GA Tech
CSE 6230
High Performance Parallel Computing

AMD Accelerators
ROCm
HIP

Jakub Kurzak
Advanced Micro Devices, Inc.

AMD
together we advance_

topics

- porting to AMD
- hardware intro
- ROCm overview
- basic debugging tools
- basic performance tools

porting to AMD

simple SAXPY kernel

```cpp
#include <cuda.h>

__constant__ float a = 2.0f;

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

- vector addition kernel in CUDA
- each thread takes one array index
- and performs one multiply-and-add operation

adding the CPU code

```
1   #include <cuda.h>
2
3   __constant__ float a = 2.0f;
4
5   __global__
6   void saxpy(int n, float const* x, float* y)
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      cudaMalloc(&d_x, size);
21      cudaMalloc(&d_y, size);
22
23      int num_blocks = 2;
24      int num_threads = 128;
25      saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);
26      cudaDeviceSynchronize();
27  }
28
```

allocate arrays in device memory

set up the grid

launch the kernel

adding host↔device copies

```cpp
#include <cuda.h>

__constant__ float a = 2.0f;

__global__
void saxpy(int n, float const* x, float* y)
{
    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* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);

    float* d_x;
    float* d_y;
    cudaMalloc(&d_x, size);
    cudaMalloc(&d_y, size);

    cudaMemcpy(d_x, h_x, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_y, h_y, size, cudaMemcpyHostToDevice);

    int num_blocks = 2;
    int num_threads = 128;
    saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);

    cudaMemcpy(h_y, d_y, size, cudaMemcpyDeviceToHost);
    cudaDeviceSynchronize();
}
```

allocate arrays in host memory

copy content to device memory

copy results back to host memory

adding memory cleanup

```cpp
#include <cuda.h>

__constant__ float a = 2.0f;

__global__
void saxpy(int n, float const* x, float* y)
{
    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* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);

    float* d_x;
    float* d_y;
    cudaMalloc(&d_x, size);
    cudaMalloc(&d_y, size);

    cudaMemcpy(d_x, h_x, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_y, h_y, size, cudaMemcpyHostToDevice);

    int num_blocks = 2;
    int num_threads = 128;
    saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);

    cudaMemcpy(h_y, d_y, size, cudaMemcpyDeviceToHost);
    cudaDeviceSynchronize();

    cudaFree(d_x);
    cudaFree(d_y);

    free(h_x);
    free(h_y);
}
```

free arrays in device memory

free arrays in host memory

adding error checks

```cpp
#include <cuda.h>
#include <cassert>

__constant__ float a = 2.0f;

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

#define CHECK(call) assert(call == cudaSuccess)

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

    float* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);
    assert(h_x != nullptr);
    assert(h_y != nullptr);

    float* d_x;
    float* d_y;
    CHECK(cudaMalloc(&d_x, size));
    CHECK(cudaMalloc(&d_y, size));

    CHECK(cudaMemcpy(d_x, h_x, size, cudaMemcpyHostToDevice));
    CHECK(cudaMemcpy(d_y, h_y, size, cudaMemcpyHostToDevice));

    int num_blocks = 2;
    int num_threads = 128;
    saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);

    CHECK(cudaMemcpy(h_y, d_y, size, cudaMemcpyDeviceToHost));
    CHECK(cudaDeviceSynchronize());

    CHECK(cudaFree(d_x));
    CHECK(cudaFree(d_y));

    free(h_x);
    free(h_y);
}
```

simple error checking macro

simple CUDA code

```cpp
#include <cuda.h>
#include <cassert>

__constant__ float a = 2.0f;

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

#define CHECK(call) assert(call == cudaSuccess)

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

    float* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);
    assert(h_x != nullptr);
    assert(h_y != nullptr);

    float* d_x;
    float* d_y;
    CHECK(cudaMalloc(&d_x, size));
    CHECK(cudaMalloc(&d_y, size));

    CHECK(cudaMemcpy(d_x, h_x, size, cudaMemcpyHostToDevice));
    CHECK(cudaMemcpy(d_y, h_y, size, cudaMemcpyHostToDevice));

    int num_blocks = 2;
    int num_threads = 128;
    saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);

    CHECK(cudaMemcpy(h_y, d_y, size, cudaMemcpyDeviceToHost));
    CHECK(cudaDeviceSynchronize());

    CHECK(cudaFree(d_x));
    CHECK(cudaFree(d_y));

    free(h_x);
    free(h_y);
}
```

simple CUDA code

```cpp
#include <cuda.h>
#include <cassert>

__constant__ float a = 2.0f;

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

#define CHECK(call) assert(call == cudaSuccess)

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

    float* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);
    assert(h_x != nullptr);
    assert(h_y != nullptr);

    float* d_x;
    float* d_y;
    CHECK(cudaMalloc(&d_x, size));
    CHECK(cudaMalloc(&d_y, size));

    CHECK(cudaMemcpy(d_x, h_x, size, cudaMemcpyHostToDevice));
    CHECK(cudaMemcpy(d_y, h_y, size, cudaMemcpyHostToDevice));

    int num_blocks = 2;
    int num_threads = 128;
    saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);

    CHECK(cudaMemcpy(h_y, d_y, size, cudaMemcpyDeviceToHost));
    CHECK(cudaDeviceSynchronize());

    CHECK(cudaFree(d_x));
    CHECK(cudaFree(d_y));

    free(h_x);
    free(h_y);
}
```

same code in HIP

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

__constant__ float a = 2.0f;

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

#define CHECK(call) assert(call == hipSuccess)

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

    float* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);
    assert(h_x != nullptr);
    assert(h_y != nullptr);

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

    CHECK(hipMemcpy(d_x, h_x, size, hipMemcpyHostToDevice));
    CHECK(hipMemcpy(d_y, h_y, size, hipMemcpyHostToDevice));

    int num_blocks = 2;
    int num_threads = 128;
    saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);

    CHECK(hipMemcpy(h_y, d_y, size, hipMemcpyDeviceToHost));
    CHECK(hipDeviceSynchronize());

    CHECK(hipFree(d_x));
    CHECK(hipFree(d_y));

    free(h_x);
    free(h_y);
}
```

spot the differences

simple CUDA code

```cpp
#include <cuda.h>
#include <cassert>

__constant__ float a = 2.0f;

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

#define CHECK(call) assert(call == cudaSuccess)

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

    float* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);
    assert(h_x != nullptr);
    assert(h_y != nullptr);

    float* d_x;
    float* d_y;
    CHECK(cudaMalloc(&d_x, size));
    CHECK(cudaMalloc(&d_y, size));

    CHECK(cudaMemcpy(d_x, h_x, size, cudaMemcpyHostToDevice));
    CHECK(cudaMemcpy(d_y, h_y, size, cudaMemcpyHostToDevice));

    int num_blocks = 2;
    int num_threads = 128;
    saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);

    CHECK(cudaMemcpy(h_y, d_y, size, cudaMemcpyDeviceToHost));
    CHECK(cudaDeviceSynchronize());

    CHECK(cudaFree(d_x));
    CHECK(cudaFree(d_y));

    free(h_x);
    free(h_y);
}
```

same code in HIP

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

__constant__ float a = 2.0f;

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

#define CHECK(call) assert(call == hipSuccess)

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

    float* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);
    assert(h_x != nullptr);
    assert(h_y != nullptr);

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

    CHECK(hipMemcpy(d_x, h_x, size, hipMemcpyHostToDevice));
    CHECK(hipMemcpy(d_y, h_y, size, hipMemcpyHostToDevice));

    int num_blocks = 2;
    int num_threads = 128;
    saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);

    CHECK(hipMemcpy(h_y, d_y, size, hipMemcpyDeviceToHost));
    CHECK(hipDeviceSynchronize());

    CHECK(hipFree(d_x));
    CHECK(hipFree(d_y));

    free(h_x);
    free(h_y);
}
```

HIP kernel language

```cpp
#include <cuda.h>

__constant__ float a = 2.0f;

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

is basically identical to CUDA

- `blockDim.[xyz]`
- `blockIdx.[xyz]`
- `threadIdx.[xyz]`
- `__global__`
- `__device__`
- `__shared__`
- etc.

HIP runtime API

device management
- `hipSetDevice(), hipGetDevice(), hipGetDeviceProperties()`

memory management
- `hipMalloc(), hipFree(), hipMemcpy()`

stream management
- `hipStreamCreate(), hipStreamFree(), hipStreamSynchronize()`

events
- hipEventCreate(), hipEventDestroy(), hipEventRecord()

error handling
- `hipGetLastError(), hipGetErrorString()`

etc.

AMD lingo

CUDA lingo		AMD lingo
block	→	work group
thread	→	work item / SIMD lane
warp	→	wave / wavefront

hipify tools

hipify-clang

https://github.com/ROCm-Developer-Tools/HIPIFY

- compiler (clang) based translator
- handles very complex constructs
- prints an error if not able to translate
- supports clang options
- requires CUDA

hipify-perl

- Perl® script
- relies on regular expressions
- may struggle with complex constructs
- does not require CUDA

```cpp
#include <cuda.h>
#include <cassert>

__constant__ float a = 2.0f;

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

#define CHECK(call) assert(call == cudaSuccess)

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

    float* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);
    assert(h_x != nullptr);
    assert(h_y != nullptr);

    float* d_x;
    float* d_y;
    CHECK(cudaMalloc(&d_x, size));
    CHECK(cudaMalloc(&d_y, size));

    CHECK(cudaMemcpy(d_x, h_x, size, cudaMemcpyHostToDevice));
    CHECK(cudaMemcpy(d_y, h_y, size, cudaMemcpyHostToDevice));

    int num_blocks = 2;
    int num_threads = 128;
    saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);

    CHECK(cudaMemcpy(h_y, d_y, size, cudaMemcpyDeviceToHost));
    CHECK(cudaDeviceSynchronize());

    CHECK(cudaFree(d_x));
    CHECK(cudaFree(d_y));

    free(h_x);
    free(h_y);
}
```

```
saxpy$ perl /opt/rocm/bin/hipify-perl -examin saxpy.cu

[HIPIFY] info: file 'saxpy.cu' statisitics:
  CONVERTED refs count: 13
  TOTAL lines of code: 46
  WARNINGS: 0
[HIPIFY] info: CONVERTED refs by names:
  cuda.h => hip/hip_runtime.h: 1
  cudaDeviceSynchronize => hipDeviceSynchronize: 1
  cudaFree => hipFree: 2
  cudaMalloc => hipMalloc: 2
  cudaMemcpy => hipMemcpy: 3
  cudaMemcpyDeviceToHost => hipMemcpyDeviceToHost: 1
  cudaMemcpyHostToDevice => hipMemcpyHostToDevice: 2
  cudaSuccess => hipSuccess: 1
saxpy$
```

hipify-perl

hipify-perl -examin

- for initial assessment
- no replacements done
- prints basic statistics and the number of replacements

```cuda
#include <cuda.h>
#include <cassert>

__constant__ float a = 2.0f;

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

#define CHECK(call) assert(call == cudaSuccess)

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

    float* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);
    assert(h_x != nullptr);
    assert(h_y != nullptr);

    float* d_x;
    float* d_y;
    CHECK(cudaMalloc(&d_x, size));
    CHECK(cudaMalloc(&d_y, size));

    CHECK(cudaMemcpy(d_x, h_x, size, cudaMemcpyHostToDevice));
    CHECK(cudaMemcpy(d_y, h_y, size, cudaMemcpyHostToDevice));

    int num_blocks = 2;
    int num_threads = 128;
    saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);

    CHECK(cudaMemcpy(h_y, d_y, size, cudaMemcpyDeviceToHost));
    CHECK(cudaDeviceSynchronize());

    CHECK(cudaFree(d_x));
    CHECK(cudaFree(d_y));

    free(h_x);
    free(h_y);
}
```

```
saxpy$ perl /opt/rocm/bin/hipify-perl saxpy.cu
#include "hip/hip_runtime.h"
#include <hip/hip_runtime.h>
#include <cassert>

__constant__ float a = 2.0f;

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

#define CHECK(call) assert(call == hipSuccess)

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

    float* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);
    assert(h_x != nullptr);
    assert(h_y != nullptr);

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

    CHECK(hipMemcpy(d_x, h_x, size, hipMemcpyHostToDevice));
    CHECK(hipMemcpy(d_y, h_y, size, hipMemcpyHostToDevice));

    int num_blocks = 2;
    int num_threads = 128;
    saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);

    CHECK(hipMemcpy(h_y, d_y, size, hipMemcpyDeviceToHost));
    CHECK(hipDeviceSynchronize());

    CHECK(hipFree(d_x));
    CHECK(hipFree(d_y));

    free(h_x);
    free(h_y);
}
saxpy$
```

hipify-perl

translating a file to standard output

but can also

- translate in place
- preserve orig copy
- recursively do folders

```cpp
#include <hip/hip_runtime.h>
#include <cassert>
#include "cuda2hip.h"

__constant__ float a = 2.0f;

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

#define CHECK(call) assert(call == cudaSuccess)

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

    float* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);
    assert(h_x != nullptr);
    assert(h_y != nullptr);

    float* d_x;
    float* d_y;
    CHECK(cudaMalloc(&d_x, size));
    CHECK(cudaMalloc(&d_y, size));

    CHECK(cudaMemcpy(d_x, h_x, size, cudaMemcpyHostToDevice));
    CHECK(cudaMemcpy(d_y, h_y, size, cudaMemcpyHostToDevice));

    int num_blocks = 2;
    int num_threads = 128;
    saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y);

    CHECK(cudaMemcpy(h_y, d_y, size, cudaMemcpyDeviceToHost));
    CHECK(cudaDeviceSynchronize());

    CHECK(cudaFree(d_x));
    CHECK(cudaFree(d_y));

    free(h_x);
    free(h_y);
}
```

```cpp
#define cudaSuccess             hipSuccess
#define cudaMalloc              hipMalloc
#define cudaMemcpy              hipMemcpy
#define cudaMemcpyHostToDevice  hipMemcpyHostToDevice
#define cudaMemcpyDeviceToHost  hipMemcpyDeviceToHost
#define cudaDeviceSynchronize   hipDeviceSynchronize
#define cudaFree                hipFree
```

alternatively

- create a file with renaming macros
- include conditionally, depending on target

```cpp
#include <cassert>
#include <cstdlib>
#include <cstdio>


const float a = 2.0f;

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

    float* x = (float*)malloc(size);
    float* y = (float*)malloc(size);
    assert(x != nullptr);
    assert(y != nullptr);


    for (int i = 0; i < n; ++i)
        y[i] += a*x[i];

    free(x);
    free(y);
}

```

alternatively

- just write CPU code

```cpp
#include <cassert>
#include <cstdlib>
#include <cstdio>
#include <omp.h>

const float a = 2.0f;

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

    float* x = (float*)malloc(size);
    float* y = (float*)malloc(size);
    assert(x != nullptr);
    assert(y != nullptr);

    #pragma omp target teams distribute parallel for map(to:x[0:n]) map(tofrom:y[0:n])
    for (int i = 0; i < n; ++i)
        y[i] += a*x[i];

    free(x);
    free(y);
}
```

alternatively

- just write CPU code
- use OpenMP® target offload constructs

Kokkos and RAJA

- portability frameworks based on C++
- portability to CPUs & GPUs – AMD, Intel, NVIDIA
- basic parallel processing constructs
- multidimensional arrays
- etc., etc., etc.

Kokkos

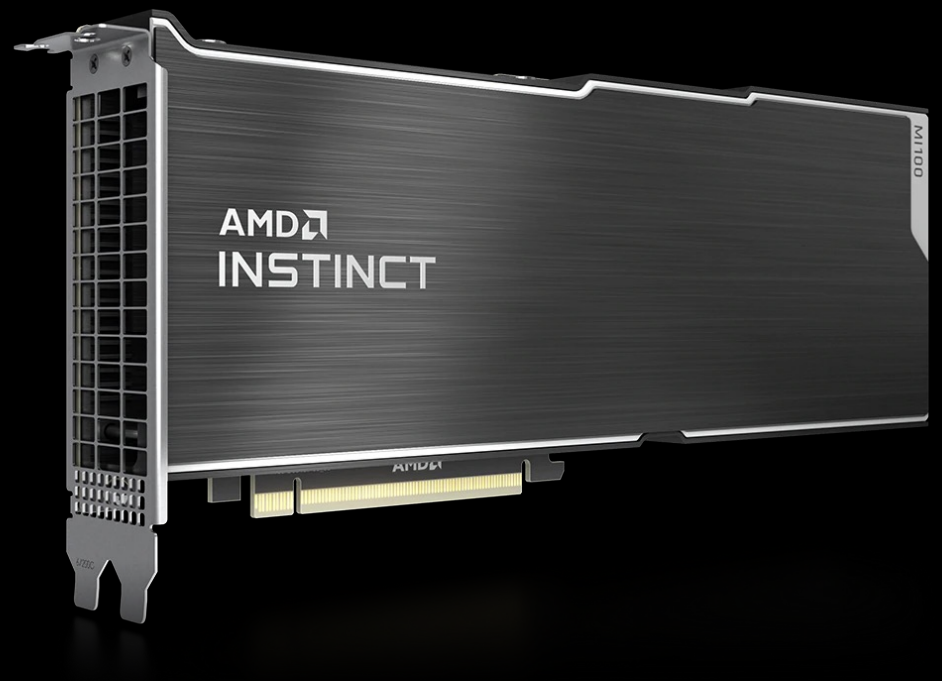
- originates from Sandia National Laboratory
- https://kokkos.org/
- https://github.com/kokkos

RAJA

- originates from Lawrence Livermore
- https://raja.readthedocs.io
- https://github.com/LLNL/RAJA

hardware intro

AMD GPUs



Radeon™ Graphics Cards
RDNA architecture
E.g.:

- RX 6000 Series
- RX 7000 Series

AMD Instinct™ Accelerators
CDNA architecture
E.g.:

- MI100
- MI200
- MI300

AMD in HPC

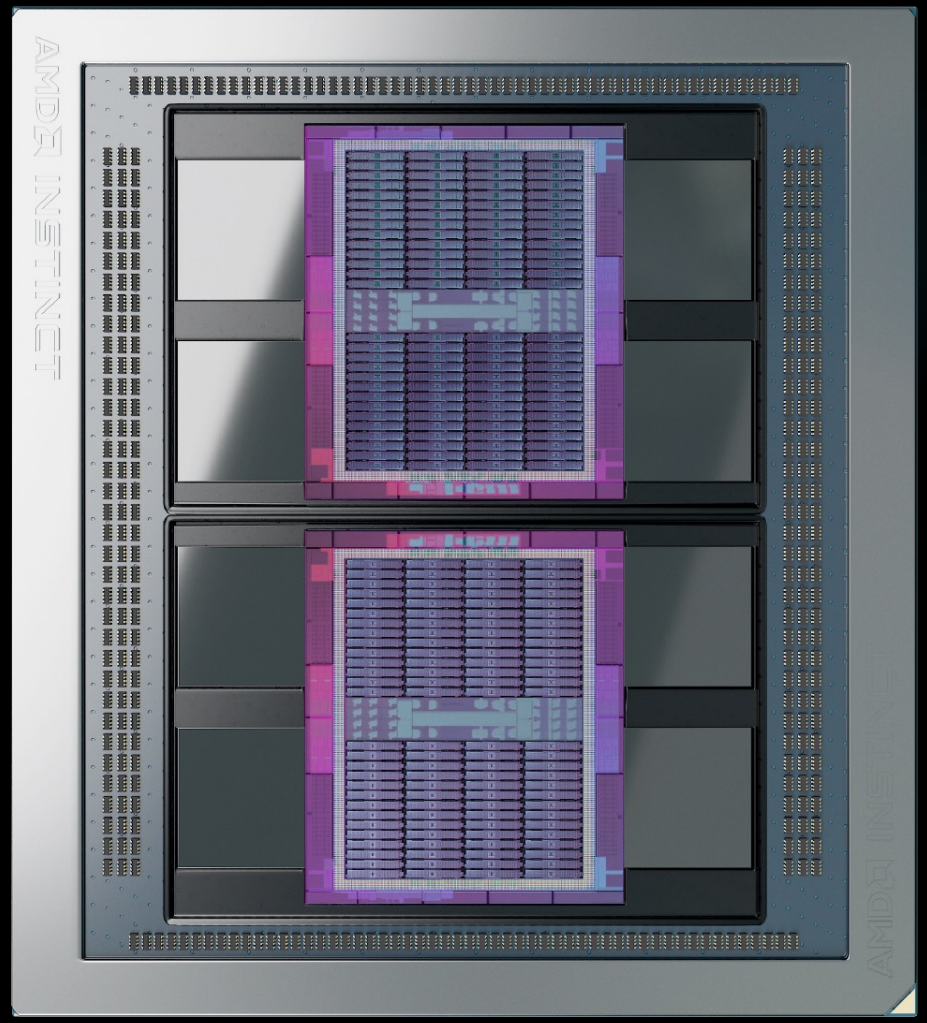
Frontier@ORNL

- currently the largest machine in the world
- the first computer to cross 1 exaFLOPS
- AMD EPYC CPUs
- AMD Instinct GPUs

LUMI@CSC

- currently the largest machine in Europe
- 3rd fastest in the world
- AMD EPYC CPUs
- AMD Instinct GPUs

AMD Instinct™ MI200



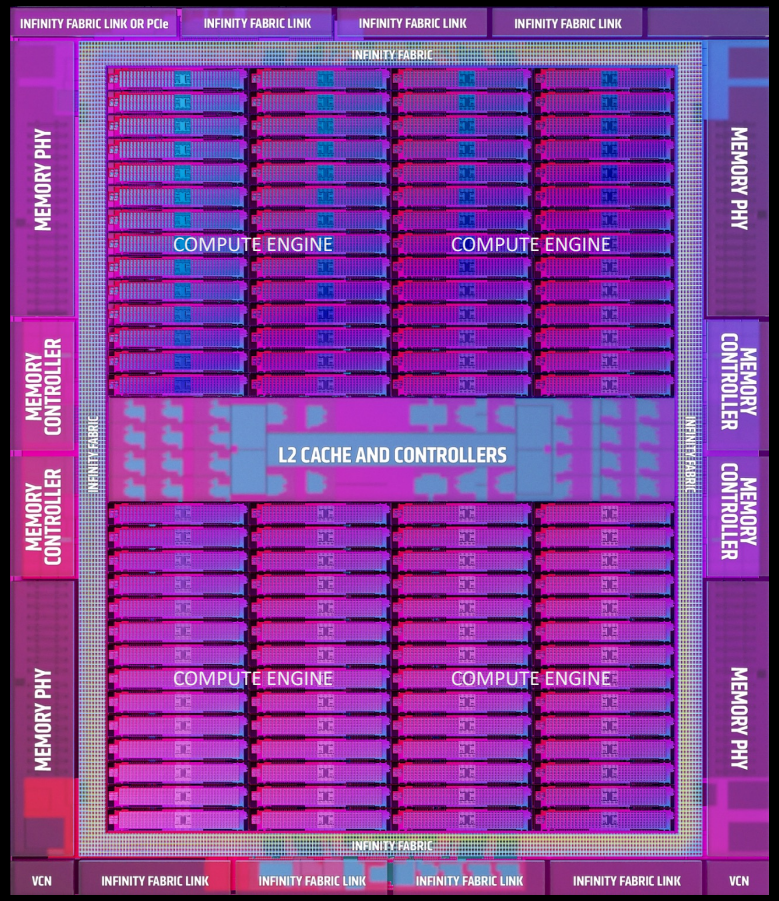
AMD INSTINCT™ MI250X

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

AMD Instinct™ MI200



2ND GENERATION CDNA ARCHITECTURE

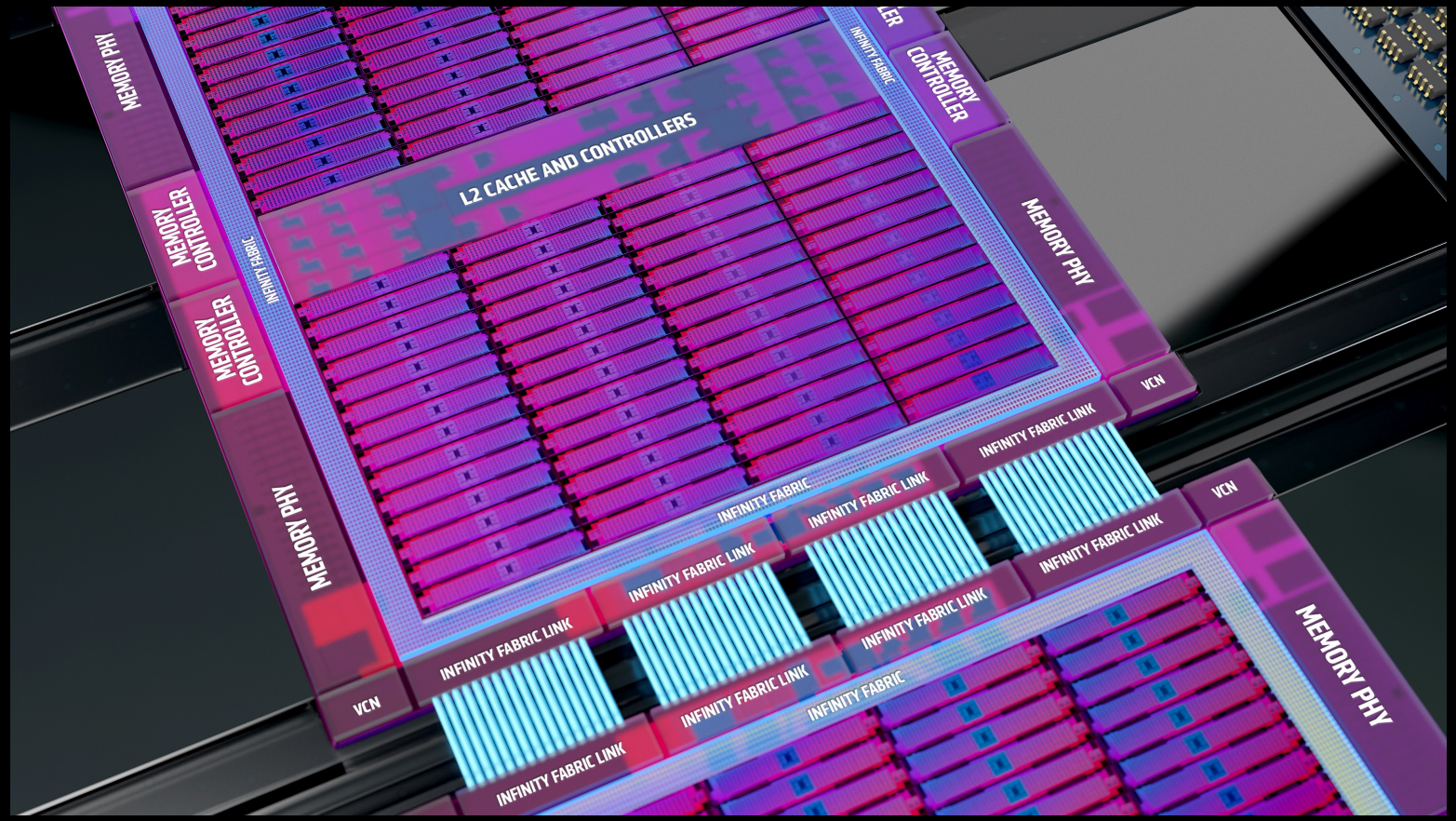
TAILORED-BUILT FOR HPC & AI

- 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

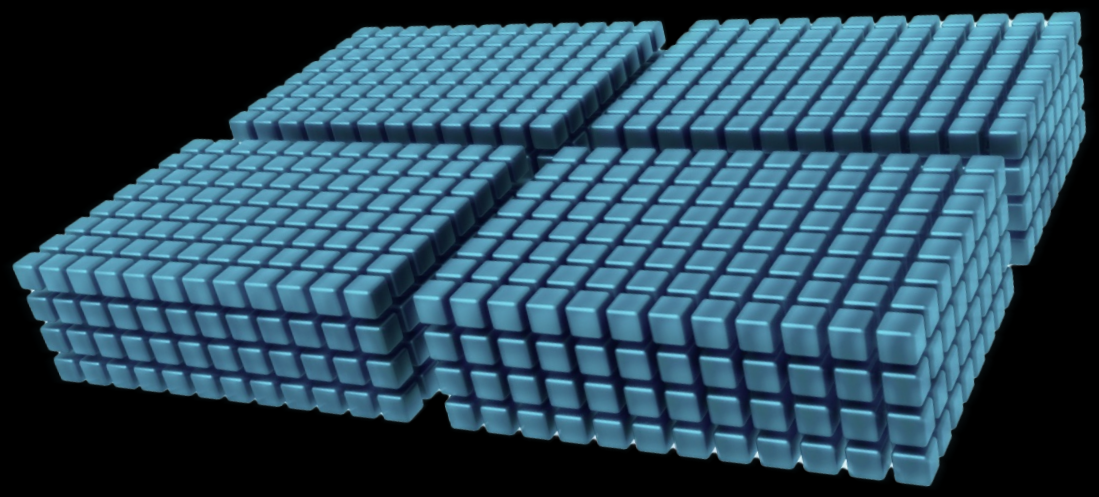
TWO GPU DIES IN PACKAGE TO MAXIMIZE COMPUTE & DATA THROUGHPUT

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



2nd GENERATION MATRIX CORES

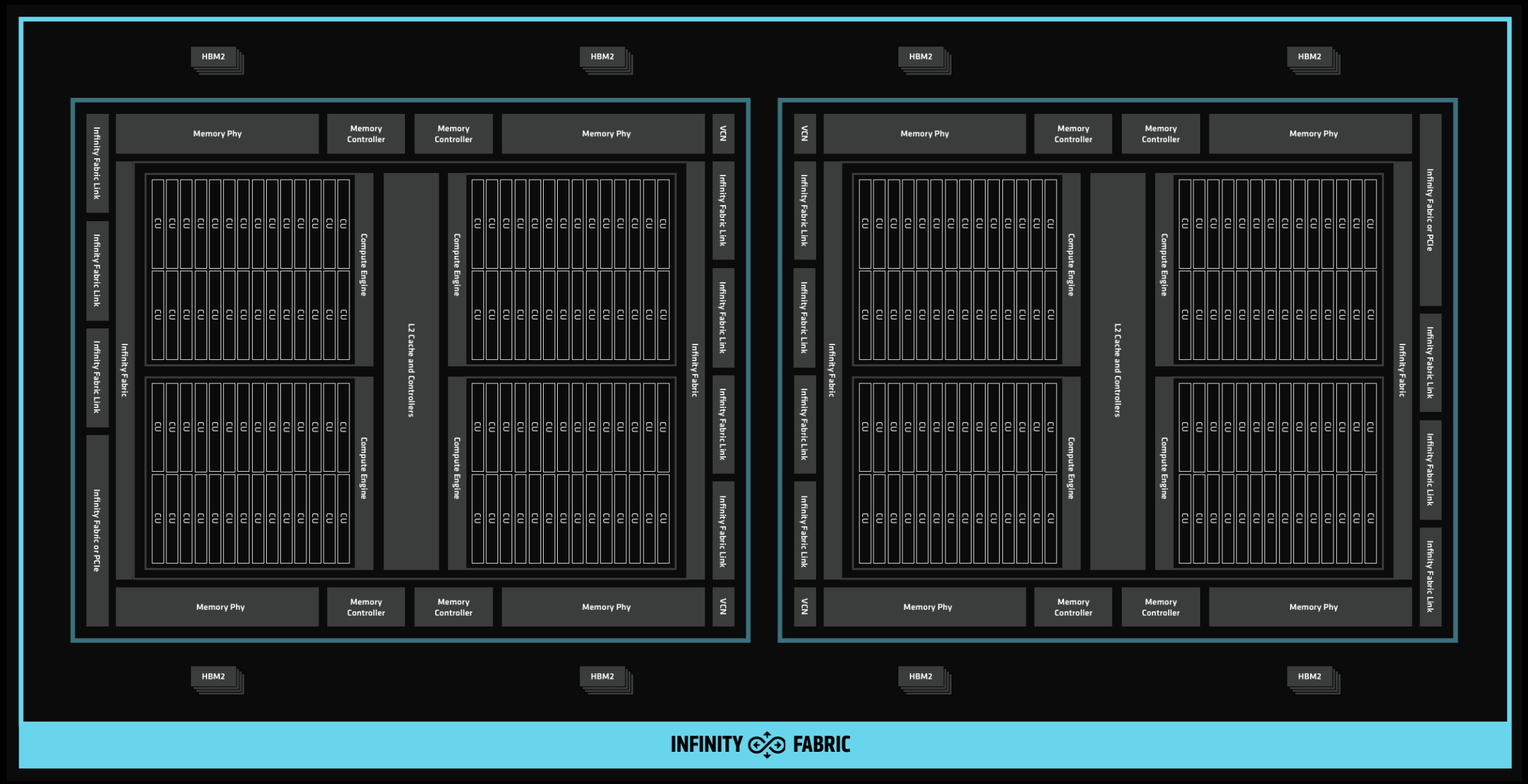
OPTIMIZED COMPUTE UNITS FOR SCIENTIFIC COMPUTING



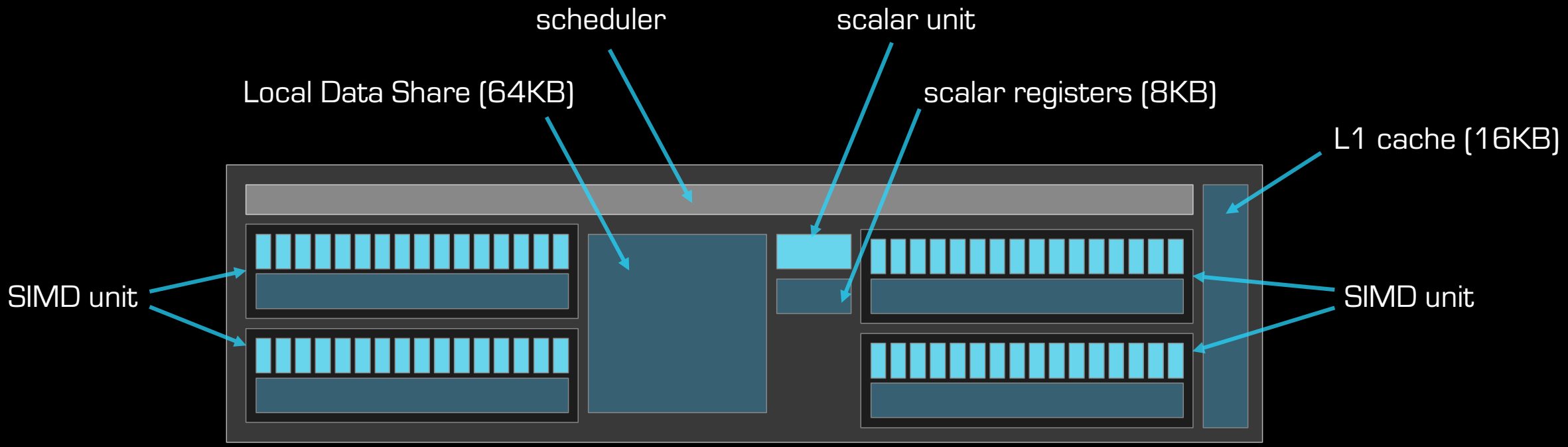
DOUBLE PRECISON (FP64)
MATRIX CORE THROUGHPUT
REPRESENTATION

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

AMD Instinct™ MI200



MI200 compute unit



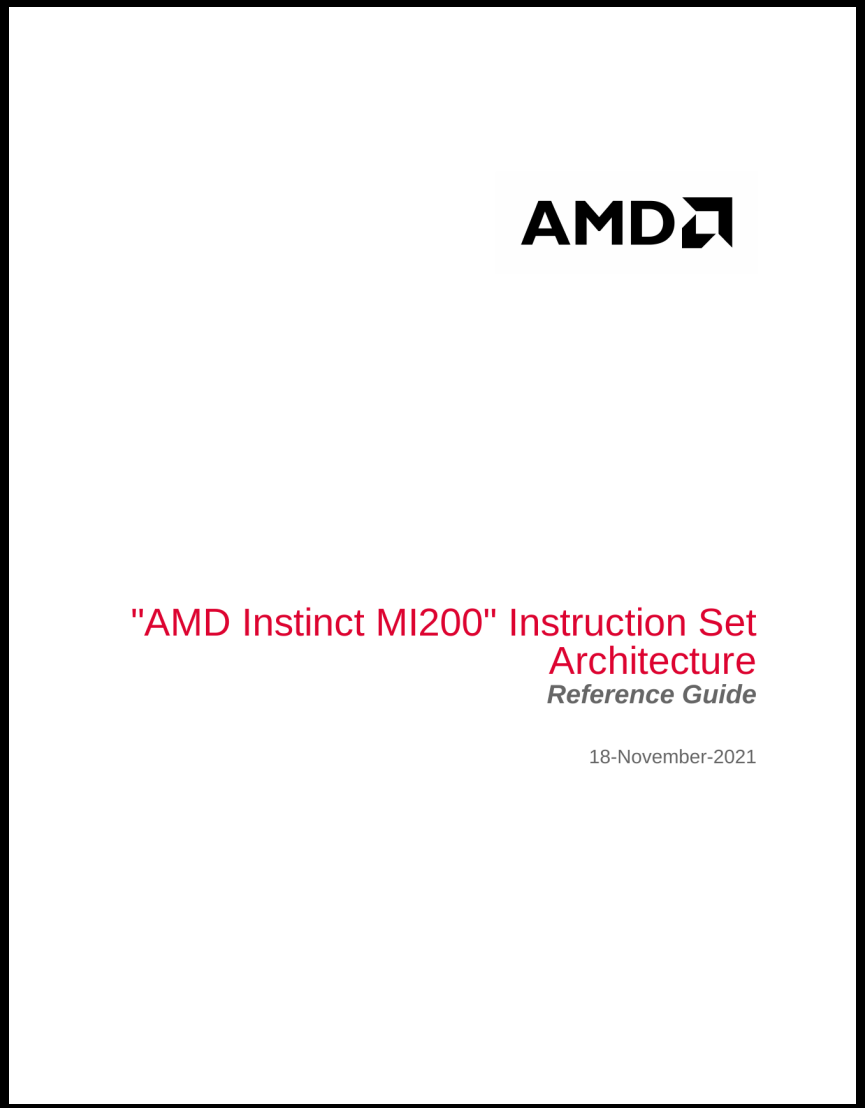
each SIMD unit

- has 16 SIMD lanes
- operates on vectors (waves) of size 64
- handles up to 10 waves simultaneously

optimization fundamentals

- thread divergence / SIMDization – 64 lanes!
- shared memory bank conflicts
- device memory access coalescing
- register pressure – avoiding spills
- occupancy – hiding all kinds of latencies

ISA is public



- the Instruction Set Architecture is public
- there is no intermediate layer like PTX
- you can write assembly code
- or compile to assembly for inspection

AMD Instinct™ MI300

The world's first integrated data center CPU + GPU

AMD INSTINCT™

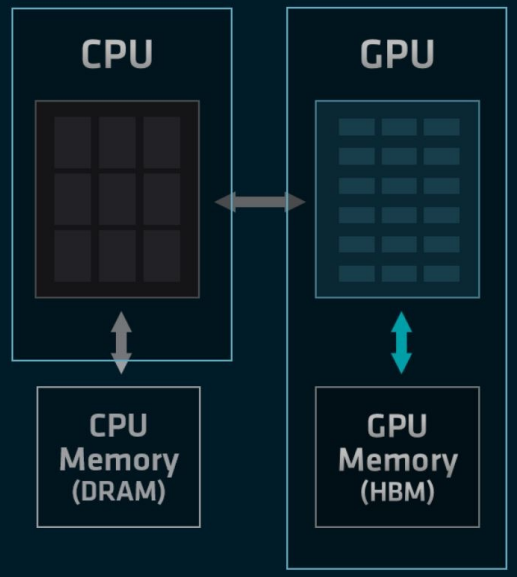
MI300

Breakthrough architecture to power the exascale AI era

UNIFIED MEMORY APU ARCHITECTURE BENEFITS

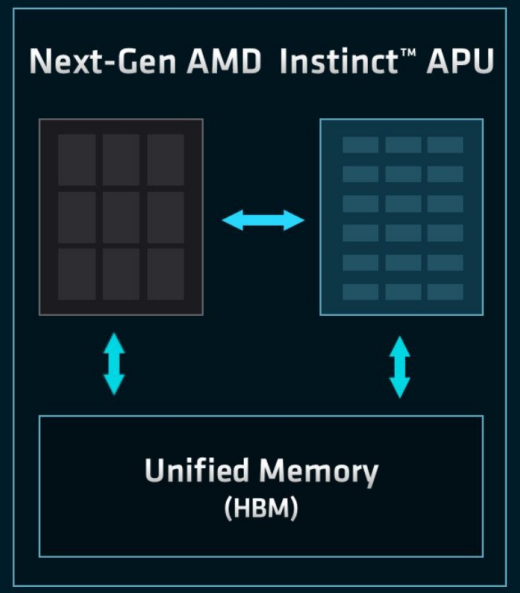
AMD CDNA™ 2 Coherent Memory Architecture

- Simplifies Programming
- Low Overhead 3rd Gen Infinity Interconnect
- Industry Standard Modular Design



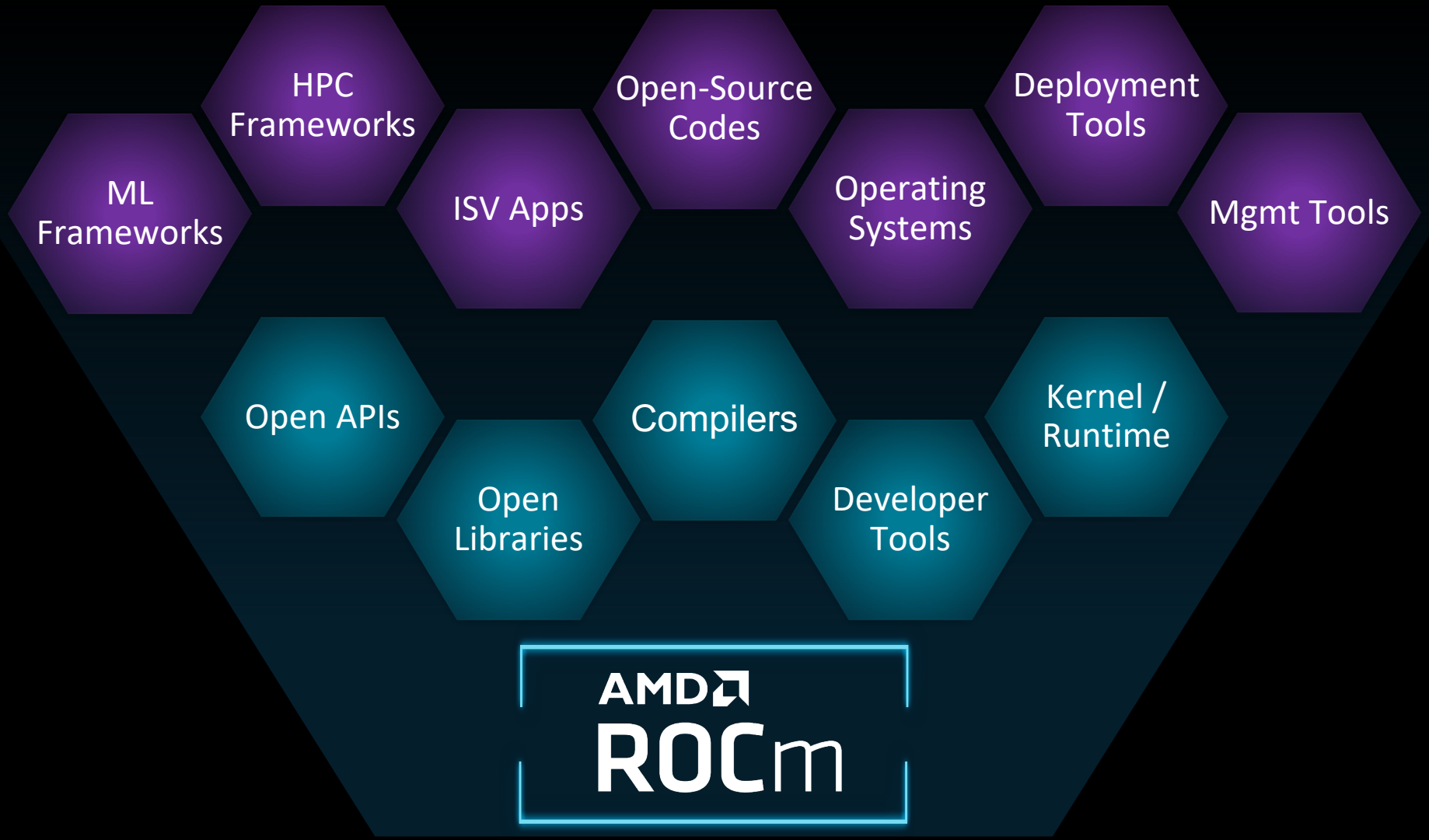
AMD CDNA™ 3 Unified Memory APU Architecture

- Eliminates Redundant Memory Copies
- High-Efficiency 4th Gen AMD Infinity Architecture
- Low TCO with Unified Memory APU Package

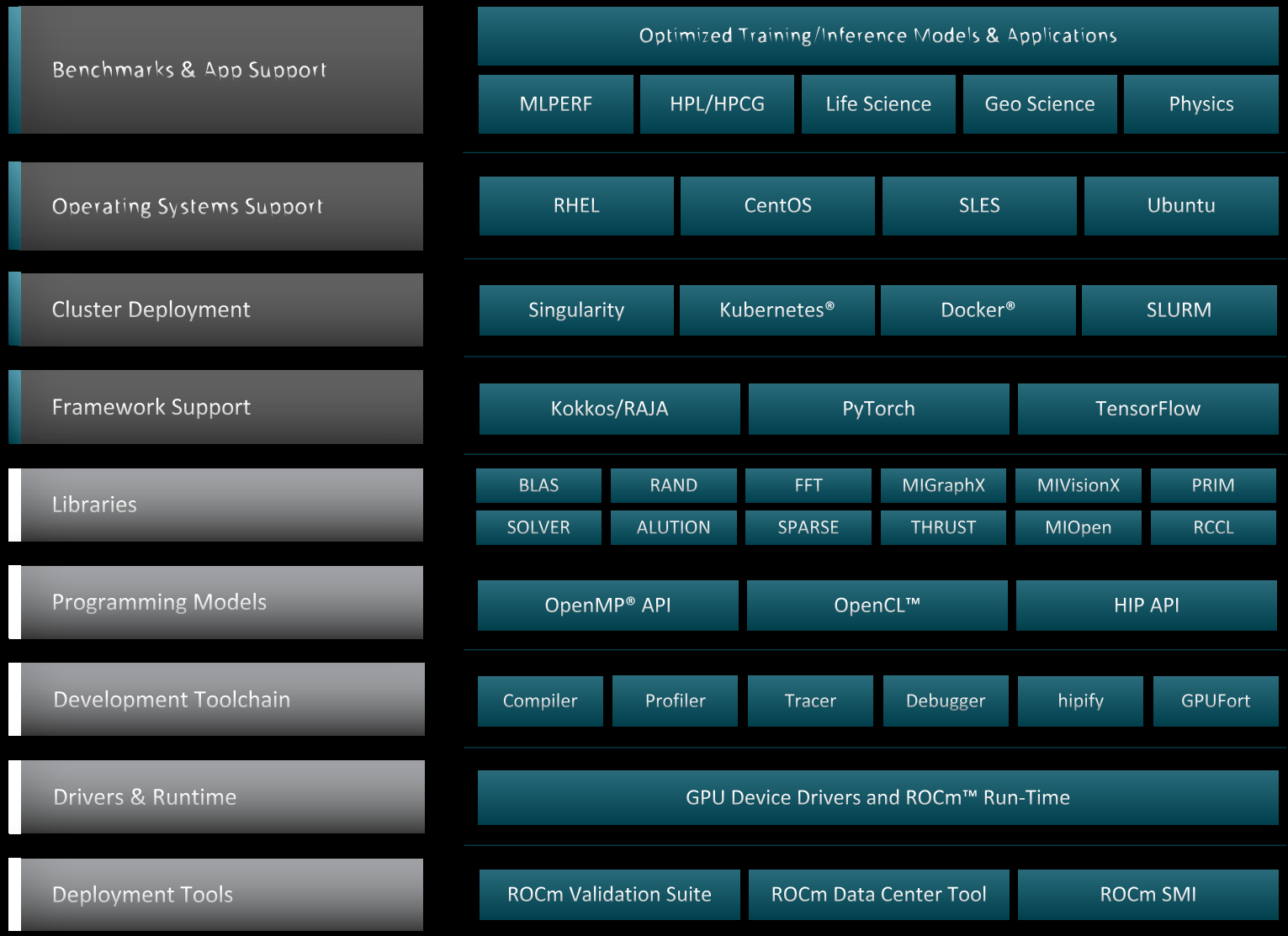


ROCm overview

AMD ROCm™ Open Software Platform for GPU Compute



AMD ROCm™ Open Software Platform for GPU Compute



libraries

rocBLAS / hipBLAS
- basic operations on dense matrices

https://github.com/ROCmSoftwarePlatform/rocBLAS
https://github.com/ROCmSoftwarePlatform/hipBLAS

rocSOLVER
- dense linear algebra solvers

https://github.com/ROCmSoftwarePlatform/rocSOLVER

rocSPARSE / hipSPARSE
- basic operations on sparse matrices

https://github.com/ROCmSoftwarePlatform/rocSPARSE
https://github.com/ROCmSoftwarePlatform/hipSPARSE

rocALUTION
- sparse linear algebra solvers

https://github.com/ROCmSoftwarePlatform/rocALUTION

rocFFT / hipFFT
- Fast Fourier transforms

https://github.com/ROCmSoftwarePlatform/rocFFT
https://github.com/ROCmSoftwarePlatform/hipFFT

rocRAND / hipRAND
- random number generation

https://github.com/ROCmSoftwarePlatform/rocRAND
https://github.com/ROCmSoftwarePlatform/hipRAND

rocPRIM / hipCUB / rocThrust
- scan, sort, reduction, etc.

https://github.com/ROCmSoftwarePlatform/rocPRIM
https://github.com/ROCmSoftwarePlatform/hipCUB
https://github.com/ROCmSoftwarePlatform/rocThrust

also open source

the compiler
- https://github.com/ROCmSoftwarePlatform/llvm-project

the runtime
- https://github.com/RadeonOpenCompute/ROCR-Runtime

the debugger
- https://github.com/ROCm-Developer-Tools/ROCgdb

the profiler
- https://github.com/ROCm-Developer-Tools/rocprofiler

the HPL benchmark
- https://github.com/ROCmSoftwarePlatform/rocHPL

the HPCG benchmark
- https://github.com/ROCmSoftwarePlatform/rocHPCG

etc.

basic debugging tools

rocgdb

- is a fork of the GNU GDB
- allows for using standard GDB tools, GUIs, etc.
- allows for debugging of GPU kernels
  - inspect the assembly code
  - step through the assembly code
  - inspect hardware registers, etc.
- https://github.com/ROCm-Developer-Tools/ROCgdb

cause a segfault

```
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.

compile 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
- fgx90a – MI200

```
saxpy$ hipcc   --offload-arch=gfx906 -o saxpy saxpy.hip.cpp
```

run

```
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=gfx906 -o saxpy saxpy.hip.cpp
saxpy$ ./saxpy
```

crash

```
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=gfx906 -o saxpy saxpy.hip.cpp
saxpy$ ./saxpy
Memory access fault by GPU node-4 (Agent handle: 0x19dee10) on address (nil). Reason: Page not
 present or supervisor privilege.
Aborted (core dumped)
saxpy$
```

run 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,
26      hipDeviceSynchronize();
27  }
28
```

```
saxpy$ rocgdb saxpy
```

get some 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,
26      hipDeviceSynchronize();
27  }
28
```

Reports segmentation fault in the saxpy kernel.

```
(gdb) run
Starting program: /mnt/shared/codes/saxpy/saxpy
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffff4d36700 (LWP 67093)]
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])]
0x00007fffe8a01094 in saxpy(int, float const*, int, float*, int) () from file:///mnt/shared/co
des/saxpy/saxpy#offset=8192&size=13992
(gdb)
```

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=gfx906 -o saxpy saxpy.hip.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,
26      hipDeviceSynchronize();
27  }
28
```

more details
- what kernel
- what file:line

```
(gdb) run
Starting program: /mnt/shared/codes/saxpy/saxpy
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffff4d36700 (LWP 67682)]
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])]
0x00007fffe8a01094 in saxpy (n=<optimized out>, x=<optimized out>, incx=<optimized out>, y=<op
timized out>, incy=<optimized out>) at saxpy.hip.cpp:10
10              y[i] += a*x[i];
(gdb)
```

But where's my stack trace?

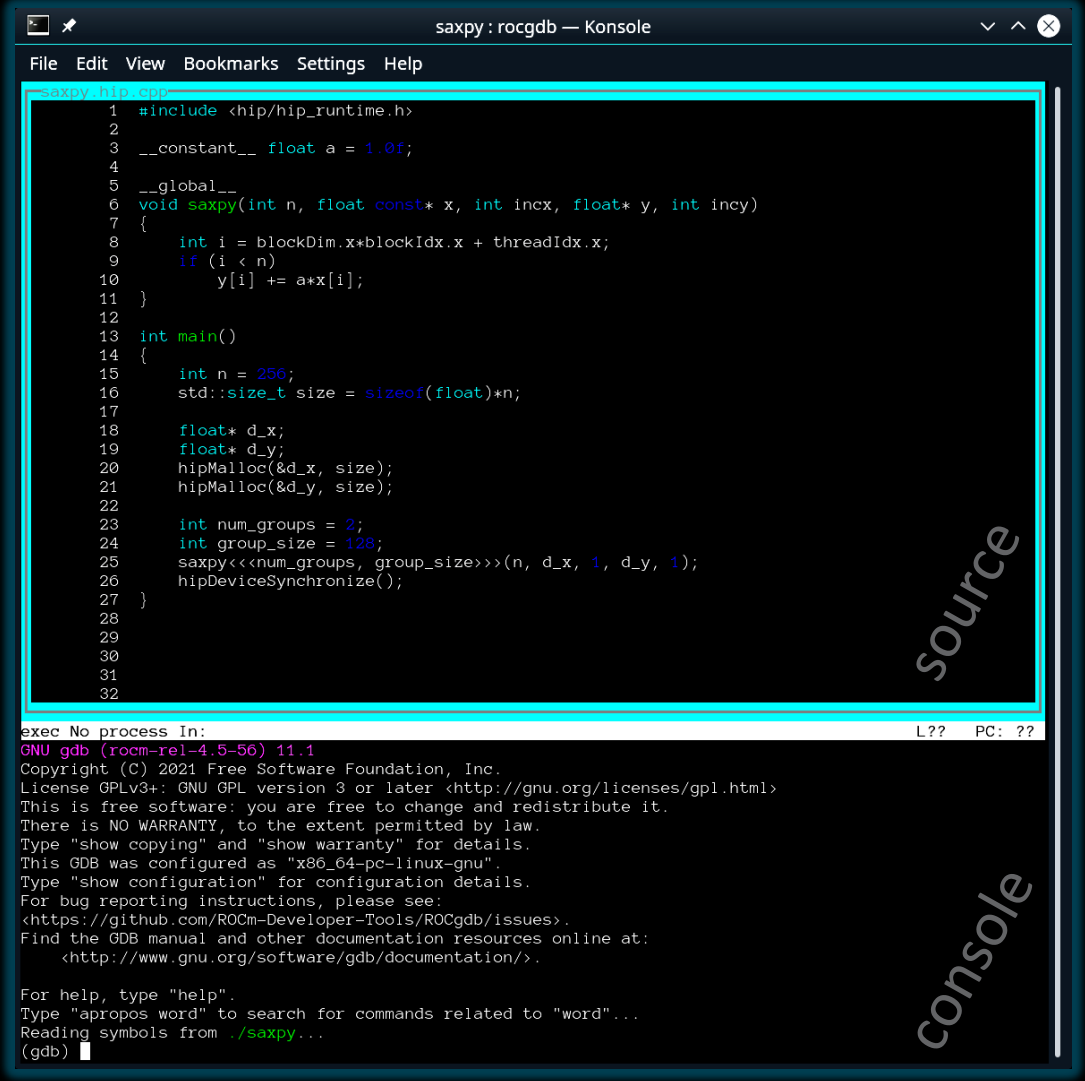
list threads

```
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
```

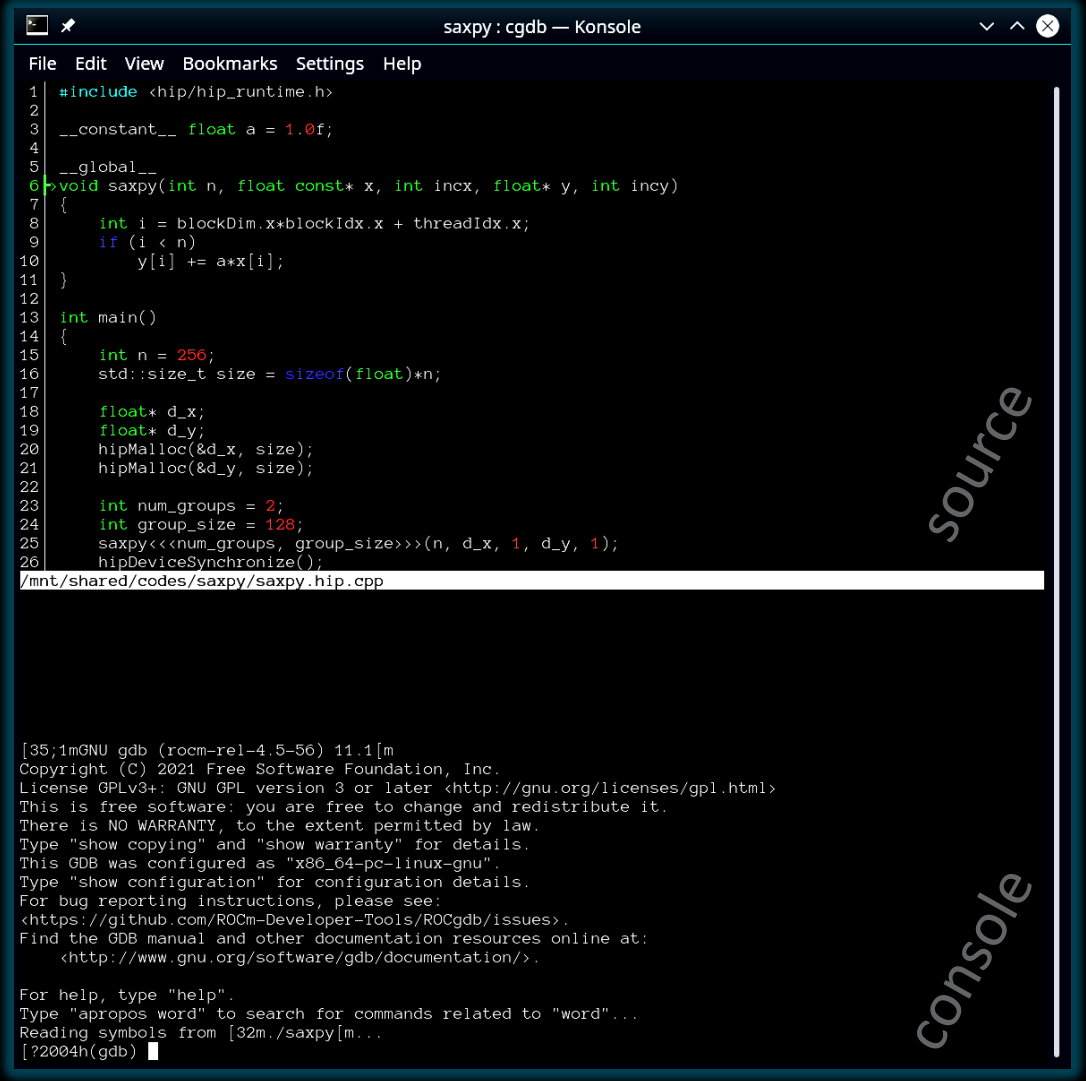
```
(gdb) i th
  Id   Target Id                                      Frame
  1    Thread 0x7ffff7fb6880 (LWP 67674) "saxpy" 0x00007ffff57f5102 in rocr::core::InterruptSigr
   from /opt/rocm-4.5.0/hip/lib/../../lib/libhsa-runtime64.so.1
  2    Thread 0x7ffff4d36700 (LWP 67682) "saxpy" 0x00007ffff5f6d317 in ioctl () at ../sysdeps/ur
* 3    AMDGPU Wave 1:2:1:1 (0,0,0)/0 "saxpy"     0x00007fffe8a01094 in saxpy (n=<optimized out>,
  4    AMDGPU Wave 1:2:1:2 (0,0,0)/1 "saxpy"     0x00007fffe8a01094 in saxpy (n=<optimized out>,
  5    AMDGPU Wave 1:2:1:3 (1,0,0)/0 "saxpy"     0x00007fffe8a01094 in saxpy (n=<optimized out>,
  6    AMDGPU Wave 1:2:1:4 (1,0,0)/1 "saxpy"     0x00007fffe8a01094 in saxpy (n=<optimized out>,
(gdb)
```

“GUIs”

rocgdb -tui saxpy



cgdb -d rocgdb saxpy



AMD_LOG_LEVEL=3

saxpy : bash — Konsole

File Edit View Bookmarks Settings Help

```
jakurzak@jakurzak-MS-7B09:/mnt/shared/codes/saxpy$ AMD_LOG_LEVEL=3 ./saxpy
:3:rocdevice.cpp            :432 : 714826105802 us: Initializing HSA stack.
:3:comgrctx.cpp             :33  : 714826149967 us: Loading COMGR library.
:3:rocdevice.cpp            :204 : 714826155354 us: Numa selects cpu agent[2]=0x10ae220(fine=0x10ae430,coarse=0x10aebb0, kern_arg=0x10e7e20) for gpu
:1:rocdevice.cpp            :1573: 714826155633 us: HSA_AMD_AGENT_INFO_SVM_DIRECT_HOST_ACCESS query failed.
:3:rocdevice.cpp            :1577: 714826155640 us: HMM support: 0, xnack: 0, direct host access: 0

:3:hip_context.cpp          :49  : 714826157657 us: Direct Dispatch: 1
:3:rocdevice.cpp            :2047: 714826157883 us: device=0x1107c60, freeMem_ = 0xefffc00
:3:hip_memory.cpp           :480 : 714826157896 us: 123767: [7f5b72543880] hipMalloc: Returned hipSuccess : 0x7f5b6e200000
:3:hip_memory.cpp           :478 : 714826157916 us: 123767: [7f5b72543880] hipMalloc ( 0x7fff555f25c0, 1024 )
:3:rocdevice.cpp            :2047: 714826157926 us: device=0x1107c60, freeMem_ = 0xefff800
:3:hip_memory.cpp           :480 : 714826157930 us: 123767: [7f5b72543880] hipMalloc: Returned hipSuccess : 0x7f5b6e201000: duration: 14 us
:3:hip_platform.cpp         :202 : 714826157940 us: 123767: [7f5b72543880] __hipPushCallConfiguration ( {2,1,1}, {128,1,1}, 0, stream:<null> )
:3:hip_platform.cpp         :206 : 714826157950 us: 123767: [7f5b72543880] __hipPushCallConfiguration: Returned hipSuccess :
:3:hip_platform.cpp         :213 : 714826157958 us: 123767: [7f5b72543880] __hipPopCallConfiguration ( {1,0,2153245}, {2,0,2157640}, 0x7fff555f25d8,
:3:hip_platform.cpp         :222 : 714826157961 us: 123767: [7f5b72543880] __hipPopCallConfiguration: Returned hipSuccess :
:3:hip_module.cpp           :492 : 714826157970 us: 123767: [7f5b72543880] hipLaunchKernel ( 0x2007b8, {2,1,1}, {128,1,1}, 0x7fff555f2610, 0, stream
:3:devprogram.cpp           :2668: 714826158275 us: Using Code Object V4.
:3:hip_module.cpp           :363 : 714826167980 us: 123767: [7f5b72543880] ihipModuleLaunchKernel ( 0x0x1141e20, 256, 1, 1, 128, 1, 1, 0, stream:<nu
:3:rocdevice.cpp            :2623: 714826168023 us: number of allocated hardware queues with low priority: 0, with normal priority: 0, with high pri
:3:rocdevice.cpp            :2695: 714826186484 us: created hardware queue 0x7f5b72558000 with size 1024 with priority 1, cooperative: 0
:3:devprogram.cpp           :2668: 714826439826 us: Using Code Object V4.
:3:rocvirtual.cpp           :748 : 714826441265 us: [7f5b72543880]!     Arg0:   = val:256
:3:rocvirtual.cpp           :669 : 714826441274 us: [7f5b72543880]!     Arg1:   = ptr:0x7f5b6e200000 obj:[0x7f5b6e200000-0x7f5b6e200400]
:3:rocvirtual.cpp           :748 : 714826441277 us: [7f5b72543880]!     Arg2:   = val:1
:3:rocvirtual.cpp           :669 : 714826441279 us: [7f5b72543880]!     Arg3:   = ptr:0x7f5b6e201000 obj:[0x7f5b6e201000-0x7f5b6e201400]
:3:rocvirtual.cpp           :748 : 714826441281 us: [7f5b72543880]!     Arg4:   = val:1
:3:rocvirtual.cpp           :2677: 714826441284 us: [7f5b72543880]!     ShaderName : _Z5saxpyiPKfiPfi
:3:hip_platform.cpp         :667 : 714826441300 us: 123767: [7f5b72543880] ihipLaunchKernel: Returned hipSuccess :
:3:hip_module.cpp           :495 : 714826441313 us: 123767: [7f5b72543880] hipLaunchKernel: Returned hipSuccess :
:3:hip_device_runtime.cpp   :460 : 714826441318 us: 123767: [7f5b72543880] hipDeviceSynchronize (  )
:3:rocdevice.cpp            :2573: 714826441324 us: No HW event
:3:rocvirtual.hpp           :61  : 714826441330 us: Host active wait for Signal = (0x7f5b72576a00) for -1 ns
:3:hip_device_runtime.cpp   :472 : 714826441344 us: 123767: [7f5b72543880] hipDeviceSynchronize: Returned hipSuccess :
jakurzak@jakurzak-MS-7B09:/mnt/shared/codes/saxpy$
```

printf from a kernel

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

__global__
void print()
{
    printf("\t%d\t%d\n", int(blockIdx.x), int(threadIdx.x));
}

#define CHECK(call) assert(call == hipSuccess)

int main()
{
    int num_blocks = 4;
    int num_threads = 4;
    print<<<num_blocks, num_threads>>>();
    hipDeviceSynchronize();
}
```

```
$ ./saxpy
        1       0
        1       1
        1       2
        1       3
        2       0
        2       1
        2       2
        2       3
        3       0
        3       1
        3       2
        3       3
        0       0
        0       1
        0       2
        0       3
$
```

- you can printf() from a kernel
- best option in some situations
- inserting a printf() changes the kernel
  - will likely affect performance
  - only use when debugging
- same goes for assert()

basic performance tools

performance tools

uProf
- AMD’s classic CPU profiler
- now also with GPU support
- https://www.amd.com/en/developer/uprof.html

rocprof
- AMD’s standard GPU profiling / tracing tool
- part of the ROCm distribution

omniperf / omnitrace
- cutting-edge research tools
- https://github.com/AMDResearch/omniperf
- https://github.com/AMDResearch/omnitrace

rocprof

```
rocprof --stats ./prog …

results.csv
results.db
results.json
results.stats.csv
results.sysinfo.txt
```

	A	B	C	D	E
1	**Name**	**Calls**	**TotalDurationNs**	**AverageNs**	**Percentage**
2	DeviceStream<double>::copy_kernel(double const*, double*)	1892	4686358503	2476933	99.88
3	DeviceArray<double>::init_kernel(double*, double, double)	2	3503998	1751999	0.07
4	DeviceArray<double>::check_kernel(double*, double, double)	1	2335679	2335679	0.05

rocprof

```
rocprof --stats --basenames on ./prog …

results.csv
results.db
results.json
results.stats.csv
results.sysinfo.txt
```

	A	B	C	D	E
1	**Name**	**Calls**	**TotalDurationNs**	**AverageNs**	**Percentage**
2	copy_kernel	1656	4093064220	2471657	99.86
3	init_kernel	2	3505439	1752719	0.09
4	check_kernel	1	2400640	2400640	0.06

rocprof

```
rocprof --stats --hip-trace ./prog …

results.csv
results.db
results.hip_stats.csv
results.json
results.stats.csv
results.sysinfo.txt
```

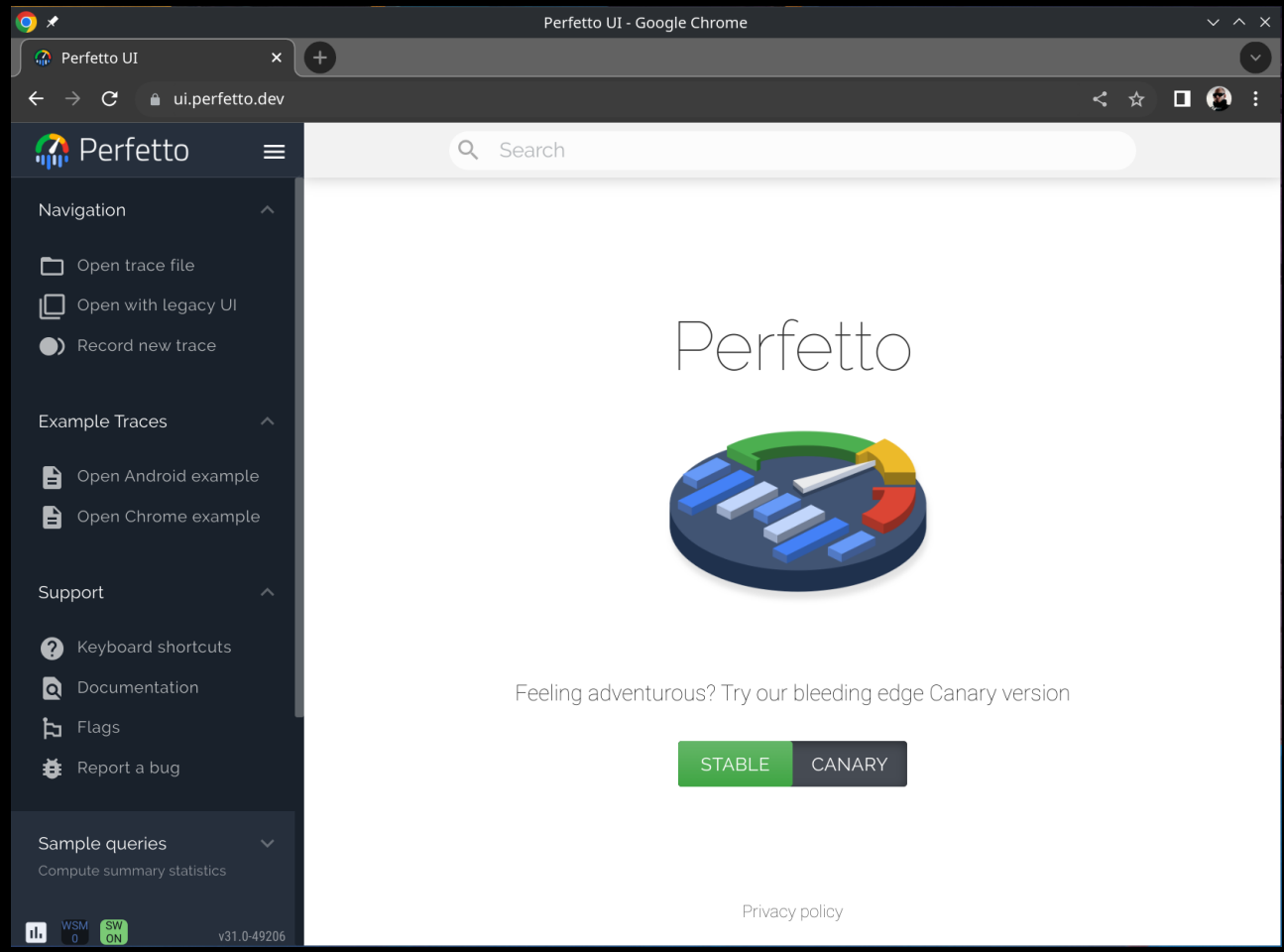
	A	B	C	D	E
1	Name	Calls	TotalDurationNs	AverageNs	Percentage
2	hipDeviceSynchronize	1659	4118533103	2482539	94.61
3	hipLaunchKernel	1659	233057606	140480	5.35
4	hipMalloc	2	950489	475244	0.02
5	hipHostMalloc	1	145842	145842	0.00
6	__hipPushCallConfiguration	1659	139180	83	0.00
7	__hipPopCallConfiguration	1659	131102	79	0.00
8	hipSetDevice	7	2130	304	0.00
9	hipGetDeviceCount	1	470	470	0.00

rocprof

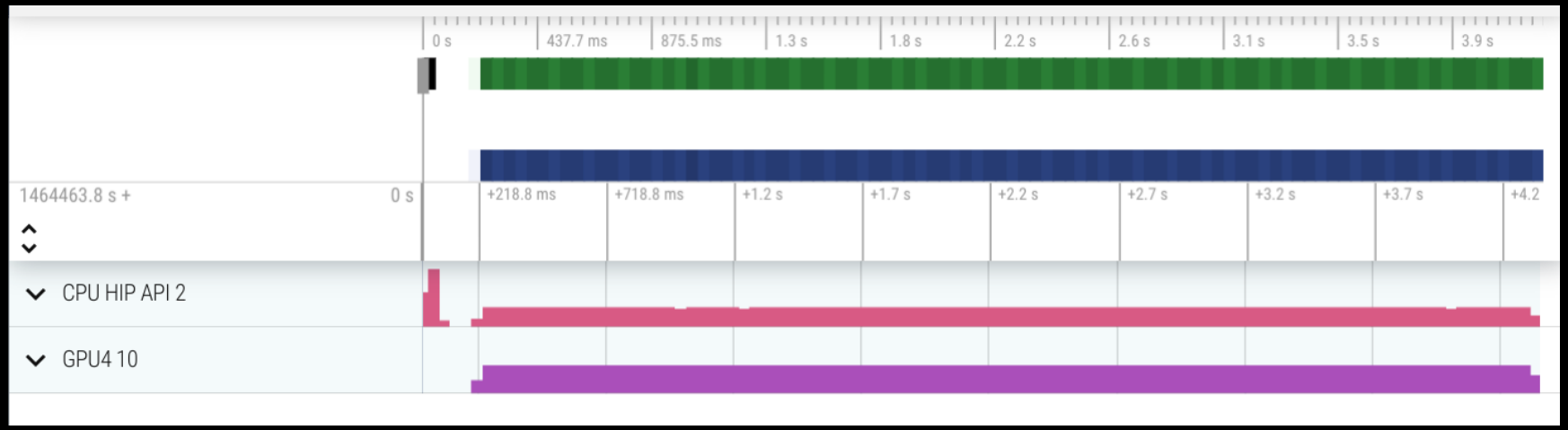
```
rocprof --stats --hip-trace ./prog …
```

```
results.csv
results.db
results.hip_stats.csv
results.json
results.stats.csv
results.sysinfo.txt
```

https://ui.perfetto.dev/



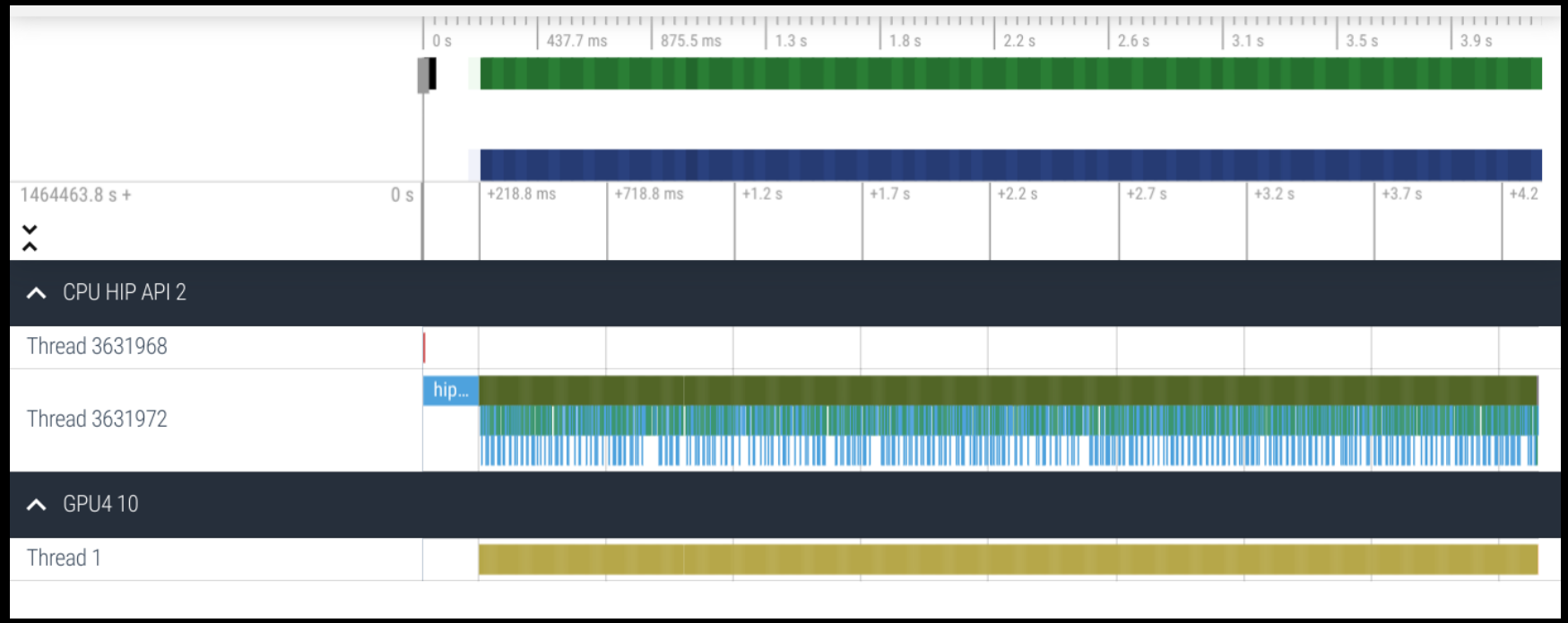
rocprof



overview of CPU / GPU activity

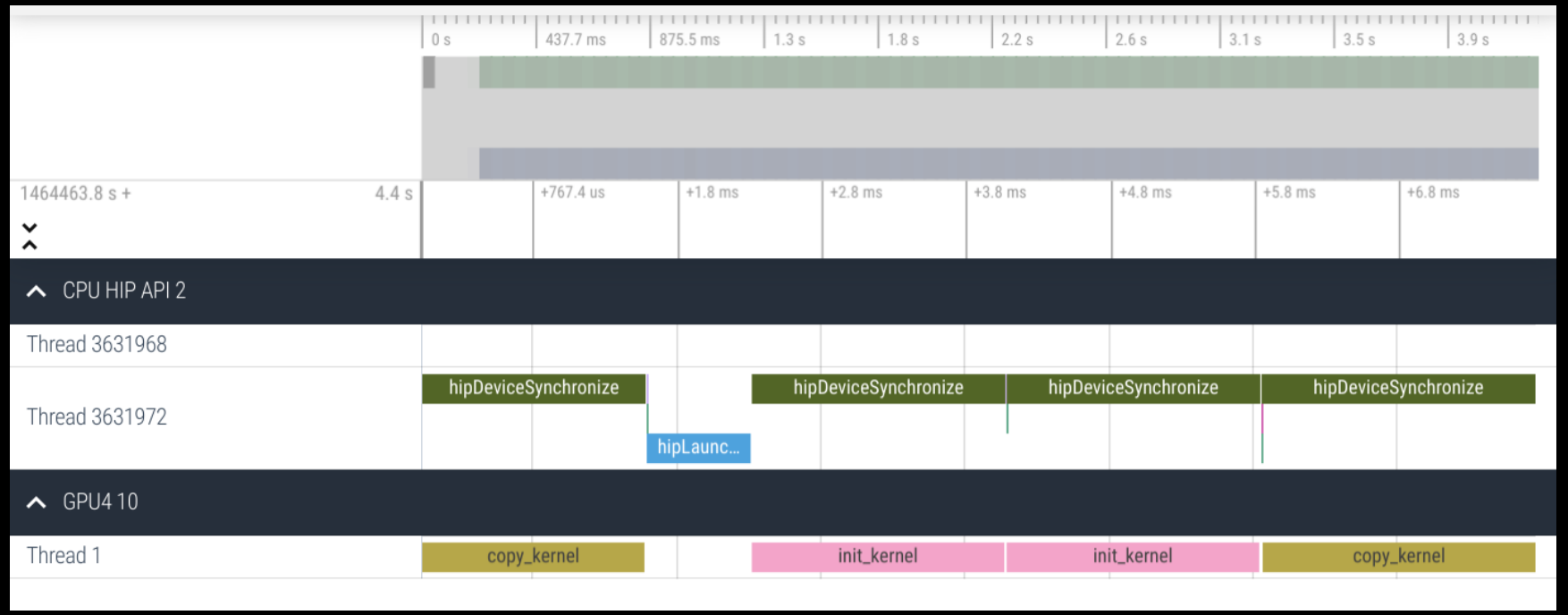
rocprof

expand to see more details



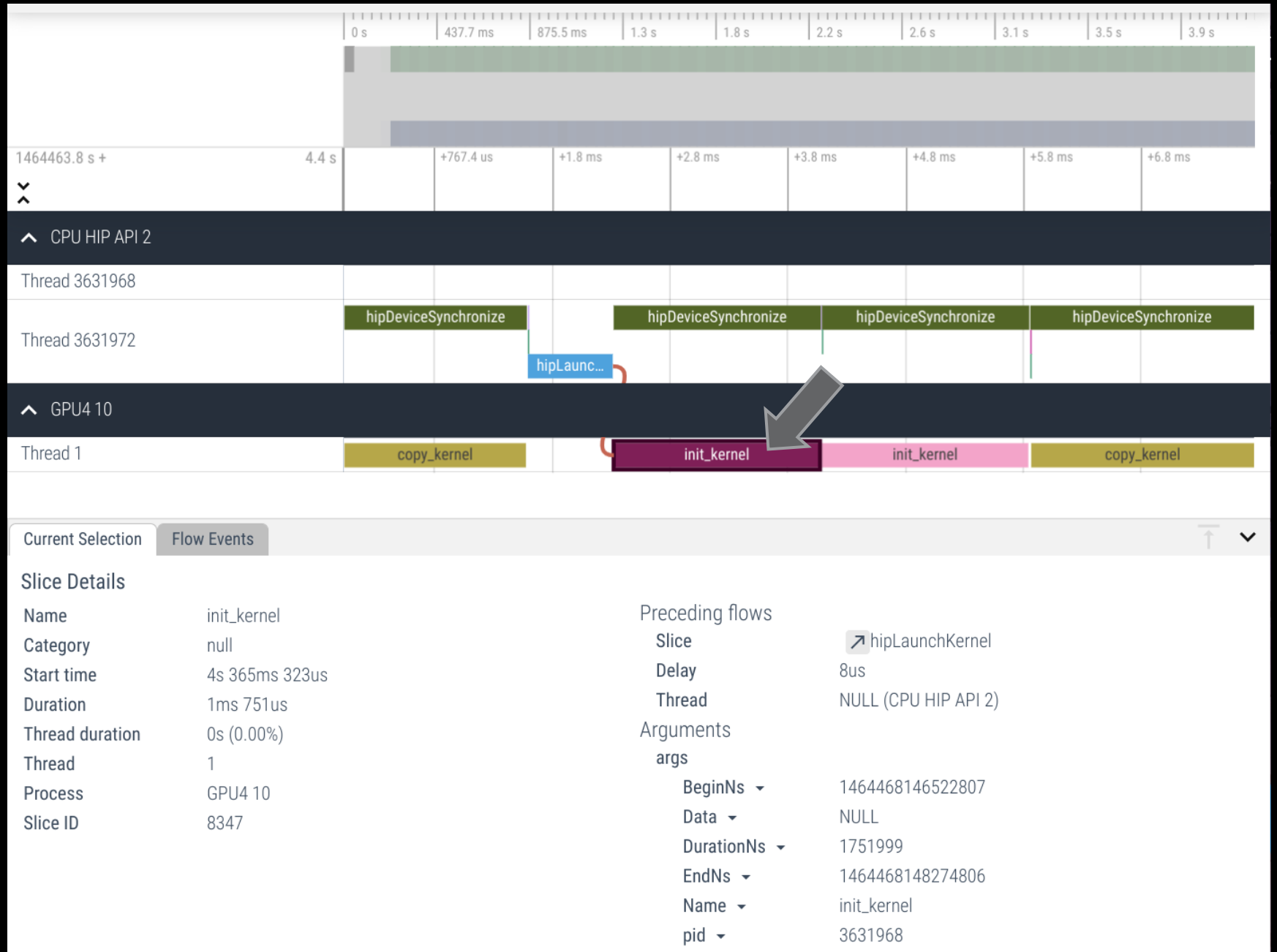
rocprof

WASD
zoom in
zoom out
scroll



rocprof

click a kernel to get more details



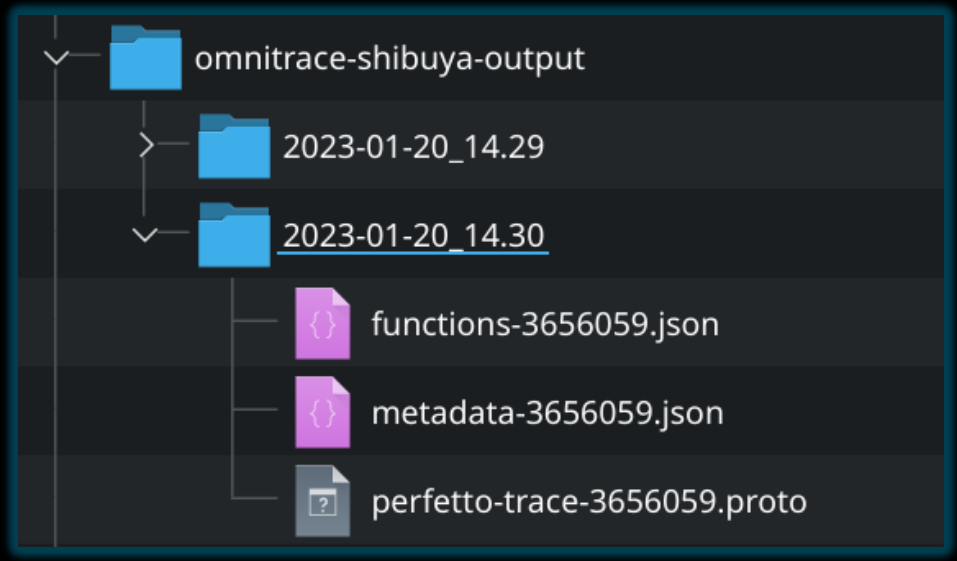
omnitrace

- very powerful
- in research stage
- based on binary instrumentation
  - dynamic instrumentation
  - binary rewriting
- can trace any activity
  - app routines
  - library routines
  - runtime routines
- can trace CPU and GPU activity
- OpenMP® and MPI support
- many more capabilities

omnitrace

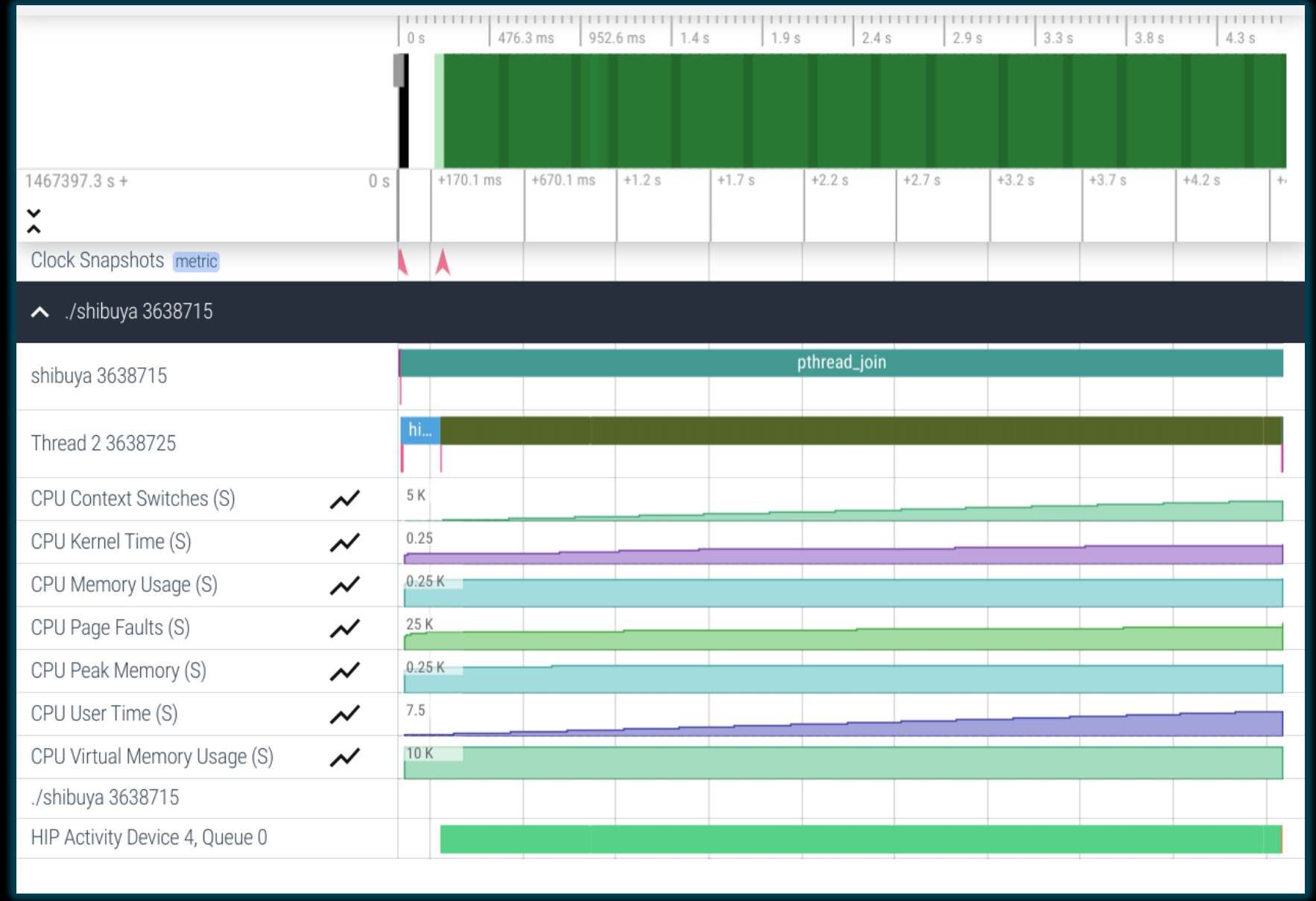
```
omnitrace -- ./prog …
```

```
Finding instrumentation functions...
 398 instrumented funcs in libamdhip64.so.5.4.50401
   2 instrumented funcs in libdrm.so.2.4.0
   8 instrumented funcs in libelf-0.186.so
  10 instrumented funcs in libz.so.1.2.11
  25 instrumented funcs in shibuya
```



https://ui.perfetto.dev/

omnitrace



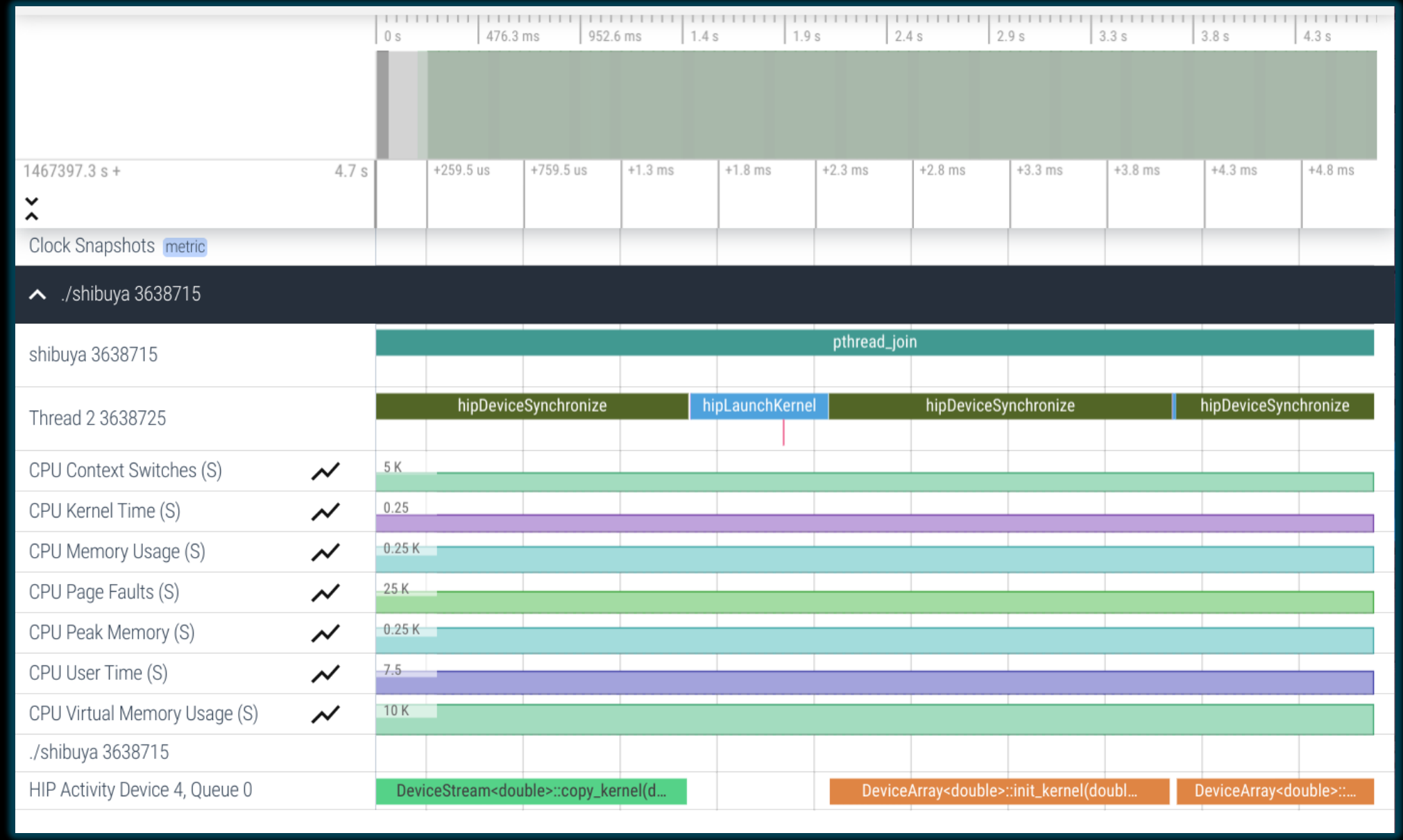
main app thread

HIP thread

CPU stats

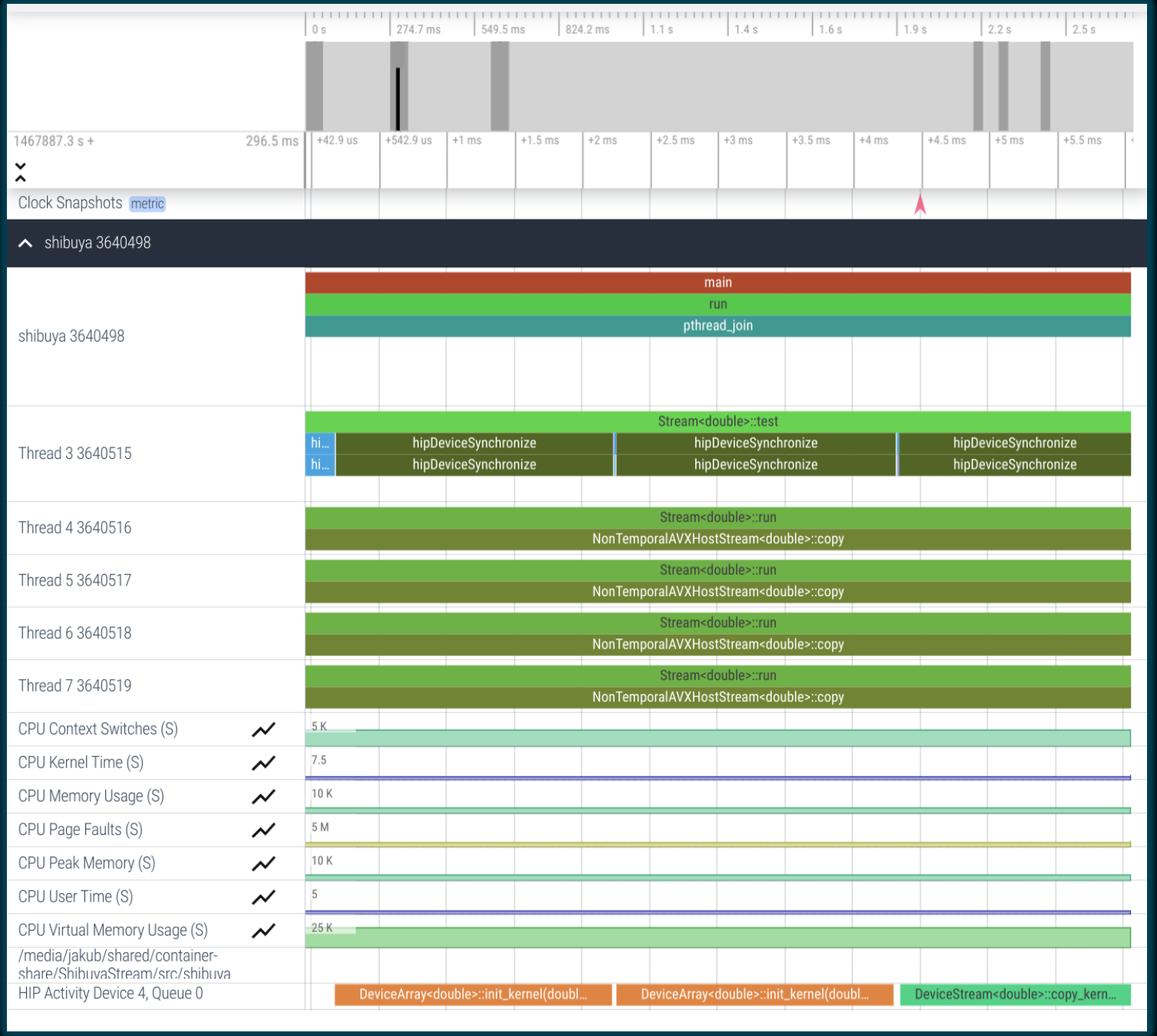
GPU activity

omnitrace



WASD
zoom in
zoom out
scroll

omnitrace



main app thread

HIP thread

app “worker” threads

CPU stats

GPU activity

ROCm
https://docs.amd.com/
https://rocmdocs.amd.com/

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

AMD Lab Notes
https://gpuopen.com/learn/amd-lab-notes/

questions?

jakub.kurzak@amd.com

DISCLAIMERS AND ATTRIBUTIONS

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

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

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

DISCLAIMERS AND ATTRIBUTIONS


AMD