Presenter: Sam Antao

LUMI Advanced Training

Mar 7th, 2025

[Public]

Contributors

- Cole Ramos
- Xiaomin Lu
- Noah Wolfe
- George Markomanolis
- Austin Ellis
- Gina Sitaraman
- Johanna Potyka
- Quentarius Moore

[Public]

Background – AMD Profilers

ROC-profiler (rocprof)

Hardware Counters
- Raw collection of GPU counters and traces
- Counter collection with user input files
- Counter results printed to a CSV

Traces and timelines
- Trace collection support for
  - CPU copy
  - HIP API
  - HSA API
  - GPU Kernels

Visualisation
- Traces visualized with Perfetto

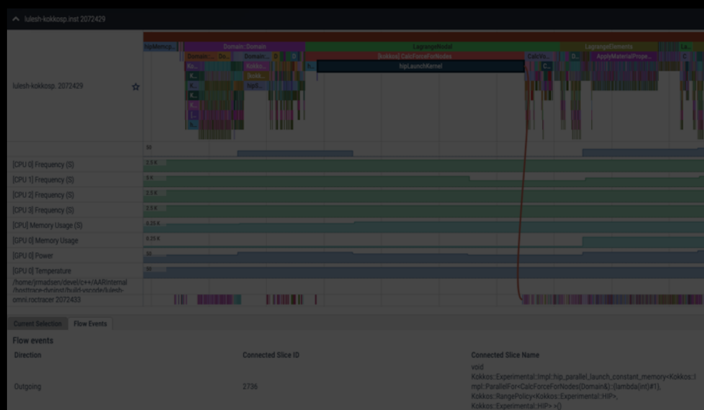
	A	B	C	D	E
1	Name	Calls	TotalDura	AverageN:	Percentage
2	hipMemcpyAsync	99	3.22E+10	3.25E+08	44.14872
3	hipEventSynchronize	330	2.42E+10	73394557	33.225
4	hipMemsetAsync	87	7.76E+09	89232696	10.64953
5	hipHostMalloc	9	5.41E+09	6.01E+08	7.415198
6	hipDeviceSynchronize	28	1.32E+09	47006288	1.805515
7	hipHostFree	17	1.05E+09	61534688	1.435014
8	hipMemcpy	41	8.11E+08	19791876	1.113161
9	hipLaunchKernel	1856	58082083	31294	0.079676
10	hipStreamCreate	2	46380834	23190417	0.063625
11	hipMemset	2	18847246	9423623	0.025854
12	hipStreamDestroy	2	15183338	7591669	0.020828
13	hipFree	38	8269713	217624	0.011344
14	hipEventRecord	330	2520035	7636	0.003457
15	hipMalloc	30	1484804	49493	0.002037
16	__hipPopCallConfigura	1856	229159	123	0.000314
17	__hipPushCallConfigur	1856	224177	120	0.000308
18	hipGetLastError	1494	100458	67	0.000138
19	hipEventCreate	330	76675	232	0.000105
20	hipEventDestroy	330	64671	195	8.87E-05
21	hipGetDevicePropertie	47	51808	1102	7.11E-05
22	hipGetDevice	64	11611	181	1.59E-05
23	hipSetDevice	1	401	401	5.50E-07
24	hipGetDeviceCount	1	220	220	3.02E-07

Omnitrace

Trace collection
- Comprehensive trace collection
  - CPU
  - GPU

Supports
- CPU copy
- HIP API
- HSA API
- GPU Kernels
- OpenMP®
- MPI
- Kokkos
- p-threads
- multi-GPU

Visualisation
- Traces visualized with Perfetto

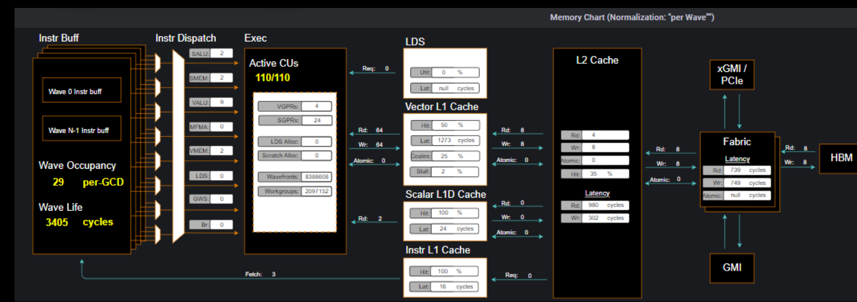


Omniperf

Performance Analysis
- Automated collection of hardware counters
  - Analysis
  - Visualization

Supports
- Speed of Light
- Memory chart
- Rooflines
- Kernel comparison

Visualisation
- With Grafana or standalone GUI



AMD together we advance_

Open-source Client-side Installation is Easy

```
wget https://github.com/ROCm/rocprofiler-compute/releases/download/v2.0.1/omniperf-v2.0.1.tar.gz

tar zxvf  omniperf-v2.0.1.tar.gz

cd omniperf-v2.0.1/
python3 -m pip install -t ${INSTALL_DIR}/python-libs -r requirements.txt
mkdir build
cd build
export PYTHONPATH=$INSTALL_DIR/python-libs:$PYTHONPATH
cmake -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR}/2.0.1 \
        -DPYTHON_DEPS=${INSTALL_DIR}/python-libs \
        -DMOD_INSTALL_PATH=${INSTALL_DIR}/modulefiles ..
make install
export PATH=$INSTALL_DIR/2.0.1/bin:$PATH
```

Dependencies

- ROCm (>=6.0.0), Python™ (>=3.8), CMake (>=3.19)

[Public]

Omniperf modes

Profile	Target application is launched using AMD ROC-profiler		
	Kernels	Dispatches	IP Blocks
Analyze	Profiled data is loaded to omniperf CLI		
	Immediate access to metrics	Lightweight standalone GUI	
Database	Profiled data is imported to Grafana™ database		
	Grafana™ GUI is based on MongoDB	Interact with saved workload database	

Basic command-line syntax:

Profile:

```
$ omniperf profile -n workload_name [profile options]
                   [roofline options] -- <CMD> <ARGS>
```

Analyze:

```
$ omniperf analyze -p
<path/to/workloads/workload_name/mi200/>
```

To use a lightweight standalone GUI with CLI analyzer:

```
$ omniperf analyze -p
<path/to/workloads/workload_name/mi200/> --gui
```

Database:

```
$ omniperf database <interaction type> [connection options]
```

For more information or help use -h/--help/? flags:

```
$ omniperf profile --help
```

For problems, create an issue here: https://github.com/ROCm/rocprofiler-compute/issues
Documentation: https://rocm.github.io/rocprofiler-compute/

AMD together we advance_

`omniperf profile` Arguments to Reduce Profiling Overhead

- **Runtime Filtering**
  `--kernel`, `--ipblocks`, `--dispatch`

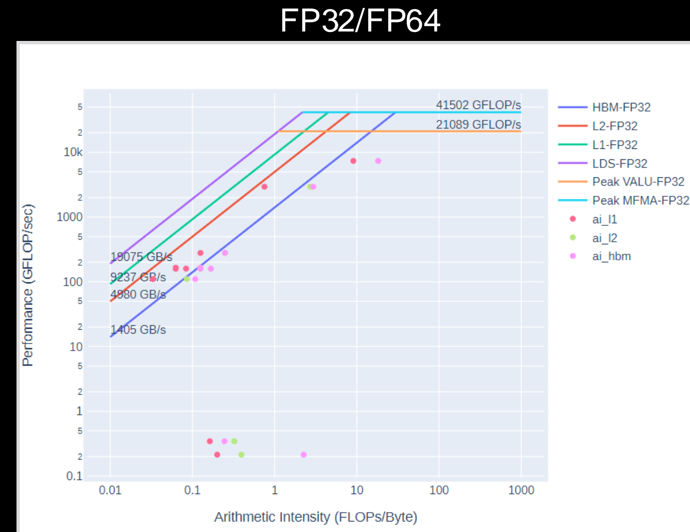
- The `-k` <kernel> flag allows for kernel filtering, which is compatible with the current rocprof utility.
- The `-d` <dispatch> flag allows for dispatch ID filtering, which is compatible with the current rocprof utility.
- The `-b` <ipblocks> allows system profiling on one or more selected IP blocks to speed up the profiling process. One can gradually incorporate more IP blocks, without overwriting performance data acquired on other IP blocks.

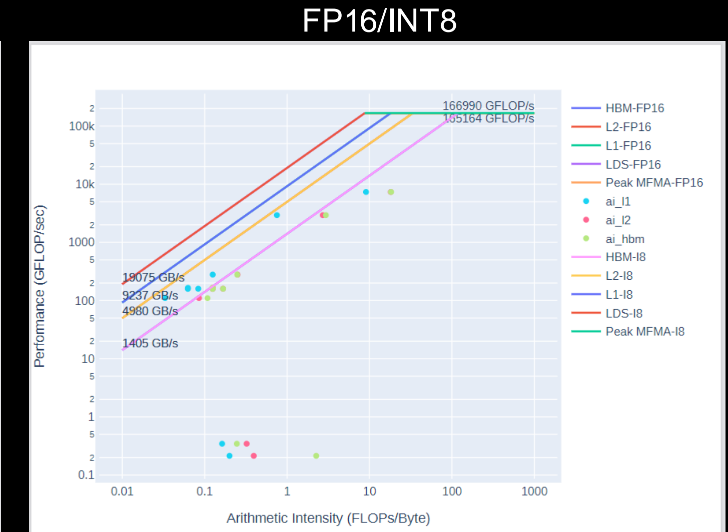
Note: the `-k/--kernel` flag takes a search string in `omniperf profile`

AMD together we advance_

`omniperf profile` Arguments to Generate Roofline PDFs

- Runtime Filtering
  `--kernel`, `--ipblocks`, `--dispatch`
- **Standalone Roofline Analysis**
  `--roof-only, --kernel-names`





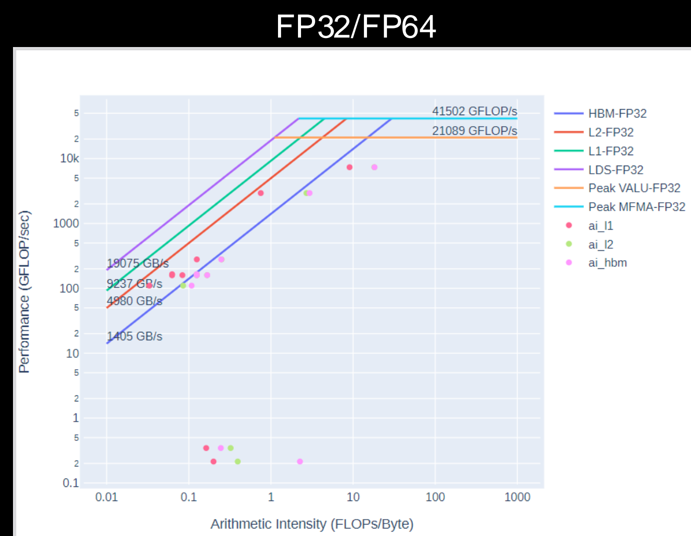
The above plots are saved as PDF output when the --roof-only option is used

[Public]

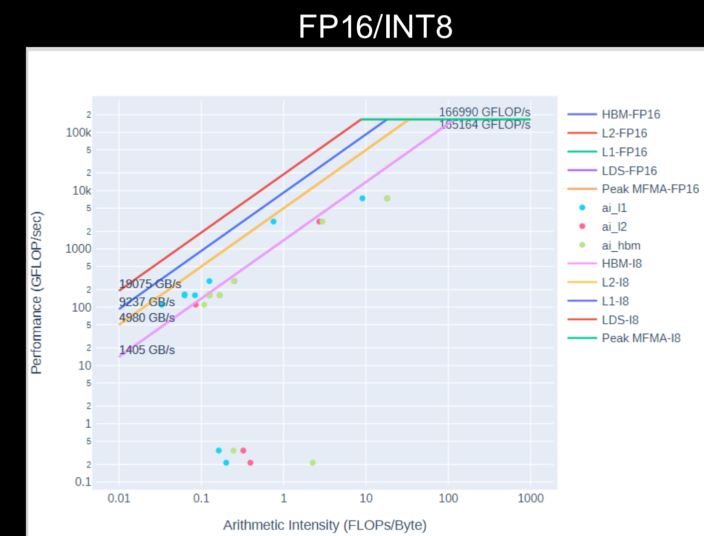
omniperf profile Argument to Turn Off Roofline Benchmarking

- Runtime Filtering
  `--kernel`, `--ipblocks`, `--dispatch`
- Standalone Roofline Analysis
  `--roof-only, --kernel-names`
- **No roofline analysis**
  `--no-roof`

FP32/FP64



FP16/INT8



--no-roof will skip the roofline microbenchmark and omit roofline from output

Omniperf profiling

We use the example sample/vcopy.cpp from the Omniperf installation folder:

```
$ wget https://github.com/rocprofiler-compute/omniperf/raw/main/sample/vcopy.cpp
```

Compile with hipcc:

```
$ hipcc --offload-arch=gfx90a -o vcopy vcopy.cpp
```

Profile with Omniperf:

```
$ omniperf profile -n vcopy_all -- ./vcopy 1048576 256
…
-------------
Profile only
-------------

omniperf ver:  1.0.4
Path:  /pfs/lustrep4/scratch/project_462000075/markoman/omniperf-
1.0.4/build/workloads
Target:  mi200
Command:  ./vcopy 1048576 256
Kernel Selection:  None
Dispatch Selection:  None
IP Blocks: All
```

A new directory will be created called `workloads/vcopy_all`

Note: Omniperf executes the code as many times as required to collect all HW metrics. Use kernel/dispatch filters especially when trying to collect roofline analysis.

AMD together we advance_

[Public]

`omniperf analyze` Arguments to Start With

- **List top kernels or view list of metrics**
  `--list-kernels`, `--list-metrics`

```
colramos@sv-pdp-2:~/GitHub/omniperf-pub$ ./src/omniperf analyze -p workloads/mix_all/mi200/ --list-kernels

--------
Analyze
--------

--------------------------------------------------------------------------------
Detected Kernels
```

	KernelName
0	void benchmark_func<int, 256, 8u, 512u>(int, int*) [clone .kd]
1	void benchmark_func<HIP_vector_type<float, 2u>, 256, 8u, 512u>(HIP_vector_type<float, 2u>, HIP_vecto
2	void benchmark_func<double, 256, 8u, 512u>(double, double*) [clone .kd]
3	void benchmark_func<int, 256, 8u, 256u>(int, int*) [clone .kd]
4	void benchmark_func<__half2, 256, 8u, 512u>(__half2, __half2*) [clone .kd]
5	void benchmark_func<float, 256, 8u, 512u>(float, float*) [clone .kd]
6	void benchmark_func<HIP_vector_type<float, 2u>, 256, 8u, 256u>(HIP_vector_type<float, 2u>, HIP_vecto
7	void benchmark_func<double, 256, 8u, 256u>(double, double*) [clone .kd]
8	void benchmark_func<int, 256, 8u, 128u>(int, int*) [clone .kd]
9	void benchmark_func<__half2, 256, 8u, 256u>(__half2, __half2*) [clone .kd]

```
colramos@sv-pdp-2:~/GitHub/omniperf-pub$ ./src/omniperf analyze -p workloads/mix_all/mi200/ --list-metrics gfx90a
```

	Metric
0	Top Stat
1	System Info
2.1.0	VALU_FLOPs
2.1.1	VALU_IOPs
2.1.2	MFMA_FLOPs_(BF16)
2.1.3	MFMA_FLOPs_(F16)
2.1.4	MFMA_FLOPs_(F32)
2.1.5	MFMA_FLOPs_(F64)
2.1.6	MFMA_IOPs_(Int8)
2.1.7	Active_CUs
2.1.8	SALU_Util
2.1.9	VALU_Util
2.1.10	MFMA_Util
2.1.11	VALU_Active_Threads/Wave
2.1.12	IPC - Issue

Output from the --list-kernel and --list-metric options, showing top kernels and available metrics

[Public]

`omniperf analyze` Arguments to Filter Kernels and GPUs

- List top kernels or view list of metrics
  `--list-kernels`, `--list-metrics`
- **Filter available kernels, dispatches, gpu-ids**
  `--kernel`, `--dispatch`, `--gpu-id`

Note: the `-k/--kernel` flag takes an index given by `--list-kernels in omniperf analyze`, and aggregates stats by kernel name

```
colramos@sv-pdp-2:~/GitHub/omniperf-pub$ ./src/omniperf analyze -p workloads/mix_all/mi200/ --kernel 0

--------
Analyze
--------

--------------------------------------------------------------------------------
0. Top Stat
```

	KernelName	Count	Sum(ns)	Mean(ns)	Median(ns)	Pct	S
0	void benchmark_func<int, 256, 8u, 512u>(int, int*) [clone .kd]	1	3353042.00	3353042.00	3353042.00	7.87	*
1	void benchmark_func<HIP_vector_type<float, 2u>, 256, 8u, 512u>(HIP_vector_type<float, 2u>, HIP_vector_type<float, 2u>...	1	1721239.00	1721239.00	1721239.00	4.04	
2	void benchmark_func<double, 256, 8u, 512u>(double, double*) [clone .kd]	1	1710840.00	1710840.00	1710840.00	4.02	
3	void benchmark_func<int, 256, 8u, 256u>(int, int*) [clone .kd]	1	1693880.00	1693880.00	1693880.00	3.98	
4	void benchmark_func<__half2, 256, 8u, 512u>(__half2, __half2*) [clone .kd]	1	1670521.00	1670521.00	1670521.00	3.92	
5	void benchmark_func<float, 256, 8u, 512u	1	1661402.00	1661402.00	1661402.00	3.90	

Filtered output from the --kernel option isolating kernel at index 0

[Public]

`omniperf analyze` Argument to Only Show Specific Statistics

- List top kernels or view list of metrics
  `--list-kernels`, `--list-metrics`
- Filter available kernels, dispatches, gpu-ids
  `--kernel`, `--dispatch`, `--gpu-id`
- **Filter by metric id(s)**
  `--metric`

```
colramos@sv-pdp-2:~/GitHub/omniperf-pub$ ./src/omniperf analyze -p workloads/mix_all/mi200/ --metric 5

--------
Analyze
--------


--------------------------------------------------------------------------------
0. Top Stat
```

	KernelName	Count	Sum(ns)	Mean(ns)	Median(ns)	Pct
0	void benchmark_func<int, 256, 8u, 512u>(int, int*) [clone .kd]	1	3353042.00	3353042.00	3353042.00	7.87
1	void benchmark_func<HIP_vector_type<float, 2u>, 256, 8u, 512u>(HIP_vector_type<float, 2u>, HIP_vector_type<float, 2u>...	1	1721239.00	1721239.00	1721239.00	4.04
2	void benchmark_func<double, 256, 8u, 512u>(double, double*) [clone .kd]	1	1710840.00	1710840.00	1710840.00	4.02
3	void benchmark_func<int, 256, 8u, 256u>(int, int*) [clone .kd]	1	1693880.00	1693880.00	1693880.00	3.98
4	void benchmark_func<__half2, 256, 8u, 512u>(__half2, __half2*) [clone .kd]	1	1670521.00	1670521.00	1670521.00	3.92
5	void benchmark_func<float, 256, 8u, 512u>(float, float*) [clone .kd]	1	1661402.00	1661402.00	1661402.00	3.90
6	void benchmark_func<HIP_vector_type<float, 2u>, 256, 8u, 256u>(HIP_vector_type<float, 2u>, HIP_vector_type<float, 2u>...	1	881739.00	881739.00	881739.00	2.07
7	void benchmark_func<double, 256, 8u, 256u>(double, double*) [clone .kd]	1	875980.00	875980.00	875980.00	2.06
8	void benchmark_func<int, 256, 8u, 128u>(int, int*) [clone .kd]	1	865100.00	865100.00	865100.00	2.03
9	void benchmark_func<__half2, 256, 8u, 256u>(__half2, __half2*) [clone .kd]	1	855660.00	855660.00	855660.00	2.01

```
--------------------------------------------------------------------------------
5. Command Processor (CPC/CPF)
5.1 Command Processor Fetcher
```

Index	Metric	Avg	Min	Max	Unit
5.1.0	GPU Busy Cycles	416535.02	29084.00	5253061.00	Cycles/kernel
5.1.1	CPF Busy	416535.02	29084.00	5253061.00	Cycles/kernel

Filtering output to isolate data table at index 5

[Public]

`omniperf analyze` Arguments to Change Units and Normalization

- List top kernels or view list of metrics
  `--list-kernels`, `--list-metrics`
- Filter available kernels, dispatches, gpu-ids
  `--kernel`, `--dispatch`, `--gpu-id`
- Filter by metric id(s)
  `--metric`
- **Change normalization unit, time unit, or decimal**
  `--normal-unit`, `--time-unit`, `--decimal`

7.2 Wavefront Runtime Stats

Index	Metric	Avg	Min	Max	Unit
7.2.0	Kernel Time (Nanosec)	255131.78	8480.00	3353042.00	Ns
7.2.1	Kernel Time (Cycles)	416535.02	29084.00	5253061.00	Cycle
7.2.2	Instr/wavefront	557.11	48.00	9300.00	Instr/wavefront
7.2.3	Wave Cycles	18777.13	1848.52	258296.68	Cycles per wave
7.2.4	Dependency Wait Cycles	2819.92	942.73	10169.97	Cycles per wave
7.2.5	Issue Wait Cycles	14105.31	100.13	211703.70	Cycles per wave
7.2.6	Active Cycles	2161.27	180.00	36172.00	Cycles per wave
7.2.7	Wavefront Occupancy	2770.80	185.11	3224.96	Wavefronts

Output showing the default normalization and time unit

`omniperf analyze` Arguments to Compare Workloads

- **Baseline Analysis**

  `--path <workload1_path> --path <workload2_path>`

2. System Speed-of-Light

Index	Metric	Value	Value	Unit	Peak	Peak	PoP	PoP
2.1.0	VALU FLOPs	7492.7178288728755	0.0 (-100.0%)	Gflop	22630.4	22630.4 (0.0%)	33.10908260071795	0.0 (-100.0%)
2.1.1	VALU IOPs	2326.1937250093497	398.91 (-82.85%)	Giop	22630.4	22630.4 (0.0%)	10.279065880449968	1.76 (-82.85%)
2.1.2	MFMA FLOPs (BF16)	0.0	0.0 (nan%)	Gflop	90521.6	90521.6 (0.0%)	0.0	0.0 (nan%)
2.1.3	MFMA FLOPs (F16)	0.0	0.0 (nan%)	Gflop	181043.2	181043.2 (0.0%)	0.0	0.0 (nan%)
2.1.4	MFMA FLOPs (F32)	0.0	0.0 (nan%)	Gflop	45260.8	45260.8 (0.0%)	0.0	0.0 (nan%)
2.1.5	MFMA FLOPs (F64)	0.0	0.0 (nan%)	Gflop	45260.8	45260.8 (0.0%)	0.0	0.0 (nan%)
2.1.6	MFMA IOPs (Int8)	0.0	0.0 (nan%)	Giop	181043.2	181043.2 (0.0%)	0.0	0.0 (nan%)
2.1.7	Active CUs	102	74.0 (-27.45%)	Cus	104	104.0 (0.0%)	98.07692307692308	71.15 (-27.45%)
2.1.8	SALU Util	2.6093901009614555	3.62 (38.57%)	Pct	100	100.0 (0.0%)	2.6093901009614555	3.62 (38.57%)
2.1.9	VALU Util	58.371669678115765	5.17 (-91.15%)	Pct	100	100.0 (0.0%)	58.371669678115765	5.17 (-91.15%)
2.1.10	MFMA Util	0.0	0.0 (nan%)	Pct	100	100.0 (0.0%)	0.0	0.0 (nan%)
2.1.11	VALU Active Threads/Wave	64.0	64.0 (0.0%)	Threads	64	64.0 (0.0%)	100.0	100.0 (0.0%)
2.1.12	IPC - Issue	1.0	1.0 (0.0%)	Instr/cycle	5	5.0 (0.0%)	20.0	20.0 (0.0%)
2.1.13	LDS BW	0.0	0.0 (nan%)	Gb/sec	22630.4	22630.4 (0.0%)	0.0	0.0 (nan%)
2.1.14	LDS Bank Conflict		0.0 (nan%)	Conflicts/access	32	32.0 (0.0%)		0.0 (nan%)
2.1.15	Instr Cache Hit Rate	99.99239800871251	99.91 (-0.08%)	Pct	100	100.0 (0.0%)	99.99239800871251	99.91 (-0.08%)
2.1.16	Instr Cache BW	1687.4579645653916	227.95 (-86.49%)	Gb/s	6092.8	6092.8 (0.0%)	27.695935605393114	3.74 (-86.49%)
2.1.17	Scalar L1D Cache Hit Rate	99.34855885851496	99.82 (0.47%)	Pct	100	100.0 (0.0%)	99.34855885851496	99.82 (0.47%)
2.1.18	Scalar L1D Cache BW	57.584644049561916	227.95 (295.85%)	Gb/s	6092.8	6092.8 (0.0%)	0.9451261168848792	3.74 (295.85%)
2.1.19	Vector L1D Cache Hit Rate	20.35928143712575	50.0 (145.59%)	Pct	100	100.0 (0.0%)	20.35928143712575	50.0 (145.59%)
2.1.20	Vector L1D Cache BW	1699.7181220813884	1823.61 (7.29%)	Gb/s	11315.199999999999	11315.2 (0.0%)	15.021547317602769	16.12 (7.29%)
2.1.21	L2 Cache Hit Rate	3.814906711045504	35.21 (822.95%)	Pct	100	100.0 (0.0%)	3.814906711045504	35.21 (822.95%)
2.1.22	L2-Fabric Read BW	1166.9922392326407	456.37 (-60.89%)	Gb/s	1638.4	1638.4 (0.0%)	71.2275536641016	27.85 (-60.89%)
2.1.23	L2-Fabric Write BW	6.623892610383628	320.42 (4737.3%)	Gb/s	1638.4	1638.4 (0.0%)	0.4042903204579851	19.56 (4737.3%)
2.1.24	L2-Fabric Read Latency	536.7282175696066	282.93 (-47.29%)	Cycles		0.0 (nan%)		0.0 (nan%)
2.1.25	L2-Fabric Write Latency	401.33373490590895	332.3 (-17.2%)	Cycles		0.0 (nan%)		0.0 (nan%)
2.1.26	Wave Occupancy	2770.796874555133	1848.05 (-33.3%)	Wavefronts	3328	3328.0 (0.0%)	83.25711762485373	55.53 (-33.3%)
2.1.27	Instr Fetch BW	405.02278909507197	0.0 (-100.0%)	Gb/s	3046.4	3046.4 (0.0%)	13.295128318509454	0.0 (-100.0%)
2.1.28	Instr Fetch Latency	18.298147264262635	21.37 (16.76%)	Cycles		0.0 (nan%)		0.0 (nan%)

5. Command Processor (CPC/CPF)
5.1 Command Processor Fetcher

Index	Metric	Avg	Avg	Min	Min	Max	Max	Unit

[Public]

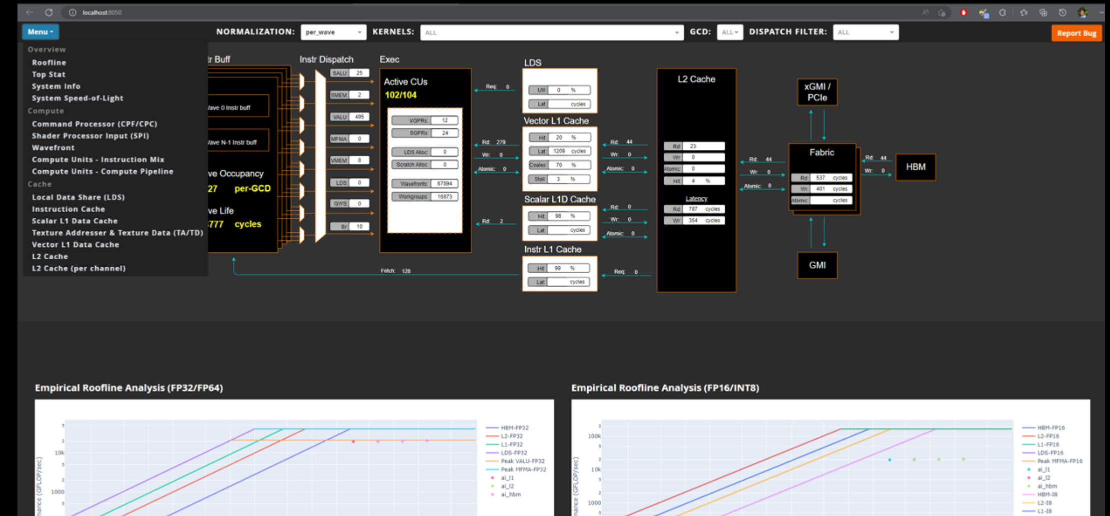
`omniperf analyze` Argument to Launch Standalone GUI

- Baseline Analysis

  `--path <workload1_path> --path <workload2_path>`

- **Launch a standalone HTML page from terminal**

  `--gui <port>`



The above webpage is launched when the --gui option is used

```
colramos@sv-pdp-2:~$ omniperf analyze -p workloads/mix_all/mi200/ --gui

--------
Analyze
--------

Dash is running on http://0.0.0.0:8050/

 * Serving Flask app 'omniperf_analyze.omniperf_analyze'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8050
 * Running on http://10.228.33.182:8050
Press CTRL+C to quit
```

Terminal output from the --gui option with full port forwarding info

Key Insights from Omniperf Analyzer

[Public]

High level Metrics

- System Info

System Info

Metric	Value
Date	Tue Nov 22 18:11:51 2022 (UTC)
App Command	./test_gemm_bf16 0 1 10000
Host Name	0d0b3c44fd0a
Host CPU	AMD EPYC 7402P 24-Core Processor
Host Distro	Ubuntu 20.04.5 LTS
Host Kernel	5.15.0-52-generic
ROCm Version	5.3.0-63
GFX SoC	mi200
GFX ID	gfx90a
Total SEs	8
Total SQCs	56
Total CUs	104
SIMDs/CU	4
Max Wavefronts Occupancy Per CU	32
Max Workgroup Size	1,024
L1Cache per CU (KB)	16
L2Cache (KB)	8,192
L2Cache Channels	32
Sys Clock (Max) - MHz	1,700
Memory Clock (Max) - MHz	1,600
Sys Clock (Cur) - MHz	800
Memory Clock (Cur) - MHz	1,600
HBM Bandwidth - GB/s	1,638.4

Detailed system info for each app is collected by default

AMD together we advance_

High level Metrics

- System Info
- System Speed-of-Light

Speed of Light

Metric	Avg	Unit	Theoretical Max	Pct-of-Peak
VALU FLOPs	2	GFLOP	22,630	0%
VALU IOPs	199	GIOP	22,630	1%
MFMA FLOPs (BF16)	27,403	GFLOP	90,522	30%
MFMA FLOPs (F16)	0	GFLOP	181,043	0%
MFMA FLOPs (F32)	0	GFLOP	45,261	0%
MFMA FLOPs (F64)	0	GFLOP	45,261	0%
MFMA IOPs (Int8)	0	GIOP	181,043	0%
Active CUs	63	CUs	104	61%
SALU Util	1	pct	100	1%
VALU Util	5	pct	100	5%
MFMA Util	14	pct	100	14%
VALU Active Threads/Wave	57	Threads	64	89%
IPC - Issue	1	Instr/cycle	5	14%
LDS BW	1,669	GB/sec	22,630	7%
LDS Bank Conflict	0	Conflicts/access	32	1%
Instr Cache Hit Rate	100	pct	100	100%
Instr Cache BW	211	GB/s	6,093	3%
Scalar L1D Cache Hit Rate	73	pct	100	73%
Scalar L1D Cache BW	3	GB/s	6,093	0%
Vector L1D Cache Hit Rate	23	pct	100	23%
Vector L1D Cache BW	858	GB/s	11,315	8%
L2 Cache Hit Rate	90	pct	100	90%
L2-Fabric Read BW	54	GB/s	1,638	3%
L2-Fabric Write BW	27	GB/s	1,638	2%
L2-Fabric Read Latency	297	Cycles		
L2-Fabric Write Latency	532	Cycles		
Wave Occupancy	246	Wavefronts	3,328	7%
Instr Fetch BW	0	GB/s	3,046	0%
Instr Fetch Latency	53	Cycles		

Dispatch IDs - Current

Dispatch ID	Kernel Name
0	_ZN2ck27kernel_gemm_xdl_cshuffle...
1	_ZN2ck27kernel_gemm_xdl_cshuffle...
2	_ZN2ck27kernel_gemm_xdl_cshuffle...
3	_ZN2ck27kernel_gemm_xdl_cshuffle...
4	_ZN2ck27kernel_gemm_xdl_cshuffle...
5	_ZN2ck27kernel_gemm_xdl_cshuffle...
6	_ZN2ck27kernel_gemm_xdl_cshuffle...
7	_ZN2ck27kernel_gemm_xdl_cshuffle...
8	_ZN2ck27kernel_gemm_xdl_cshuffle...
9	_ZN2ck27kernel_gemm_xdl_cshuffle...
10	_ZN2ck27kernel_gemm_xdl_cshuffle...
11	_ZN2ck27kernel_gemm_xdl_cshuffle...
12	_ZN2ck27kernel_gemm_xdl_cshuffle...
13	_ZN2ck27kernel_gemm_xdl_cshuffle...
14	_ZN2ck27kernel_gemm_xdl_cshuffle...
15	_ZN2ck27kernel_gemm_xdl_cshuffle...
16	void kernel_gemm_xdl_cshuffle_v1<...
17	void kernel_gemm_xdl_cshuffle_v1<...
18	void kernel_gemm_xdl_cshuffle_v1<...
19	void kernel_gemm_xdl_cshuffle_v1<...
20	void kernel_gemm_xdl_cshuffle_v1<...
21	void kernel_gemm_xdl_cshuffle_v1<...
22	void kernel_gemm_xdl_cshuffle_v1<...
23	void kernel_gemm_xdl_cshuffle_v1<...
24	void kernel_gemm_xdl_cshuffle_v1<...
25	void kernel_gemm_xdl_cshuffle_v1<...
26	void kernel_gemm_xdl_cshuffle_v1<...
27	void kernel_gemm_xdl_cshuffle_v1<...
28	void kernel_gemm_xdl_cshuffle_v1<...

Dispatch IDs - Baseline

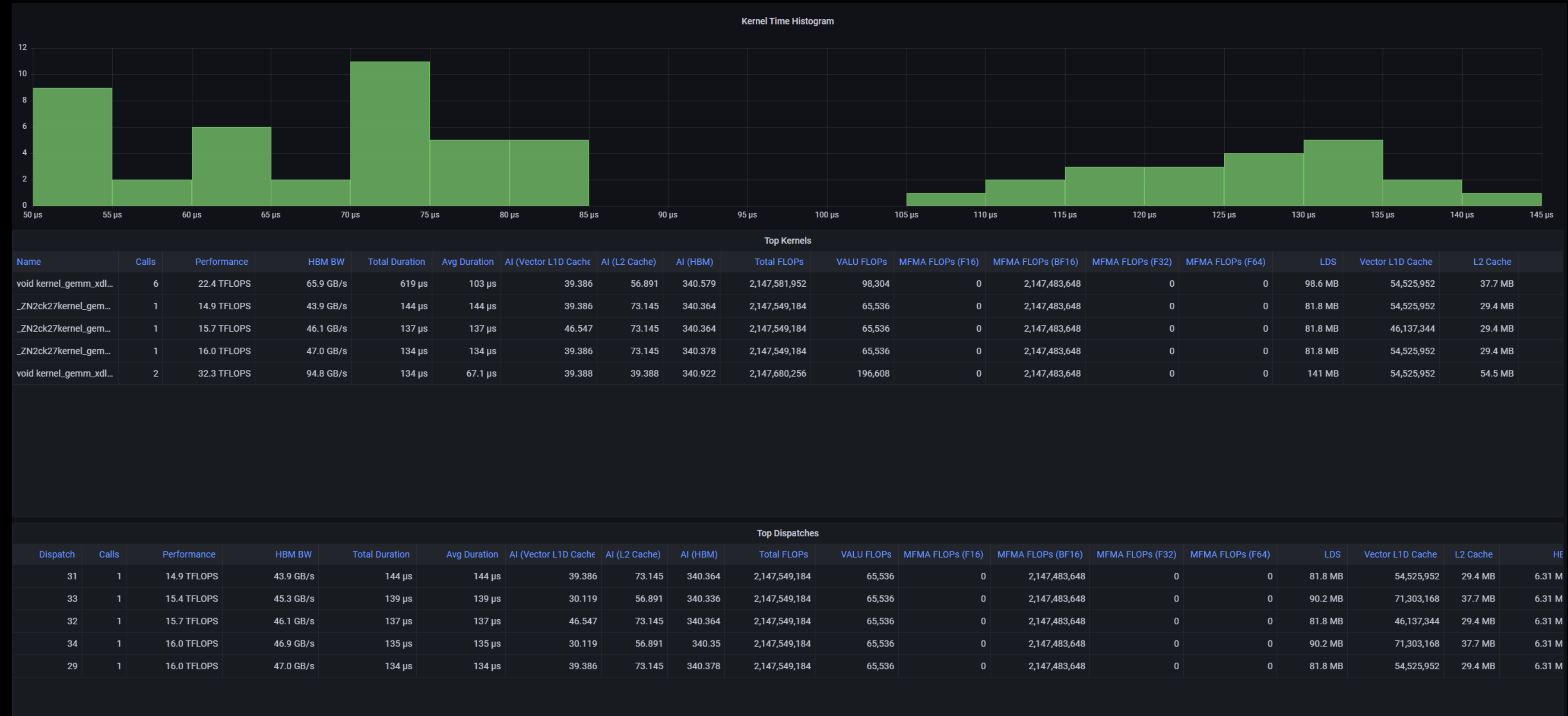
Dispatch ID	Kernel Name
0	axpy(double*, double*, double*, int)

Calls attention to high level performance stats to preview overall application performance

AMD
together we advance_

High level Metrics

- System Info
- System Speed-of-Light
- Kernel Stats



Top Kernels

Name	Calls	Performance	HBM BW	Total Duration	Avg Duration	AI (Vector L1D Cache	AI (L2 Cache)	AI (HBM)	Total FLOPs
void kernel_gemm_xdl...	6	22.4 TFLOPS	65.9 GB/s	619 µs	103 µs	39.386	56.891	340.579	2,147,581,952
_ZN2ck27kernel_gem...	1	14.9 TFLOPS	43.9 GB/s	144 µs	144 µs	39.386	73.145	340.364	2,147,549,184
_ZN2ck27kernel_gem...	1	15.7 TFLOPS	46.1 GB/s	137 µs	137 µs	46.547	73.145	340.364	2,147,549,184
_ZN2ck27kernel_gem...	1	16.0 TFLOPS	47.0 GB/s	134 µs	134 µs	39.386	73.145	340.378	2,147,549,184
void kernel_gemm_xdl...	2	32.3 TFLOPS	94.8 GB/s	134 µs	67.1 µs	39.388	39.388	340.922	2,147,680,256

Name	VALU FLOPs	MFMA FLOPs (F16)	MFMA FLOPs (BF16)	MFMA FLOPs (F32)	MFMA FLOPs (F64)	LDS	Vector L1D Cache	L2 Cache
void kernel_gemm_xdl...	98,304	0	2,147,483,648	0	0	98.6 MB	54,525,952	37.7 MB
_ZN2ck27kernel_gem...	65,536	0	2,147,483,648	0	0	81.8 MB	54,525,952	29.4 MB
_ZN2ck27kernel_gem...	65,536	0	2,147,483,648	0	0	81.8 MB	46,137,344	29.4 MB
_ZN2ck27kernel_gem...	65,536	0	2,147,483,648	0	0	81.8 MB	54,525,952	29.4 MB
void kernel_gemm_xdl...	196,608	0	2,147,483,648	0	0	141 MB	54,525,952	54.5 MB

Top Dispatches

Dispatch	Calls	Performance	HBM BW	Total Duration	Avg Duration	AI (Vector L1D Cache	AI (L2 Cache)	AI (HBM)	Total FLOPs
31	1	14.9 TFLOPS	43.9 GB/s	144 µs	144 µs	39.386	73.145	340.364	2,147,549,184
33	1	15.4 TFLOPS	45.3 GB/s	139 µs	139 µs	30.119	56.891	340.336	2,147,549,184
32	1	15.7 TFLOPS	46.1 GB/s	137 µs	137 µs	46.547	73.145	340.364	2,147,549,184
34	1	16.0 TFLOPS	46.9 GB/s	135 µs	135 µs	30.119	56.891	340.35	2,147,549,184
29	1	16.0 TFLOPS	47.0 GB/s	134 µs	134 µs	39.386	73.145	340.378	2,147,549,184

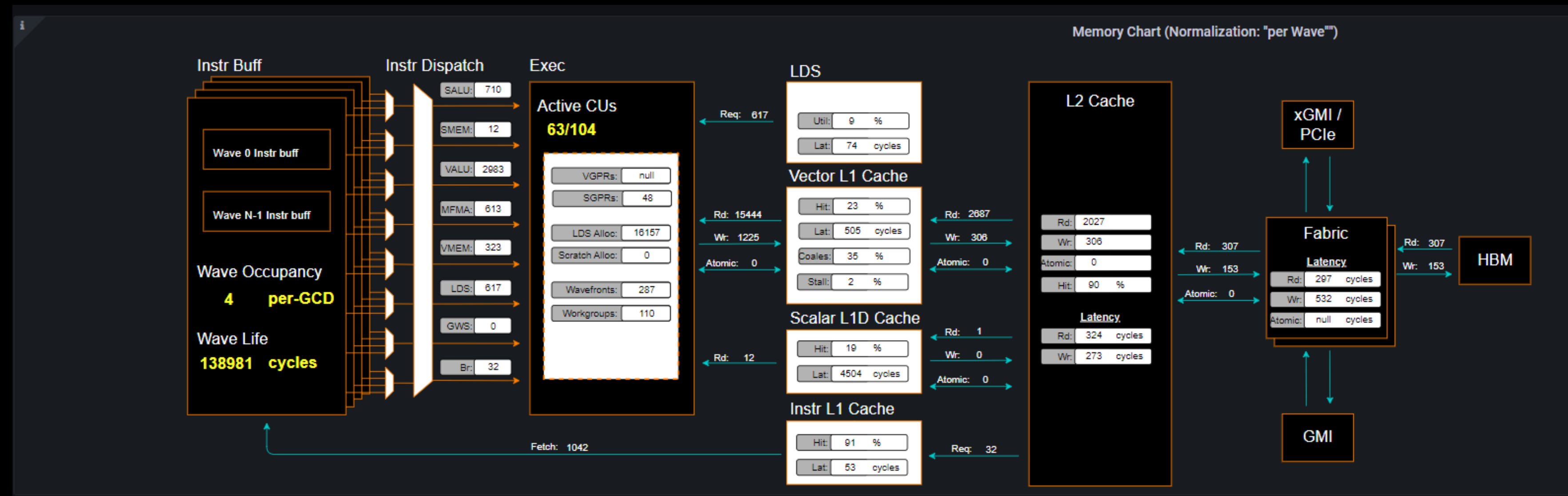
Dispatch	VALU FLOPs	MFMA FLOPs (F16)	MFMA FLOPs (BF16)	MFMA FLOPs (F32)	MFMA FLOPs (F64)	LDS	Vector L1D Cache	L2 Cache	HE
31	65,536	0	2,147,483,648	0	0	81.8 MB	54,525,952	29.4 MB	6.31 M
33	65,536	0	2,147,483,648	0	0	90.2 MB	71,303,168	37.7 MB	6.31 M
32	65,536	0	2,147,483,648	0	0	81.8 MB	46,137,344	29.4 MB	6.31 M
34	65,536	0	2,147,483,648	0	0	90.2 MB	71,303,168	37.7 MB	6.31 M
29	65,536	0	2,147,483,648	0	0	81.8 MB	54,525,952	29.4 MB	6.31 M

Preview performance of top N kernels and individual kernel invocations (dispatches)

[Public]

High level Metrics

- System Info
- System Speed-of-Light
- Kernel Stats
- Memory Chart Analysis

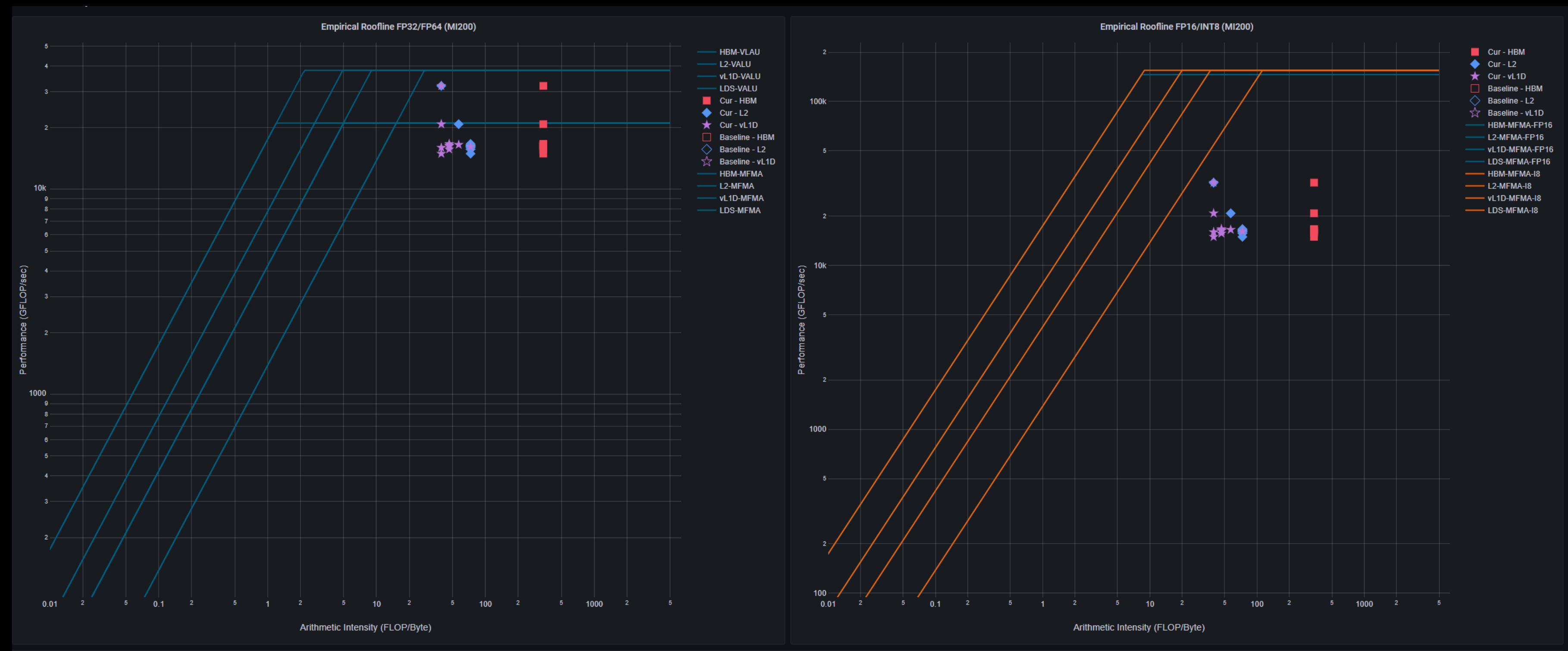


Illustrate data movement and performance on key components of target architecture

AMD
together we advance_

High level Metrics

- System Info
- System Speed-of-Light
- Kernel Stats
- Memory Chart Analysis
- Roofline Analysis

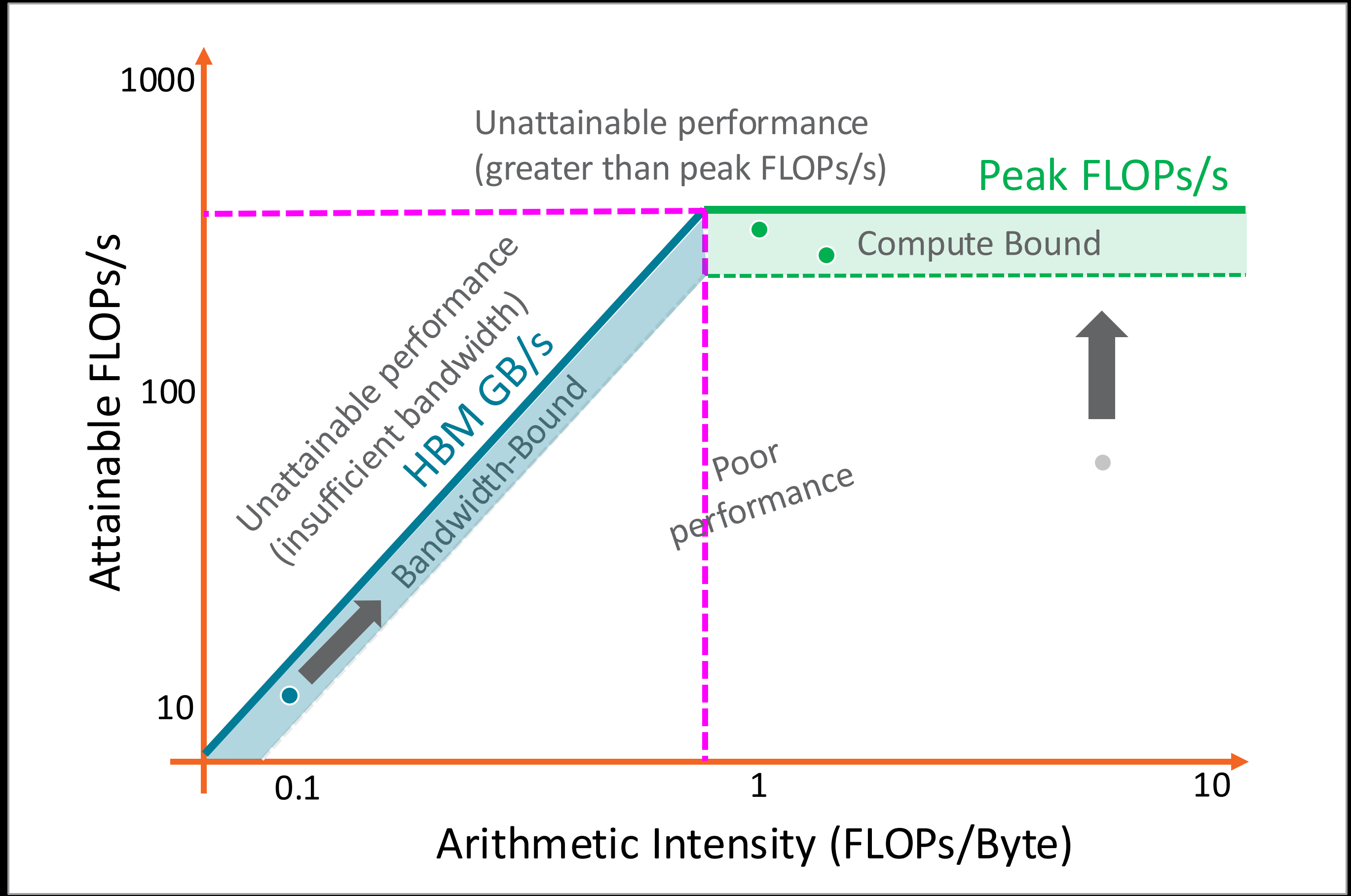


Derived Empirical Roofline analysis broken into two major instruction mixes. Showing application performance relative to meas ured maximum achievable performance

AMD
together we advance_

Background – What is roofline?

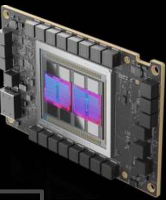
- Attainable FLOPs/s =
  - $min\begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
- Five Performance Regions:
  - Unattainable Compute
  - Unattainable Bandwidth
  - Compute Bound
  - Bandwidth Bound
  - Poor Performance



AMD
together we advance_

[Public]

Overview - AMD Instinct™ MI200 Architecture



Graphics Compute Die (GCD)

Compute Unit				
Local Data Share (LDS)				
Scalar Unit	SIMD0	SIMD1	SIMD2	SIMD3
SGPR	VGPR	VGPR	VGPR	VGPR
Vector L1 Data Cache (vL1D)				

Compute Unit				
Local Data Share (LDS)				
Scalar Unit	SIMD0	SIMD1	SIMD2	SIMD3
SGPR	VGPR	VGPR	VGPR	VGPR
Vector L1 Data Cache (vL1D)				

… …

Compute Unit				
Local Data Share (LDS)				
Scalar Unit	SIMD0	SIMD1	SIMD2	SIMD3
SGPR	VGPR	VGPR	VGPR	VGPR
Vector L1 Data Cache (vL1D)				

L2 Cache (L2)

(Peer GCD) ⇔ Data Fabric ⇔ Remote Socket (CPU, GPU)

Memory Controller

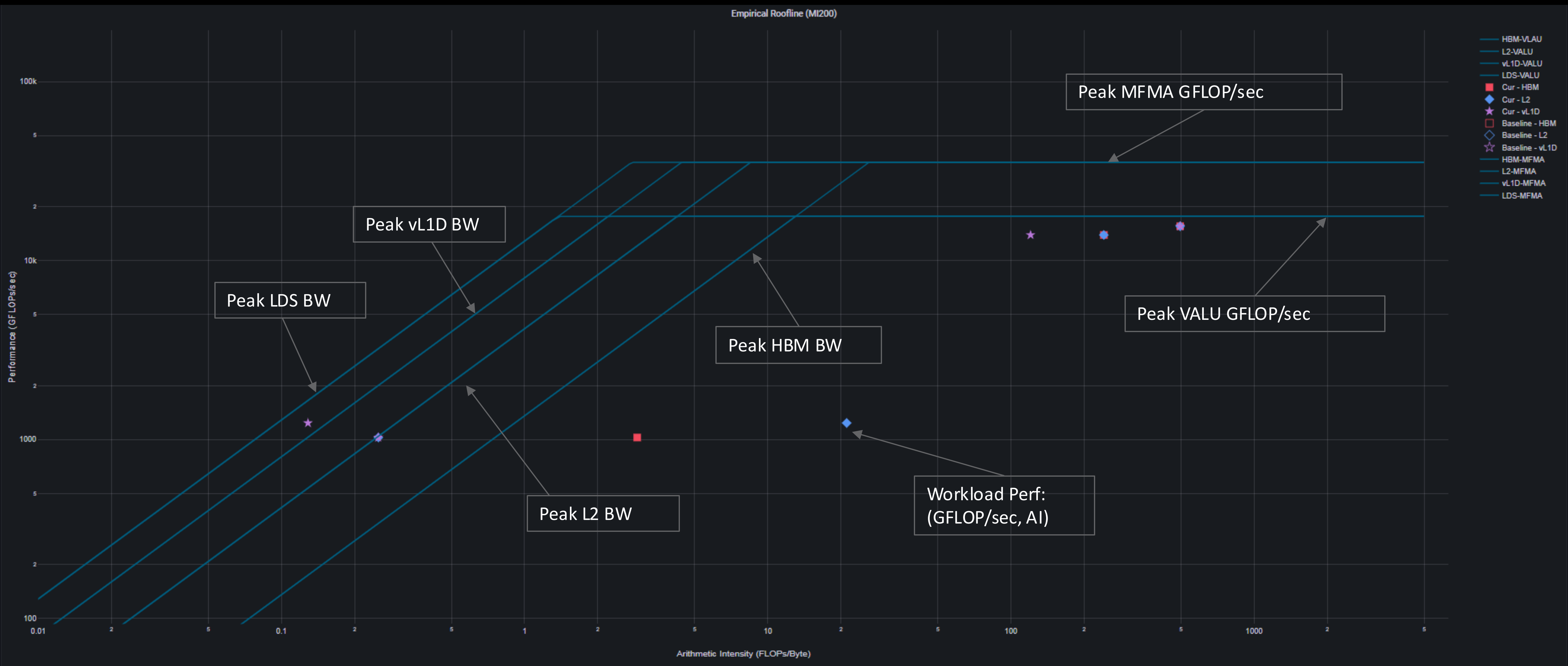
(GCD1) HBM Memory

(GCD2) HBM Memory

AMD
together we advance_

[Public]

Empirical Hierarchical Roofline on MI200 - Overview



AMD
together we advance_

[Public]

Empirical Hierarchical Roofline on MI200 – Roofline Benchmarking

- 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

[Public]

Low level Metrics

Section Title	Comments
Command Processor (CPC/CPF)	Packet processor data
Shader Processor Input (SPI)	Connecting packet processor and CUs
Wavefront Stats	Kernel launch stats
Compute Unit – Instruction Mix	Breakdown of instructions issued
Compute Unit – Compute Pipeline	
Texture Addressor & Texture Data (TA/TD)	Fetch & receive reqs for lookup in vL1D RAM
Local Data Share (LDS)	Cache level stats
Instruction Cache	
Scalar L1 Data Cache	
Vector L1 Data Cache	
L2 Cache	
L2 Cache (per channel)	

Tips for Long-Running Benchmarks

- Filtering by kernel name and metrics during omniperf profile will cut down on profiling time
  - omniperf profile –k “<kernel name>” filters a single kernel name
  - omniperf profile –k “<kernel1>” “<kernel2>” filters two kernel names.
  - Note: surrounding a kernel name in quotes allows spaces to appear in your kernel search string
  - Also Note: omniperf applies the wildcard automatically, so only a unique substring of kernel names are required
  - Finally, Note: omniperf **analyze** –k does not take a kernel name, but an index
    - Index to kernel name mapping is given by omniperf analyze --list-stats

- If you know which metrics you want to collect ahead of time, you can cut down how many rocprof runs are required
  - omniperf profile --block SQ SQC –n <workload name> -- ./benchmark.sh (SQC – Shader Sequencer Controller)
  - omniperf profile --help displays all block strings you can filter by

  - Omniperf documentation: https://rocm.github.io/omniperf/performance_model.html Goes over some of the meaning behind lower-level hardware units and metrics.

AMD
together we advance_

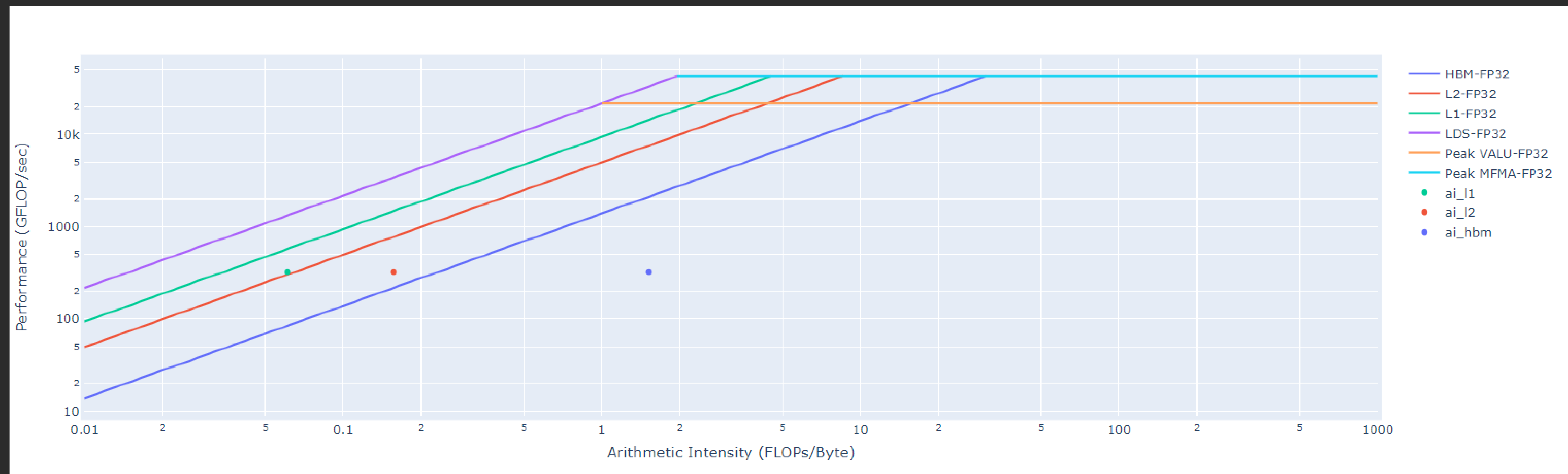
Example – DAXPY with a loop in the kernel

DAXPY – with a loop in the kernel

```
#include <hip/hip_runtime.h>

__constant__ double a = 1.0f;

__global__
void daxpy (int n, double const* x, int incx, double* y, int incy)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < n)
       for(int ll=0;ll<20;ll++) {
        y[i] = a*x[i] + y[i];
       }
}

int main()
{
    int n = 1<<24;
    std::size_t size = sizeof(double)*n;

    double* d_x;
    double *d_y;
    hipMalloc(&d_x, size);
    hipMalloc(&d_y, size);

    int num_groups = (n+255)/256;
    int group_size = 256;
    daxpy<<<num_groups, group_size>>>(n, d_x, 1, d_y, 1);
    hipDeviceSynchronize();
```

[Public]

Roofline



- Performance: almost 330 GFLOPs

AMD
together we advance_

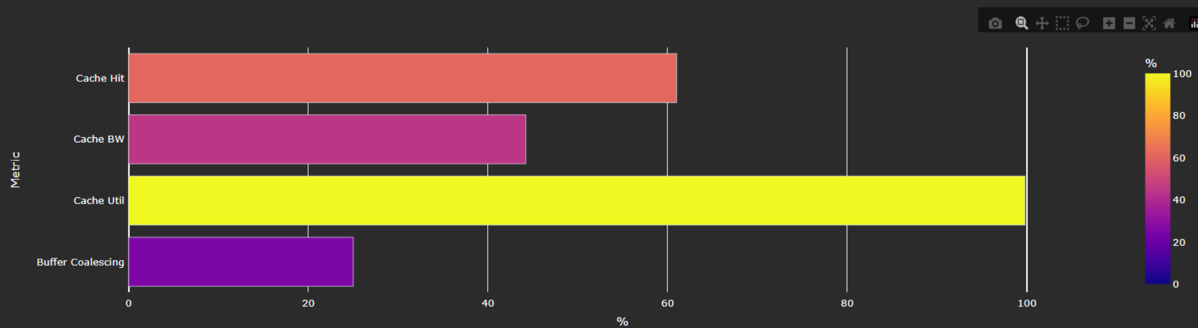
[Public]

Kernel execution time and L1D Cache Accesses

KernelName	Count	Sum(ns)	Mean(ns)	Median(ns)	Pct
daxpy(int, double const*, int, double*, int) [clone .kd]	1.00	2024491.00	2024491.00	2024491.00	100.00

16. Vector L1 Data Cache

16.1 Speed-of-Light



16.2 L1D Cache Stalls

Metric	Mean	Min	Max	unit
Stalled on L2 Data	73.69	73.69	73.69	Pct
Stalled on L2 Req	19.47	19.47	19.47	Pct
Tag RAM Stall (Read)	0.00	0.00	0.00	Pct
Tag RAM Stall (Write)	0.00	0.00	0.00	Pct
Tag RAM Stall (Atomic)	0.00	0.00	0.00	Pct

16.3 L1D Cache Accesses

Metric	Avg	Min	Max	Unit
Total Req	2624.00	2624.00	2624.00	Req per wave
Read Req	1344.00	1344.00	1344.00	Req per wave
Write Req	1280.00	1280.00	1280.00	Req per wave
Atomic Req	0.00	0.00	0.00	Req per wave
Cache BW	5291.66	5291.66	5291.66	Gb/s
Cache Accesses	656.00	656.00	656.00	Req per wave
Cache Hits	400.16	400.16	400.16	Req per wave
Cache Hit Rate	61.00	61.00	61.00	Pct

AMD together we advance_

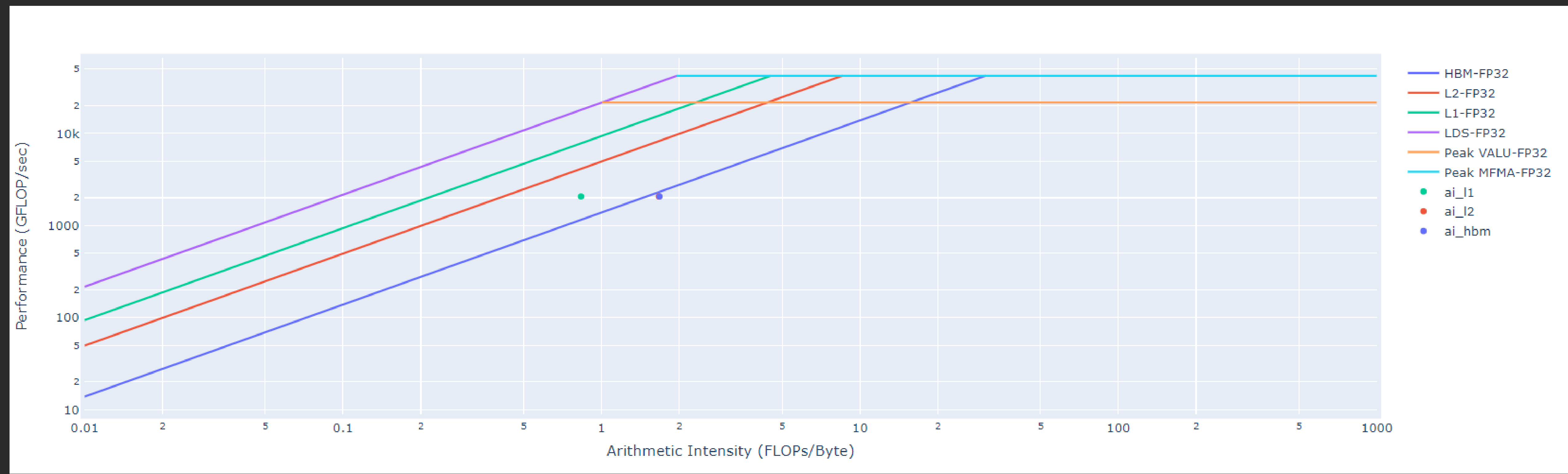
DAXPY – with a loop in the kernel - Optimized

```
#include <hip/hip_runtime.h>

__constant__ double a = 1.0f;

__global__
void daxpy (int n, double const* __restrict__ x, int incx, double* __restrict__ y, int incy)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < n)
       for(int ll=0;ll<20;ll++) {
        y[i] = a*x[i] + y[i];
       }
}

int main()
{
    int n = 1<<24;
    std::size_t size = sizeof(double)*n;

    double* d_x;
    double *d_y;
    hipMalloc(&d_x, size);
    hipMalloc(&d_y, size);

    int num_groups = (n+255)/256;
    int group_size = 256;
    daxpy<<<num_groups, group_size>>>(n, d_x, 1, d_y, 1);
    hipDeviceSynchronize();
```


AMD together we advance_

[Public]

Roofline - Optimized



- Performance: almost 2 TFLOPs

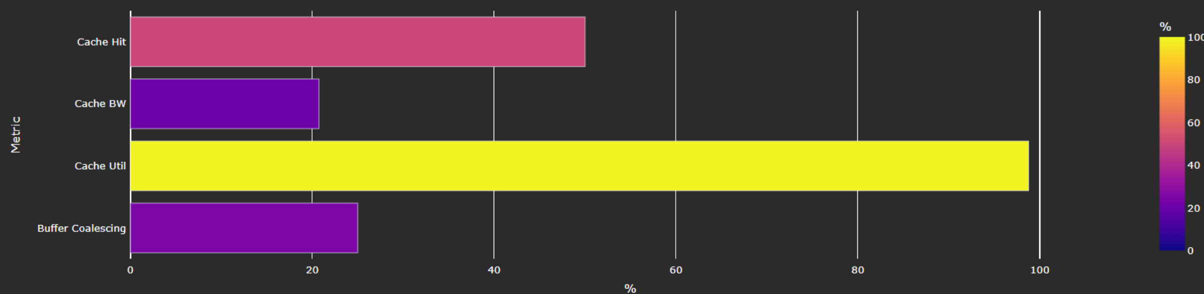
AMD
together we advance_

Kernel execution time and L1D Cache Accesses - Optimized

KernelName	Count	Sum(ns)	Mean(ns)	Median(ns)	Pct
daxpy(int, double const*, int, double*, int) [clone .kd]	1.00	323522.00	323522.00	323522.00	100.00

6.2 times faster!

16.1 Speed-of-Light



16.2 L1D Cache Stalls

Metric	Mean	Min	Max	unit
Stalled on L2 Data	79.08	79.08	79.08	Pct
Stalled on L2 Req	15.17	15.17	15.17	Pct
Tag RAM Stall (Read)	0.00	0.00	0.00	Pct
Tag RAM Stall (Write)	0.00	0.00	0.00	Pct
Tag RAM Stall (Atomic)	0.00	0.00	0.00	Pct

16.3 L1D Cache Accesses

Metric	Avg	Min	Max	Unit
Total Req	192.00	192.00	192.00	Req per wave
Read Req	128.00	128.00	128.00	Req per wave
Write Req	64.00	64.00	64.00	Req per wave
Atomic Req	0.00	0.00	0.00	Req per wave
Cache BW	2480.60	2480.60	2480.60	Gb/s
Cache Accesses	48.00	48.00	48.00	Req per wave
Cache Hits	24.00	24.00	24.00	Req per wave
Cache Hit Rate	50.00	50.00	50.00	Pct
Invalidate	0.00	0.00	0.00	Req per wave

[Public]

Hands-on exercises

https://hackmd.io/@sfantao/lumi-training-sto-2025#Omniperf

We welcome you to explore our HPC Training Examples repo:

https://github.com/amd/HPCTrainingExamples

A table of contents for the READMEs if available at the top-level **README** in the repo

Relevant exercises for this presentation located in **Omniperf** directory.

Link to instructions on how to run the tests: **Omniperf/README.md and subdirectories**

AMD together we advance_

[Public]

Questions?

DISCLAIMERS AND ATTRIBUTIONS

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Git and the Git logo are either registered trademarks or trademarks of Software Freedom Conservancy, Inc., corporate home of the Git Project, in the United States and/or other countries

© 2025 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo, Radeon™, Instinct™, EPYC, Infinity Fabric, ROCm™, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

AMD