GA Tech **CSE 6230 High Performance Parallel Computing AMD Accelerators** ROCm HIP Jakub Kurzak Advanced Micro Devices, Inc.



topics

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



[Public]

porting to AMD



1 2	<pre>#include <cuda.h></cuda.h></pre>
3 4	constant <i>float</i> a = 2.0 <i>f</i> ;
5	global
6	void saxpy(int n, float const* x, float* y)
7	{
8	<pre>int i = blockDim.x*blockIdx.x + threadIdx.x;</pre>
9	if (i < n)
10	y[i] += a*x[i];
11	}

simple SAXPY kernel

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

```
[Public]
```

```
adding the CPU code
```

```
#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()
14
        int n = 256;
        std::size_t size = sizeof(float)*n;
         float* d_x;
         float* d_y;
         cudaMalloc(&d_x, size);
                                                                         allocate arrays in device memory
         cudaMalloc(&d_y, size);
         int num_blocks = 2;
                                                                         set up the grid
24
         int num_threads = 128;
                                                                         launch the kernel
         saxpy<<<num_blocks, num_threads>>>(n, d_x, d_y); <-</pre>
         cudaDeviceSynchronize();
```

```
[Public]
```

```
adding host↔device copies
    #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);
                                                                      allocate arrays in host memory
        float* h_y = (float*)malloc(size);
        float* d_x;
        float* d_v;
        cudaMalloc(&d_x, size);
        cudaMalloc(&d_y, size);
24
        cudaMemcpy(d_x, h_x, size, cudaMemcpyHostToDevice);
                                                                      copy content to device memory
        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);
                                                                      copy results back to host memory
        cudaMemcpy(h_y, d_y, size, cudaMemcpyDeviceToHost);
        cudaDeviceSynchronize();
```

```
[Public]
```

```
\_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()
   std::size_t size = sizeof(float)*n;
    float* h_x = (float*)malloc(size);
    float* h_y = (float*)malloc(size);
    float* d_x;
    float* d_v;
   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);
                                                                   free arrays in device memory
    cudaFree(d_y);
    free(h_x);
                                                                   free arrays in host memory
    free(h_y);
```

adding memory cleanup

```
[Public]
```

```
#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];
                                                                       simple error checking macro
#define CHECK(call) assert(call == cudaSuccess) 
int main()
    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);
```

adding error checks

simple CUDA code

[Public]

```
#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()
    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

```
#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()
    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

```
#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];
int main()
    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

[Public]

```
#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()
    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

```
#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()
    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

is basically identical to CUDA

- o blockDim.[xyz]
- o blockIdx.[xyz]
- o threadIdx.[xyz]
- o __global___
- o __device__
- o ___shared___
- o etc.



HIP runtime API

device management

o hipSetDevice(), hipGetDevice(), hipGetDeviceProperties()

memory management
o hipMalloc(), hipFree(), hipMemcpy()

stream management

```
o hipStreamCreate(), hipStreamFree(), hipStreamSynchronize()
```

events

hipEventCreate(), hipEventDestroy(), hipEventRecord()

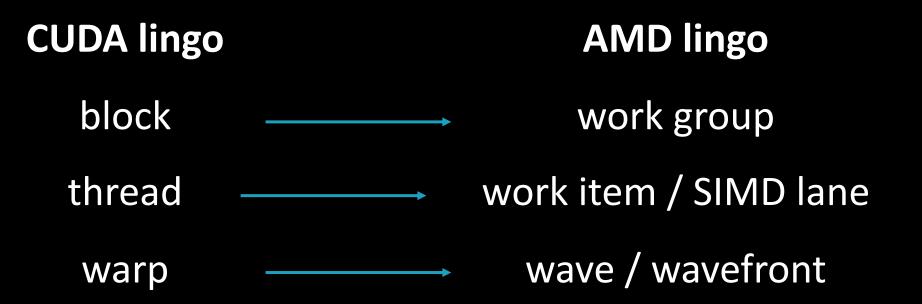
error handling

```
o hipGetLastError(), hipGetErrorString()
```

etc.



AMD lingo





hipify tools

hipify-clang

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

hipify-perl

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

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

```
[Public]
```

```
#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()
    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
- \circ no replacements done
- prints basic statistics and the number of replacements

```
[Public]
```

```
#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()
    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];</pre>
```

#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

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

```
AMD
together we advance_
```

```
[Public]
```

```
#include <cassert>
#include "cuda2hip.h" <
__constant__ float a = 2.0f;
__global__
    int i = blockDim.x*blockIdx.x + threadIdx.x;
        y[i] += a*x[i];
#define CHECK(call) assert(call == cudaSuccess)
int main()
     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);
```

	#define	cudaSuccess	hipSuccess
2	#define	cudaMalloc	hipMalloc
3	#define	cudaMemcpy	hipMemcpy
	#define	cudaMemcpyHostToDevice	hipMemcpyHostToDevice
5	#define	cudaMemcpyDeviceToHost	hipMemcpyDeviceToHost
6	#define	cudaDeviceSynchronize	hipDeviceSynchronize
7	#define	cudaFree	hipFree
8			

alternatively

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

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

```
[Public]
```

```
#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);
   #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

- $\circ~$ 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
- o multidimensional arrays
- etc., etc., etc.

Kokkos

- $\,\circ\,\,$ originates from Sandia National Laboratory
- o https://kokkos.org/
- o <u>https://github.com/kokkos</u>

RAJA

- $\circ~$ originates from Lawrence Livermore
- o <u>https://raja.readthedocs.io</u>
- o <u>https://github.com/LLNL/RAJA</u>

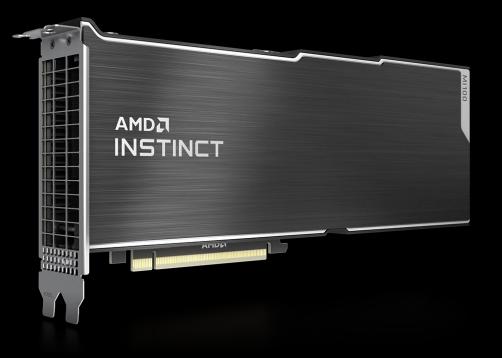
[Public]

hardware intro



AMD GPUs





Radeon[™] Graphics Cards RDNA architecture E.g.:

- RX 6000 Series
- \circ RX 7000 Series

AMD Instinct[™] Accelerators
CDNA architecture
E.g.:
MI100
MI200
MI300

AMD in HPC





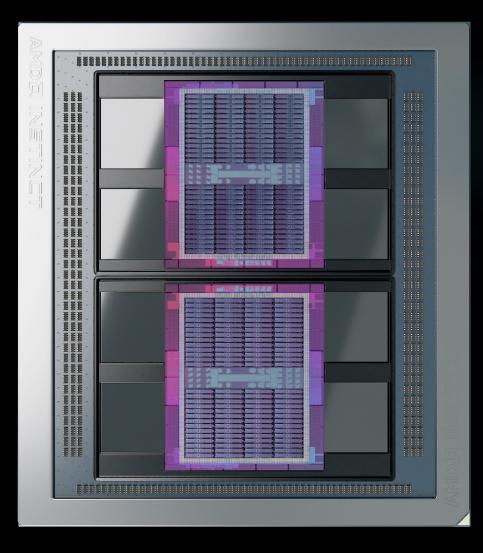
Frontier@ORNL

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

LUMI@CSC

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

AMD Instinct[™] MI200

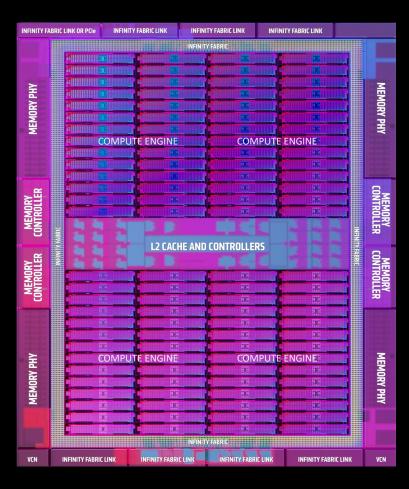


AMD INSTINCT^M MI250X WORLD'S MOST ADVANCED DATA CENTER ACCELERATOR

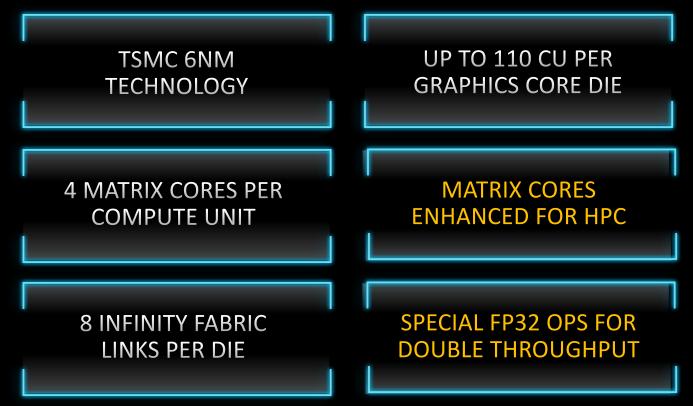


https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

AMD Instinct[™] MI200



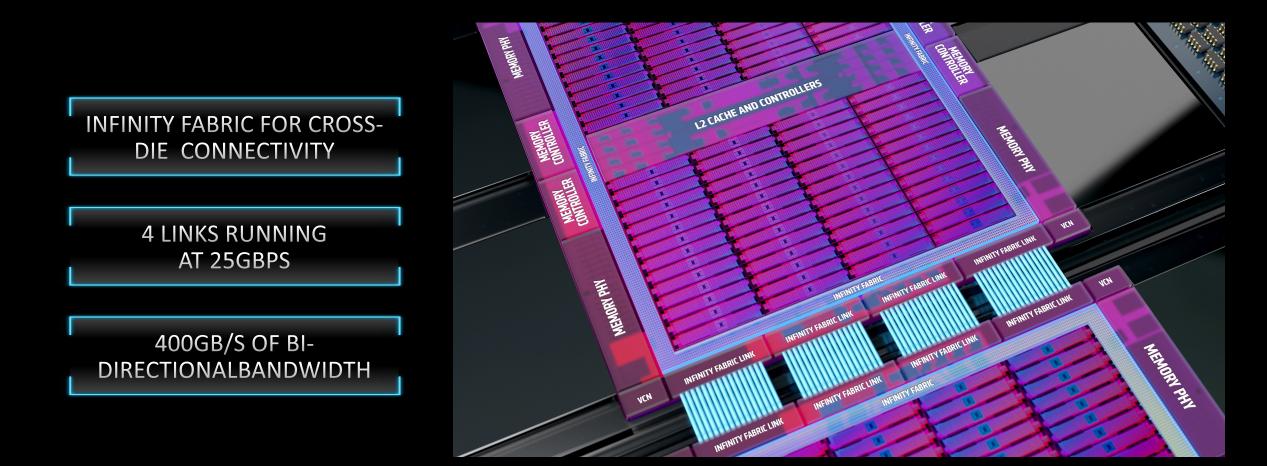
2ND GENERATION CDNA ARCHITECTURE TAILORED-BUILT FOR HPC & AI





MULTI-CHIP DESIGN

TWO GPU DIES IN PACKAGE TO MAXIMIZE COMPUTE & DATA THROUGHPUT







2nd GENERATION MATRIX CORES

OPTIMIZED COMPUTE UNITS FOR SCIENTIFIC COMPUTING



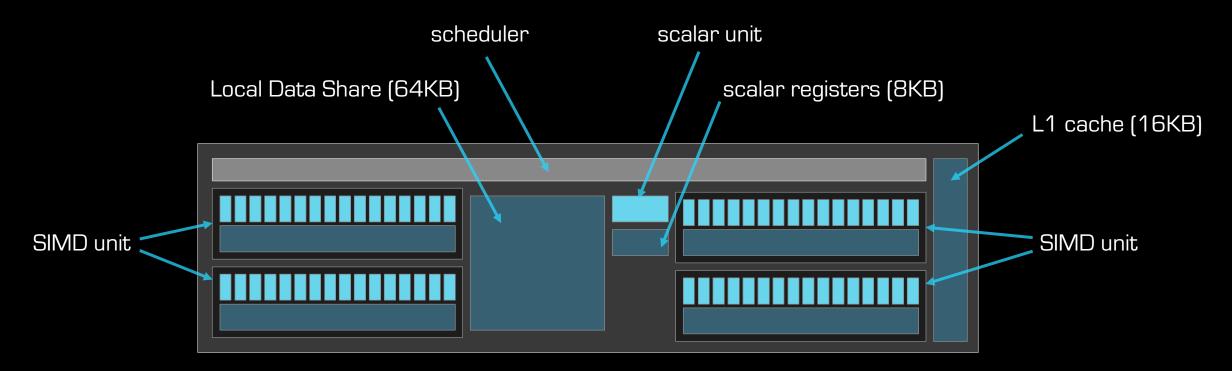
DOUBLE PRECISON (FP64) MATRIX CORE THROUGHPUT REPRESENTATION

AMD Instinct[™] MI200

		Infinity Fabric or PC le	Infinity Fabric Infinity Fabric Link	Infi InfinityFabricLink Infin	Infinity Fabric Link Infin	Infinity	
Memory Phy					88888888888888	Memory Phy	НВМ2
Controller	Memory	Compute Engine	and Controllers	L2 Cache Compute Engine CU CU	Comp CU	Memory Controller	
	Memory Controller	Compute Engine) CU CU CU CU CU	Comi CU CU	Memory Controller	
	Memory Phy				2222222222	Memory Phy	НВМ2
	VCN	bric Link Infinity Fabric Link	ink Infinity Fabric Link Infinity Fabric	Infinity Fabric Link	Infinity Fabric Link	VCN	
	S Memo	bric Link	cu cu unfinity Fabric nk Infinity Fabric Link	Lutinity Fabric Link	CU CU CU Lufinity Fabric Link	Ç Memor	НВМ
	ry Phy		8888888	8888888	8 8 8 8 8 8	y Phy	2
	Memory Controller	Compute Engine	L2 Cathe and Controllers	L2 Cache Compute Engline CU	CU Com	Memory Controller	
	Memory Controller	Compute Engine) (C) (Comp Comp	Memory Controller	
	Memory Phy					Memory Phy	НВМ2
	bric Link	Infinity Fabric Link Infinity Fabric Link	Infinity Fabric Link Infinity Fabric	Infinity	Infinity Fabric or PCIe	П	



MI200 compute unit



each SIMD unit

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



optimization fundamentals

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

ISA is public

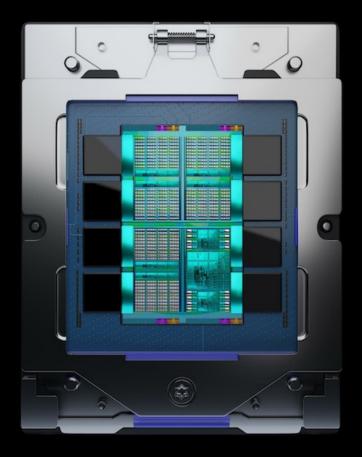
"AMD Instinct MI200" Instruction Set Architecture Reference Guide

18-November-2021

- 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

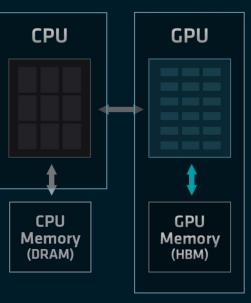
Breakthrough architecture to power the exascale AI era



UNIFIED MEMORY APU ARCHITECTURE BENEFITS

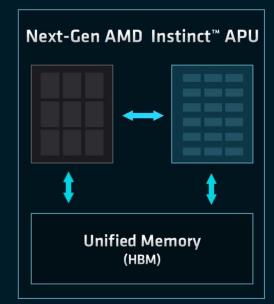
AMD CDNA[™] 2 Coherent Memory Architecture

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



AMD CDNA[™] 3 Unified Memory APU Architecture

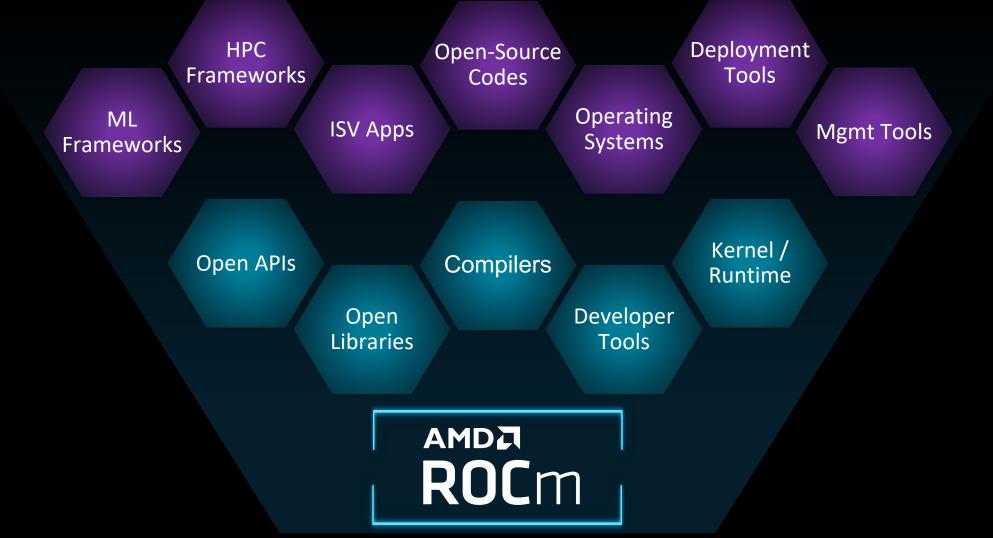
- Eliminates Redundant Memory Copies
- High-Efficiency 4th Gen AMD Infinity Architecture
- Low TCO with Unified Memory APU Package



[Public]

ROCm overview

AMD ROCm[™] Open Software Platform for GPU Compute





AMD ROCm[™] Open Software Platform for GPU Compute

	Optimized Training/Inference Models & Applications								
Benchmarks & App Support	MLPERF	HPL/HI	HPL/HPCG Life So		Geo Science	e	Physics		
Operating Systems Support	RHEL		CentOS		.ES	U	buntu		
Cluster Deployment	Singularit	:y K	Kubernetes®		ker®	SLURM			
Framework Support	Kokkos	s/RAJA	РуТо			TensorFlow			
Libraries	BLAS SOLVER	RAND			X MIVisi MIOp		PRIM RCCL		
Programming Models	OpenM	P [®] API	API Oper			ΗΙΡ ΑΡΙ			
Development Toolchain	Compiler	Profiler	Tracer	Debugge	er hipi	fy	GPUFort		
Drivers & Runtime		GPU	Device Driver	s and ROCm™	Run-Time				
Deployment Tools	ROCm Valid	lation Suite	Suite ROCm Data		bl	ROCm SMI			

libraries

rocBLAS / hipBLAS

• basic operations on dense matrices

rocSOLVER

 \circ dense linear algebra solvers

rocSPARSE / hipSPARSE

 \circ basic operations on sparse matrices

rocALUTION

 \circ sparse linear algebra solvers

rocFFT / hipFFT

• Fast Fourier transforms

rocRAND / hipRAND

 \circ random number generation

rocPRIM / hipCUB / rocThrust

 \circ scan, sort, reduction, etc.

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

https://github.com/ROCmSoftwarePlatform/rocSOLVER

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

https://github.com/ROCmSoftwarePlatform/rocALUTION

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

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

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



also open source

the compiler

o <u>https://github.com/ROCmSoftwarePlatform/llvm-project</u>

the runtime

<u>https://github.com/RadeonOpenCompute/ROCR-Runtime</u>

the debugger

o <u>https://github.com/ROCm-Developer-Tools/ROCgdb</u>

the profiler

o <u>https://github.com/ROCm-Developer-Tools/rocprofiler</u>

the HPL benchmark

o <u>https://github.com/ROCmSoftwarePlatform/rocHPL</u>

the HPCG benchmark

o <u>https://github.com/ROCmSoftwarePlatform/rocHPCG</u>

etc.

basic debugging tools





rocgdb

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

cause a segfault

```
#include <hip/hip_runtime.h>
1
     \_constant_ float a = 1.0f;
4
     __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];
10
11
12
13
     int main()
14
         int n = 256;
         std::size_t size = sizeof(float)*n;
17
         float* d_x;
         float* d_y;
19
         // hipMalloc(&d_x, size);
         // hipMalloc(&d_y, size);
21
         int num_groups = 2;
         int group_size = 128;
24
         saxpy < < <num_groups, group_size>>> (n, d_x, 1, d_y, 1);
         hipDeviceSynchronize();
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

2 3cor 4 5glc 6 void 7 { 8 i 9 i	<pre>lude <hip hip_runtime.h=""> nstant float a = 1.0f; obal saxpy(int n, float const* x, int inc int i = blockDim.x*blockIdx.x + threa if (i < n) </hip></pre>		 Need be, set the target gfx906 - MI50, MI60, Radeon™ 7 gfx908 - MI100 fgx90a - MI200
14 { 15 ii 16 s 17 1 18 ii 19 ii 20 / 21 / 22 ii 23 ii 24 ii 25 s	<pre>y[i] += a*x[i]; 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, hipDeviceSynchronize();</num_groups,></pre>	saxpy\$ hipccoffload-	arch=gfx906 –o saxpy saxpy.hip.cpp

run

1	<pre>#include <hip hip_runtime.h=""></hip></pre>		
2			
3	constant <i>float</i> a = 1.0 <i>f</i> ;		
4			
5	global		
6	<pre>void saxpy(int n, float const* x, int ind</pre>	cx, float∗ y, int incy)	
7	{		
8	<pre>int i = blockDim.x*blockIdx.x + threa</pre>	adIdx.x;	
9	if (i < n)		
10	y[i] += a*x[i];	carpy thipggoffload	arch=gfx906 –o saxpy saxpy.hip.cpp
11	}	saxpy\$./saxpy	arch-grx900 -0 Saxpy Saxpy.htp.cpp
12		Saxpyş ./Saxpy	
13	<pre>int main()</pre>		
14			
15	int n = 256;		
16 17	<pre>std::size_t size = sizeof(float)*n;</pre>		
18	float* d_x;		
19	float* d_y;		
20	// hipMalloc(&d_x, size);		
20	// hipMalloc(&d_y, size);		
22	// hipharioe(ad_y, 512c),		
23	<pre>int num_groups = 2;</pre>		
24	int group_size = 128;		
25	<pre>saxpy<<<<num_groups, group_size="">>>(n,</num_groups,></pre>		
26	hipDeviceSynchronize();		
27	}		
28	Ĩ.		
	-		

crash

```
#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];
10
                                              saxpy$ hipcc --offload-arch=gfx906 -o saxpy saxpy.hip.cpp
11
                                              saxpy$ ./saxpy
12
                                              Memory access fault by GPU node-4 (Agent handle: 0x19dee10) on address (nil). Reason: Page not
13
     int main()
                                               present or supervisor privilege.
14
                                              Aborted (core dumped)
         int n = 256;
                                              saxpy$
         std::size_t size = sizeof(float)*n;
17
         float* d_x;
19
         float* d_y;
         // hipMalloc(&d_x, size);
         // hipMalloc(&d_y, size);
         int num_groups = 2;
         int group_size = 128;
24
         saxpy<<<num_groups, group_size>>>(n,
         hipDeviceSynchronize();
28
```

run with rocgdb

```
#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];
                                              saxpy$ rocgdb saxpy
11
12
13
     int main()
14
         int n = 256;
         std::size_t size = sizeof(float)*n;
17
         float* d_x;
19
         float* d_y;
         // hipMalloc(&d_x, size);
         // hipMalloc(&d_y, size);
         int num_groups = 2;
         int group_size = 128;
24
         saxpy<<<num_groups, group_size>>>(n,
         hipDeviceSynchronize();
28
```

get some details

1 2 3 4 5 6 7 8 9 10	<pre>#include <hip hip_runtime.h="">constant float a = 1.0f;global void saxpy(int n, float const* x, int inc { int i = blockDim.x*blockIdx.x + threa if (i < n) y[i] += a*x[i];</hip></pre>		Reports segmentation fault in the saxpy kernel.
10 11 12 13 14 15 16 17 18 20 21 22 23 24 25 26 27 28	<pre>} 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); }</pre>	[New Thread 0x7ffff4d367 Warning: precise memory location may not be accu Thread 3 "saxpy" receive [Switching to thread 3,	<pre>libthread_db enabled] library "/lib/x86_64-linux-gnu/libthread_db.so.1". 00 (LWP 67093)] violation signal reporting is not enabled, reported rate. See "show amdgpu precise-memory". d signal SIGSEGV, Segmentation fault. lane 0 (AMDGPU Lane 1:2:1:1/0 (0,0,0)[0,0,0])] xpy(int, float const*, int, float*, int) () from file:///mnt/shared/co</pre>

compile with -ggdb

1 2	<pre>#include <hip hip_runtime.h=""></hip></pre>			
3	constant float a = 1.0f;			
4 5 6 7	global void saxpy(int n, float const* x, int inc	x, float∗y, int incy)		
7 8 9 10	i int i = blockDim.x*blockIdx.x + threa if (i < n) y[i] += a*x[i];	odIdx.x;		
10 11 12	} }	saxpy\$ hipcc_ggdbo	ffload-arch=gfx906 –o saxpy saxpy.hip.cpp	
13	int main()			
14 15 16	{ int n = 256; std::size_t size = sizeof(float)*n;			
17 18 19	float* d_x; float* d_y;			
20 21 22	// hipMalloc(&d_x, size); // hipMalloc(&d_y, size);			
23 24	<pre>int num_groups = 2; int group_size = 128;</pre>			
25 26	<pre>saxpy<<<num_groups, group_size="">>>(n, hipDeviceSynchronize();</num_groups,></pre>			
27 28	} 			

get more details

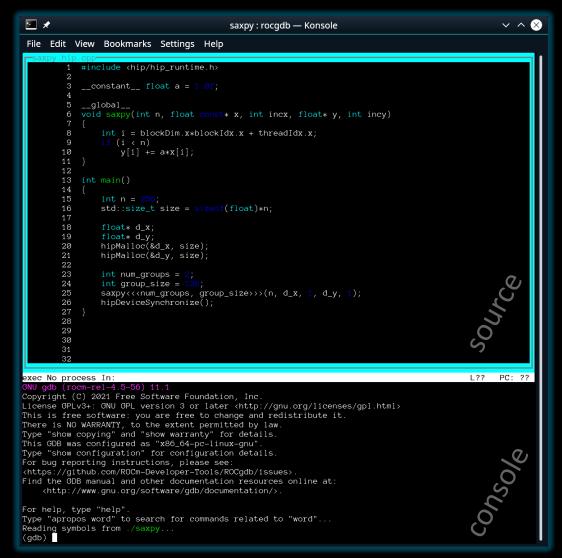
1	<pre>#include <hip hip_runtime.h=""></hip></pre>		
2 3 4 5 6 7 8 9	<pre>constant float a = 1.0f; global void saxpy(int n, float const* x, int inc { int i = blockDim.x*blockIdx.x + threa if (i < n)</pre>		more details what kernel what file:line
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28	<pre>int main() { int n = 256;</pre>	[New Thread 0x7ffff4d367 Warning: precise memory location may not be accu Thread 3 "saxpy" received [Switching to thread 3, 0x00007fffe a01094 in sat	<pre>libthread_db enabled] library "/lib/x86_64-linux-gnu/libthread_db.so.1". 00 (LWP 67682)] violation signal reporting is not enabled, reported rate. See "show amdgpu precise-memory". d signal SIGSEGV, Segmentation fault. lane 0 (AMDGPU Lane 1:2:1:1/0 (0,0,0)[0,0,0])] xpy (n=<optimized out="">, x=<optimized out="">, incx=<optimized out="">, y=<optimized out="">, at saxpy.hip.cpp:10</optimized></optimized></optimized></optimized></pre>
	But where's my stack trace?		

list threads

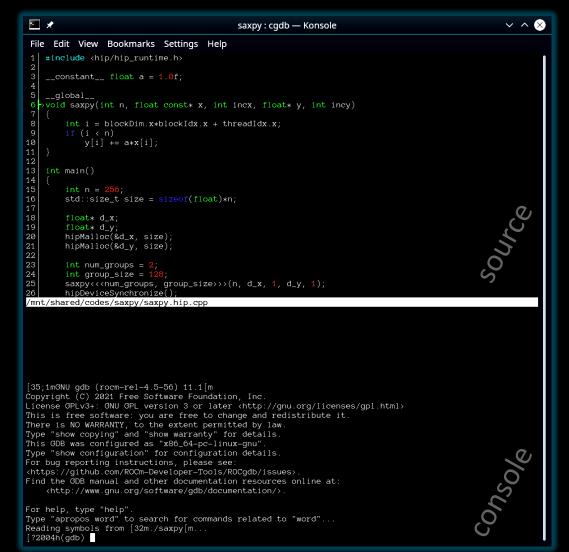
```
#include <hip/hip_runtime.h>
1
     \_constant_ float a = 1.0f;
     __global_
     void saxpy(int n, float const* x, int incx, float* y, int incy)
         int i = blockDim.x*blockIdx.x + threadIdx.x;
         if (i < n)
             y[i] += a*x[i];
10
                                              (gdb) i th
11
                                                     Target Id
                                                                                                Frame
                                                Ιd
12
                                                     Thread 0x7ffff7fb6880 (LWP 67674) "saxpy" 0x00007ffff57f5102 in rocr::core::InterruptSigr
                                                1
13
     int main()
                                                 from /opt/rocm-4.5.0/hip/lib/../../lib/libhsa-runtime64.so.1
14
                                                     Thread 0x7ffff4d36700 (LWP 67682) "saxpy" 0x00007ffff5f6d317 in ioctl () at ../sysdeps/ur
                                                2
         int n = 256;
                                              * 3
                                                     AMDGPU Wave 1:2:1:1 (0,0,0)/0 "saxpy"
                                                                                                0x00007fffe8a01094 in saxpy (n=<optimized out>,
         std::size_t size = sizeof(float)*n;
                                                     AMDGPU Wave 1:2:1:2 (0,0,0)/1 "saxpy"
17
                                                4
                                                                                                0x00007fffe8a01094 in saxpy (n=<optimized out>,
                                                5
                                                     AMDGPU Wave 1:2:1:3 (1,0,0)/0 "saxpy"
                                                                                                0x00007fffe8a01094 in saxpy (n=<optimized out>,
         float* d_x;
                                                     AMDGPU Wave 1:2:1:4 (1,0,0)/1 "saxpy"
                                                                                                0x00007fffe8a01094 in saxpy (n=<optimized out>,
                                                6
19
         float* d_y;
                                              (gdb)
         // hipMalloc(&d_x, size);
         // hipMalloc(&d_y, size);
21
         int num_groups = 2;
24
         int group_size = 128;
         saxpy<<<num_groups, group_size>>>(n,
         hipDeviceSynchronize();
28
```

"GUIs"

rocgdb -tui saxpy



cgdb -d rocgdb saxpy



```
[Public]
```

AMD_LOG_LEVEL=3

M 🗶	saxpy : bash — Konsole 🛛 🗸 🔨	\otimes
File Edit View Bookmarks	Settings Help	
	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 gp	bu
: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_ = 0xfefffc00	
: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_ = 0xfefff800	
: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>)</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	Β, Ι
: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, strea	am
: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: <n< td=""><td></td></n<>	
:3:rocdevice.cpp	:2623: 714826168023 us: number of allocated hardware queues with low priority: 0, with normal priority: 0, with high pr	ci
: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 :	
	: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	
	:472 : 714826441344 us: 123767: [7f5b72543880] hipDeviceSynchronize: Returned hipSuccess :	
jakurzak@jakurzak-MS-7B09:/m	mnt/shared/codes/saxpy\$	

J

```
[Public]
```

```
#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)

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

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

printf from a kernel

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

basic performance tools





performance tools

uProf

- AMD's classic CPU profiler
- $\circ~$ now also with GPU support
- o <u>https://www.amd.com/en/developer/uprof.html</u>

rocprof

- AMD's standard GPU profiling / tracing tool
- $\circ~$ part of the ROCm distribution

omniperf / omnitrace

- $\circ~$ cutting-edge research tools
- o <u>https://github.com/AMDResearch/omniperf</u>
- o <u>https://github.com/AMDResearch/omnitrace</u>

rocprof --stats ./prog ...

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

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

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

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

	А	В	С	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

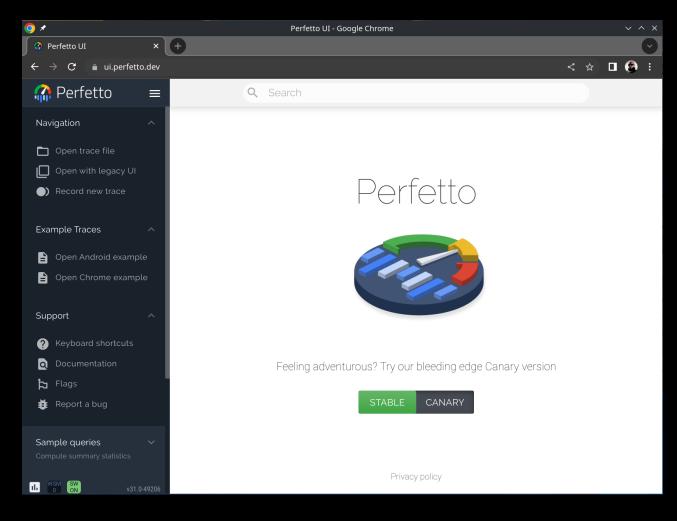
```
rocprof --stats --hip-trace ./prog ...
```

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

	A	В	C	D	E
1	Name	Calls	TotalDurationNs	<u>AverageNs</u>	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 --stats --hip-trace ./prog ...

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



https://ui.perfetto.dev/

1464463.8 s + 0 s +218.8 ms +718.8 ms +1.2 s +1.7 s +2.2 s +2.7 s +3.2 s +3.7 s +4.7 s		1 I I I 0 s	437.7 ms	875.5 ms	1.3 s	1.8 s	2.2 s	2.6 s 3	.1 s 3.5	s 3.9 s	
CPU HIP API 2 Image: CPU HIP A		1									
	^		+218.8 ms	+718.8 ms	+1.2 s	+1.7 s	+2.2 s	+2.7 s	+3.2 s	+3.7 s	+4.2
✓ GPU410	✓ CPU HIP API 2										
	✔ GPU4 10										

overview of CPU / GPU activity



	0 s		875.5 ms	1.3 s	1.8 s	2.2 s	2.6 s	3.1 s 3.		S
	Γ.									
	S	+218.8 ms	+718.8 ms	+1.2 s	+1.7 s	+2.2 s	+2.7 s	+3.2 s	+3.7 s	+4.2
CPU HIP API 2										
Thread 3631968										
Thread 3631972	hip									
▲ GPU410										
Thread 1										
	 CPU HIP API 2 Thread 3631968 Thread 3631972 GPU4 10 	0 s 1464463.8 s + 0 s ★ CPU HIP API 2 Thread 3631968 I Thread 3631972 hip ↑ GPU4 10	1464463.8 s + 0 s +218.8 ms * 0 s +218.8 ms * 0 s 10 * CPU HIP API 2 - Thread 3631968 10 - Thread 3631972 hip - * GPU4 10 - -	0 s 437.7 ms 875.5 ms 1464463.8 s + 0 s	0 s 437.7 ms 875.5 ms 1.3 s 1464463.8 s + 0 s	0s 437.7 ms 875.5 ms 1.3 s 1.8 s 1464463.8 s+ 0 s +218.8 ms +718.8 ms +1.2 s +1.7 s * CPU HIP API 2 - - - - - Thread 3631968 Image: Comparison of the second of t	0s 437.7 ms 875.5 ms 1.3 s 1.8 s 2.2 s 1464463.8 s + 0 s + <td< th=""><th>0 s 437.7 ms 875.5 ms 1.3 s 1.8 s 2.2 s 2.6 s 1464463.8 s+ 0 s 1 118 ms 11.2 s 11.7 s 12.2 s 2.6 s 1464463.8 s+ 0 s 1 118 ms 11.2 s 11.7 s 12.2 s 12.7 s 1464463.8 s+ 0 s 1 118 ms 11.2 s 11.7 s 12.2 s 12.7 s 1464463.8 s+ 0 s 1 118 ms 11.2 s 11.7 s 12.2 s 12.7 s 1464463.8 s+ 0 s 1 118 ms 11.2 s 11.7 s 12.2 s 12.7 s 1464463.8 s+ 0 s 1</th><th>0s 437.7 ms 875.5 ms 1.3 s 1.8 s 2.2 s 2.6 s 3.1 s</th><th>0 s 437.7 ms 875.5 ms 1.3 s 1.8 s 2.2 s 2.6 s 3.1 s 3.5 s 3.9 1464463.8 s + 0 s +218.8 ms +718.8 ms +12.5 +1.7 s +22.s +27.s +32.s +37.s * CPU HIP API 2 - <t< th=""></t<></th></td<>	0 s 437.7 ms 875.5 ms 1.3 s 1.8 s 2.2 s 2.6 s 1464463.8 s+ 0 s 1 118 ms 11.2 s 11.7 s 12.2 s 2.6 s 1464463.8 s+ 0 s 1 118 ms 11.2 s 11.7 s 12.2 s 12.7 s 1464463.8 s+ 0 s 1 118 ms 11.2 s 11.7 s 12.2 s 12.7 s 1464463.8 s+ 0 s 1 118 ms 11.2 s 11.7 s 12.2 s 12.7 s 1464463.8 s+ 0 s 1 118 ms 11.2 s 11.7 s 12.2 s 12.7 s 1464463.8 s+ 0 s 1	0s 437.7 ms 875.5 ms 1.3 s 1.8 s 2.2 s 2.6 s 3.1 s	0 s 437.7 ms 875.5 ms 1.3 s 1.8 s 2.2 s 2.6 s 3.1 s 3.5 s 3.9 1464463.8 s + 0 s +218.8 ms +718.8 ms +12.5 +1.7 s +22.s +27.s +32.s +37.s * CPU HIP API 2 - <t< th=""></t<>



WASD

scroll

zoom in

zoom out

	0 s 437.7 ms	875.5 ms	1.3 s 1.8 s	2.2 s	2.6 s 3.1	ls 3.5 s	3.9 s
1464463.8 s + 4.4 s	+767.4 us	+1.8 ms	+2.8 ms	+3.8 ms	+4.8 ms	+5.8 ms	+6.8 ms
▲ 4.4 S	1707.4 65	110 110	12.0 110	10.0 110	14.0 110	10.0 110	10.0 mo
▲ CPU HIP API 2							
Thread 3631968							
Thread 3631972	hipDeviceSynchronize	hipLaunc	hipDeviceSynchronize	e hipDevi	ceSynchronize	hipDeviceS	Synchronize
∧ GPU410							
Thread 1	copy_kernel		init_kernel	in	iit_kernel	сору.	_kernel

437.7 ms 875.5 ms 1.3 s 1.8 s 2.2 s 2.6 s 3.1 s 3.5 s 3.9 s 0 s 1464463.8 s + 4.4 s +767.4 us +1.8 ms +2.8 ms +3.8 ms +4.8 ms +5.8 ms +6.8 ms × ▲ CPU HIP API 2 Thread 3631968 hipDeviceSynchronize hipDeviceSynchronize hipDeviceSynchronize hipDeviceSynchronize Thread 3631972 ∧ GPU410 Thread 1 copy_kernel init_kernel init_kernel copy_kernel Current Selection Flow Events \mathbf{v} Slice Details Preceding flows Name init_kernel Slice ↗ hipLaunchKernel Category null Delay 8us Start time 4s 365ms 323us NULL (CPU HIP API 2) Thread Duration 1ms 751us Arguments Thread duration 0s (0.00%) args 1 Thread BeginNs -1464468146522807 GPU4 10 Process Data 👻 NULL Slice ID 8347 DurationNs -1751999 EndNs 👻 1464468148274806 init_kernel Name 👻

pid 👻

3631968

click a kernel to get more details

> AMD together we advance_



omnitrace

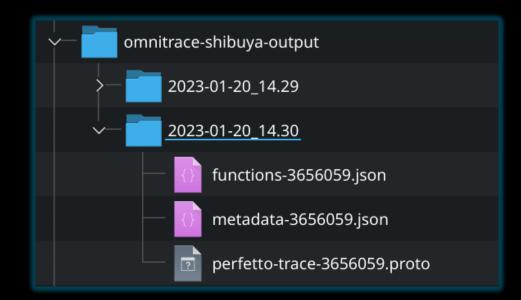
- \circ very powerful
- \circ in research stage
- \circ based on binary instrumentation
 - \circ dynamic instrumentation
 - \circ binary rewriting
- $\circ~$ can trace any activity
 - \circ app routines
 - \circ library routines
 - \circ runtime routines
- $\circ~$ can trace CPU and GPU activity
- OpenMP[®] 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



<u>https://ui.perfetto.dev/</u>

omnitrace 952.6 ms 0 s 476.3 ms 1.4 s 1.9 s 2.4 s 2.9 s 3.3 s 3.8 s 4.3 s 1467397.3 s + 0 s +170.1 ms +670.1 ms +1.2 s +1.7 s +2.2 s +2.7 s +3.2 s +3.7 s +4.2 s +. ž Clock Snapshots metric ▲ ./shibuya 3638715 main app thread pthread_join shibuya 3638715 HIP thread Thread 2 3638725 CPU Context Switches (S) 5 K \sim 0.25 CPU Kernel Time (S) \sim 0.25 K CPU Memory Usage (S) \sim CPU Page Faults (S) \sim 25 K **CPU** stats 0.25 K CPU Peak Memory (S) \sim CPU User Time (S) \sim 7.5 10 K CPU Virtual Memory Usage (S) \sim ./shibuya 3638715 **GPU** activity HIP Activity Device 4, Queue 0

AMD together we advance_

omnitrace

WASD

scroll

zoom in

zoom out

		0 s	476.3	ms 952.6	ms 1.4	s 1.9	s 2.4	1 s 2	.9 s 3	3.3 s	3.8 s	4.3 s
1467397.3 s +	4.7 s		+259.5 us	+759.5 us	+1.3 ms	+1.8 ms	+2.3 ms	+2.8 ms	+3.3 ms	+3.8 ms	+4.3 ms	+4.8 ms
Clock Snapshots metric									ĺ			
∧ ./shibuya 3638715												
shibuya 3638715							pthread_joi	n				
Thread 2 3638725			hipDevice	Synchronize	hi	ipLaunchKerne		hipDeviceSy	nchronize		hipDeviceSync	chronize
CPU Context Switches (S)	/	5 K										
CPU Kernel Time (S)	/	0.25										
CPU Memory Usage (S)	/	0.25 K										
CPU Page Faults (S)	/	25 K										
CPU Peak Memory (S)	/	0.25 K										
CPU User Time (S)	/	7.5										
CPU Virtual Memory Usage (S)	/	10 K										
./shibuya 3638715												
HIP Activity Device 4, Queue 0		Devi	ceStream <dou< td=""><td>ble>::copy_ke</td><td>rnel(d</td><td></td><td>Device</td><td>Array<double></double></td><td>::init_kernel(do</td><td>oubl (</td><td>DeviceArray<do< td=""><td>ouble>::</td></do<></td></dou<>	ble>::copy_ke	rnel(d		Device	Array <double></double>	::init_kernel(do	oubl (DeviceArray <do< td=""><td>ouble>::</td></do<>	ouble>::

[Public]

omnitrace

	0 s		549.5 ms 824.2 m		1.4 s 1.6			2.5 s		
	05	2/4./ ms 5	149.5 ms 624.2 m	5 1.15	1.4 5 1.0	2 1.3	5 2.25	12.58		
1467887.3 s + 296.5	5 ms +42.9 us	+542.9 us +1 ms	+1.5 ms +2	ms +2.5 ms	+3 ms +3.5 ms	+4 ms	+4.5 ms +5 ms	+5.5 ms +		
*										
Clock Snapshots metric										
∧ shibuya 3640498										
					nain				main ann thr	and
- hihung 06 40 400			main app thread							
shibuya 3640498					ead_join					
					ouble>::test					
Thread 3 3640515	hi	hipDeviceSyr hipDeviceSyr		·	DeviceSynchronize DeviceSynchronize		hipDeviceSyn hipDeviceSyn		HIP thread	
Thread 4 3640516			No		louble>::run stStream <double>::cop</double>)V				
Thread 5 3640517				Stream<0	louble>::run stStream <double>::cop</double>					
Thread 6 3640518				Stream<0	louble>::run stStream <double>::cop</double>				app "worker"	thread
Thread 7 3640519			No		louble>::run stStream <double>::cop</double>	by				
CPU Context Switches (S)	5 K									
CPU Kernel Time (S)	7.5									
CPU Memory Usage (S)	10 K									
CPU Page Faults (S)	5 M									
CPU Peak Memory (S)	10 K								CPU stats	
CPU User Time (S)	5									
CPU Virtual Memory Usage (S)	25 K									
/media/jakub/shared/container- share/ShibuvaStream/src/shibuva										
HIP Activity Device 4, Queue 0	D	eviceArray <double>::</double>	init_kernel(doubl	DeviceArray	<double>::init_kernel(de</double>	oubl D	DeviceStream <doub< td=""><td>le>::copy_kern</td><td>GPU activity</td><td>AMD</td></doub<>	le>::copy_kern	GPU activity	AMD

together we advance_

68

AMDA

ROCm

<u>https://docs.amd.com/</u> https://rocmdocs.amd.com/

CDNA[™] 2

<u>https://www.amd.com/en/technologies/cdna2</u> 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.

© 2023 Advanced Micro Devices, Inc. All rights reserved.

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

DISCLAIMERS AND ATTRIBUTIONS

Docker and the Docker logo are trademarks or registered trademarks of Docker, Inc. Intel is a trademark of Intel Corporation or its subsidiaries. Kubernetes is a registered trademark of The Linux Foundation. OpenCL is a trademark of Apple Inc. used by permission by Khronos Group, Inc. The OpenMP name and the OpenMP logo are registered trademarks of the OpenMP Architecture Review Board. Perl is a trademark of Perl Foundation.

#