AMD

Hierarchical Roofline on AMD Instinct™ MI200 GPUs

Noah Wolfe, Xiaomin Lu

October 11, 2022

Agenda

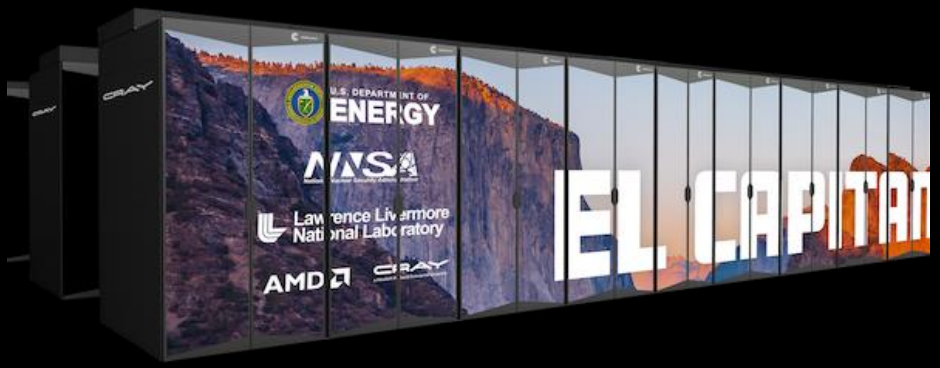
- Introduction
- Roofline Fundamentals
- Empirical Hierarchical Roofline on MI200
  - Roofline Overview
  - Roofline Arithmetic
  - Empirical Roofline Benchmarking
- Roofline Analysis Workflow
  - Tooling
  - Features
  - Bottleneck Analysis Recipe
- Examples
  - Add/Mul/FMA
  - N-Body
- HPC Application Results

AMD Fueling the Era of Exascale

OAK RIDGE
FRONTIER

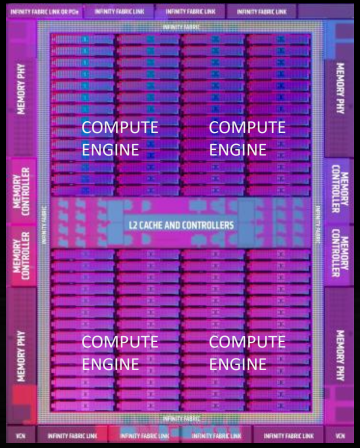
AMD
EPYC | INSTINCT

LAWRENCE LIVERMORE
EL CAPITAN



AMD INSTINCT™ MI250X ACCELERATOR

TSMC 6NM TECHNOLOGY	UP TO 110 CU PER GRAPHICS COMPUTE DIE
4 MATRIX CORES PER COMPUTE UNIT	MATRIX CORES ENHANCED FOR HPC
8 INFINITY FABRIC LINKS PER DIE	SPECIAL FP32 OPS FOR DOUBLE THROUGHPUT

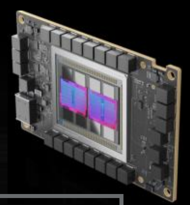


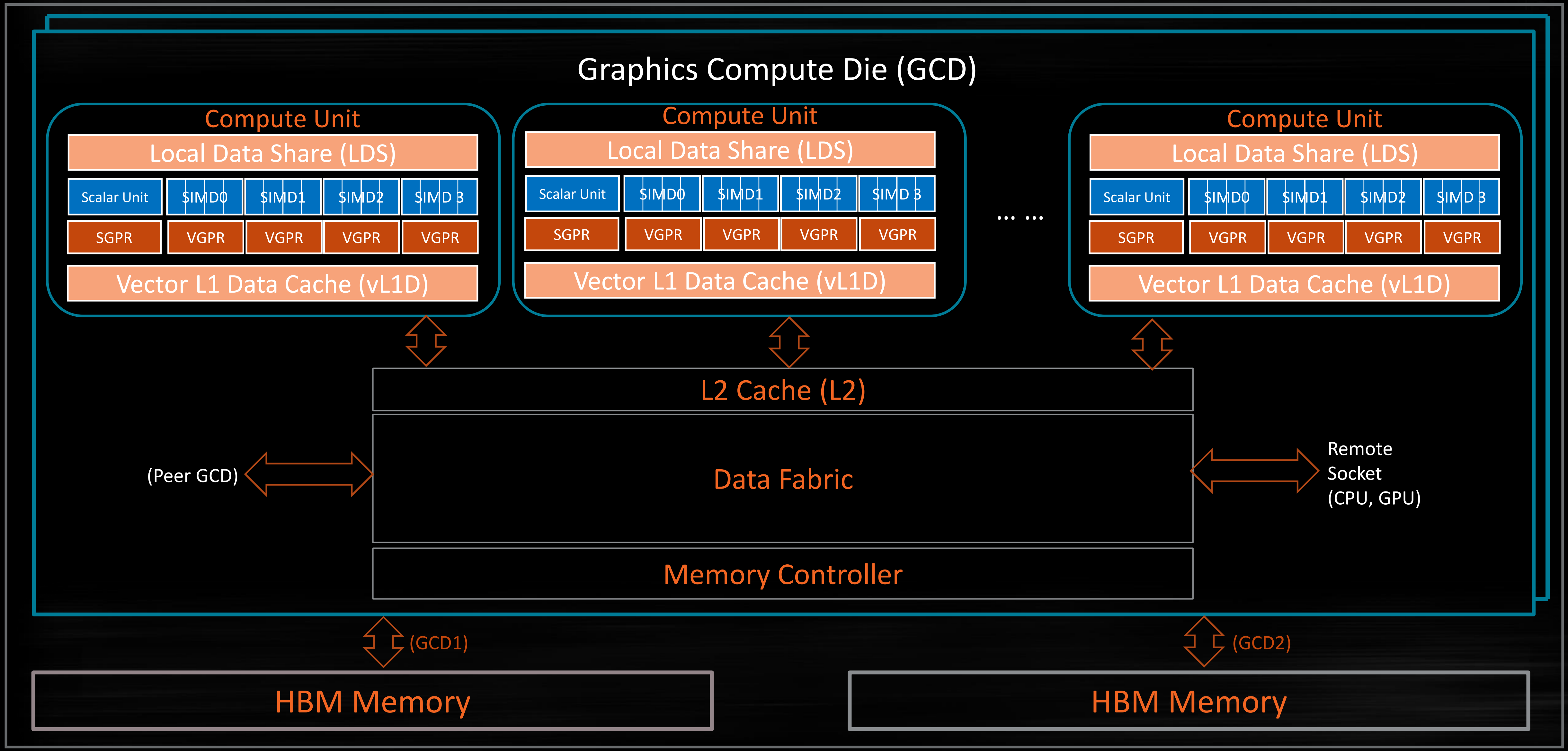
FRONTIER NODE AT A GLANCE

- Optimized 3rd Gen AMD EPYC™ processor
- Four Instinct MI250X accelerators
- Coherent connectivity
  - Via Infinity Fabric™ interconnect
  - Tightly integrated
  - Unified memory space

AMD

Overview - AMD Instinct™ MI200 Architecture





Roofline – All Workloads

Orange: Synthetic Workload Yellow: Proxy app Green: Full app

*MI250x. RESULTS MAY VARY. SEE ENDNOTE: MI200-62

AMD

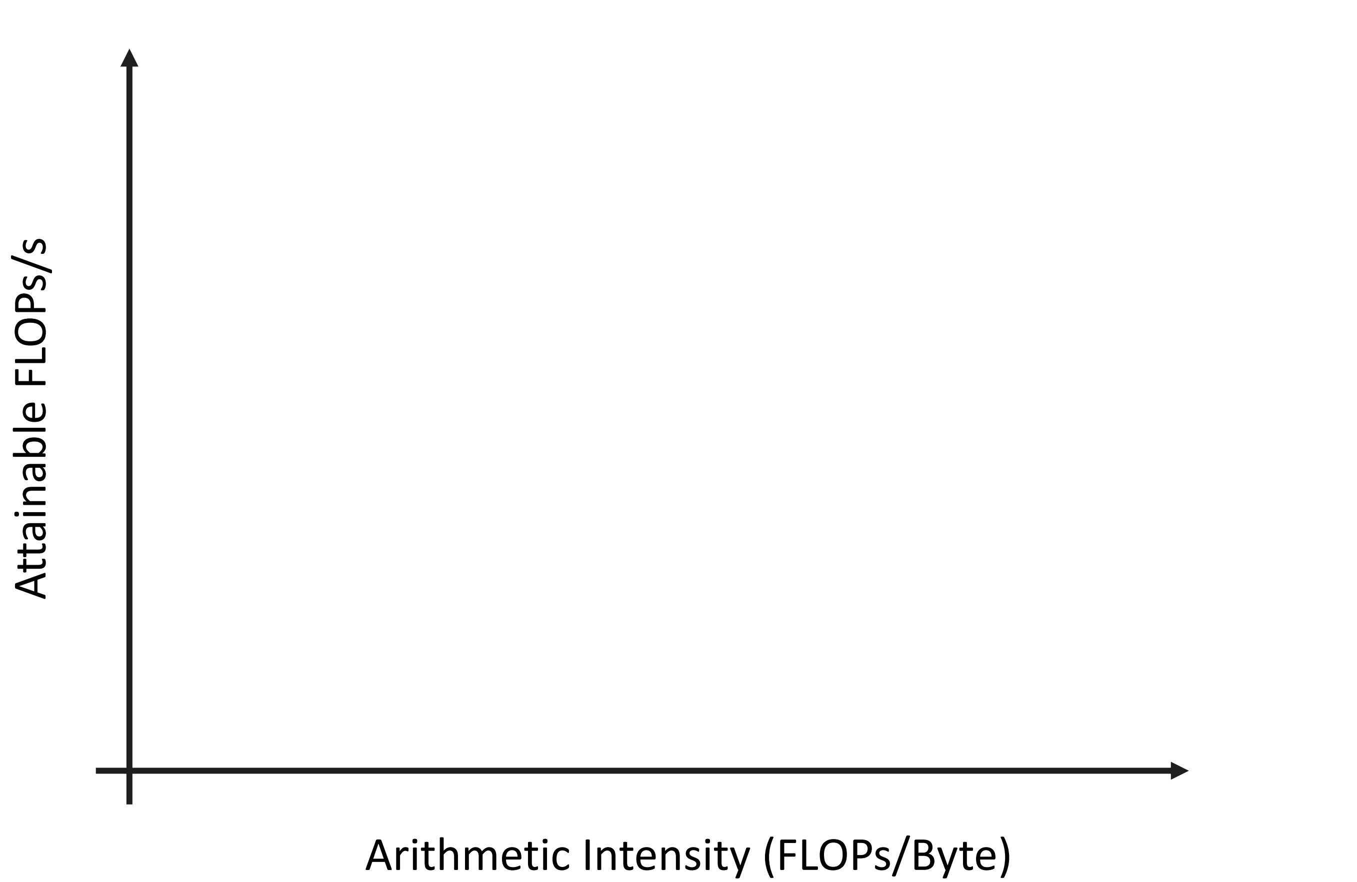
Background – What is Roofline

Background – What is Roofline

- Attainable FLOPs/s
  - FLOPs/s rate as measured empirically on a given device
  - FLOP = floating point operation
  - FLOP counts for common operations
    - Add: 1 FLOP
    - Mul: 1 FLOP
    - FMA: 2 FLOP
  - FLOPs/s = Number of floating-point operations performed per second

Background – What is Roofline

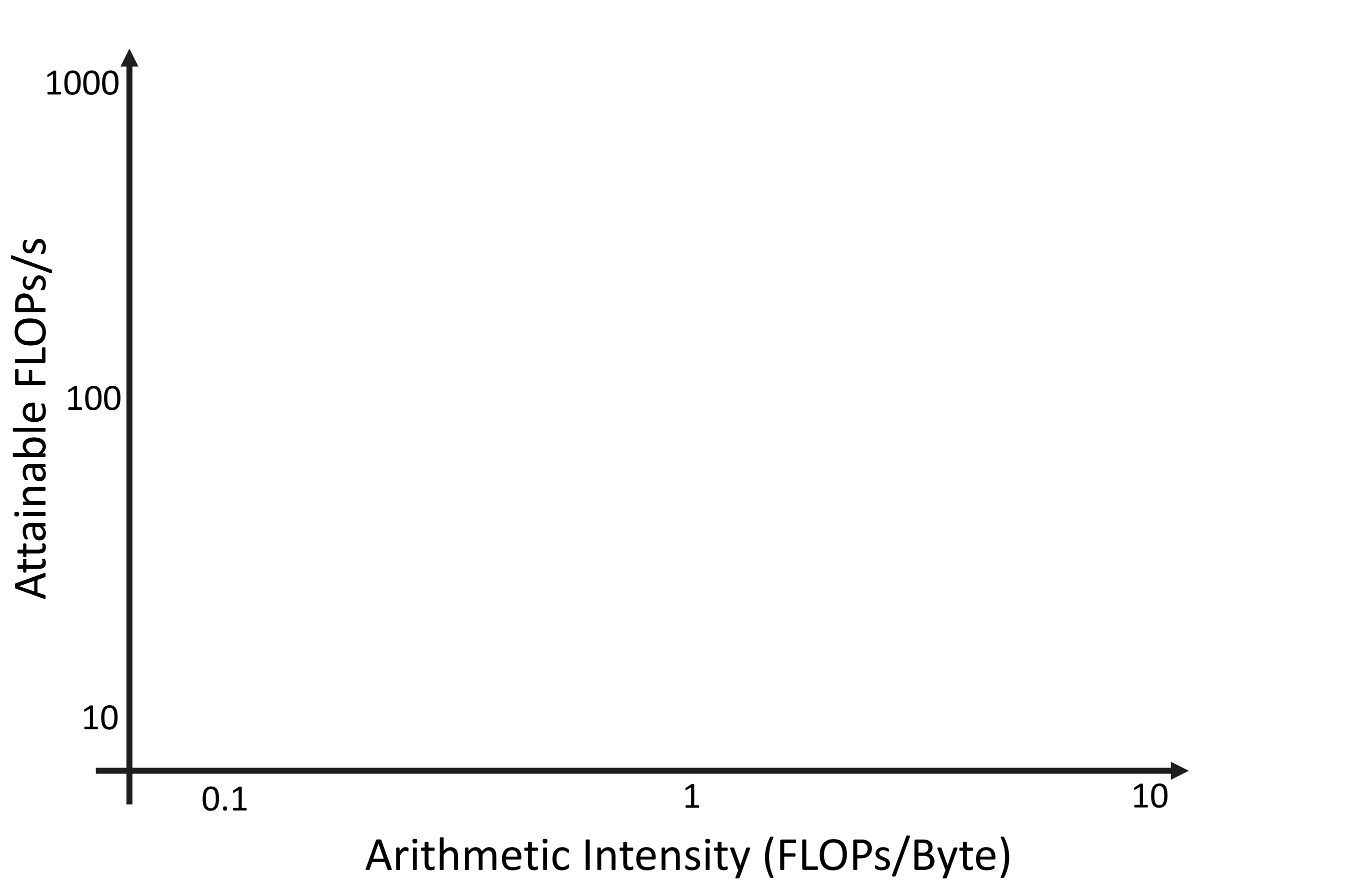
- Arithmetic Intensity (AI)
  - characteristic of the workload indicating how much compute (FLOPs) is performed per unit of data movement (Byte)
  - Ex: x[i] = y[i] + c
    - FLOPs = 1
    - Bytes = 1xRD + 1xWR = 4 + 4 = 8
    - AI = 1 / 8



AMD

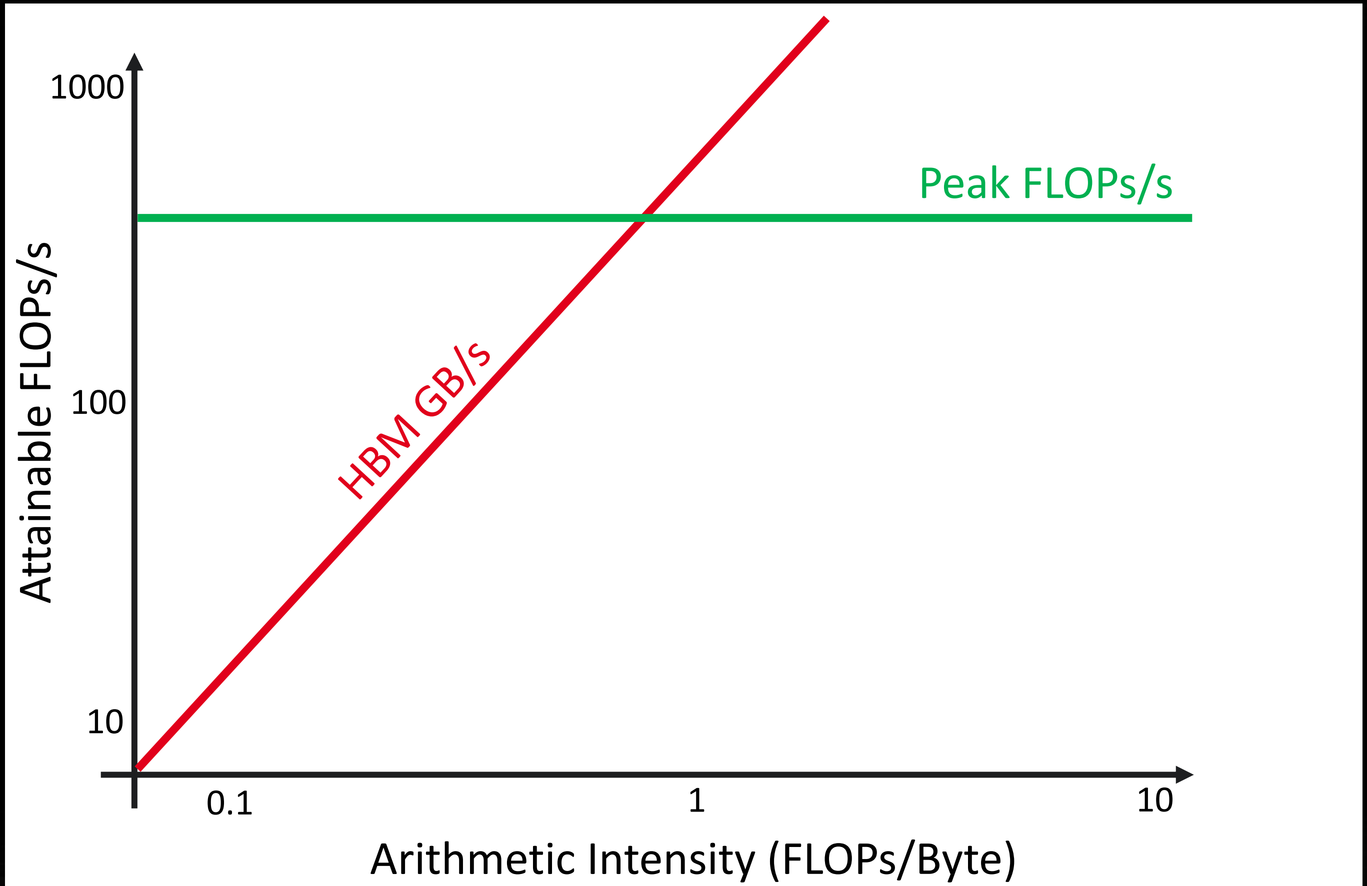
Background – What is Roofline

- Log-Log plot
  - makes it easy to doodle, extrapolate performance along Moore's Law, etc...



Background – What is Roofline

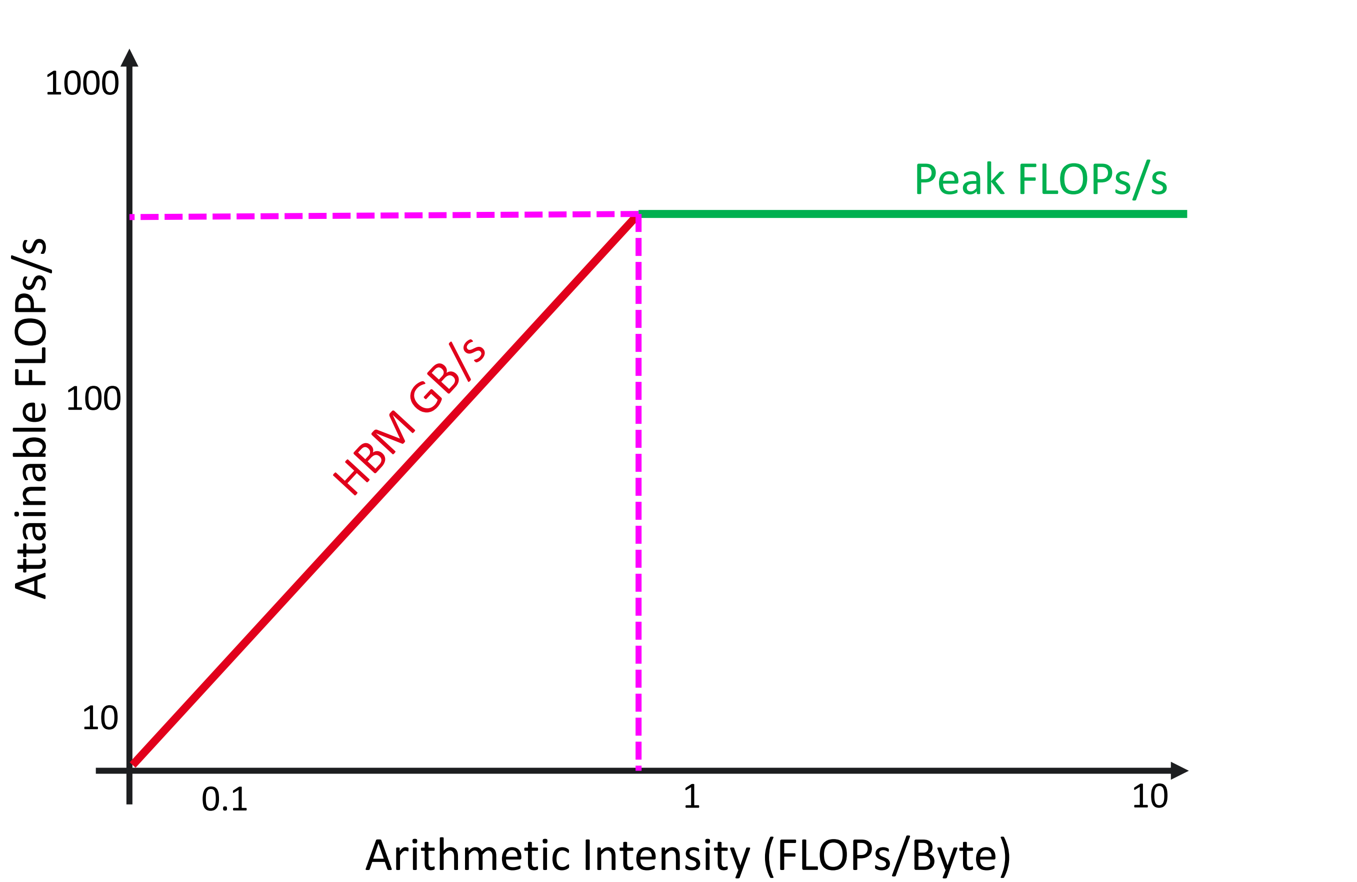
- Roofline Limiters
  - Compute
    - Peak FLOPs/s
  - Memory BW
    - AI * Peak GB/s
- Note:
  - These are empirically measured values
  - Different SKUs will have unique plots
  - Individual devices within a SKU will have slightly different plots based on thermal solution, system power, etc.
  - MIBench uses suite of simple kernels to empirically derive these values
  - These are NOT theoretical values indicating peak performance under "unicorn" conditions



AMD

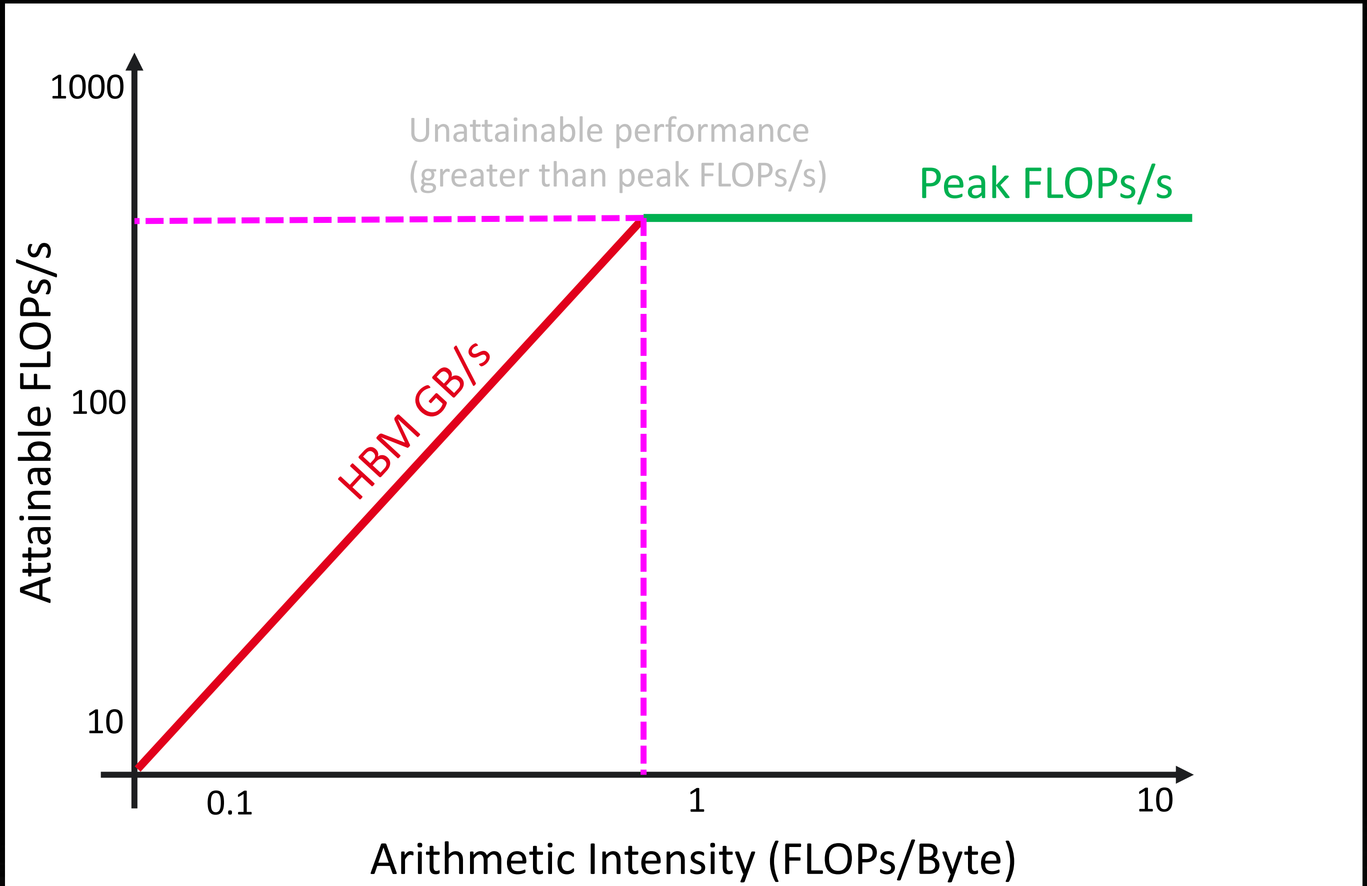
Background – What is Roofline

- Attainable FLOPs/s =
  - $min \begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
  - Typical machine balance: 5-10 FLOPs/B
    - **40-80** FLOPs per double to exploit compute capability
  - MI250x machine balance: ~16 FLOPs/B
    - **128** FLOPs per double to exploit compute capability



Background – What is Roofline

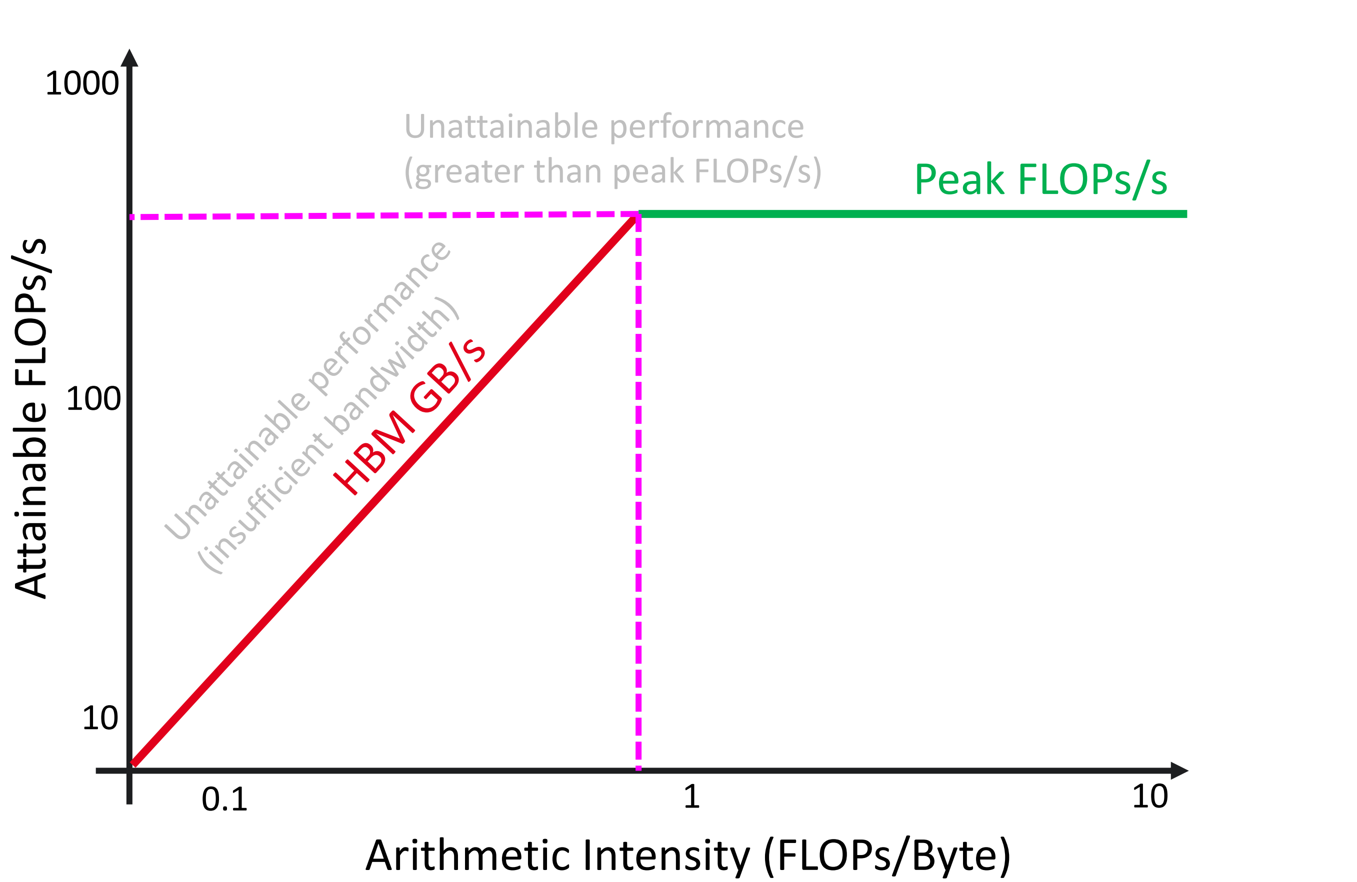
- Attainable FLOPs/s =
  - $min\begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
- Five Performance Regions:
  - Unattainable Compute



AMD

Background – What is Roofline

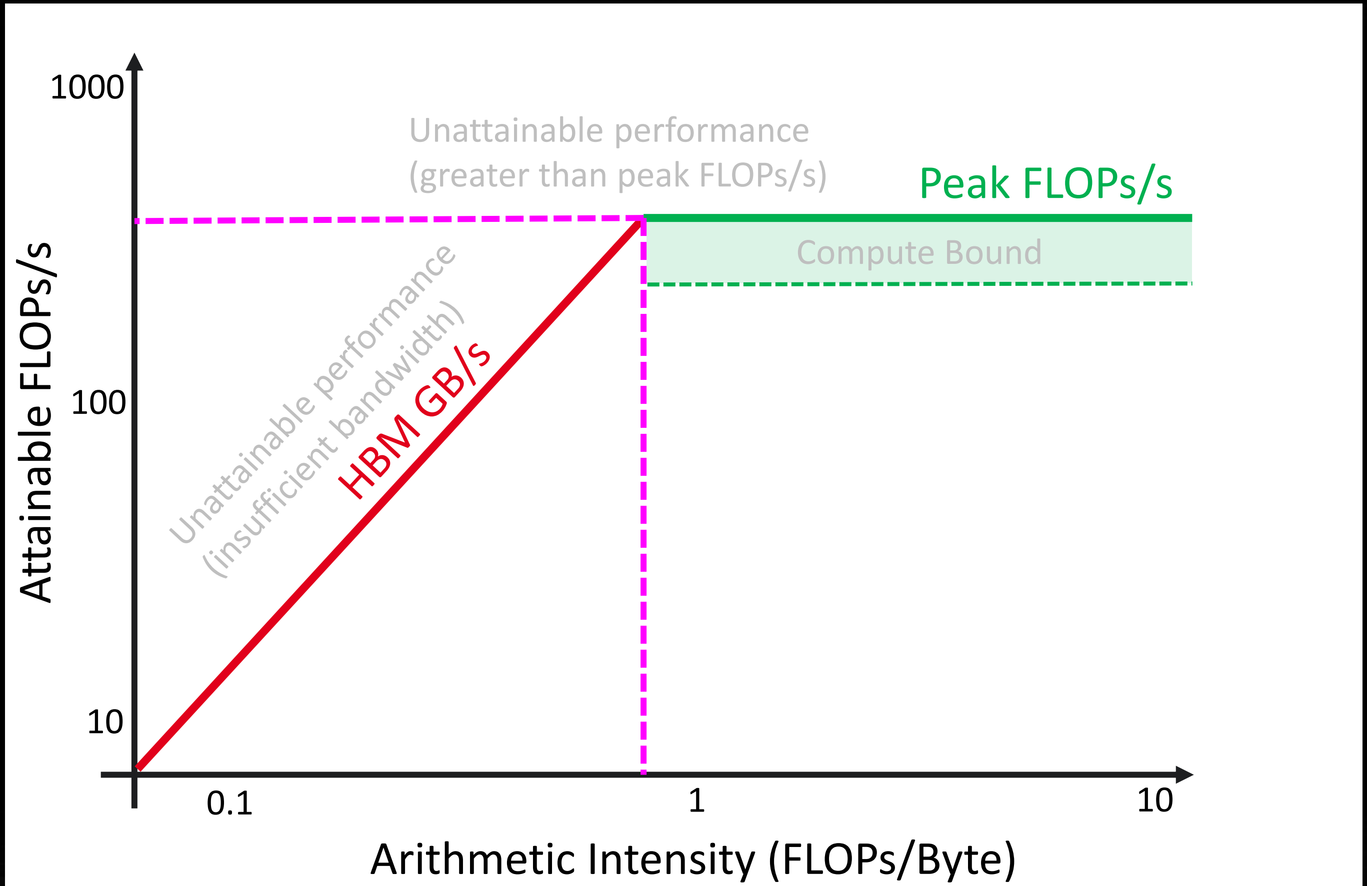
- Attainable FLOPs/s =
  - $min \begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
- Five Performance Regions:
  - Unattainable Compute
  - Unattainable Bandwidth



AMD

Background – What is Roofline

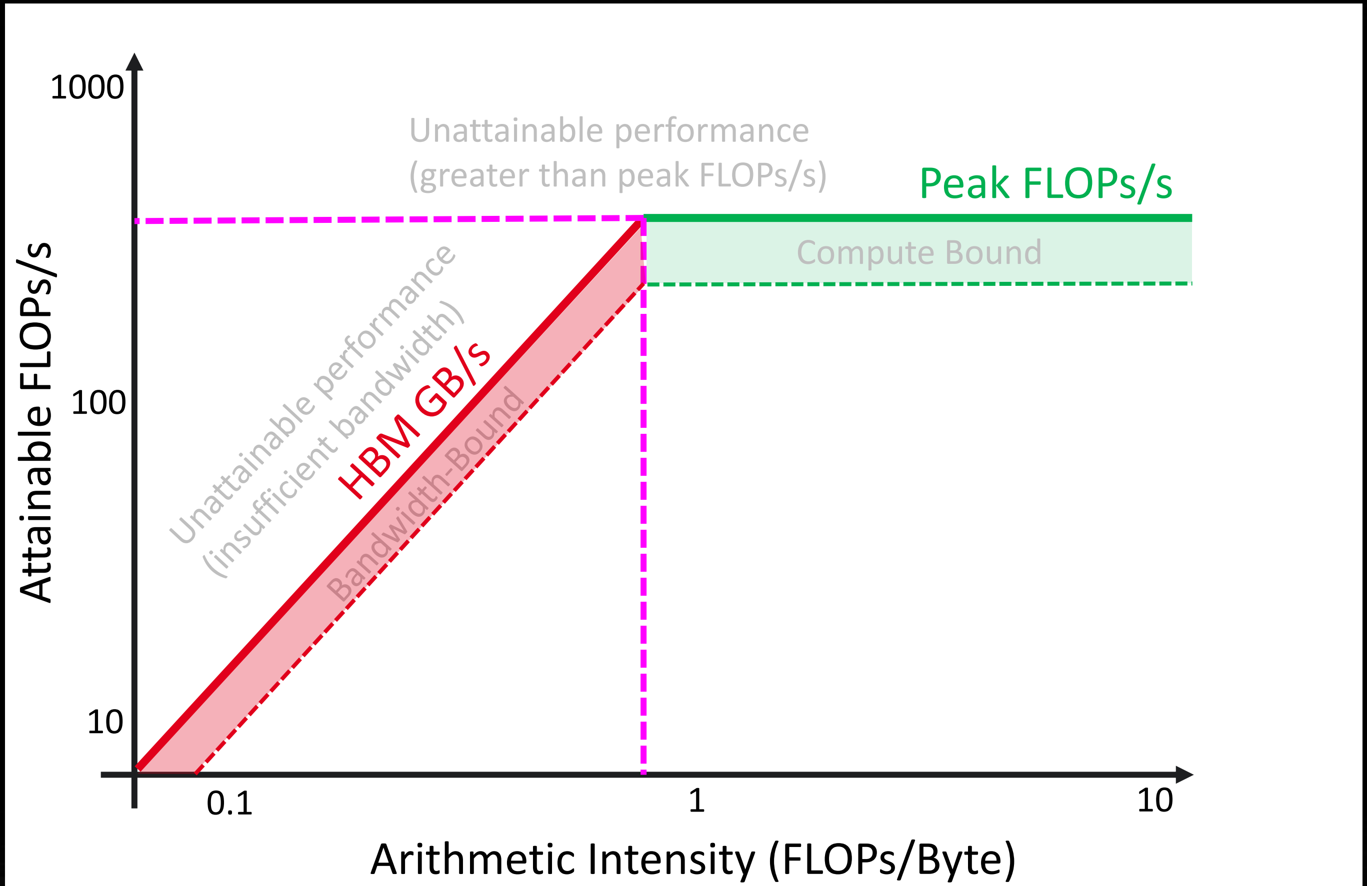
- Attainable FLOPs/s =
  - $min\begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
- Five Performance Regions:
  - Unattainable Compute
  - Unattainable Bandwidth
  - Compute Bound



AMD

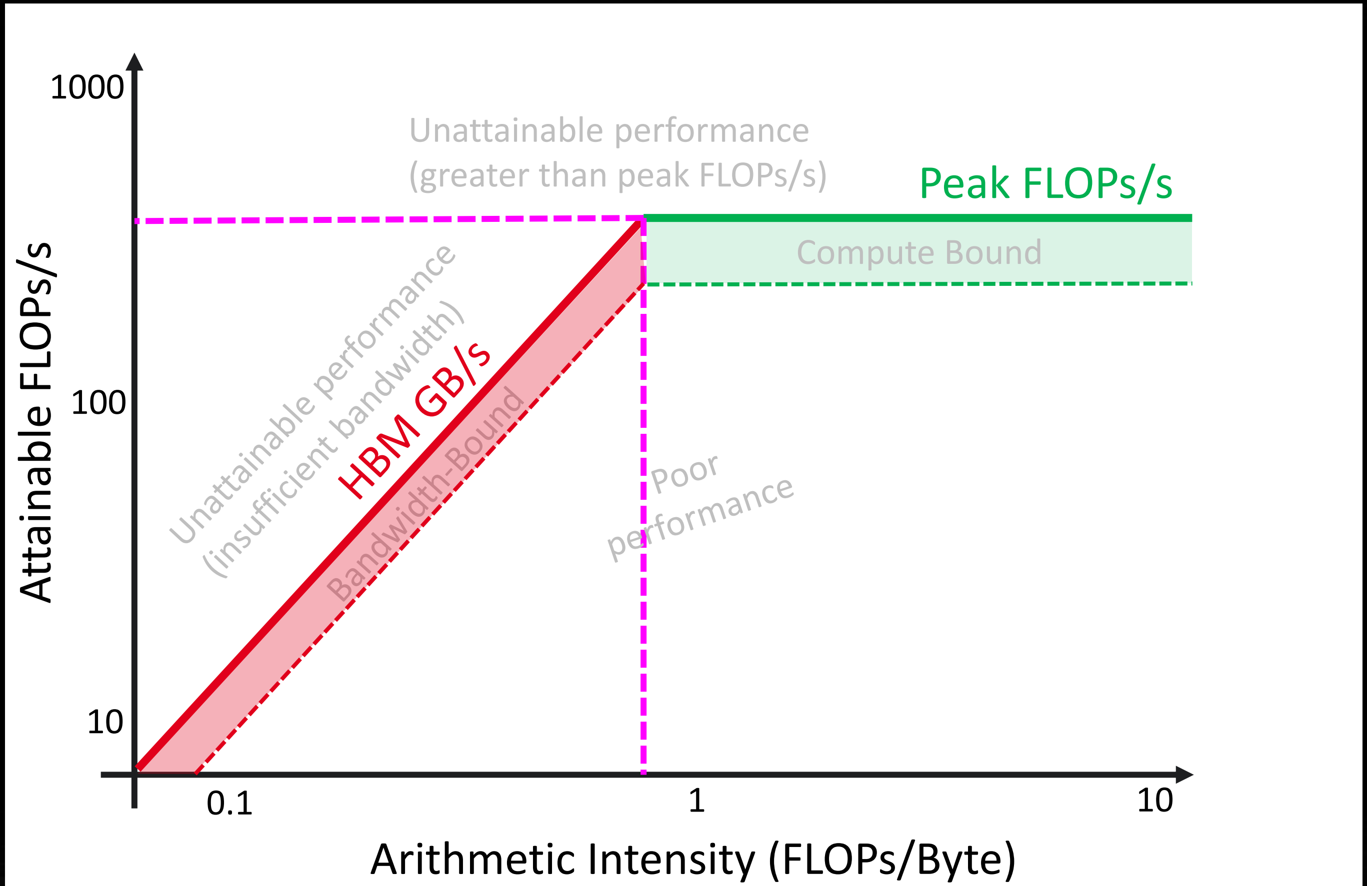
Background – What is Roofline

- Attainable FLOPs/s =
  - $min\begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
- Five Performance Regions:
  - Unattainable Compute
  - Unattainable Bandwidth
  - Compute Bound
  - Bandwidth Bound



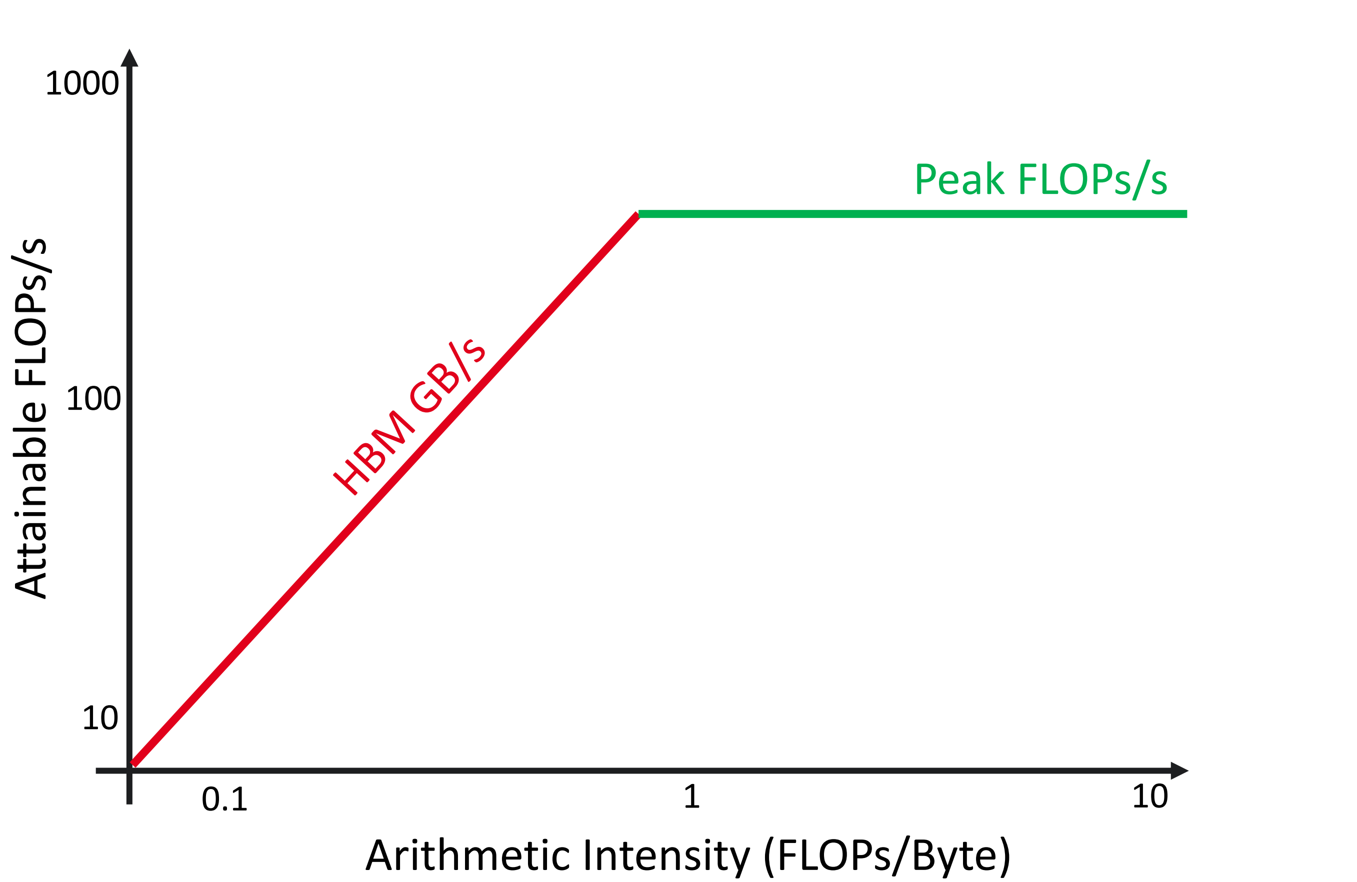
Background – What is Roofline

- Attainable FLOPs/s =
  - $min\begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
- Five Performance Regions:
  - Unattainable Compute
  - Unattainable Bandwidth
  - Compute Bound
  - Bandwidth Bound
  - Poor Performance



Background – What is Roofline

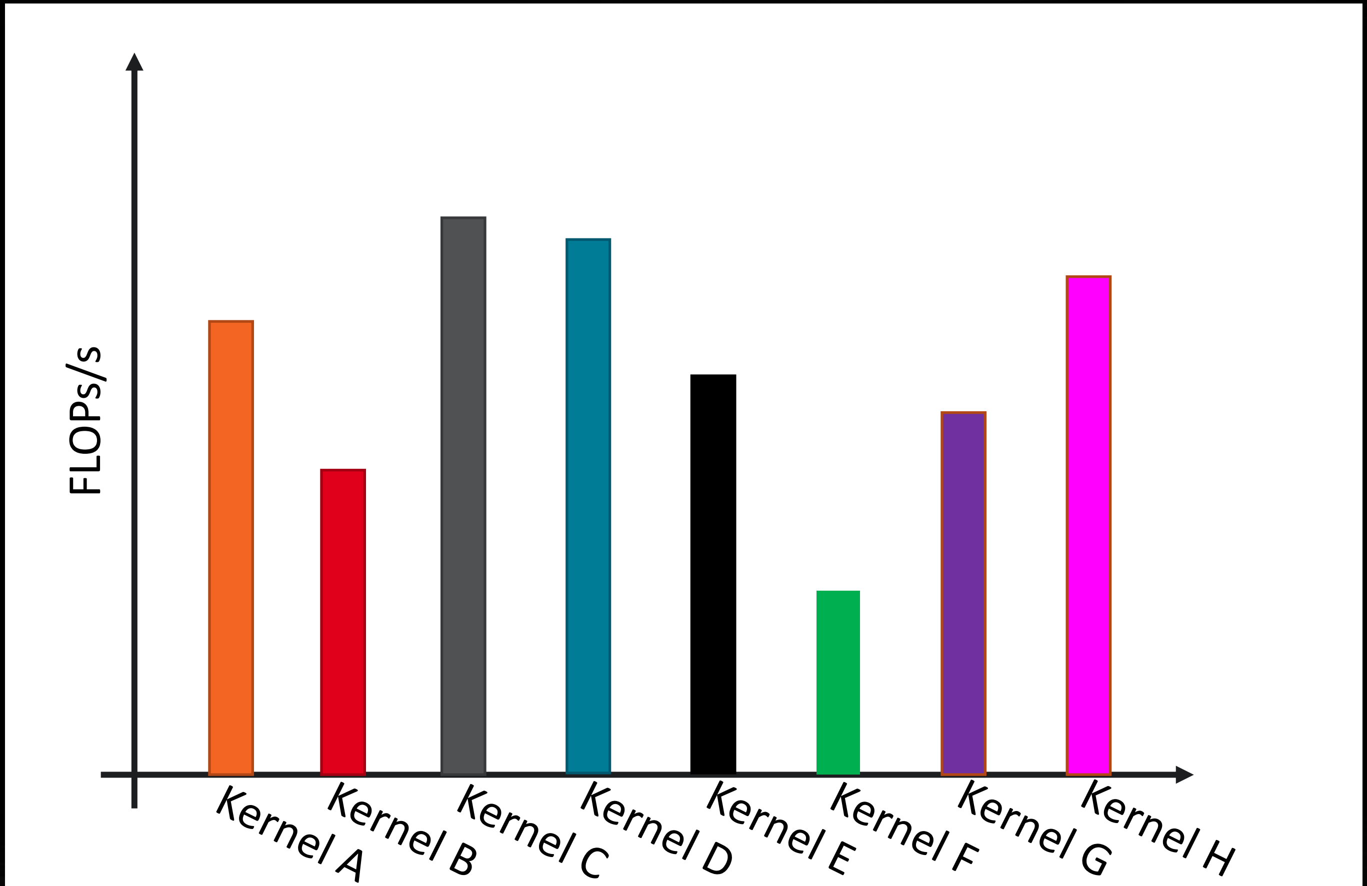
- Attainable FLOPs/s =
  - $min\begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Final result is a single roofline plot presenting the peak attainable performance (in terms of FLOPs/s) on a given device based on the arithmetic intensity of any potential workload
- We have an application independent way of measuring and comparing performance on any platform



AMD

Background – What is "Good" Performance

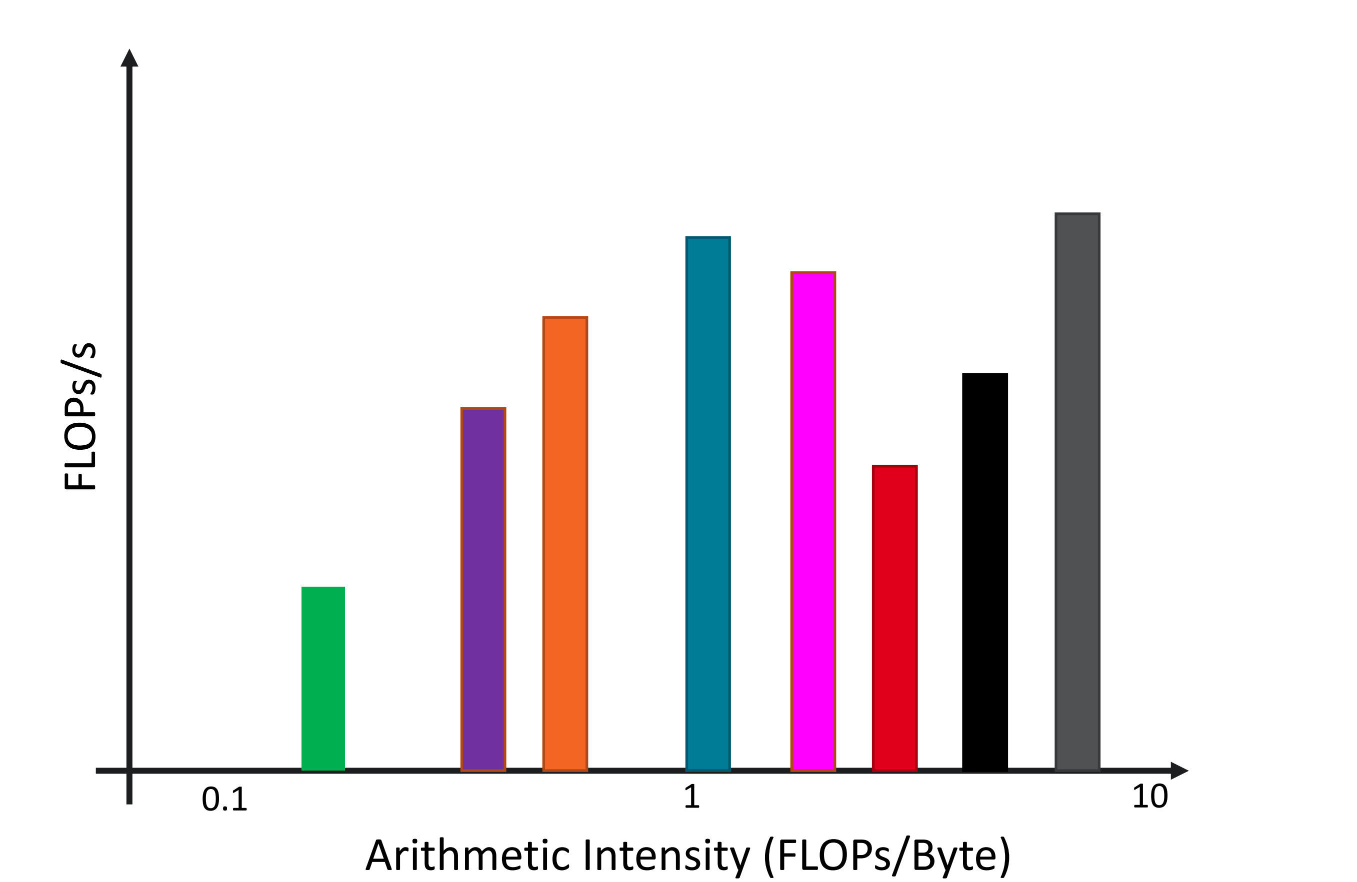
- Example:
  - We run a number of kernels and measure FLOPs/s



AMD

Background – What is “Good” Performance

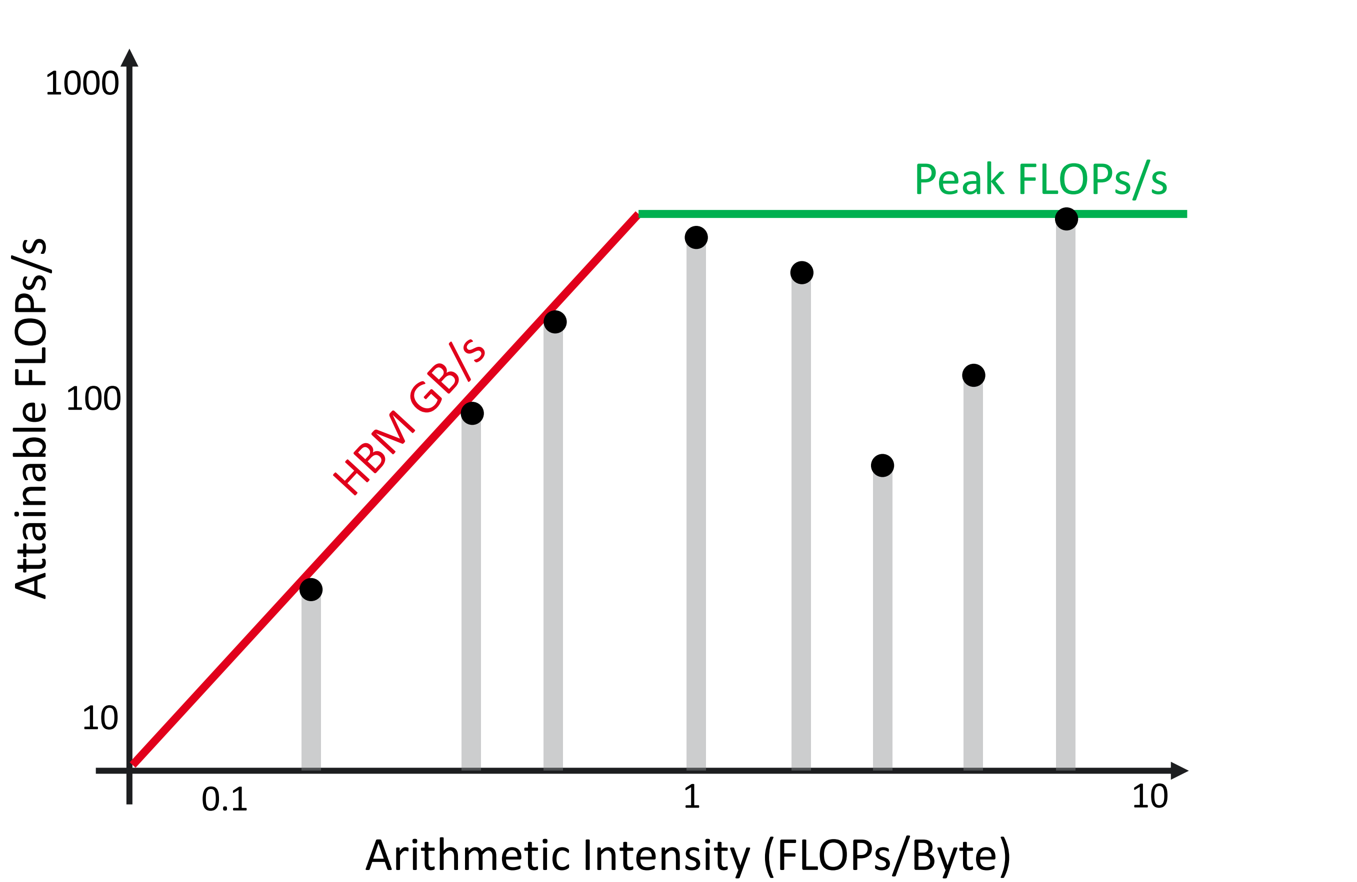
- Example:
  - We run a number of kernels and measure FLOPs/s
  - Sort kernels by arithmetic intensity



AMD

Background – What is "Good" Performance

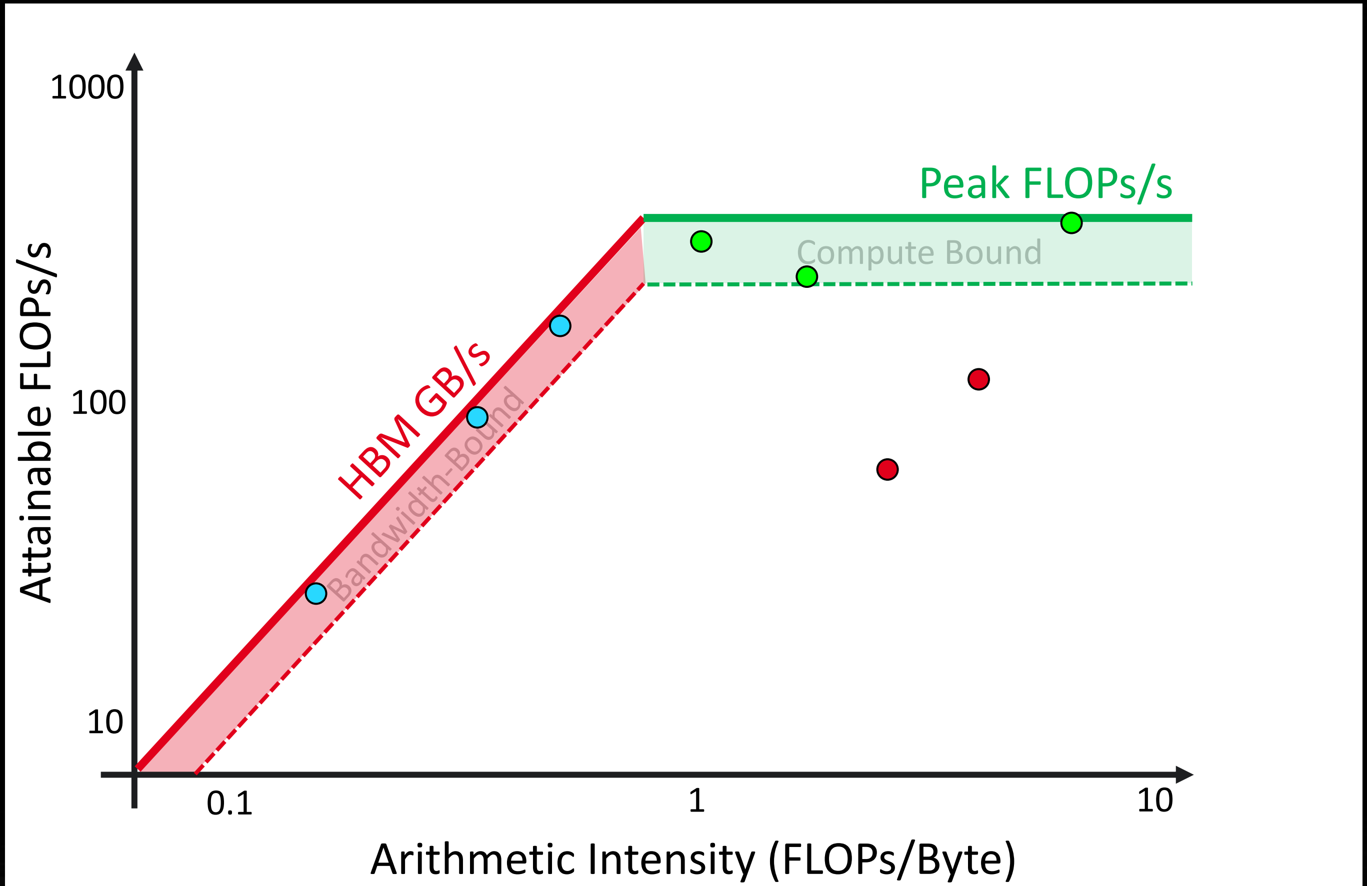
- Example:
  - We run a number of kernels and measure FLOPs/s
  - Sort kernels by arithmetic intensity
  - Compare performance relative to hardware capabilities



AMD

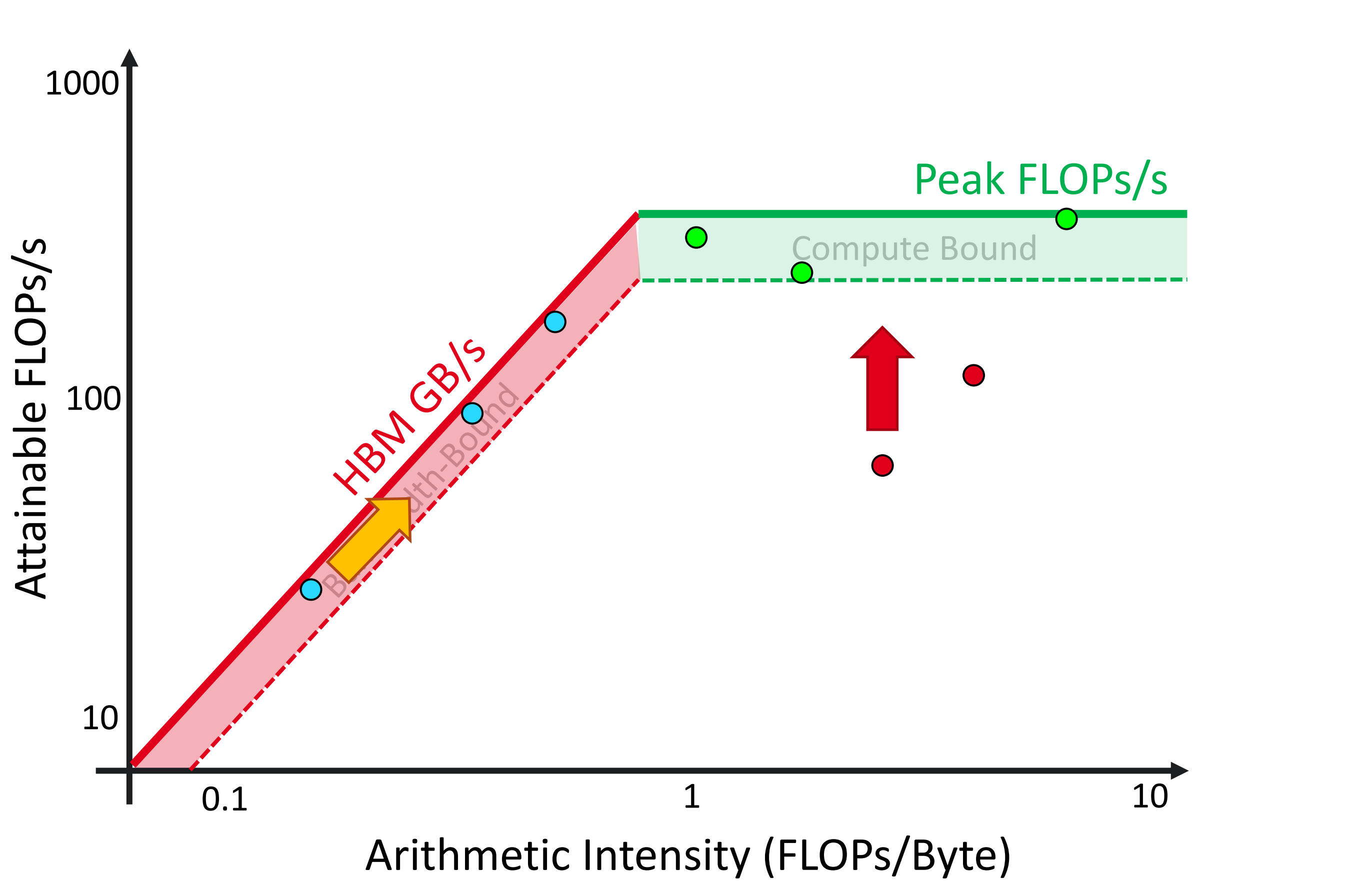
Background – What is “Good” Performance

- Example:
  - We run a number of kernels and measure FLOPs/s
  - Sort kernels by arithmetic intensity
  - Compare performance relative to hardware capabilities
  - Kernels near the roofline are making good use of computational resources
    - Kernels can have low performance (FLOPS/s), but make good use of BW



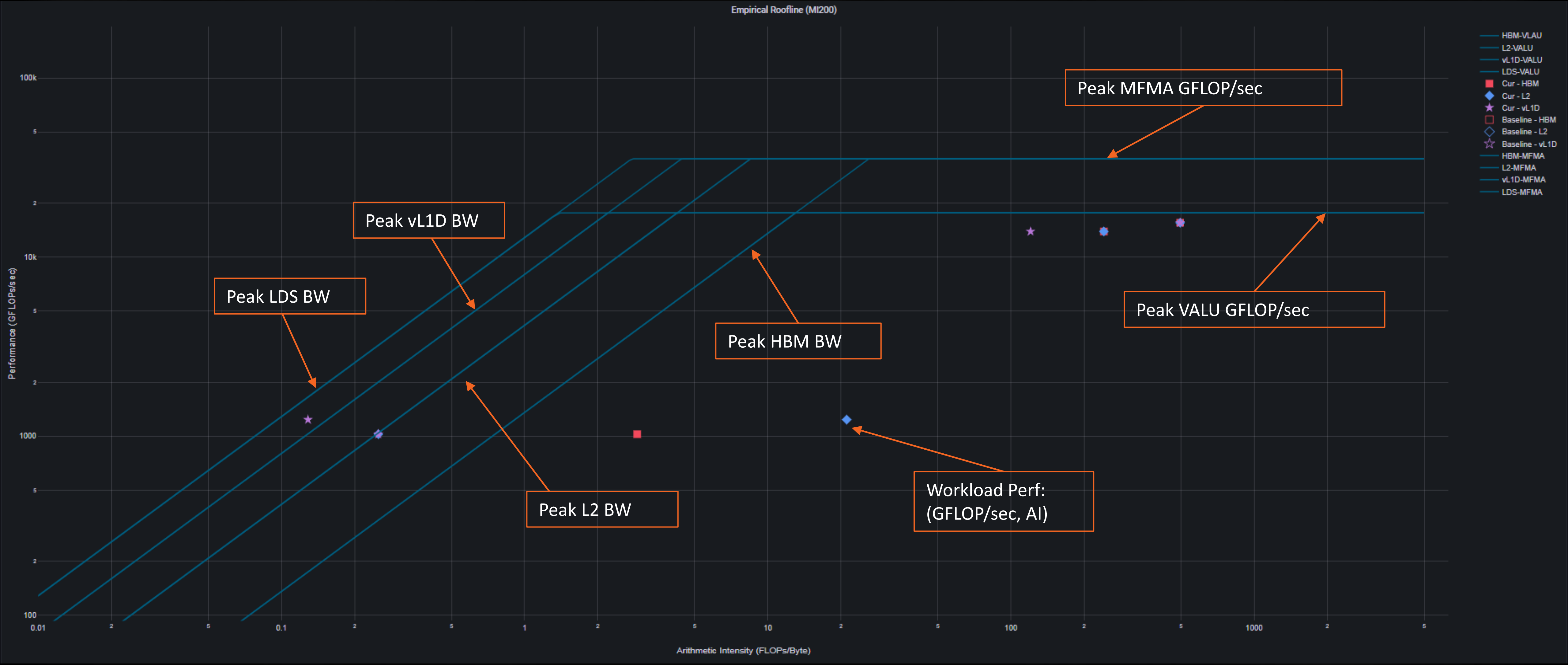
Background – What is “Good” Performance

- Example:
  - We run a number of kernels and measure FLOPs/s
  - Sort kernels by arithmetic intensity
  - Compare performance relative to hardware capabilities
  - Kernels near the roofline are making good use of computational resources
    - Kernels can have low performance (FLOPS/s), but make good use of BW
  - Increase arithmetic intensity when bandwidth limited
    - Reducing data movement increases AI
  - Kernels not near the roofline *should** have optimizations that can be made to get closer to the roofline



AMD

Empirical Hierarchical Roofline on MI200 - Overview



AMD

Empirical Roofline Benchmarking (MIPerf)

- Empirical Roofline Benchmarking
  - Measure achievable Peak FLOPS
    - VALU: F32, F64
    - MFMA: F16, BF16, F32, F64
  - Measure achievable Peak BW
    - LDS
    - Vector L1D Cache
    - L2 Cache
    - HBM

```
10:57:35 amd@node-bp126-014a utils ±|master ✗|→ ./roofline
Total detected GPU devices: 2
GPU Device 0: Profiling...
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
HBM BW, GPU ID: 0, workgroupSize:256, workgroups:2097152, experiments:100, Total Bytes=8589934592, Duration=6.2 ms, Mean=1382.7 GB/sec, stdev=2.6 GB/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
L2 BW, GPU ID: 0, workgroupSize:256, workgroups:8192, experiments:100, Total Bytes=687194767360, Duration=157.3 ms, Mean=4321.3 GB/sec, stdev=59.1 GB/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
L1 BW, GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total Bytes=26843545600, Duration=3.3 ms, Mean=8262.6 GB/sec, stdev=5.9 GB/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
LDS BW, GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total Bytes=33554432000, Duration=1.8 ms, Mean=18780.4 GB/sec, stdev=33.0 GB/s
nSize:134217728, 268435456000
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
Peak FLOPs (FP32), GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total FLOPS=274877906944, Duration=14.482 ms, Mean=18977.7 GFLOPs/sec, stdev=3.6 GFLOPs/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
Peak FLOPs (FP64), GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total FLOPS=137438953472, Duration=7.5 ms, Mean=18336.156250.1 GFLOPs/sec, stdev=5.0 GFLOPs/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
Peak MFMA FLOPs (BF16), GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total FLOPS=2147483648000, Duration=14.0 ms, Mean=153763.7 GFLOPs/sec, stdev=61.0 GFLOPs/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
Peak MFMA FLOPs (F16), GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total FLOPS=2147483648000, Duration=14.5 ms, Mean=147890.9 GFLOPs/sec, stdev=32.2 GFLOPs/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
Peak MFMA FLOPs (F32), GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total FLOPS=536870912000, Duration=14.4 ms, Mean=37200.4 GFLOPs/sec, stdev=9.3 GFLOPs/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
Peak MFMA FLOPs (F64), GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total FLOPS=268435456000, Duration=7.3 ms, Mean=36978.4 GFLOPs/sec, stdev=10.0 GFLOPs/s
```

- Internally developed micro benchmark algorithms
  - Peak VALU FLOP: axpy
  - Peak MFMA FLOP: Matrix multiplication based on MFMA intrinsic
  - Peak LDS/vL1D/L2 BW: Pointer chasing
  - Peak HBM BW: Streaming copy

AMD

Roofline Arithmetic: Perfmon Counters

- Weight
  - ADD: 1
  - MUL: 1
  - FMA: 2
  - Transcendental: 1
- FLOP Count
  - VALU: derived from VALU math instructions (assuming 64 active threads)
  - MFMA: count FLOP directly, in unit of 512
- Transcendental Instructions (7 in total)
  - e^x, $\log(x)$: F16, F32
  - $\frac{1}{x}$, $\sqrt{x}$, $\frac{1}{\sqrt{x}}$: F16, F32, F64
  - $\sin x$, $\cos x$: F16, F32
- Profiling Overhead
  - Require 3 application replays

```
v_rcp_f64_e32 v[4:5], v[2:3]
v_sin_f32_e32 v2, v2
v_cos_f32_e32 v2, v2
v_rsq_f64_e32 v[6:7], v[2:3]
v_sqrt_f32_e32 v3, v2
v_log_f32_e32 v2, v2
v_exp_f32_e32 v2, v2
```

ID	HW Counter	Category
1	SQ_INSTS_VALU_ADD_F16	FLOP counter
2	SQ_INSTS_VALU_MUL_F16	FLOP counter
3	SQ_INSTS_VALU_FMA_F16	FLOP counter
4	SQ_INSTS_VALU_TRANS_F16	FLOP counter
5	SQ_INSTS_VALU_ADD_F32	FLOP counter
6	SQ_INSTS_VALU_MUL_F32	FLOP counter
7	SQ_INSTS_VALU_FMA_F32	FLOP counter
8	SQ_INSTS_VALU_TRANS_F32	FLOP counter
9	SQ_INSTS_VALU_ADD_F64	FLOP counter
10	SQ_INSTS_VALU_MUL_F64	FLOP counter
11	SQ_INSTS_VALU_FMA_F64	FLOP counter
12	SQ_INSTS_VALU_TRANS_F64	FLOP counter
13	SQ_INSTS_VALU_INT32	IOP counter
14	SQ_INSTS_VALU_INT64	IOP counter
15	SQ_INSTS_VALU_MFMA_MOPS_I8	IOP counter

ID	HW Counter	Category
16	SQ_INSTS_VALU_MFMA_MOPS_F16	FLOP counter
17	SQ_INSTS_VALU_MFMA_MOPS_BF16	FLOP counter
18	SQ_INSTS_VALU_MFMA_MOPS_F32	FLOP counter
19	SQ_INSTS_VALU_MFMA_MOPS_F64	FLOP counter
20	SQ_LDS_IDX_ACTIVE	LDS Bandwidth
21	SQ_LDS_BANK_CONFLICT	LDS Bandwidth
22	TCP_TOTAL_CACHE_ACCESSES_sum	vL1D Bandwidth
23	TCP_TCC_WRITE_REQ_sum	L2 Bandwidth
24	TCP_TCC_ATOMIC_WITH_RET_REQ_sum	L2 Bandwidth
25	TCP_TCC_ATOMIC_WITHOUT_RET_REQ_sum	L2 Bandwidth
26	TCP_TCC_READ_REQ_sum	L2 Bandwidth
27	TCC_EA_RDREQ_sum	HBM Bandwidth
28	TCC_EA_RDREQ_32B_sum	HBM Bandwidth
29	TCC_EA_WRREQ_sum	HBM Bandwidth
30	TCC_EA_WRREQ_64B_sum	HBM Bandwidth

Roofline Arithmetic: Metrics

$$
\begin{aligned}
\text{Total\_FLOP} = \; & 64 * (\text{SQ\_INSTS\_VALU\_ADD\_F16} + \text{SQ\_INSTS\_VALU\_MUL\_F16} + \text{SQ\_INSTS\_VALU\_TRANS\_F16} + 2 * \text{SQ\_INSTS\_VALU\_FMA\_F16}) \\
& + 64 * (\text{SQ\_INSTS\_VALU\_ADD\_F32} + \text{SQ\_INSTS\_VALU\_MUL\_F32} + \text{SQ\_INSTS\_VALU\_TRANS\_F32} + 2 * \text{SQ\_INSTS\_VALU\_FMA\_F32}) \\
& + 64 * (\text{SQ\_INSTS\_VALU\_ADD\_F64} + \text{SQ\_INSTS\_VALU\_MUL\_F64} + \text{SQ\_INSTS\_VALU\_TRANS\_F64} + 2 * \text{SQ\_INSTS\_VALU\_FMA\_F64}) \\
& + 512 * \text{SQ\_INSTS\_VALU\_MFMA\_MOPS\_F16} \\
& + 512 * \text{SQ\_INSTS\_VALU\_MFMA\_MOPS\_BF16} \\
& + 512 * \text{SQ\_INSTS\_VALU\_MFMA\_MOPS\_F32} \\
& + 512 * \text{SQ\_INSTS\_VALU\_MFMA\_MOPS\_F64}
\end{aligned}
$$

$$\text{Total\_IOP} = 64 * (\text{SQ\_INSTS\_VALU\_INT32} + \text{SQ\_INSTS\_VALU\_INT64})$$

$$LDS_{BW} = 32 * 4 * (\text{SQ\_LDS\_IDX\_ACTIVE} - \text{SQ\_LDS\_BANK\_CONFLICT})$$

$$vL1D_{BW} = 64 * \text{TCP\_TOTAL\_CACHE\_ACCESSES\_sum}$$

$$
\begin{aligned}
L2_{BW} = \; & 64 * \text{TCP\_TCC\_READ\_REQ\_sum} \\
& + 64 * \text{TCP\_TCC\_WRITE\_REQ\_sum} \\
& + 64 * (\text{TCP\_TCC\_ATOMIC\_WITH\_RET\_REQ\_sum} + \text{TCP\_TCC\_ATOMIC\_WITHOUT\_RET\_REQ\_sum})
\end{aligned}
$$

$$
\begin{aligned}
HBM_{BW} = \; & 32 * \text{TCC\_EA\_RDREQ\_32B\_sum} + 64 * (\text{TCC\_EA\_RDREQ\_sum} - \text{TCC\_EA\_RDREQ\_32B\_sum}) \\
& + 32 * (\text{TCC\_EA\_WRREQ\_sum} - \text{TCC\_EA\_WRREQ\_64B\_sum}) + 64 * \text{TCC\_EA\_WRREQ\_64B\_sum}
\end{aligned}
$$

$$AI_{LDS} \; \frac{TOTAL\_FLOP}{LDS_{BW}}$$

$$AI_{vL1D} \; \frac{TOTAL\_FLOP}{vL1D_{BW}}$$

$$AI_{L2} \; \frac{TOTAL\_FLOP}{L2_{BW}}$$

$$AI_{HBM} = \frac{TOTAL\_FLOP}{HBM_{BW}}$$

*** All calculations are subject to change***

AMD

Roofline Analysis: Manual Rocprof

- For those who like getting their hands dirty
- Generate input file
  - See example roof-counters.txt →
- Run rocprof

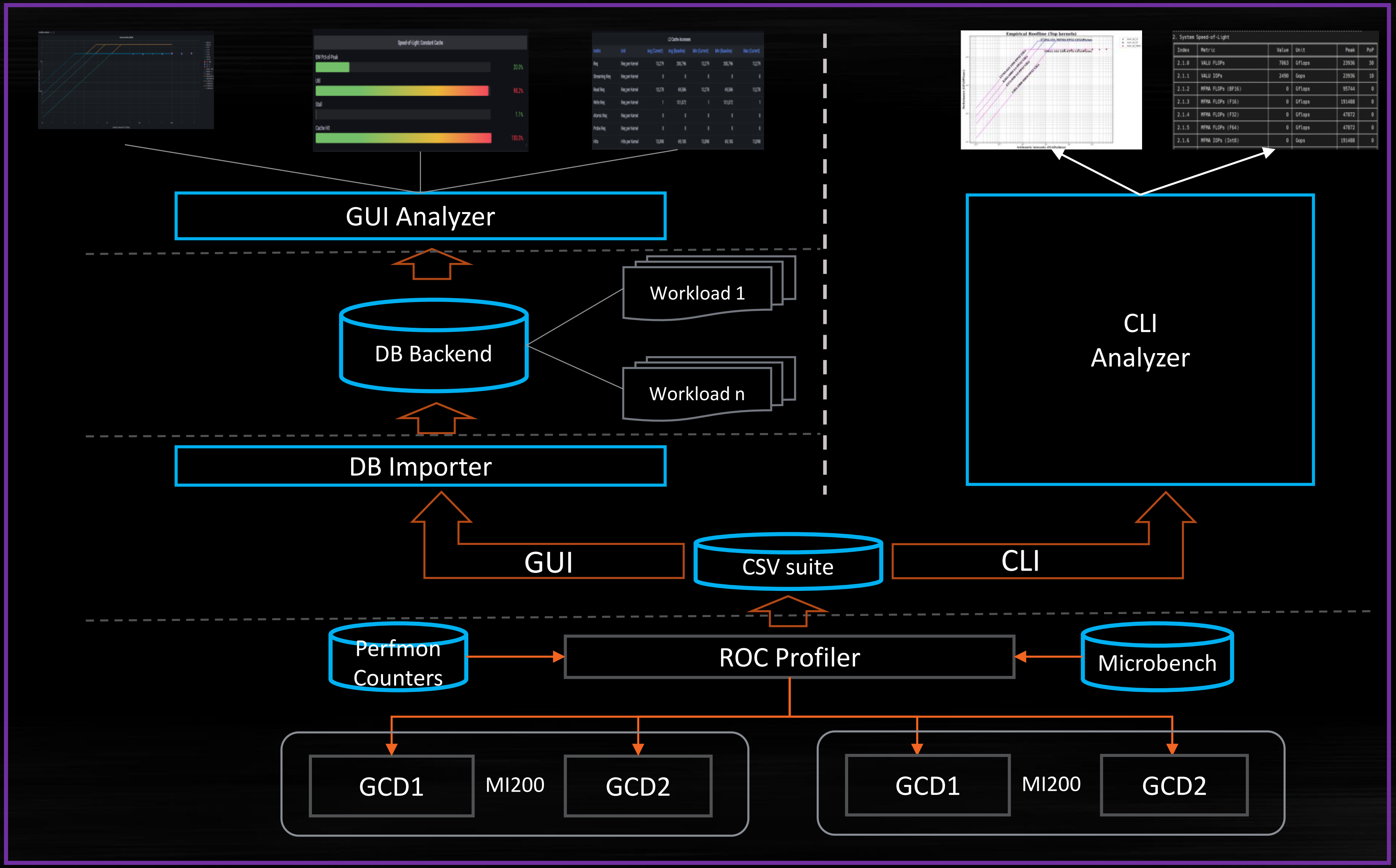
```
foo@bar:~$ rocprof -i roof-counters.txt --timestamp on ./myCoolApp
```

- Analyze results
  - Load *results.csv* output file in csv viewer of choice
  - Derive final metric values using equations on previous slide
- Profiling Overhead
  - Requires one application replay for each pmc line

```
## roof-counters.txt

# FP32 FLOPs
pmc: SQ_INSTS_VALU_ADD_F32 SQ_INSTS_VALU_MUL_F32 SQ_INSTS_VALU_FMA_F32 SQ_INSTS_VALU_TRANS_F32

# HBM Bandwidth
pmc: TCC_EA_RDREQ_sum TCC_EA_RDREQ_32B_sum TCC_EA_WRREQ_sum TCC_EA_WRREQ_64B_sum

# LDS Bandwidth
pmc: SQ_LDS_IDX_ACTIVE SQ_LDS_BANK_CONFLICT

# L2 Bandwidth
pmc: TCP_TCC_READ_REQ_sum TCP_TCC_WRITE_REQ_sum TCP_TCC_ATOMIC_WITH_RET_REQ_sum
TCP_TCC_ATOMIC_WITHOUT_RET_REQ_sum

# vL1D Bandwidth
pmc: TCP_TOTAL_CACHE_ACCESSES_sum
```


AMD

Roofline Analysis Workflow – Tooling (MIPerf)



AMD

Roofline Analysis Workflow – Tooling

Workload Profiling

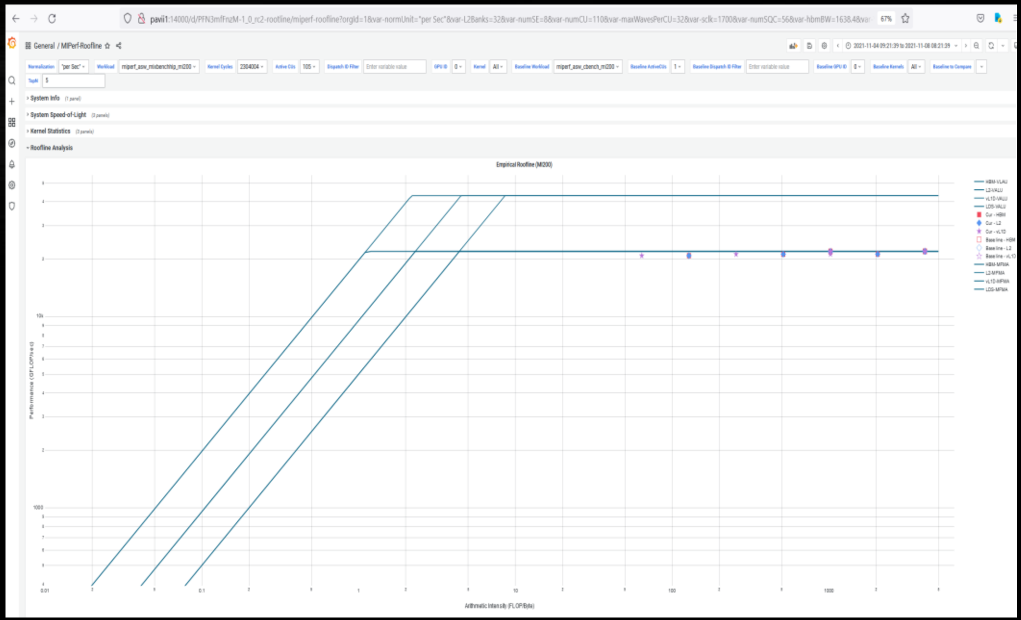
```
miperf profile -n mixbench-hip -c "./mixbench-hip-ro"
```

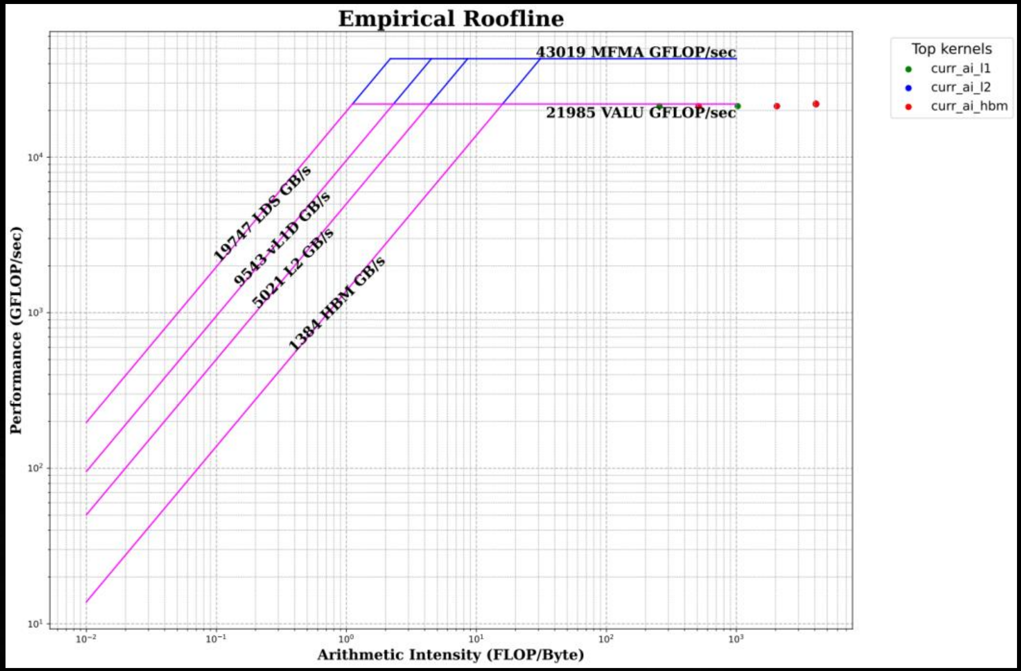
GUI Import

```
miperf import -i -n mixbenchhip --path workloads/mixbench-
hip/mi200/ -H pavii1 -u amd -t asw
```

CLI Roofline Generation

```
miperf analyze --gen-roofline -p workloads/mixbench-hip/mi200/
-c "./mixbench-hip-ro"
```



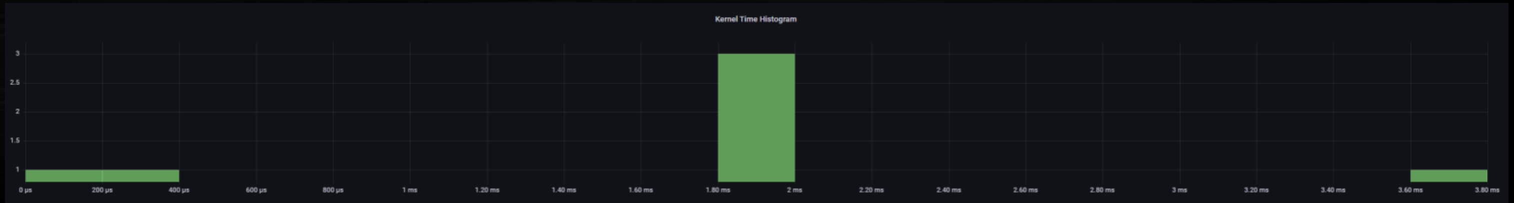


**** All CLI options are subject to change due to fast prototyping. Refer to** [MI Performance Profiler (MIPerf) - Audacious Software Team - Confluence (amd.com)]* for the up-to-date documentation.

AMD

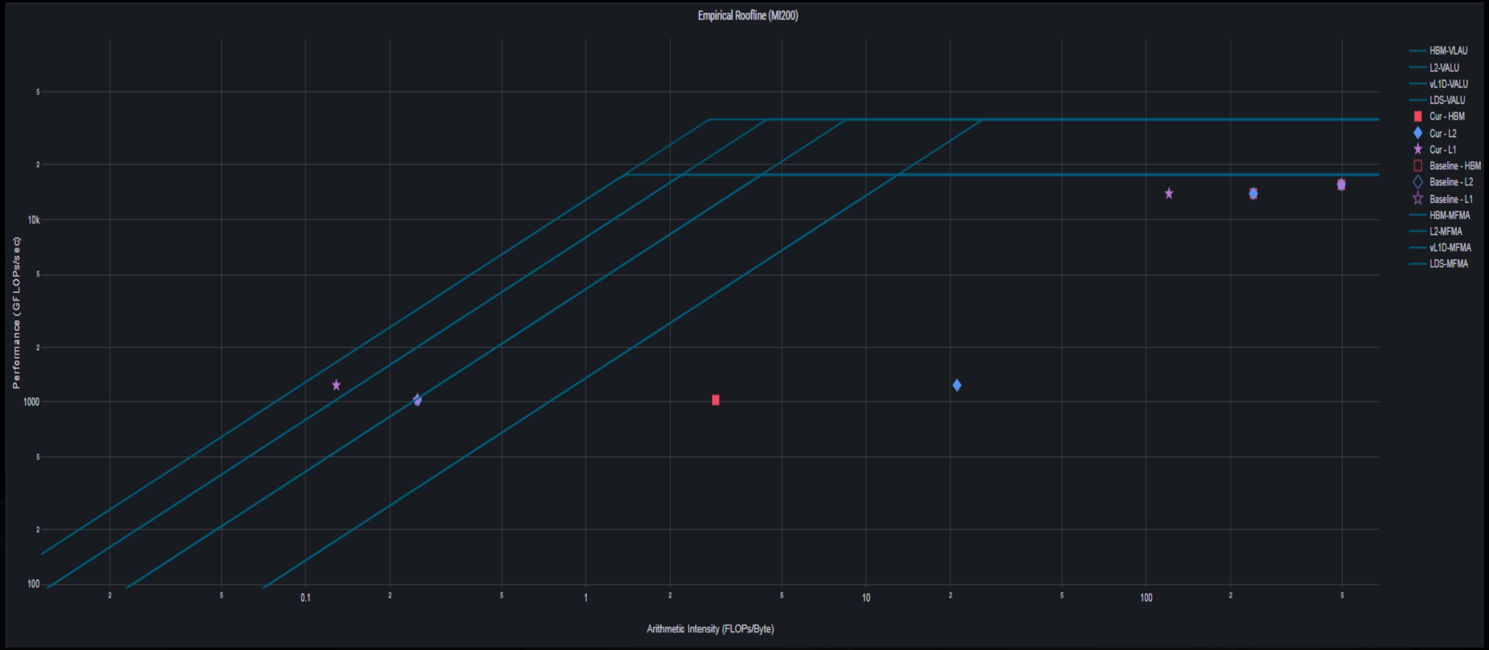
Roofline Analysis Workflow – Features

- GUI Analyzer
  - Histogram
  - Kernel statistics
  - Kernel filtering
  - Baseline comparison
  - Memory hierarchy selection
- CLI Analyzer
  - Kernel filtering
  - Memory hierarchy selection
  - x/y-axis range selection
  - Pdf/jpg file export



Top Kernels

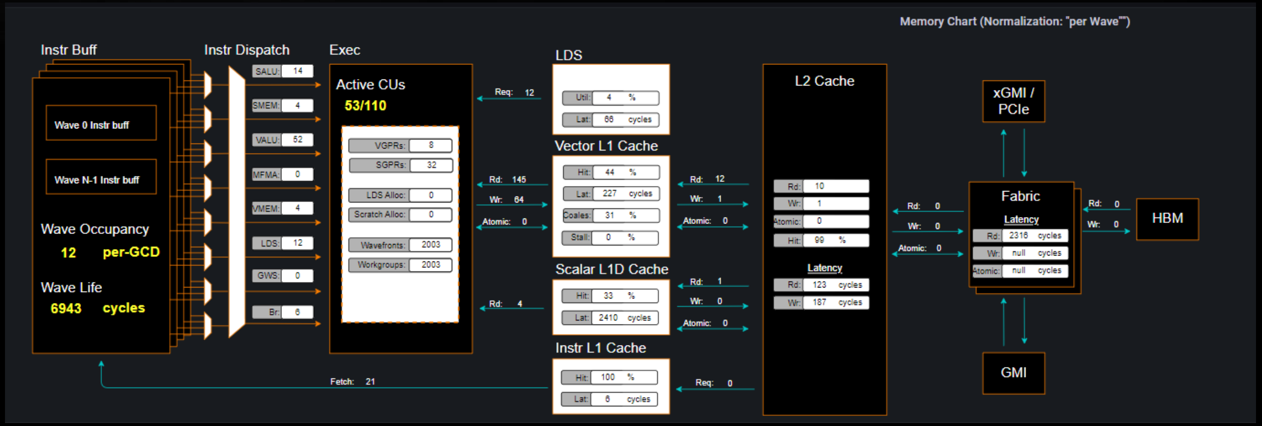
Name	Calls	Performance	HBM BW	Total Duration	Avg Duration	AI (L1 Cache)	AI (L2 Cache)	AI (HBM)	Total FLOPs	VALU FLOPs	MFMA FLOPs (F16)	MFMA FLOPs (BF16)	MFMA FLOPs (F32)	MFMA FLOPs (F64)	LDS	L1 Cache	L2 Cache	HBM
void benchmark_func...	1	0 GFLOPS	35.9 GB/s	3.74 ms	3.74 ms	0	0	0	0	0	0	0	0	0	0 B	134 MB	134 MB	134 MB
void benchmark_func...	1	18.0 TFLOPS	141 GB/s	1.91 ms	1.91 ms	64.063	128.125	128.117	34,393,292,800	34,393,292,800	0	0	0	0	0 B	537 MB	268 MB	268 MB
void benchmark_func...	1	18.5 TFLOPS	72.3 GB/s	1.86 ms	1.86 ms	256.25	256.25	256.228	34,393,292,800	34,393,292,800	0	0	0	0	0 B	134 MB	134 MB	134 MB
void benchmark_func...	1	18.6 TFLOPS	72.7 GB/s	1.85 ms	1.85 ms	256.25	256.25	256.218	34,393,292,800	34,393,292,800	0	0	0	0	0 B	134 MB	134 MB	134 MB
__amd_rocclr_fillBuffe...	1	0 GFLOPS	1.05 TB/s	252 μs	252 μs	0	0	0	0	0	0	0	0	0	0 B	537 MB	268 MB	264 MB
void benchmark_func...	1	0 GFLOPS	73.5 GB/s	7.20 μs	7.20 μs	0	0	0	0	0	0	0	0	0	0 B	1.05 MB	524 kB	529 kB



AMD

Roofline Analysis Workflow – Bottleneck Analysis Recipes

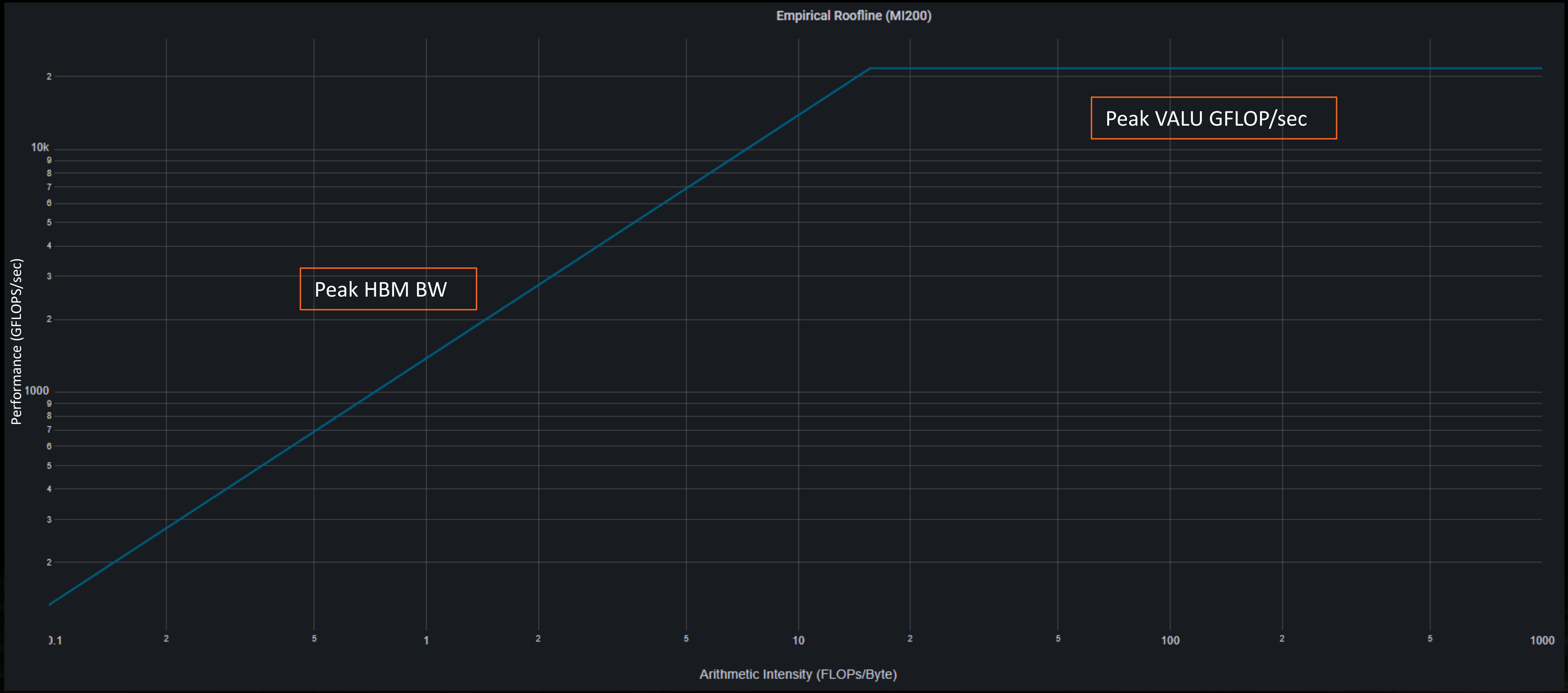
- Roofline analysis is part of the integrated performance analysis on GPUs
  - Except for simple Compute and Memory BW bound, in-depth profiling and tracing analysis is needed
- Compute Bound workloads
  - Compute Unit Analysis (Instruction Mix, pipeline perf
- Memory BW Bound workloads
  - L2 Cache analysis (BW, Util, cache hit)
  - Vector L1D Cache analysis (BW, Util, cache hit)
  - LDS analysis (BW, Util, latency, bank conflict)
- Memory Latency Bound workloads
  - Wavefront Life analysis
    - Dependency wait, instr issue wait
  - Vector L1D Cache analysis
    - vL1D stall/util/bw/latency
  - Compute Unit analysis
    - VMEM/SMEM/LDS latency
- Dispatch Bound workloads
  - SPI analysis
    - Waveslot/LDS/VGPR/SGPR limit



Roofline Examples on AMD Instinct™ MI250X GPU

Roofline Plot – AMD Instinct™ MI250X Accelerators

- Device: Instinct MI250X
  - ORNL Frontier GPU
- Instinct MI250X (Dual GCDs)
- Figure shows single GCD
- Methodology applies to all AMD Instinct MI200 series GPUs



*AMD Instinct™ MI250X accelerator Datasheet: amd.com/system/files/documents/amd-instinct-mi200-datasheet.pdf

Roofline Example #1 – Add

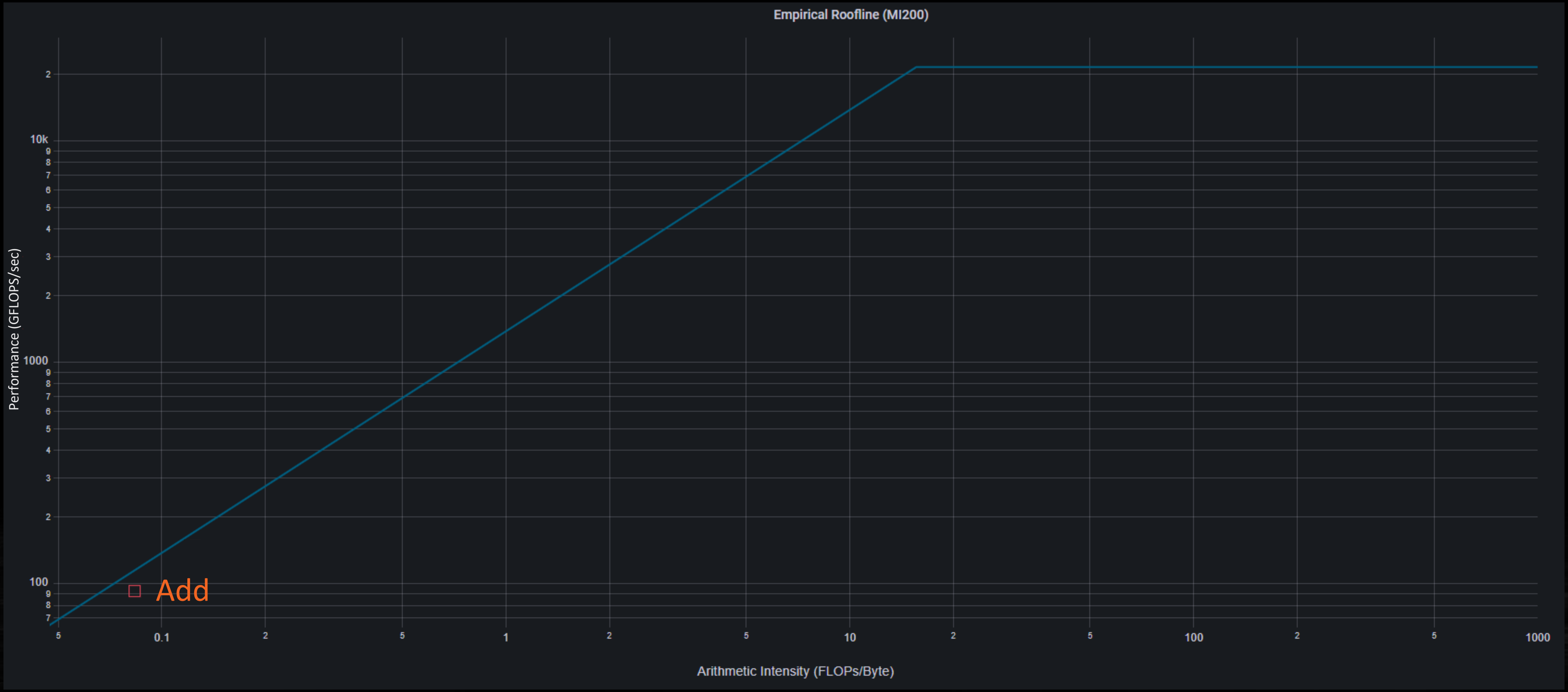
- Calculation:
  - a[i] = a[i] + b[i]
- VALU Ops Per Thread:
  - 1x V_ADD
- HBM MEM Ops Per Thread:
  - 2x RD
  - 1x WR
- Arithmetic Intensity:
  - 1 FLOP / (3 * 4Byte) = 1/12

```
template<typename T>
__global__ void add_benchmark(T *buf1, T *buf2, uint32_t nSize)
{
    const uint32_t gid = hipBlockDim_x * hipBlockIdx_x + hipThreadIdx_x;
    const uint32_t nThreads  = gridDim.x * blockDim.x;


    T *a, *b;
    a = &buf1[gid];
    b = &buf2[gid];


    for(uint32_t offset=0; offset < nSize; offset += nThreads)
    {
        a[offset] = a[offset] + b[offset];
    }
}
```

Roofline Example #1 – Add

- Calculation:
  - a[i] = a[i] + b[i]

- Reading two floats for every add results in low arithmetic intensity and HBM limited



*MI250x. RESULTS MAY VARY. SEE ENDNOTE: MI200-57

AMD

Roofline Example #2 – Mul

- Calculation:
  - a[i] = x * b[i]
- VALU Ops Per Thread:
  - 1x V_MUL
- HBM MEM Ops Per Thread:
  - 1x RD
  - 1x WR
- Arithmetic Intensity:
  - 1 FLOP / (2 * 4Byte) = 1/8

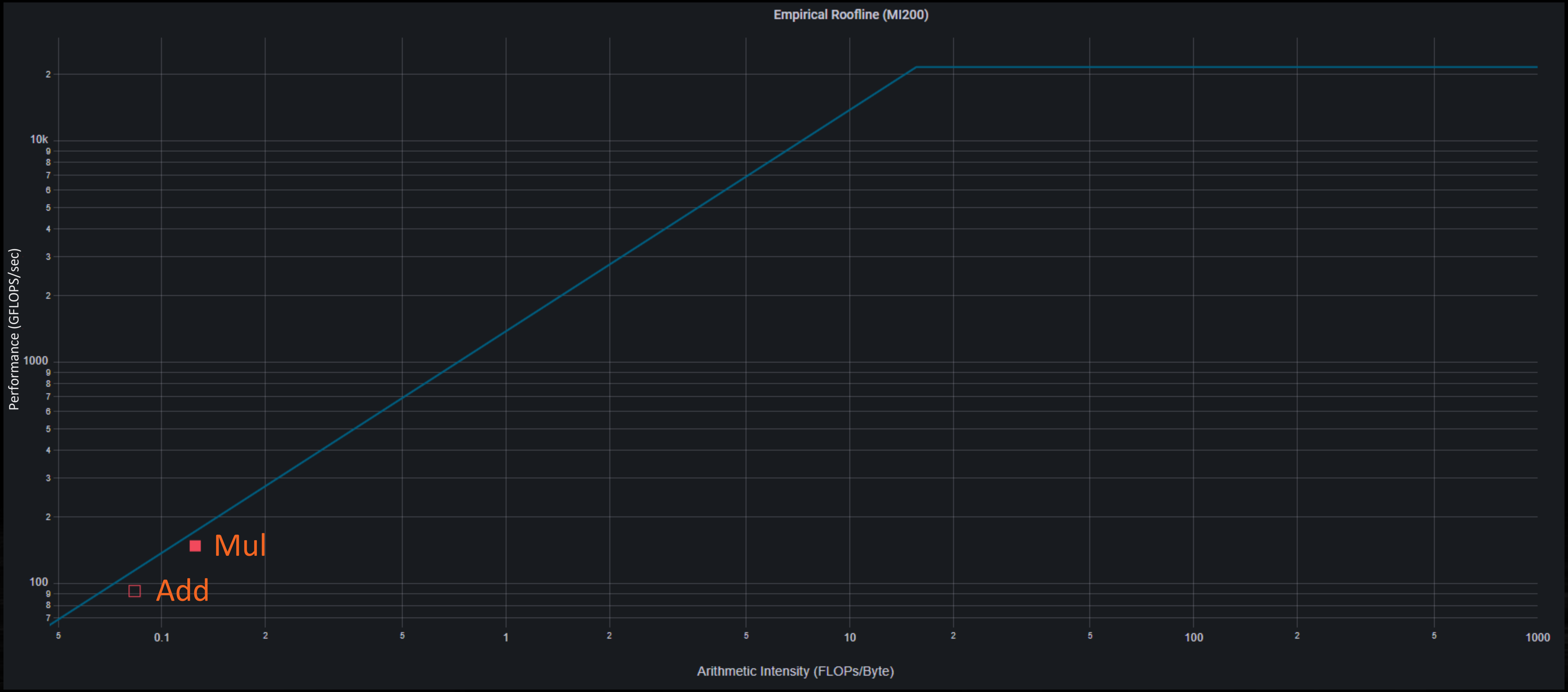
```
template<typename T>
__global__ void mul_benchmark(T *buf1, T *buf2, uint32_t nSize)
{
    const uint32_t gid = hipBlockDim_x * hipBlockIdx_x + hipThreadIdx_x;
    const uint32_t nThreads  = gridDim.x * blockDim.x;


    T *a, *b;
    a = &buf1[gid];
    b = &buf2[gid];
    const T x = (T)1.2;


    for(uint32_t offset=0; offset < nSize; offset += nThreads)
    {
      a[offset] = x * b[offset];
    }
}
```


AMD

Roofline Example #2 – Mul

- Calculation:
  - a[i] = c * b[i]

- Reading one less float (compared to Add) increases our arithmetic intensity and reduces sensitivity to HBM



*MI250x. RESULTS MAY VARY. SEE ENDNOTE: MI200-57, MI200-58

Roofline Example #3 – Triad

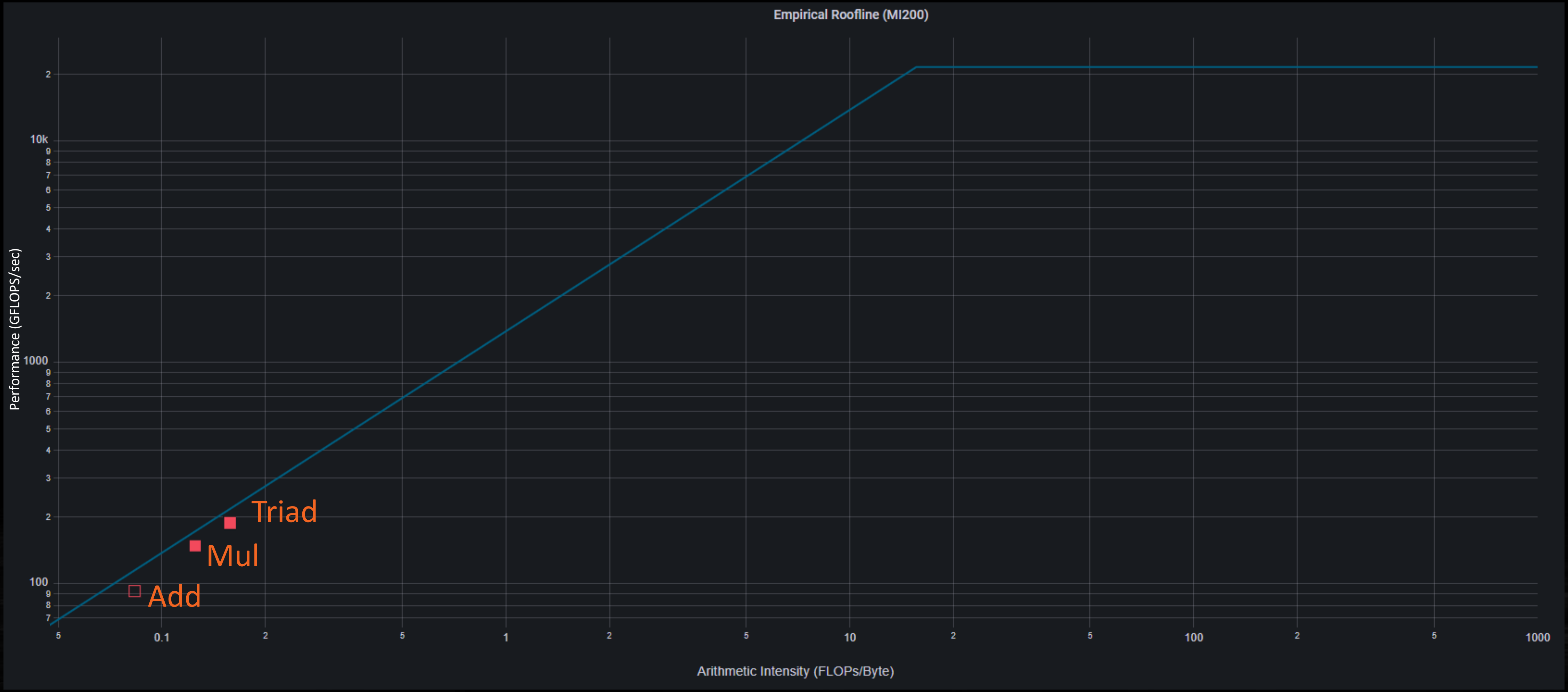
- Calculation:
  - a[i] = b[i] + x * a[i]
- VALU Ops Per Thread:
  - 1x V_ADD
  - 1x V_MUL

  1x V_FMA
- HBM MEM Ops Per Thread:
  - 2x RD
  - 1x WR
- Arithmetic Intensity:
  - 2 FLOP / (3 * 4Byte) = 1/6

```
template<typename T>
__global__ void triad_benchmark(T *buf1, T *buf2, uint32_t nSize)
{
    const uint32_t gid = hipBlockDim_x * hipBlockIdx_x + hipThreadIdx_x;
    const uint32_t nThreads  = gridDim.x * blockDim.x;


    T *a, *b;
    a = &buf1[gid];
    b = &buf2[gid];
    const T x = (T)1.2;


    for(uint32_t offset=0; offset < nSize; offset += nThreads)
    {
      a[offset] = b[offset] + x * a[offset];
    }
}
```

Roofline Example #3 – Triad

- Calculation:
  - a[i] = b[i] + x * a[i]
- Performing an extra operation increases arithmetic intensity and further reduces sensitivity to HBM as compared to Add and Mul



*MI250x. RESULTS MAY VARY. SEE ENDNOTE: MI200-57, MI200-58, MI200-59

Roofline Example #4 – FMA

- Calculation:
  - x = a[i] * x + y
- VALU Ops Per Thread:
  - 1x V_ADD
  - 1x V_MUL

  → 1x V_FMA
- HBM MEM Ops Per Thread:
  - 1x RD
- Arithmetic Intensity:
  - 2 FLOP / (1 * 4Byte) = 1/2

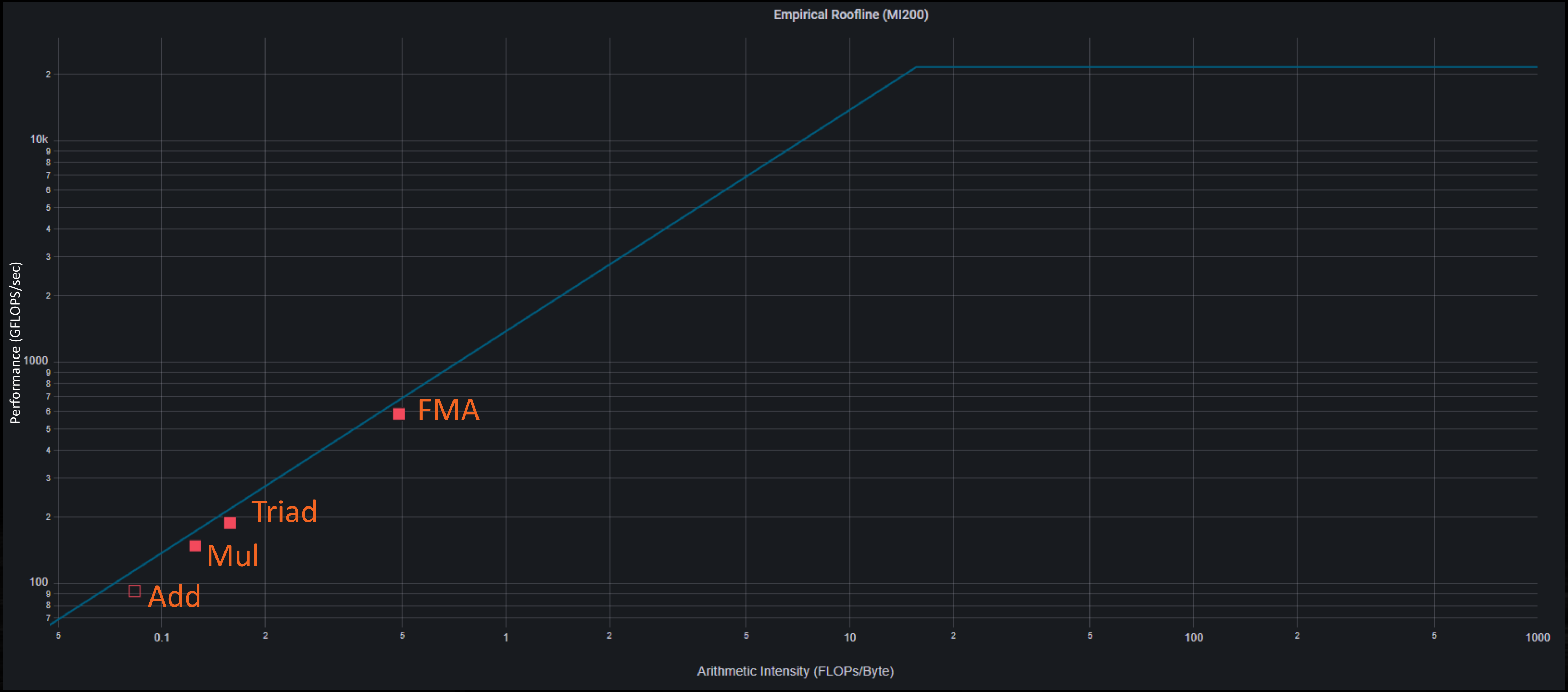
```
template<typename t>
__global__ void flops_benchmark(T *buf, uint32_t nSize)
{
    const uint32_t gid = hipBlockDim_x * hipBlockIdx_x + hipThreadIdx_x;
    const uint32_t nThreads  = gridDim.x * blockDim.x;


    T *a;
    a = &buf[gid];
    const T y = (T) 1.0;
    T x = (T) 2.0;


    for(uint32_t offset=0; offset < nSize; offset += nThreads)
    {
        x = a[offset] * x + y;
    }
    a[0] = -x;
}
```


AMD

Roofline Example #4 – FMA

- Calculation:
  - x = a[i] * x + y

- Each thread having to load one less value from HBM further increases arithmetic intensity and improves FLOPs/s performance



*MI250x. RESULTS MAY VARY. SEE ENDNOTE: MI200-57, MI200-58, MI200-59, MI200-60

Roofline Example #5 – FMA 1024

- Calculation:
  - x = a[i] * x + y
- VALU Ops Per Thread:
  - 1x V_ADD
  - 1x V_MUL

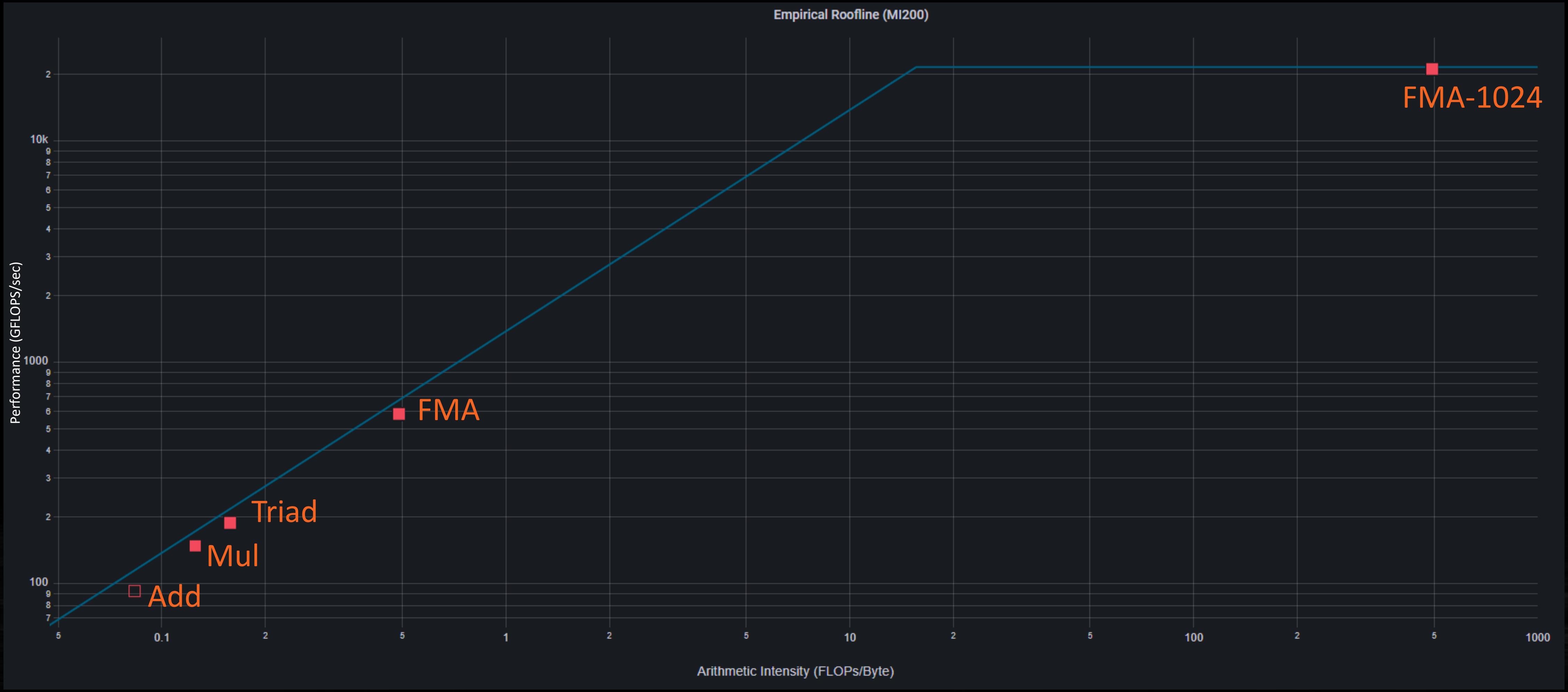
  1x V_FMA
- HBM MEM Ops Per Thread:
  - 1x RD
- Arithmetic Intensity:
  - 1024 * 2 FLOP / (1 * 4Byte) = 512

1024

```
template<typename T, int nFMA>
__global__ void flops_benchmark(T *buf, uint32_t nSize)
{
    const uint32_t gid = hipBlockDim_x * hipBlockIdx_x + hipThreadIdx_x;
    const uint32_t nThreads  = gridDim.x * blockDim.x;


    T *a;
    a = &buf[gid];
    const T y = (T) 1.0;
    T x = (T) 2.0;


    for(uint32_t offset=0; offset < nSize; offset += nThreads)
    {
        for(int j=0; j<nFMA; j++)
        {
            x = a[offset] * x + y;
        }
    }
    a[0] = -x;
}
```

Roofline Example #5 – FMA 1024

- Calculation:
  - x = a[i] * x + y
- Each thread looping over many FMAs with only one read significantly increases arithmetic intensity and becomes compute VALU limited

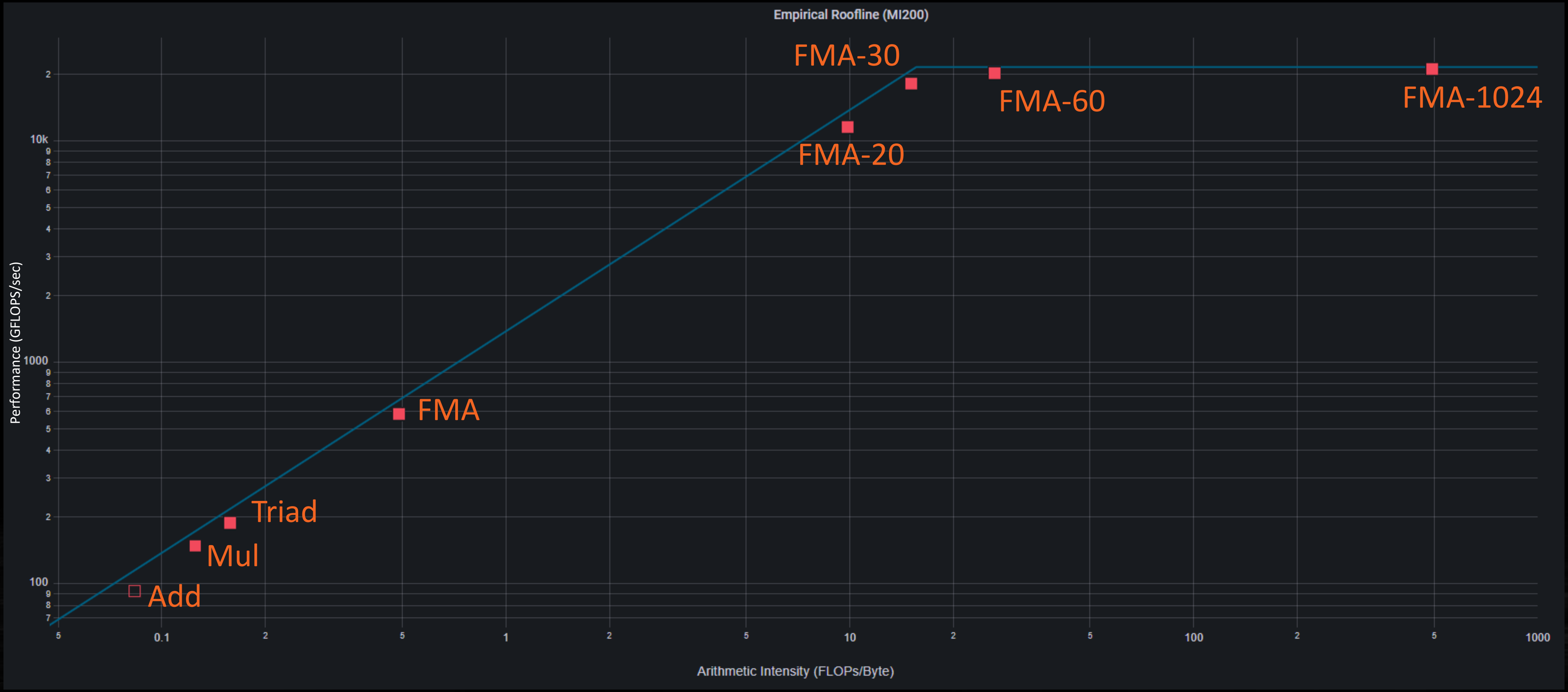


*MI250x. RESULTS MAY VARY. SEE ENDNOTE: MI200-57, MI200-58, MI200-59, MI200-60

Roofline Example #6 – FMA Sweep

- Calculation:
  - x = a[i] * x + y

- Further sweeping the number of FMA instructions from 20 to 60 shows the workload transitioning from HBM limited to VALU limited



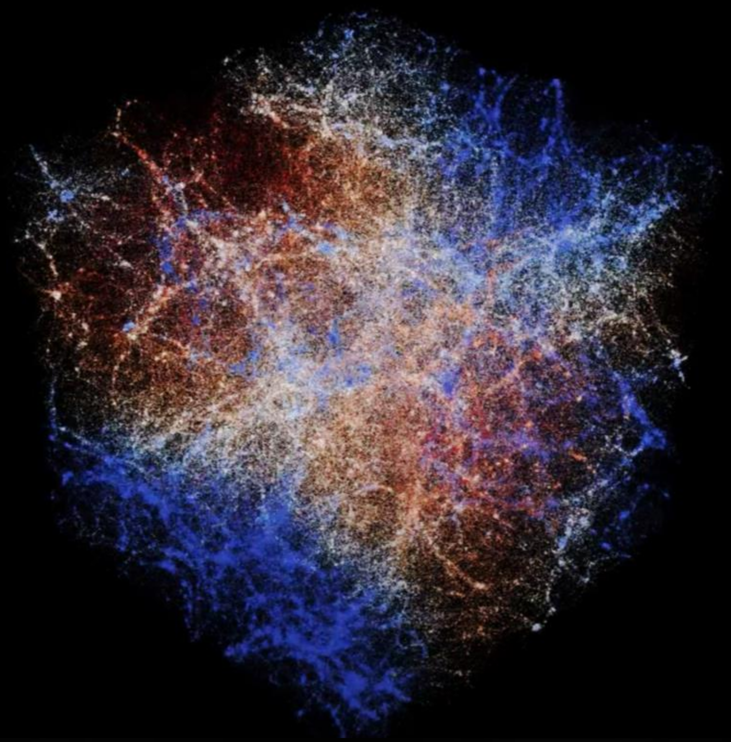
*MI250x. RESULTS MAY VARY. SEE ENDNOTE: MI200-57, MI200-58, MI200-59, MI200-60

AMD

Roofline Use Case - HPC Particle Codes

- Particle interactions form the foundation of many computational science codes from multiple domains
  - Domains: Cosmology, astrophysics, molecular dynamics, and more
  - Applications: HACC, LAMMPS, NAMD, Amber, GROMACS

HACC – Cosmology



Nbody

- One such computational algorithm for computing particle interactions leveraged by these applications
- Direct particle-particle method
- Highly accurate
- Computationally expensive (N^2)

AMD

Roofline Example #1 – Nbody

- Repo: https://github.com/ROCm-Developer-Tools/HIP-Examples/tree/master/mini-nbody/hip
  - Fundamental particle-particle algorithm
  - Single collection of N particles calculating N^2 pair-wise interactions
  - Double precision (FP64)
  - Multiple implementations leveraging different optimization approaches
- "orig"
  - Numerical Computing 101 unoptimized implementation
- "soa"
  - Converting particle data layout from array of structures to structure of arrays
- "block"
  - Loading and computing particle data in "tiles" to increase cache hits
- "unroll"
  - Adding #pragma unroll to particle "tile" processing for loop

Roofline Example #1 – Nbody

- "orig"
  - Numerical Computing 101 unoptimized implementation
- $O(n^2)$ Interaction Ops:
  - 3x V_ADD
  - 6x V_FMA
  - 2x V_MUL
  - 1x V_DIV
  - 1x V_SQRT
  - 3x RD

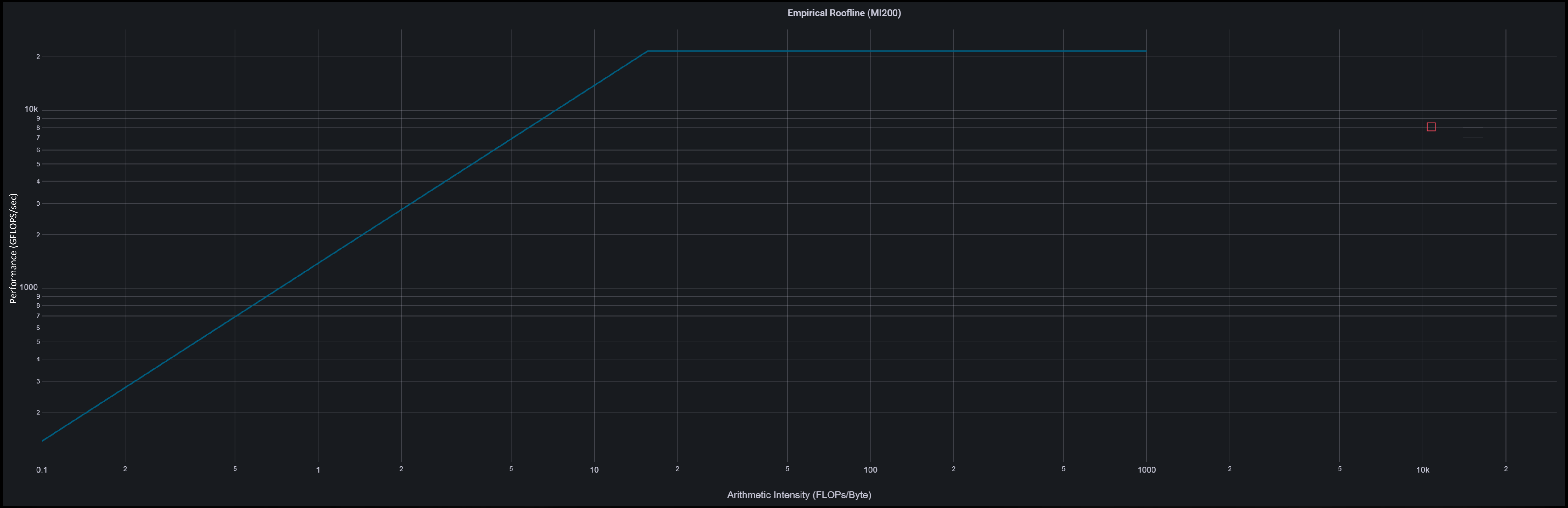
  (1x V_DIV and 1x V_SQRT → V_RSQ)
- O(n) Accumulation Ops:
  - 3x V_FMA
  - 3x RD
  - 3x WR
- Interaction AI:
  - $[(3 + 12 + 2 + 1)\text{FLOPs} / 24\text{Bytes}] * n^2 = (3/4)n^2$
- Accumulation AI:
  - $(6\text{ FLOPs} / 24\text{ Bytes}) * n = n/4$

```
typedef struct { double x, y, z, vx, vy, vz; } Body;

__global__
void bodyForce(Body *p, double dt, int n) {
  int i = blockDim.x * blockIdx.x + threadIdx.x;
  if (i < n) {
    double Fx = 0.0f; double Fy = 0.0f; double Fz = 0.0f;

    for (int j = 0; j < n; j++) {
      double dx = p[j].x - p[i].x;
      double dy = p[j].y - p[i].y;
      double dz = p[j].z - p[i].z;
      double distSqr = dx*dx + dy*dy + dz*dz + SOFTENING;
      double invDist = rsqrtf(distSqr);
      double invDist3 = invDist * invDist * invDist;

      Fx += dx * invDist3; Fy += dy * invDist3; Fz += dz * invDist3;
    }

    p[i].vx += dt*Fx; p[i].vy += dt*Fy; p[i].vz += dt*Fz;
  }
}
```

Roofline Example #1 – Nbody

- "orig"
  - Numerical Computing 101 unoptimized implementation

- Nbody has a very high arithmetic intensity and therefore closer to the top of the roofline (compute sensitive)
- Transcendentals like RSQ do not complete at same rate as ADD, MUL and FMA and therefore limit the peak FLOPS/s performance



*MI250x. RESULTS MAY VARY. SEE ENDNOTE: MI200-61

AMD

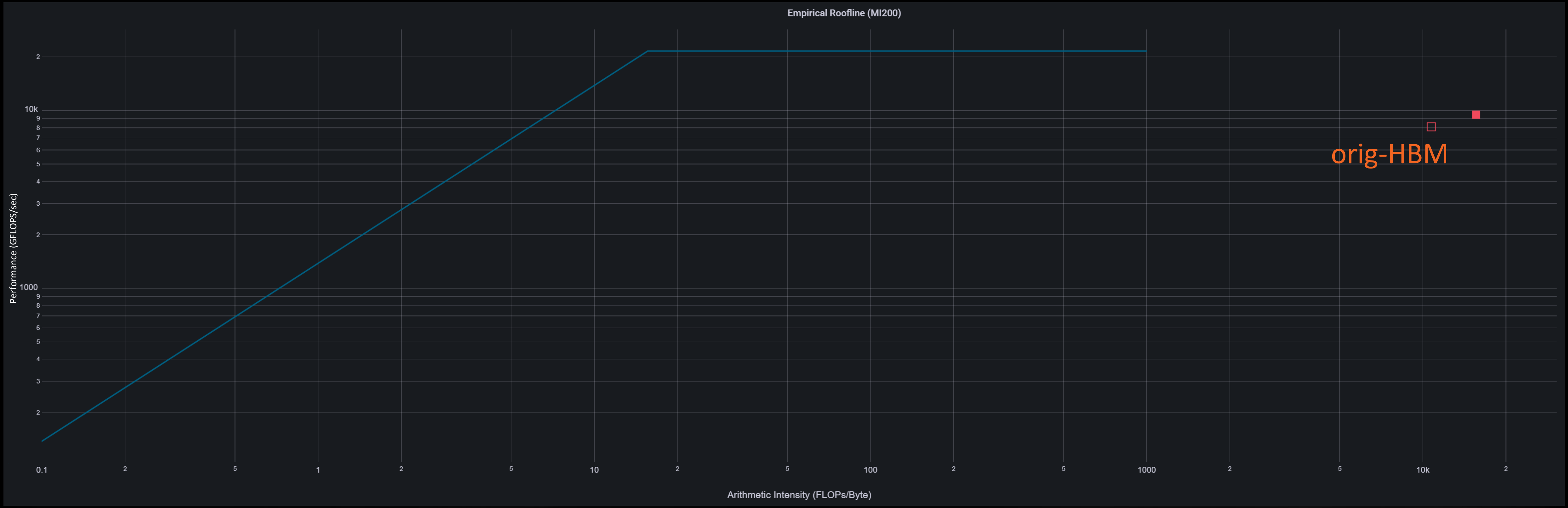
Roofline Example #1 – Nbody

- “block”
  - Preload a “tile” size worth of particle data into faster shared memory for computing O(n2) forces
- Processing in “tiles” improves reuse and increases cache hits

```
typedef struct { double4 *pos, *vel; } BodySystem;

__global__
void bodyForce(double4 *p, double4 *v, double dt, int n) {
  int i = blockDim.x * blockIdx.x + threadIdx.x;
  if (i < n) {
    double Fx = 0.0f; double Fy = 0.0f; double Fz = 0.0f;

    for (int tile = 0; tile < gridDim.x; tile++) {
      __shared__ double3 spos[BLOCK_SIZE];
      double4 tpos = p[tile * blockDim.x + threadIdx.x];
      spos[threadIdx.x] = make_double3(tpos.x, tpos.y, tpos.z);
      __syncthreads();

      for (int j = 0; j < BLOCK_SIZE; j++) {
        double dx = spos[j].x - p[i].x;
        double dy = spos[j].y - p[i].y;
        double dz = spos[j].z - p[i].z;
        double distSqr = dx*dx + dy*dy + dz*dz + SOFTENING;
        double invDist = rsqrtf(distSqr);
        double invDist3 = invDist * invDist * invDist;

        Fx += dx * invDist3; Fy += dy * invDist3; Fz += dz * invDist3;
      }
      __syncthreads();
    }

    v[i].x += dt*Fx; v[i].y += dt*Fy; v[i].z += dt*Fz;
  }
}
```

Roofline Example #1 – Nbody

- “block”
  - Loading and computing particle data in “tiles” to increase cache hits
- Working on smaller “tiles” of particles improves cache hits, removing loads from HBM and increasing FLOPs performance



*MI250x. RESULTS MAY VARY. SEE ENDNOTE: MI200-62

AMD

Roofline – All Workloads

Orange: Synthetic Workload Yellow: Proxy app Green: Full app

*MI250x. RESULTS MAY VARY. SEE ENDNOTE: MI200-62

DISCLAIMERS AND ATTRIBUTIONS

Math Kernel Endnotes

MI200-57 - Testing Conducted by AMD performance lab as of 4/19/2022 on a single socket optimized 3rd Gen AMD EPYC™ CPU powered server with 1x AMD Instinct™ MI250X OAM (128 GB HBM2e) 560W GPU with AMD Infinity Fabric™ technology resulted in a median score of 92.6 GFLOPS/s on Add Kernel. Server manufacturers may vary configurations, yielding different results. Performance may vary based on use of latest drivers and optimizations MI200-57

MI200-58 - Testing Conducted by AMD performance lab as of 4/19/2022 on a single socket optimized 3rd Gen AMD EPYC™ CPU powered server with 1x AMD Instinct™ MI250X OAM (128 GB HBM2e) 560W GPU with AMD Infinity Fabric™ technology resulted in a median score of 149.8 GFLOPS/s on Mul Kernel. Server manufacturers may vary configurations, yielding different results. Performance may vary based on use of latest drivers and optimizations MI200-58

MI200-59 - Testing Conducted by AMD performance lab as of 4/19/2022 on a single socket optimized 3rd Gen AMD EPYC™ CPU powered server with 1x AMD Instinct™ MI250X OAM (128 GB HBM2e) 560W GPU with AMD Infinity Fabric™ technology resulted in a median score of 184.7 GFLOPS/s on Triad Kernel. Server manufacturers may vary configurations, yielding different results. Performance may vary based on use of latest drivers and optimizations MI200-59

MI200-60 - Testing Conducted by AMD performance lab as of 4/19/2022 on a single socket optimized 3rd Gen AMD EPYC™ CPU powered server with 1x AMD Instinct™ MI250X OAM (128 GB HBM2e) 560W GPU with AMD Infinity Fabric™ technology resulted in a median score of up to 21.7 TFLOPS/s on FMA Kernel. Server manufacturers may vary configurations, yielding different results. Performance may vary based on use of latest drivers and optimizations MI200-60

Nbody Endnotes

MI200-61 - Testing Conducted by AMD performance lab as of 4/19/2022 on a single socket optimized 3rd Gen AMD EPYC™ CPU powered server with 1x AMD Instinct™ MI250X OAM (128 GB HBM2e) 560W GPU with AMD Infinity Fabric™ technology resulted in a median score of 8.7 TFLOPS/s on benchmark mini-nbody-orig. Information on mini-nbody-orig: https://github.com/ROCm-Developer-Tools/HIP-Examples/blob/master/mini-nbody/hip/nbody-orig.cpp. Server manufacturers may vary configurations, yielding different results. Performance may vary based on use of latest drivers and optimizations MI200-61

MI200-62 - Testing Conducted by AMD performance lab as of 4/19/2022 on a single socket optimized 3rd Gen AMD EPYC™ CPU powered server with 1x AMD Instinct™ MI250X OAM (128 GB HBM2e) 560W GPU with AMD Infinity Fabric™ technology resulted in a median score of 9.5 TFLOPS/s on benchmark mini-nbody-block. Information on mini-nbody-block: https://github.com/ROCm-Developer-Tools/HIP-Examples/blob/master/mini-nbody/hip/nbody-block.cpp. . Server manufacturers may vary configurations, yielding different results. Performance may vary based on use of latest drivers and optimizations MI200-62

AMD

Q&A