Introduction to AMD ROCm™ Ecosystem

Suyash Tandon, Justin Chang, Julio Maia, Noel Chalmers, Paul T. Bauman, Nicholas Curtis, Nicholas Malaya, Alessandro Fanfarillo, Jose Noudohouenou, Chip Freitag, Damon McDougall, Noah Wolfe, Jakub Kurzak, Samuel Antao, George Markomanolis

Introduction to LUMI-G hardware and programming environment
11/01/2023

AMD
together we advance_

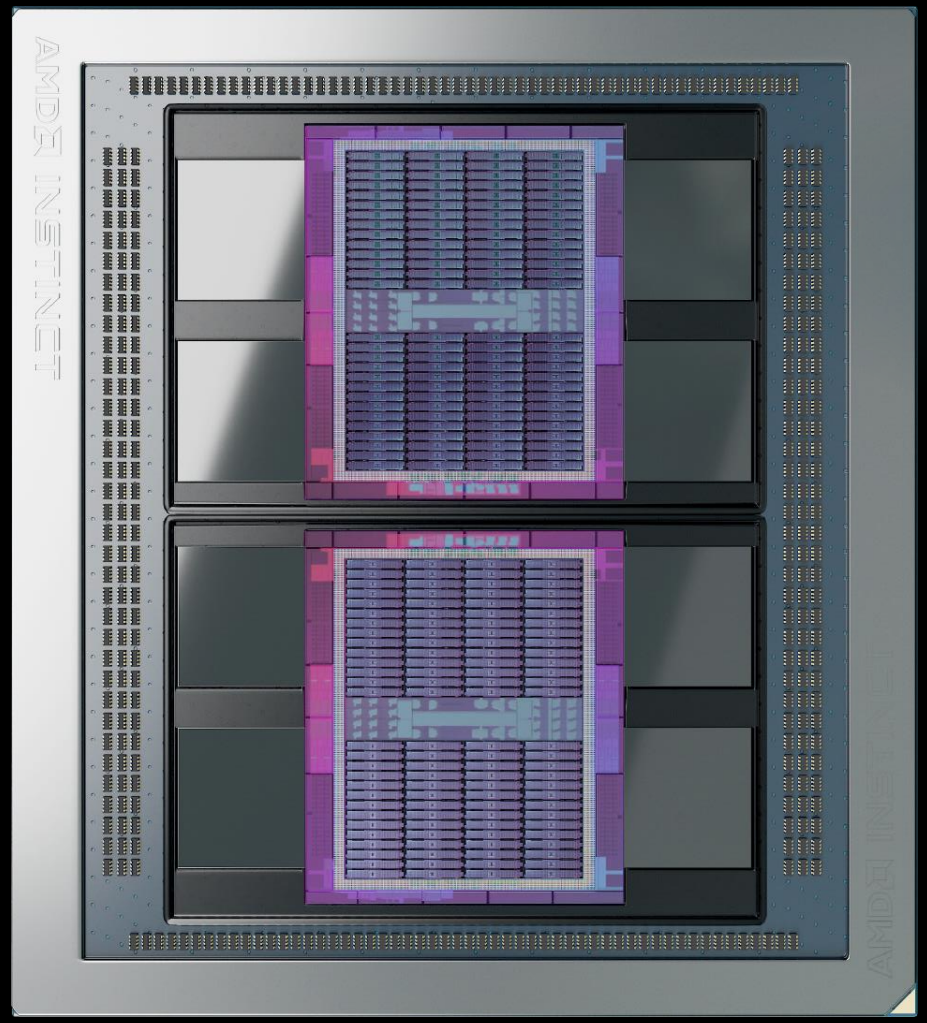
Agenda

1. Introduction to the Architecture
2. Introduction to ROCm and HIP
3. Porting Applications to HIP
4. ROCm libraries
5. Profiling
6. Debugging

Introduction/Expectations

- This talk is a high level of our ecosystem presentation
- We avoid deep dive topics as the audience is from various domains and levels
- We plan to give more extensive introduction and advanced training
- We hope that you can identify topics that you would like further training
- Contact the LUMI User Support Team for further training requests

Introduction to the Architecture

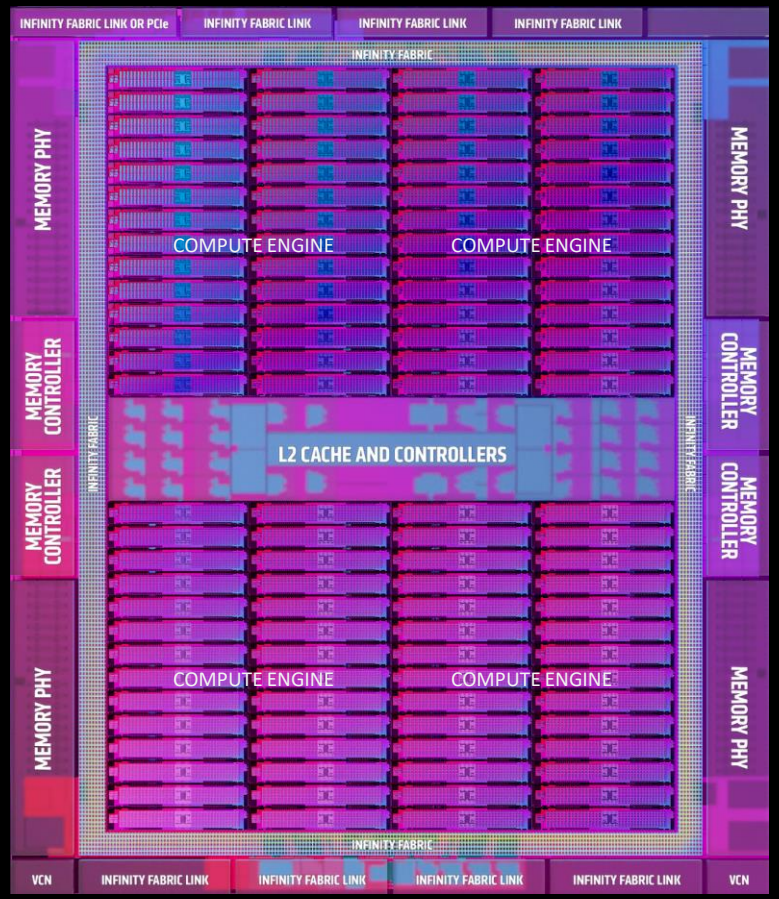


AMD INSTINCT™ MI250X

WORLD'S MOST ADVANCED DATA CENTER ACCELERATOR

58B Transistors in 6nm	**220** Compute Units
880 2nd Gen Matrix Cores	**128** GB HBM2E @ 3.2 TB/s

https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf



2ND GENERATION CDNA ARCHITECTURE

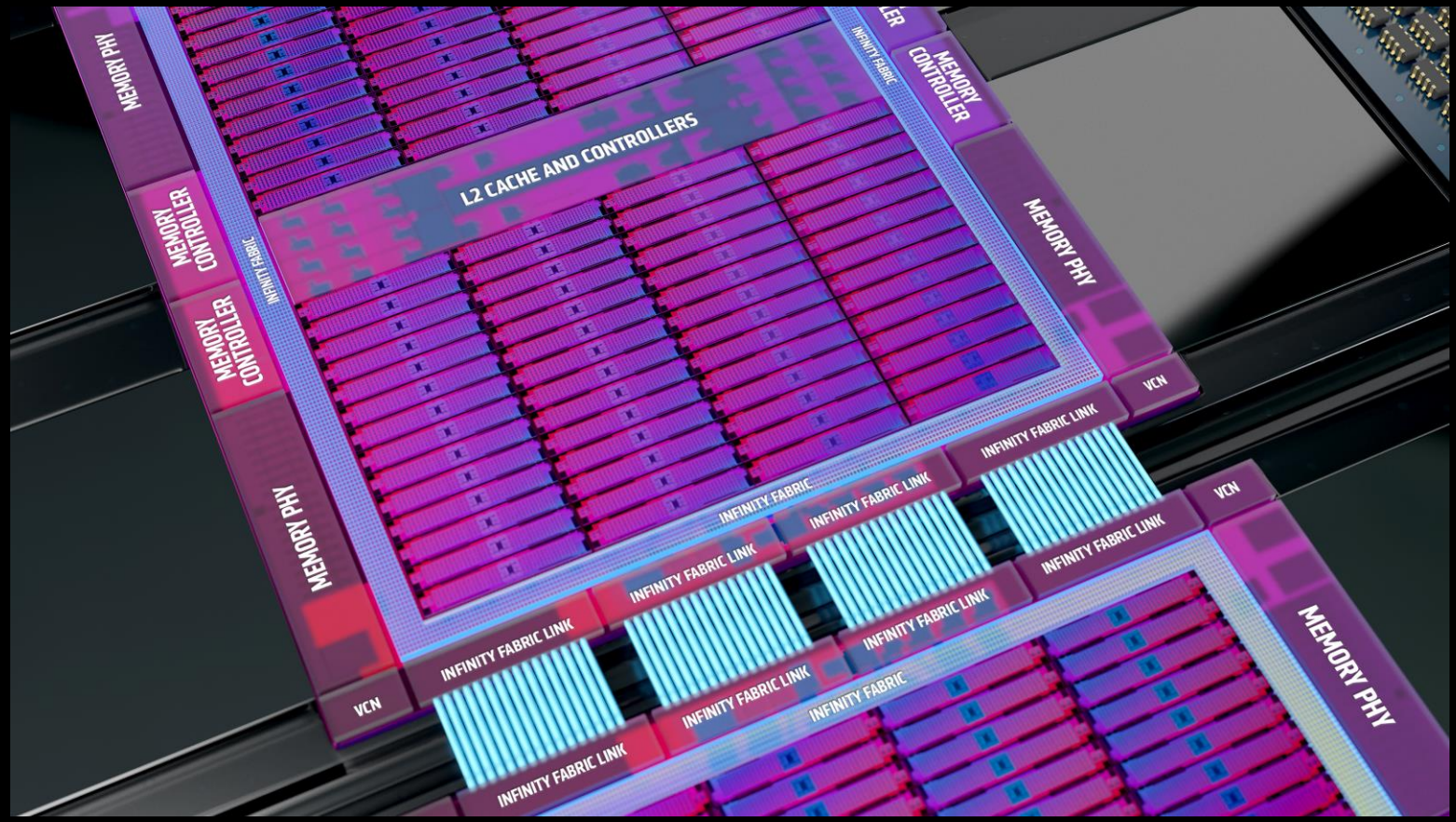
TAILORED-BUILT FOR HPC & AI

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

MULTI-CHIP DESIGN

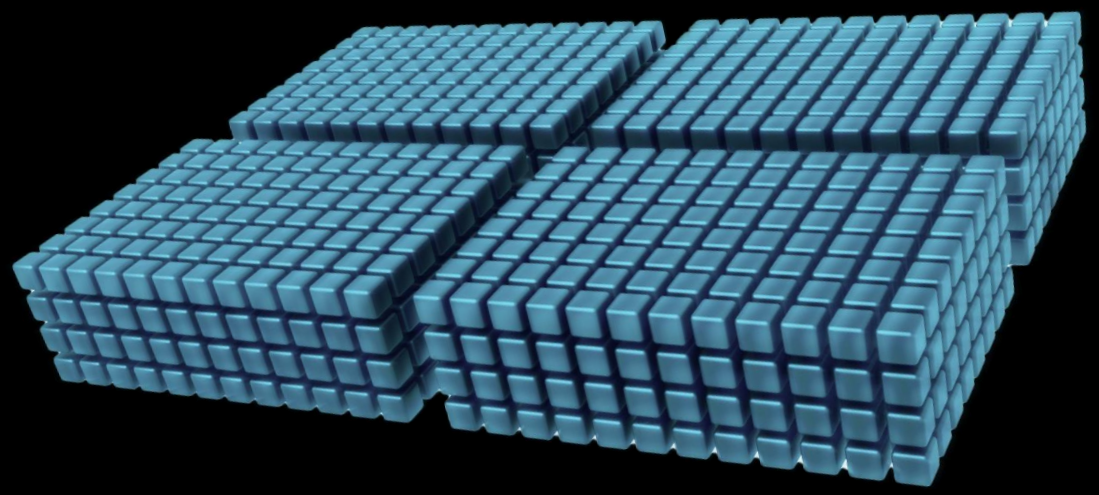
TWO GPU DIES IN PACKAGE TO MAXIMIZE COMPUTE & DATA THROUGHPUT

- INFINITY FABRIC FOR CROSS-DIE CONNECTIVITY
- 4 LINKS RUNNING AT 25GBPS
- 400GB/S OF BI-DIRECTIONALBANDWIDTH



2nd GENERATION MATRIX CORES

OPTIMIZED COMPUTE UNITS FOR SCIENTIFIC COMPUTING

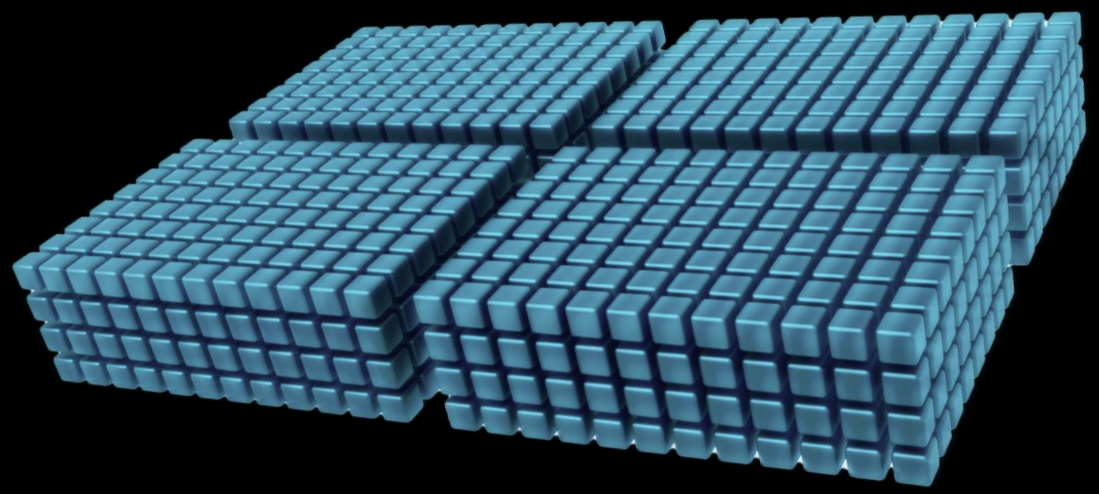


DOUBLE PRECISON (FP64)
MATRIX CORE THROUGHPUT
REPRESENTATION

MI100 MATRIX CORES OPS/CLOCK/COMPUTE UNIT	MI250X MATRIX CORES OPS/CLOCK/COMPUTE UNIT
No FP64 Matrix Core	256 FP64
256 FP32	256 FP32
1024 FP16	1024 FP16
512 BF16	1024 BF16
512 INT8	1024 INT8

2nd GENERATION MATRIX CORES

OPTIMIZED COMPUTE UNITS FOR SCIENTIFIC COMPUTING



- Current support for using MFMA instructions:
  - AMD libraries: rocBLAS
  - Intrinsics
  - Inline assembly
- Not currently supported:
  - Libraries of device functions, utilizing the matrix operations, that can be called from kernels
  - Abstraction frameworks (Kokkos, Raja, OCCA)
    - These would have to use one of the other mechanisms internally

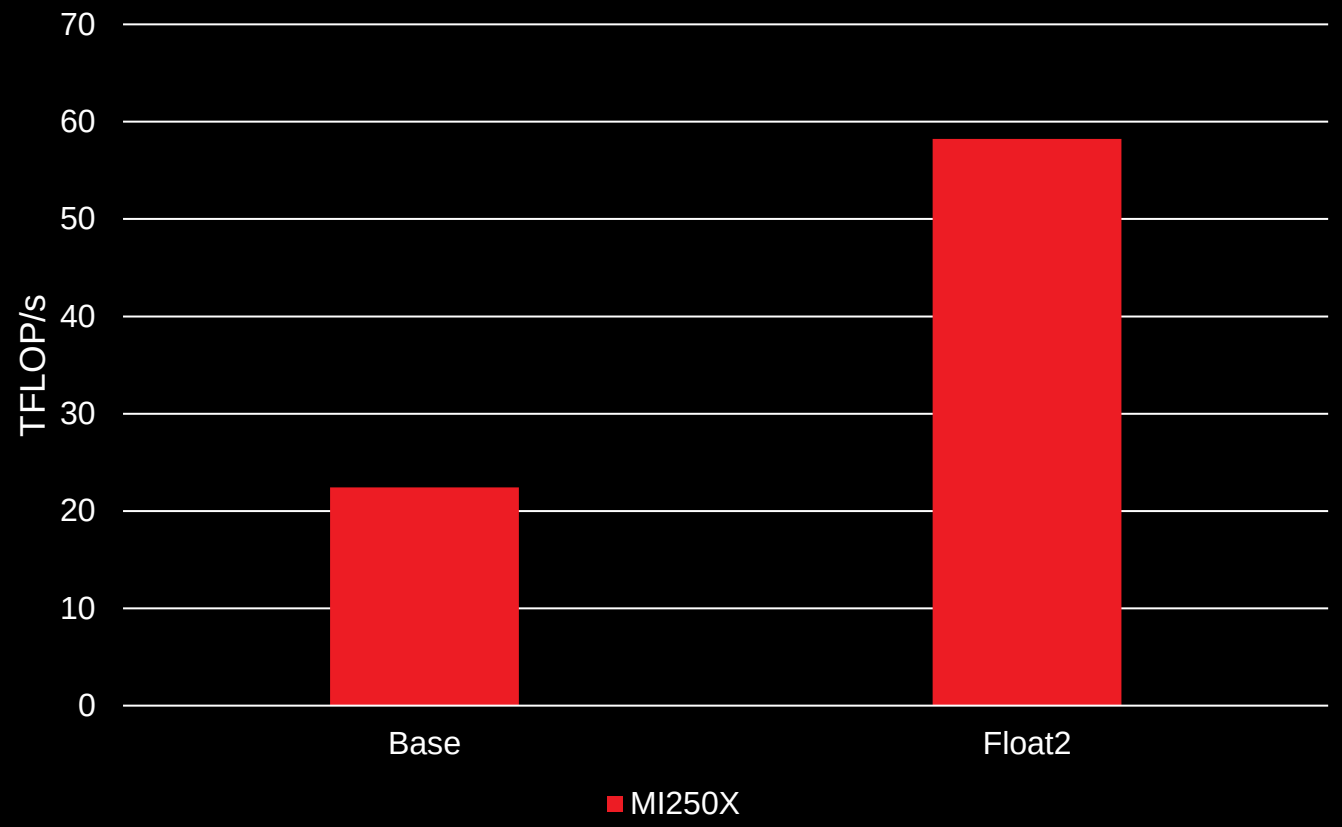
NEW IN AMD INSTINCT MI250X

PACKED FP32

FP64 PATH USED TO EXECUTE TWO COMPONENT VECTOR INSTRUCTIONS ON FP32

DOUBLES FP32 THROUGHPUT PER CLOCK PER COMPUTE UNIT

pk_FMA, pk_ADD, pk_MUL, pk_MOV operations



https://www.amd.com/en/technologies/infinity-hub/mini-hacc

From AMD MI100 to AMD MI250X

MI100

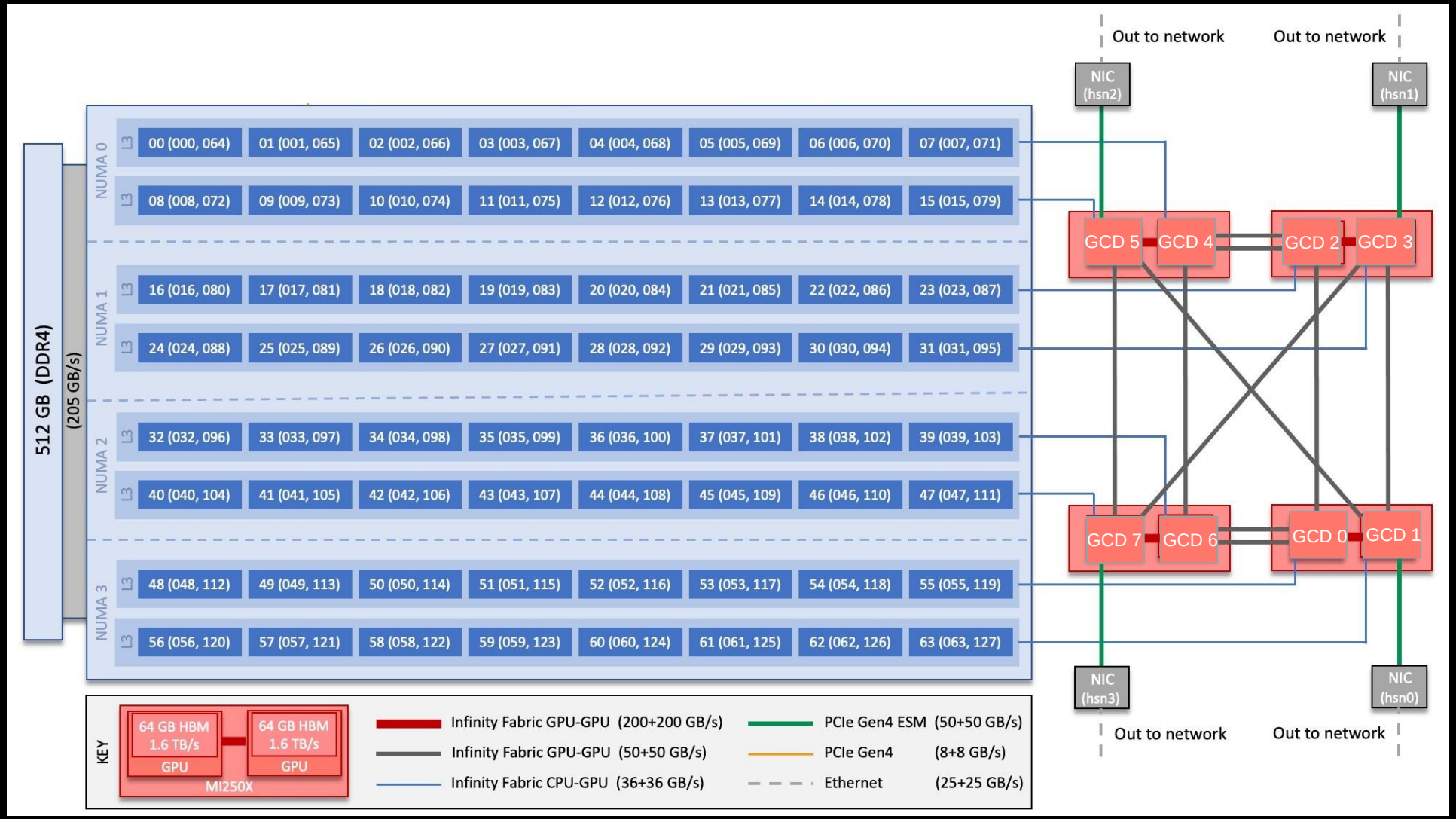
- One graphic compute die (GCD)
- 32GB of HBM2 memory
- 11.5 TFLOPS peak performance per GCD
- 1.2 TB/s peak memory bandwidth per GCD
- 120 CU per GPU
- The interconnection is attached on the CPU

AMD CDNA™ 2 white paper:
https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

MI250X

- Two graphic compute dies (GCDs)
- 64GB of HBM2e memory per GCD (total 128GB)
- 26.5 TFLOPS peak performance per GCD
- 1.6 TB/s peak memory bandwidth per GCD
- 110 CU per GCD, totally 220 CU per GPU
- The interconnection is attached on the GPU (not on the CPU)
- Both GCDs are interconnected with 200 GB/s per direction
- 128 single precision FMA operations per cycle
- AMD CDNA 2 Matrix Core supports double-precision data
- Memory coherency

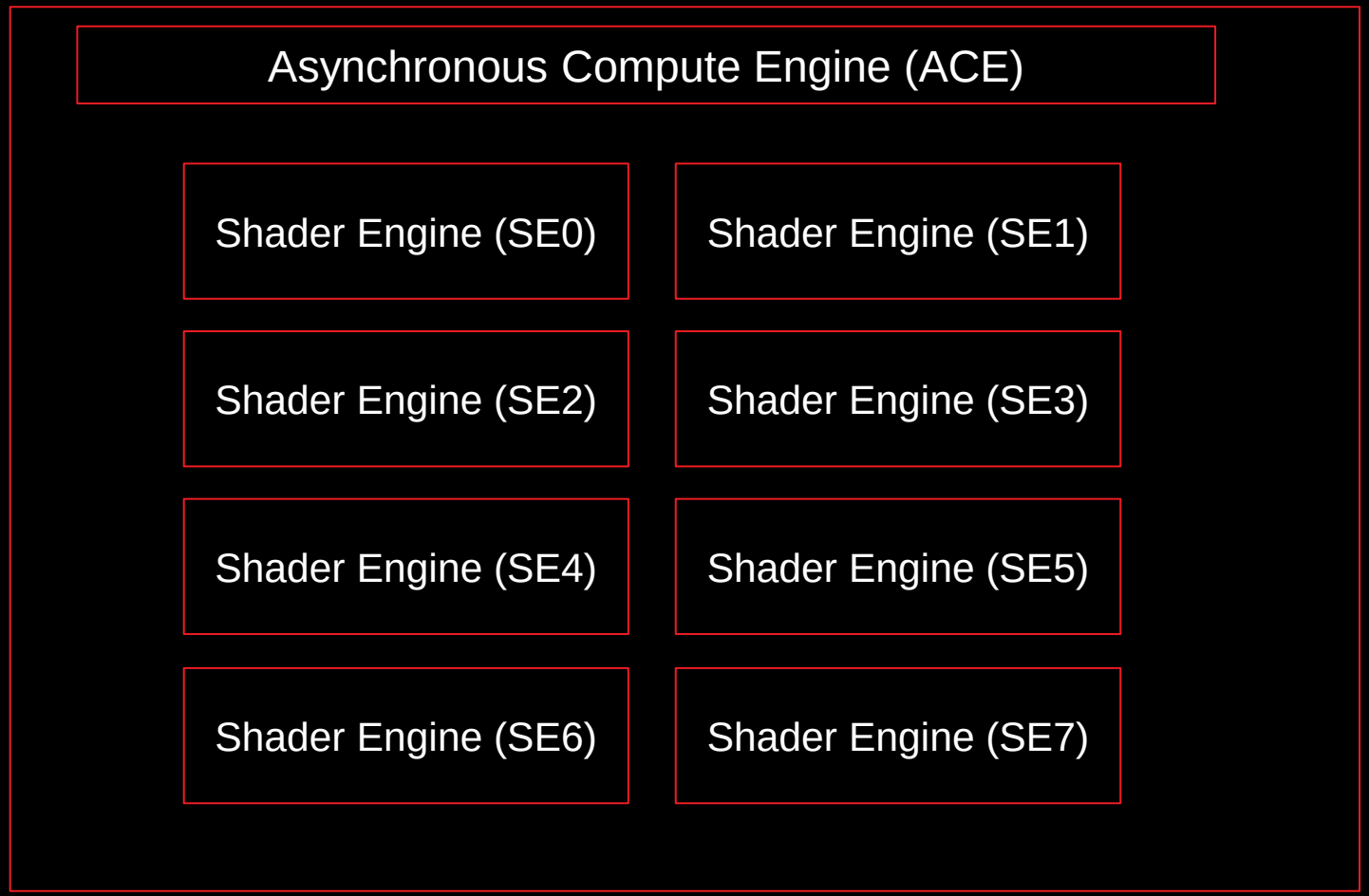
LUMI – MI250X



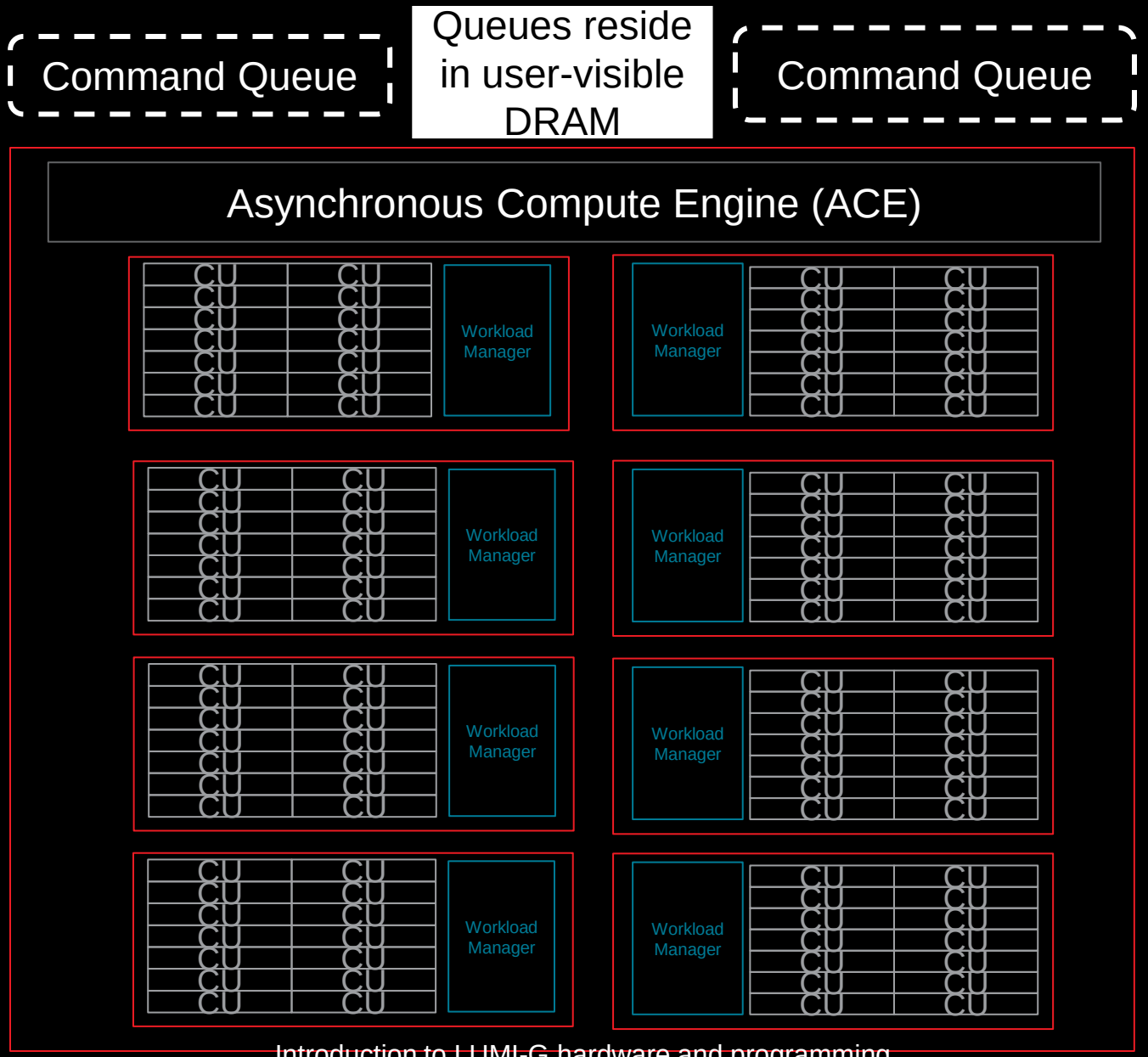
Credit: ORNL, https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html

64-core AMD “Optimized 3rd Gen EPYC” CPU Core Chiplet Die connected to GCD via Infinity Fabric CPU-GPU

AMD GCN GPU Hardware Layout (MI250X one GCD)



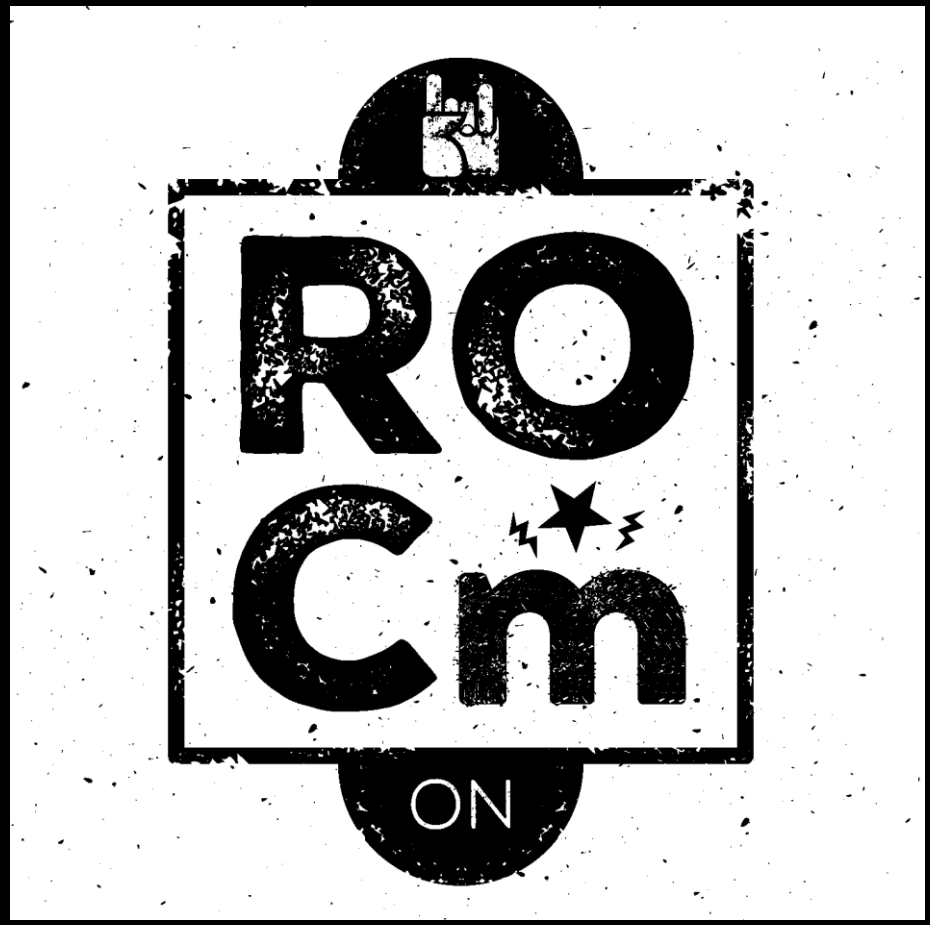
AMD GCN GPU Hardware Layout (MI250X one GCD)



ROCm and HIP

ROCm - Radeon Open Compute Platform

- HIP is part of a larger software distribution called the Radeon Open Compute Platform, or ROCm, Package
- Install instructions and documentation can be found here:
  - https://rocmdocs.amd.com/en/latest/Installation_Guide/Installation-Guide.html
- The ROCm package provides libraries and programming tools for developing HPC and ML applications on AMD GPUs
- All the ROCm environment and the libraries are provided from the supercomputer, usually, there is no need to install something yourselves
- Heterogeneous System Architecture (HSA) runtime is an API that exposes the necessary interfaces to access and interact with the hardware driven by AMDGPU driver

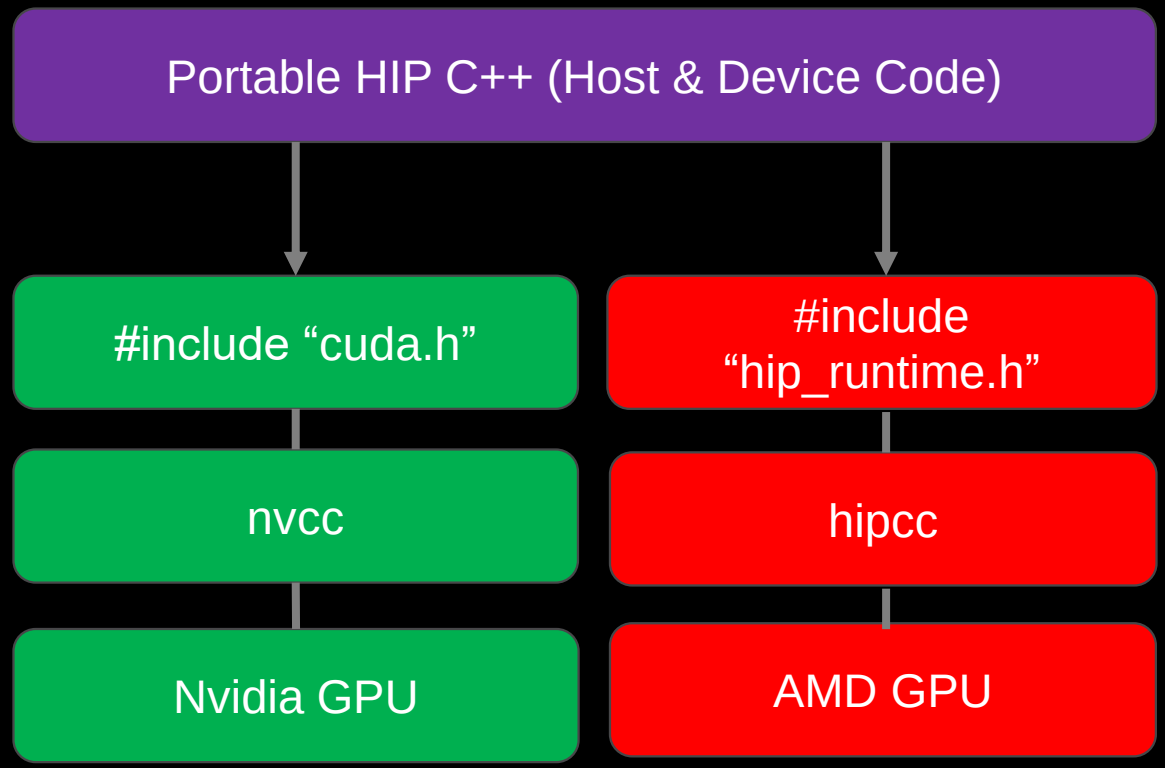


What is HIP?

AMD's **H**eterogeneous-compute **I**nterface for **P**ortability, or **HIP**, is a C++ runtime API and kernel language that allows developers to create portable applications that can run on AMD's accelerators as well as CUDA devices

HIP:

- Is open-source
- Provides an API for an application to leverage GPU acceleration for both AMD and CUDA devices
- Syntactically similar to CUDA. Most CUDA API calls can be converted in place: cuda -> hip
- Supports a strong subset of CUDA runtime functionality



Getting started with HIP

CUDA VECTOR ADD

```
__global__ void add(int n,
                    double *x,
                    double *y){
  int index = blockIdx.x * blockDim.x
              + threadIdx.x;
  int stride = blockDim.x * gridDim.x;

  for (int i = index; i < n; i += stride){
    y[i] = x[i] + y[i];
  }
}
```

HIP VECTOR ADD

```
__global__ void add(int n,
                    double *x,
                    double *y){
  int index = blockIdx.x * blockDim.x
              + threadIdx.x;
  int stride = blockDim.x * gridDim.x;

  for (int i = index; i < n; i += stride){
    y[i] = x[i] + y[i];
  }
}
```

KERNELS ARE SYNTACTICALLY THE SAME

CUDA APIs vs HIP API

CUDA	HIP
`cudaMalloc(&d_x, N*sizeof(double));`	`hipMalloc(&d_x, N*sizeof(double));`
`cudaMemcpy(d_x, x, N*sizeof(double), cudaMemcpyHostToDevice);`	`hipMemcpy(d_x, x, N*sizeof(double), hipMemcpyHostToDevice);`
`cudaDeviceSynchronize();`	`hipDeviceSynchronize();`

Launching a kernel

CUDA KERNEL LAUNCH SYNTAX

```
some_kernel<<<gridsize, blocksize,
              shared_mem_size, stream>>>
              (arg0, arg1, ...);
```

HIP KERNEL LAUNCH SYNTAX

```
hipLaunchKernelGGL(some_kernel,
                   gridsize, blocksize,
                   shared_mem_size, stream,
                   arg0, arg1, ...);
```

Or

```
some_kernel<<<gridsize, blocksize,
              shared_mem_size, stream>>>
              (arg0, arg1, ...);
```

Device Kernels: The Grid

- In HIP, kernels are executed on a 3D ”grid”
  - You might feel comfortable thinking in terms of a mesh of points, but it’s not required
- The “grid” is what you can map your problem to
  - It’s not a physical thing, but it can be useful to think that way
- AMD devices (GPUs) support 1D, 2D, and 3D grids, but most work maps well to 1D
- Each dimension of the grid partitioned into equal sized “blocks”
- Each block is made up of multiple “threads”
- The grid and its associated blocks are just organizational constructs
  - The threads are the things that do the work
- If you’re familiar with CUDA already, the grid+block structure is very similar in HIP

Device Kernels: The Grid

Some Terminology:

CUDA	HIP	OpenCL™
grid	grid	NDRange
block	block	work group
thread	work item / thread	work item
warp	wavefront	sub-group

The Grid: blocks of threads in 1D

Threads in grid have access to:

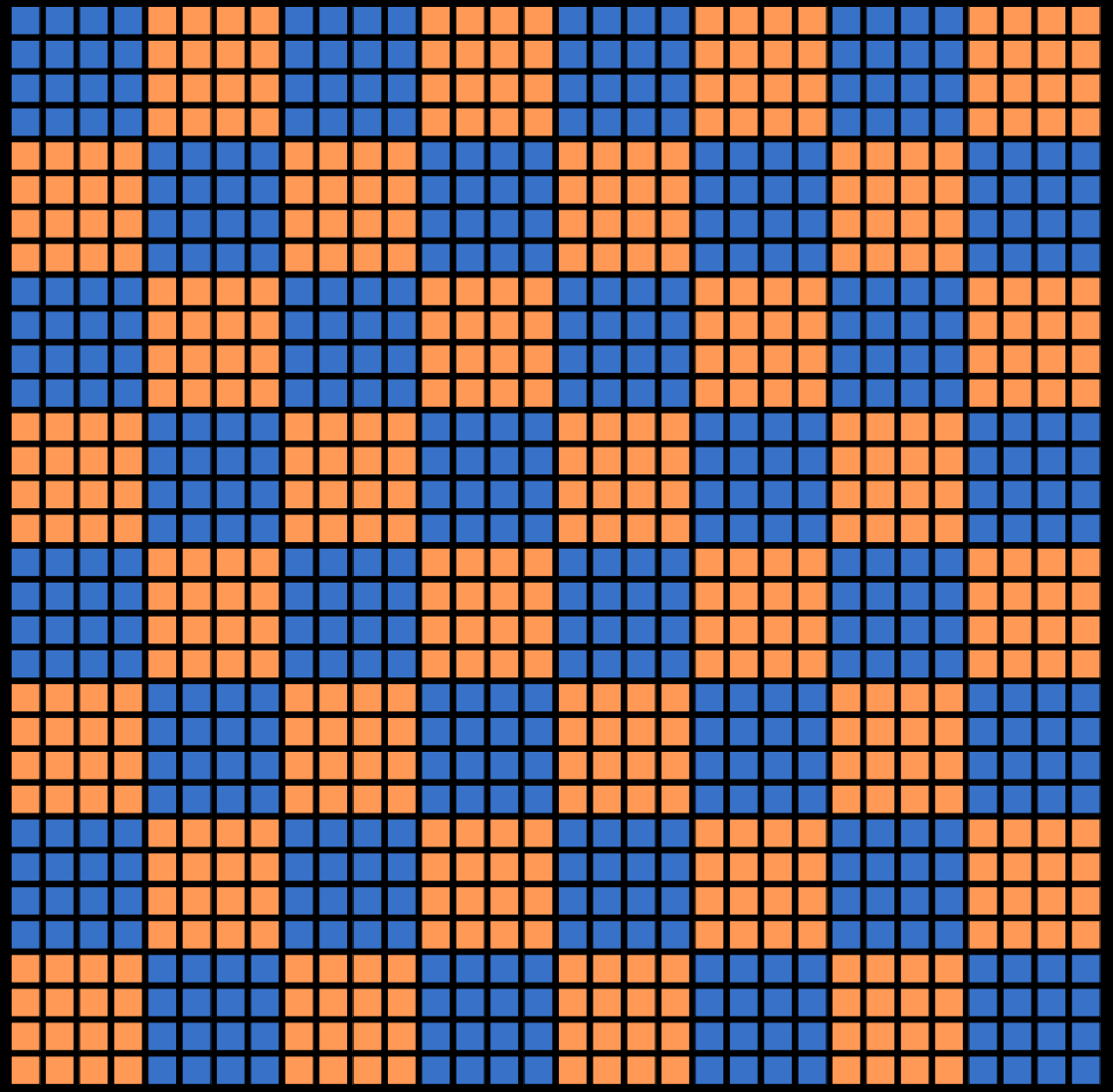
- Their respective block: blockIdx.x
- Their respective thread ID in a block: threadIdx.x
- Their block's dimension: blockDim.x
- The number of blocks in the grid: gridDim.x

The Grid: blocks of threads in 2D

- Each color is a block of threads
- Each small square is a thread
- The concept is the same in 1D and 2D
- In 2D each block and thread now has a two-dimensional index

Threads in grid have access to:

- Their respective block IDs: blockIdx.x, blockIdx.y
- Their respective thread IDs in a block: threadIdx.x, threadIdx.y
- Etc.



Kernels

A simple embarrassingly parallel loop

```
for (int i=0;i<N;i++) {
  h_a[i] *= 2.0;
}
```

Can be translated into a GPU kernel:

```
__global__ void myKernel(int N, double *d_a) {
  int i = threadIdx.x + blockIdx.x*blockDim.x;
  if (i<N) {
    d_a[i] *= 2.0;
  }
}
```

- A device function that will be launched from the host program is called a kernel and is declared with the __global__ attribute
- Kernels should be declared void
- All threads execute the kernel's body "simultaneously"
- Each thread uses its unique thread and block IDs to compute a global ID
- There could be more than N threads in the grid

Kernels

Kernels are launched from the host:

```
dim3 threads(256,1,1);                  //3D dimensions of a block of threads
dim3 blocks((N+256-1)/256,1,1);         //3D dimensions the grid of blocks

hipLaunchKernelGGL(myKernel,            //Kernel name (__global__ void function)
                   blocks,              //Grid dimensions
                   threads,             //Block dimensions
                   0,                   //Bytes of dynamic LDS space
                   0,                   //Stream (0=NULL stream)
                   N, a);               //Kernel arguments
```

Also supported similar to CUDA kernel launch syntax:

```
myKernel<<<blocks, threads, 0, 0>>>(N,a);
```

Difference between HIP and CUDA

Some things to be aware of writing HIP, or porting from CUDA:

- AMD GCN hardware ‘warp’ size = 64 (warps are referred to as ‘wavefronts’ in AMD documentation)
- Device and host pointers allocated by HIP API use flat addressing
  - Unified virtual addressing is available
- Dynamic parallelism not currently supported
- CUDA 9+ thread independent scheduling not supported (e.g., no __syncwarp)
- Some CUDA library functions do not have AMD equivalents
- Shared memory and registers per thread can differ between AMD and Nvidia hardware
- Inline PTX or AMD GCN assembly is not portable

Despite differences, majority of CUDA code in applications can be simply translated.

Usage of hipcc

Usage is straightforward. Accepts all/any flags that clang accepts, e.g.,

```
hipcc --offload-arch=gfx90a dotprod.cpp -o dotprod
```

Set HIPCC_VERBOSE=7 to see a bunch of useful information

- Compile and link lines
- Various paths

```
$ HIPCC_VERBOSE=7 hipcc --offload-arch=gfx90a dotprod.cpp -o dotprod
HIP_PATH=/opt/rocm-5.2.0
HIP_PLATFORM=amd
HIP_COMPILER=clang
HIP_RUNTIME=rocclr
ROCM_PATH=/opt/rocm-5.2.0
...
hipcc-args: --offload-arch=gfx90a dotprod.cpp -o dotprod
hipcc-cmd: /opt/rocm-5.2.0/llvm/bin/clang++ -stdc=c++11 -hc  -D__HIPCC__ -isystem /opt/rocm-
5.2.0/llvm/lib/clang/14.0.0/include
-isystem /opt/rocm-5.2.0/has/include -isystem /opt/rocm-5.2.0/include –offload-arch=gfx90a –O3 ...
```

- You can use also *`hipcc -v`* … to print some information
- With the command *`hipconfig`* you can see many information about environment variables declaration

HIP API

- Device Management: hipSetDevice(), hipGetDevice(), hipGetDeviceProperties()
- Memory Management: hipMalloc(), hipMemcpy(), hipMemcpyAsync(), hipFree(), hipHostMalloc()
- Streams: hipStreamCreate(), hipSynchronize(), hipStreamSynchronize(), hipStreamFree()
- Events: hipEventCreate(), hipEventRecord(), hipStreamWaitEvent(), hipEventElapsedTime()
- Device Kernels: __global__, __device__, hipLaunchKernelGGL()
- Device code:
  - threadIdx, blockIdx, blockDim, __shared__
  - 200+ math functions covering entire CUDA math library
- Error handling: hipGetLastError(), hipGetErrorString()
- More information: https://docs.amd.com/bundle/HIP_API_Guide/page/modules.html

Error Checking

- Most HIP API functions return error codes of type hipError_t

```
hipError_t status1 = hipMalloc(…);
hipError_t status2 = hipMemcpy(…);
```

- If API function was error-free, returns hipSuccess, otherwise returns an error code

- Can also peek/get at last error returned with

```
hipError_t status3 = hipGetLastError();
hipError_t status4 = hipPeekLastError();
```

- Can get a corresponding error string using hipGetErrorString(status). Helpful for debugging, e.g.,

```
#define HIP_CHECK(command) {     \
  hipError_t status = command;   \
  if (status!=hipSuccess) {      \
    std::cerr << “Error: HIP reports ” << hipGetErrorString(status) << std::endl; \
    std::abort(); } }
```

Streams

- A stream in HIP is a queue of tasks (e.g., kernels, memcpys, events)
  - Tasks enqueued in a stream are **completed in the order enqueued**
  - Tasks being executed in different streams are allowed to overlap and share device resources

- Streams are created via:

```
hipStream_t stream;
hipStreamCreate(&stream);
```

- And destroyed via:

```
hipStreamDestroy(stream);
```

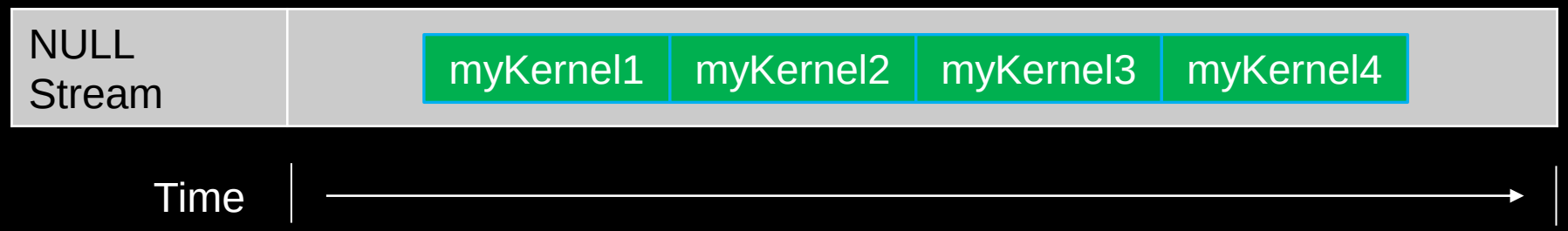
- Passing 0 or NULL as the hipStream_t argument to a function instructs the function to execute on a stream called the ‘NULL Stream’:
  - No task on the NULL stream will begin until **all previously enqueued tasks in all other streams have completed**
  - Blocking calls like hipMemcpy run on the NULL stream

Streams

- Suppose we have 4 small kernels to execute:

```
hipLaunchKernelGGL(myKernel1, dim3(1), dim3(256), 0, 0, 256, d_a1);
hipLaunchKernelGGL(myKernel2, dim3(1), dim3(256), 0, 0, 256, d_a2);
hipLaunchKernelGGL(myKernel3, dim3(1), dim3(256), 0, 0, 256, d_a3);
hipLaunchKernelGGL(myKernel4, dim3(1), dim3(256), 0, 0, 256, d_a4);
```

- Even though these kernels use only one block each, they'll execute in serial on the NULL stream:



Streams

- With streams we can effectively share the GPU's compute resources:

```
hipLaunchKernelGGL(myKernel1, dim3(1), dim3(256), 0, stream1, 256, d_a1);
hipLaunchKernelGGL(myKernel2, dim3(1), dim3(256), 0, stream2, 256, d_a2);
hipLaunchKernelGGL(myKernel3, dim3(1), dim3(256), 0, stream3, 256, d_a3);
hipLaunchKernelGGL(myKernel4, dim3(1), dim3(256), 0, stream4, 256, d_a4);
```

NULL Stream			
Stream1		myKernel1	
Stream2		myKernel2	
Stream3		myKernel3	
Stream4		myKernel4	

Note 1: Kernels must modify different parts of memory to avoid data races.

Note 2: With large kernels, overlapping computations may not help performance.

SIMD operations

Why blocks and threads?

Natural mapping of kernels to hardware:

- Blocks are dynamically scheduled onto CUs
- All threads in a block execute on the same CU
- Threads in a block share LDS memory and L1 cache
- Threads in a block are executed in **64-wide** chunks called “wavefronts”
- Wavefronts execute on SIMD units (Single Instruction Multiple Data)
- If a wavefront stalls (e.g., data dependency) CUs can quickly context switch to another wavefront

A good practice is to make the block size a multiple of 64 and have several wavefronts (e.g., 256 threads)

Porting Applications to HIP

HIPification Tools for faster code porting

- ROCm provides ‘HIPification’ tools to do the heavy-lifting on porting CUDA codes to ROCm
  - Hipify-perl
  - Hipify-clang

- Good resource to help with porting: https://github.com/ROCm-Developer-Tools/HIPIFY/blob/master/README.md

- In practice, large portions of many HPC codes have been automatically Hipified:
  - ~90% of CUDA code in CORAL-2 HACC
  - ~80% of CUDA code in CORAL-2 PENNANT
  - ~80% of CUDA code in CORAL-2 QMCPack
  - ~95% of CUDA code in CORAL-2 Laghos

  The remaining code requires programmer intervention

Hipify tools

- Hipify-perl:
  - Easy to use –point at a directory and it will attempt to hipify CUDA code
  - Very simple string replacement technique: may make incorrect translations
    - **`sed -e 's/cuda/hip/g'`, (e.g., cudaMemcpy becomes hipMemcpy)**
  - Recommended for quick scans of projects
  - It will not translate if it does not recognize a CUDA call and it will report it

- Hipify-clang:
  - Requires clang compiler
  - More robust translation of the code. Uses clang to parse files and perform semantic translation
  - Can generate warnings and assistance for code for additional user analysis
  - High quality translation, particularly for cases where the user is familiar with the make system

Hipify-perl

- It is located in $HIP/bin/ (**export PATH=$PATH:[MYHIP]/bin**)
- Command line tool: **hipify-perl foo.cu > new_foo.cpp**
- Compile: **hipcc new_foo.cpp**
- How does this this work in practice?
  - Hipify source code
  - Check it in to your favorite version control
  - Try to build
  - Manually work on the rest

Hipify-clang

- Build from source
- hipify-clang has unit tests using LLVM lit/FileCheck (44 tests)

- Hipification requires same headers that would be needed to compile it with clang:
- ./hipify-clang foo.cu -I /usr/local/cuda-8.0/samples/common/inc

- https://github.com/ROCm-Developer-Tools/HIP/tree/master/hipify-clang

Gotchas

- Hipify tools are not running your application, or checking correctness
- Code relying on specific Nvidia hardware aspects (e.g., warp size == 32) may need attention after conversion
- Certain functions may not have a correspondent hip version (e.g., __shfl_down_sync)
- Hipifying can’t handle inline PTX assembly
  - Can either use inline GCN ISA, or convert it to HIP
- Hipify-perl and hipify-clang can both convert library calls

- None of the tools convert your build system script such as CMAKE or whatever else you use. The user is responsible to find the appropriate flags and paths to build the new converted HIP code.

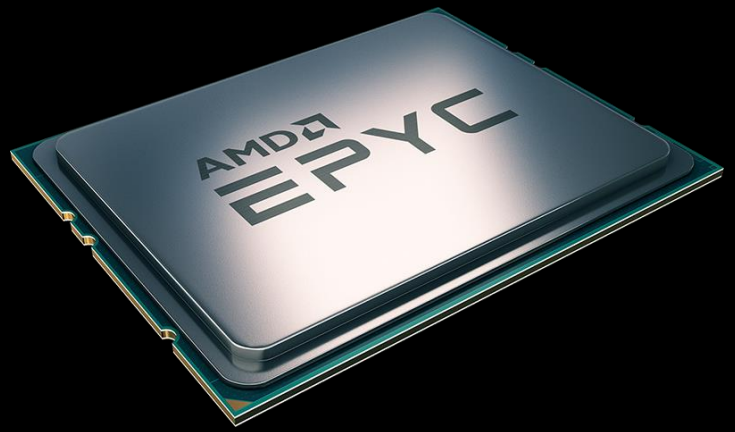
What to look for when porting:

- Inline PTX assembly
- CUDA Intrinsics
- Hardcoded dependencies on warp size, or shared memory size
  - Grep for "32" *just in case*
  - Do not hardcode the warpsize! Rely on warpSize device definition, #define WARPSIZE *size,* or props.warpSize from host
- Code geared toward limiting size of register file on NVIDIA hardware
- Unsupported functions

A Tale of Host and Device

Source code in HIP has two flavors: Host code and Device code

- The Host is the CPU
- Host code runs here
- Usual C++ syntax and features
- Entry point is the ‘main’ function
- HIP API can be used to create device buffers, move between host and device, and launch device code.



- The Device is the GPU
- Device code runs here
- C-like syntax
- Device codes are launched via “kernels”
- Instructions from the Host are enqueued into “streams”

Fortran

- First Scenario: Fortran + CUDA C/C++
  - Assuming there is no CUDA code in the Fortran files.
  - Hipify CUDA
  - Compile and link with hipcc
- Second Scenario: CUDA Fortran
  - There is no hipify equivalent but there is another approach…
  - HIP functions are callable from C, using `extern C`
  - See hipfort

CUDA Fortran -> Fortran + HIP C/C++

- There is no HIP equivalent to CUDA Fortran
- But HIP functions are callable from C, using `extern C`, so they can be called directly from Fortran
- The strategy here is:
  - **Manually port** CUDA Fortran code to HIP kernels in C-like syntax
  - Wrap the kernel launch in a C function
  - Call the C function from Fortran through Fortran's ISO_C_binding. It requires Fortran 2008 because of the pointers utilization.
- This strategy should be usable by Fortran users since it is standard conforming Fortran
- ROCm has an interface layer, hipFort, which provides the wrapped bindings for use in Fortran
  - https://github.com/ROCmSoftwarePlatform/hipfort

Alternatives to HIP

- Can also target AMD GPUs through OpenMP 5.0 target offload
  - ROCm provides OpenMP support
  - AMD OpenMP compiler (AOMP) could integrate updated improvements regarding OpenMP offloading performance, sometimes experimental stuff to validate before ROCm integration (https://github.com/ROCm-Developer-Tools/aomp)
  - GCC provides OpenMP offload support.
- GCC will provide OpenACC
- Clacc from ORNL: https://github.com/llvm-doe-org/llvm-project/tree/clacc/main OpenACC from LLVM only for C (Fortran and C++ in the future)
  - Translate OpenACC to OpenMP Offloading

OpenMP Offload GPU Support

- ROCm and AOMP
  - ROCm supports both HIP and OpenMP
  - AOMP: the AMD OpenMP research compiler, it is used to prototype the new OpenMP features for ROCm
- HPE Compilers
  - Provides offloading support to AMD GPUs, through OpenMP, HIP, and OpenACC (only for Fortran)
- GNU compilers:
  - Provide OpenMP and OpenACC offloading support for AMD GPUs
  - GCC 11: Supports AMD GCN gfx908
  - GCC 13: Supports AMD GCN gfx90a

Understanding the hardware options

- ***rocminfo***
  - 110 CUs
  - Wavefront of size 64
  - 4 SIMDs per CU

```
Node:                    11
Device Type:             GPU
Cache Info:
  L1:                      16(0x10) KB
  L2:                      8192(0x2000) KB
Chip ID:                 29704(0x7408)
Cacheline Size:          64(0x40)
Max Clock Freq. (MHz):   1700
BDFID:                   56832
Internal Node ID:        11
Compute Unit:            110
SIMDs per CU:            4
Shader Engines:          8
Shader Arrs. per Eng.:   1
WatchPts on Addr. Ranges:4
Features:                KERNEL_DISPATCH
Fast F16 Operation:      TRUE
Wavefront Size:          64(0x40)
Workgroup Max Size:      1024(0x400)
Workgroup Max Size per Dimension:
  x                        1024(0x400)
  y                        1024(0x400)
  z                        1024(0x400)
Max Waves Per CU:        32(0x20)
Max Work-item Per CU:    2048(0x800)
```

`#pragma omp target teams distribute parallel for` *`simd`*

Options for pragma omp teams target:

- num_teams(220): Multiple number of workgroups with regards the compute units
- thread_limit(256): Threads per workgroup

- Thread limit is multiple of 64
- Teams*thread_limit should be multiple or a divisor of the trip count of a loop

ROCm Libraries

ROCm GPU Libraries

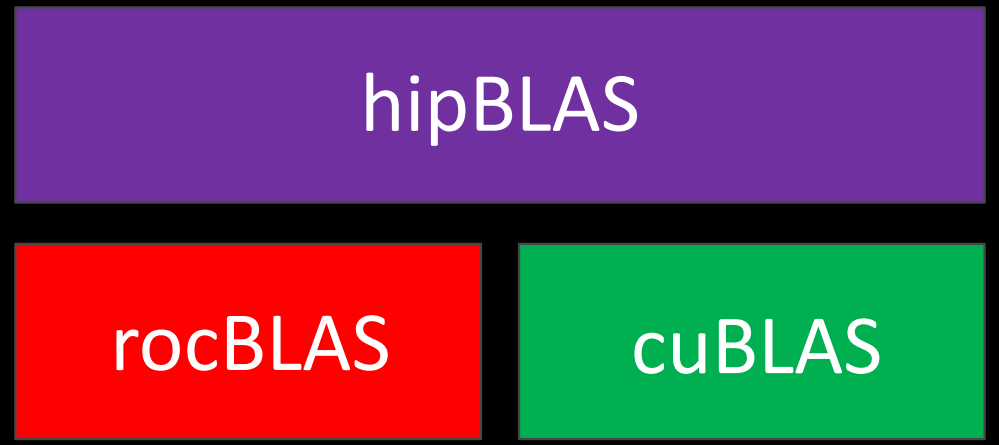
ROCm provides several GPU math libraries

- Typically, two versions:
  - roc* -> AMD GPU library, usually written in HIP
  - hip* -> Thin interface between roc* and Nvidia cu* library

When developing an application meant to target both CUDA and AMD devices, use the hip* libraries (portability)

When developing an application meant to target only AMD devices, may prefer the roc* library API (performance).

- Some roc* libraries perform **better** by using addition APIs not available in the cu* equivalents



AMD Math Library Equivalents: “Decoder Ring”

CUBLAS	**ROCBLAS**	Basic Linear Algebra Subroutines
CUFFT	**ROCFFT**	Fast Fourier Transforms
CURAND	**ROCRAND**	Random Number Generation
THRUST	**ROCTHRUST**	C++ Parallel Algorithms
CUB	**ROCPRIM**	Optimized Parallel Primitives

AMD Math Library Equivalents: “Decoder Ring”

CUSPARSE	ROCSPARSE	Sparse BLAS, SpMV, etc.
CUSOLVER	ROCSOLVER	Linear Solvers
AMGX	ROCALUTION	Solvers and preconditioners for sparse linear systems

GITHUB.COM/ROCM-DEVELOPER-TOOLS/HIP → HIP_PORTING_GUIDE.MD FOR A COMPLETE LIST

Some Links to Key Libraries

- BLAS
  - rocBLAS (https://github.com/ROCmSoftwarePlatform/rocBLAS)
  - hipBLAS (https://github.com/ROCmSoftwarePlatform/hipBLAS)
- FFTs
  - rocFFT (https://github.com/ROCmSoftwarePlatform/rocFFT)
  - hipFFT (https://github.com/ROCmSoftwarePlatform/hipFFT)
- Random number generation
  - rocRAND (https://github.com/ROCmSoftwarePlatform/rocRAND)
- Sparse linear algebra
  - rocSPARSE (https://github.com/ROCmSoftwarePlatform/rocSPARSE)
  - hipSPARSE (https://github.com/ROCmSoftwarePlatform/hipSPARSE)
- Iterative solvers
  - rocALUTION (https://github.com/ROCmSoftwarePlatform/rocALUTION)
- Parallel primitives
  - rocPRIM (https://github.com/ROCmSoftwarePlatform/rocPRIM)
  - hipCUB (https://github.com/ROCmSoftwarePlatform/hipCUB)

AMD Machine Learning Library Support

Machine Learning Frameworks:

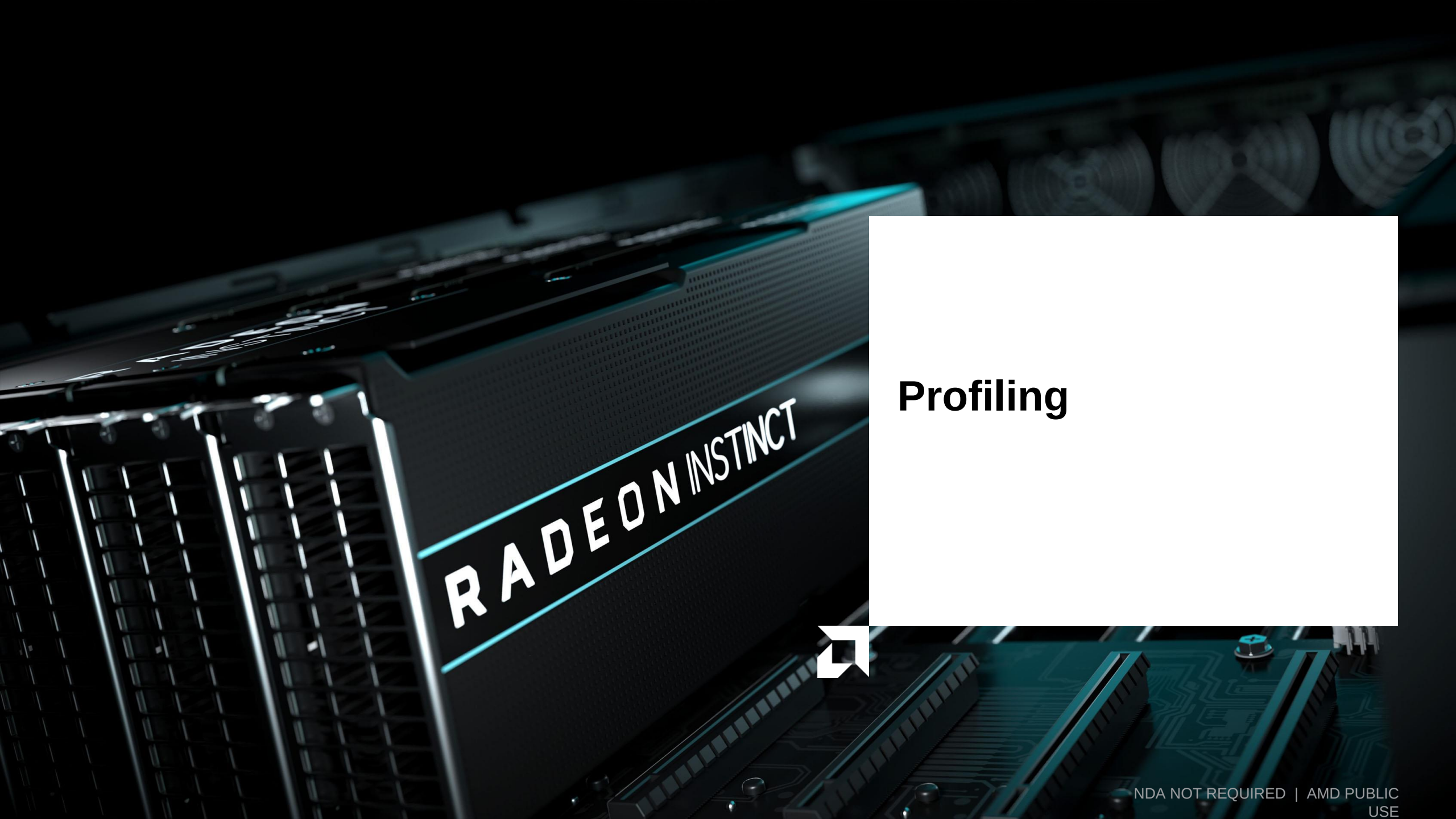
- Tensorflow: https://github.com/ROCmSoftwarePlatform/tensorflow-upstream
- Pytorch: https://github.com/ROCmSoftwarePlatform/pytorch
- Caffe: https://github.com/ROCmSoftwarePlatform/hipCaffe

Machine Learning Libraries:

- MIOpen (similar to cuDNN): https://github.com/ROCmSoftwarePlatform/MIOpen
- Tensile (GEMM Autotuner): https://github.com/ROCmSoftwarePlatform/Tensile
- RCCL (ROCm analogue of NCCL): https://github.com/ROCmSoftwarePlatform/rccl
- Horovod (Distributed ML): https://github.com/ROCmSoftwarePlatform/horovod

Benchmarks:

- DeepBench: https://github.com/ROCmSoftwarePlatform/DeepBench
- MLPerf: https://mlperf.org



Profiling

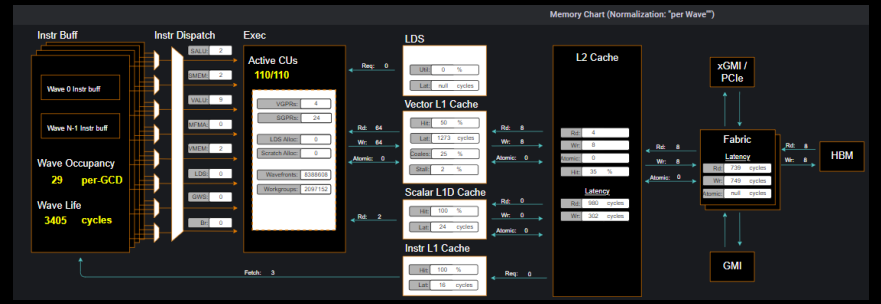
Background – AMD Profilers

- rocprof
  - github.com/ROCm-Developer-Tools/rocprofiler
  - Raw collection of GPU counters and traces
  - Counter collection driven by user provided input files
  - Counter results output in CSV
  - Trace collection support for:
    - HIP
    - HSA
    - GPU
  - 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

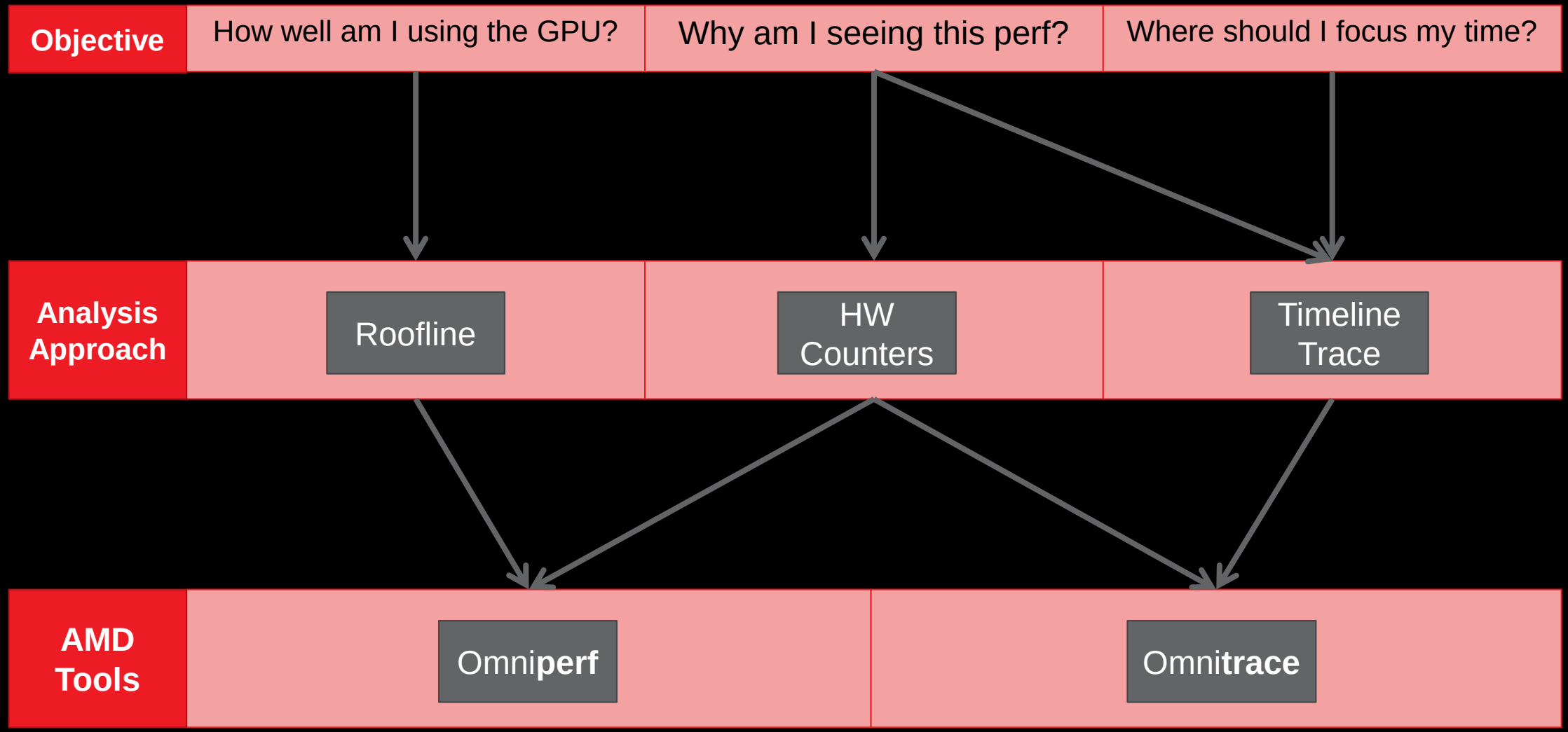
- Omni**trace**
  - github.com/AMDResearch/omnitrace
  - Comprehensive trace collection and visualization of CPU+GPU
  - Includes support for:
    - HIP, HSA, GPU
    - OpenMP®
    - MPI
    - Kokkos
    - Pthreads
    - Multi-GPU
  - Visualizations with Perfetto

- Omni**perf**
  - github.com/AMDResearch/omniperf
  - Automated collection, analysis and visualization of performance counters
  - Includes support for:
    - GPU Speed-of-Light Analysis
    - Memory Chart Analysis
    - Roofline Analysis
    - Kernel comparison
  - Visualizations with Grafana or standalone GUI



ing environment - 11 January 2023

Background – AMD Profilers



Rocprof

AMD GPU Profiling

- ROC-profiler (or simply rocprof) is the command line front-end for AMD's GPU profiling libraries
  - Repo: https://github.com/ROCm-Developer-Tools/rocprofiler
- rocprof contains the central components allowing the collection of application tracing and counter collection
  - Under constant development
- Provided in the ROCm releases
- The output of rocprof can be visualized using the chrome browser with Perfetto (https://ui.perfetto.dev/)

rocProf: Getting started + useful flags

- To get help:
  - `$ /opt/rocm-5.2.0/bin/rocprof -h`
- Useful housekeeping flags:
  - --timestamp <on|off> : turn on/off gpu kernel timestamps
  - --basenames <on|off>: turn on/off truncating gpu kernel names (i.e., removing template parameters and argument types)
  - -o <output csv file>: Direct counter information to a particular file name
  - -d <data directory>: Send profiling data to a particular directory
  - -t <temporary directory>: Change the directory where data files typically created in /tmp are placed. This allows you to save these temporary files.
- Flags directing rocprofiler activity:
  - -i input<.txt|.xml> - specify an input file (note the output files will now be named input.*)
  - --hsa-trace - to trace GPU Kernels, host HSA events (more later) and HIP memory copies.
  - --hip-trace - to trace HIP API calls
  - --roctx-trace - to trace roctx markers
  - --kfd-trace - to trace GPU driver calls
- Advanced usage
  - -m <metric file>: Allows the user to define and collect custom metrics. See rocprofiler/test/tool/*.xml on GitHub for examples.

rocProf: Collecting application traces

- rocProf can collect a variety of trace event types, and generate timelines in JSON format for use with Perfetto, currently:

Trace Event	rocprof Trace Mode
HIP API call	--hip-trace
GPU Kernels	--hip-trace
Host <-> Device Memory copies	--hip-trace
CPU HSA Calls	--hsa-trace
User code markers	--roctx-trace

- You can combine modes like --hip-trace --hsa-trace

rocProf: Information about the kernels

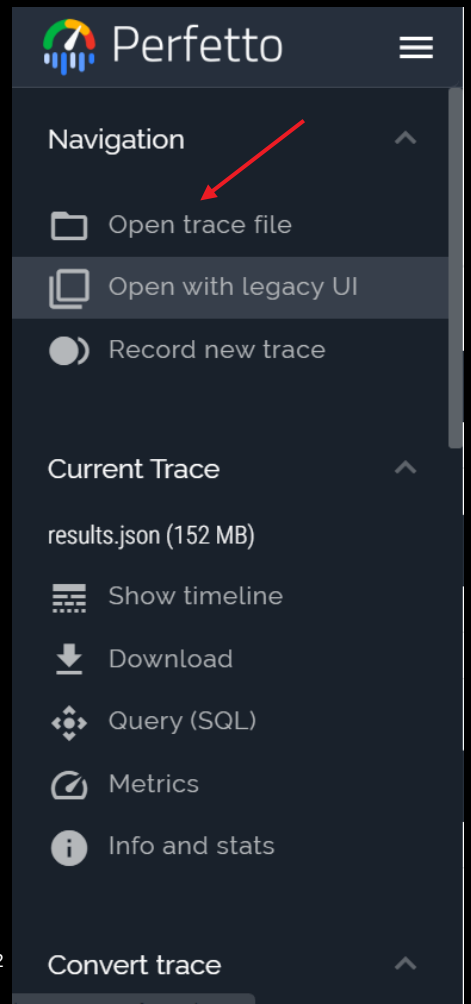
- rocprofiler can collect kernels information
  - **`$ /opt/rocm/bin/rocprof --stats --basenames on <app with arguments>`**
  - This will output two csv files, one with information per each call of the kenel *results.csv* and one with statistics grouped by each kernel *results.stats.csv.*
  - Content of results.stats.csv:

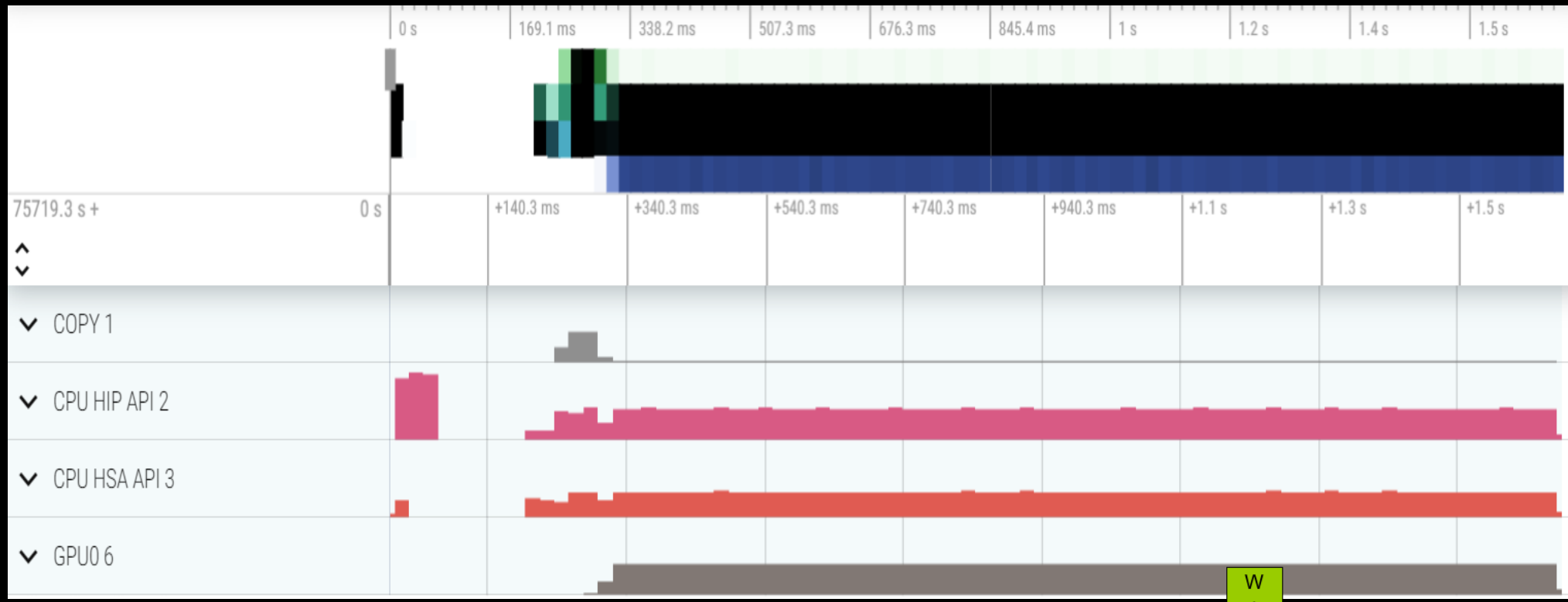
"Name",	"Calls",	"TotalDurationNs",	"AverageNs",	"Percentage"
"LocalLaplacianKernel",	1000,	817737586,	817737,	40.908259879301134
"JacobiIterationKernel",	1000,	699515425,	699515,	34.994060790890174
"NormKernel1",	1001,	454737348,	454283,	22.748756969583884
"HaloLaplacianKernel",	1000,	14561933,	14561,	0.7284773865206329
"NormKernel2",	1001,	12395374,	12382,	0.620092789636225
"__amd_rocclr_fillBufferAligned.kd",	1,	7040,	7040,	0.00035218406794656007

  - This way you know directly which kernels consume most of the time, it does not mean that the performance is slow, for now.

rocProf and Perfetto: Collecting and visualizing application traces

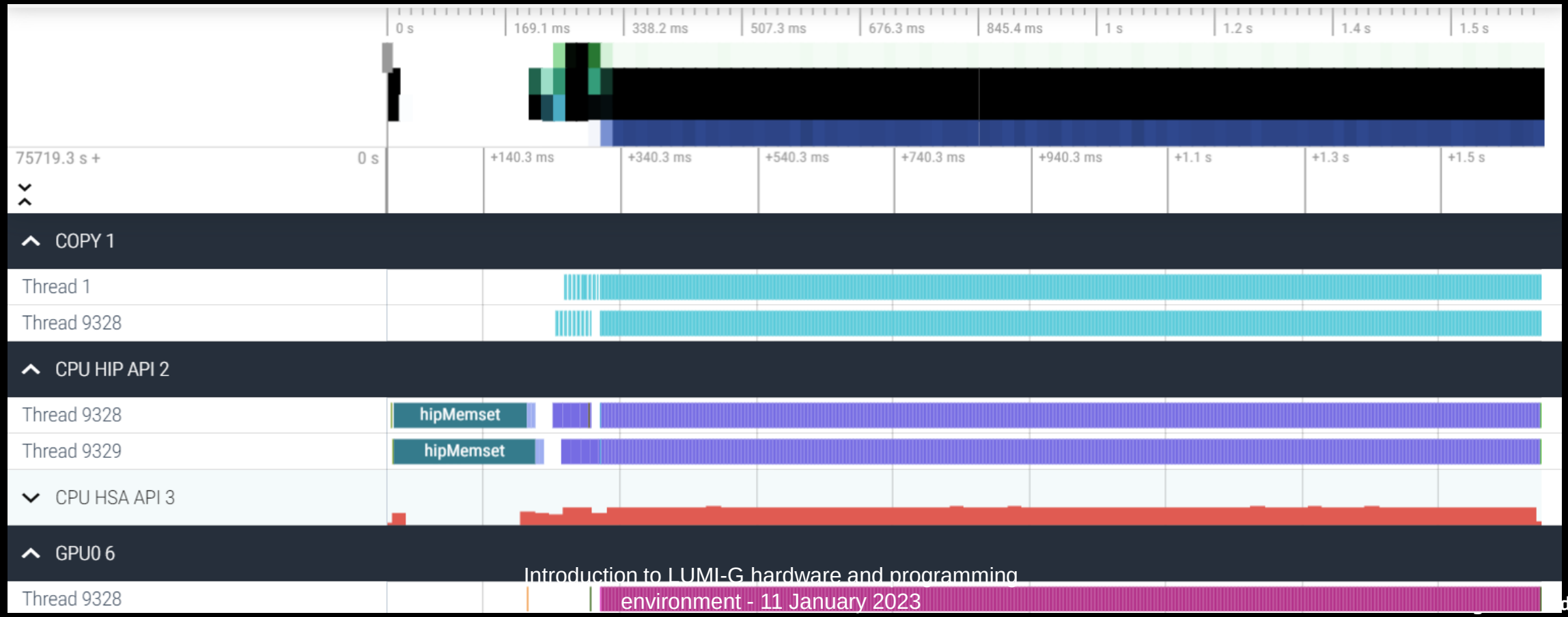
- rocprofiler can collect traces
  - **`$ /opt/rocm/bin/rocprof --hip-trace --hsa-trace <app with arguments>`**
  - This will output a .json file that can be visualized using the chrome browser and Perfetto (https://ui.perfetto.dev/)





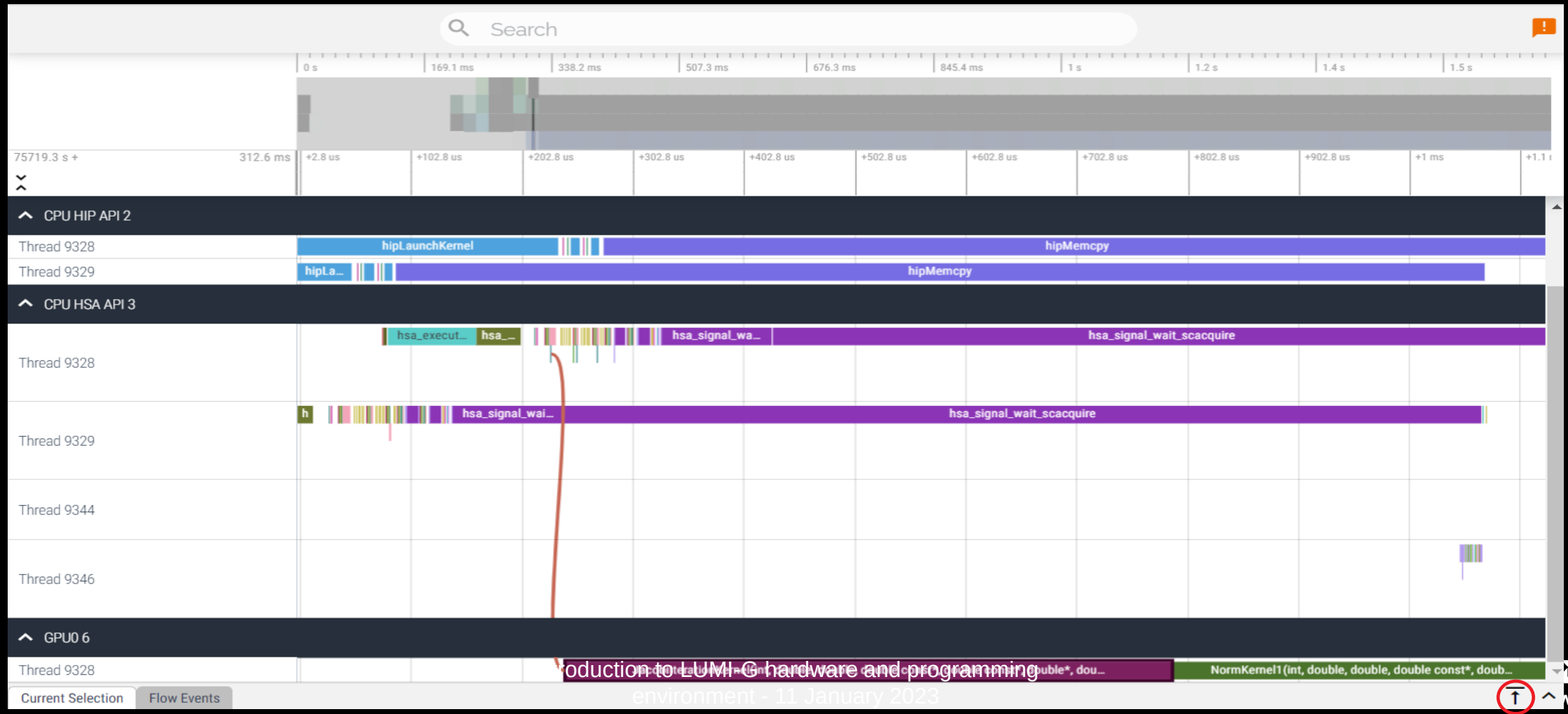
Perfetto: Visualizing application traces

- We have expanded the COPY 1, CPU HIP API 2 and GPU0 6
- X axis is time and it displays events or counters.
- Handle the zoom by keystrokes: W zoom, S zoom out, A move left, D move right

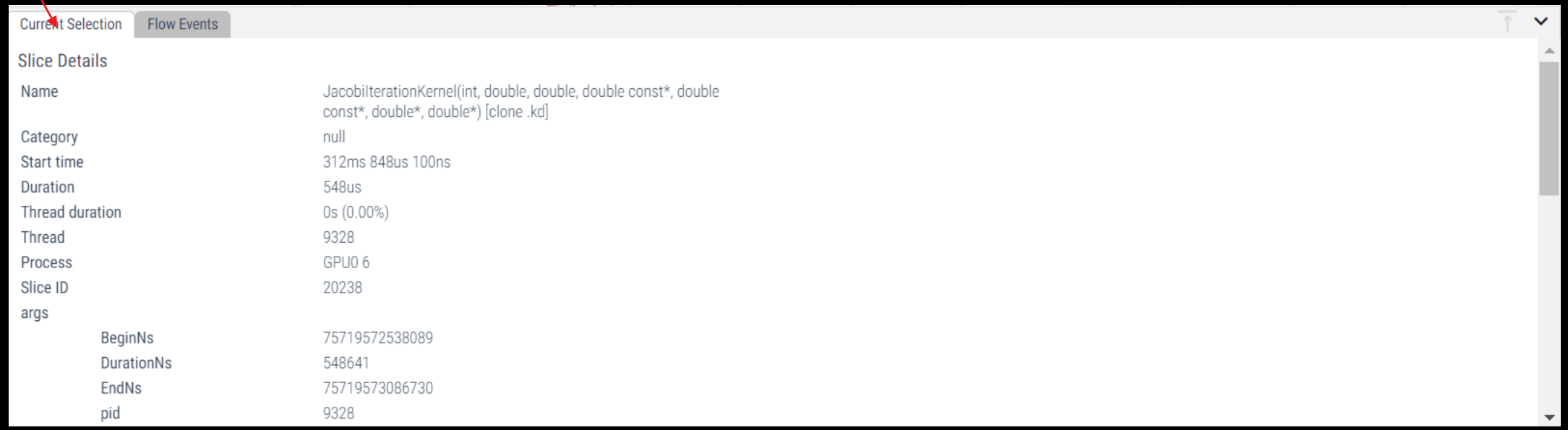


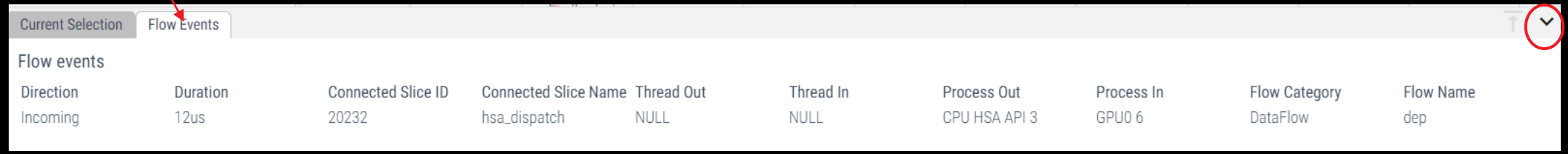
Perfetto: Kernel and flows

- Zoom and select a kernel, you can see the link to the HSA call enables the kernel
- Try to open the information for the kernel (button right down)



Perfetto: Information about kernels and flow events

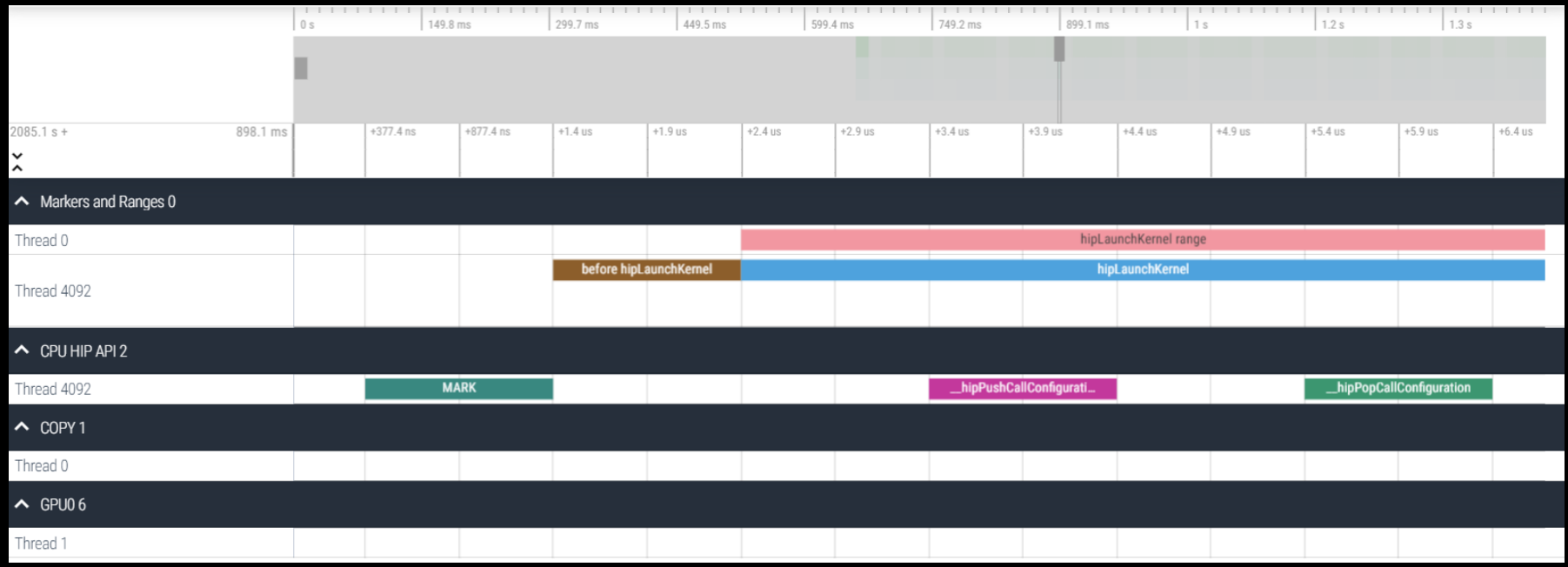




rocprof: Collecting application traces with markers

- Rocprof can collect user code-markers using rocTX
  - See MatrixTranspose.cpp example on roctracer GitHub page for sample in-code usage
  - **`$ /opt/rocm/bin/rocprof --hip-trace --roctx-trace <app with arguments>`**

```
roctracer_mark("before HIP
LaunchKernel");

roctxMark("before hipLaunchKernel");
int rangeId =
roctxRangeStart("hipLaunchKernel
range");

roctxRangePush("hipLaunchKernel");
hipLaunchKernelGGL(matrixTranspose,…)
;
roctracer_mark("after HIP
LaunchKernel");
 roctxMark("after hipLaunchKernel");
```



rocprof: Collecting hardware counters

- rocprofiler can collect a number of hardware counters and derived counters
  - **$ /opt/rocm/bin/rocprof --list-basic**
  - **$ /opt/rocm/bin/rocprof --list-derived**
- Specify counters in a counter file. For example:
  - **$ /opt/rocm/bin/rocprof -i rocprof_counters.txt <app with args>**
  - **$ cat rocprof_counters.txt**

```
pmc : Wavefronts VALUInsts VFetchInsts VWriteInsts VALUUtilization VALUBusy WriteSize
pmc : SALUInsts SFetchInsts LDSInsts FlatLDSInsts GDSInsts SALUBusy FetchSize
pmc : L2CacheHit MemUnitBusy MemUnitStalled WriteUnitStalled ALUStalledByLDS LDSBankConflict
...
```

  - A limited number of counters can be collected during a specific pass of code
    - Each line in the counter file will be collected in one pass
    - You will receive an error suggesting alternative counter ordering if you have too many / conflicting counters on one line
  - A csv file will be created by this command containing all of the requested counters

rocprof: Commonly Used Counters

- VALUUtilization: The percentage of ALUs active in a wave. Low VALUUtilization is likely due to high divergence or a poorly sized grid
- VALUBusy: The percentage of GPUTime vector ALU instructions are processed. Can be thought of as something like compute utilization
- FetchSize: The total kilobytes fetched from global memory
- WriteSize: The total kilobytes written to global memory
- L2CacheHit: The percentage of fetch, write, atomic, and other instructions that hit the data in L2 cache
- MemUnitBusy: The percentage of GPUTime the memory unit is active. The result includes the stall time
- MemUnitStalled: The percentage of GPUTime the memory unit is stalled
- WriteUnitStalled: The percentage of GPUTime the write unit is stalled

Full list at: https://github.com/ROCm-Developer-Tools/rocprofiler/blob/amd-master/test/tool/metrics.xml

Performance counters tips and tricks

- GPU Hardware counters are global
  - Kernel dispatches are serialized to ensure that only one dispatch is ever in flight
  - It is recommended that no other applications are running that use the GPU when collecting performance counters.
- Use **“`--basenames on`”** which will report only kernel names, leaving off kernel arguments.
- How do you time a kernel’s duration?
  - **`$ /opt/rocm/bin/rocprof --timestamp on -i rocprof_counters.txt <app with args>`**
  - This produces four times: DispatchNs, BeginNs, EndNs, and CompleteNs
  - Closest thing to a kernel duration: EndNs - BeginNs
  - If you run with **“`--stats`”** the resultant results file will automatically include a column that calculates kernel duration
    - Note: the duration is aggregated over repeated calls to the same kernel

rocprof: Multiple MPI Ranks

- rocprof can collect counters and traces for multiple MPI ranks
- Say you want to profile an application usually called like this:
  - **`mpiexec –np <n> ./Jacobi_hip –g <x> <y>`**
  - Then invoke the profiler by executing:

```
mpiexec -np <n> rocprof --hip-trace ./Jacobi_hip -g <x> <y>
or
srun --ntasks=n rocprof --hip-trace ./Jacobi_hip -g <x> <y>
```

- This will produce a single CSV file per MPI process
- Multi-node profiling currently isn’t supported

Profiling Per MPI Rank: From Another Node(1)

- Let’s consider a 3-step run:
  - sbatch_profiling.sh with sbatch command line to launch the app
  - rocprof_batch.slurm This file contains sbatch parameters and the call to srun command line
  - rocprof_wrapper.sh calls rocprof command line with input parameters to run the application to be profiled

- $cat sbatch_profiling.sh
  - sbatch -p <partition> -w <node> rocprof_batch.slurm

- $cat rocprof_batch.slurm

```
#!/bin/bash
#SBATCH --job-name=run
#SBATCH --ntasks=2
#SBATCH --ntasks-per-node=2
#SBATCH --gpus-per-task=1
#SBATCH --cpus-per-task=1
#SBATCH --distribution=block:block
#SBATCH --time=00:20:00
#SBATCH --output=out.txt
#SBATCH --error=err.txt
#SBATCH -A XXXXX
cd ${SLURM_SUBMIT_DIR}
```

- **load necessary modules**
- **export necessary environment variables**

```
 make clean all
srun ./rocprof_wrapper.sh ${repository} triad_off_mpi triad_off_mpi
```

Profiling Per MPI Rank: From Another Node(2)

- $cat rocprof_wrapper.sh

```
#!/bin/bash
set -euo pipefail
# depends on ROCM_PATH being set outside;  input arguments are the output directory & the name
outdir="$1"
name="$2"
if [[ -n ${OMPI_COMM_WORLD_RANK+z} ]]; then
  # mpich
  export MPI_RANK=${OMPI_COMM_WORLD_RANK}
elif [[ -n ${MV2_COMM_WORLD_RANK+z} ]]; then
  # ompi
  export MPI_RANK=${MV2_COMM_WORLD_RANK}
elif [[ -n ${SLURM_PROCID+z} ]]; then
  export MPI_RANK=${SLURM_PROCID}
else
  echo "Unknown MPI layer detected! Must use OpenMPI, MVAPICH, or SLURM"
  exit 1
fi
rocprof="${ROCM_PATH}/bin/rocprof"

pid="$$"
outdir="${outdir}/rank_${pid}_${MPI_RANK}"
outfile="${name}_${pid}_${MPI_RANK}.csv"
${rocprof} -d ${outdir} --hsa-trace -o ${outdir}/${outfile} "${@:3}"
```

rocprof: Profiling Overhead

- As with every profiling tool that collects data, there is an overhead
- The percentage of the overhead depends on many aspects, for example if you try to instrument tiny tasks in a loop, this can take more time than tasks outside a loop
- If you try to collect many counters and especially ones that need more than one pass, then this could cause overhead if there a lot of related calls
- Also, if a lot of markers are added and especially in a loop then the roctx-trace can take significantly more time than the non instrumented execution time
- In general, more the data you collect, more the overhead can be, and it depends on the application.

Omnitrace

Omnitrace: Application Profiling, Tracing, and Analysis

- It is an AMD Research tool, repository: https://github.com/AMDResearch/omnitrace
- It is not part of ROCm stack
- Omnitrace is a comprehensive profiling and tracing tool for parallel applications written in C, C++, Fortran, HIP, OpenCL™, and Python™ which execute on the CPU or CPU+GPU
- Data collection modes:
  - Dynamic instrumentation
  - Statistical sampling
  - Process-level sampling
  - Critical trace generation
- Data analysis:
  - High-level summary profiles
  - Comprehensive traces
  - Critical trace analysis
- Parallelism support: HIP, HSA, Pthreads, MPI, Kokkos, OpenMP®
- GPU Metrics: GPU hardware counters, HIP/HSA API, HIP kernel tracing, HSA operation tracing, memory/power/temperature/utilization
- CPU Metrics: Hardware counters, timing metrics, memory metrics, network statistics, I/O, and more

Installation (if required)

- Instructions for binary installation
- Visit the Omnitrace releases page: https://github.com/AMDResearch/omnitrace/releases
- Select the version that matches your operating system, ROCm version, etc.
- For an HPE/AMD system, we select OpenSuse operating system
- For example, download the installer *omnitrace-1.7.2-opensuse-15.4-ROCm-50300-PAPI-OMPT-Python3.sh*
- Any user can install it in his project space but it should not be required
- There are rpm and deb files for installation also
- Full documentation: https://amdresearch.github.io/omnitrace/

```
wget https://github.com/AMDResearch/omnitrace/releases/download/v1.7.3/omnitrace-1.7.3-opensuse-15.4-ROCm-50300-PAPI-OMPT-Python3.sh

mkdir /opt/omnitrace/
module load rocm        // not required if you build it on your laptop
chmod +x omnitrace-1.7.3-opensuse-15.4-ROCm-50300-PAPI-OMPT-Python3.sh
./omnitrace-1.7.3-opensuse-15.4-ROCm-50300-PAPI-OMPT-Python3.sh --prefix=/opt/omnitrace --exclude-subdir
export PATH=/opt/omnitrace/bin:$PATH
source omnitrace_installation_path/share/omnitrace/setup-env.sh
```

Omnitrace instrumentation modes

- Runtime instrumentation: Dynamic binary instrumentation, it can instrument a lot of data and increased overhead
- Sampling instrumentation (omnitrace-sample)
- Attaching to a process (-p)
- Binary rewriting (-o)
  - It will not instrument the dynamically-linked libraries, thus lower overhead and faster execution
  - This approach is recommended when the target executable uses process-level parallelism (e.g. MPI)
  - To instrument dynamic libraries: https://amdresearch.github.io/omnitrace/instrumenting.html#binary-rewriting-a-library

For problems, create an issue here: https://github.com/AMDResearch/omnitrace/issues
Documentation: https://amdresearch.github.io/omniperf/

Execution

- Runtime instrumentation

```
srun … omnitrace <omnitrace-options> -- <exe> [<exe-options>]
```

- Sampling instrumentation

```
srun … omnitrace-sample <omnitrace-options> -- <exe> [<exe-options>]
```

- Binary rewriting

```
srun … omnitrace <omnitrace-options> -o <name-of-new-exe-or-library> -- <exe-or-library>
srun … <name-of-new-exe>
```

Omnitrace configuration (I)

srun -n 1 --gpus 1 omnitrace-avail --categories omnitrace

ENVIRONMENT VARIABLE	VALUE	CATEGORIES
OMNITRACE_CONFIG_FILE	%env{HOME}%/.omnitrace.cfg;%env{HOME}%/.omnitrace.json	config, core, libomnitrace, omnitrace, timemory
OMNITRACE_CRITICAL_TRACE	false	backend, critical_trace, custom, libomnitrace, omnitrace
OMNITRACE_OUTPUT_PATH	omnitrace-%tag%-output	filename, io, libomnitrace, omnitrace, timemory
OMNITRACE_OUTPUT_PREFIX		filename, io, libomnitrace, omnitrace, timemory
OMNITRACE_PERFETTO_BACKEND	inprocess	custom, libomnitrace, omnitrace, perfetto
OMNITRACE_PERFETTO_BUFFER_SIZE_KB	1024000	custom, data, libomnitrace, omnitrace, perfetto
OMNITRACE_PERFETTO_FILL_POLICY	discard	custom, data, libomnitrace, omnitrace, perfetto
OMNITRACE_PROCESS_SAMPLING_DURATION	-1	custom, libomnitrace, omnitrace, process_sampling, sampling
OMNITRACE_PROCESS_SAMPLING_FREQ	0	custom, libomnitrace, omnitrace, process_sampling
OMNITRACE_ROCM_EVENTS		custom, hardware_counters, libomnitrace, omnitrace, rocm, rocprofiler
OMNITRACE_SAMPLING_CPUS	0-3	custom, libomnitrace, omnitrace, process_sampling
OMNITRACE_SAMPLING_DELAY	0.5	custom, libomnitrace, omnitrace, process_sampling, sampling
OMNITRACE_SAMPLING_DURATION	0	custom, libomnitrace, omnitrace, process_sampling, sampling
OMNITRACE_SAMPLING_FREQ	100	custom, libomnitrace, omnitrace, process_sampling, sampling
OMNITRACE_SAMPLING_GPUS	all	custom, libomnitrace, omnitrace, process_sampling, rocm, rocm_smi
OMNITRACE_TIMEMORY_COMPONENTS	wall_clock,cpu_clock,page_rss,cpu_util,papi_vector	component, custom, libomnitrace, omnitrace, timemory
OMNITRACE_TIME_OUTPUT	true	filename, io, libomnitrace, omnitrace, timemory
OMNITRACE_USE_KOKKOSP	false	backend, custom, kokkos, libomnitrace, omnitrace
OMNITRACE_USE_PERFETTO	true	backend, custom, libomnitrace, omnitrace, perfetto
OMNITRACE_USE_PID	false	custom, filename, io, libomnitrace, omnitrace
OMNITRACE_USE_PROCESS_SAMPLING	true	backend, custom, libomnitrace, omnitrace, process_sampling, sampling
OMNITRACE_USE_RCCLP	false	backend, custom, libomnitrace, omnitrace, rccl, rocm
OMNITRACE_USE_ROCM_SMI	true	backend, custom, libomnitrace, omnitrace, rocm, rocm_smi
OMNITRACE_USE_ROCPROFILER	true	backend, custom, libomnitrace, omnitrace, rocm, rocprofiler
OMNITRACE_USE_ROCTRACER	true	backend, custom, libomnitrace, omnitrace, rocm, roctracer
OMNITRACE_USE_ROCTX	false	backend, custom, libomnitrace, omnitrace, rocm, roctracer, roctx
OMNITRACE_USE_SAMPLING	false	backend, custom, libomnitrace, omnitrace, sampling
OMNITRACE_USE_TIMEMORY	true	backend, custom, libomnitrace, omnitrace, timemory
OMNITRACE_VERBOSE	0	core, debugging, libomnitrace, omnitrace, timemory

Omnitrace configuration (II)

srun -n 1 --gpus 1 omnitrace-avail --categories omnitrace --brief --description

```
[omnitrace] /proc/sys/kernel/perf_event_paranoid has a value of 3. Disabling PAPI (requires a value <= 1)...
[omnitrace] In order to enable PAPI support, run 'echo N | sudo tee /proc/sys/kernel/perf_event_paranoid' where N is < 2
```

ENVIRONMENT VARIABLE	DESCRIPTION
OMNITRACE_CONFIG_FILE	Configuration file for omnitrace
OMNITRACE_CRITICAL_TRACE	Enable generation of the critical trace
OMNITRACE_OUTPUT_PATH	Explicitly specify the output folder for results
OMNITRACE_OUTPUT_PREFIX	Explicitly specify a prefix for all output files
OMNITRACE_PERFETTO_BACKEND	Specify the perfetto backend to activate. Options are: 'inprocess', 'system', or 'all'
OMNITRACE_PERFETTO_BUFFER_SIZE_KB	Size of perfetto buffer (in KB)
OMNITRACE_PERFETTO_FILL_POLICY	Behavior when perfetto buffer is full. 'discard' will ignore new entries, 'ring_buffer' will overwrite old entries
OMNITRACE_PROCESS_SAMPLING_DURATION	If > 0.0, time (in seconds) to sample before stopping. If less than zero, uses OMNITRACE_SAMPLING_DURATION
OMNITRACE_PROCESS_SAMPLING_FREQ	Number of measurements per second when OMNITTRACE_USE_PROCESS_SAMPLING=ON. If set to zero, uses OMNITRACE_SAMPLING_FREQ value
OMNITRACE_ROCM_EVENTS	ROCm hardware counters. Use ':device=N' syntax to specify collection on device number N, e.g. ':device=0'. If no device specification is provided, the event is collected on every available device
OMNITRACE_SAMPLING_CPUS	CPUs to collect frequency information for. Values should be separated by commas and can be explicit or ranges, e.g. 0,1,5-8. An empty value implies 'all' and 'none' suppresses all CPU frequency sampling
OMNITRACE_SAMPLING_DELAY	Time (in seconds) to wait before the first sampling signal is delivered, increasing this value can fix deadlocks during init
OMNITRACE_SAMPLING_DURATION	If > 0.0, time (in seconds) to sample before stopping
OMNITRACE_SAMPLING_FREQ	Number of software interrupts per second when OMNITTRACE_USE_SAMPLING=ON
OMNITRACE_SAMPLING_GPUS	Devices to query when OMNITRACE_USE_ROCM_SMI=ON. Values should be separated by commas and can be explicit or ranges, e.g. 0,1,5-8. An empty value implies 'all' and 'none' suppresses all GPU sampling
OMNITRACE_TIMEMORY_COMPONENTS	List of components to collect via timemory (see `omnitrace-avail -C`)
OMNITRACE_TIME_OUTPUT	Output data to subfolder w/ a timestamp (see also: TIME_FORMAT)
OMNITRACE_USE_KOKKOSP	Enable support for Kokkos Tools
OMNITRACE_USE_PERFETTO	Enable perfetto backend
OMNITRACE_USE_PID	Enable tagging filenames with process identifier (either MPI rank or pid)
OMNITRACE_USE_PROCESS_SAMPLING	Enable a background thread which samples process-level and system metrics such as the CPU/GPU freq, power, memory usage, etc.
OMNITRACE_USE_RCCLP	Enable support for ROCm Communication Collectives Library (RCCL) Performance
OMNITRACE_USE_ROCM_SMI	Enable sampling GPU power, temp, utilization, and memory usage
OMNITRACE_USE_ROCPROFILER	Enable ROCm hardware counters
OMNITRACE_USE_ROCTRACER	Enable ROCm API and kernel tracing
OMNITRACE_USE_ROCTX	Enable ROCtx API. Warning! Out-of-order ranges may corrupt perfetto flamegraph
OMNITRACE_USE_SAMPLING	Enable statistical sampling of call-stack
OMNITRACE_USE_TIMEMORY	Enable timemory backend
OMNITRACE_VERBOSE	Verbosity level

Create a configuration file

- Use a name of non-existing config file

```
srun -n 1 omnitrace-avail -G omnitrace.cfg
[omnitrace-avail] Outputting text configuration file './omnitrace.cfg'...
```

- To add also description for each variable

```
srun -n 1 omnitrace-avail -G omnitrace_all.cfg --all
[omnitrace-avail] Outputting text configuration file './omnitrace_all.cfg'...
```

- Declare which cfg file to use :

```
export OMNITRACE_CONFIG_FILE=/path/omnitrace.cfg
```

Executing MatrixTranspose

- Get and compile the https://github.com/ROCm-Developer-Tools/HIP/tree/develop/samples/2_Cookbook/0_MatrixTranspose/MatrixTranspose.cpp
- Compile: hipcc –offload-arch=gfx90a –o MatrixTranspose MatrixTranspose.cpp
- Non instrumented execution:

```
time srun -n 1 --gpus 1 ./MatrixTranspose
real    0m1.245s
```

- Dynamic instrumentation

```
time srun -n 1 -gpus 1 omnitrace -- ./MatrixTranspose
[omnitrace][exe]
[omnitrace][exe] command ::
'/pfs/lustrep4/scratch/project_462000075/markoman/HIP/samples/2_Cookbook/0_MatrixTranspose/MatrixTranspose'...
[omnitrace][exe]
[omnitrace][118151][metadata]> Outputting 'omnitrace-MatrixTranspose-output/2022-10-16_22.53/metadata-118151.json' and
'omnitrace-MatrixTranspose-output/2022-10-16_22.53/functions-118151.json'
[omnitrace][118151][0][omnitrace_finalize] Finalized
[706.822]      perfetto.cc:57383 Tracing session 1 ended, total sessions:0
[omnitrace][exe] End of omnitrace
real    1m27.841s
```

Identify overhead

Command: `nm --demangle MatrixTranspose | egrep -i ' (t|u) '`

```
000000000020d080 t _GLOBAL__sub_I_MatrixTranspose.cpp
000000000020c970 T __device_stub__warmup()
000000000020ca40 T matrixTransposeCPUReference(float*, float*, unsigned int)
000000000020c9c0 T __device_stub__matrixTranspose(float*, float*, int)
                 U std::ctype<char>::_M_widen_init() const
                 U std::ostream::put(char)
                 U std::ostream::flush()
                 U std::ios_base::Init::Init()
                 U std::ios_base::Init::~Init()
                 U std::basic_ostream<char, std::char_traits<char> >& std::__ostream_insert<char, std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >&, char const*, long)
                 U std::__throw_bad_cast()
                 U __cxa_atexit
000000000020c930 t __do_global_dtors_aux
                 U __hipPopCallConfiguration
                 U __hipPushCallConfiguration
                 U __hipRegisterFatBinary
                 U __hipRegisterFunction
                 U __hipUnregisterFatBinary
000000000020cfd0 t __hip_module_ctor
000000000020d060 t __hip_module_dtor
000000000020d12e T __libc_csu_fini
000000000020d0ae T __libc_csu_init
                 U __libc_start_main
000000000020d178 t _fini
000000000020d160 t _init
000000000020c890 T _start
000000000020d14e t atexit
000000000020c8c0 t deregister_tm_clones
000000000020c960 t frame_dummy
                 U free
                 U hipFree
                 U hipGetDeviceProperties
                 U hipLaunchKernel
                 U hipMalloc
                 U hipMemcpy
000000000020cb00 T main
                 U malloc
                 U printf
                 U puts
000000000020c8f0 t register_tm_clones
                 U strlen
```

Available functions to instrument

```
srun -n 1 --gpus 1 omnitrace -v -1 --simulate --print-available functions --
./MatrixTranspose
```

```
[omnitrace][exe]
[omnitrace][exe] command :: '/pfs/lustrep4/scratch/project_462000075/markoman/HIP/samples/2_Cookbook/0_MatrixTranspose/MatrixTranspose'...
[omnitrace][exe]

[available] ../sysdeps/x86_64/crti.S:
[available]    [_fini][3]
[available]    [_init][7]

[available] ../sysdeps/x86_64/start.S:
[available]    [_start][12]

[available] /home/omnitrace/build-release/opensuse-15.2-rocm-5.0-python/external/dyninst/tbb/src/TBB-External/src/../src/old/concurrent_queue_v2.cpp:
[available]    [tbb::internal::concurrent_queue_base::concurrent_queue_base][51]
[available]    [tbb::internal::concurrent_queue_base::internal_pop][27]
[available]    [tbb::internal::concurrent_queue_base::internal_pop_if_present][50]
[available]    [tbb::internal::concurrent_queue_base::internal_push][46]
[available]    [tbb::internal::concurrent_queue_base::internal_push_if_not_full][54]
[available]    [tbb::internal::concurrent_queue_base::internal_set_capacity][5]
[available] MatrixTranspose:
[available]    [__device_stub__matrixTranspose][26]
[available]    [__device_stub__warmup][17]
[available]    [__do_global_dtors_aux][9]
[available]    [__hip_module_ctor][33]
[available]    [__hip_module_dtor][8]
[available]    [frame_dummy][4]
[available]    [targ20d132][1]

[available] MatrixTranspose.cpp:
[available]    [_GLOBAL__sub_I_MatrixTranspose.cpp][8]
[available]    [main][278]
[available]    [matrixTransposeCPUReference][51]

[available] atexit.c:
[available]    [atexit][5]

[available] elf-init.c:
[available]    [__libc_csu_fini][3]
[available]    [__libc_csu_init][36]
```

More than 36000 functions

Custom including/excluding functions

- Include functions

```
srun -n 1 --gpus 1 omnitrace -v -1 --simulate --print-available functions -I
'function_name1' 'function_name2' -- ./MatrixTranspose
```

- Exclude functions

```
srun -n 1 --gpus 1 omnitrace -v -1 --simulate --print-available functions -E
'function_name1' 'function_name2' -- ./MatrixTranspose
```

The above commands include the simulate flag that it will demonstrate the available functions but it will not run the MatrixTranspose executable

Decreasing profiling overhead

- Binary rewriting and print available functions

```
srun -n 1 --gpus 1 omnitrace -v -1 --print-available functions -o matrix.inst --
./MatrixTranspose
```

```
[omnitrace][exe]
[omnitrace][exe] command :: '/pfs/lustrep4/scratch/project_462000075/markoman/HIP/samples/2_Cookbook/0_MatrixTranspose/MatrixTranspose'...
[omnitrace][exe]
[omnitrace][exe] Resolved 'libomnitrace-rt.so' to '/pfs/lustrep4/scratch/project_462000075/markoman/omnitrace_install/lib/libomnitrace-rt.so.11.0.1'...
[omnitrace][exe] DYNINST_API_RT: /pfs/lustrep4/scratch/project_462000075/markoman/omnitrace_install/lib/libomnitrace-rt.so.11.0.1
[omnitrace][exe] instrumentation target: /pfs/lustrep4/scratch/project_462000075/markoman/HIP/samples/2_Cookbook/0_MatrixTranspose/MatrixTranspose
[omnitrace][exe] Opening '/pfs/lustrep4/scratch/project_462000075/markoman/HIP/samples/2_Cookbook/0_MatrixTranspose/MatrixTranspose' for binary rewrite... Done
[omnitrace][exe] Getting the address space image, modules, and procedures...
[omnitrace][exe]
[omnitrace][exe] Found 16 functions in 6 modules in instrumentation target
[omnitrace][exe] Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.21_PM/instrumentation/available.json'... Done
[omnitrace][exe] Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.21_PM/instrumentation/available.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.21_PM/instrumentation/overlapping.json'... Done
[omnitrace][exe] Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.21_PM/instrumentation/overlapping.txt'... Done
[omnitrace][exe] function: 'main' ... found
[omnitrace][exe] function: 'omnitrace_user_start_trace' ... not found
[omnitrace][exe] function: 'omnitrace_user_stop_trace' ... not found
[omnitrace][exe] function: 'MPI_Init' ... not found
[omnitrace][exe] function: 'MPI_Init_thread' ... not found
[omnitrace][exe] function: 'MPI_Finalize' ... not found
[omnitrace][exe] function: 'MPI_Comm_rank' ... not found
[omnitrace][exe] function: 'MPI_Comm_size' ... not found
[omnitrace][exe] Resolved 'libomnitrace-dl.so' to '/pfs/lustrep4/scratch/project_462000075/markoman/omnitrace_install/lib/libomnitrace-dl.so.1.6.0'...
[omnitrace][exe] loading library: '/pfs/lustrep4/scratch/project_462000075/markoman/omnitrace_install/lib/libomnitrace-dl.so.1.6.0'...
[omnitrace][exe] Finding instrumentation functions...
[omnitrace][exe] function: 'omnitrace_init' ... found
[omnitrace][exe] function: 'omnitrace_finalize' ... found
[omnitrace][exe] function: 'omnitrace_set_env' ... found
[omnitrace][exe] function: 'omnitrace_set_mpi' ... found
[omnitrace][exe] function: 'omnitrace_push_trace' ... found
[omnitrace][exe] function: 'omnitrace_pop_trace' ... found
[omnitrace][exe] function: 'omnitrace_register_source' ... found
[omnitrace][exe] function: 'omnitrace_register_coverage' ... found
[omnitrace][exe] Resolved 'libomnitrace-dl.so' to '/pfs/lustrep4/scratch/project_462000075/markoman/omnitrace_install/lib/libomnitrace-dl.so.1.6.0'...
[omnitrace][exe] Adding main entry snippets...
[omnitrace][exe] Adding main exit snippets...
[omnitrace][exe]
[omnitrace][exe] Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.21_PM/instrumentation/available.json'... Done
[omnitrace][exe] Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.21_PM/instrumentation/available.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.21_PM/instrumentation/instrumented.json'... Done
[omnitrace][exe] Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.21_PM/instrumentation/instrumented.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.21_PM/instrumentation/excluded.json'... Done
[omnitrace][exe] Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.21_PM/instrumentation/excluded.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.21_PM/instrumentation/overlapping.json'... Done
[omnitrace][exe] Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.21_PM/instrumentation/overlapping.txt'... Done

[instrumented] MatrixTranspose.cpp:
[instrumented]    [main][278]
```

- Default instrumentation is main function and functions of 1024 instructions and more (for CPU)
- To instrument routines with for example 50 instructions, add the option "–i 50" to instrument function of 50 instructions and above (move overhead)

Executing the new instrumented binary

```
time srun -n 1 --gpus 1 ./matrix.inst
```

```
[omnitrace][omnitrace_init_tooling] Instrumentation mode: Trace


[omnitrace] /proc/sys/kernel/perf_event_paranoid has a value of 3. Disabling PAPI (requires a value <= 1)...
[omnitrace] In order to enable PAPI support, run 'echo N | sudo tee /proc/sys/kernel/perf_event_paranoid' where N is < 2
[730.689]       perfetto.cc:55910 Configured tracing session 1, #sources:1, duration:0 ms, #buffers:1, total buffer size:1024000 KB, total sessions:1, uid:0 session name: ""

Device name
Device name
[omnitrace][91915][1][hip_activity_callback]    1 :: CopyHostToDevice     :: CopyHostToDevice     :: cid=7, time_ns=(357731149538957:357731140299748) delta=-9239209, device_id=0, stream_id=0, pid=0, tid=0
PASSED!

[omnitrace][91915][0][omnitrace_finalize] finalizing...


[omnitrace][91915][0][omnitrace_finalize] omnitrace/process/91915 : 0.471434 sec wall_clock,  217.600 MB peak_rss,  210.379 MB page_rss, 0.480000 sec cpu_clock,  101.8 % cpu_util [laps: 1]
[omnitrace][91915][0][omnitrace_finalize] omnitrace/process/91915/thread/0 : 0.471373 sec wall_clock, 0.237256 sec thread_cpu_clock,   50.3 % thread_cpu_util,  217.600 MB peak_rss [laps: 1]

[omnitrace][91915][0][omnitrace_finalize] Finalizing perfetto...
[omnitrace][91915][perfetto]> Outputting '/scratch/project_462000075/markoman/HIP/samples/2_Cookbook/0_MatrixTranspose/omnitrace-matrix.inst-output/2022-11-14_12.33_PM/perfetto-trace.proto' (1008.42 KB / 1.01 MB / 0.00 GB)... Done
[omnitrace][91915][roctracer]> Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.33_PM/roctracer.json'
[omnitrace][91915][roctracer]> Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.33_PM/roctracer.txt'
[omnitrace][91915][wall_clock]> Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.33_PM/wall_clock.json'
[omnitrace][91915][wall_clock]> Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.33_PM/wall_clock.txt'
[omnitrace][91915][manager::finalize][metadata]> Outputting 'omnitrace-matrix.inst-output/2022-11-14_12.33_PM/metadata.json' and 'omnitrace-matrix.inst-output/2022-11-14_12.33_PM/functions.json'
[omnitrace][91915][0][omnitrace_finalize] Finalized
[731.210]       perfetto.cc:57383 Tracing session 1 ended, total sessions:0

real    0m0.803s
```

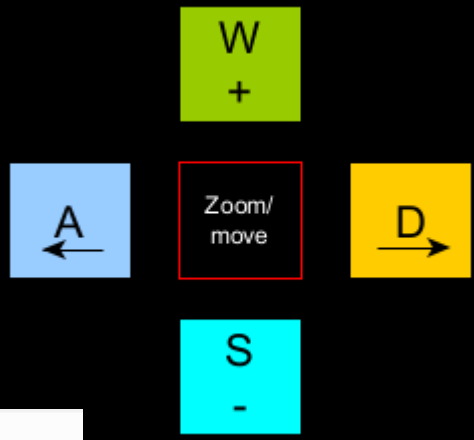
Check the list of the GPU calls instrumented

omnitrace-matrix.inst-output/2022-11-14_12.33_PM/roctracer.txt

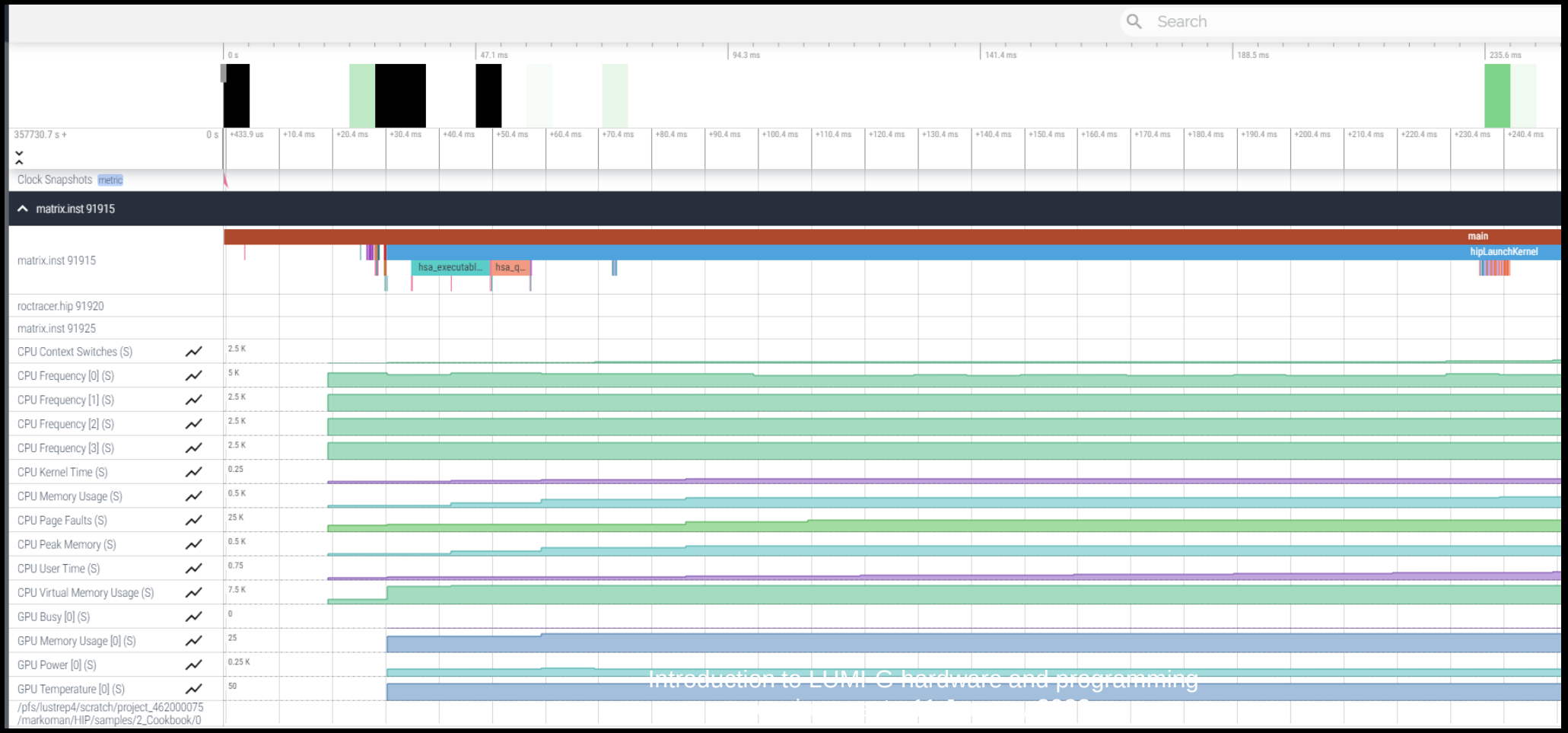
ROCM TRACER (ACTIVITY API)

LABEL	COUNT	DEPTH	METRIC	UNITS	SUM	MEAN	% SELF
|0>>> pthread_create	5	0	roctracer	sec	0.001036	0.000207	100.0
|2>>> |_start_thread	-	1	-	-	-	-	-
|2>>> |_hsa_amd_memory_pool_allocate	5	2	roctracer	sec	0.000750	0.000150	100.0
|2>>> |_hsa_iterate_agents	2	2	roctracer	sec	0.000018	0.000009	100.0
|2>>> |_hsa_amd_agents_allow_access	4	2	roctracer	sec	0.000118	0.000030	100.0
|2>>> |_hsa_agent_iterate_isas	1	2	roctracer	sec	0.000001	0.000001	100.0
|2>>> |_hsa_signal_create	15	2	roctracer	sec	0.000068	0.000005	100.0
|2>>> |_hsa_executable_load_agent_code_object	1	2	roctracer	sec	0.014825	0.014825	100.0
|2>>> |_hsa_amd_memory_lock_to_pool	3	2	roctracer	sec	0.000538	0.000179	100.0
|2>>> |_hsa_signal_silent_store_relaxed	5	2	roctracer	sec	0.000001	0.000000	100.0
|2>>> |_hsa_queue_add_write_index_screlease	3	2	roctracer	sec	0.000001	0.000000	100.0
|2>>> |_hsa_signal_store_screlease	4	2	roctracer	sec	0.000001	0.000000	100.0
|2>>> |_hsa_amd_signal_async_handler	3	2	roctracer	sec	0.000001	0.000000	100.0
|2>>> |_hsa_signal_wait_scacquire	5	2	roctracer	sec	0.009013	0.001803	100.0
|2>>> |_hsa_signal_load_relaxed	7	2	roctracer	sec	0.000003	0.000000	100.0
|2>>> |_hsa_queue_load_read_index_relaxed	2	2	roctracer	sec	0.000000	0.000000	100.0
|2>>> |_hsa_signal_destroy	1	2	roctracer	sec	0.000000	0.000000	100.0
|2>>> |_hsa_amd_memory_unlock	2	2	roctracer	sec	0.000098	0.000049	100.0
|2>>> |_hsa_queue_load_read_index_scacquire	2	2	roctracer	sec	0.000000	0.000000	100.0
|2>>> |_hsa_amd_memory_async_copy	1	2	roctracer	sec	0.000002	0.000002	100.0
|4>>> |_start_thread	-	1	-	-	-	-	-
|4>>> |_hsa_amd_memory_pool_allocate	1	2	roctracer	sec	0.000092	0.000092	100.0
|4>>> |_hsa_signal_create	11	2	roctracer	sec	0.000003	0.000000	100.0
|4>>> |_hsa_executable_load_agent_code_object	1	2	roctracer	sec	0.005452	0.005452	100.0
|4>>> |_hsa_queue_load_read_index_relaxed	1	2	roctracer	sec	0.000000	0.000000	100.0
|4>>> |_hsa_amd_memory_lock_to_pool	1	2	roctracer	sec	0.000068	0.000068	100.0
|4>>> |_hsa_queue_load_read_index_scacquire	1	2	roctracer	sec	0.000000	0.000000	100.0
|4>>> |_hsa_signal_load_relaxed	5	2	roctracer	sec	0.000001	0.000000	100.0
|4>>> |_hsa_signal_destroy	2	2	roctracer	sec	0.000000	0.000000	100.0
|4>>> |_hsa_signal_wait_scacquire	2	2	roctracer	sec	0.000182	0.000091	100.0
|4>>> |_hsa_amd_memory_unlock	1	2	roctracer	sec	0.000043	0.000043	100.0
|4>>> |_hsa_amd_memory_async_copy	1	2	roctracer	sec	0.000304	0.000304	100.0
|4>>> |_hsa_signal_store_screlease	1	2	roctracer	sec	0.000000	0.000000	100.0
|4>>> |_hsa_amd_memory_pool_free	1	2	roctracer	sec	0.000062	0.000062	100.0
|5>>> |_start_thread	-	1	-	-	-	-	-
|5>>> |_hsa_signal_create	8	2	roctracer	sec	0.000001	0.000000	100.0
|5>>> |_hsa_queue_add_write_index_screlease	1	2	roctracer	sec	0.000000	0.000000	100.0
|5>>> |_hsa_signal_store_screlease	2	2	roctracer	sec	0.000001	0.000001	100.0
|5>>> |_hsa_signal_silent_store_relaxed	2	2	roctracer	sec	0.000000	0.000000	100.0
|5>>> |_hsa_signal_load_relaxed	1	2	roctracer	sec	0.000000	0.000000	100.0
|5>>> |_hsa_amd_memory_pool_free	1	2	roctracer	sec	0.000047	0.000047	100.0
|3>>> |_start_thread	-	1	-	-	-	-	-
|3>>> |_hsa_queue_create	1	2	roctracer	sec	0.007257	0.007257	100.0
|3>>> |_hsa_signal_create	10	2	roctracer	sec	0.000003	0.000000	100.0
|3>>> |_hsa_signal_load_relaxed	3	2	roctracer	sec	0.000001	0.000000	100.0
|3>>> |_hsa_queue_load_read_index_scacquire	1	2	roctracer	sec	0.000000	0.000000	100.0
|3>>> |_hsa_queue_load_read_index_relaxed	1	2	roctracer	sec	0.000000	0.000000	100.0
|3>>> |_hsa_amd_memory_async_copy	1	2	roctracer	sec	0.000281	0.000281	100.0
|1>>> |_start_thread	-	1	-	-	-	-	-
|0>>> hipGetDeviceProperties	1	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipMalloc	2	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipLaunchKernel	2	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipMemcpy	3	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipFree	2	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> |_warmup()	1	1	roctracer	sec	0.000001	0.000001	100.0
|0>>> |_vectoradd_float(float*, float*, int)	1	1	roctracer	sec	0.000085	0.000085	100.0

Visualizing trace

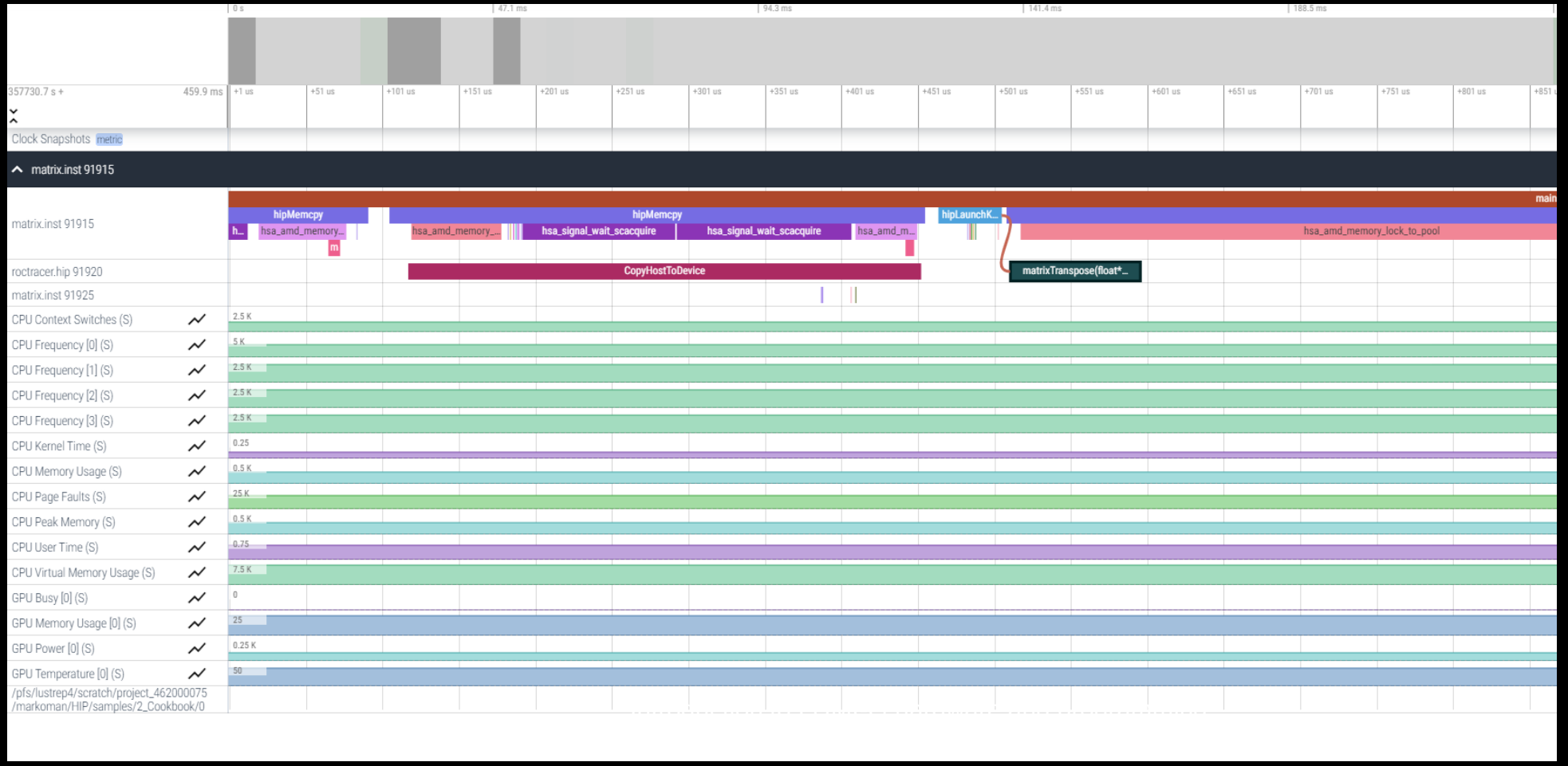


- Copy the perfetto-trace.proto to your laptop
- Go to https://ui.perfetto.dev/ click open trace and select the perfetto-trace.proto



Visualizing trace

- Copy the perfetto-trace.proto to your laptop
- Go to https://ui.perfetto.dev/ click open trace and select the perfetto-trace.proto



Hardware counters (I)

```
srun -n 1 --gpus 1 omnitrace-avail --all
```

GPU		
SQ_INSTS_VMEM_WR:device=0	true	Number of VMEM write instructions issued (including FLAT). (per-simd, emulated)
SQ_INSTS_VMEM_RD:device=0	true	Number of VMEM read instructions issued (including FLAT). (per-simd, emulated)
SQ_INSTS_SALU:device=0	true	Number of SALU instructions issued. (per-simd, emulated)
SQ_INSTS_SMEM:device=0	true	Number of SMEM instructions issued. (per-simd, emulated)
SQ_INSTS_FLAT:device=0	true	Number of FLAT instructions issued. (per-simd, emulated)
SQ_INSTS_FLAT_LDS_ONLY:device=0	true	Number of FLAT instructions issued that read/wrote only from/to LDS (only works if EARLY_TA_DONE is enabled). (per-simd, emulated)
SQ_INSTS_LDS:device=0	true	Number of LDS instructions issued (including FLAT). (per-simd, emulated)
SQ_INSTS_GDS:device=0	true	Number of GDS instructions issued. (per-simd, emulated)
SQ_WAIT_INST_LDS:device=0	true	Number of wave-cycles spent waiting for LDS instruction issue. In units of 4 cycles. (per-simd, nondeterministic)
SQ_ACTIVE_INST_VALU:device=0	true	regspec 71? Number of cycles the SQ instruction arbiter is working on a VALU instruction. (per-simd, nondeterministic)
SQ_INST_CYCLES_SALU:device=0	true	Number of cycles needed to execute non-memory read scalar operations. (per-simd, emulated)
SQ_THREAD_CYCLES_VALU:device=0	true	Number of thread-cycles used to execute VALU operations (similar to INST_CYCLES_VALU but multiplied by # of active threads). (per-simd)
SQ_LDS_BANK_CONFLICT:device=0	true	Number of cycles LDS is stalled by bank conflicts. (emulated)
TCC_HIT[0]:device=0	true	Number of cache hits.
TCC_HIT[1]:device=0	true	Number of cache hits.

...

FETCH_SIZE:device=0	true	The total kilobytes fetched from the video memory. This is measured with all extra fetches and any cache or memory effects taken into account.
WRITE_SIZE:device=0	true	The total kilobytes written to the video memory. This is measured with all extra fetches and any cache or memory effects taken into account.
WRITE_REQ_32B:device=0	true	The total number of 32-byte effective memory writes.
GPUBusy:device=0	true	The percentage of time GPU was busy.
Wavefronts:device=0	true	Total wavefronts.
VALUInsts:device=0	true	The average number of vector ALU instructions executed per work-item (affected by flow control).
SALUInsts:device=0	true	The average number of scalar ALU instructions executed per work-item (affected by flow control).
VFetchInsts:device=0	true	The average number of vector fetch instructions from the video memory executed per work-item (affected by flow control). Excludes FLAT instructions that fetch...
SFetchInsts:device=0	true	The average number of scalar fetch instructions from the video memory executed per work-item (affected by flow control).
VWriteInsts:device=0	true	The average number of vector write instructions to the video memory executed per work-item (affected by flow control). Excludes FLAT instructions that write t...
FlatVMemInsts:device=0	true	The average number of FLAT instructions that read from or write to the video memory executed per work item (affected by flow control). Includes FLAT instructi...
LDSInsts:device=0	true	The average number of LDS read or LDS write instructions executed per work item (affected by flow control). Excludes FLAT instructions that read from or writ...
FlatLDSInsts:device=0	true	The average number of FLAT instructions that read or write to LDS executed per work item (affected by flow control).
GDSInsts:device=0	true	The average number of GDS read or GDS write instructions executed per work item (affected by flow control).
VALUUtilization:device=0	true	The percentage of active vector ALU threads in a wave. A lower number can mean either more thread divergence in a wave or that the work-group size is not a mu...
VALUBusy:device=0	true	The percentage of GPUTime vector ALU instructions are processed. Value range: 0% (bad) to 100% (optimal).
SALUBusy:device=0	true	The percentage of GPUTime scalar ALU instructions are processed. Value range: 0% (bad) to 100% (optimal).
FetchSize:device=0	true	The total kilobytes fetched from the video memory. This is measured with all extra fetches and any cache or memory effects taken into account.
WriteSize:device=0	true	The total kilobytes written to the video memory. This is measured with all extra fetches and any cache or memory effects taken into account.
MemWrites32B:device=0	true	The total number of effective 32B write transactions to the memory
L2CacheHit:device=0	true	The percentage of fetch, write, atomic, and other instructions that hit the data in L2 cache. Value range: 0% (no hit) to 100% (optimal).
MemUnitBusy:device=0	true	The percentage of GPUTime the memory unit is active. The result includes the stall time (MemUnitStalled). This is measured with all extra fetches and writes a...
MemUnitStalled:device=0	true	The percentage of GPUTime the memory unit is stalled. Try reducing the number or size of fetches and writes if possible. Value range: 0% (optimal) to 100% (bad).
WriteUnitStalled:device=0	true	The percentage of GPUTime the write unit is stalled. Value range: 0% (optimal) to 100% (bad).
ALUStalledByLDS:device=0	true	The percentage of GPUTime ALU units are stalled by the LDS input queue being full or the output queue being not ready. If there are LDS bank conflicts, reduce...
LDSBankConflict:device=0	true	The percentage of GPUTime LDS is stalled by bank conflicts. Value range: 0% (optimal) to 100% (bad).

Commonly Used Counters

- VALUUtilization: The percentage of ALUs active in a wave. Low VALUUtilization is likely due to high divergence or a poorly sized grid
- VALUBusy: The percentage of GPUTime vector ALU instructions are processed. Can be thought of as something like compute utilization
- FetchSize: The total kilobytes fetched from global memory
- WriteSize: The total kilobytes written to global memory
- L2CacheHit: The percentage of fetch, write, atomic, and other instructions that hit the data in L2 cache
- MemUnitBusy: The percentage of GPUTime the memory unit is active. The result includes the stall time
- MemUnitStalled: The percentage of GPUTime the memory unit is stalled
- WriteUnitStalled: The percentage of GPUTime the write unit is stalled

Full list at: https://github.com/ROCm-Developer-Tools/rocprofiler/blob/amd-master/test/tool/metrics.xml

Hardware counters (II)

- Declare in your cfg file the metrics you want to profile
- For example, profile metrics only for the GPU with id 0:

```
OMNITRACE_ROCM_EVENTS = GPUBusy:device=0,Wavefronts:device=0,
VALUBusy:device=0,L2CacheHit:device=0,MemUnitBusy:device=0
```

- Profile for all the participated GPUs:

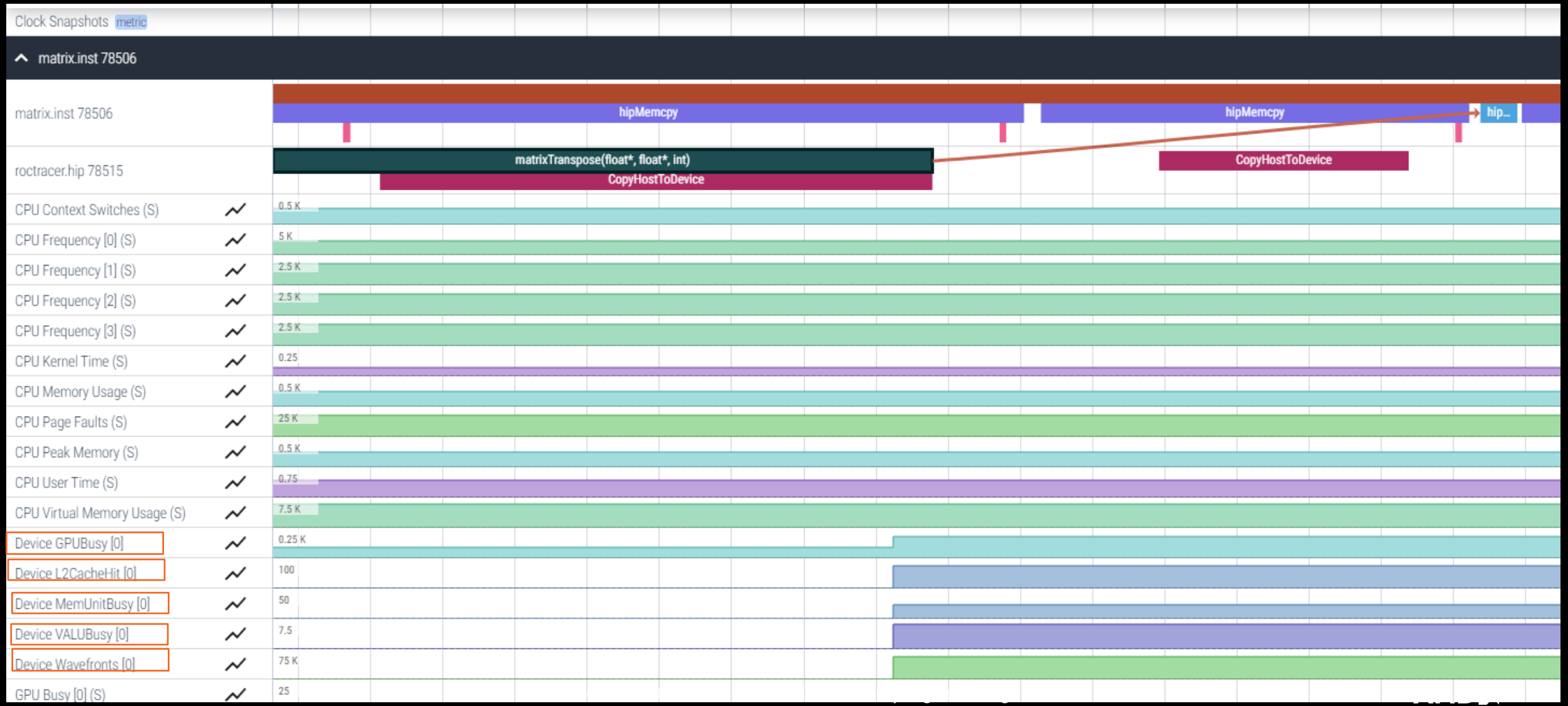
```
OMNITRACE_ROCM_EVENTS = GPUBusy,Wavefronts,VALUBusy,L2CacheHit,MemUnitBusy
```

Execution with hardware counters

```
srun -n 1 --gpus 1 ./matrix.inst
```

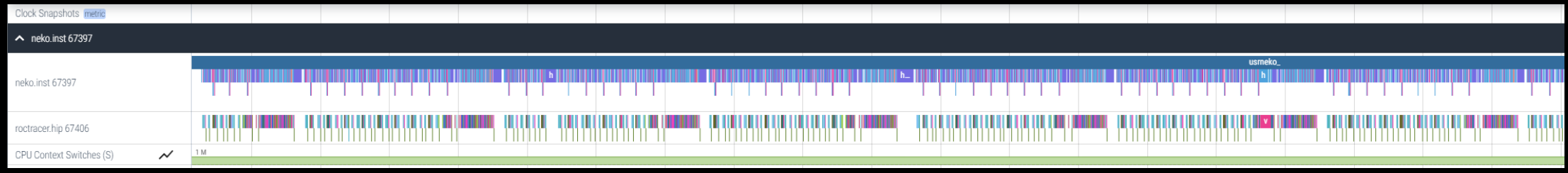
```
[omnitrace] /proc/sys/kernel/perf_event_paranoid has a value of 3. Disabling PAPI (requires a value <= 2)...
[omnitrace] In order to enable PAPI support, run 'echo N | sudo tee /proc/sys/kernel/perf_event_paranoid' where N is <= 2
[297.589]       perfetto.cc:55910 Configured tracing session 1, #sources:1, duration:0 ms, #buffers:1, total buffer size:1024000 KB, total sessions:1, uid:0 session name: ""
Device name
Device name

PASSED!
[omnitrace][78506][0][omnitrace_finalize] finalizing...
[omnitrace][78506][0][omnitrace_finalize]
[omnitrace][78506][0][omnitrace_finalize] omnitrace/process/78506 : 0.717209 sec wall_clock,  219.768 MB peak_rss,  212.754 MB page_rss, 0.740000 sec cpu_clock,  103.2 % cpu_util [laps: 1]
[omnitrace][78506][0][omnitrace_finalize] omnitrace/process/78506/thread/0 : 0.715605 sec wall_clock, 0.233719 sec thread_cpu_clock,   32.7 % thread_cpu_util,  219.768 MB peak_rss [laps: 1]
[omnitrace][78506][0][omnitrace_finalize]
[omnitrace][78506][0][omnitrace_finalize] Finalizing perfetto...
[omnitrace][78506][perfetto]> Outputting '/scratch/project_462000075/markoman/HIP/samples/2_Cookbook/0_MatrixTranspose/omnitrace-matrix.inst-output/2022-11-16_00.45/perfetto-trace.proto' (96.15 KB / 0.10 MB / 0.00 GB)... Done
[omnitrace][78506][0][omnitrace_finalize] Finalization metrics: 0.137393 sec wall_clock,    0.000 MB peak_rss,    1.085 MB page_rss, 0.130000 sec cpu_clock,   94.6 % cpu_util
[omnitrace][78506][rocprof-device-0-GPUBusy]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/rocprof-device-0-GPUBusy.json'
[omnitrace][78506][rocprof-device-0-GPUBusy]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/rocprof-device-0-GPUBusy.txt'
[omnitrace][78506][rocprof-device-0-Wavefronts]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/rocprof-device-0-Wavefronts.json'
[omnitrace][78506][rocprof-device-0-Wavefronts]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/rocprof-device-0-Wavefronts.txt'
[omnitrace][78506][rocprof-device-0-VALUBusy]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/rocprof-device-0-VALUBusy.json'
[omnitrace][78506][rocprof-device-0-VALUBusy]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/rocprof-device-0-VALUBusy.txt'
[omnitrace][78506][rocprof-device-0-L2CacheHit]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/rocprof-device-0-L2CacheHit.json'
[omnitrace][78506][rocprof-device-0-L2CacheHit]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/rocprof-device-0-L2CacheHit.txt'
[omnitrace][78506][rocprof-device-0-MemUnitBusy]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/rocprof-device-0-MemUnitBusy.json'
[omnitrace][78506][rocprof-device-0-MemUnitBusy]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/rocprof-device-0-MemUnitBusy.txt'
[omnitrace][78506][roctracer]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/roctracer.json'
[omnitrace][78506][roctracer]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/roctracer.txt'
[omnitrace][78506][sampling_gpu_memory_usage]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/sampling_gpu_memory_usage.json'
[omnitrace][78506][sampling_gpu_memory_usage]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/sampling_gpu_memory_usage.txt'
[omnitrace][78506][sampling_gpu_power]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/sampling_gpu_power.json'
[omnitrace][78506][sampling_gpu_power]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/sampling_gpu_power.txt'
[omnitrace][78506][sampling_gpu_temperature]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/sampling_gpu_temperature.json'
[omnitrace][78506][sampling_gpu_temperature]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/sampling_gpu_temperature.txt'
[omnitrace][78506][sampling_gpu_busy_percent]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/sampling_gpu_busy_percent.json'
[omnitrace][78506][sampling_gpu_busy_percent]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/sampling_gpu_busy_percent.txt'
[omnitrace][78506][wall_clock]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/wall_clock.json'
[omnitrace][78506][wall_clock]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/wall_clock.txt'
[omnitrace][78506][metadata]> Outputting 'omnitrace-matrix.inst-output/2022-11-16_00.45/metadata-78506.json'
[omnitrace][78506][0][omnitrace_finalize] Finalized
[303.572]       perfetto.cc:57383 Tracing session 1 ended, total sessions:0
```

Visualization with hardware counters

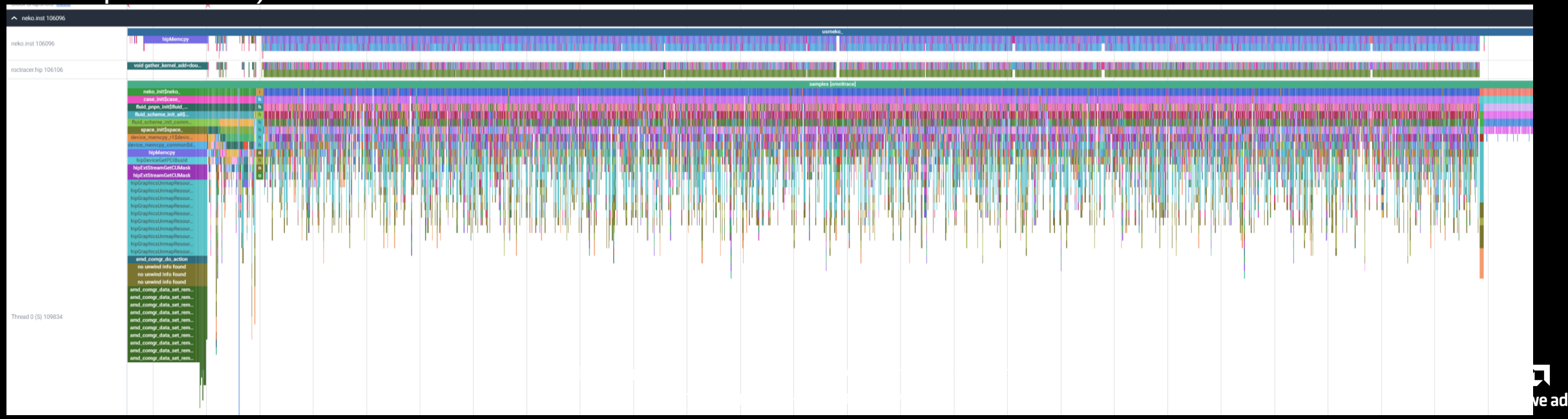


Sampling call-stack (I)

- Another application with OMNITRACE_USE_SAMPLING = false

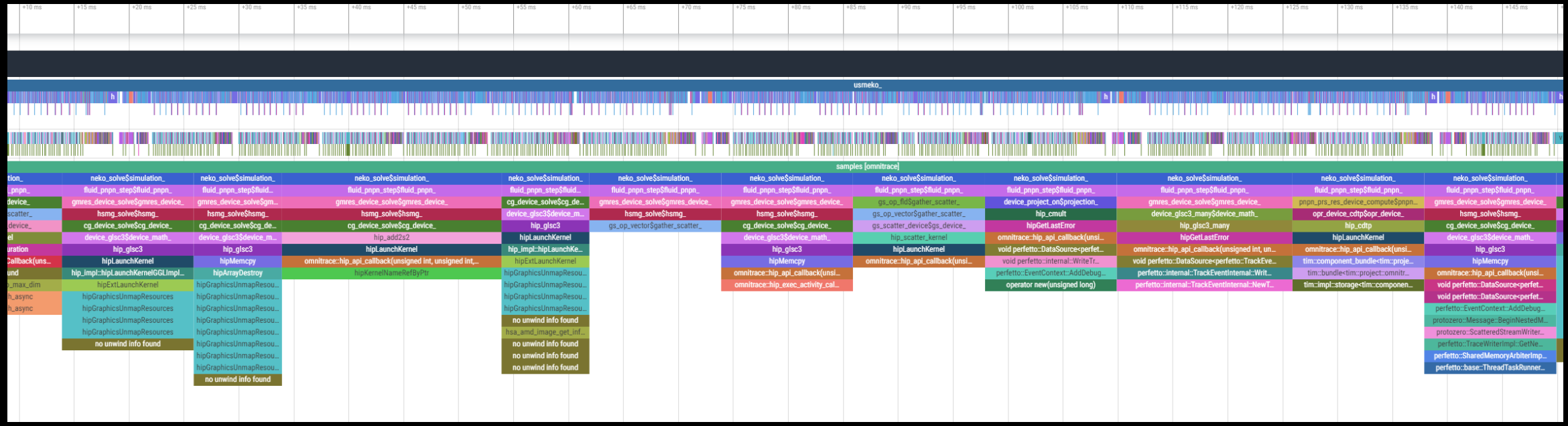


- With OMNITRACE_USE_SAMPLING = true and OMNITRACE_SAMPLING_FREQ = 100 (100 samples per second)



Sampling call-stack (II)

- Zoom in call-stack sampling



How to see kernels timing?

- omnitrace-binary-output/timestamp/wall_clock.txt

REAL-CLOCK TIMER (I.E. WALL-CLOCK TIMER)

LABEL	COUNT	DEPTH	METRIC	UNITS	SUM	MEAN	MIN	MAX	VAR	STDDEV	% SELF
|0>>> main	1	0	wall_clock	sec	21.811922	21.811922	21.811922	21.811922	0.000000	0.000000	46.3
|0>>> |_mbind	23	1	wall_clock	sec	0.000041	0.000002	0.000001	0.000004	0.000000	0.000001	100.0
|0>>> |_pthread_create	1	1	wall_clock	sec	0.023345	0.023345	0.023345	0.023345	0.000000	0.000000	100.0
|1>>> |_start_thread	-	2	-	-	-	-	-	-	-	-	-
|0>>> |_hipDeviceGetName	1	1	wall_clock	sec	0.001030	0.001030	0.001030	0.001030	0.000000	0.000000	100.0
|0>>> |_hipMalloc	1076	1	wall_clock	sec	0.019050	0.000018	0.000001	0.000583	0.000000	0.000046	100.0
|0>>> |_hipMemcpy	92578	1	wall_clock	sec	6.052626	0.000065	0.000001	0.181018	0.000000	0.000605	99.7
|0>>> |_mbind	146	2	wall_clock	sec	0.000167	0.000001	0.000001	0.000003	0.000000	0.000001	100.0
|0>>> |_void gather_kernel_add<double>(double*, int, int, int const*, double const*, int, int const*, int, int cons...	52100	2	wall_clock	sec	0.001629	0.000000	0.000000	0.000006	0.000000	0.000000	100.0
|0>>> |_void scatter_kernel<double>(double*, int, int const*, double*, int, int const*, int, int const*, int const*)	52106	2	wall_clock	sec	0.002148	0.000000	0.000000	0.000248	0.000000	0.000001	100.0
|0>>> |_void coef_generate_dxyz_kernel<double, 8, 1024>(double*, double*, double*, double*, double*, double*, doubl...	1	2	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> |_void coef_generate_drst_kernel<double>(double*, double*, double*, double*, double*, double*, double*, doubl...	3	2	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> |_void coef_generate_geo_kernel<double, 8, 1024>(double*, double*, double*, double*, double*, double*, double...	1	2	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> |_void invcol1_kernel<double>(double*, int)	509	2	wall_clock	sec	0.000016	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> |_void glsum_kernel<double>(double const*, double*, int)	3	2	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> |_void reduce_kernel<double>(double*, int)	78705	2	wall_clock	sec	0.003255	0.000000	0.000000	0.000001	0.000000	0.000000	100.0

How to see kernels timing? (II)

- Add/edit in your omnitrace.cfg file, OMNITRACE_USE_TIMEMORY = true and OMNITRACE_FLAT_PROFILE = true

REAL-CLOCK TIMER (I.E. WALL-CLOCK TIMER)

LABEL	COUNT	DEPTH	METRIC	UNITS	SUM	MEAN	MIN	MAX	VAR	STDDEV	% SELF
|0>>> usrneko_	1	0	wall_clock	sec	24.024075	24.024075	24.024075	24.024075	0.000000	0.000000	100.0
|0>>> mbind	580	0	wall_clock	sec	0.000540	0.000001	0.000000	0.000004	0.000000	0.000000	100.0
|0>>> pthread_create	1	0	wall_clock	sec	0.006690	0.006690	0.006690	0.006690	0.000000	0.000000	100.0
|0>>> hipDeviceGetName	1	0	wall_clock	sec	0.000632	0.000632	0.000632	0.000632	0.000000	0.000000	100.0
|0>>> hipMalloc	1076	0	wall_clock	sec	0.029519	0.000027	0.000001	0.000373	0.000000	0.000061	100.0
|0>>> hipMemcpy	92578	0	wall_clock	sec	6.805347	0.000074	0.000001	0.621693	0.000004	0.002046	100.0
|0>>> hipDeviceSynchronize	20	0	wall_clock	sec	0.020044	0.001002	0.000002	0.002453	0.000000	0.000698	100.0
|0>>> hipLaunchKernel	510053	0	wall_clock	sec	4.547851	0.000009	0.000004	0.014506	0.000000	0.000030	100.0
|0>>> hipGetLastError	510053	0	wall_clock	sec	0.762807	0.000001	0.000001	0.031479	0.000000	0.000055	100.0
|0>>> void gather_kernel_add<double>(double*, int, int, int const*, double const*, int, int const*, int, int const*, ...	54121	0	wall_clock	sec	0.001754	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> void scatter_kernel<double>(double*, int, int const*, double*, int, int const*, int, int const*, int const*)	54121	0	wall_clock	sec	0.002088	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> hipFree	937	0	wall_clock	sec	0.016387	0.000017	0.000002	0.001981	0.000000	0.000097	100.0
|0>>> hip_coef_generate_dxyzdrst	3	0	wall_clock	sec	0.006214	0.002071	0.000063	0.006060	0.000012	0.003455	100.0
|0>>> void coef_generate_dxyz_kernel<double, 8, 1024>(double*, double*, double*, double*, double*, double*, double*, ...	1	0	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> void coef_generate_drst_kernel<double>(double*, double*, double*, double*, double*, double*, double*, double*, ...	3	0	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> hip_coef_generate_geo	3	0	wall_clock	sec	0.000125	0.000042	0.000032	0.000055	0.000000	0.000012	100.0
|0>>> void coef_generate_geo_kernel<double, 8, 1024>(double*, double*, double*, double*, double*, double*, double con...	1	0	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> void invcol1_kernel<double>(double*, int)	509	0	wall_clock	sec	0.000017	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> hipHostMalloc	16	0	wall_clock	sec	0.000871	0.000054	0.000035	0.000071	0.000000	0.000014	100.0
|0>>> void glsum_kernel<double>(double const*, double*, int)	3	0	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> void reduce_kernel<double>(double*, int)	78719	0	wall_clock	sec	0.003757	0.000000	0.000000	0.000271	0.000000	0.000001	100.0
|0>>> void cfill_kernel<double>(double*, double, int)	2014	0	wall_clock	sec	0.000066	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> void jacobi_kernel<double, 8>(double*, double const*, double const*, double const*, double const*, double const...	502	0	wall_clock	sec	0.000016	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> void col2_kernel<double>(double*, double const*, int)	10501	0	wall_clock	sec	0.000915	0.000000	0.000000	0.000574	0.000000	0.000006	100.0
|0>>> void coef_generate_dxyz_kernel<double, 2, 1024>(double*, double*, double*, double*, double*, double*, double*, ...	1	0	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> void coef_generate_geo_kernel<double, 2, 1024>(double*, double*, double*, double*, double*, double*, double con...	1	0	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> void coef_generate_dxyz_kernel<double, 4, 1024>(double*, double*, double*, double*, double*, double*, double*, ...	1	0	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> void coef_generate_geo_kernel<double, 4, 1024>(double*, double*, double*, double*, double*, double*, double con...	1	0	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> hipMemcpyAsync	10012	0	wall_clock	sec	0.081116	0.000008	0.000004	0.000568	0.000000	0.000011	100.0
|0>>> void jacobi_kernel<double, 2>(double*, double const*, double const*, double const*, double const*, double const...	1	0	wall_clock	sec	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	100.0
|0>>> void tnsr3d_kernel<double>(double*, int, double const*, int, double const*, double const*, double const*)	11011	0	wall_clock	sec	0.000388	0.000000	0.000000	0.000003	0.000000	0.000000	100.0
|0>>> void cfl_kernel<double, 8, 1024>(double, double const*, double const*, double const*, double const*, double con...	501	0	wall_clock	sec	0.000028	0.000000	0.000000	0.000000	0.000000	0.000000	100.0

User API

- Omnitrace provides an API to control the instrumentation

API Call	Description
`int` **`omnitrace_user_start_trace(void)`**	Enable tracing on this thread and all subsequently created threads
`int` **`omnitrace_user_stop_trace(void)`**	Disable tracing on this thread and all subsequently created threads
`int` **`omnitrace_user_start_thread_trace(void)`**	Enable tracing on this specific thread. Does not apply to subsequently created threads
`int` **`omnitrace_user_stop_thread_trace(void)`**	Disable tracing on this specific thread. Does not apply to subsequently created threads

All the API calls: https://amdresearch.github.io/omnitrace/user_api.html

MPI

- We use the example omnitrace/examples/mpi/mpi.cpp
- Compile and run it to check the output, then create an instrumented binary

```
srun -n 1 omnitrace -o mpi.inst -- ./mpi
srun -n 2 ./mpi.inst
```

MPI 0

REAL-CLOCK TIMER (I.E. WALL-CLOCK TIMER)

LABEL	COUNT	DEPTH	METRIC	UNITS	SUM	MEAN	MIN	MAX	VAR	STDDEV	% SELF
|0>>> main	1	0	wall_clock	sec	2.308613	2.308613	2.308613	2.308613	0.000000	0.000000	86.7
|0>>> |_MPI_Init_thread	1	1	wall_clock	sec	0.298743	0.298743	0.298743	0.298743	0.000000	0.000000	99.5
|0>>> |_mbind	10	2	wall_clock	sec	0.000011	0.000001	0.000002	0.000000	0.000001	100.0	
|0>>> |_pthread_create	2	2	wall_clock	sec	0.001410	0.000705	0.000564	0.000847	0.000000	0.000200	0.0
|2>>> |_start_thread	1	3	wall_clock	sec	0.195632	0.195632	0.195632	0.195632	0.000000	0.000000	100.0
|1>>> |_start_thread	-	3	-	-	-	-	-	-	-	-	-
|0>>> |_pthread_create	1	1	wall_clock	sec	0.001182	0.001182	0.001182	0.001182	0.000000	0.000000	0.0
|3>>> |_start_thread	1	2	wall_clock	sec	0.002902	0.002902	0.002902	0.002902	0.000000	0.000000	62.7
|3>>> |_MPI_Comm_size	13	3	wall_clock	sec	0.000031	0.000002	0.000000	0.000024	0.000000	0.000006	100.0
|3>>> |_MPI_Comm_rank	5	3	wall_clock	sec	0.000004	0.000001	0.000000	0.000000	0.000001	100.0	
|3>>> |_MPI_Barrier	6	3	wall_clock	sec	0.000972	0.00					
|3>>> |_MPI_Send	8	3	wall_clock	sec	0.000017	0.00					
|3>>> |_MPI_Recv	8	3	wall_clock	sec	0.000021	0.00					
|3>>> |_MPI_Alltoall	8	3	wall_clock	sec	0.000030	0.00					
|3>>> |_MPI_Comm_dup	1	3	wall_clock	sec	0.000008	0.00					
|0>>> |_pthread_join	2	1	wall_clock	sec	0.007953	0.00					

MPI 1

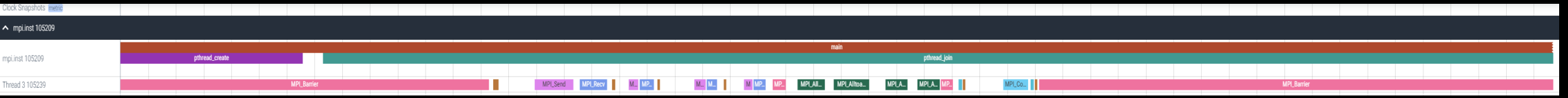
REAL-CLOCK TIMER (I.E. WALL-CLOCK TIMER)

LABEL	COUNT	DEPTH	METRIC	UNITS	SUM	MEAN	MIN	MAX	VAR	STDDEV	% SELF
|0>>> main	1	0	wall_clock	sec	2.306350	2.306350	2.306350	2.306350	0.000000	0.000000	86.8
|0>>> |_MPI_Init_thread	1	1	wall_clock	sec	0.293291	0.293291	0.293291	0.293291	0.000000	0.000000	99.2
|0>>> |_mbind	10	2	wall_clock	sec	0.000014	0.000001	0.000001	0.000004	0.000000	0.000001	100.0
|0>>> |_pthread_create	2	2	wall_clock	sec	0.002338	0.001169	0.000897	0.001441	0.000000	0.000384	0.0
|2>>> |_start_thread	1	3	wall_clock	sec	0.193902	0.193902	0.193902	0.193902	0.000000	0.000000	100.0
|1>>> |_start_thread	-	3	-	-	-	-	-	-	-	-	-
|0>>> |_pthread_create	1	1	wall_clock	sec	0.006592	0.006592	0.006592	0.006592	0.000000	0.000000	0.0
|3>>> |_start_thread	1	2	wall_clock	sec	0.007850	0.007850	0.007850	0.007850	0.000000	0.000000	16.4
|3>>> |_MPI_Comm_size	13	3	wall_clock	sec	0.000031	0.000002	0.000000	0.000024	0.000000	0.000007	100.0
|3>>> |_MPI_Comm_rank	5	3	wall_clock	sec	0.000009	0.000002	0.000000	0.000006	0.000000	0.000002	100.0
|3>>> |_MPI_Barrier	6	3	wall_clock	sec	0.006405	0.001068	0.000001	0.005604	0.000005	0.002244	100.0
|3>>> |_MPI_Send	8	3	wall_clock	sec	0.000020	0.000003	0.000001	0.000012	0.000000	0.000004	100.0
|3>>> |_MPI_Recv	8	3	wall_clock	sec	0.000027	0.000003	0.000002	0.000009	0.000000	0.000002	100.0
|3>>> |_MPI_Alltoall	8	3	wall_clock	sec	0.000060	0.000007	0.000003	0.000011	0.000000	0.000003	100.0
|3>>> |_MPI_Comm_dup	1	3	wall_clock	sec	0.000008	0.000008	0.000008	0.000008	0.000000	0.000000	100.0
|0>>> |_pthread_join	2	1	wall_clock	sec	0.005277	0.002638	0.001800	0.003477	0.000001	0.001186	100.0

MPI visualizing one Perfetto per MPI process

MPI 0

MPI 1

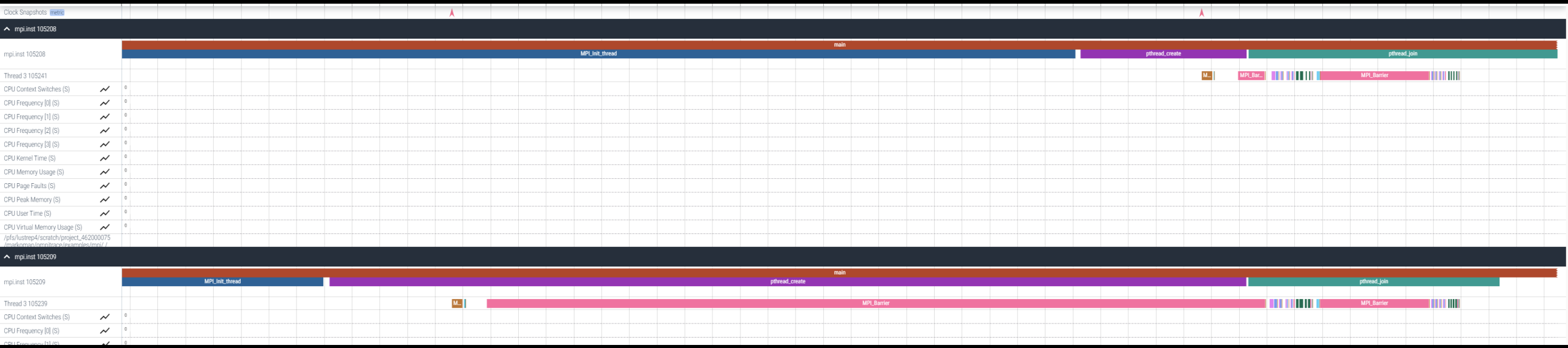


Visualizing all the MPI processes in once

- Merge the Perfetto files:

```
cat omnitrace-mpi.inst-output/timestamp/perfetto-trace-0.proto omnitrace-mpi.inst-output/timestamp/perfetto-trace-1.proto > allprocesses.proto
```

- For large number or processes a different approach is required if willing to visualize many processes



OpenMP®

- We use the example /omnitrace/examples/openmp/
- Build the code:

```
cmake –B build .
```

- We use the openmp-lu binary, execution:

```
export OPENMP_NUM_THREADS=4
srun –n 1 –c 4 ./openmp-lu
```

- Create a new instrumented binary:

```
srun -n 1 omnitrace -o openmp-lu.inst -- ./openmp-lu
```

OpenMP® (II)

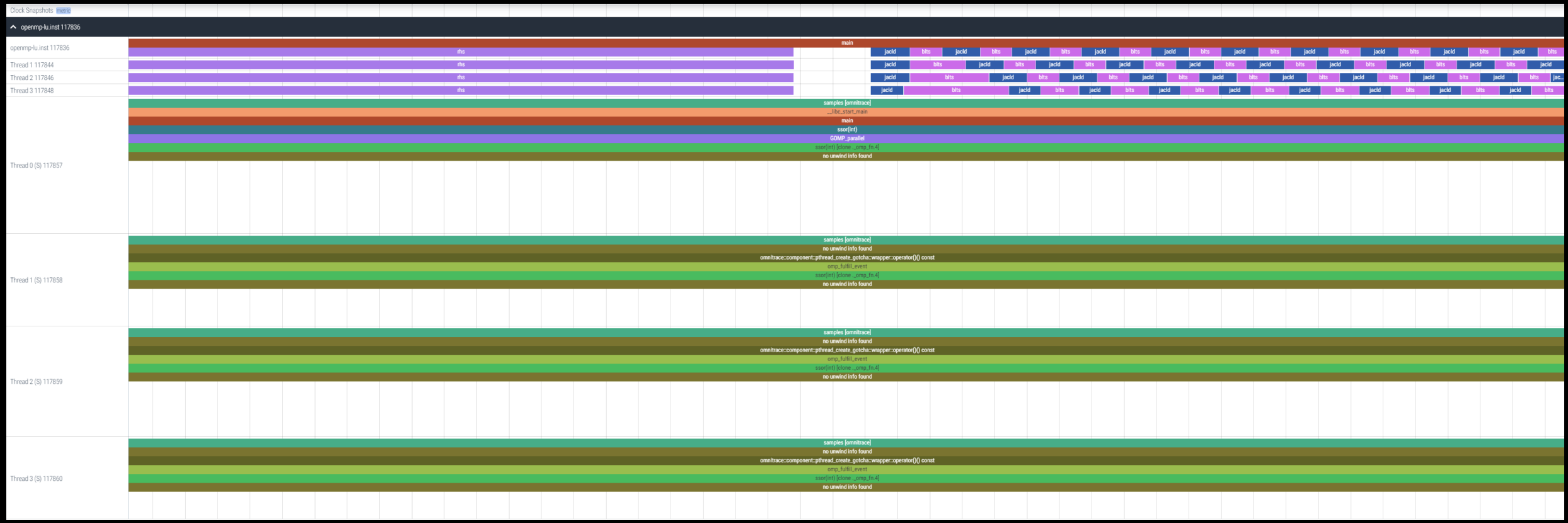
- Execution:

srun -n 1 –c 4 ./openmp-lu.inst

REAL-CLOCK TIMER (I.E. WALL-CLOCK TIMER)

LABEL	COUNT	DEPTH	METRIC	UNITS	SUM	MEAN	MIN	MAX	VAR	STDDEV	% SELF
|0>>> main	1	0	wall_clock	sec	1.096702	1.096702	1.096702	1.096702	0.000000	0.000000	9.2
|0>>> |_pthread_create	3	1	wall_clock	sec	0.002931	0.000977	0.000733	0.001420	0.000000	0.000385	0.0
|3>>> |_start_thread	1	2	wall_clock	sec	2.451520	2.451520	2.451520	2.451520	0.000000	0.000000	57.7
|3>>> |_erhs	1	3	wall_clock	sec	0.001906	0.001906	0.001906	0.001906	0.000000	0.000000	100.0
|3>>> |_rhs	153	3	wall_clock	sec	0.229893	0.001503	0.001410	0.001893	0.000000	0.000116	100.0
|3>>> |_jacld	3473	3	wall_clock	sec	0.170568	0.000049	0.000047	0.000135	0.000000	0.000005	100.0
|3>>> |_blts	3473	3	wall_clock	sec	0.232512	0.000067	0.000040	0.000959	0.000000	0.000034	100.0
|3>>> |_jacu	3473	3	wall_clock	sec	0.166229	0.000048	0.000046	0.000148	0.000000	0.000005	100.0
|3>>> |_buts	3473	3	wall_clock	sec	0.236484	0.000068	0.000041	0.000391	0.000000	0.000031	100.0
|2>>> |_start_thread	1	2	wall_clock	sec	2.452309	2.452309	2.452309	2.452309	0.000000	0.000000	58.1
|2>>> |_erhs	1	3	wall_clock	sec	0.001895	0.001895	0.001895	0.001895	0.000000	0.000000	100.0
|2>>> |_rhs	153	3	wall_clock	sec	0.229776	0.001502	0.001410	0.001893	0.000000	0.000115	100.0
|2>>> |_jacld	3473	3	wall_clock	sec	0.204609	0.000059	0.000057	0.000152	0.000000	0.000006	100.0
|2>>> |_blts	3473	3	wall_clock	sec	0.192986	0.000056	0.000047	0.000358	0.000000	0.000026	100.0
|2>>> |_jacu	3473	3	wall_clock	sec	0.199029	0.000057	0.000055	0.000188	0.000000	0.000007	100.0
|2>>> |_buts	3473	3	wall_clock	sec	0.198972	0.000057	0.000048	0.000372	0.000000	0.000026	100.0
|1>>> |_start_thread	1	2	wall_clock	sec	2.453072	2.453072	2.453072	2.453072	0.000000	0.000000	58.6
|1>>> |_erhs	1	3	wall_clock	sec	0.001905	0.001905	0.001905	0.001905	0.000000	0.000000	100.0
|1>>> |_rhs	153	3	wall_clock	sec	0.229742	0.001502	0.001410	0.001894	0.000000	0.000115	100.0
|1>>> |_jacld	3473	3	wall_clock	sec	0.206418	0.000059	0.000057	0.000934	0.000000	0.000016	100.0
|1>>> |_blts	3473	3	wall_clock	sec	0.186097	0.000054	0.000047	0.000344	0.000000	0.000023	100.0
|1>>> |_jacu	3473	3	wall_clock	sec	0.198689	0.000057	0.000055	0.000186	0.000000	0.000006	100.0
|1>>> |_buts	3473	3	wall_clock	sec	0.192470	0.000055	0.000048	0.000356	0.000000	0.000022	100.0
|0>>> |_erhs	1	1	wall_clock	sec	0.001961	0.001961	0.001961	0.001961	0.000000	0.000000	100.0
|0>>> |_rhs	153	1	wall_clock	sec	0.229889	0.001503	0.001410	0.001891	0.000000	0.000116	100.0
|0>>> |_jacld	3473	1	wall_clock	sec	0.208903	0.000060	0.000057	0.000359	0.000000	0.000017	100.0
|0>>> |_blts	3473	1	wall_clock	sec	0.172646	0.000050	0.000047	0.000822	0.000000	0.000020	100.0
|0>>> |_jacu	3473	1	wall_clock	sec	0.202130	0.000058	0.000055	0.000350	0.000000	0.000016	100.0
|0>>> |_buts	3473	1	wall_clock	sec	0.176975	0.000051	0.000048	0.000377	0.000000	0.000016	100.0
|0>>> |_pintgr	1	1	wall_clock	sec	0.000054	0.000054	0.000054	0.000054	0.000000	0.000000	100.0

OpenMP® visualization



Python™

- The omnitrace Python package is installed in /path/omnitrace_install/lib/pythonX.Y/site-packages/omnitrace
- Setup the environment

```
export PYTHONPATH=/path/omnitrace/lib/python/site-packages/:${PYTHONPATH}
```

- We use the Fibonacci example:

omnitrace/examples/python/source.py

- Execute:

```
srun -n 1 --gpus 1 omnitrace-python ./external.py
```

There will be a new directory called omnitrace-source-output with contents

Python documentation: https://amdresearch.github.io/omnitrace/python.html

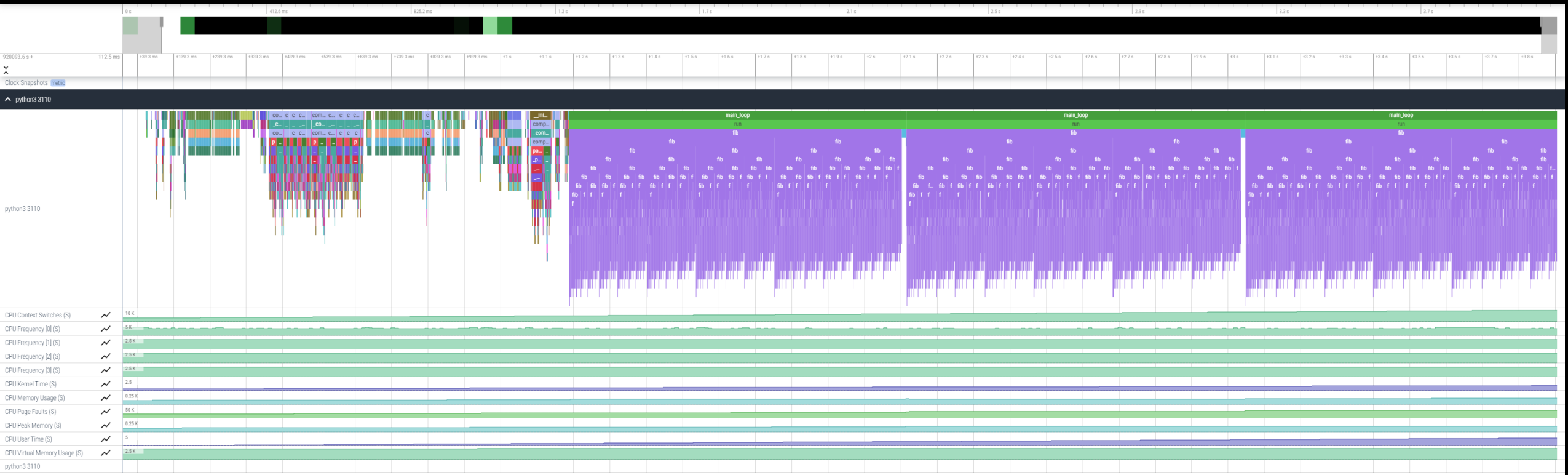
Python™ (II)

- omnitrace-source-output/timestamp/wall_clock.txt

REAL-CLOCK TIMER (I.E. WALL-CLOCK TIMER)

LABEL	COUNT	DEPTH	METRIC	UNITS	SUM	MEAN	MIN	MAX	VAR	STDDEV	% SELF
|0>>> main_loop	3	0	wall_clock	sec	2.786075	0.928692	0.926350	0.932130	0.000009	0.003042	0.0
|0>>> |_run	3	1	wall_clock	sec	2.785799	0.928600	0.926250	0.932037	0.000009	0.003043	0.0
|0>>> |_fib	3	2	wall_clock	sec	2.750104	0.916701	0.914454	0.919577	0.000007	0.002619	0.0
|0>>> |_fib	6	3	wall_clock	sec	2.749901	0.458317	0.348962	0.567074	0.013958	0.118145	0.0
|0>>> |_fib	12	4	wall_clock	sec	2.749511	0.229126	0.133382	0.350765	0.006504	0.080650	0.0
|0>>> |_fib	24	5	wall_clock	sec	2.748734	0.114531	0.050867	0.217030	0.002399	0.048977	0.1
|0>>> |_fib	48	6	wall_clock	sec	2.747118	0.057232	0.019302	0.134596	0.000806	0.028396	0.1
|0>>> |_fib	96	7	wall_clock	sec	2.743922	0.028583	0.007181	0.083350	0.000257	0.016026	0.2
|0>>> |_fib	192	8	wall_clock	sec	2.737564	0.014258	0.002690	0.051524	0.000079	0.008887	0.5
|0>>> |_fib	384	9	wall_clock	sec	2.724966	0.007096	0.000973	0.031798	0.000024	0.004865	0.9
|0>>> |_fib	768	10	wall_clock	sec	2.699251	0.003515	0.000336	0.019670	0.000007	0.002637	1.9
|0>>> |_fib	1536	11	wall_clock	sec	2.648006	0.001724	0.000096	0.012081	0.000002	0.001417	3.9
|0>>> |_fib	3072	12	wall_clock	sec	2.545260	0.000829	0.000016	0.007461	0.000001	0.000758	8.0
|0>>> |_fib	6078	13	wall_clock	sec	2.342276	0.000385	0.000016	0.004669	0.000000	0.000404	16.0
|0>>> |_fib	10896	14	wall_clock	sec	1.967475	0.000181	0.000015	0.002752	0.000000	0.000218	28.6
|0>>> |_fib	15060	15	wall_clock	sec	1.404069	0.000093	0.000015	0.001704	0.000000	0.000123	43.6
|0>>> |_fib	14280	16	wall_clock	sec	0.791873	0.000055	0.000015	0.001044	0.000000	0.000076	58.3
|0>>> |_fib	8826	17	wall_clock	sec	0.330189	0.000037	0.000015	0.000620	0.000000	0.000050	70.9
|0>>> |_fib	3456	18	wall_clock	sec	0.096120	0.000028	0.000015	0.000380	0.000000	0.000034	81.0
|0>>> |_fib	822	19	wall_clock	sec	0.018294	0.000022	0.000015	0.000209	0.000000	0.000024	88.9
|0>>> |_fib	108	20	wall_clock	sec	0.002037	0.000019	0.000016	0.000107	0.000000	0.000015	94.9
|0>>> |_fib	6	21	wall_clock	sec	0.000104	0.000017	0.000016	0.000019	0.000000	0.000001	100.0
|0>>> |_inefficient	3	2	wall_clock	sec	0.035450	0.011817	0.010096	0.012972	0.000002	0.001519	95.8
|0>>> |__sum	3	3	wall_clock	sec	0.001494	0.000498	0.000440	0.000537	0.000000	0.000051	100.0

Visualizing Python™ Perfeto tracing



Kokkos (I)

- The Omnitrace can instrument Kokkos applications
- Edit your omnitrace.cfg file and enable Kokkos:

OMNITRACE_USE_KOKKOSP = true

```
[redacted]$ ls -ltr omnitrace-idefix.inst-output/2022-12-07_16.48
total 29176
-rw-r--r--. [redacted]  182160 Dec  7 16:49 trip_count-0.txt
-rw-r--r--.              797524 Dec  7 16:49 trip_count-0.json
-rw-r--r--.              211968 Dec  7 16:49 sampling_percent-0.txt
-rw-r--r--.              925935 Dec  7 16:49 sampling_percent-0.json
-rw-r--r--.               32111 Dec  7 16:49 roctracer-0.txt
-rw-r--r--.              293068 Dec  7 16:49 roctracer-0.json
-rw-r--r--.            21180508 Dec  7 16:49 perfetto-trace-0.proto
-rw-r--r--.              332328 Dec  7 16:49 wall_clock-0.txt
-rw-r--r--.             1718005 Dec  7 16:49 wall_clock-0.json
-rw-r--r--.              276000 Dec  7 16:49 sampling_wall_clock-0.txt
-rw-r--r--.             1275958 Dec  7 16:49 sampling_wall_clock-0.json
-rw-r--r--.                5825 Dec  7 16:49 sampling_gpu_temperature-0.txt
-rw-r--r--.               42414 Dec  7 16:49 sampling_gpu_temperature-0.json
-rw-r--r--.                5700 Dec  7 16:49 sampling_gpu_power-0.txt
-rw-r--r--.               42899 Dec  7 16:49 sampling_gpu_power-0.json
-rw-r--r--.                6000 Dec  7 16:49 sampling_gpu_memory_usage-0.txt
-rw-r--r--.               45629 Dec  7 16:49 sampling_gpu_memory_usage-0.json
-rw-r--r--.                5775 Dec  7 16:49 sampling_gpu_busy_percent-0.txt
-rw-r--r--.               41991 Dec  7 16:49 sampling_gpu_busy_percent-0.json
-rw-r--r--.              273792 Dec  7 16:49 sampling_cpu_clock-0.txt
-rw-r--r--.             1272968 Dec  7 16:49 sampling_cpu_clock-0.json
-rw-r--r--.              249585 Dec  7 16:49 metadata-0.json
-rw-r--r--.              109785 Dec  7 16:49 kokkos_memory-0.txt
-rw-r--r--.              328960 Dec  7 16:49 kokkos_memory-0.json
-rw-r--r--.              166581 Dec  7 16:49 functions-0.json
```

Kokkos (II)

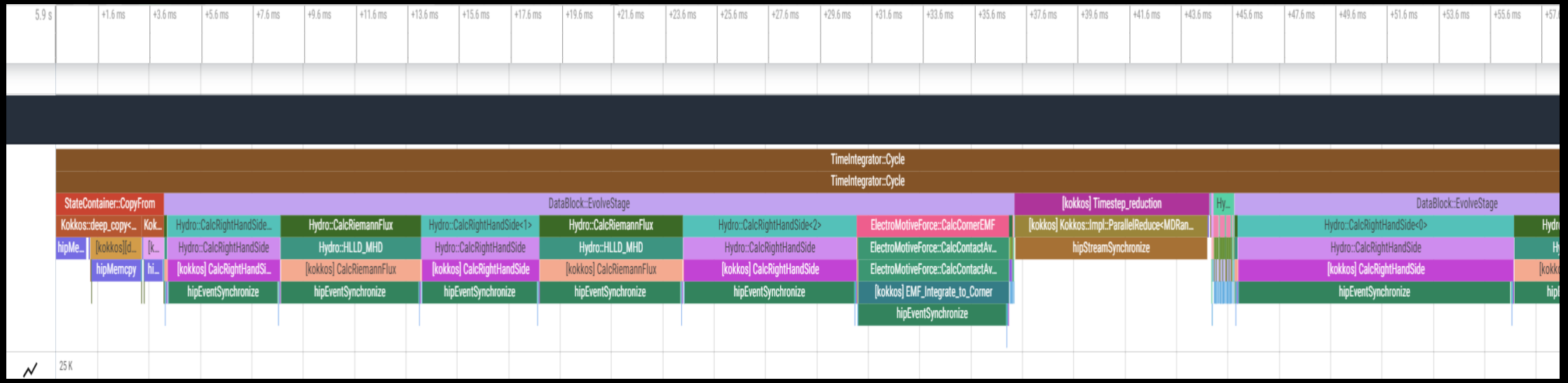
- Check the file kokkos_memory0.txt

KOKKOS MEMORY TRACKER

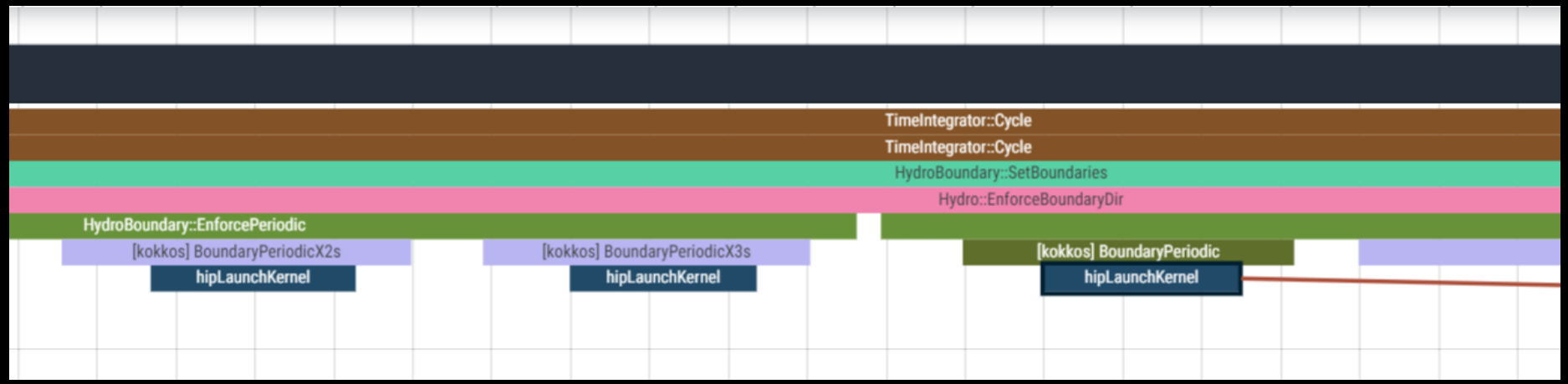
LABEL	COUNT	DEPTH	METRIC	UNITS	SUM	MEAN	% SELF
|0>>> |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, post deep copy fence	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos] Kokkos::deep_copy: copy between contiguous views, post deep copy fence	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos][deep_copy] Host=DataBlock_A2_mirror HIP=DataBlock_A2	1	2	kokkos_memory	MB	142	142	100
|0>>> |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, pre view equality check	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos] Kokkos::deep_copy: copy between contiguous views, pre view equality check	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, post deep copy fence	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos] Kokkos::deep_copy: copy between contiguous views, post deep copy fence	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos][deep_copy] Host=DataBlock_dV_mirror HIP=DataBlock_dV	1	2	kokkos_memory	MB	140	140	100
|0>>> |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, pre view equality check	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos] Kokkos::deep_copy: copy between contiguous views, pre view equality check	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, post deep copy fence	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos] Kokkos::deep_copy: copy between contiguous views, post deep copy fence	1	3	kokkos_memory	MB	0	0	0
|0>>> |_DataBlockHost::SyncToDevice()	1	1	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos][deep_copy] HIP=Hydro_Vc Host=Hydro_Vc_mirror	1	2	kokkos_memory	MB	1124	1124	100
|0>>> |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, pre view equality check	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos] Kokkos::deep_copy: copy between contiguous views, pre view equality check	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, post deep copy fence	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos] Kokkos::deep_copy: copy between contiguous views, post deep copy fence	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos][deep_copy] HIP=Hydro_InvDt Host=Hydro_InvDt_mirror	1	2	kokkos_memory	MB	140	140	100
|0>>> |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, pre view equality check	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos] Kokkos::deep_copy: copy between contiguous views, pre view equality check	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, post deep copy fence	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos] Kokkos::deep_copy: copy between contiguous views, post deep copy fence	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos][deep_copy] HIP=Hydro_Vs Host=Hydro_Vs_mirror	1	2	kokkos_memory	MB	426	426	100
|0>>> |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, pre view equality check	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos] Kokkos::deep_copy: copy between contiguous views, pre view equality check	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, post deep copy fence	1	3	kokkos_memory	MB	0	0	0
|0>>> |_[kokkos] Kokkos::deep_copy: copy between contiguous views, post deep copy fence	1	3	kokkos_memory	MB	0	0	0

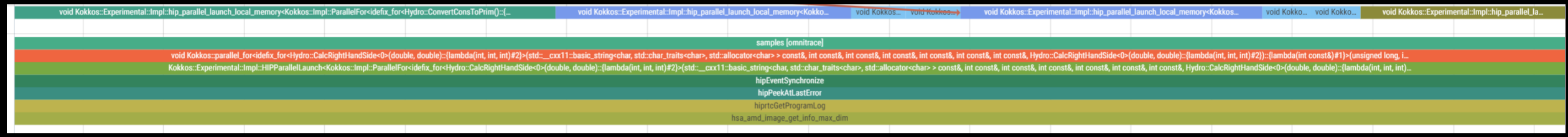
Kokkos – Perfetto I

- Visualize perfetto-trace-0.proto (with sampling enabled)



Kokkos – Perfetto II





Omnitrace-sample

- For easy usage of Omnitrace there is also the omnitrace-sample that does sampling with less overhead.
- It provides less overhead but you need to be sure that you do not miss information
- Not all the declarations of a cfg file apply, for example to use hardware counters, ou need to execute the following command:

```
srun -n 1 omnitrace-sample -TPHD -G
"GPUBusy:device=0,Wavefronts:device=0,VALUBusy:device=0,L2CacheHit:device=0,MemUnitBusy:device=0” -- ./binary
```

See omnitrace-sample -h for more information

Tips & Tricks

- My Perfetto timeline seems weird how can I check the clock skew?
  - OMNITRACE_VERBOSE equal to 1 or higher for verbose mode and it will print the timestamp skew
- Omnitrace takes too long time in the finalization, how to check which part takes a lot of time?
  - Use OMNITRACE_VERBOSE equal to 1 or higher for verbose mode
- It takes too long time to map rocm-smi samples to the kernels
  - Use temporarily OMNITRACE_USE_ROCM_SMI=OFF
- If you are doing binary rewriting and you do not get information about kernels, declare:
  - HSA_TOOLS_LIB=libomnitrace.so in the environment and be sure that OMNITRACE_USE_ROCTRACER=ON in the cfg file
- My HIP application hangs in different points, what to do?
  - Try to set HSA_ENABLE_INTERRUPT=0 in the environment, this handles different how HIP is notified that GPU kernels completed
- It is preferred to use binary rewriting for MPI applications, in order to write one file per MPI process, and not aggregated, use: OMNITRACE_USE_PID=ON
- My Perfetto trace is too big, can I decrease it?
  - Yes, with v1.7.3 and later declare OMNITRACE_PERFETTO_ANNOTATIONS to false.
- Full documentation: https://amdresearch.github.io/omnitrace/

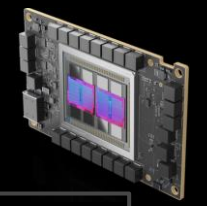
Omniperf

Omniperf

- The Omniperf executes the code as many times required based on the job submission
- Without specific option the application will be executed many times with various hardware counters (more than 100), so this can take long time. It does not mean that all the counters will provide useful data for a specific code.
- There are various options for filtering (kernel, metric) even to execute mainly for roofline analysis, roofline is supported only for MI200 GPU series.
- There are many data per metric/HW and we will show a few, Omniperf provides tables for every metric
- With Omniperf first we profile, then we analyze and then we can import to database or visualize with standalone GUI
- The Omniperf targets MI100 and MI200 and later future generation AMD GPUs

- For problems, create an issue here: https://github.com/AMDResearch/omniperf/issues

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)

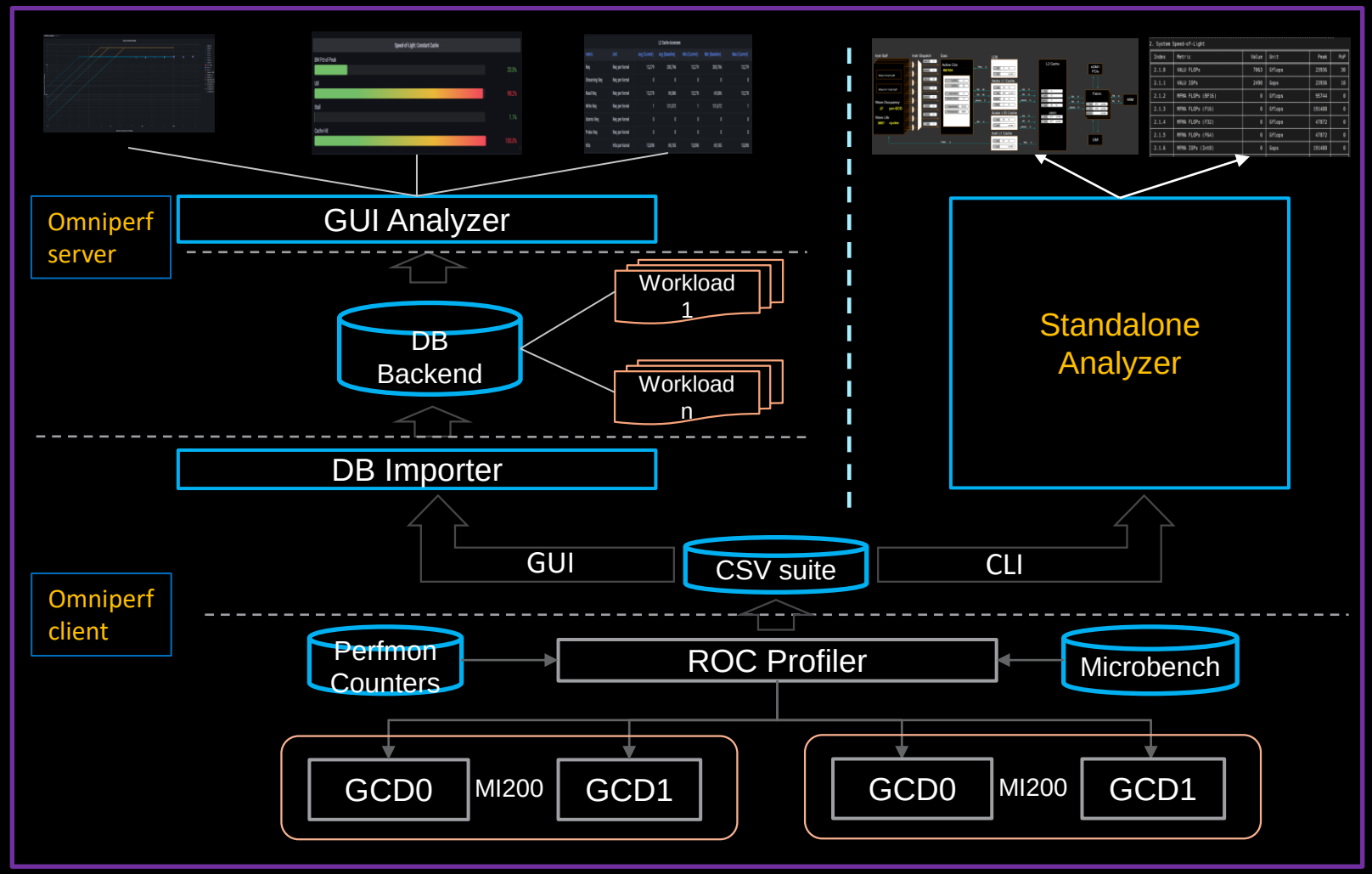
(GCD2)

HBM Memory

HBM Memory

Performance Analysis on MI200 GPUs - Omniperf

- Opensource github repos
  - *https://github.com/AMDResearch/omniperf*
- Built on top of ROC Profiler
- Integrated Performance Analyzer for AMD GPUs
  - Roofline Analyzer
  - Mem Chart Analyzer
  - Speed-of-Light
  - Baseline Comparison
  - Shared Workload Database
  - Flexible Filtering and Normalization
  - Comprehensive Profiling
    - Wavefront Dispatching
    - Shader Compute
    - Local Data Share (LDS) Accesses
    - L1/L2 Cache Accesses
    - HBM Accesses
- User Interfaces
  - Grafana™ Based GUI
  - Standalone GUI



Empirical Hierarchical Roofline on MI200 - 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

Empirical Hierarchical Roofline on MI200 - Arithmetic

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

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

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

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

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

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

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

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

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

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

**** All calculations are subject to change without notice***

Omniperf features

Omniperf Features	
MI200 support	Roofline Analysis Panel (*Supported on MI200 only, SLES 15 SP3 or RHEL8*)
MI100 support	Command Processor (CP) Panel
Standalone GUI Analyzer	Shader Processing Input (SPI) Panel
Grafana/MongoDB GUI Analyzer	Wavefront Launch Panel
Dispatch Filtering	Compute Unit - Instruction Mix Panel
Kernel Filtering	Compute Unit - Pipeline Panel
GPU ID Filtering	Local Data Share (LDS) Panel
Baseline Comparison	Instruction Cache Panel
Multi-Normalizations	Scalar L1D Cache Panel
System Info Panel	Texture Addresser and Data Panel
System Speed-of-Light Panel	Vector L1D Cache Panel
Kernel Statistic Panel	L2 Cache Panel
Memory Chart Analysis Panel	L2 Cache (per-Channel) Panel

Client-side installation (if required)

- Download the latest version from here: https://github.com/AMDResearch/omniperf/releases

```
wget https://github.com/AMDResearch/omniperf/releases/download/v1.0.4/omniperf-
1.0.4.tar.gz

tar zxvf omniperf-1.0.4.tar.gz

cd omniperf-1.0.4/
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}/1.0.4 \
        -DPYTHON_DEPS=${INSTALL_DIR}/python-libs \
        -DMOD_INSTALL_PATH=${INSTALL_DIR}/modulefiles ..
make install
export PATH=$INSTALL_DIR/1.0.4/bin:$PATH
```

Omniperf modes

- Profiling

```
omniperf profile  -n workload_name [profile options] [roofline options] --
<profile_cmd>
```

- Analysis

```
omniperf analyze -p workloads/workload_name/mi200/
```

- GUI import

```
omniperf database --import [CONNECTION OPTIONS]
```

- GUI standalone

```
omniperf analyze -p workloads/workload_name/mi200/ --gui
```

Then follow the instructions to open the web page for the GUI

Omniperf Profiling

- We use the example sample/vcopy.cpp from the Omniperf installation folder (cp omniperf/1.0.4/share/sample/vcopy.cpp .)
- Compile with hipcc, let's call the binary vcopy
- Load Omniperf module
- Profiling with the default set pf data for all kernels, execute:

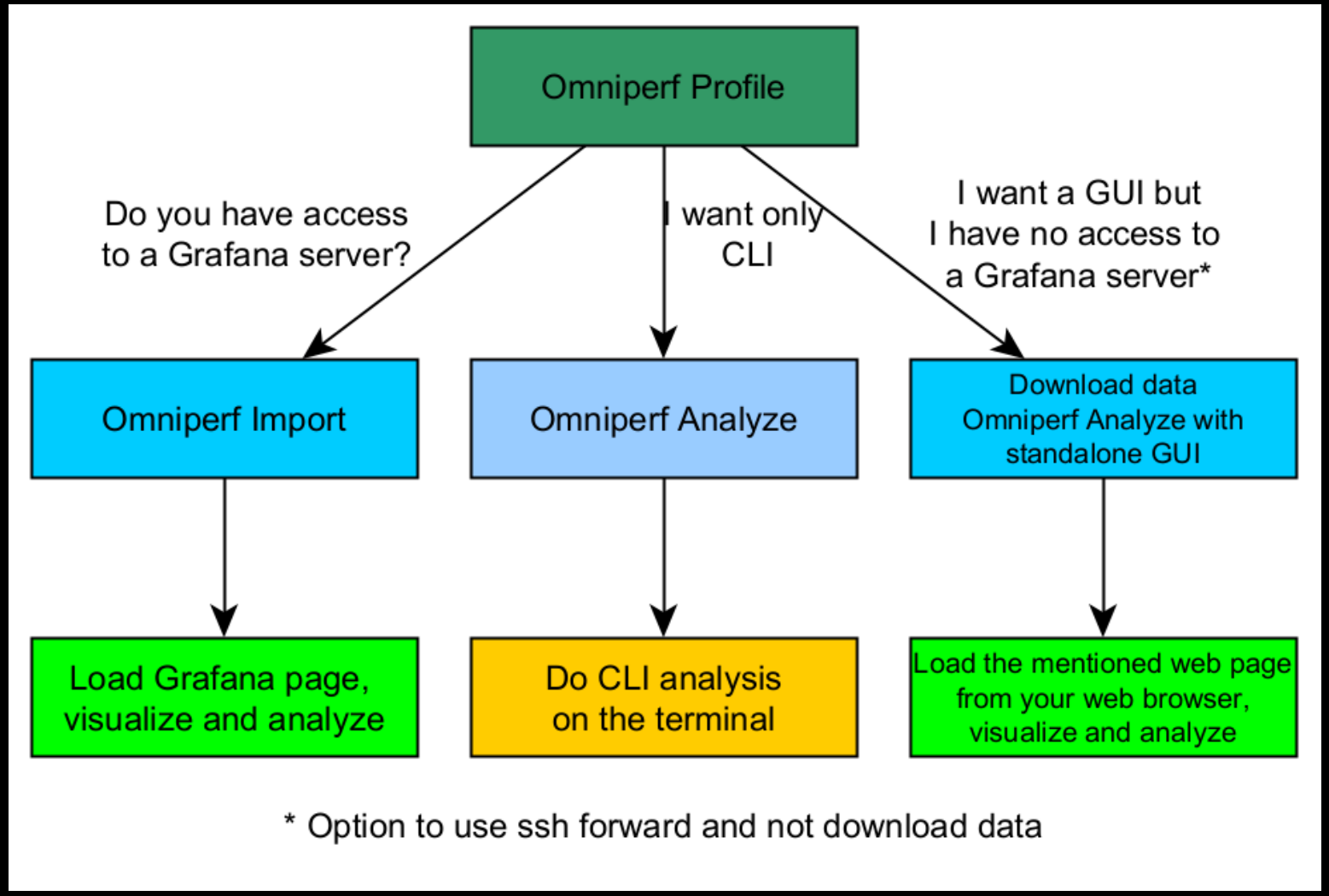
```
srun -n 1 --gpus 1 omniperf profile -n vcopy_all -- ./vcopy 1048576 256
```

```
…
-------------
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
```

In this case we call the workload name "vcopy_all" and after the "--" everything is about the application we execute. In this case, the application will be executed many times for collecting different metrics, if the application takes significant time to run once, then this could b not the optimum approach.

At the end of the execution, we have a folder workloads/vcopy_all/mi200/
You can see all the options with the command `omniperf profile --help`

Omniperf workflows



Omniperf Analyze

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

```
srun -n 1 --gpus 1 omniperf analyze -p workloads/vcopy_all/mi200/ &>
vcopy_analyze.txt
```

0. Top Stat

	KernelName	Count	Sum(ns)	Mean(ns)	Median(ns)	Pct
0	vecCopy(double*, double*, double*, int, int) [clone .kd]	1	341123.00	341123.00	341123.00	100.00

2. System Speed-of-Light

Index	Metric	Value	Unit	Peak	PoP
2.1.0	VALU FLOPs	0.00	Gflop	23936.0	0.0
2.1.1	VALU IOPs	89.14	Giop	23936.0	0.37242200388114116
2.1.2	MFMA FLOPs (BF16)	0.00	Gflop	95744.0	0.0
2.1.3	MFMA FLOPs (F16)	0.00	Gflop	191488.0	0.0
2.1.4	MFMA FLOPs (F32)	0.00	Gflop	47872.0	0.0
2.1.5	MFMA FLOPs (F64)	0.00	Gflop	47872.0	0.0
2.1.6	MFMA IOPs (Int8)	0.00	Giop	191488.0	0.0
2.1.7	Active CUs	58.00	Cus	110	52.72727272727273
2.1.8	SALU Util	3.69	Pct	100	3.6862586934167525
2.1.9	VALU Util	5.90	Pct	100	5.895531580380328
2.1.10	MFMA Util	0.00	Pct	100	0.0
2.1.11	VALU Active Threads/Wave	32.71	Threads	64	51.105263157894[illegible]
2.1.12	IPC - Issue	0.98	Instr/cycle	5	19.576640[illegible]

7.1 Wavefront Launch Stats

Index	Metric	Avg	Min	Max	Unit
7.1.0	Grid Size	1048576.00	1048576.00	1048576.00	Work items
7.1.1	Workgroup Size	256.00	256.00	256.00	Work items
7.1.2	Total Wavefronts	16384.00	16384.00	16384.00	Wavefronts
7.1.3	Saved Wavefronts	0.00	0.00	0.00	Wavefronts
7.1.4	Restored Wavefronts	0.00	0.00	0.00	Wavefronts
7.1.5	VGPRs	44.00	44.00	44.00	Registers
7.1.6	SGPRs	48.00	48.00	48.00	Registers
7.1.7	LDS Allocation	0.00	0.00	0.00	Bytes
7.1.8	Scratch Allocation	16496.00	16496.00	16496.00	Bytes

Omniperf Analyze (II)

- Execute omniperf analyze –h to see various options
- Use specific IP block (-b)
- Top kernel:

```
srun -n 1 --gpus 1 omniperf analyze -p workloads/vcopy_all/mi200/ -b 0
```

- IP Block of wavefronts: `srun -n 1 --gpus 1 omniperf analyze -p workloads/vcopy_all/mi200/ -b 7.1.2`

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

	KernelName	Count	Sum(ns)	Mean(ns)	Median(ns)	Pct
0	vecCopy(double*, double*, double*, int, int) [clone .kd]	1	20960.00	20960.00	20960.00	100.00

```
--------------------------------------------------------------------------------
7. Wavefront
7.1 Wavefront Launch Stats
```

Index	Metric	Avg	Min	Max	Unit
7.1.2	Total Wavefronts	16384.00	16384.00	16384.00	Wavefronts

Omniperf Analyze (III)

omniperf analyze -h

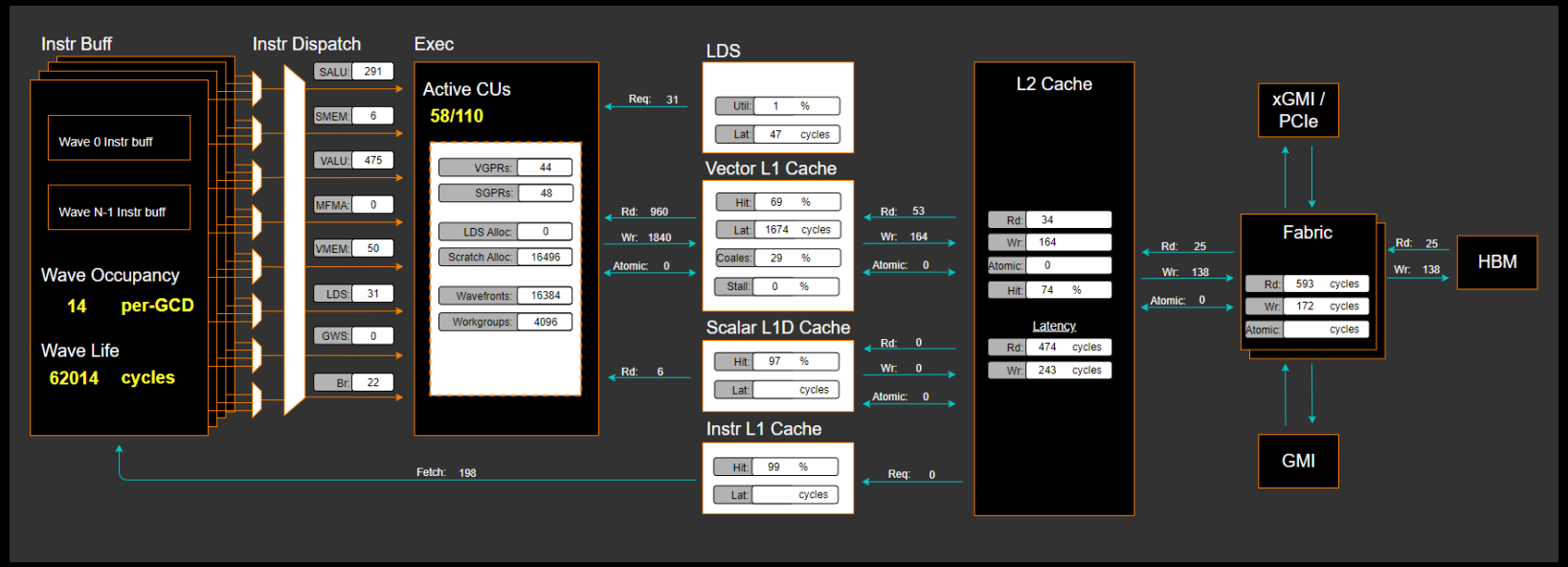
```
Help:
  -h, --help                            show this help message and exit

General Options:
  -v, --version                         show program's version number and exit
  -V, --verbose                         Increase output verbosity

Analyze Options:
  -p  [ ...], --path  [ ...]                        Specify the raw data root dirs or desired results directory.
  -o , --output                                     Specify the output file.
  --list-kernels                                    List kernels.
  --list-metrics                                    List metrics can be customized to analyze on specific arch:
                                                       gfx906
                                                       gfx908
                                                       gfx90a
  -b  [ ...], --filter-metrics  [ ...]              Specify IP block/metric Ids from --list-metrics.
  -k  [ ...], --filter-kernels  [ ...]              Specify kernel id from --list-kernels.
  --filter-dispatch-ids  [ ...]                     Specify dispatch IDs.
  --filter-gpu-ids  [ ...]                          Specify GPU IDs.
  -n , --normal-unit                                Specify the normalization unit: (DEFAULT: per_wave)
                                                       per_wave
                                                       per_cycle
                                                       per_second
  --config-dir                                      Specify the directory of customized configs.
  -t , --time-unit                                  Specify display time unit in kernel top stats: (DEFAULT: ns)
                                                       s
                                                       ms
                                                       us
                                                       ns
  --decimal                                         Specify the decimal to display. (DEFAULT: 2)
  --cols  [ ...]                                    Specify column indices to display.
  -g                                                Debug single metric.
  --dependency                                      List the installation dependency.
  --gui [GUI]                                       Activate a GUI to interate with Omniperf metrics.
                                                    Optionally, specify port to launch application (DEFAULT: 8050)
```

Omniperf Analyze with standalone GUI

- Download the data on your computer (workloads/vcopy_all/), install Omniperf without ROCm, and execute:

```
omniperf analyze -p workloads/vcopy_all/mi200/ --gui
```

Open web page http://IP:8050/



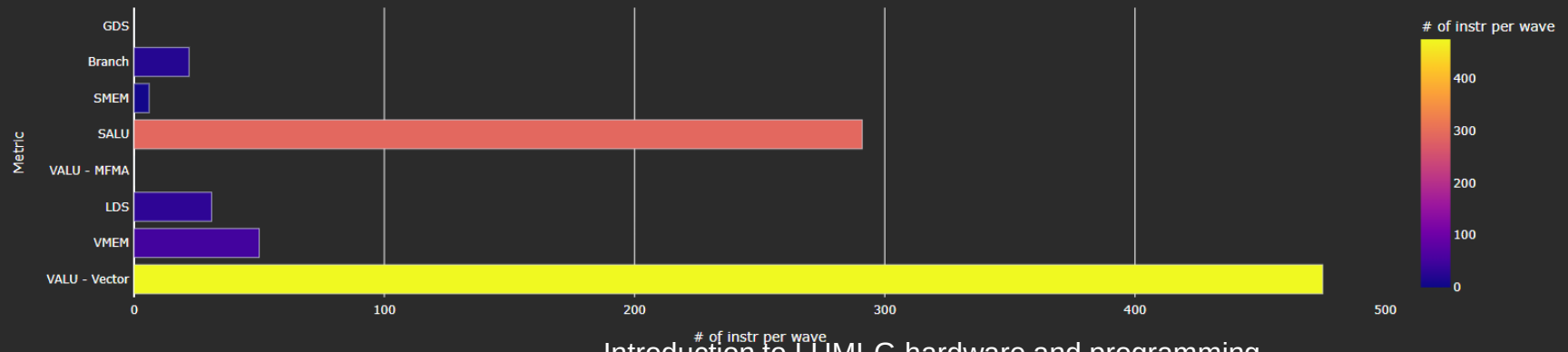
Omniperf Analyze with standalone GUI (II)

2. System Speed-of-Light

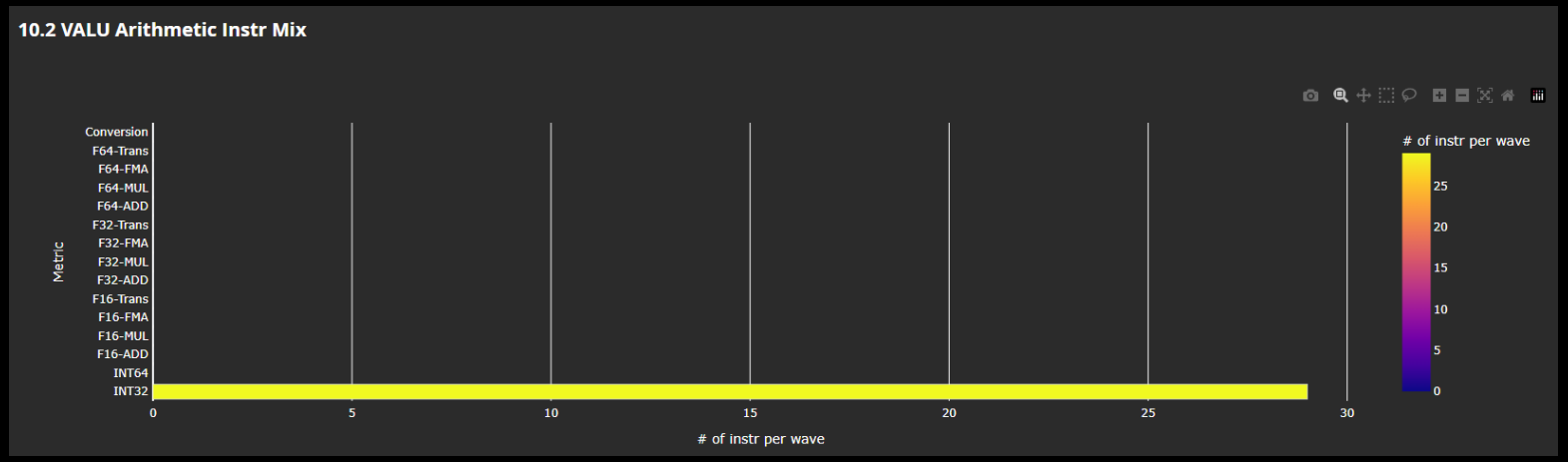
Metric	Value	Unit	Peak	PoP
VALU FLOPs	0.00	Gflop	23936.00	0.00
VALU IOPs	89.14	Giop	23936.00	0.37
MFMA FLOPs (BF16)	0.00	Gflop	95744.00	0.00
MFMA FLOPs (F16)	0.00	Gflop	191488.00	0.00
MFMA FLOPs (F32)	0.00	Gflop	47872.00	0.00
MFMA FLOPs (F64)	0.00	Gflop	47872.00	0.00
MFMA IOPs (Int8)	0.00	Giop	191488.00	0.00
Active CUs	58.00	Cus	110.00	52.73

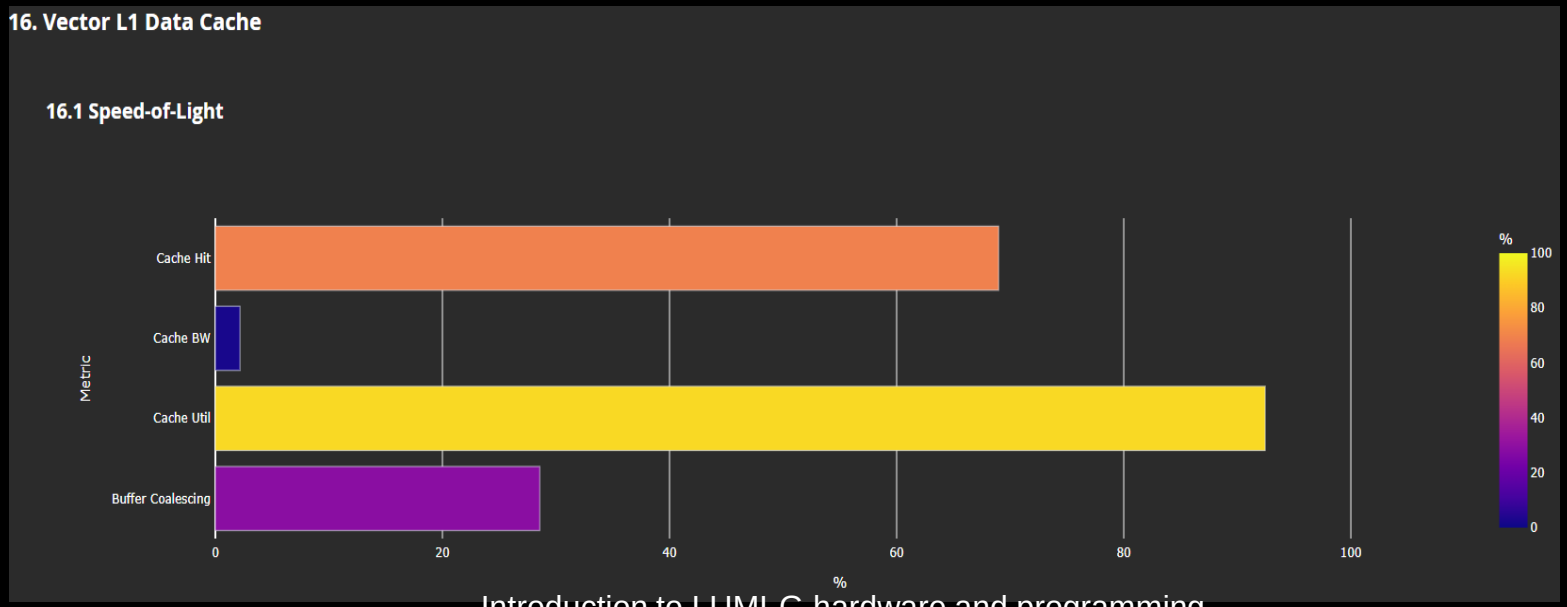
10. Compute Units - Instruction Mix

10.1 Instruction Mix



Omniperf Analyze with standalone GUI (III)



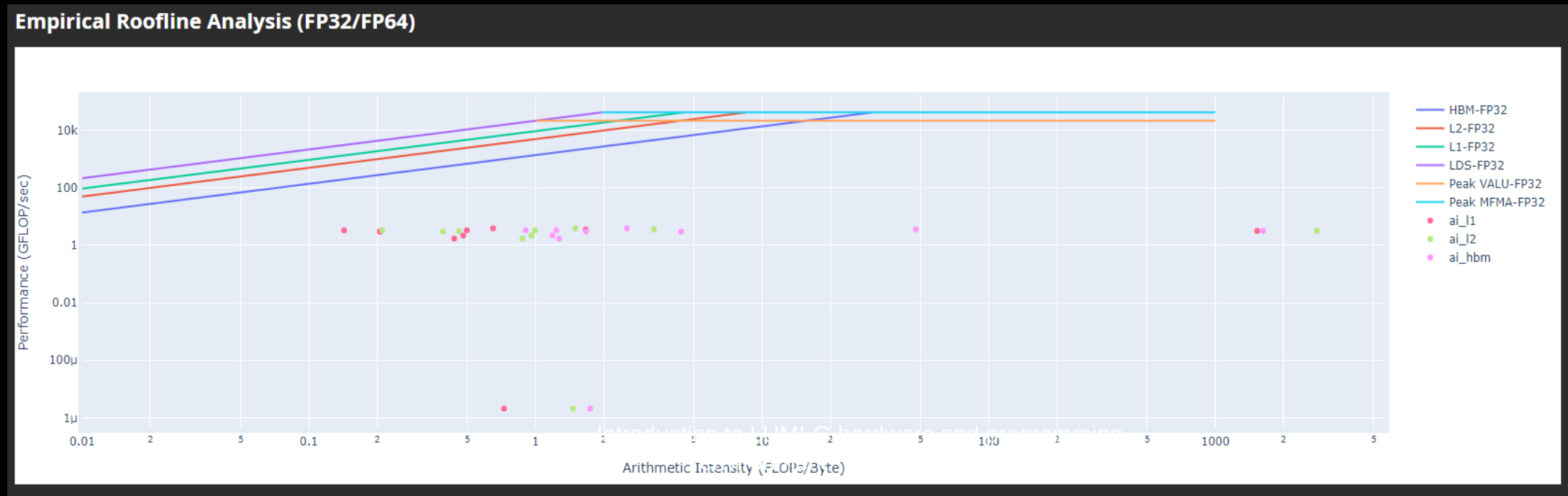


Roofline Analysis

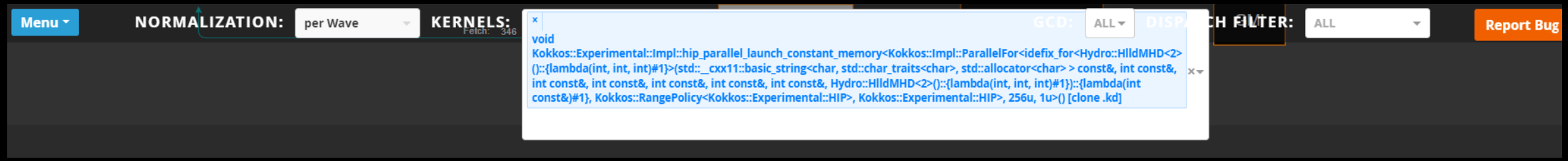
- Profile with roofline:

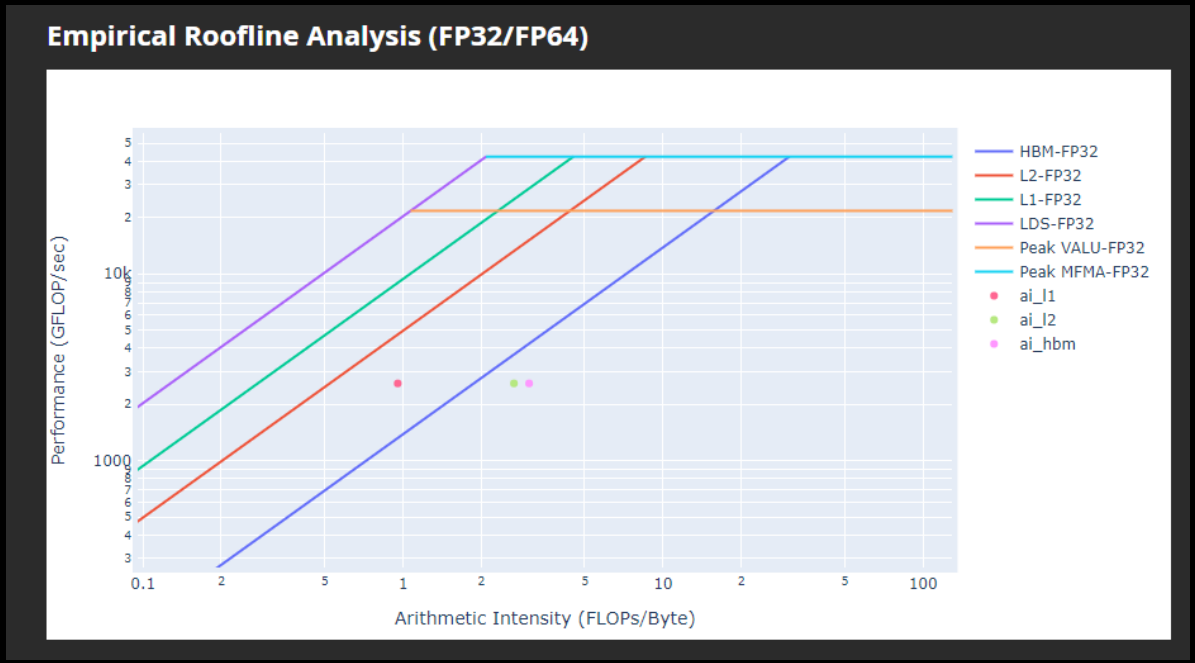
```
srun -n 1 --gpus 1 omniperf profile -n roofline_case_app --roof-only -- ./app
```

- Prepare GUI:
  Copy the workload to your computer
  Execute: omniperf analyze -p workloads/roofline_case_app/mi200/ --gui
  Open the web page http://IP:8050/



Roofline Analysis – Kokkos code





- Roofline: the first-step characterization of workload performance
  - Workload characterization
    - Compute bound
    - Memory bound
    - Performance margin
    - L1/L2 cache accesses
- Thorough SoC perf analysis for each subsystem to identify bottlenecks
  - HBM
  - L1/L2
  - LDS
  - Shader compute
  - Wavefront dispatch
- Omniperf tooling support
  - Roofline plot (float, integer)
  - Baseline roofline comparison
  - Kernel statistics

SPI Resource Allocation

- Dispatch Bound
  - Wavefront dispatching failure due to resources limitation
    - Wavefront slots
    - VGPR
    - SGPR
    - LDS allocation
    - Barriers
    - Etc.
  - Omniperf tooling support
    - Shader Processor Input (SPI) metrics

6.2 SPI Resource Allocation

Metric	Avg	Min	Max	Unit
Wave request Failed (CS)	613303.00	613303.00	613303.00	Cycles
CS Stall	356961.00	356961.00	356961.00	Cycles
CS Stall Rate	62.95	62.95	62.95	Pct
Scratch Stall	0.00	0.00	0.00	Cycles
Insufficient SIMD Waveslots	0.00	0.00	0.00	Simd
Insufficient SIMD VGPRs	16252333.00	16252333.00	16252333.00	Simd
Insufficient SIMD SGPRs	0.00	0.00	0.00	Simd
Insufficient CU LDS	0.00	0.00	0.00	Cu
Insufficient CU Barries	0.00	0.00	0.00	Cu
Insufficient Bulky Resource	0.00	0.00	0.00	Cu
Reach CU Threadgroups Limit	0.00	0.00	0.00	Cycles
Reach CU Wave Limit	0.00	0.00	0.00	Cycles
VGPR Writes	4.00	4.00	4.00	Cycles/wave
SGPR Writes	5.00	5.00	5.00	Cycles/wave

Grafana – System Info

General / Omniperf_v1.0.3_pub

Normalization: "per Wave" | Workload: miperf_aaa_vcopy_mi200 | Dispatch Filter: Enter variable value | GCD: 0 | Kernels: All | Baseline Workload: miperf_asw_vcopy_mi200 | Baseline Dispatch Filter: Enter variable value | Baseline GCD: 0 | Baseline Kernels: All | Comparison Panels: System Info | TopN: 5

System Info

System Info

Metric	Current	Baseline
Date	Tue Jul 5 20:50:45 2022 (UTC)	Tue Jun 21 18:31:40 2022 (CDT)
Host Name	6fb5ce5e50da	node-bp126-014a
Host CPU	AMD Eng Sample: 100-000000248-08_35/21_N	AMD Eng Sample: 100-000000248-08_35/21_N
Host Distro	Ubuntu 20.04.4 LTS	Ubuntu 20.04.4 LTS
Host Kernel	5.9.1-amdsos-build32-1+	5.9.1-amdsos-build32-1+
ROCm Version	5.1.3-66	5.2.0-9768
GFX SoC	mi200	mi200
GFX ID	gfx90a	gfx90a
Total SEs	8	8
Total SQCs	56	56
Total CUs	110	110
SIMDs/CU	4	4
Max Wavefronts Occupancy Per CU	32	32
Max Workgroup Size	1,024	1,024
L1Cache per CU (KB)	16	16
L2Cache (KB)	8,192	8,192
L2Cache Channels	32	32
Sys Clock (Max) - MHz	1,700	1,700
Memory Clock (Max) - MHz	1,600	1,600
Sys Clock (Cur) - MHz	800	800
Memory Clock (Cur) - MHz	1,600	1,600
HBM Bandwidth - GB/s	1,638.4	1,638.4

Grafana – System Speed-of-Light

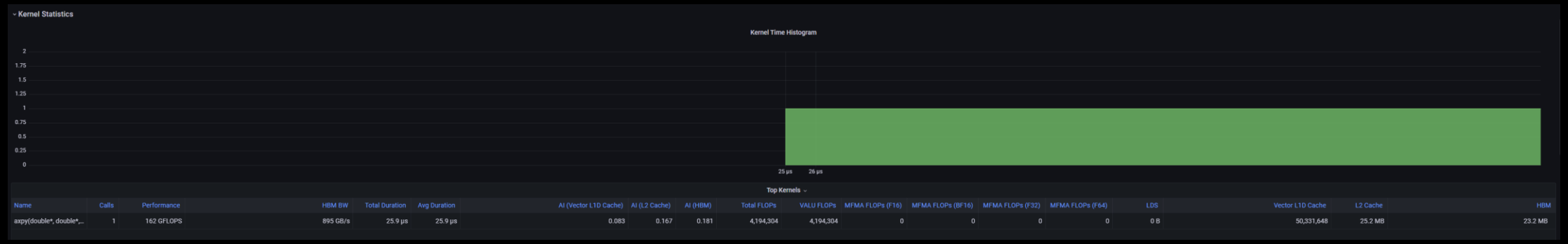
```
$omniperf database --import -H pavii1 -u amd -t asw -w
workloads/vcopy_demo/mi200/
ROC Profiler:  /usr/bin/rocprof

--------
Import Profiling Results
--------

Pulling data from  /root/test/workloads/vcopy_demo/mi200
The directory exists
Found sysinfo file
KernelName shortening enabled
Kernel name verbose level: 2
Password:
Password recieved
-- Conversion & Upload in Progress -
… …
9 collections added.
Workload name uploaded
-- Complete! --
```

System Speed-of-Light

Speed of Light

Metric	Avg	Unit	Theoretical Max	Pct-of-Peak
VALU FLOPs	162	GFLOP	23,936	1%
VALU IOPs	364	GIOP	23,936	2%
MFMA FLOPs (BF16)	0	GFLOP	95,744	0%
MFMA FLOPs (F16)	0	GFLOP	191,488	0%
MFMA FLOPs (F32)	0	GFLOP	47,872	0%
MFMA FLOPs (F64)	0	GFLOP	47,872	0%
MFMA IOPs (Int8)	0	GIOP	191,488	0%
Active CUs	75	CUs	110	68%
SALU Util	4	pct	100	4%
VALU Util	9	pct	100	9%
MFMA Util	0	pct	100	0%
VALU Active Threads/Wave	64	Threads	64	100%
IPC - Issue	1	Instr/cycle	5	18%
LDS BW	0	GB/sec	23,936	0%
LDS Bank Conflict		Conflicts/access	32	
Instr Cache Hit Rate	100	pct	100	100%
Instr Cache BW	243	GB/s	6,093	4%
Scalar L1D Cache Hit Rate	100	pct	100	100%
Scalar L1D Cache BW	162	GB/s	6,093	3%
Vector L1D Cache Hit Rate	50	pct	100	50%
Vector L1D Cache BW	1,942	GB/s	11,968	16%
L2 Cache Hit Rate	30	pct	100	30%
L2-Fabric Read BW	648	GB/s	1,638	40%
L2-Fabric Write BW	247	GB/s	1,638	15%
L2-Fabric Read Latency	402	Cycles		
L2-Fabric Write Latency	432	Cycles		
Wave Occupancy	1,998	Wavefronts	3,520	57%
Instr Fetch BW	0	GB/s	3,046	0%
Instr Fetch Latency	52	Cycles		

Grafana- Kernel Statistics

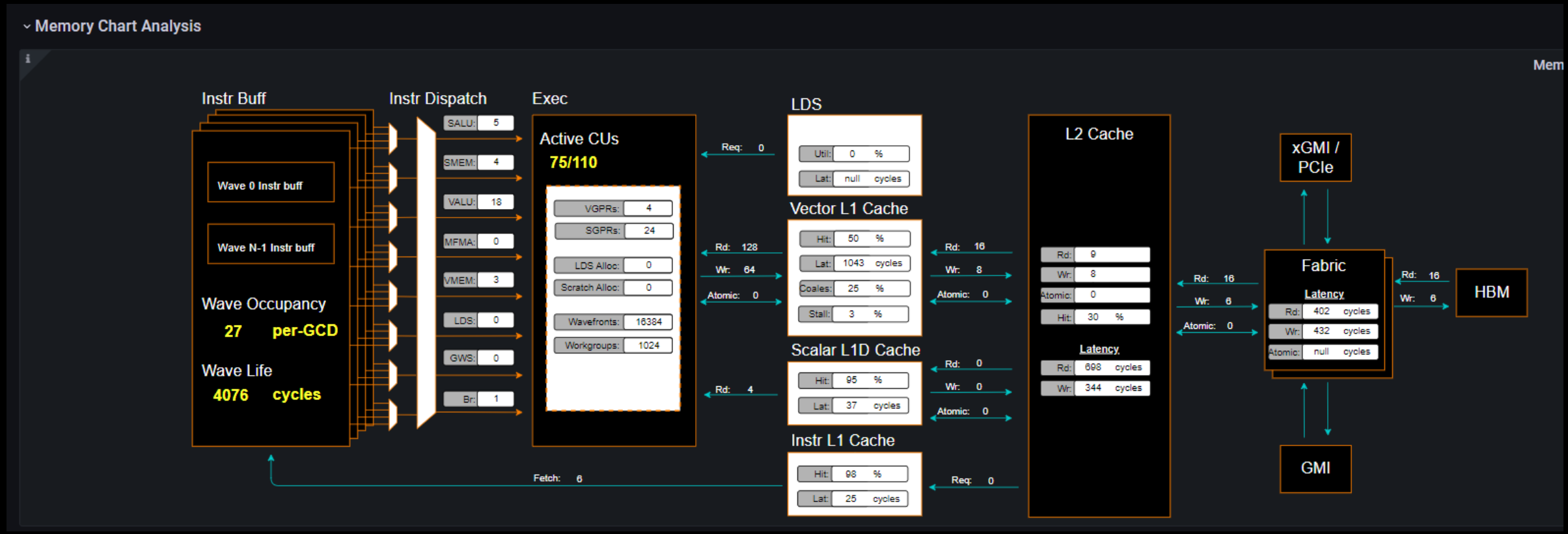


Name	Calls	Performance	HBM BW	Total Duration	Avg Duration	AI (Vector L1D Cache)	AI (L2 Cache)	AI (HBM)	Total FLOPs	VALU FLOPs	MFMA FLOPs (F16)	MFMA FLOPs (BF16)	MFMA FLOPs (F32)	MFMA FLOPs (F64)	LDS	Vector L1D Cache	L2 Cache	HBM
axpy(double*, double*,...	1	162 GFLOPS	895 GB/s	25.9 µs	25.9 µs	0.083	0.167	0.181	4,194,304	4,194,304	0	0	0	0	0 B	50,331,648	25.2 MB	23.2 MB

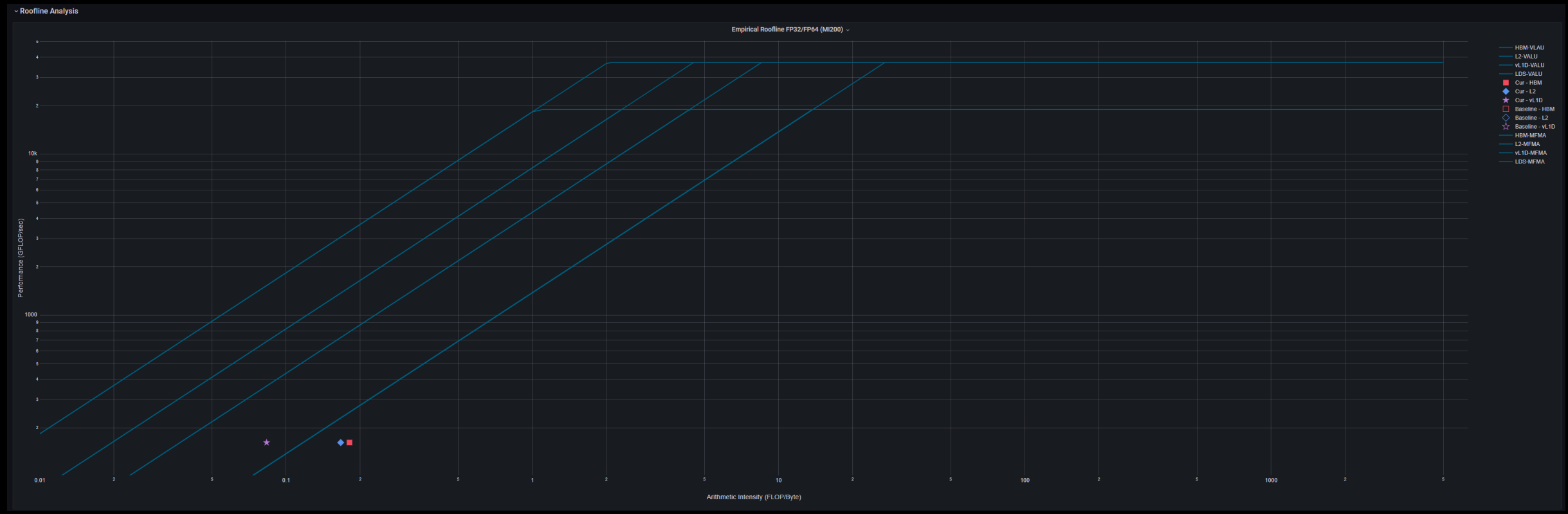
Top Dispatches

Dispatch	Calls	Performance	HBM BW	Total Duration	Avg Duration	AI (Vector L1D Cache)	AI (L2 Cache)	AI (HBM)	Total FLOPs	VALU FLOPs	MFMA FLOPs (F16)	MFMA FLOPs (BF16)	MFMA FLOPs (F32)	MFMA FLOPs (F64)	LDS	Vector L1D Cache	L2 Cache	HBM
0	1	162 GFLOPS	895 GB/s	25.9 µs	25.9 µs	0.083	0.167	0.181	4,194,304	4,194,304	0	0	0	0	0 B	50,331,648	25.2 MB	23.2 MB

Grafana – Memory Chart Analysis



Grafana - Roofline



Grafana – Wavefront & Compute Unit

Wavefront

Wavefront Launch Stats

Metric	Avg	Min	Max	Unit
Grid Size	1,048,576	1,048,576	1,048,576	Work Items
Workgroup Size	1,024	1,024	1,024	Work Items
Total Wavefronts	16,384	16,384	16,384	Wavefronts
Saved Wavefronts	0	0	0	Wavefronts
Restored Wavefronts	0	0	0	Wavefronts
VGPRs	4	4	4	Registers
SGPRs	24	24	24	Registers
LDS Allocation	0	0	0	Bytes
Scratch Allocation	0	0	0	Bytes

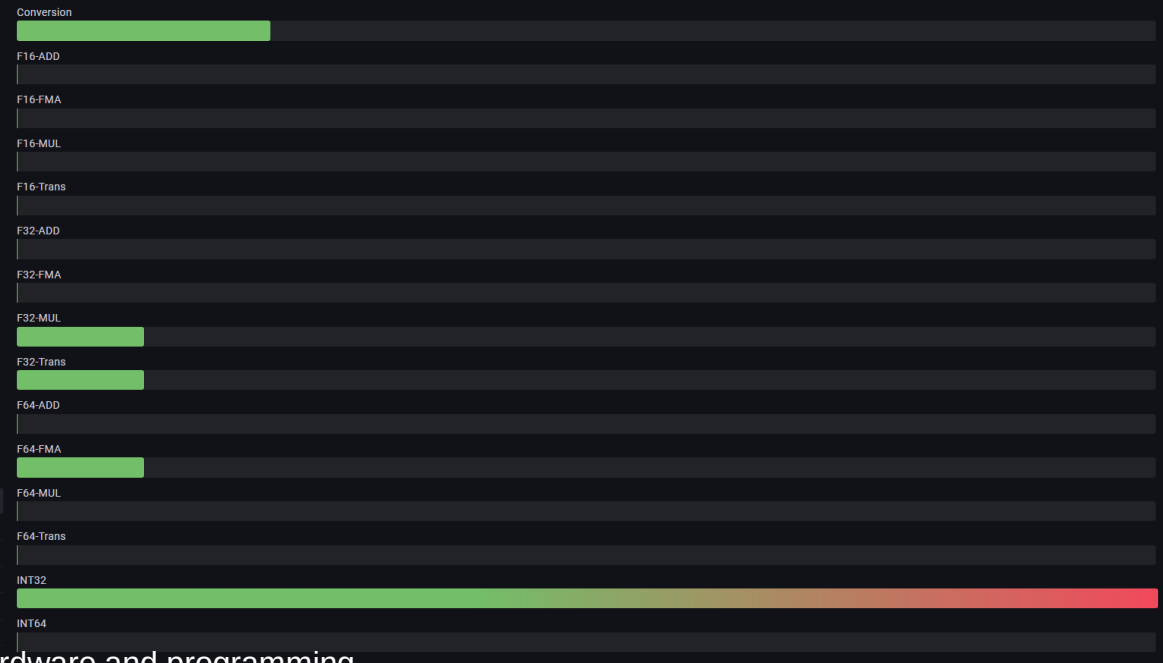
Wavefront Runtime Stats

Metric	Avg	Min	Max	Unit
Kernel Time (Nanosec)	25,920	25,920	25,920	ns
Kernel Time (Cycles)	34,367	34,367	34,367	Cycle
Instr/wavefront	36	36	36	Instr/wavefront
Wave Cycles	4,076	4,076	4,076	Cycles/wave
Dependency Wait Cycles	3,683	3,683	3,683	Cycles/wave
Issue Wait Cycles	780	780	780	Cycles/wave
Active Cycles	140	140	140	Cycles/wave
Wavefront Occupancy	1,998	1,998	1,998	Wavefronts

Compute Unit - Instruction Mix

Instruction Mix

VALU Arithmetic Instr Mix



VMEM Instr Mix

VMEM Instr	Count
Buffer Instr	0
Buffer Read	0
Buffer Write	0
Buffer Atomic	0
Flat Instr	3
Flat Read	
Flat Write	
Flat Atomic	

MFMA Arithmetic Instr Mix

MFMA Instr	Count
MFMA-I8	0

Grafana – Instruction Cache & Scalar L1 Data Cache

Instruction Cache Accesses

Metric	Avg (Current)	Min (Current)	Max (Current)	Unit
Req	6	6	6	Req per Wave
Hits	6	6	6	Hits per Wave
Misses - Non Duplicated	0	0	0	Misses per Wave
Misses - Duplicated	0	0	0	Misses per Wave
Cache Hit	98	98	98	pct

Scalar L1D Cache - L2 Interface

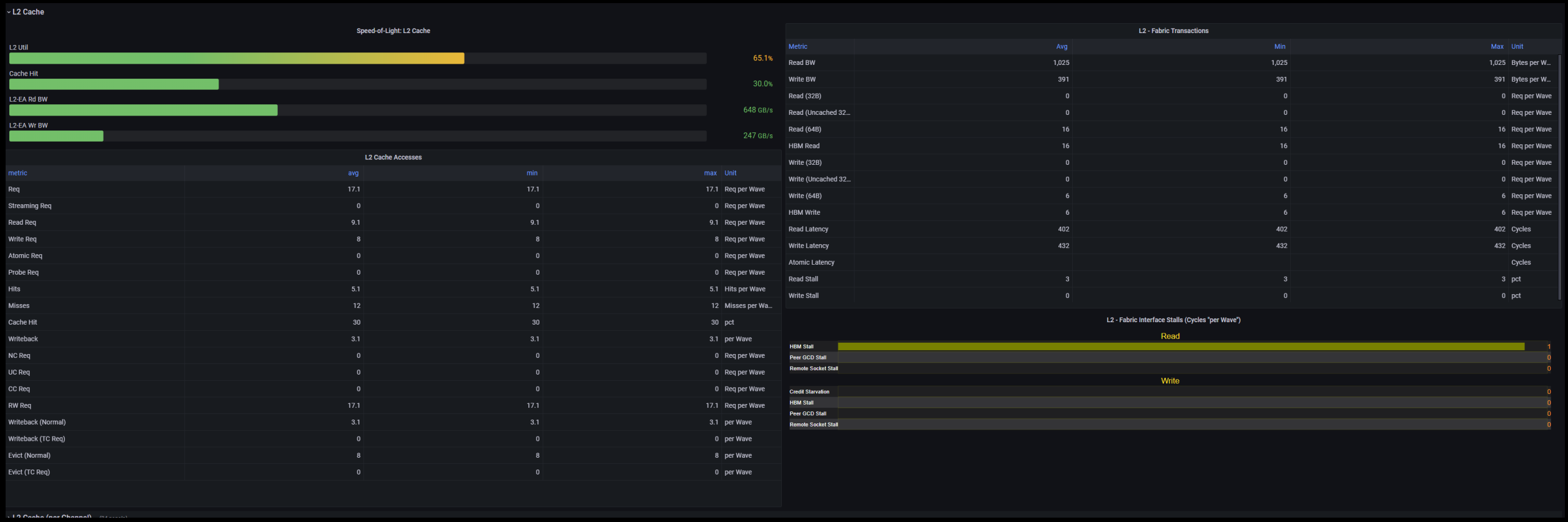
Metric	Mean	Min	Max	Unit
Read Req	0.007	0.007	0.007	Req per Wave
Write Req	0	0	0	Req per Wave
Atomic Req	0	0	0	Req per Wave
Stall	0	0	0	Cycles per Wave

Scalar L1D Cache Accesses

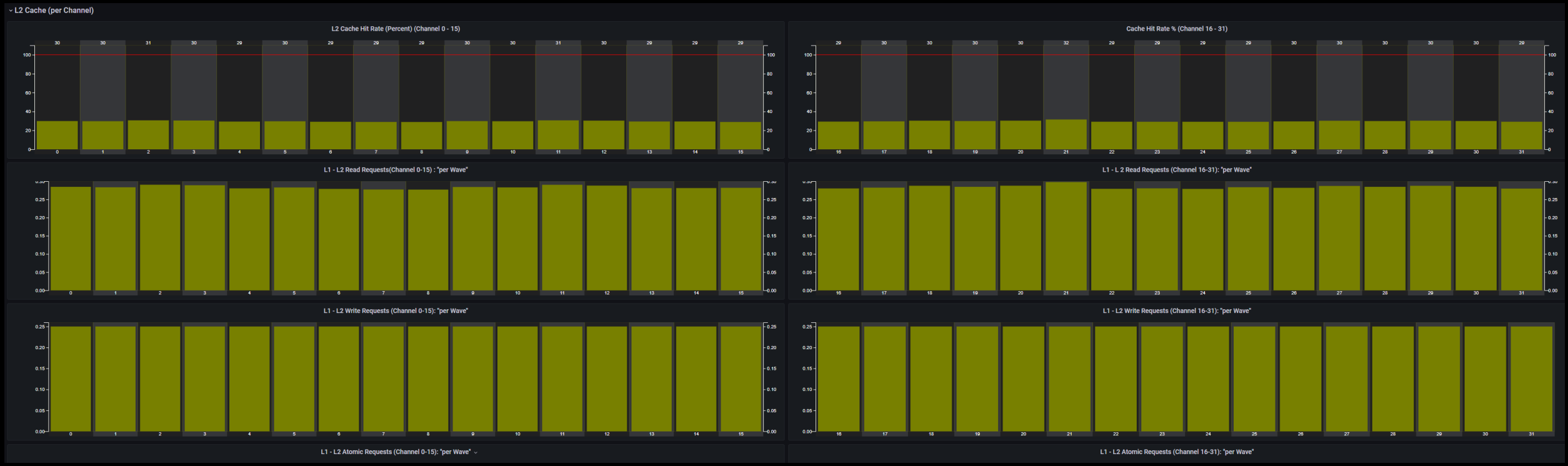
Metric	Avg (Current)	Min (Current)	Max (Current)	Unit
Req	4	4	4	Req per Wave
Hits	4	4	4	Req per Wave
Misses - Non Duplicated	0	0	0	Req per Wave
Misses- Duplicated	0	0	0	Req per Wave
Cache Hit	95	95	95	pct
Read Req (Total)	4	4	4	Req per Wave
Atomic Req	0	0	0	Req per Wave
Read Req (1 DWord)	2	2	2	Req per Wave
Read Req (2 DWord)	1	1	1	Req per Wave
Read Req (4 DWord)	1	1	1	Req per Wave
Read Req (8 DWord)	0	0	0	Req per Wave
Read Req (16 DWord)	0	0	0	Req per Wave

Grafana – Vector L1 Data Cache

Grafana – L2 Cache



Grafana – L2 Cache (per Channel)



ROCgdb

Debugging

Rocgdb

- AMD ROCm source-level debugger for Linux
- based on the GNU Debugger (GDB)
  - tracks upstream GDB master
  - standard GDB commands for both CPU and GPU debugging
- considered a prototype
  - focus on source line debugging
  - no symbolic variable debugging yet

Simple saxpy kernel

```
1   #include <hip/hip_runtime.h>
2
3   __constant__ float a = 1.0f;
4
5   __global__
6   void saxpy(int n, float const* x, int incx, float* y, int incy)
7   {
8       int i = blockDim.x*blockIdx.x + threadIdx.x;
9       if (i < n)
10          y[i] += a*x[i];
11  }
12
13  int main()
14  {
15      int n = 256;
16      std::size_t size = sizeof(float)*n;
17
18      float* d_x;
19      float* d_y;
20      hipMalloc(&d_x, size);
21      hipMalloc(&d_y, size);
22
23      int num_groups = 2;
24      int group_size = 128;
25      saxpy<<<num_groups, group_size>>>(n, d_x, 1, d_y, 1);
26      hipDeviceSynchronize();
27  }
28
```

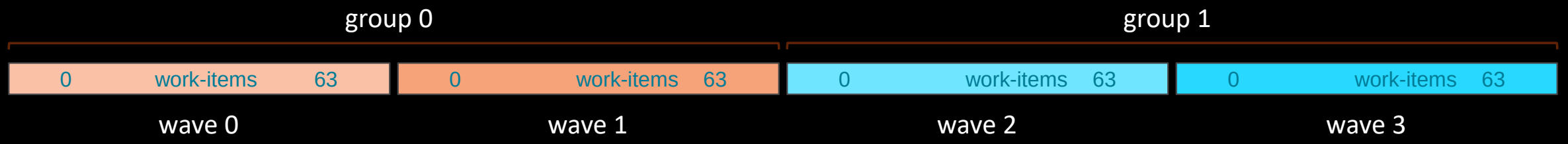
classic saxpy operation

one array index = one work-item

size of arrays = 256

two groups

each 128 work-items



Cause a page fault

```
1   #include <hip/hip_runtime.h>
2
3   __constant__ float a = 1.0f;
4
5   __global__
6   void saxpy(int n, float const* x, int incx, float* y, int incy)
7   {
8       int i = blockDim.x*blockIdx.x + threadIdx.x;
9       if (i < n)
10          y[i] += a*x[i];
11  }
12
13  int main()
14  {
15      int n = 256;
16      std::size_t size = sizeof(float)*n;
17
18      float* d_x;
19      float* d_y;
20      // hipMalloc(&d_x, size);
21      // hipMalloc(&d_y, size);
22
23      int num_groups = 2;
24      int group_size = 128;
25      saxpy<<<num_groups, group_size>>>(n, d_x, 1, d_y, 1);
26      hipDeviceSynchronize();
27  }
28
```

Break it by commenting out the allocations.
(better to initialize the pointers to nullptr)

It’s important to synchronize before exit.
Otherwise, the CPU thread may quit before the GPU gets a chance to report the error.

Compilation with hipcc

```
1   #include <hip/hip_runtime.h>
2
3   __constant__ float a = 1.0f;
4
5   __global__
6   void saxpy(int n, float const* x, int incx, float* y, int incy)
7   {
8       int i = blockDim.x*blockIdx.x + threadIdx.x;
9       if (i < n)
10          y[i] += a*x[i];
11  }
12
13  int main()
14  {
15      int n = 256;
16      std::size_t size = sizeof(float)*n;
17
18      float* d_x;
19      float* d_y;
20      // hipMalloc(&d_x, size);
21      // hipMalloc(&d_y, size);
22
23      int num_groups = 2;
24      int group_size = 128;
25      saxpy<<<num_groups, group_size>>>(n,
26      hipDeviceSynchronize();
27  }
28
```

Need be, set the target

- gfx906 – MI50, MI60, Radeon 7
- gfx908 – MI100
- gfx90a – MI200

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

Execution

```
1   #include <hip/hip_runtime.h>
2
3   __constant__ float a = 1.0f;
4
5   __global__
6   void saxpy(int n, float const* x, int incx, float* y, int incy)
7   {
8       int i = blockDim.x*blockIdx.x + threadIdx.x;
9       if (i < n)
10          y[i] += a*x[i];
11  }
12
13  int main()
14  {
15      int n = 256;
16      std::size_t size = sizeof(float)*n;
17
18      float* d_x;
19      float* d_y;
20      // hipMalloc(&d_x, size);
21      // hipMalloc(&d_y, size);
22
23      int num_groups = 2;
24      int group_size = 128;
25      saxpy<<<num_groups, group_size>>>(n,
26      hipDeviceSynchronize();
27  }
28
```

```
saxpy$ hipcc --offload-arch=gfx90a -o saxpy saxpy.cpp
saxpy$ ./saxpy
```

- In this example we have already allocated a GPU with salloc

Get a page fault

```
1   #include <hip/hip_runtime.h>
2
3   __constant__ float a = 1.0f;
4
5   __global__
6   void saxpy(int n, float const* x, int incx, float* y, int incy)
7   {
8       int i = blockDim.x*blockIdx.x + threadIdx.x;
9       if (i < n)
10          y[i] += a*x[i];
11  }
12
13  int main()
14  {
15      int n = 256;
16      std::size_t size = sizeof(float)*n;
17
18      float* d_x;
19      float* d_y;
20      // hipMalloc(&d_x, size);
21      // hipMalloc(&d_y, size);
22
23      int num_groups = 2;
24      int group_size = 128;
25      saxpy<<<num_groups, group_size>>>(n, d
26      hipDeviceSynchronize();
27  }
28
```

```
saxpy$ hipcc --offload-arch=gfx90a -o saxpy saxpy.cpp
saxpy$ ./saxpy
Memory access fault by GPU node-2 (Agent handle: 0x2284d90) on address (nil). Reason: Unknown.
Aborted (core dumped)
saxpy$
```

Execution with rocgdb

```
1   #include <hip/hip_runtime.h>
2
3   __constant__ float a = 1.0f;
4
5   __global__
6   void saxpy(int n, float const* x, int incx, float* y, int incy)
7   {
8       int i = blockDim.x*blockIdx.x + threadIdx.x;
9       if (i < n)
10          y[i] += a*x[i];
11  }
12
13  int main()
14  {
15      int n = 256;
16      std::size_t size = sizeof(float)*n;
17
18      float* d_x;
19      float* d_y;
20      // hipMalloc(&d_x, size);
21      // hipMalloc(&d_y, size);
22
23      int num_groups = 2;
24      int group_size = 128;
25      saxpy<<<num_groups, group_size>>>(n, d_
26      hipDeviceSynchronize();
27  }
28
```

```
saxpy$ rocgdb saxpy
```

Get more information

```
1   #include <hip/hip_runtime.h>
2
3   __constant__ float a = 1.0f;
4
5   __global__
6   void saxpy(int n, float const* x, int incx, float* y, int incy)
7   {
8       int i = blockDim.x*blockIdx.x + threadIdx.x;
9       if (i < n)
10          y[i] += a*x[i];
11  }
12
13  int main()
14  {
15      int n = 256;
16      std::size_t size = sizeof(float)*n;
17
18      float* d_x;
19      float* d_y;
20      // hipMalloc(&d_x, size);
21      // hipMalloc(&d_y, size);
22
23      int num_groups = 2;
24      int group_size = 128;
25      saxpy<<<num_groups, group_size>>>(n,
26      hipDeviceSynchronize();
27  }
28
```

```
(gdb) run
Starting program: /home/gmarkoma/saxpy
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
[New Thread 0x7fffed428700 (LWP 10456)]
Warning: precise memory violation signal reporting is not enabled, reported
location may not be accurate.  See "show amdgpu precise-memory".

Thread 3 "saxpy" received signal SIGSEGV, Segmentation fault.
[Switching to thread 3, lane 0 (AMDGPU Lane 1:2:1:1/0 (0,0,0)[0,0,0])]
0x00007ffff7ec1094 in saxpy(int, float const*, int, float*, int) () from file:///home/gmarkoma/s
axpy#offset=8192&size=13832
(gdb)
```

Reports segmentation fault in the saxpy kernel.

Compile with -ggdb

```
1   #include <hip/hip_runtime.h>
2
3   __constant__ float a = 1.0f;
4
5   __global__
6   void saxpy(int n, float const* x, int incx, float* y, int incy)
7   {
8       int i = blockDim.x*blockIdx.x + threadIdx.x;
9       if (i < n)
10          y[i] += a*x[i];
11  }
12
13  int main()
14  {
15      int n = 256;
16      std::size_t size = sizeof(float)*n;
17
18      float* d_x;
19      float* d_y;
20      // hipMalloc(&d_x, size);
21      // hipMalloc(&d_y, size);
22
23      int num_groups = 2;
24      int group_size = 128;
25      saxpy<<<num_groups, group_size>>>(n,
26      hipDeviceSynchronize();
27  }
28
```

```
saxpy$ hipcc -ggdb --offload-arch=gfx90a -o saxpy saxpy.cpp
```

Get more details

```
1   #include <hip/hip_runtime.h>
2
3   __constant__ float a = 1.0f;
4
5   __global__
6   void saxpy(int n, float const* x, int incx, float* y, int incy)
7   {
8       int i = blockDim.x*blockIdx.x + threadIdx.x;
9       if (i < n)
10          y[i] += a*x[i];
11  }
12
13  int main()
14  {
15      int n = 256;
16      std::size_t size = sizeof(float)*n;
17
18      float* d_x;
19      float* d_y;
20      // hipMalloc(&d_x, size);
21      // hipMalloc(&d_y, size);
22
23      int num_groups = 2;
24      int group_size = 128;
25      saxpy<<<num_groups, group_size>>>(n, d
26      hipDeviceSynchronize();
27  }
28
```

```
(gdb) run
Starting program: /home/gmarkoma/saxpy
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
[New Thread 0x7fffed428700 (LWP 10637)]
Warning: precise memory violation signal reporting is not enabled, reported
location may not be accurate.  See "show amdgpu precise-memory".

Thread 3 "saxpy" received signal SIGSEGV, Segmentation fault.
[Switching to thread 3, lane 0 (AMDGPU Lane 1:2:1:1/0 (0,0,0)[0,0,0])]
0x00007ffff7ec1094 in saxpy () at saxpy.cpp:10
10          y[i] += a*x[i];
(gdb)
```

more details
- what kernel
- what file:line

But where's my stack trace?

To get exceptions reported precisely: set amdgpu precise-memory on

List threads

```
1   #include <hip/hip_runtime.h>
2
3   __constant__ float a = 1.0f;
4
5   __global__
6   void saxpy(int n, float const* x, int incx, float* y, int incy)
7   {
8       int i = blockDim.x*blockIdx.x + threadIdx.x;
9       if (i < n)
10          y[i] += a*x[i];
11  }
12
13  int main()
14  {
15      int n = 256;
16      std::size_t size = sizeof(float)*n;
17
18      float* d_x;
19      float* d_y;
20      // hipMalloc(&d_x, size);
21      // hipMalloc(&d_y, size);
22
23      int num_groups = 2;
24      int group_size = 128;
25      saxpy<<<num_groups, group_size>>>(n,
26      hipDeviceSynchronize();
27  }
28
```

```
(gdb) i th
  Id   Target Id                                    Frame
  1    Thread 0x7ffff7fe6e80 (LWP 10633) "saxpy" 0x00007fffee0fc499 in rocr::core::InterruptSign
al::WaitRelaxed(hsa_signal_condition_t, long, unsigned long, hsa_wait_state_t) ()
   from /opt/rocm-5.2.0/lib/libhsa-runtime64.so.1
  2    Thread 0x7fffed428700 (LWP 10637) "saxpy" 0x00007ffff5e1972b in ioctl ()
   from /lib64/libc.so.6
* 3    AMDGPU Wave 1:2:1:1 (0,0,0)/0 "saxpy"     0x00007ffff7ec1094 in saxpy () at saxpy.cpp:10
  4    AMDGPU Wave 1:2:1:2 (0,0,0)/1 "saxpy"     0x00007ffff7ec1094 in saxpy () at saxpy.cpp:10
  5    AMDGPU Wave 1:2:1:3 (1,0,0)/0 "saxpy"     0x00007ffff7ec1094 in saxpy () at saxpy.cpp:10
  6    AMDGPU Wave 1:2:1:4 (1,0,0)/1 "saxpy"     0x00007ffff7ec1094 in saxpy () at saxpy.cpp:10
(gdb)
```

What segfaulted is a GPU wave.
It does not have your CPU stack.
List threads to see what's going on.

Switch to the CPU thread

```
1   #include <hip/hip_runtime.h>
2
3   __constant__ float a = 1.0f;
4
5   __global__
6   void saxpy(int n, float const* x, int incx, float* y, int incy)
7   {
8       int i = blockDim.x*blockIdx.x + threadIdx.x;
9       if (i < n)
10          y[i] += a*x[i];
11  }
12
13  int main()
14  {
15      int n = 256;
16      std::size_t size = sizeof(float)*n;
17
18      float* d_x;
19      float* d_y;
20      // hipMalloc(&d_x, size);
21      // hipMalloc(&d_y, size);
22
23      int num_groups = 2;
24      int group_size = 128;
25      saxpy<<<num_groups, group_size>>>(n,
26      hipDeviceSynchronize();
27  }
28
```

t 1
(thread 1)
It's in the HSA runtime.

```
(gdb) t 1
[Switching to thread 1 (Thread 0x7ffff7fe6e80 (LWP 10633))]
#0  0x00007fffee0fc499 in rocr::core::InterruptSignal::WaitRelaxed(hsa_signal_condition_t, long,
 unsigned long, hsa_wait_state_t) () from /opt/rocm-5.2.0/lib/libhsa-runtime64.so.1
(gdb)
```

But how did it get there?

See the stack trace of the CPU thread

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

__constant__ float a = 1.0f;

__global__
void saxpy(int n, float const* x, int incx, float* y, int incy)
{
    int i = blockDim.x*blockIdx.x + threadIdx.x;
    if (i < n)
        y[i] += a*x[i];
}

int main()
{
    int n = 256;
    std::size_t size = sizeof(float)*n;

    float* d_x;
    float* d_y;
    // hipMalloc(&d_x, size);
    // hipMalloc(&d_y, size);

    int num_groups = 2;
    int group_size = 128;
    saxpy<<<num_groups, group_size>>>(n, d_
    hipDeviceSynchronize();
}
```

where

HSA runtime

HIP runtime

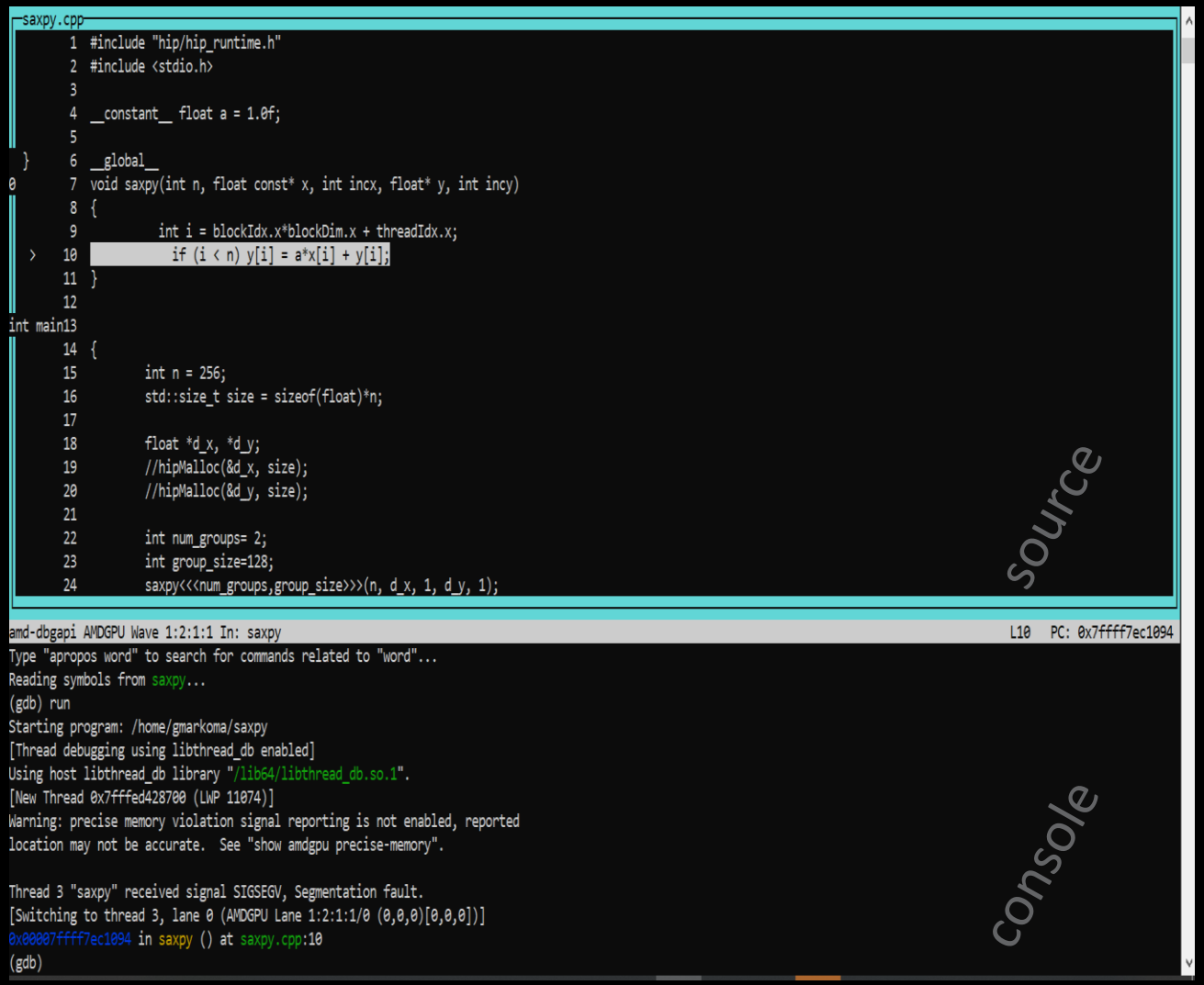
```
(gdb) where
#0  0x00007fffee0fc499 in rocr::core::InterruptSignal::WaitRelaxed(hsa_signal_condition_t, lo
ng, unsigned long, hsa_wait_state_t) () from /opt/rocm-5.2.0/lib/libhsa-runtime64.so.1
#1  0x00007fffee0fc36a in rocr::core::InterruptSignal::WaitAcquire(hsa_signal_condition_t, long,
 unsigned long, hsa_wait_state_t) () from /opt/rocm-5.2.0/lib/libhsa-runtime64.so.1
#2  0x00007fffee0f0869 in rocr::HSA::hsa_signal_wait_scacquire(hsa_signal_s, hsa_signal_conditio
n_t, long, unsigned long, hsa_wait_state_t) () from /opt/rocm-5.2.0/lib/libhsa-runtime64.so.1
#3  0x00007ffff67bdd43 in bool roc::WaitForSignal<false>(hsa_signal_s, bool) ()
   from /opt/rocm-5.2.0/lib/libamdhip64.so.5
#4  0x00007ffff67b5836 in roc::VirtualGPU::HwQueueTracker::CpuWaitForSignal(roc::ProfilingSignal
*) () from /opt/rocm-5.2.0/lib/libamdhip64.so.5
#5  0x00007ffff67b77cf in roc::VirtualGPU::releaseGpuMemoryFence(bool) ()
   from /opt/rocm-5.2.0/lib/libamdhip64.so.5
#6  0x00007ffff67b9523 in roc::VirtualGPU::flush(amd::Command*, bool) ()
   from /opt/rocm-5.2.0/lib/libamdhip64.so.5
#7  0x00007ffff67b9db0 in roc::VirtualGPU::submitMarker(amd::Marker&) ()
   from /opt/rocm-5.2.0/lib/libamdhip64.so.5
#8  0x00007ffff678ec2e in amd::Command::enqueue() () from /opt/rocm-5.2.0/lib/libamdhip64.so.5
#9  0x00007ffff678f1e0 in amd::Event::notifyCmdQueue(bool) ()
   from /opt/rocm-5.2.0/lib/libamdhip64.so.5
#10 0x00007ffff678f28c in amd::Event::awaitCompletion() ()
   from /opt/rocm-5.2.0/lib/libamdhip64.so.5
#11 0x00007ffff6791fdc in amd::HostQueue::finish() () from /opt/rocm-5.2.0/lib/libamdhip64.so.5
#12 0x00007ffff65c25f9 in hipDeviceSynchronize () from /opt/rocm-5.2.0/lib/libamdhip64.so.5
#13 0x0000000000020d615 in main () at saxpy.cpp:25
(gdb)
```

Quick tip

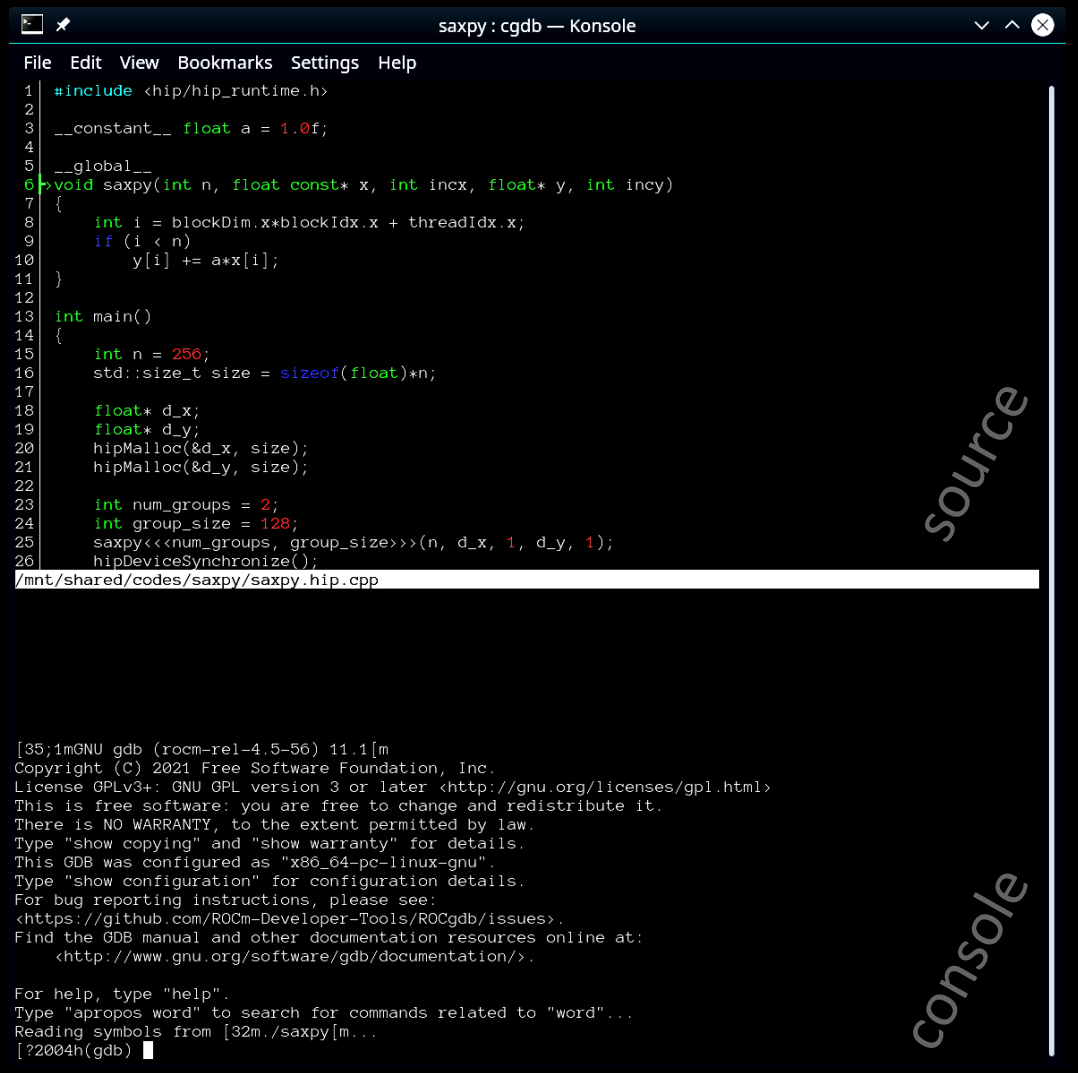
- Frontier and LUMI CPUs have 64 cores / 128 threads.
- If you’re debugging an app with OpenMP threading and OMP_NUM_THREADS is not set you will see 128 CPU threads in rocgdb.
- Set OMP_NUM_THREADS=1 when debugging GPU codes.

”GUIs”

rocgdb -tui saxpy



cgdb -d rocgdb saxpy



Breakpoint

Declare a breakpoint

```
--Type <RET> for more, q to quit, c to continue without paging--For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from saxpy...
(gdb) b saxpy.cpp:22
```

Running with the keystroke ***r*** and stops at the breakpoint

```
--Type <RET> for more, q to quit, c to continue without paging--For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from saxpy...
(gdb) b saxpy.cpp:22
Breakpoint 1 at 0x219dec: file saxpy.cpp, line 22.
(gdb) r
Starting program: /home/gmarkoma/saxpy
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
[New Thread 0x7fffed428700 (LWP 16916)]

Thread 1 "saxpy" hit Breakpoint 1, main () at saxpy.cpp:22
(gdb)
```

Running and architecture

More information about the thread with the command ***i th***

```
(gdb) i th
  Id   Target Id                                  Frame
* 1    Thread 0x7ffff7fe6e80 (LWP 16912) "saxpy" main () at saxpy.cpp:22
  2    Thread 0x7fffed428700 (LWP 16916) "saxpy" 0x00007ffff5e1972b in ioctl () from /lib64/libc.so.6
(gdb)
```

We can see on what device is the thread with the ***show architecture*** command

```
(gdb) show architecture
The target architecture is set to "auto" (currently "i386:x86-64").
(gdb)
```

Breakpoint kernel and architecture

Breakpoint on the kernel called saxpy with the command ***b saxpy***

```
(gdb) b saxpy
Function "saxpy" not defined.
Make breakpoint pending on future shared library load? (y or [n]) yBreakpoint 2 (saxpy) pending.
(gdb)
```

You can continue with he command ***c***

```
(gdb) c
Continuing.
[New Thread 0x7fffdefff700 (LWP 16937)]
[New Thread 0x7fffecaff700 (LWP 16938)]
[Thread 0x7fffdefff700 (LWP 16937) exited]
[Switching to thread 5, lane 0 (AMDGPU Lane 1:2:1:1/0 (0,0,0)[0,0,0])]

Thread 5 "saxpy" hit Breakpoint 2, with lanes [0-63], saxpy (n=256, x=0x7fffec700000, incx=1, y=0x7fffec701000, incy=1) at saxpy.cpp:9
```

We can see on what device is the thread with the command ***show architecture***

```
(gdb) show architecture
The target architecture is set to "auto" (currently "amdgcn:gfx90a").
```

rocgdb + gdbgui

breakpoint in CPU code

Load Binary /mnt/shared/codes/saxpy/saxpy

show filesystem | fetch disassembly | reload file | jump to line | /mnt/shared/codes/saxpy/saxpy.hip.cpp:22 (27 lines total)

```
1  #include <hip/hip_runtime.h>
2
3  __constant__ float a = 1.0f;
4
5  __global__
6  void saxpy(int n, float const* x, int incx, float* y, int incy)
7  {
8      int i = blockDim.x*blockIdx.x + threadIdx.x;
9      if (i < n)
10         y[i] += a*x[i];
11 }
12
13 int main()
14 {
15     int n = 256;
16     std::size_t size = sizeof(float)*n;
17
18     float* d_x;
19     float* d_y;
20     hipMalloc(&d_x, size);
21     hipMalloc(&d_y, size);
22
23     int num_groups = 2;
24     int group_size = 128;
25     saxpy<<<num_groups, group_size>>>(n, d_x, 1, d_y, 1);
26     hipDeviceSynchronize();
27 }
   (end of file)
```

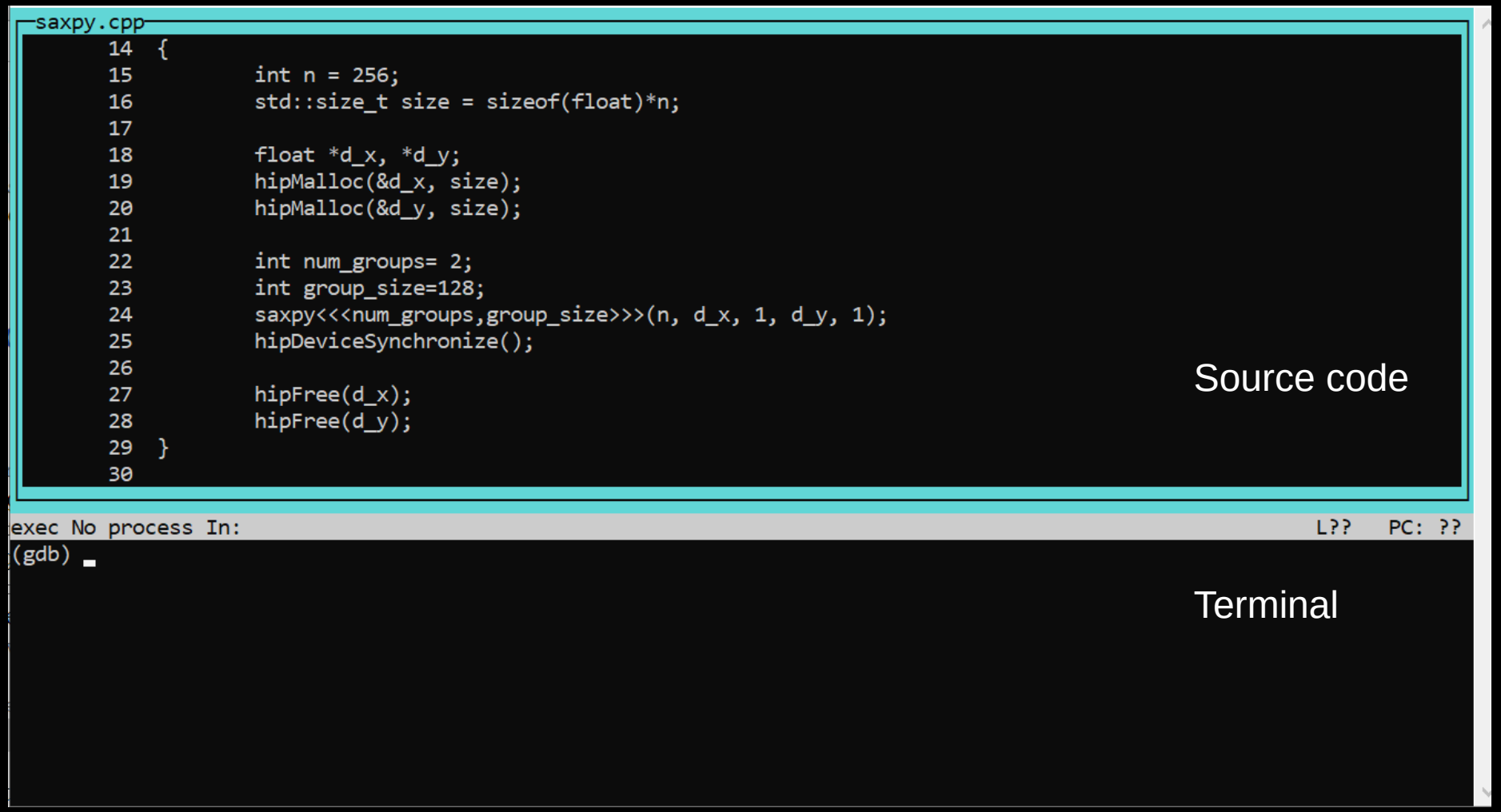
source

```
running command: /opt/rocm/bin/rocgdb

GNU gdb (rocm-rel-4.5-56) 11.1
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://github.com/ROCm-Developer-Tools/ROCgdb/issues>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"
New UI allocated
(gdb)
```

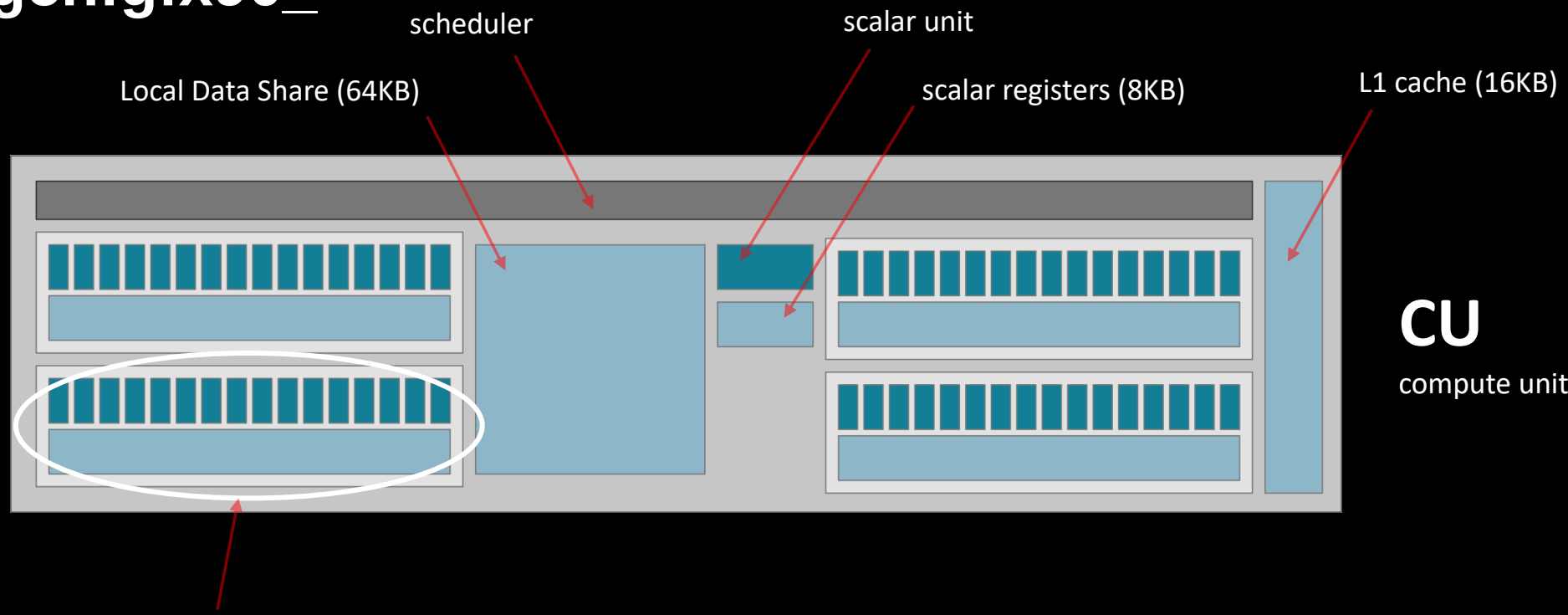
console

Rocgdb with GUI

Execute:
rocgdb –tui saxpy



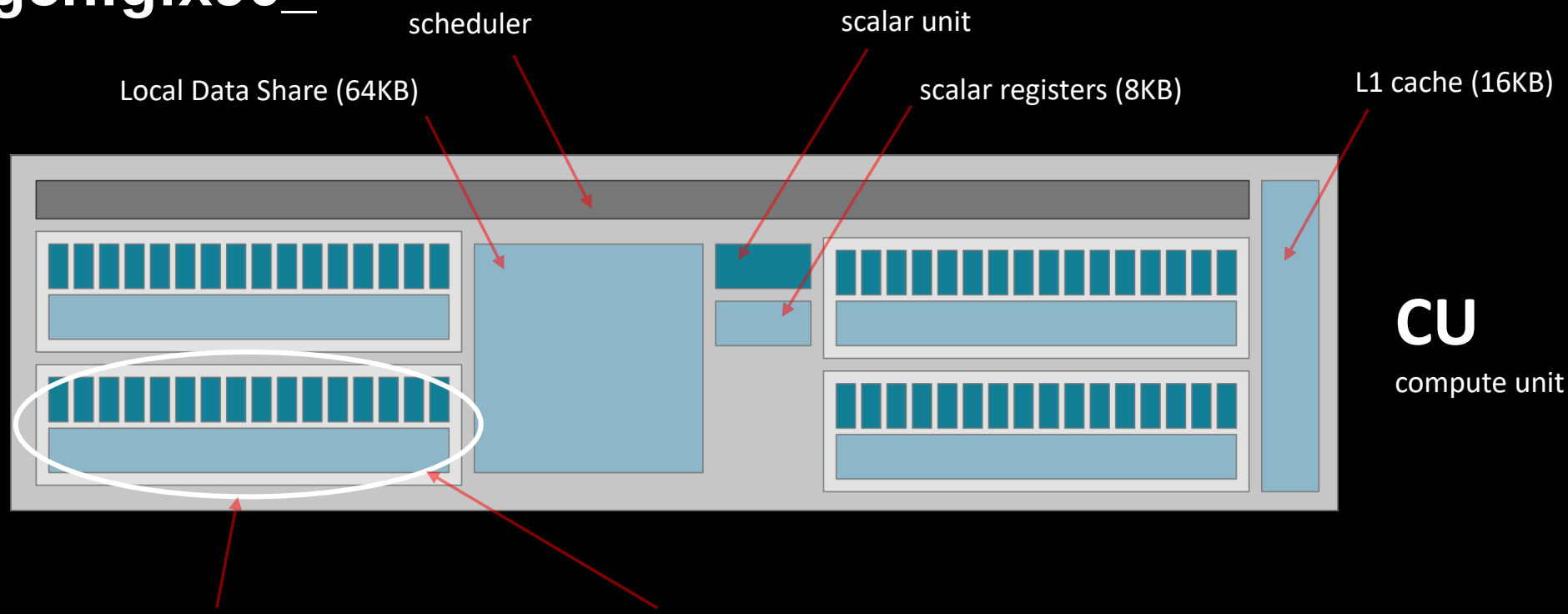
amdgcn:gfx90_



CU
compute unit

typically described as
- a 16-way SIMD unit
- with 64KB of registers

amdgcn:gfx90_



CU
compute unit

typically described as
- a 16-way SIMD unit
- with 64KB of registers

from the standpoint of rocGDB
- a **core**
- executing up to 10 **threads**
- with vector length of 64 **lanes**
- and containing 256 vector **registers**

List threads / waves

i th
(info threads)

some CPU threads

4 GPU “threads” (waves)

```
(gdb) i th
  Id   Target Id                                        Frame
  1    Thread 0x7ffff7fe6e80 (LWP 16912) "saxpy" 0x00007fffee0fc4c0 in rocr::core::InterruptSignal:
t) () from /opt/rocm-5.2.0/lib/libhsa-runtime64.so.1
  2    Thread 0x7fffed428700 (LWP 16916) "saxpy" 0x00007ffff5e1972b in ioctl () from /lib64/libc.so
  4    Thread 0x7fffecaff700 (LWP 16938) "saxpy" 0x00007fffee0fc4af in rocr::core::InterruptSignal:
t) () from /opt/rocm-5.2.0/lib/libhsa-runtime64.so.1
* 5    AMDGPU Wave 1:2:1:1 (0,0,0)/0 "saxpy"     saxpy (n=256, x=0x7fffec700000, incx=1, y=0x7fffec
  6    AMDGPU Wave 1:2:1:2 (0,0,0)/1 "saxpy"     saxpy (n=256, x=0x7fffec700000, incx=1, y=0x7fffec
  7    AMDGPU Wave 1:2:1:3 (1,0,0)/0 "saxpy"     saxpy (n=256, x=0x7fffec700000, incx=1, y=0x7fffec
  8    AMDGPU Wave 1:2:1:4 (1,0,0)/1 "saxpy"     saxpy (n=256, x=0x7fffec700000, incx=1, y=0x7fffec
```

Wave details

`agent-id:queue-id:dispatch-num:wave-id (work-group-x,work-group-y,work-group-z)/work-group-thread-index`

```
(gdb) i th
  Id   Target Id                                        Frame
  1    Thread 0x7ffff7fe6e80 (LWP 16912) "saxpy" 0x00007fffee0fc4c0 in rocr::core::InterruptSignal:
t) () from /opt/rocm-5.2.0/lib/libhsa-runtime64.so.1
  2    Thread 0x7fffed428700 (LWP 16916) "saxpy" 0x00007ffff5e1972b in ioctl () from /lib64/libc.so
  4    Thread 0x7fffecaff700 (LWP 16938) "saxpy" 0x00007fffee0fc4af in rocr::core::InterruptSignal:
t) () from /opt/rocm-5.2.0/lib/libhsa-runtime64.so.1
* 5    AMDGPU Wave 1:2:1:1 (0,0,0)/0 "saxpy"     saxpy (n=256, x=0x7fffec700000, incx=1, y=0x7fffec
  6    AMDGPU Wave 1:2:1:2 (0,0,0)/1 "saxpy"     saxpy (n=256, x=0x7fffec700000, incx=1, y=0x7fffec
  7    AMDGPU Wave 1:2:1:3 (1,0,0)/0 "saxpy"     saxpy (n=256, x=0x7fffec700000, incx=1, y=0x7fffec
  8    AMDGPU Wave 1:2:1:4 (1,0,0)/1 "saxpy"     saxpy (n=256, x=0x7fffec700000, incx=1, y=0x7fffec
```

agent (GPU) ID

(HSA) queue ID

dispatch number

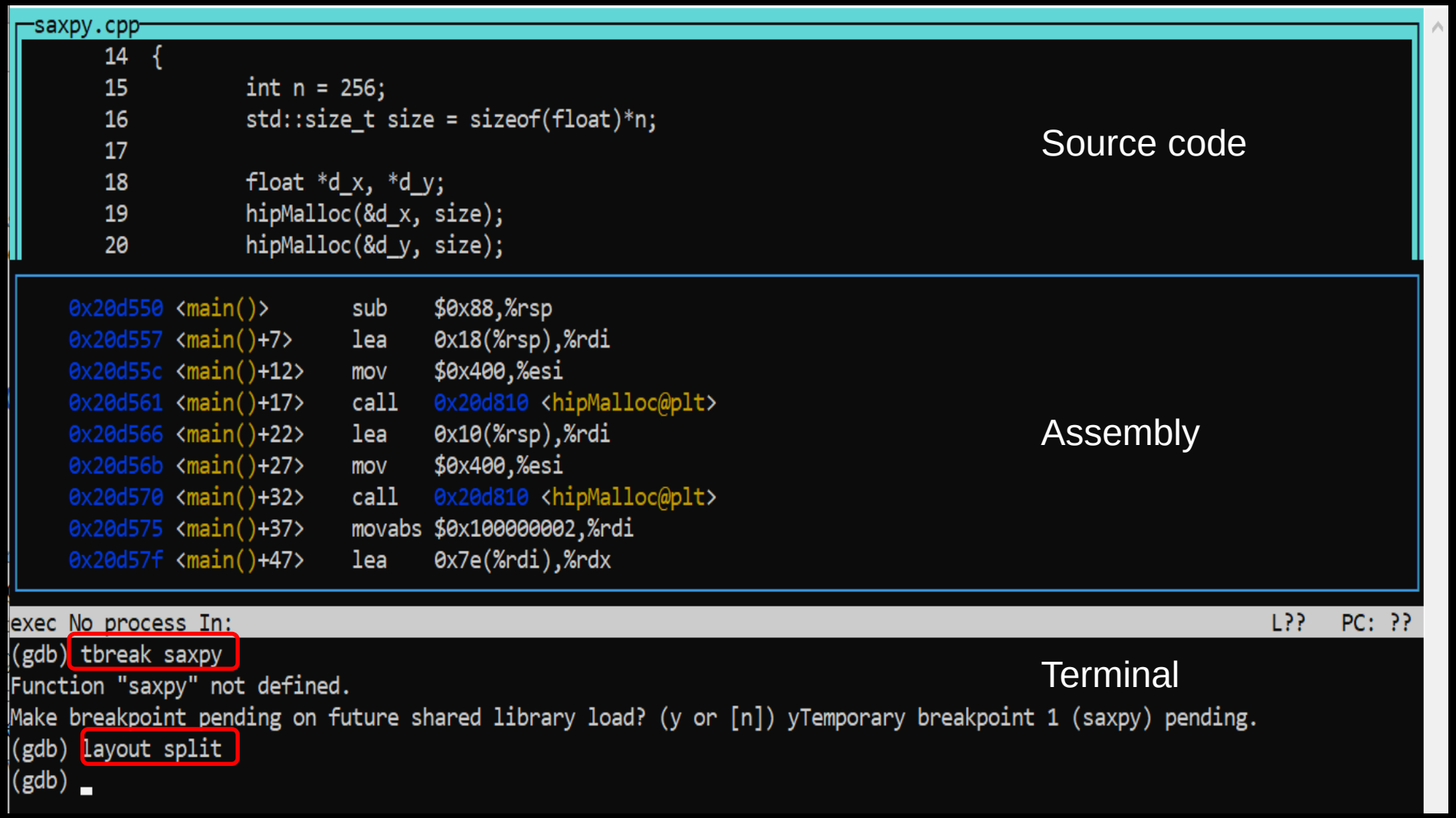
wave ID

workgroup (x, y, z)

wave ID (within group)

Temporary breakpoints and Assembly

- Temporary breakpoint for saxpy kernel:
  tbreak saxpy
- Split to see source code and assembly:
  layout split
- For this example we have compiled with default -O3
- Compiling with -O0 it could give better ISA correlation



List agents

info agents
- shows devices + properties

```
(gdb) info agents
  Id State Target Id                    Architecture Device Name Cores Threads Location
* 1  A     AMDGPU Agent (GPUID 31957) gfx90a       aldebaran   440   3520    29:00.0
```

gfx90a
MI200 series

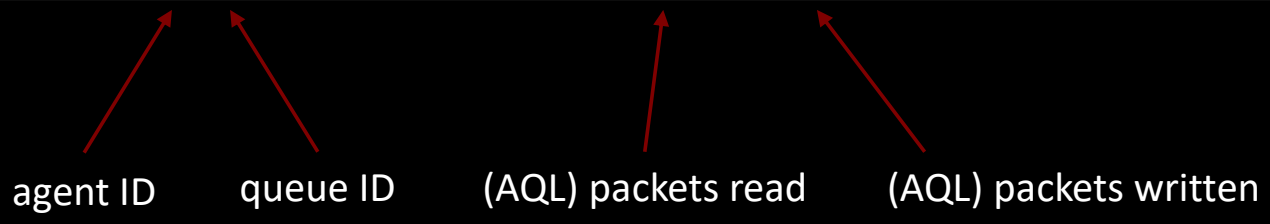
SIMDs
(CUs x 4)

max waves
(SIMDs x 8)

List queues

info queues
- shows HSA queues

```
(gdb) info queues
  Id   Target Id                    Type         Read   Write  Size     Address
  1    AMDGPU Queue 1:1 (QID 0) HSA (Multi)  4      4      4096     0x00007ffff7eb6000
* 2    AMDGPU Queue 1:2 (QID 1) HSA (Multi)  0      2      262144   0x00007ffff7e40000
(gdb)
```



Dispatch details

info dispatches

- shows kernel dispatches

```
(gdb) info dispatches
  Id   Target Id                          Grid      Workgroup Fence   Kernel Function
* 1    AMDGPU Dispatch 1:2:1 (PKID 0) [256,1,1] [128,1,1] B|Aa|Ra saxpy(int, float const*, int, float*, int)
```

agent ID

queue ID

dispatch ID

grid dimensions

group dimensions

kernel

More resources

- /opt/rocm-5.2.0/share/doc/rocgdb/
  - rocannotate.pdf
  - rocgdb.pdf
  - rocrefcard.pdf
  - rocstabs.pdf

- For LUMI: /opt/rocm-5.0.2/share/doc/rocgdb/

AMD_LOG_LEVEL=3

```
:3:devprogram.cpp           :2978: 157529658660 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: _Z5saxpyiPKfiPfi
:3:hip_module.cpp           :365 : 157529658684 us: 224178: [tid:0x7f59c7439e80] ihipModuleLaunchKernel ( 0x0x12e9720, 256, 1, 1, 128, 1, 1, 0, stream:<null>, 0x7fff94e2e07
0, char array:<null>, event:0, event:0, 0, 0 )
:3:rocdevice.cpp            :2686: 157529658695 us: 224178: [tid:0x7f59c7439e80] number of allocated hardware queues with low priority: 0, with normal priority: 0, with hig
h priority: 0, maximum per priority is: 4
:3:rocdevice.cpp            :2757: 157529663975 us: 224178: [tid:0x7f59c7439e80] created hardware queue 0x7f59c72f4000 with size 4096 with priority 1, cooperative: 0
:3:devprogram.cpp           :2675: 157529852150 us: 224178: [tid:0x7f59c7439e80] Using Code Object V4.
:3:devprogram.cpp           :2978: 157529853058 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_fillImage
:3:devprogram.cpp           :2978: 157529853065 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_fillBufferAligned2D
:3:devprogram.cpp           :2978: 157529853070 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_fillBufferAligned
:3:devprogram.cpp           :2978: 157529853076 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_copyImage1DA
:3:devprogram.cpp           :2978: 157529853080 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_copyBufferAligned
:3:devprogram.cpp           :2978: 157529853084 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_streamOpsWait
:3:devprogram.cpp           :2978: 157529853087 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_copyBuffer
:3:devprogram.cpp           :2978: 157529853091 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_streamOpsWrite
:3:devprogram.cpp           :2978: 157529853094 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_copyBufferRectAligned
:3:devprogram.cpp           :2978: 157529853096 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_gwsInit
:3:devprogram.cpp           :2978: 157529853099 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_copyBufferRect
:3:devprogram.cpp           :2978: 157529853101 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_copyImageToBuffer
:3:devprogram.cpp           :2978: 157529853105 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_copyBufferToImage
:3:devprogram.cpp           :2978: 157529853108 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: __amd_rocclr_copyImage
:3:rocvirtual.cpp           :753 : 157529853195 us: 224178: [tid:0x7f59c7439e80] Arg0:   = val:256
:3:rocvirtual.cpp           :679 : 157529853200 us: 224178: [tid:0x7f59c7439e80] Arg1:   = ptr:0x7f59bbb00000 obj:[0x7f59bbb00000-0x7f59bbb00400]
:3:rocvirtual.cpp           :753 : 157529853205 us: 224178: [tid:0x7f59c7439e80] Arg2:   = val:1
:3:rocvirtual.cpp           :679 : 157529853209 us: 224178: [tid:0x7f59c7439e80] Arg3:   = ptr:0x7f59bbb01000 obj:[0x7f59bbb01000-0x7f59bbb01400]
:3:rocvirtual.cpp           :753 : 157529853213 us: 224178: [tid:0x7f59c7439e80] Arg4:   = val:1
:3:rocvirtual.cpp           :2723: 157529853216 us: 224178: [tid:0x7f59c7439e80] ShaderName : _Z5saxpyiPKfiPfi
:3:hip_platform.cpp         :676 : 157529853233 us: 224178: [tid:0x7f59c7439e80] ihipLaunchKernel: Returned hipSuccess :
:3:hip_module.cpp           :509 : 157529853237 us: 224178: [tid:0x7f59c7439e80] hipLaunchKernel: Returned hipSuccess :
:3:hip_device_runtime.cpp   :476 : 157529853243 us: 224178: [tid:0x7f59c7439e80] hipDeviceSynchronize (  )
:3:rocdevice.cpp            :2636: 157529853248 us: 224178: [tid:0x7f59c7439e80] No HW event
:3:rocvirtual.hpp           :62  : 157529853255 us: 224178: [tid:0x7f59c7439e80] Host active wait for Signal = (0x7f59c7442600) for -1 ns
:3:hip_device_runtime.cpp   :488 : 157529853267 us: 224178: [tid:0x7f59c7439e80] hipDeviceSynchronize: Returned hipSuccess :
:3:hip_memory.cpp           :536 : 157529853279 us: 224178: [tid:0x7f59c7439e80] hipFree ( 0x7f59bbb00000 )
:3:rocdevice.cpp            :2093: 157529853291 us: 224178: [tid:0x7f59c7439e80] device=0x12d34f0, freeMem_ = 0xfefffc00
:3:hip_memory.cpp           :538 : 157529853296 us: 224178: [tid:0x7f59c7439e80] hipFree: Returned hipSuccess :
:3:hip_memory.cpp           :536 : 157529853300 us: 224178: [tid:0x7f59c7439e80] hipFree ( 0x7f59bbb01000 )
:3:rocdevice.cpp            :2093: 157529853306 us: 224178: [tid:0x7f59c7439e80] device=0x12d34f0, freeMem_ = 0xff000000
:3:hip_memory.cpp           :538 : 157529853310 us: 224178: [tid:0x7f59c7439e80] hipFree: Returned hipSuccess :
:3:devprogram.cpp           :2978: 157529853333 us: 224178: [tid:0x7f59c7439e80] For Init/Fini: Kernel Name: _Z5saxpyiPKfiPfi
```

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

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.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED “AS IS” WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.

Questions?


AMD