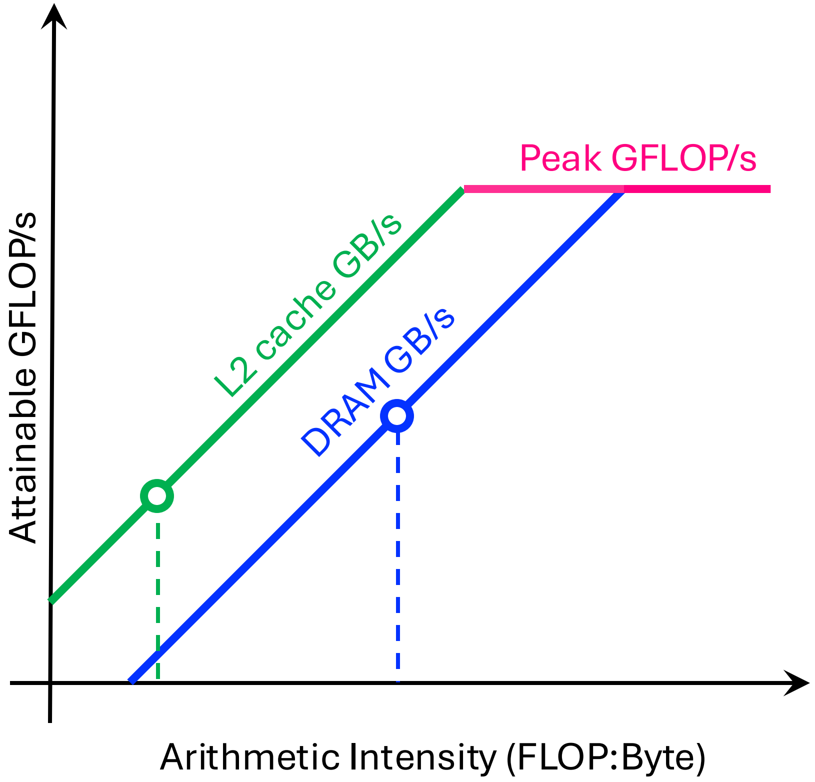
- Plot equation in a single figure…
  - ○ “**Hierarchical Roofline**” Model
  - ○ Bandwidth ceiling (diagonal line) for each level of memory



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

# Cache Bottlenecks

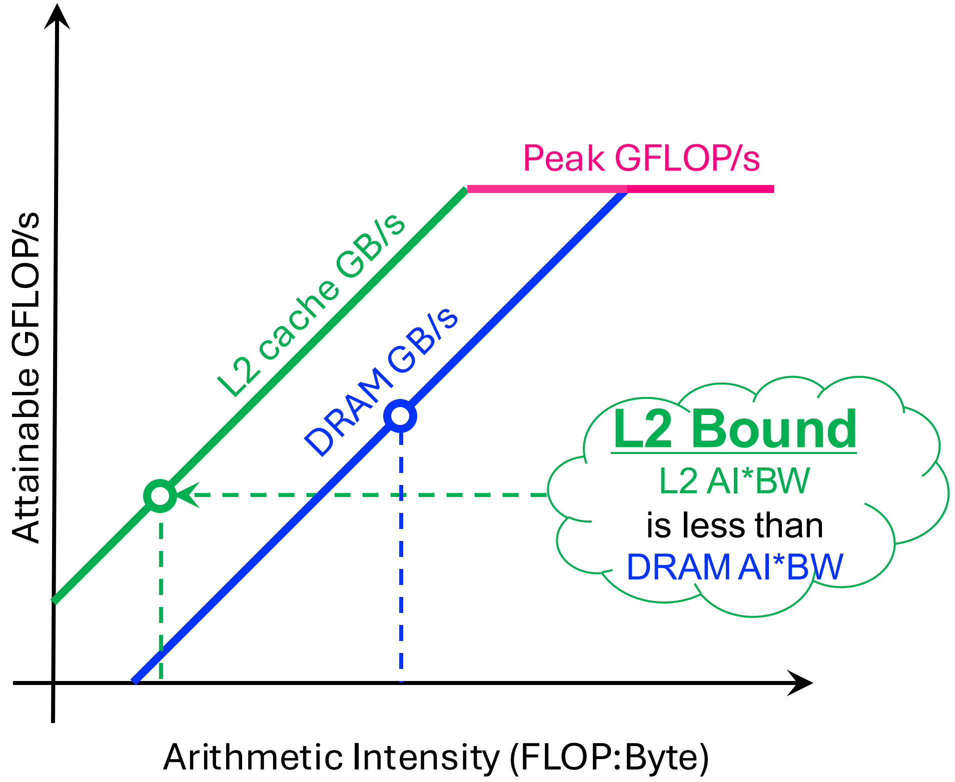
- Plot equation in a single figure…
  - “**Hierarchical Roofline**” Model
  - Bandwidth ceiling (diagonal line) for each level of memory
  - Arithmetic Intensity (dot) for each level of memory



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

# Cache Bottlenecks

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

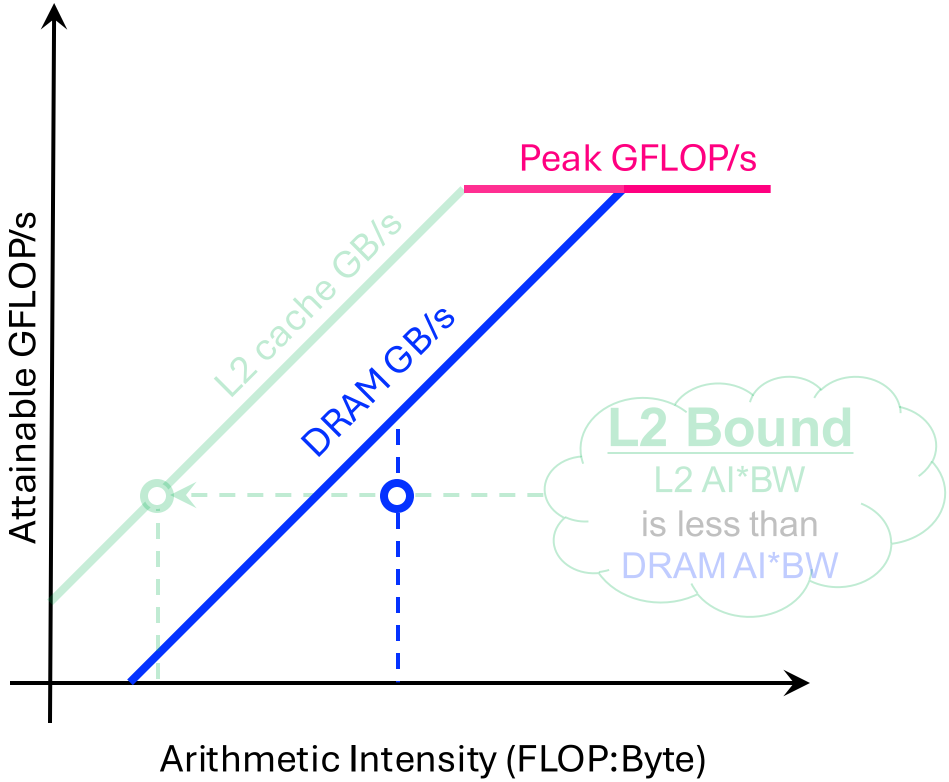


T. Koskela, Z. Matveev, C. Yang, A. Adedoyin, R. Belenov, P. Thierry, Z. Zhao, R. Gayatri, H. Shan, L. Oliker, J. Deslippe, R. Green, S. Williams, "A Novel Multi-Level Integrated Roofline Model Approach for Performance Characterization", ISC, 2018.

# Cache Bottlenecks

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

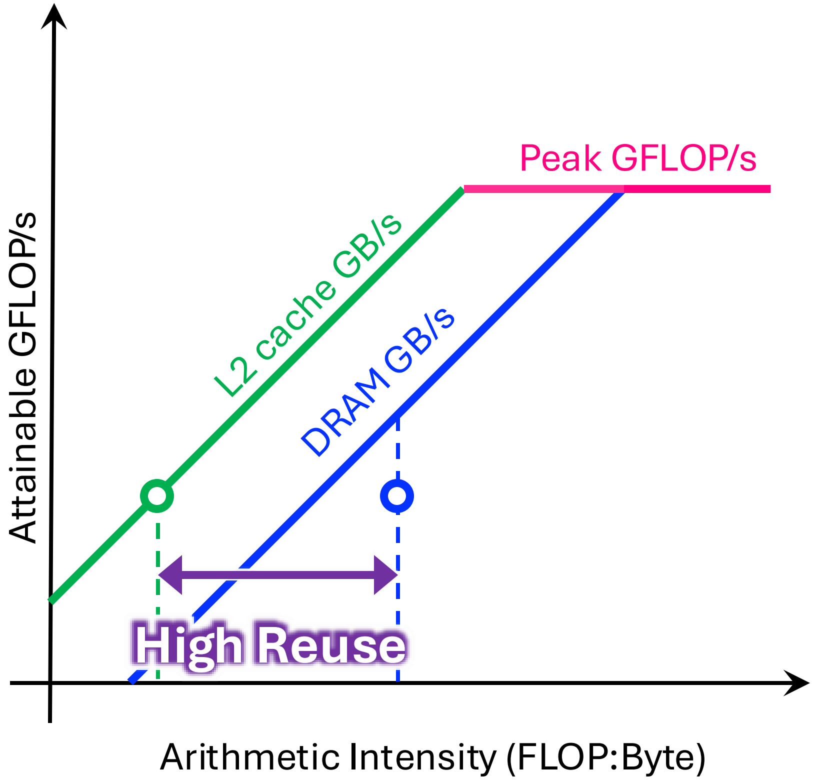
- **If L2 bound, we see DRAM dot well below DRAM ceiling**



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

# Cache Hit Rates

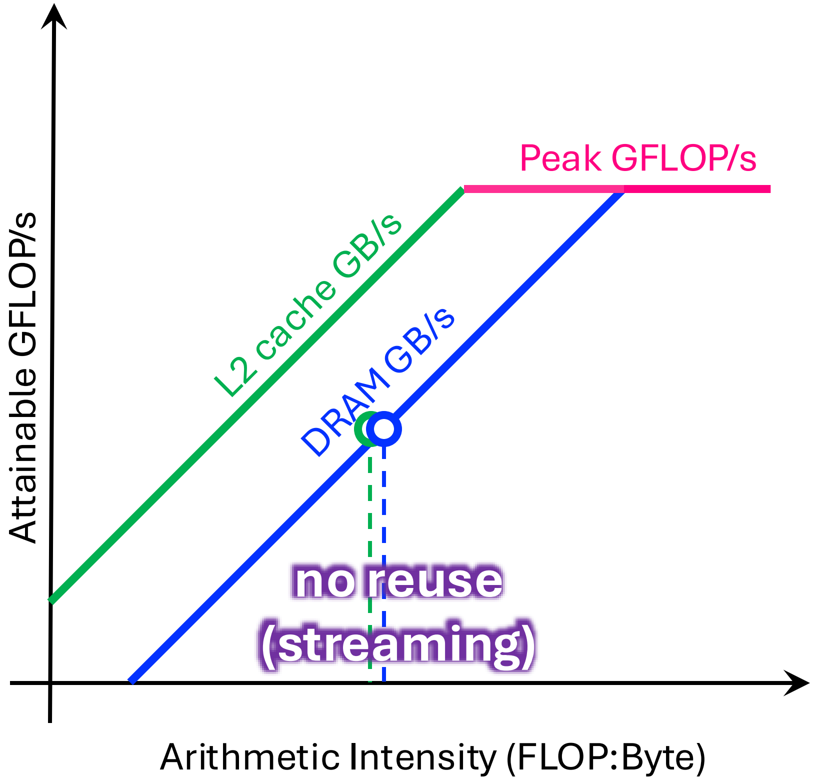
- Widely separated Arithmetic Intensities indicate high reuse in the (L2) cache



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

# Cache Hit Rates

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



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

# Below the Roofline?

## 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)

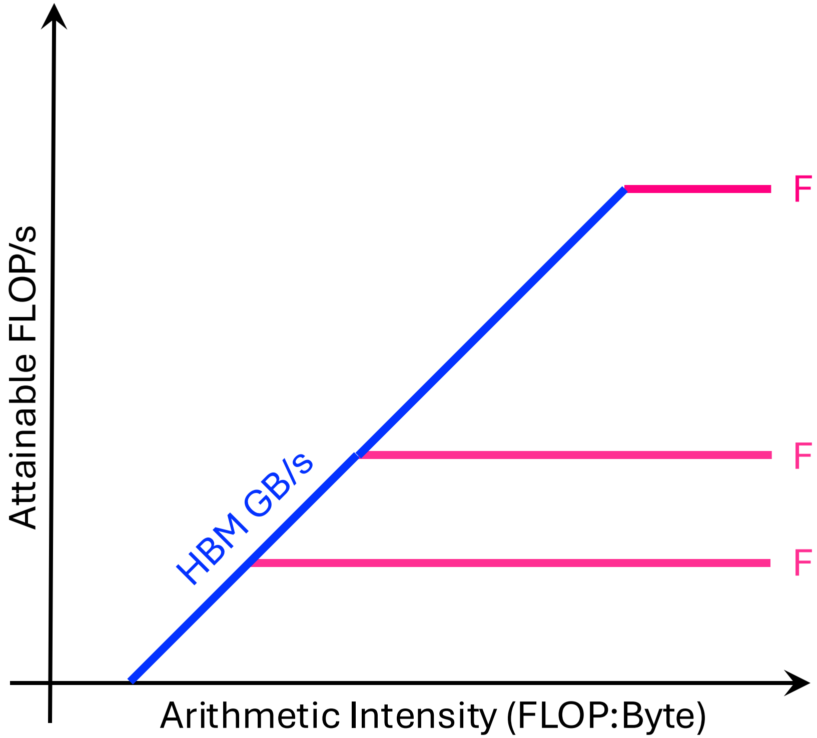
- Modern CPUs and GPUs are increasingly reliant on special (fused) instructions that perform multiple operations (fuse common instruction sequences)…
  - FMA (Fused Multiply Add): z=a*x+y …*z,x,y are vectors or scalars*
  - 4FMA (Quad FMA): z=A*x+z …*A is a FP32 matrix; x,z are vectors*
  - WMMA (Tensor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

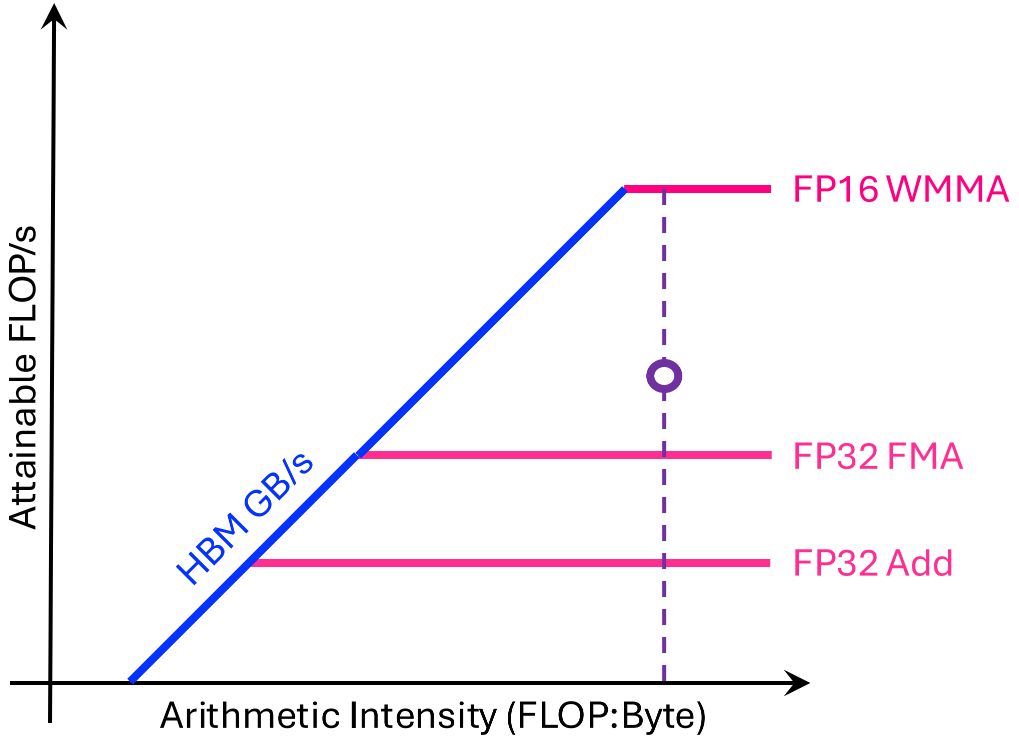
- Consider NVIDIA Volta GPU
- We may define 3 performance ceilings…
  - 15 TFLOPS for FP32 FMA
  - 7.5 TFLOPs for FP32 Add
  - ~100 TFLOPs for FP16 Tensor



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

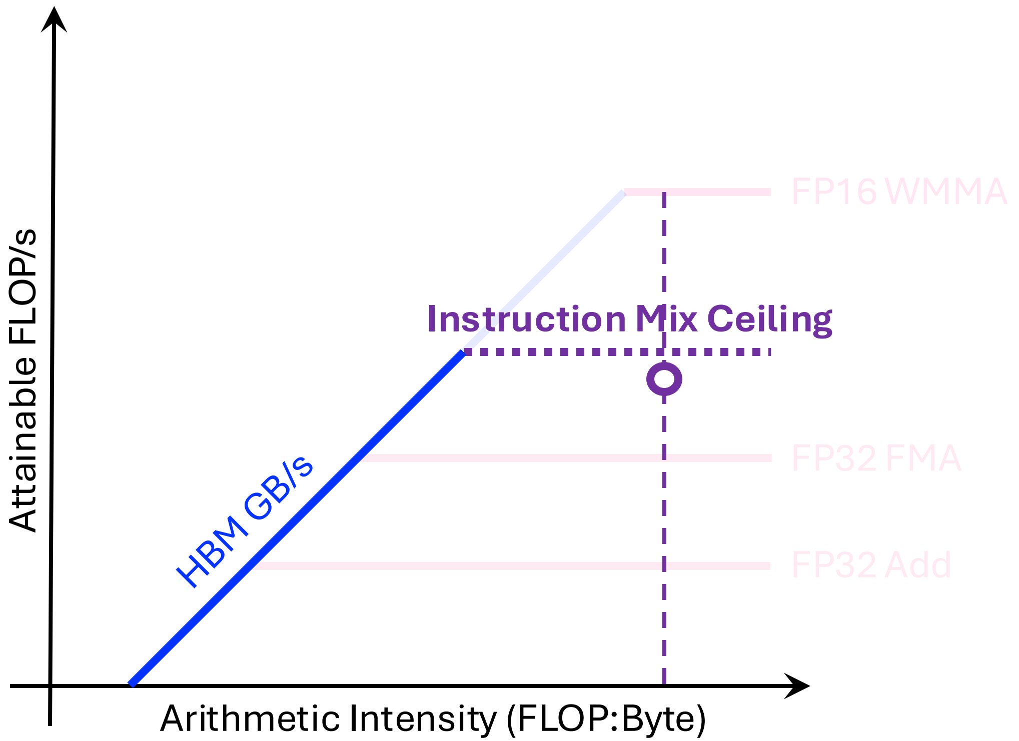
- When calculating (AI,GFLOP/s), count the total FLOPs from all types of instructions
- DL performance can often be well below nominal Tensor Core peak



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

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

$AI_x$ (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…

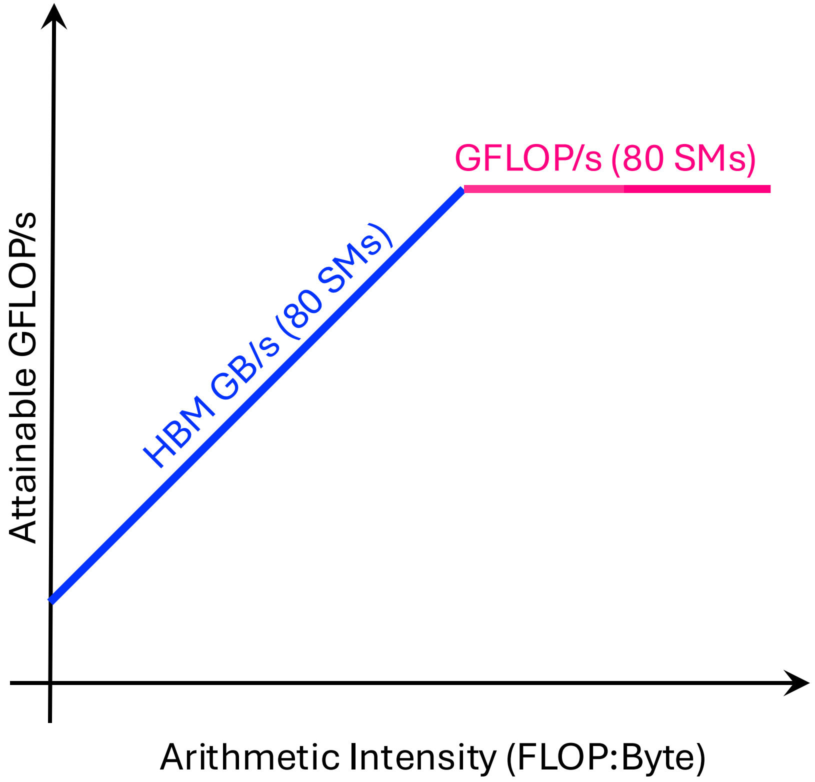
$$\text{GFLOP/s}(P) = \min \begin{cases} \text{GFLOP/s}_{\text{Peak}}(P) \\ \text{AI}_{\text{DRAM}}(P) * \text{GB/s}_{\text{DRAM}}(P) \end{cases}$$

$\text{AI}_x$ (Arithmetic Intensity at level “x”) = FLOPs / Bytes (moved to/from level “x” )

***$\text{AI}_{\text{DRAM}}$ is a function of parallelism because cache contention can generate superfluous LLC capacity misses (==DRAM data movement)***

# Roofline and Parallelism

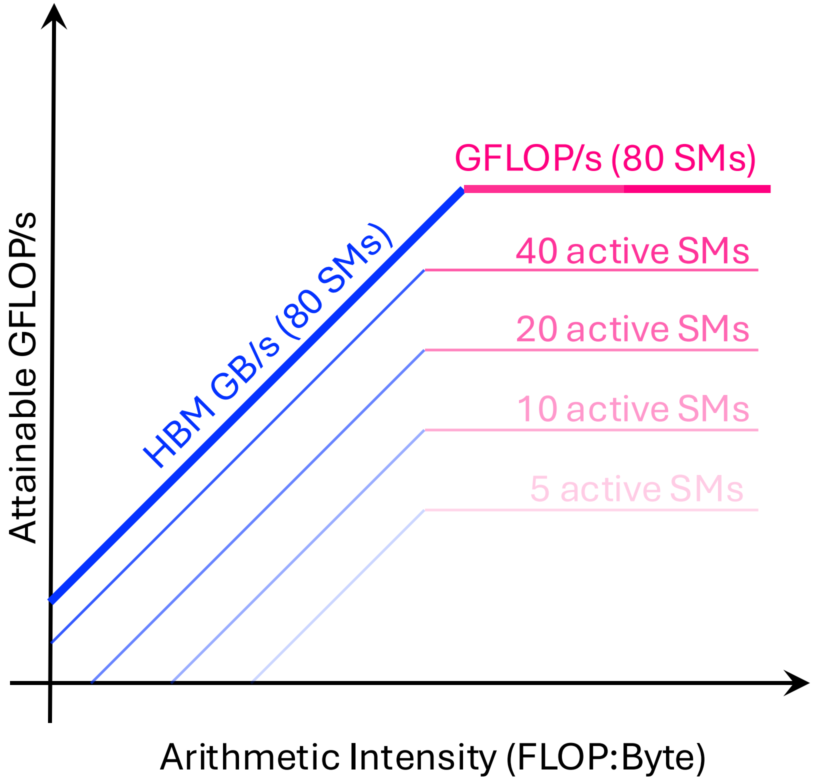
- How do we visualize parallelism in the Roofline?
  - Naively, GFLOP/s(P) and GB/s(P) are proportional to parallelism P
  - SMs are capable of pulling more than their fair share of HBM
  - DVFS implies not true for GFLOP/s



# Roofline and Parallelism

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

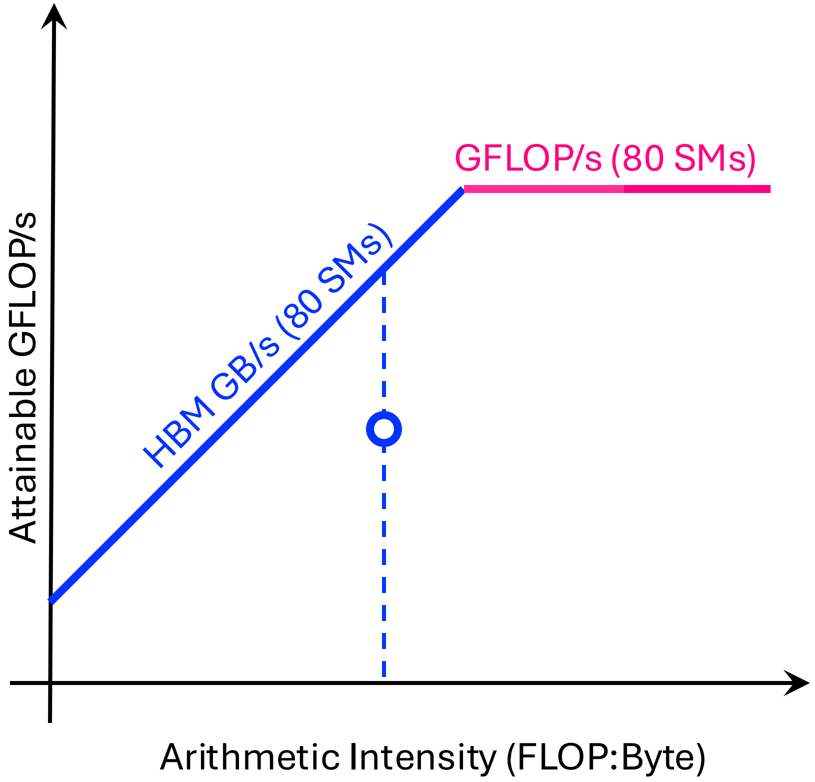
➢ **one must benchmark GFLOP/s and GB/s at each concurrency**



ATPESC2025

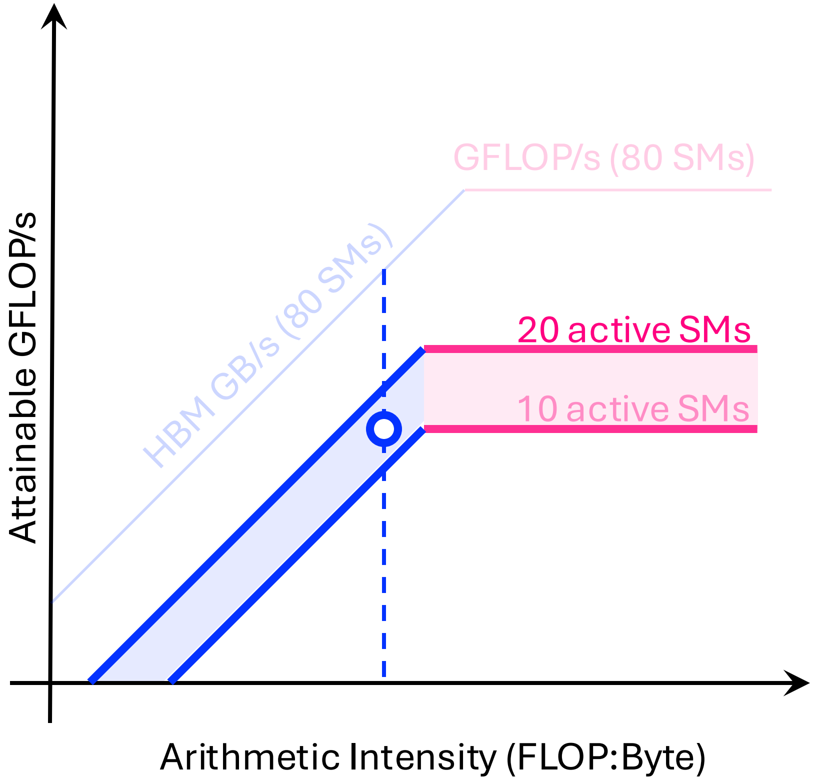
# Roofline and Parallelism

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



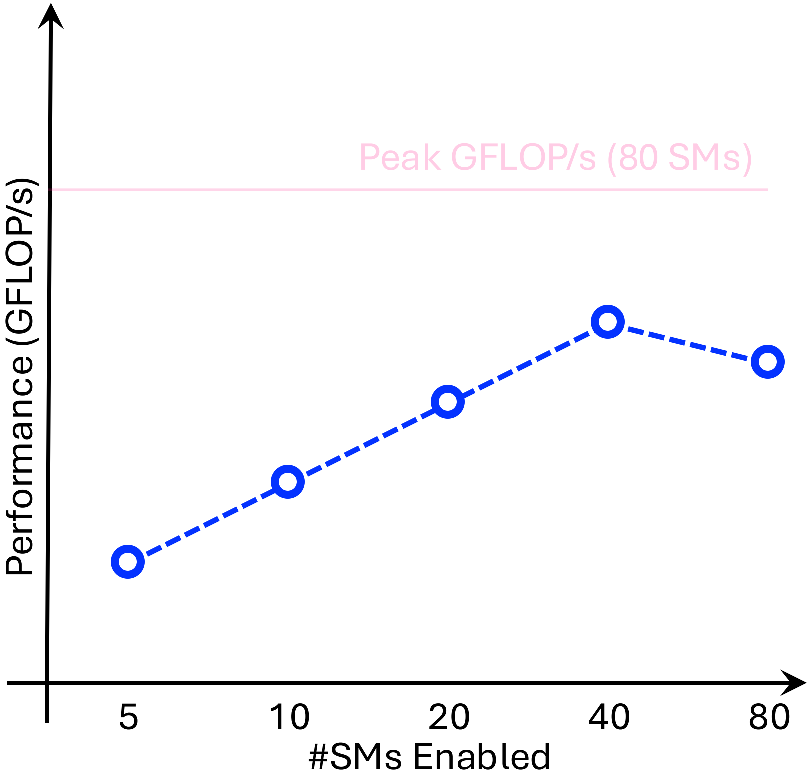
# Roofline and Parallelism

- 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?



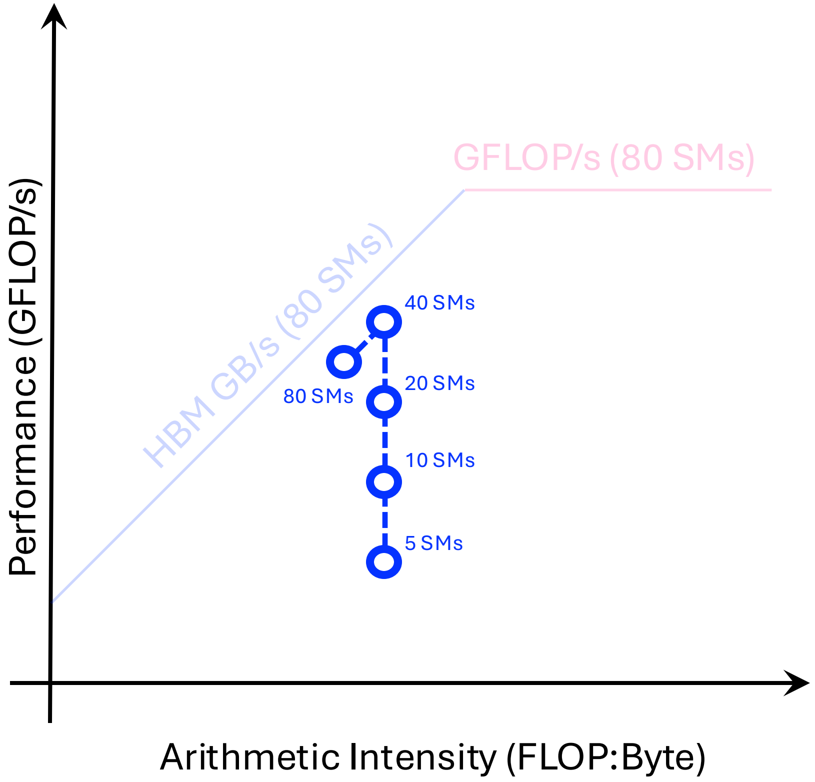
ATPESC2025

# 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

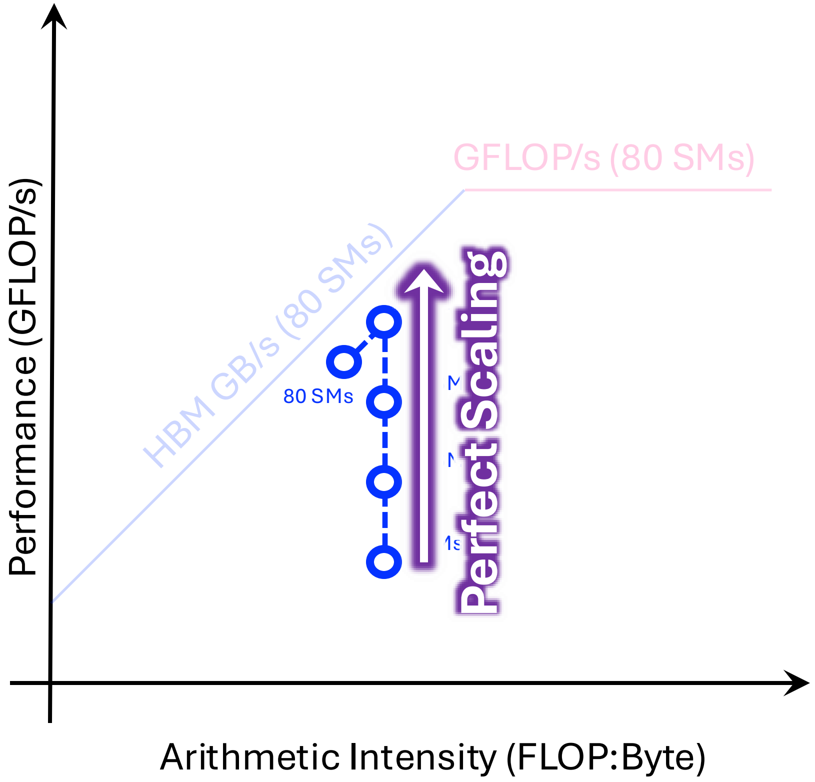


# Roofline Scaling Trajectories

- 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**

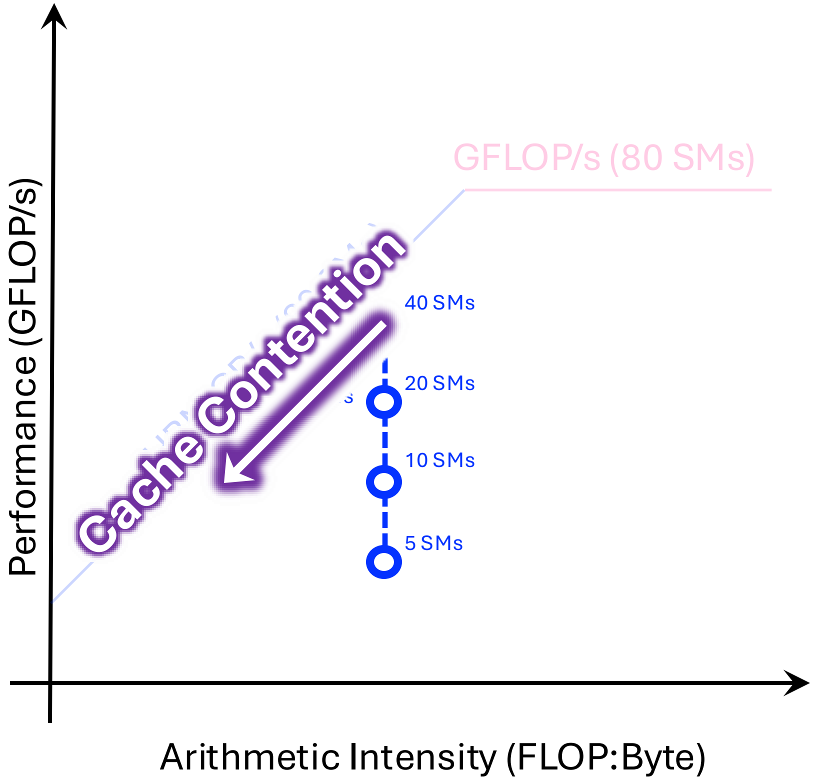


# Roofline Scaling Trajectories

- 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 on Aurora

# Vendor tools for Roofline analysis

- Intel
  - Intel Advisor
    - https://www.intel.com/content/www/us/en/developer/tools/oneapi/advisor.html

- 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

- We won’t try every tool. They have different instructions for the same concept. (I know it is annoying, but that is what we have. ☹)

# Intel Advisor for roofline analysis

# Getting Roofline data in Intel® Advisor: **two-pass approach**

| Roofline :<br>Axis X: **AI** = #**FLOP** / #**Bytes**<br>Axis Y: **FLOP/S** = **#FLOP** (mask aware) / **#Seconds** | Overhead |
|---|---|
| Step 1: Survey (-collect survey)<br>- Provide **#Seconds**<br>- Root access not needed<br>- User mode sampling, non-intrusive. | **1x** |
| Step 2: FLOPS (-collect tripcounts –flops)<br>- Provide #**FLOP,** #**Bytes**, AVX-512 Mask<br>- Root access not needed<br>- Precise, instrumentation based, count number of instructions | **3-5x** |

# Original, Cache-Aware (CARM) and Memory-Level Roofline

## CARM (cache-aware roofline)

- Single **AI** based on aggregated traffic:

  **CPU core (GPU EUs) <-> memory sub-system**
- Ceilings for compute, cache/memory levels
- AI independent of problem size

Unique features: algorithmic focus and simplicity

## Original Roofline

- AI based on external memory :

  **DDR (GPU GTI)**
- Ceilings for DDR and compute
- AI dependent of problem size

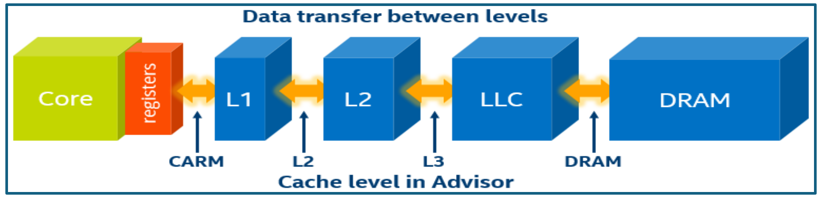
Unique features: DDR bound focus and simplicity

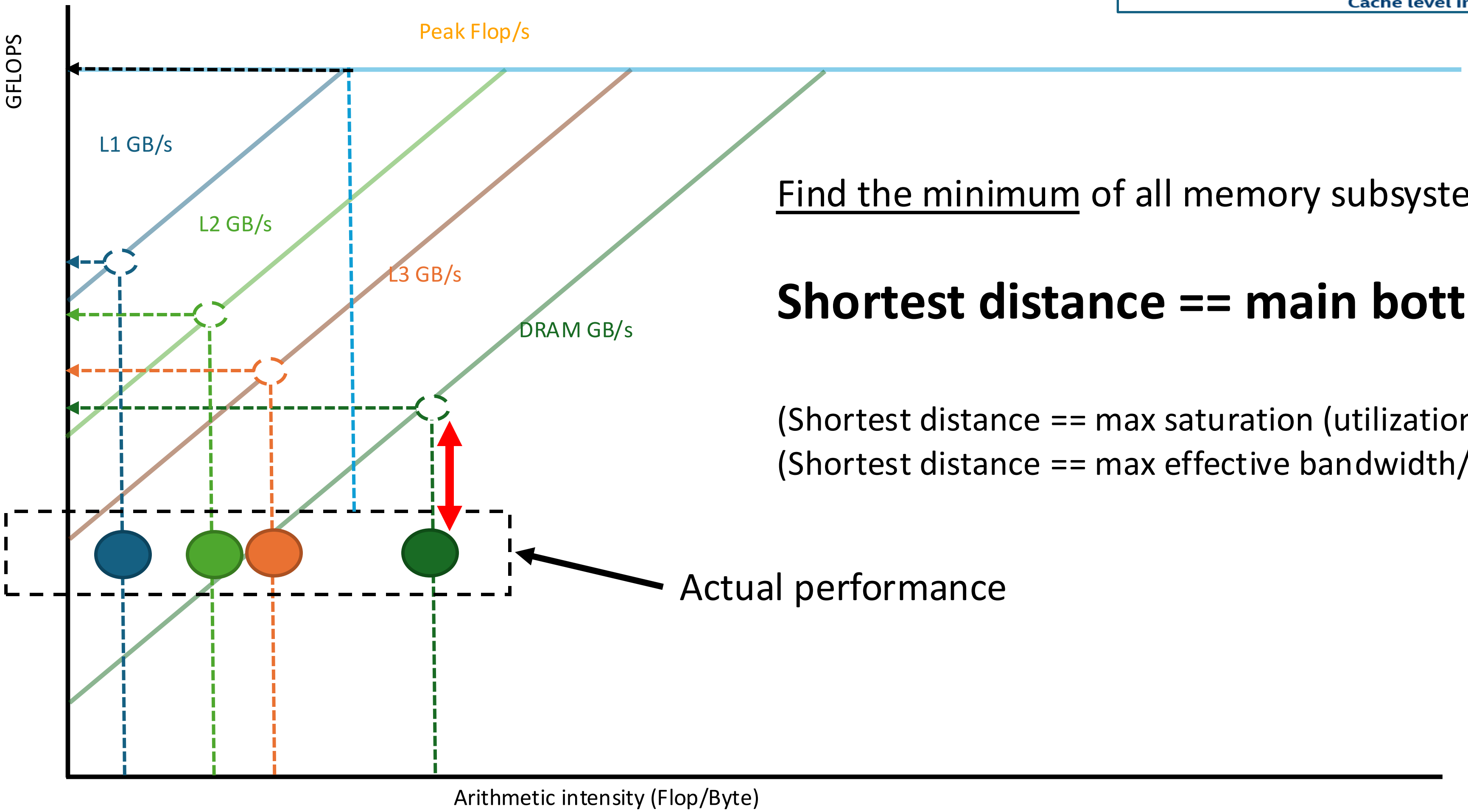
## Memory Level Roofline - **MLR** (see also “Hierarchical Roofline” by LBL)

- **AI** for all memory sub-system levels, combines (1), CARM, (2)Original and (3) Lx-only perspectives
- Harder to interpret for multiple kernels at a time

Unique features: unambiguous bottleneck detection

# How to interpret MLR on CPU ?





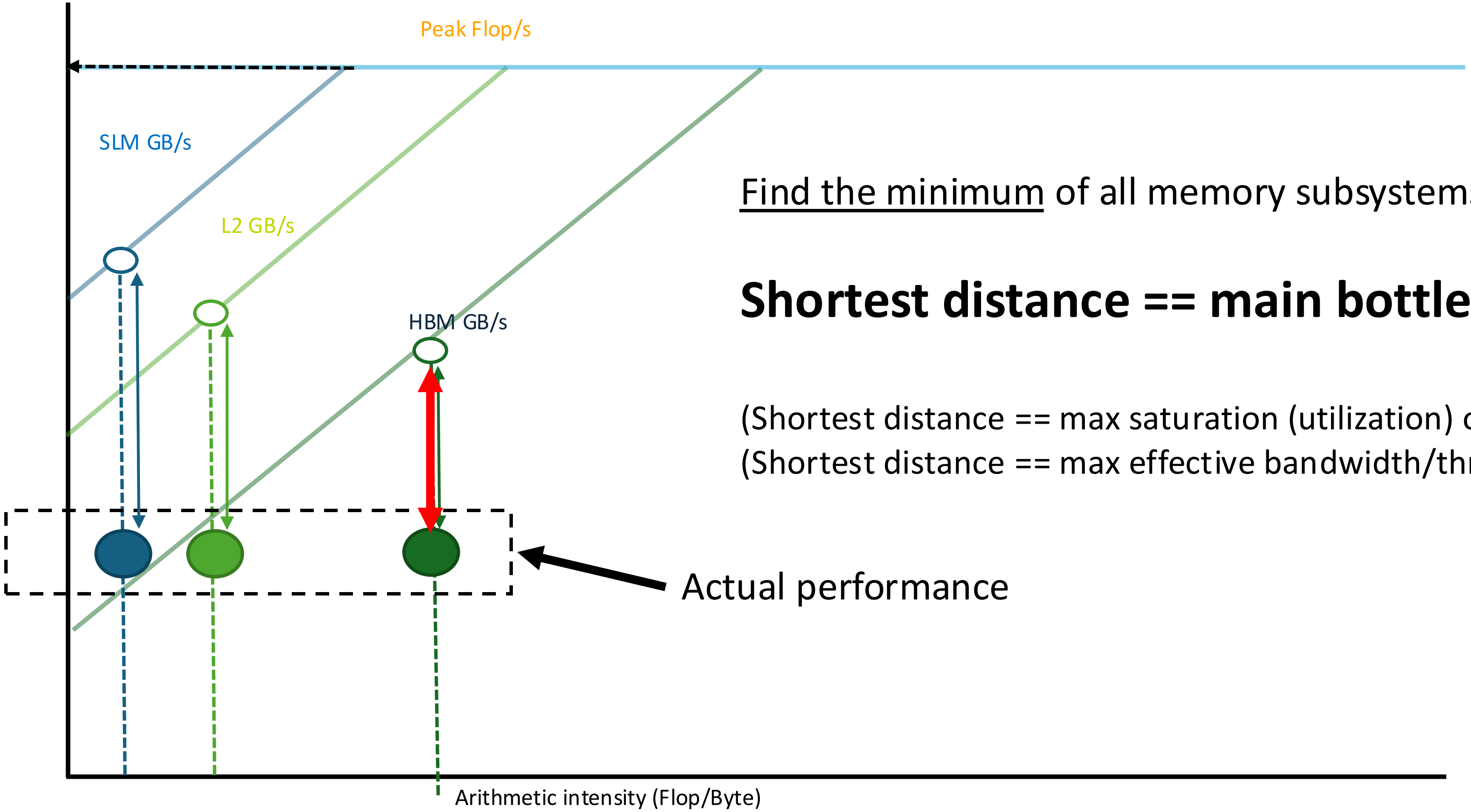
Actual performance

Find the minimum of all memory subsystems

**Shortest distance == main bottleneck**

(Shortest distance == max saturation (utilization) observed)
(Shortest distance == max effective bandwidth/throughput observed)

# How to interpret MLR on GPU ?



Actual performance

Find the minimum of all memory subsystems

**Shortest distance == main bottleneck**

(Shortest distance == max saturation (utilization) observed)
(Shortest distance == max effective bandwidth/throughput observed)

# How to generate (profile*) Roofline for your application

# How to generate **CARM** CPU Roofline profile?

```
As simple as: $ advisor -collect roofline -- <your-executable-with-parameters>
```

## More details / How-To

```
$ source advisor-vars.sh
```

**1st method. Not compatible with MPI applications :**

```
$ advisor -collect roofline --project-dir ./your_project -- <your-executable-with-parameters>
```

**2nd method (compatible with MPI, more flexible):**

```
$ advisor -collect survey --project-dir ./your_project -- <your-executable-with-parameters>

$ advisor -collect tripcounts --flop --project-dir ./your_project -- <your-executable-with-parameters>
```

```
(optional) copy data to your UI desktop system

$ advisor-gui ./your_project

$ advisor -report roofline --project-dir ./your_project > roofline.html
```

# How to generate **MLR**+CARM CPU Roofline profile?

```
As simple as: $ advisor -collect roofline -enable-cache-simulation -- <your-executable-with-parameters>
```

## More details / How-To

```
$ source advisor-vars.sh
```

**1st method. Not compatible with MPI applications :**

```
$ advisor -collect roofline -enable-cache-simulation --project-dir ./your_project -- <your-executable-with-parameters>
```

**2nd method (compatible with MPI, more flexible):**

```
$ advisor -collect survey --project-dir ./your_project -- <your-executable-with-parameters>
```

```
$ advisor -collect tripcounts -flop -enable-cache-simulation --project-dir ./your_project -- <your-executable-with-parameters>
```

*(optional) copy data to your UI desktop system*

```
$ advisor-gui ./your_project
```

```
$ advisor -report roofline --project-dir ./your_project > roofline.html
```

# How to generate **GPU** (MLR & CARM) Roofline profile?

```
As simple as: $ advisor -collect roofline --profile-gpu -- <your-executable-with-parameters>
```

## More details / How-To

```
$ source advisor-vars.sh
```

**1st method. Not compatible with MPI applications :**

```
$ advisor -collect roofline --profile-gpu --project-dir ./your_project -- <your-executable-with-parameters>
```

**2nd method (compatible with MPI, more flexible):**

```
$ advisor -collect survey --profile-gpu --project-dir ./your_project -- <your-executable-with-parameters>

$ advisor -collect tripcounts -flop --profile-gpu --project-dir ./your_project -- <your-executable-with-parameters>
```

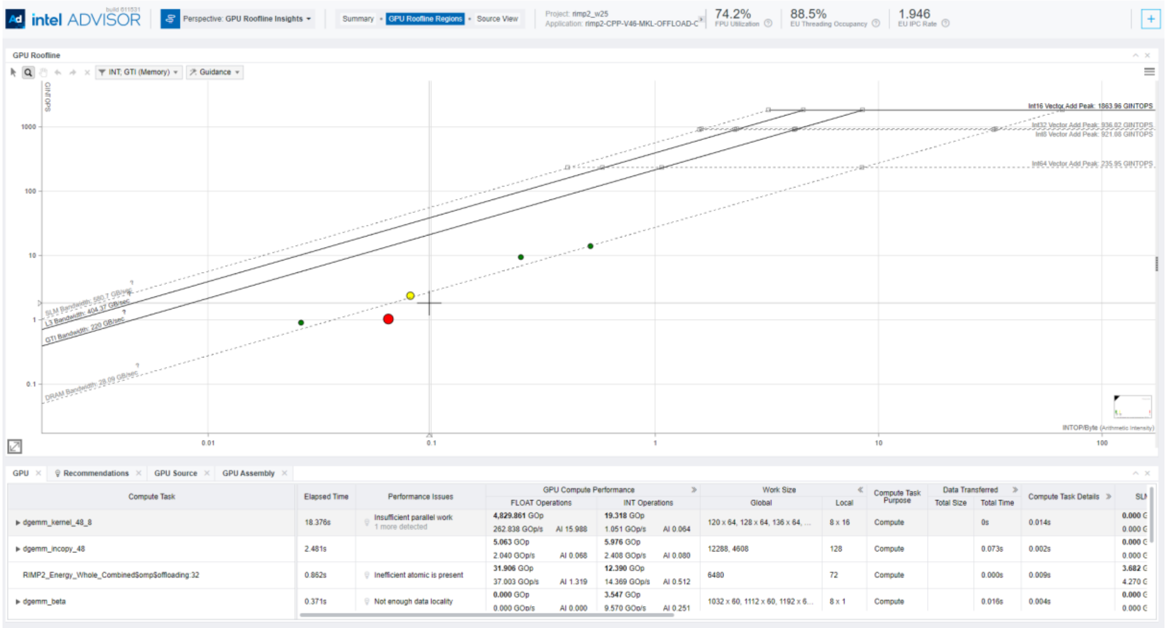
*(optional) copy data to your UI desktop system*

```
$ advisor-gui ./your_project

$ advisor -report roofline --gpu --project-dir ./your_project > roofline.html
```

# GPU Roofline: *Extended* HTML GUI

## See HTML report in project-dir/e000|rank.*/report folder by default

```
source advisor_install_dir/advisor-vars.sh

advisor
 --report all
 --project-dir ./your_project
 --report-output ./roofline.html
```


intel ADVISOR
build 611531
Perspective: GPU Roofline Insights
Summary
GPU Roofline Regions
Source View
GPU Roofline
INT, GTI (Memory)
Guidance
GINTOPS
1000
100
10
1
0.1
0.01
0.1
1
10
100
INTOP/Byte (Arithmetic Intensity)
Int16 Vector Add Peak: 1863.96 GINTOPS
Int32 Vector Add Peak: 936.02 GINTOPS
Int8 Vector Add Peak: 921.08 GINTOPS
Int64 Vector Add Peak: 235.95 GINTOPS
SLM Bandwidth: 580.7 GB/sec
L3 Bandwidth: 404.37 GB/sec
GTI Bandwidth: 220 GB/sec
DRAM Bandwidth: 28.09 GB/sec
GPU
Recommendations
GPU Source
GPU Assembly

Project: rimp2_w25
Application: rimp2-CPP-V46-MKL-OFFLOAD-C

74.2% FPU Utilization

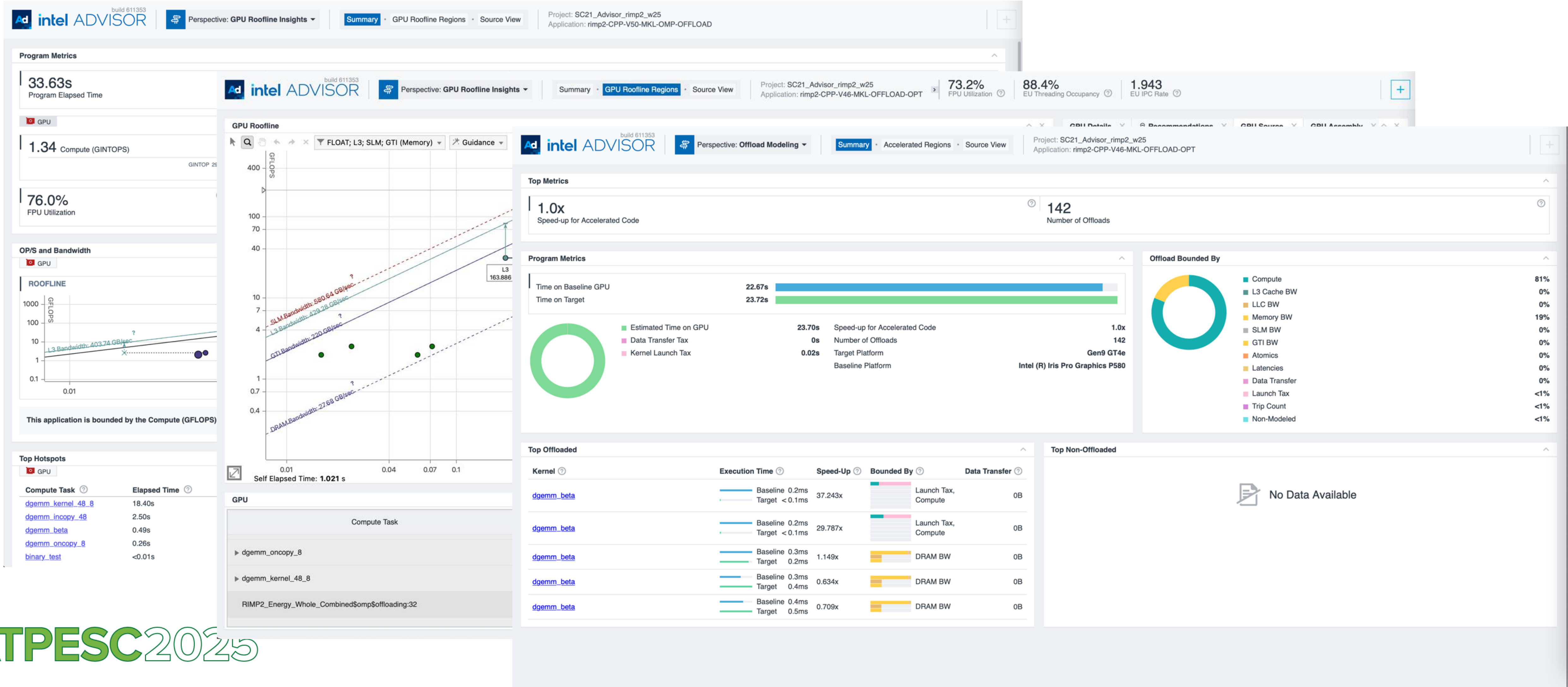
88.5% EU Threading Occupancy

1.946 EU IPC Rate

| Compute Task | Elapsed Time | Performance Issues | GPU Compute Performance |  | Work Size |  | Compute Task Purpose | Data Transferred |  | Compute Task Details | SL |
|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  | FLOAT Operations | INT Operations | Global | Local |  | Total Size | Total Time |  |  |
| ▶ dgemm_kernel_48_8 | 18.376s | Insufficient parallel work<br>1 more detected | **4,829.861** GOp<br>262.838 GOp/s AI 15.988 | **19.318** GOp<br>1.051 GOp/s AI 0.064 | 120 x 64, 128 x 64, 136 x 64, ... | 8 x 16 | Compute |  | 0s | 0.014s | **0.000** G<br>0.000 G |
| ▶ dgemm_incopy_48 | 2.481s |  | **5.063** GOp<br>2.040 GOp/s AI 0.068 | **5.976** GOp<br>2.408 GOp/s AI 0.080 | 12288, 4608 | 128 | Compute |  | 0.073s | 0.002s | **0.000** G<br>0.000 G |
| RIMP2_Energy_Whole_Combined$omp$offloading:32 | 0.862s | Inefficient atomic is present | **31.906** GOp<br>37.003 GOp/s AI 1.319 | **12.390** GOp<br>14.369 GOp/s AI 0.512 | 6480 | 72 | Compute |  | 0.000s | 0.009s | **3.682** G<br>4.270 G |
| ▶ dgemm_beta | 0.371s | Not enough data locality | **0.000** GOp<br>0.000 GOp/s AI 0.000 | **3.547** GOp<br>9.570 GOp/s AI 0.251 | 1032 x 60, 1112 x 60, 1192 x 6... | 8 x 1 | Compute |  | 0.016s | 0.004s | **0.000** G<br>0.000 G |

# *Extended* HTML GUI

## For any system with web browsers



Project: SC21_Advisor_rimp2_w25
Application: rimp2-CPP-V50-MKL-OMP-OFFLOAD

**Program Metrics**

33.63s
Program Elapsed Time

GPU

1.34 Compute (GINTOPS)

GINTOP 2

76.0%
FPU Utilization

**OP/S and Bandwidth**

GPU

ROOFLINE

This application is bounded by the Compute (GFLOPS)

**Top Hotspots**

GPU

| Compute Task | Elapsed Time |
|---|---|
| dgemm_kernel_48_8 | 18.40s |
| dgemm_incopy_48 | 2.50s |
| dgemm_beta | 0.49s |
| dgemm_oncopy_8 | 0.26s |
| binary_test | <0.01s |

Project: SC21_Advisor_rimp2_w25
Application: rimp2-CPP-V46-MKL-OFFLOAD-OPT

73.2%
FPU Utilization

88.4%
EU Threading Occupancy

1.943
EU IPC Rate

**GPU Roofline**

FLOAT; L3; SLM; GTI (Memory)
Guidance
GFLOPS
400
100
70
40
10
7
4
1
0.7
0.4
0.01
0.04
0.07
0.1
SLM Bandwidth: 580.64 GB/sec
L3 Bandwidth: 429.28 GB/sec
GTI Bandwidth: 220 GB/sec
DRAM Bandwidth: 27.68 GB/sec
?
L3
163.886

Self Elapsed Time: **1.021** s

**GPU**

Compute Task

- dgemm_oncopy_8
- dgemm_kernel_48_8

RIMP2_Energy_Whole_Combined$omp$offloading:32

Project: SC21_Advisor_rimp2_w25
Application: rimp2-CPP-V46-MKL-OFFLOAD-OPT

**Top Metrics**

1.0x
Speed-up for Accelerated Code

142
Number of Offloads

**Program Metrics**

| | |
|---|---|
| Time on Baseline GPU | 22.67s |
| Time on Target | 23.72s |

| | |
|---|---|
| 🟩 Estimated Time on GPU | 23.70s |
| 🟪 Data Transfer Tax | 0s |
| 🟪 Kernel Launch Tax | 0.02s |

| | |
|---|---|
| Speed-up for Accelerated Code | 1.0x |
| Number of Offloads | 142 |
| Target Platform | Gen9 GT4e |
| Baseline Platform | Intel (R) Iris Pro Graphics P580 |

**Offload Bounded By**

| | |
|---|---|
| Compute | 81% |
| L3 Cache BW | 0% |
| LLC BW | 0% |
| Memory BW | 19% |
| SLM BW | 0% |
| GTI BW | 0% |
| Atomics | 0% |
| Latencies | 0% |
| Data Transfer | 0% |
| Launch Tax | <1% |
| Trip Count | <1% |
| Non-Modeled | <1% |

**Top Offloaded**

| Kernel | Execution Time | Speed-Up | Bounded By | Data Transfer |
|---|---|---|---|---|
| dgemm_beta | Baseline 0.2ms<br>Target < 0.1ms | 37.243x | Launch Tax, Compute | 0B |
| dgemm_beta | Baseline 0.2ms<br>Target < 0.1ms | 29.787x | Launch Tax, Compute | 0B |
| dgemm_beta | Baseline 0.3ms<br>Target 0.2ms | 1.149x | DRAM BW | 0B |
| dgemm_beta | Baseline 0.3ms<br>Target 0.4ms | 0.634x | DRAM BW | 0B |
| dgemm_beta | Baseline 0.4ms<br>Target 0.5ms | 0.709x | DRAM BW | 0B |

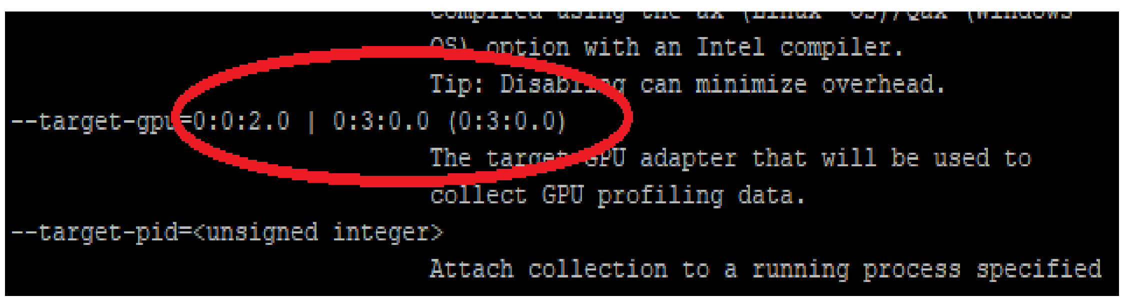
**Top Non-Offloaded**

No Data Available

# Roofline on Multi-GPU systems

Add --target-gpu option in command line

```
advisor
 --collect roofline
 --profile-gpu
 --project-dir  ./your_project
 --target-gpu 0:77:0.0
 -- <your-executable-with-
parameters>
```



```
                              compiled using the ax (Linux  OS)/Qax (Windows
                              OS) option with an Intel compiler.
                              Tip: Disabling can minimize overhead.
--target-gpu=0:0:2.0 | 0:3:0.0 (0:3:0.0)
                              The target GPU adapter that will be used to
                              collect GPU profiling data.
--target-pid=<unsigned integer>
                              Attach collection to a running process specified
```

# ISO3DFD Code

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

# ISO3dfd code

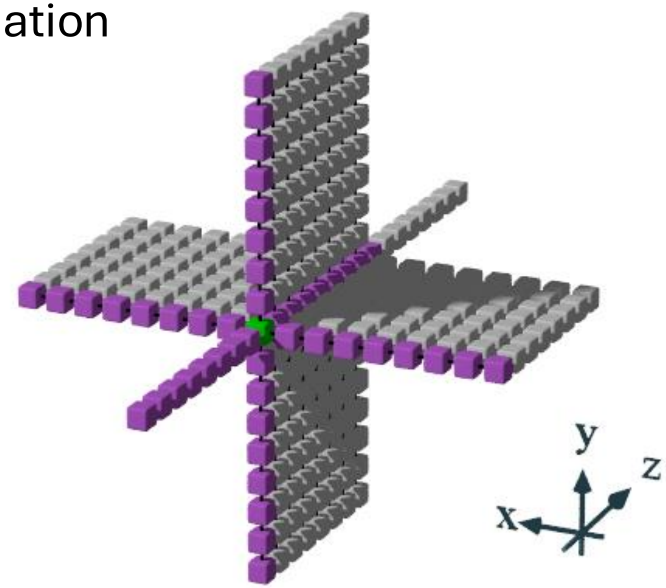
- A Finite Difference stencil kernel for solving the 3D acoustic isotropic wave equation
  - A proxy for propagating a seismic wave
  - 16th order in space, with symmetric coefficients
  - 2nd order in time scheme without boundary conditions.
- Problem Statement
  - Partial Differential Equation (PDE)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

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

, 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 each direction of 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 $z$.

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

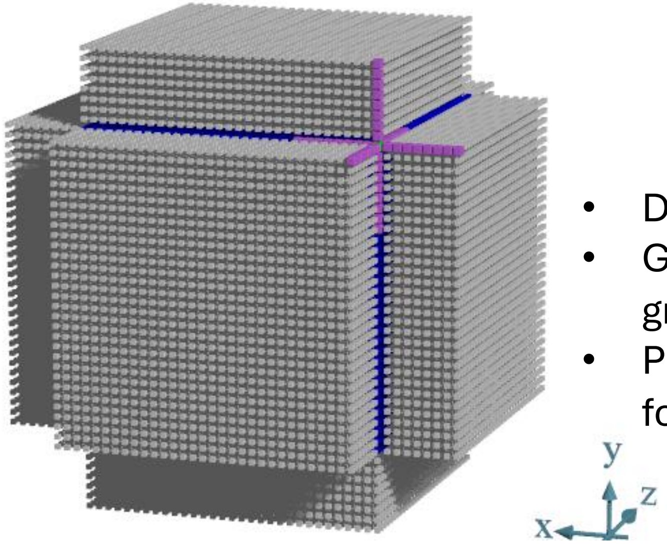


https://www.intel.com/content/www/us/en/developer/articles/technical/iso3dfd-code-walkthrough.html

# 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.

```
…
 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);
  }
```



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

# ISO3dFD **code versions**

- In this hands-on, we use four ISO3DFD variants.
- These variants add progressive/incremental levels of optimization as follow:
  - **1_CPU_only.cpp**: an initial CPU version
  - **2_GPU_basic.cpp**: basic GPU offloading using SYCL
  - **3_GPU_linear.cpp**: reduced index calculation
  - **4_GPU_private_memory_I.cpp:** addition of private array for coefficients

*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=02:00:00 -l filesystems=home:flare  -A ATPESC2025 -q ATPESC

$ cd /flare/ATPESC2025/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/ATPESC2025/EXAMPLES/track6-tools/roofline/oneAPI-samples/DirectProgramming/C++SYCL/StructuredGrids/guided_iso3dfd_GPUOptimization/ .
$ cd guided_iso3dfd_GPUOptimization

$ mkdir build
$ cd build

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

# 1_CPU_only.cpp

```
…
for (auto iz = kHalfLength; iz < n3_end; iz++) {
    for (auto iy = kHalfLength; iy < n2_end; iy++) {
      // Calculate start pointers for the row over X dimension
      float* ptr_next = ptr_next_base + iz * dimn1n2 + iy * n1;
      float* ptr_prev = ptr_prev_base + iz * dimn1n2 + iy * n1;
      float* ptr_vel = ptr_vel_base + iz * dimn1n2 + iy * n1;

      // Iterate over X
      for (auto ix = kHalfLength; ix < n1_end; ix++) {
        // Calculate values for each cell
        float value = ptr_prev[ix] * coeff[0];
        for (int i = 1; i <= kHalfLength; i++) {
          value +=
              coeff[i] *
               (ptr_prev[ix + i] + ptr_prev[ix - i] +
                ptr_prev[ix + i * n1] + ptr_prev[ix - i * n1] +
                ptr_prev[ix + i * dimn1n2] + ptr_prev[ix - i * dimn1n2]);
        }
        ptr_next[ix] = 2.0f * ptr_prev[ix] - ptr_next[ix] + value * ptr_vel[ix];
      }
    }
}
…
```

**Computing values for each cell**

**Iterate over x**

**Iterate over y and z**

# Run 1_CPU_only

```
$ src/1_CPU_only 512 512 512 10
Running on CPU serial version
--------------------------------------
time         : 11.309 secs
throughput   : 118.682 Mpts/s
flops        : 7.23962 GFlops
bytes        : 1.42419 GBytes/s
--------------------------------------
```

# 2_GPU_basic: offloading the CPU code to GPU using SYCL

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

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

**Computing values for each cell**

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

# Run 2_GPU_basic

```
$ export ZE_AFFINITY_MASK=0.0
$ src/2_GPU_basic 512 512 512 100
 Running GPU basic offload 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          : 7.25 secs
throughput    : 1851.28 Mpts/s
flops         : 112.928 GFlops
bytes         : 22.2153 GBytes/s
--------------------------------------
```

Compared 113.09 sec on CPU for 100 steps

- 15.6X speed-up on GPU from on a CPU core

```
$ advisor -collect roofline --profile-gpu --project-dir ADV_02_512 --  ./src/2_GPU_basic 512 512 512 100
```

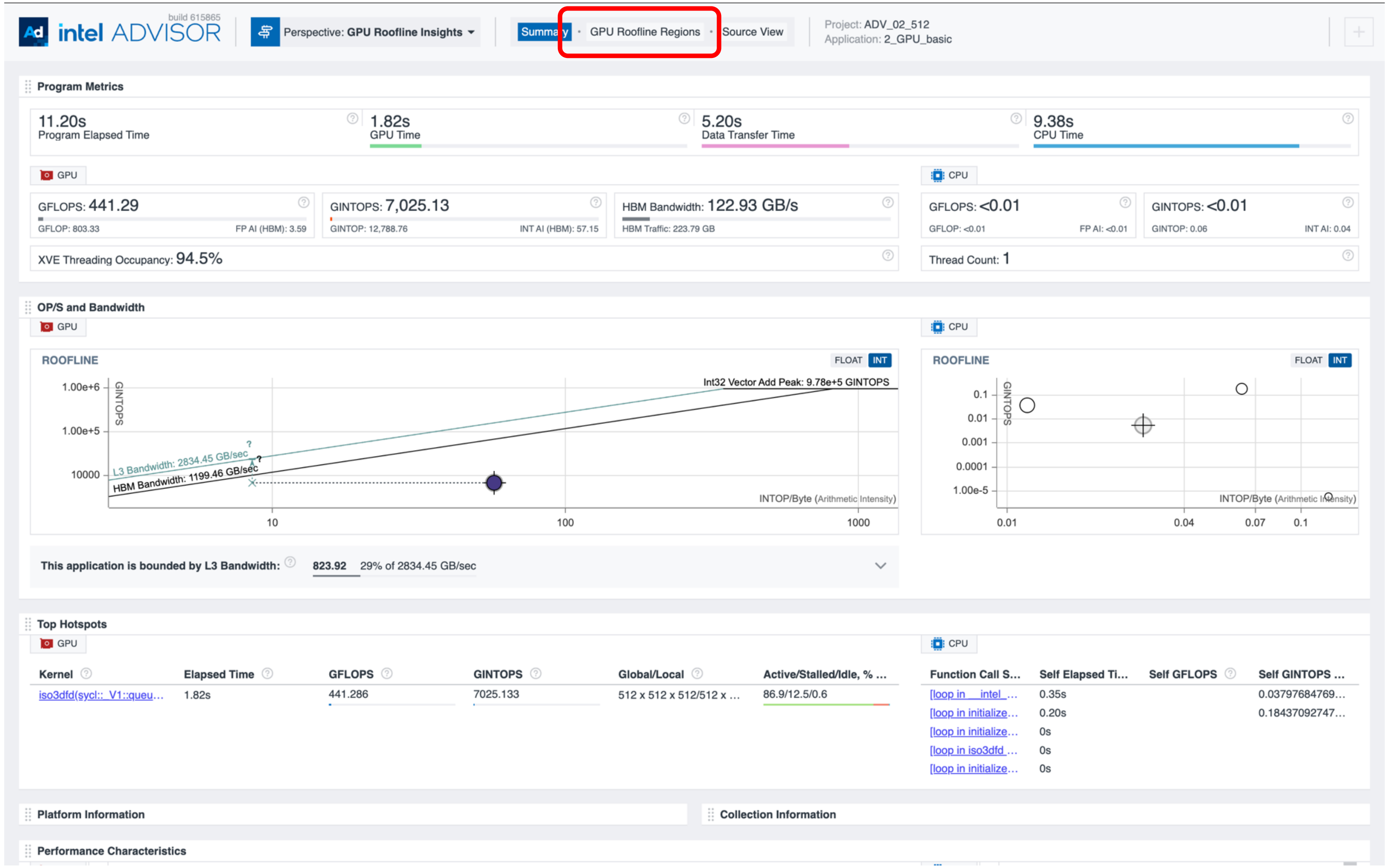
or

```
$ cp /flare/ATPESC2025/EXAMPLES/track6-tools/roofline/ADV_results/ADV_02_512 .

# Download advisor-report.html from /ADV_02_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”)**



intel ADVISOR build 615865 | Perspective: **GPU Roofline Insights** | Summary · GPU Roofline Regions · Source View | Project: ADV_02_512 Application: 2_GPU_basic

**Program Metrics**

11.20s Program Elapsed Time | 1.82s GPU Time | 5.20s Data Transfer Time | 9.38s CPU Time

GPU

GFLOPS: 441.29 — GFLOP: 803.33 — FP AI (HBM): 3.59

GINTOPS: 7,025.13 — GINTOP: 12,788.76 — INT AI (HBM): 57.15

HBM Bandwidth: 122.93 GB/s — HBM Traffic: 223.79 GB

XVE Threading Occupancy: 94.5%

CPU

GFLOPS: <0.01 — GFLOP: <0.01 — FP AI: <0.01

GINTOPS: <0.01 — GINTOP: 0.06 — INT AI: 0.04

Thread Count: 1

**OP/S and Bandwidth**

GPU
ROOFLINE
FLOAT
INT
1.00e+6
1.00e+5
10000
GINTOPS
Int32 Vector Add Peak: 9.78e+5 GINTOPS
L3 Bandwidth: 2834.45 GB/sec
HBM Bandwidth: 1199.46 GB/sec
INTOP/Byte (Arithmetic Intensity)
10
100
1000
CPU
ROOFLINE
FLOAT
INT
0.1
0.01
0.001
0.0001
1.00e-5
GINTOPS
INTOP/Byte (Arithmetic Intensity)
0.01
0.04
0.07
0.1

This application is bounded by L3 Bandwidth: 823.92 29% of 2834.45 GB/sec

**Top Hotspots**

GPU

| Kernel | Elapsed Time | GFLOPS | GINTOPS | Global/Local | Active/Stalled/Idle, % ... |
|---|---|---|---|---|---|
| iso3dfd(sycl::_V1::queu... | 1.82s | 441.286 | 7025.133 | 512 x 512 x 512/512 x ... | 86.9/12.5/0.6 |

CPU

| Function Call S... | Self Elapsed Ti... | Self GFLOPS | Self GINTOPS ... |
|---|---|---|---|
| [loop in __intel_... | 0.35s | | 0.03797684769... |
| [loop in initialize... | 0.20s | | 0.18437092747... |
| [loop in initialize... | 0s | | |
| [loop in iso3dfd ... | 0s | | |
| [loop in initialize... | 0s | | |

**Platform Information**

**Collection Information**

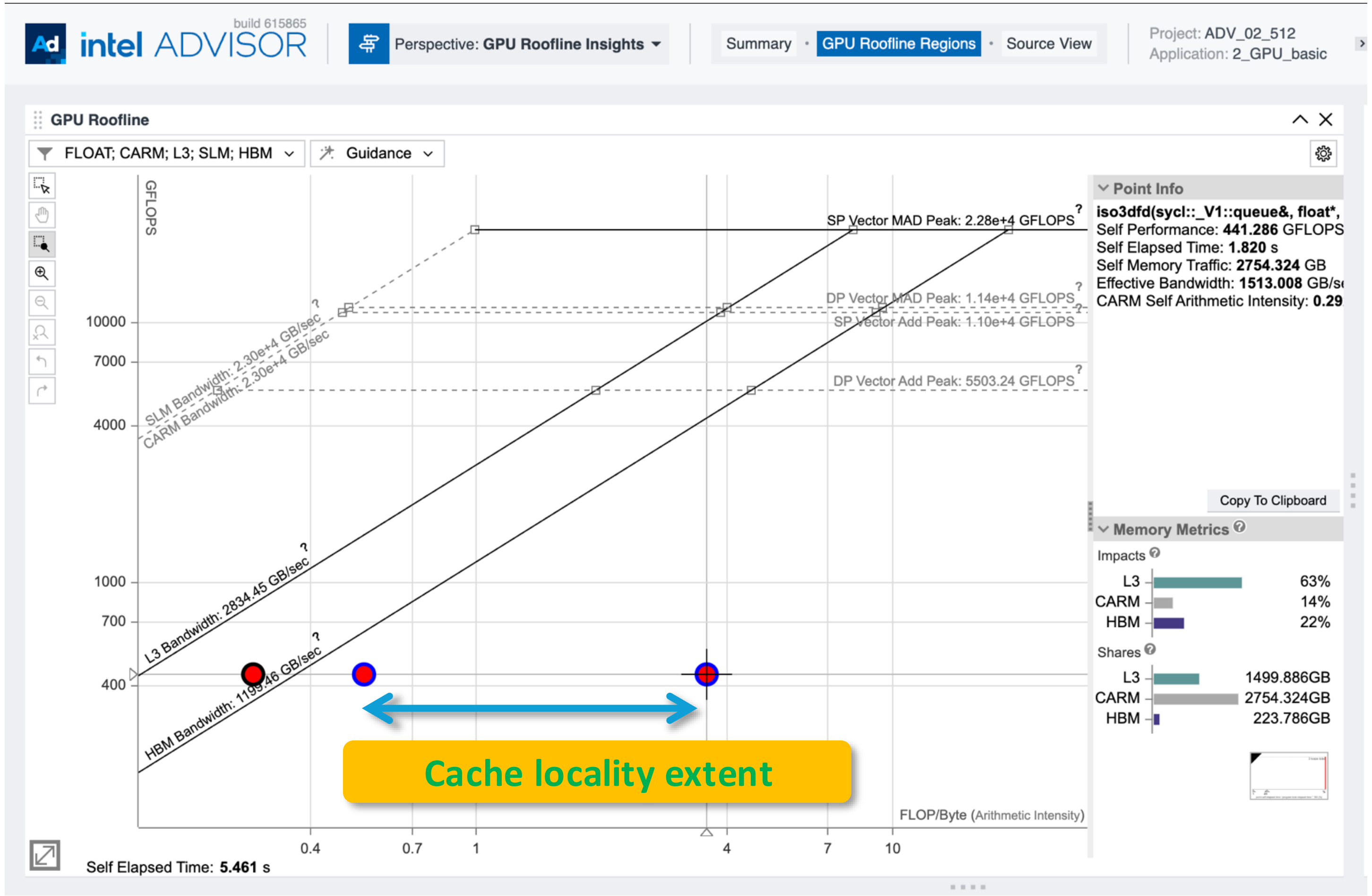
**Performance Characteristics**

# 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)


build 615865
intel ADVISOR
Perspective: GPU Roofline Insights
Summary
GPU Roofline Regions
Source View
GPU Roofline
FLOAT; CARM; L3; SLM; HBM
Guidance
GFLOPS
10000
7000
4000
1000
700
400
SP Vector MAD Peak: 2.28e+4 GFLOPS
DP Vector MAD Peak: 1.14e+4 GFLOPS
SP Vector Add Peak: 1.10e+4 GFLOPS
DP Vector Add Peak: 5503.24 GFLOPS
SLM Bandwidth: 2.30e+4 GB/sec
CARM Bandwidth: 2.30e+4 GB/sec
L3 Bandwidth: 2834.45 GB/sec
HBM Bandwidth: 1199.46 GB/sec
FLOP/Byte (Arithmetic Intensity)
0.4
0.7
1
4
7
10
Point Info
Copy To Clipboard
Memory Metrics
Impacts
Shares

Project: ADV_02_512
Application: 2_GPU_basic

**iso3dfd(sycl::_V1::queue&, float*,**
Self Performance: **441.286** GFLOPS
Self Elapsed Time: **1.820** s
Self Memory Traffic: **2754.324** GB
Effective Bandwidth: **1513.008** GB/s
CARM Self Arithmetic Intensity: **0.29**

Impacts

| | |
|---|---|
| L3 | 63% |
| CARM | 14% |
| HBM | 22% |

Shares

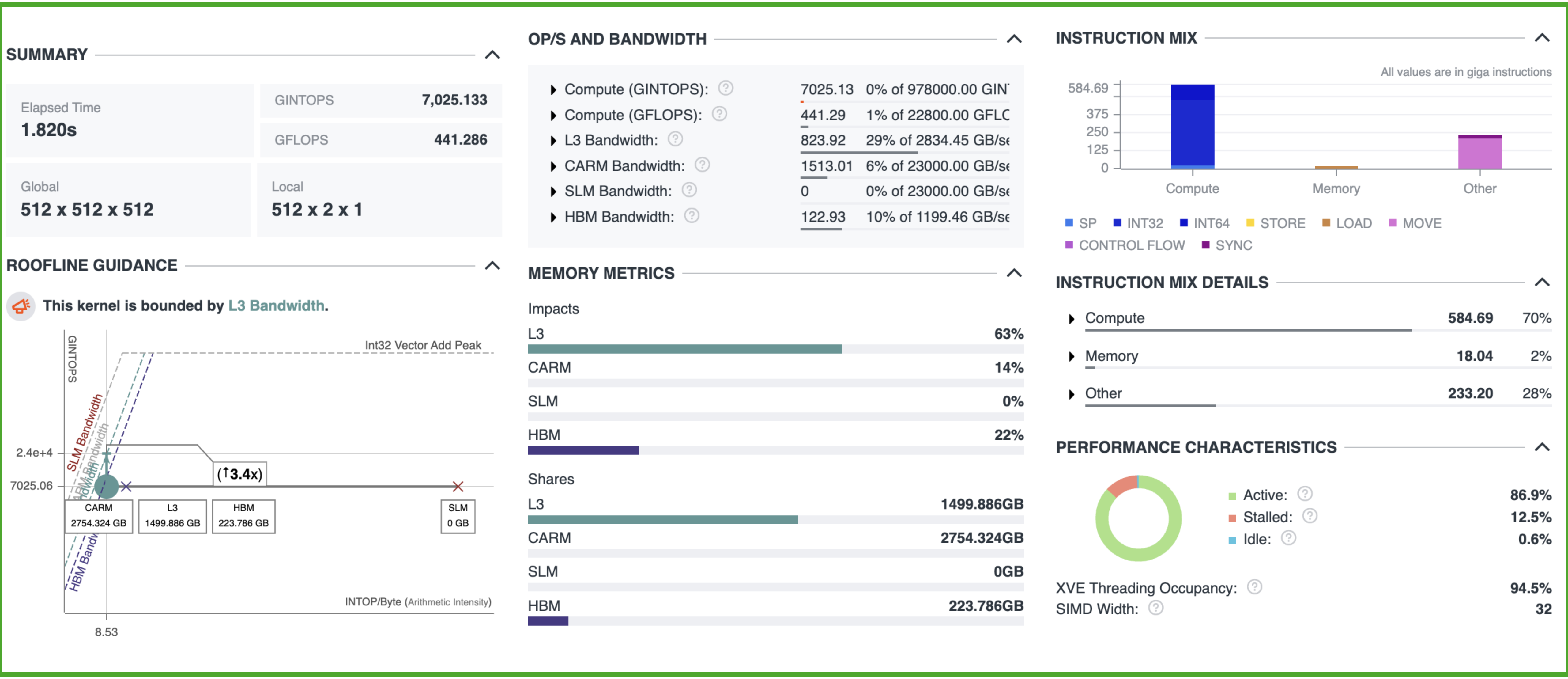
| | |
|---|---|
| L3 | 1499.886GB |
| CARM | 2754.324GB |
| HBM | 223.786GB |

Self Elapsed Time: **5.461** s

Cache locality extent

ATPESC2025

# 2_GPU_basic



## SUMMARY

| | |
|---|---|
| Elapsed Time<br>1.820s | GINTOPS 7,025.133<br>GFLOPS 441.286 |
| Global<br>512 x 512 x 512 | Local<br>512 x 2 x 1 |

## ROOFLINE GUIDANCE

This kernel is bounded by L3 Bandwidth.

## OP/S AND BANDWIDTH

| | | |
|---|---|---|
| ▸ Compute (GINTOPS): ? | 7025.13 | 0% of 978000.00 GIN |
| ▸ Compute (GFLOPS): ? | 441.29 | 1% of 22800.00 GFLC |
| ▸ L3 Bandwidth: ? | 823.92 | 29% of 2834.45 GB/se |
| ▸ CARM Bandwidth: ? | 1513.01 | 6% of 23000.00 GB/se |
| ▸ SLM Bandwidth: ? | 0 | 0% of 23000.00 GB/se |
| ▸ HBM Bandwidth: ? | 122.93 | 10% of 1199.46 GB/se |

## MEMORY METRICS

Impacts

| | |
|---|---|
| L3 | 63% |
| CARM | 14% |
| SLM | 0% |
| HBM | 22% |

Shares

| | |
|---|---|
| L3 | 1499.886GB |
| CARM | 2754.324GB |
| SLM | 0GB |
| HBM | 223.786GB |

## INSTRUCTION MIX

All values are in giga instructions

## INSTRUCTION MIX DETAILS

| | | |
|---|---|---|
| ▸ Compute | 584.69 | 70% |
| ▸ Memory | 18.04 | 2% |
| ▸ Other | 233.20 | 28% |

## PERFORMANCE CHARACTERISTICS

| | |
|---|---|
| Active: ? | 86.9% |
| Stalled: ? | 12.5% |
| Idle: ? | 0.6% |
| XVE Threading Occupancy: ? | 94.5% |
| SIMD Width: ? | 32 |

# 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            : 0.866 secs
throughput      : 15498.6 Mpts/s
flops           : 945.414 GFlops
bytes           : 185.983 GBytes/s
--------------------------------------
```

**Compared 7.25 sec from 2_GPU_basic**

- **8.4X speed-up**

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

or

```
$ cp /flare/ATPESC2025/EXAMPLES/track6-tools/roofline/ADV_results/ADV_03_512 .
```

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

# Check 3_GPU_linear

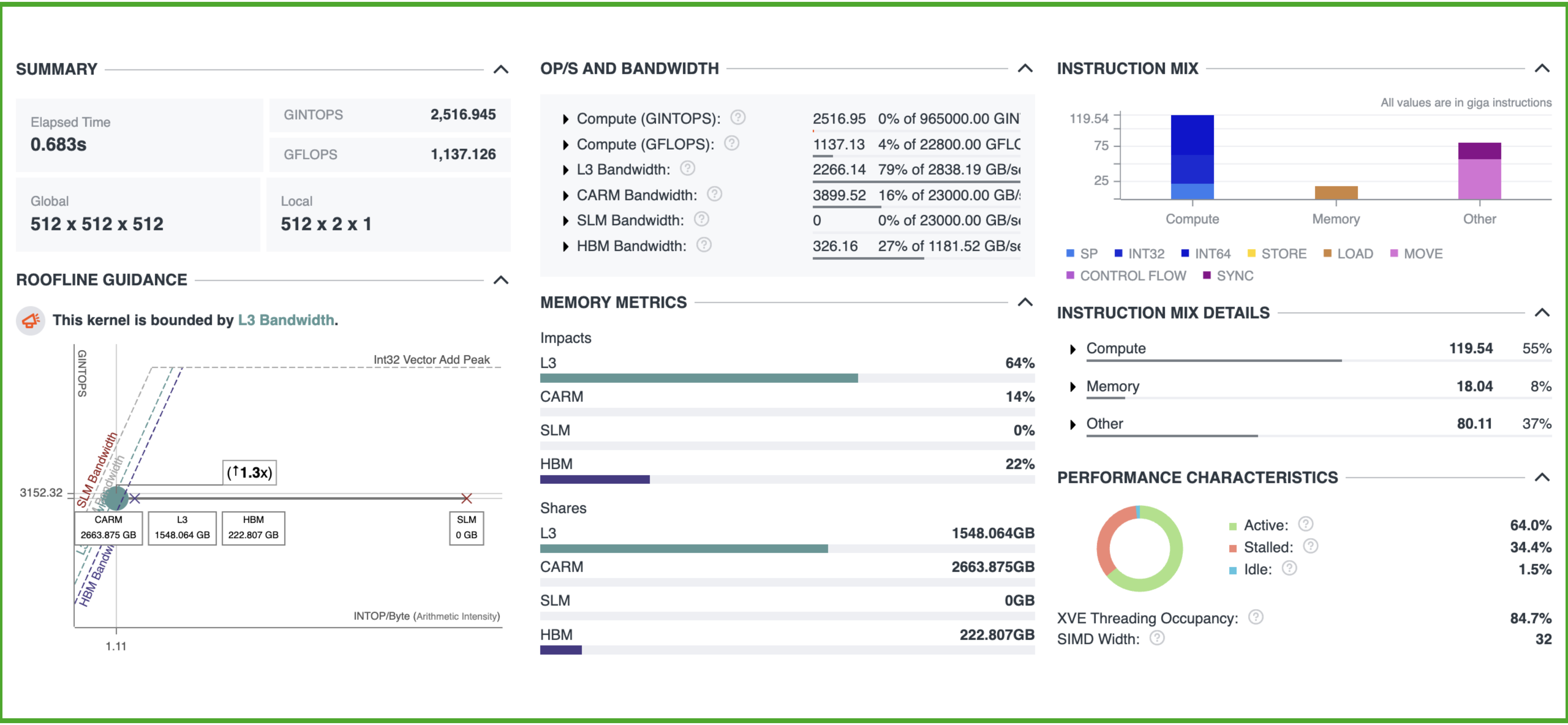
- Please open the html for 3_GPU_linear

  - **Quiz: GFLOPS? Did it change?**

  (hint: look at Summary , and then go back to GPU Roofline Regions)

  - **Quiz: what is a main bottleneck for the kernel?**
  - **Quiz: Any changes in INTOP?**

# 3_GPU_linear



## SUMMARY

| | |
|---|---|
| Elapsed Time<br>0.683s | GINTOPS 2,516.945<br>GFLOPS 1,137.126 |
| Global<br>512 x 512 x 512 | Local<br>512 x 2 x 1 |

## ROOFLINE GUIDANCE

This kernel is bounded by L3 Bandwidth.

## OP/S AND BANDWIDTH

| | | |
|---|---|---|
| Compute (GINTOPS): | 2516.95 | 0% of 965000.00 GIN |
| Compute (GFLOPS): | 1137.13 | 4% of 22800.00 GFLC |
| L3 Bandwidth: | 2266.14 | 79% of 2838.19 GB/s |
| CARM Bandwidth: | 3899.52 | 16% of 23000.00 GB/ |
| SLM Bandwidth: | 0 | 0% of 23000.00 GB/s |
| HBM Bandwidth: | 326.16 | 27% of 1181.52 GB/s |

## MEMORY METRICS

Impacts

| | |
|---|---|
| L3 | 64% |
| CARM | 14% |
| SLM | 0% |
| HBM | 22% |

Shares

| | |
|---|---|
| L3 | 1548.064GB |
| CARM | 2663.875GB |
| SLM | 0GB |
| HBM | 222.807GB |

## INSTRUCTION MIX

All values are in giga instructions

## INSTRUCTION MIX DETAILS

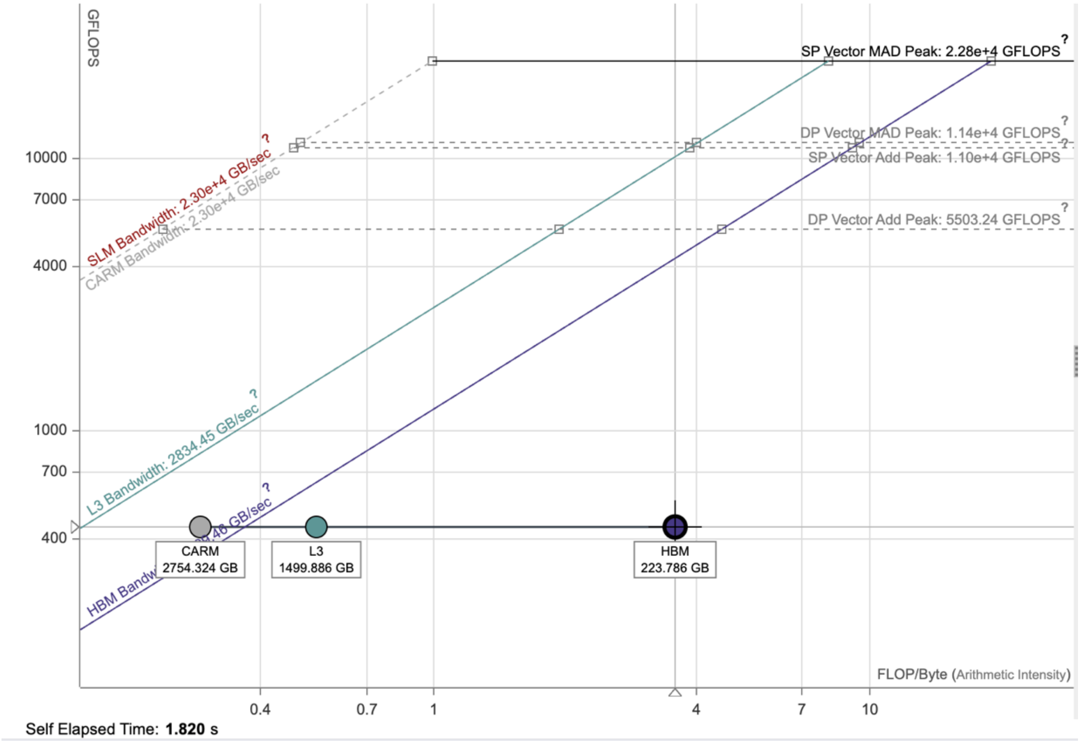
| | | |
|---|---|---|
| Compute | 119.54 | 55% |
| Memory | 18.04 | 8% |
| Other | 80.11 | 37% |

## PERFORMANCE CHARACTERISTICS

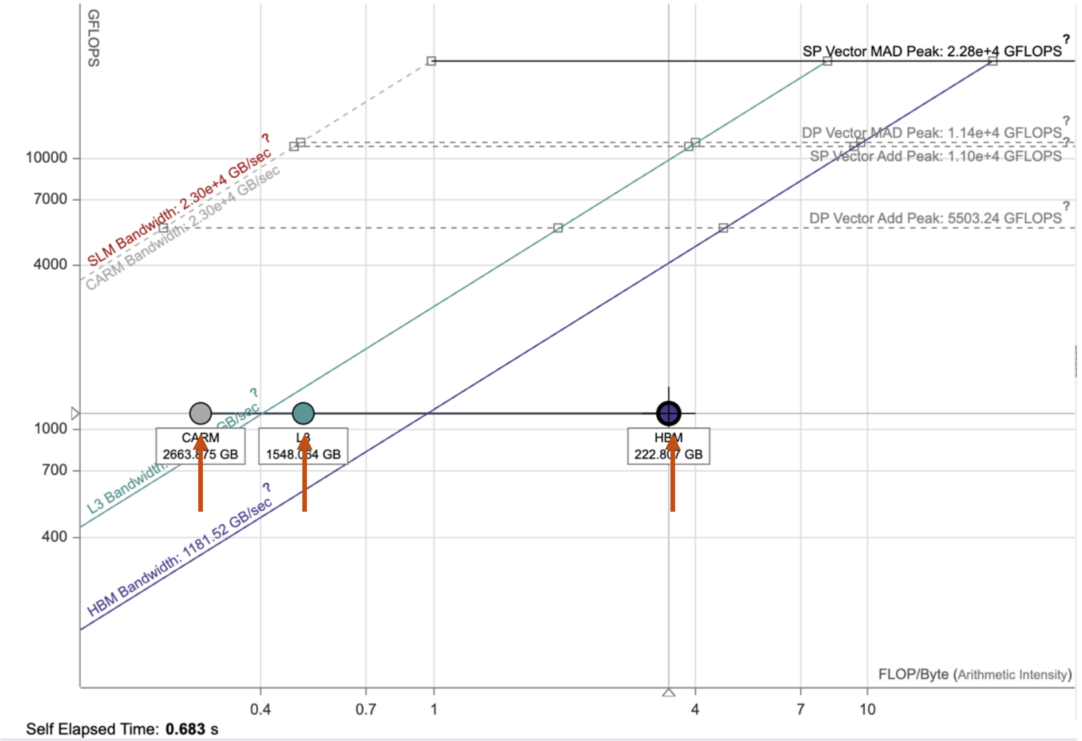
| | |
|---|---|
| Active: | 64.0% |
| Stalled: | 34.4% |
| Idle: | 1.5% |

| | |
|---|---|
| XVE Threading Occupancy: | 84.7% |
| SIMD Width: | 32 |

# Comparison from 2_GPU_basic to 3_GPU_linear in FLOAT roofline chart



Self Elapsed Time: **1.820** s



Self Elapsed Time: **0.683** s

**ANY OPTIMIZATION OPPORTUNITIES FOR 3_GPU_LINEAR?**

Possible to increase AI by re-using data multiple times?

# 4_GPU_private_memory_I: adding private array for coefficients

```
// Send a SYCL kernel(lambda) to the device for parallel execution
// Each kernel runs single row over first dimension
h.parallel_for(kernel_range, [=](id<2> nidx) {
   // Start of device code
   // Add offsets to indices to exclude HALO
   // Start and end index used in loop
   int n2n3 = n2 * n3;
   int i = kHalfLength;
   int j = nidx[0] + kHalfLength;
   int k = nidx[1] + kHalfLength;
   int end_i = n1 - kHalfLength;

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

   // Create arrays to store data used multiple times
   // Local copy of coeff buffer/continous values over 1st dim which
   // are used to calculate stencil front and back arrays are used to
   // ensure the values over 1st dimension are read once, shifted in`
   // these array and re-used multiple times before being discarded
   // This is an optimization technique to enable data-reuse and
   // improve overall FLOPS to BYTES read ratio
   float coeff[kHalfLength + 1];
   float front[kHalfLength + 1];
   float back[kHalfLength];

   // Fill local arrays, front[0] contains current cell value
   for (int x = 0; x <= kHalfLength; x++) {
           coeff[x] = coeff_acc[x];
           front[x] = prev_acc[idx + n2n3 * x];
   }
   for (int x = 1; x <= kHalfLength; x++) {
           back[x-1] = prev_acc[idx - n2n3 * x];
   }
```

**Local array and reuse multiple times**

```
 // Iterate over first dimension excluding HALO
 for (; i < end_i; i++) {
   // Calculate values for each cell
   float value = front[0] * coeff[0];
   #pragma unroll(kHalfLength)
   for (int x = 1; x <= kHalfLength; x++) {
       value += coeff[x] *
              (prev_acc[idx + x]        + prev_acc[idx - x] +
              prev_acc[idx + x * n3] + prev_acc[idx - x * n3] +
              front[x] + back[x - 1] );
   }
   next_acc[idx] = 2.0f * front[0] - next_acc[idx] +
                               value * vel_acc[idx];

   // Increase linear index, jump to the next cell in first dimension
   idx += n2n3;

   // Shift values in front and back arrays
   for (auto x = kHalfLength - 1; x > 0; x--) {
       back[x] = back[x - 1];
   }
   back[0] = front[0];

   for (auto x = 0; x < kHalfLength; x++) {
       front[x] = front[x + 1];
   }
   front[kHalfLength] = prev_acc[idx + kHalfLength * n2n3];
 }
 // End of device code
});
```

**Computing values for each cell**

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

# Run 4_GPU_private_memory_I

```
$ export ZE_AFFINITY_MASK=0.0
$ src/4_GPU_private_memory_I 512 512 512 100
```

```
 Running GPU private memory version with iterations over first dimension
  Running on Intel(R) Data Center GPU Max 1550
  The Device Max Work Group Size is : 1024
  The Device Max EUCount is : 448
--------------------------------------
time            : 0.637 secs
throughput      : 21070.3 Mpts/s
flops           : 1285.29 GFlops
bytes           : 252.843 GBytes/s
--------------------------------------
```

**Compared 0.866 sec from 3_GPU_linear**

- **1.35X speed-up**

**Compared 7.25 sec from 2_GPU_basic**

- **11.4X speed-up**

```
$ advisor -collect roofline --profile-gpu --project-dir ADV_04_512 --  ./src/4_GPU_private_memory_I 512 512 512 100
```

or

```
$ cp /flare/ATPESC2025/EXAMPLES/track6-tools/roofline/ADV_results/ADV_04_512 .
```

```
# Download advisor-report.html from /ADV_04_512/e000/report
```

# Check 4_gpu_private_memory_I

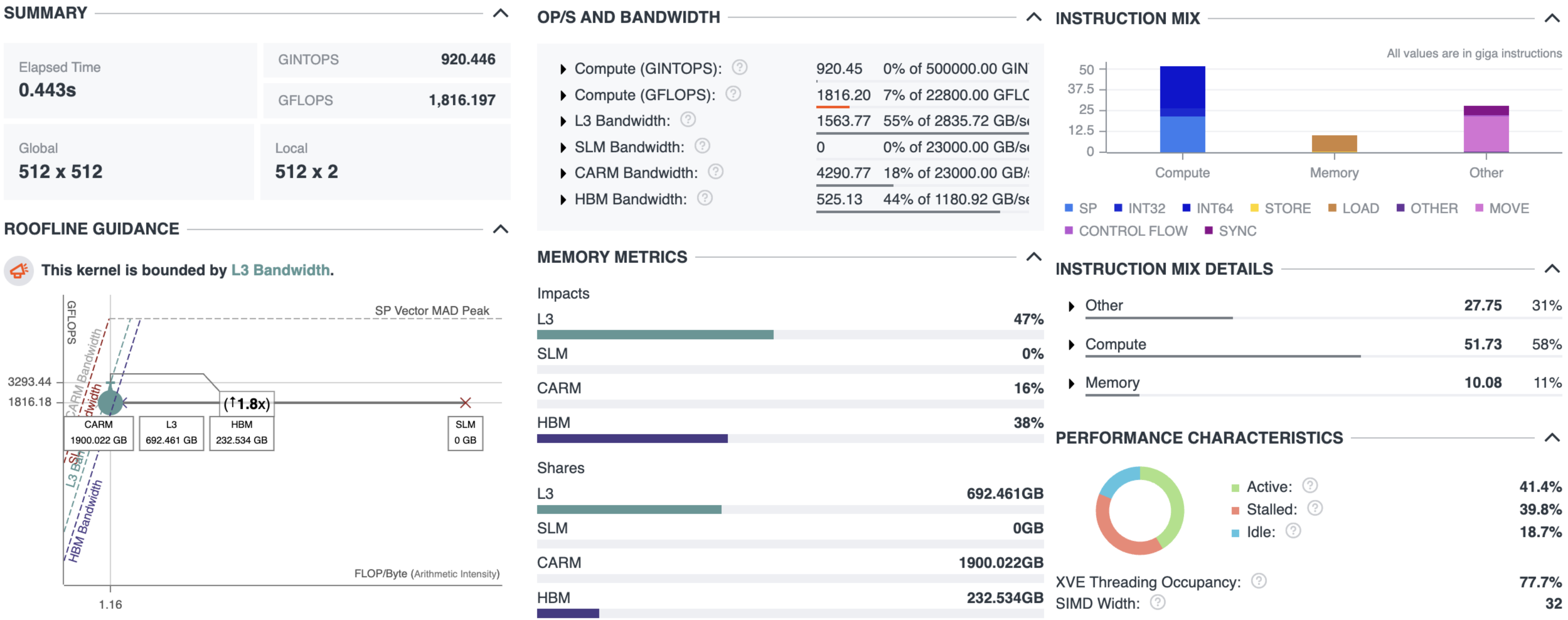
- Please open the html for 4_GPU_Private_memory_I

  - **Quiz: GFLOPS? Did it change?**

    (hint: look at Summary , and then go back to GPU Roofline Regions)

  - **Quiz: what is a main bottleneck for the kernel?**

# 4_GPU_private_memory_I



## SUMMARY

| | |
|---|---|
| Elapsed Time<br>0.443s | GINTOPS 920.446<br>GFLOPS 1,816.197 |
| Global<br>512 x 512 | Local<br>512 x 2 |

## ROOFLINE GUIDANCE

This kernel is bounded by L3 Bandwidth.

## OP/S AND BANDWIDTH

| | | |
|---|---|---|
| ▸ Compute (GINTOPS): ? | 920.45 | 0% of 500000.00 GIN |
| ▸ Compute (GFLOPS): ? | 1816.20 | 7% of 22800.00 GFLC |
| ▸ L3 Bandwidth: ? | 1563.77 | 55% of 2835.72 GB/s |
| ▸ SLM Bandwidth: ? | 0 | 0% of 23000.00 GB/s |
| ▸ CARM Bandwidth: ? | 4290.77 | 18% of 23000.00 GB/ |
| ▸ HBM Bandwidth: ? | 525.13 | 44% of 1180.92 GB/s |

## MEMORY METRICS

Impacts

| | |
|---|---|
| L3 | 47% |
| SLM | 0% |
| CARM | 16% |
| HBM | 38% |

Shares

| | |
|---|---|
| L3 | 692.461GB |
| SLM | 0GB |
| CARM | 1900.022GB |
| HBM | 232.534GB |

## INSTRUCTION MIX

All values are in giga instructions

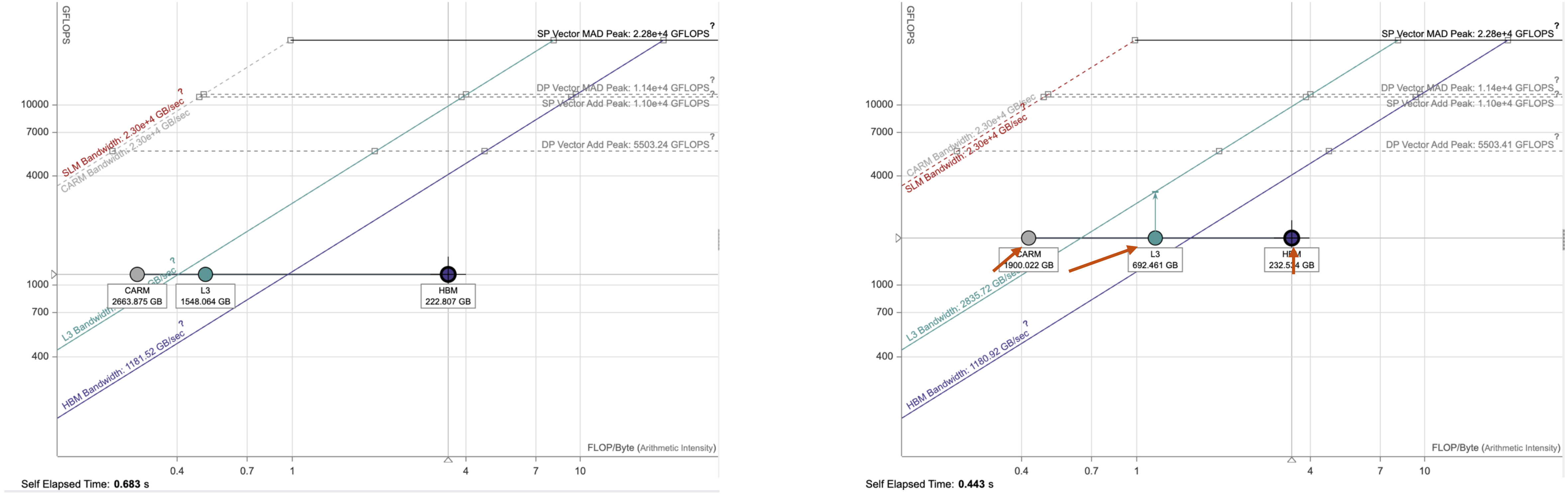
## INSTRUCTION MIX DETAILS

| | | |
|---|---|---|
| ▸ Other | 27.75 | 31% |
| ▸ Compute | 51.73 | 58% |
| ▸ Memory | 10.08 | 11% |

## PERFORMANCE CHARACTERISTICS

| | |
|---|---|
| Active: ? | 41.4% |
| Stalled: ? | 39.8% |
| Idle: ? | 18.7% |
| XVE Threading Occupancy: ? | 77.7% |
| SIMD Width: ? | 32 |

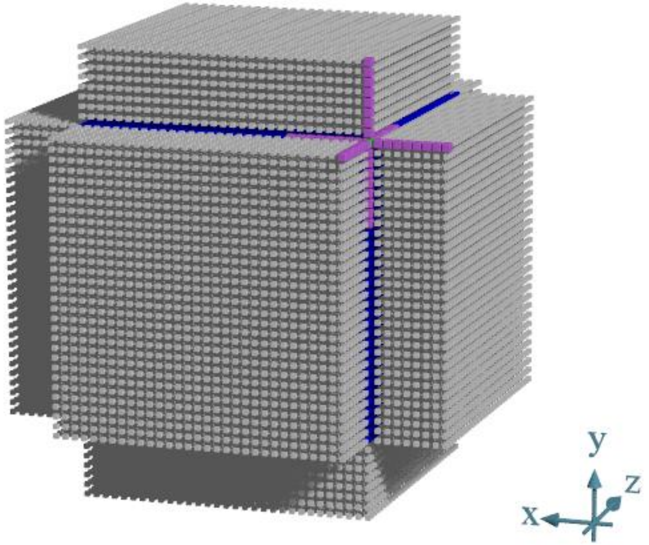
# Comparison from 3_GPU_linear to 4_GPU_private_memory_I



**AI for L3 increases by re-using data; as a result, GFLOPS improves from 1,137 to 1,816: 1.6X speed-up**

# Recap

# Performance results



Grid size: 512x512x512
Number of iteration: 100
Employed Compute Platform

- CPU: Intel Xeon CPU Max
- GPU: Intel Data Center GPU Max

| Selected versions | Target platform | GFLOPs | Kernel time (s) | Speed up from 1_CPU_only | Speed-up from 2_GPU_basic |
|---|---|---|---|---|---|
| 1_CPU_only* | 1 core from CPU | 7.24 | 113.09 | 1 x | - |
| 2_GPU_basic | 1 stack from GPU | 112.9 | 7.25 | 15.7 x | 1 x |
| 3_GPU_linear | | 945.4 | 0.866 | 131 x | 8.4 x |
| 4_GPU_private_memory_I | | 1285.3 | 0.637 | 179 x | 11.4 x |

* 1_CPU_only ran 10 iterations instead of 100 iterations since its performance is too low; therefore, the kernel time is projected for 100 iterations.

# Thank you!

ARGONNE
ATPESC2025
EXTREME-SCALE COMPUTING

# 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.