

# AMD GPU Documentation, Benchmarking, and Roadmap

Justin Chang, Suyash Tandon, Bill Brantley

# Who are we?

- Part of the Data Center GPU Software Solutions Group
- Role comprises generally of three tasks:
  1. Application porting and optimization – working with code owners
  2. Provide feedback to ROCm developers and hardware architects
  3. Train people to leverage our data center GPUs

## Questions often asked

1. This new ROCm release broke/regressed my application, what should I do?
2. Which tools should I use to profile my code?
3. What optimization tips and tricks do you have for performance improvements?
4. Where can I find documentation?

# Official ROCm documentation

<https://rocm.docs.amd.com>

Comprehensive documentation tailored to each ROCm version:

ROCm Documentation is transitioning to this site. For the legacy documentation, please visit [docs.amd.com](https://docs.amd.com). For more information or to provide feedback about this documentation transition, please see [our announcement](#).

AMD | ROCm™ Platform

GitHub Community **AMD Lab Notes** Infinity Hub Support Feedback

ROCm Documentation

What is ROCm?

Deploy ROCm

Linux Quick Start

Linux Overview

Docker

Release Info

Release Notes

GPU and OS Support (Linux)

Known Issues

Compatibility

Licensing Terms

APIs and Reference

All Reference Material

Compilers and Development Tools

## AMD ROCm™ Platform - Powering Your GPU Computational Needs

Applies to Linux 2023-05-24 4 min read time

What is ROCm? Deploy ROCm Release Info

APIs and Reference

- Compilers and Development Tools
- HIP
- OpenMP
- Math Libraries
- C++ Primitives Libraries
- Communication Libraries
- AI Libraries
- Computer Vision

Understand ROCm

- Compiler Disambiguation
- Using CMake
- Linux Folder Structure Reorganization
- GPU Isolation Techniques
- GPU Architecture

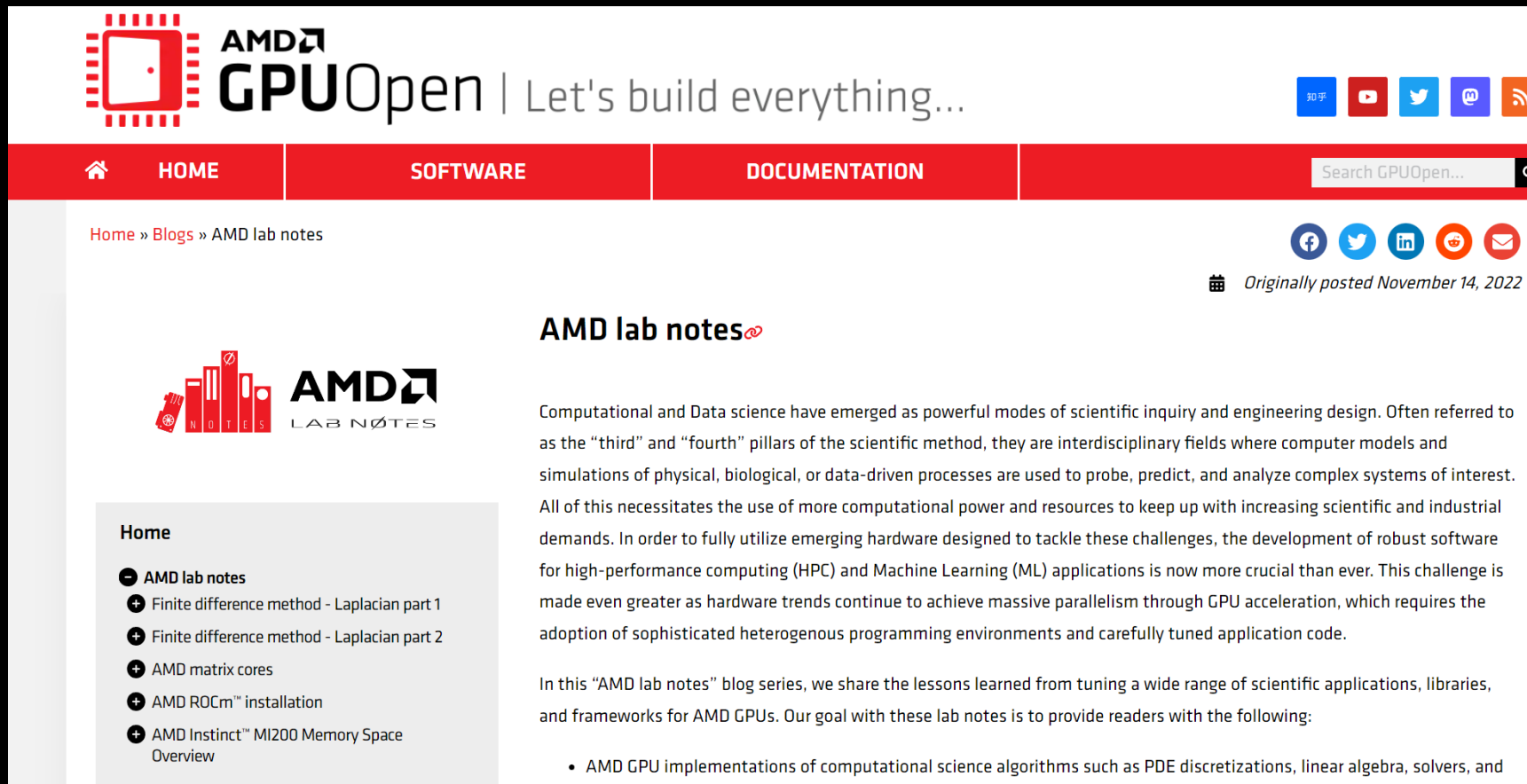
- Supported GPUs and OS
- Installing ROCm
- Compilers and tools
- HIP and math library APIs
- Link to the whitepapers
- Code examples

GitHub repository: <https://github.com/RadeonOpenCompute/ROCm>

# AMD lab notes

<https://gpuopen.com/learn/amd-lab-notes>

Technical blog post series covering:



The screenshot shows the AMD GPUOpen website. The header includes the AMD GPUOpen logo and the tagline "Let's build everything...". A navigation bar contains links for HOME, SOFTWARE, and DOCUMENTATION, along with a search bar. The main content area features a breadcrumb trail "Home » Blogs » AMD lab notes" and social media icons. The article title is "AMD lab notes" with a red lock icon. The text describes computational and data science as interdisciplinary fields. A sidebar on the left lists other articles under the "Home" section.

**AMD lab notes**

Computational and Data science have emerged as powerful modes of scientific inquiry and engineering design. Often referred to as the “third” and “fourth” pillars of the scientific method, they are interdisciplinary fields where computer models and simulations of physical, biological, or data-driven processes are used to probe, predict, and analyze complex systems of interest. All of this necessitates the use of more computational power and resources to keep up with increasing scientific and industrial demands. In order to fully utilize emerging hardware designed to tackle these challenges, the development of robust software for high-performance computing (HPC) and Machine Learning (ML) applications is now more crucial than ever. This challenge is made even greater as hardware trends continue to achieve massive parallelism through GPU acceleration, which requires the adoption of sophisticated heterogenous programming environments and carefully tuned application code.

In this “AMD lab notes” blog series, we share the lessons learned from tuning a wide range of scientific applications, libraries, and frameworks for AMD GPUs. Our goal with these lab notes is to provide readers with the following:

- AMD GPU implementations of computational science algorithms such as PDE discretizations, linear algebra, solvers, and

**Home**

- AMD lab notes
  - + Finite difference method - Laplacian part 1
  - + Finite difference method - Laplacian part 2
  - + AMD matrix cores
  - + AMD ROCm™ installation
  - + AMD Instinct™ MI200 Memory Space Overview

- Lessons learned from tuning a wide range of applications, libraries, and frameworks
- Implementations of computational science algorithms such as PDE discretizations, linear algebra, solvers, and more
- Instructions, guidance, and references on using libraries and tools from the ROCm software stack
- Best practices for porting and optimizing both HPC and AI applications
- Monthly release cadence

GitHub repository: <https://github.com/AMD/amd-lab-notes>

# Current AMD lab notes

Topic	Description
Matrix cores	Discusses how to leverage AMD GPU's matrix core processing units to accelerate GEMM computations
Finite Difference Method - Laplacian	Three-part blog series. Develops and optimizes a HIP kernel performing a finite difference operator for the Laplace operator
ROCm installation	Outlines three possible ways to quickly install specific versions of ROCm on your Ubuntu OS desktop/server
MI200 memory space overview	Provides a high-level overview of the MI200 memory architecture and all the different use cases
Profiling on AMD hardware	Covers the various profiling tools for AMD hardware and why a developer might leverage one tool over another
Register Pressure	Emphasizes the importance of understanding and controlling register usage when designing high performance GPU kernels

# Finite Difference Method – Laplacian Part 1

A three-part blog series covering a standard finite difference discretization of the Laplace operator. We begin with a simple HIP implementation:

```
27 template <typename T>
28 __global__ void laplacian_kernel(T * f, const T * u, int nx, int ny, int nz,
   T invhx2, T invhy2, T invhz2, T invhxyz2) {
29
30     int i = threadIdx.x + blockIdx.x * blockDim.x;
31     int j = threadIdx.y + blockIdx.y * blockDim.y;
32     int k = threadIdx.z + blockIdx.z * blockDim.z;
33
34     // Exit if this thread is on the boundary
35     if (i == 0 || i >= nx - 1 ||
36         j == 0 || j >= ny - 1 ||
37         k == 0 || k >= nz - 1)
38         return;
39
40     const int slice = nx * ny;
41     size_t pos = i + nx * j + slice * k;
42
43     // Compute the result of the stencil operation
44     f[pos] = u[pos] * invhxyz2
45             + (u[pos - 1]      + u[pos + 1]) * invhx2
46             + (u[pos - nx]    + u[pos + nx]) * invhy2
47             + (u[pos - slice] + u[pos + slice]) * invhz2;
48 }
```

Provided in this blog series are:

- Code examples for users to experiment around with
- Rocprof numbers guiding readers on the limiting factors of performance
- Roofline methodology to estimate the expected performance target
- Initial performance is not great
- Next two parts explain possible ways to improve

Full blog: [https://gpuopen.com/learn/amd-lab-notes/amd-lab-notes-finite-difference-docs-laplacian\\_part1/](https://gpuopen.com/learn/amd-lab-notes/amd-lab-notes-finite-difference-docs-laplacian_part1/)

# Finite Difference Method – Laplacian Part 2 & 3

Optimization tricks to incrementally improve the performance:

- Loop tiling
- Reordered memory access patterns
- Launch bounds
- Nontemporal memory access

```

27 // tiling factor
28 #define m 8
29 // Launch bounds
30 #define LB 256
31 template <typename T>
32 __launch_bounds__(LB)
33 __global__ void laplacian_kernel(T * f, const T * u, int nx, int ny, int nz,
34 T invhx2, T invhy2, T invhz2, T invhxyz2) {
35     int i = threadIdx.x + blockIdx.x * blockDim.x;
36     int j = m*(threadIdx.y + blockIdx.y * blockDim.y);
37     int k = threadIdx.z + blockIdx.z * blockDim.z;
38
39     // Exit if this thread is on the xz boundary
40     if (i == 0 || i >= nx - 1 ||
41         k == 0 || k >= nz - 1)
42         return;
43
44     const int slice = nx * ny;
45     size_t pos = i + nx * j + slice * k;
46
47     // Each thread accumulates m stencils in the y direction
48     T Lu[m] = {0};
49
50     // Scalar for reusable data
51     T center;

```

```

52
53 // z - 1, loop tiling
54 for (int n = 0; n < m; n++)
55     Lu[n] += u[pos - slice + n*nx] * invhz2;
56
57 // y - 1
58 Lu[0] += j > 0 ? u[pos - 1*nx] * invhy2 : 0; // bound check
59
60 // x direction, loop tiling
61 for (int n = 0; n < m; n++) {
62     // x - 1
63     Lu[n] += u[pos - 1 + n*nx] * invhx2;
64
65     // x
66     center = u[pos + n*nx]; // store for reuse
67     Lu[n] += center * invhxyz2;
68
69     // x + 1
70     Lu[n] += u[pos + 1 + n*nx] * invhx2;
71
72     // reuse: y + 1 for prev n
73     if (n > 0) Lu[n-1] += center * invhy2;
74
75     // reuse: y - 1 for next n
76     if (n < m - 1) Lu[n+1] += center * invhy2;
77 }
78
79 // y + 1
80 Lu[m-1] += j < ny - m ? u[pos + m*nx] * invhy2 : 0; // bound check
81
82 // z + 1, loop tiling
83 for (int n = 0; n < m; n++)
84     Lu[n] += u[pos + slice + n*nx] * invhz2;
85
86 // Store only if thread is inside y boundary
87 for (int n = 0; n < m; n++)
88     if (i + j > 0 && n + j < ny - 1)
89         __builtin_nontemporal_store(Lu[n], &f[pos + n*nx]);
90 }

```

Full blog: [https://gpuopen.com/learn/amd-lab-notes/amd-lab-notes-finite-difference-docs-laplacian\\_part2/](https://gpuopen.com/learn/amd-lab-notes/amd-lab-notes-finite-difference-docs-laplacian_part2/)



# MI200 memory space overview

The HIP API supports a wide variety of allocation methods for host and device memory. This post specifically focuses on the MI200 series GPUs and:

1. Introduces a set of commonly used memory spaces
2. Identifies what makes each memory space unique
3. Discusses some common use cases for each space

Specific topics cover:

- Host vs device memory
- Pageable vs pinned (host) memory
- Coarse-grained vs fine-grained coherence
- Managed memory
- Page migrations

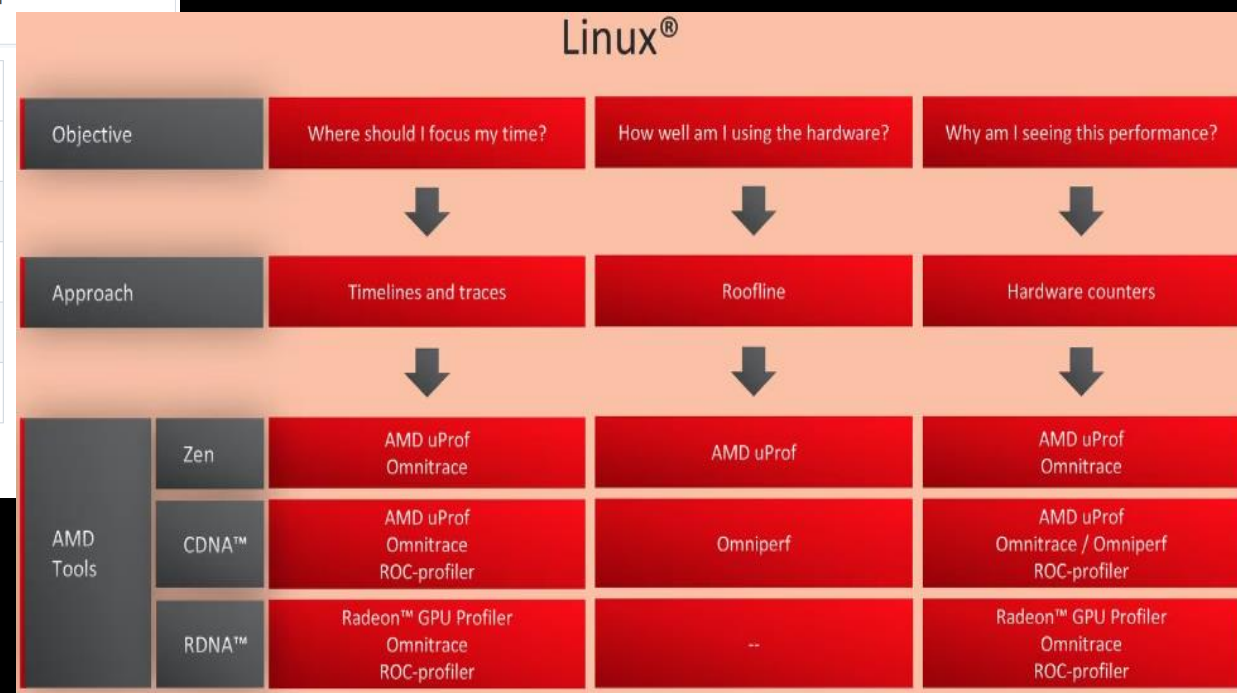
Full details and code snippets: <https://gpuopen.com/learn/amd-lab-notes/amd-lab-notes-mi200-memory-space-overview/>

# Introduction to profiling tools for AMD hardware

Term	Description
AMD "Zen" Core	AMD's x86-64 processor core architecture design. Used by the AMD EPYC™, AMD Ryzen™, AMD Ryzen™ PRO, and AMD Threadripper™ PRO processor series.
RDNA™	AMD's Traditional GPU architecture optimized for graphically demanding workloads like gaming and visualization. Includes the RX 5000, 6000 and 7000 GPUs.
CDNA™	AMD's Compute dedicated GPU architecture optimized for accelerating HPC, ML/AI, and data center type workloads. Includes the AMD Instinct™ MI50/60, MI100, and MI200 series accelerators.

AMD Profiling Tools	AMD "Zen" Core	RDNA™	CDNA™	Windows	Linux®
ROC-profiler	Not supported	☆	★	Not supported	★
Omniperf	Not supported	Not supported	★	Not supported	★
Omnitrace	★	☆	★	Not supported	★
Radeon™ GPU Profiler	Not supported	★	☆	★	☆
AMD uProf	★	Not supported	☆	★	☆

★ Full support | ☆ Partial support



Full blog with references to the appropriate documentation

<https://gpuopen.com/learn/amd-lab-notes/amd-lab-notes-profilers-readme/>

# Future AMD lab notes

Topic	Tentative Description
Jacobi Solver – HIP/OpenMP	Demonstrates how to implement a Jacobi solver in both HIP and OpenMP target offloading
GPU aware MPI	How to build and enable GPU aware communication across various MPI implementations (OpenMPI, Cray MPICH, etc.)
Sparse Matrix-Vector Multiply	Develop a HIP implementation of the SpMV operation, covering implementations like CSR and Ellpack
Reading ISA	Provides the necessary training to allow users to read and understand Instruction Set Architecture (ISA) for AMD GPU
Graph analytics	Implements common graph algorithms like Breadth-First Search (BFS) on AMD GPUs and compares against Gunrock
Seismic stencil codes	Applies optimization techniques to high-order finite difference schemes commonly seen in seismic wave propagation

If you have any questions or comments, please reach out to us on GitHub discussions:  
<https://github.com/amd/amd-lab-notes/discussions>

# PETSc on AMD GPUs

Introduced Completed the HIP backend to PETSc late last year.

- Initially part of the OpenFOAM® application porting and optimization effort.
- OpenFOAM® community preferred not to use the Kokkos backend of PETSc
- AMD ported the PETSc Mat class to the HIP backend in mid 2021, finishing in late 2022

One possible way of building with HIP/ROCm

```
1 #!/usr/bin/env python3
2 if __name__ == '__main__':
3     import sys
4     import os
5     sys.path.insert(0, os.path.abspath('config'))
6     import configure
7     ROCM_PATH='/opt/rocm-5.5.0/'
8     configure_options = [
9         '--package-prefix-hash='+petsc_hash_pkgs,
10        'COPTFLAGS=-g -O3',
11        'FOPTFLAGS=-g -O3',
12        'CXXOPTFLAGS=-g -O3',
13        'HIPOPTFLAGS=-g -O3',
14        '--with-cuda=0',
15        '--with-hip=1',
16        '--with-hipcc='+ROCM_PATH+'/bin/hipcc',
17        '--with-hip-dir='+ROCM_PATH,
18        '--with-cc=mpicc',
19        '--with-cxx=mpicxx',
20        '--with-fc=mpif90',
21        '--download-fblaslapack=1',
22        '--with-precision=double',
23    ]
24    configure.petsc_configure(configure_options)
```

NOTE: Due to the ever-changing nature of the ROCm software ecosystem, new releases will occasionally break something. **Our team is working on QA/CI for ROCm releases. In the meantime, please report issues to me @jychang48 on GitLab®**

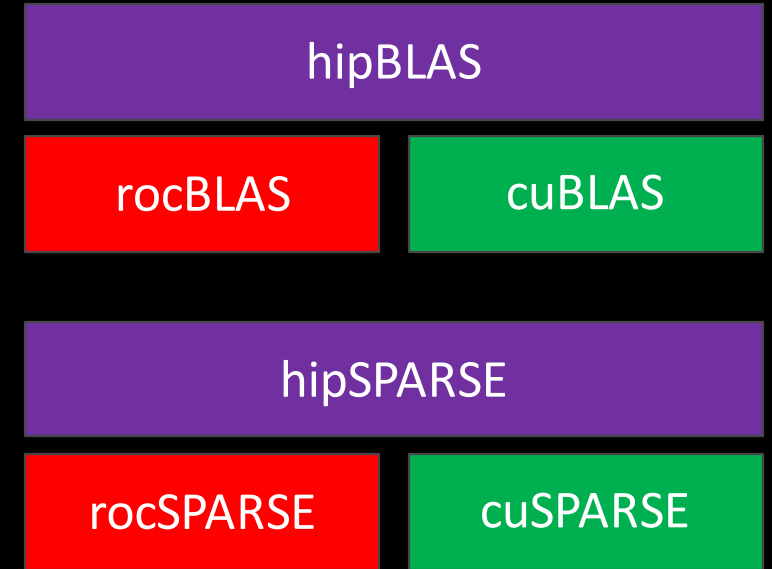
# AMD GPU Math Libraries

- A note on naming conventions:
  - roc\* -> AMGCN library usually written in HIP
  - cu\* -> NVIDIA PTX libraries
  - hip\* -> usually interface layer on top of roc\*/cu\* backends
- hip\* libraries:
  - Can be compiled by hipcc and can generate a call for the device you have:
    - hipcc->clang->AMD GCN ISA
    - hipcc->nvcc (inlined)->NVPTX
  - Just a thin wrapper that marshals calls off to a “backend” library:
    - corresponding roc\* library backend containing optimized GCN
    - corresponding cu\* library backend containing NVPTX for NVIDIA devices

We chose to port Mat class with hip\* libraries because

1. Vec class already ported using hip\*
2. Easier to implement

However, we recommend using roc\* libraries for greatest performance benefits



# PETSc GPU benchmark – Overview

- 27-point finite difference stencil for the Poisson in 3D, exact solution  $u(x,y,z) = 1.0$
- Located at: `src/ksp/ksp/tutorials/bench_kspsolve.c`
- Uses `MatSetValuesCOO()` to assemble the Matrix
- Designed to measure only the performance of `KSPSolve()` or just `MatMult()`
- Intentionally avoids use of `DMDA` or `DMPLex` to mimic how third-party applications leverage PETSc

```
./bench_kspsolve -mat_type aijhipsparse -n 200 -matmult
```

```
=====
Test: MatMult performance - Poisson
      Input matrix: 27-pt finite difference stencil
      -n 200
      -its 100
      DoFs = 8000000
      Number of nonzeros = 213847192

Step1 - creating Vecs and Mat...
Step2 - running MatMult() 100 times...

Average time: 0.00001 seconds
FOM:         8.301e+04 Gflops/sec
=====
```

```
./bench_kspsolve -mat_type aijhipsparse -n 200
```

```
=====
Test: KSP performance - Poisson
      Input matrix: 27-pt finite difference stencil
      -n 200
      DoFs = 8000000
      Number of nonzeros = 213847192

Step1 - creating Vecs and Mat...
Step2 - running KSPSolve()...
Step3 - calculating error norm...

Error norm: 1.375e+00
KSP iters: 103
KSPSolve: 4.65272 seconds
FOM: 1.719e+06 DoFs/sec
=====
```

Not limited to just GPU use

Can be used to quickly compare different solvers, backends, software versions, or even hardware

# PETSc GPU benchmark – Implementation details

Step 0 – Read command-line arguments

Step 1 – Create Vecs and Mat:

- Create the Mat Object and split across MPI ranks
- Allocate three COO arrays of size nnz based on user.Istart, user.Iend, and user.n
- Fill three COO arrays with corresponding FD coefficients and row/column indices
- Spawn Vecs and set exact solution 1.0
- Compute RHS Vec b

Step 2 – Measure performance

- Run MatMult() or KSPSolve()

Step 3 (KSPSolve only) – Compute error norm

```

/* Step 0 */
/* User input */
user.n      = 100;          /* Default grid points per dimension */
user.matmult = PETSC_FALSE; /* Test MatMult only */
user.its     = 100;        /* Default no. of iterations for MatMult test */
PetscCall(PetscOptionsGetInt(NULL, NULL, "-n", &user.n, NULL));
PetscCall(PetscOptionsGetBool(NULL, NULL, "-matmult", &user.matmult, NULL));
PetscCall(PetscOptionsGetInt(NULL, NULL, "-its", &user.its, NULL));
user.dim     = user.n * user.n * user.n;

/* Step 1 */
/* Create Mat Object */
PetscCall(MatCreate(PETSC_COMM_WORLD, &A));
PetscCall(MatSetSizes(A, PETSC_DECIDE, PETSC_DECIDE, user.dim, user.dim));
PetscCall(MatSetFromOptions(A));

/* Domain decomposition */
PetscCall(PetscSplitOwnership(PetscObjectComm((PetscObject)A), &nlocal, &user.dim));
PetscCallMPI(MPI_Scan(&nlocal, &user.Istart, 1, MPIU_INT, MPI_SUM, PetscObjectComm((PetscObject)A)));
user.Istart -= nlocal;
user.Iend   = user.Istart + nlocal;

/* Preallocate and fill COO matrix */
PetscCall(PreallocateCOO(A, &user)); /* Determine local number of nonzeros */
PetscCall(FillCOO(A, &user));        /* Fill COO Matrix */

/* Creating vecs */
PetscCall(MatCreateVecs(A, &u, &b));
if (!user.matmult) PetscCall(VecDuplicate(b, &x)); /* KSPSolve only */
PetscCall(VecSet(u, 1.0)); /* Exact solution */
PetscCall(MatMult(A, u, b)); /* Compute RHS based on exact solution */

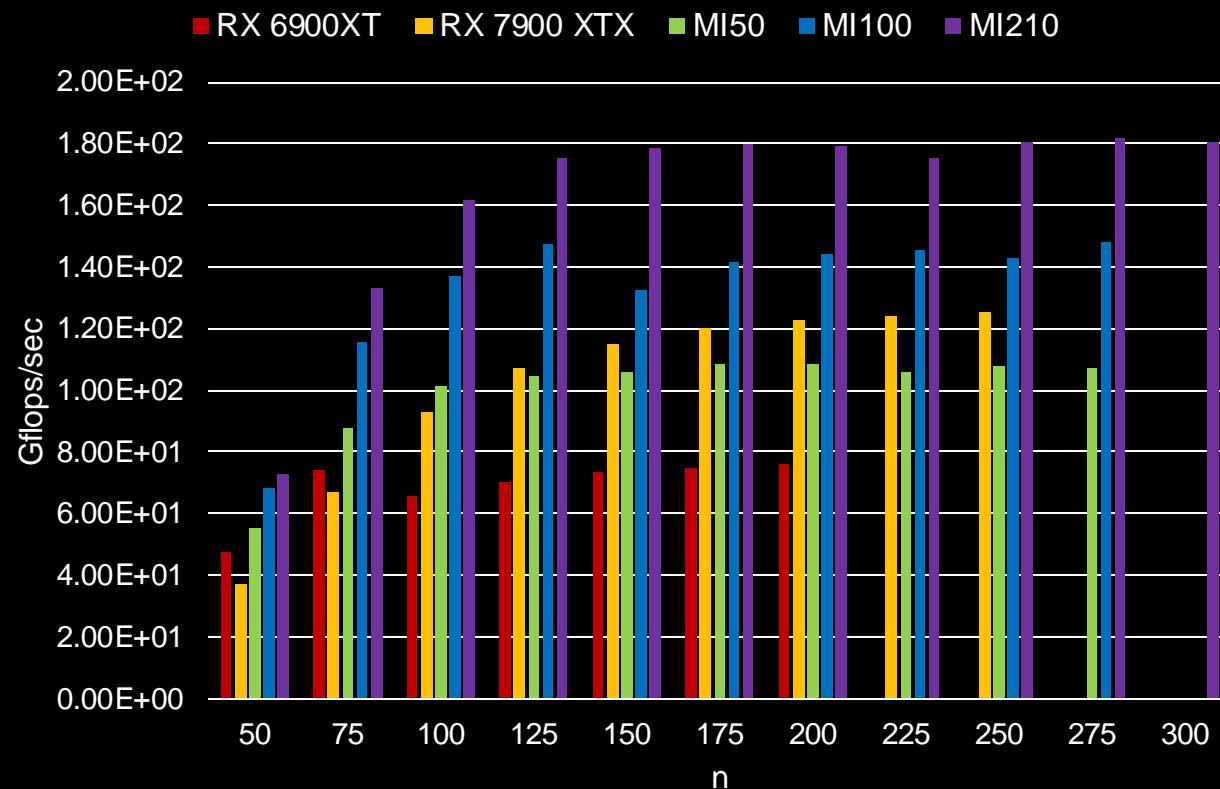
/* Step 2 */
/* Measure performance */
PetscCall(PetscTime(&time_start));
if (user.matmult) /* MatMult only */
    for (int i = 0; i < user.its; i++)
        PetscCall(MatMult(A, u, b));
else /* KSPSolve */
    PetscCall(KSPSolve(ksp, b, x));
PetscCall(PetscTime(&time_end));

/* Step 3 */
/* Calculate error norm, KSPSolve only */
if (!user.matmult) {
    PetscCall(VecAXPY(x, -1.0, u));
    PetscCall(VecNorm(x, NORM_2, &norm));
}

```

See bench\_kspsolve.c for full implementation details

# Work-time spectrum (MatMult only)



- Problem size is scaled up on a single AMD Radeon™/Instinct™ GPU
- Starting with  $-n$  50 (125k DoF, 3,241,792 nnz)
- Ending with  $-n$  300 (27M DoF, 724,150,792 nnz)
- Only the `MatMult()` operation is examined
- HIP backend built using ROCm 5.4.0

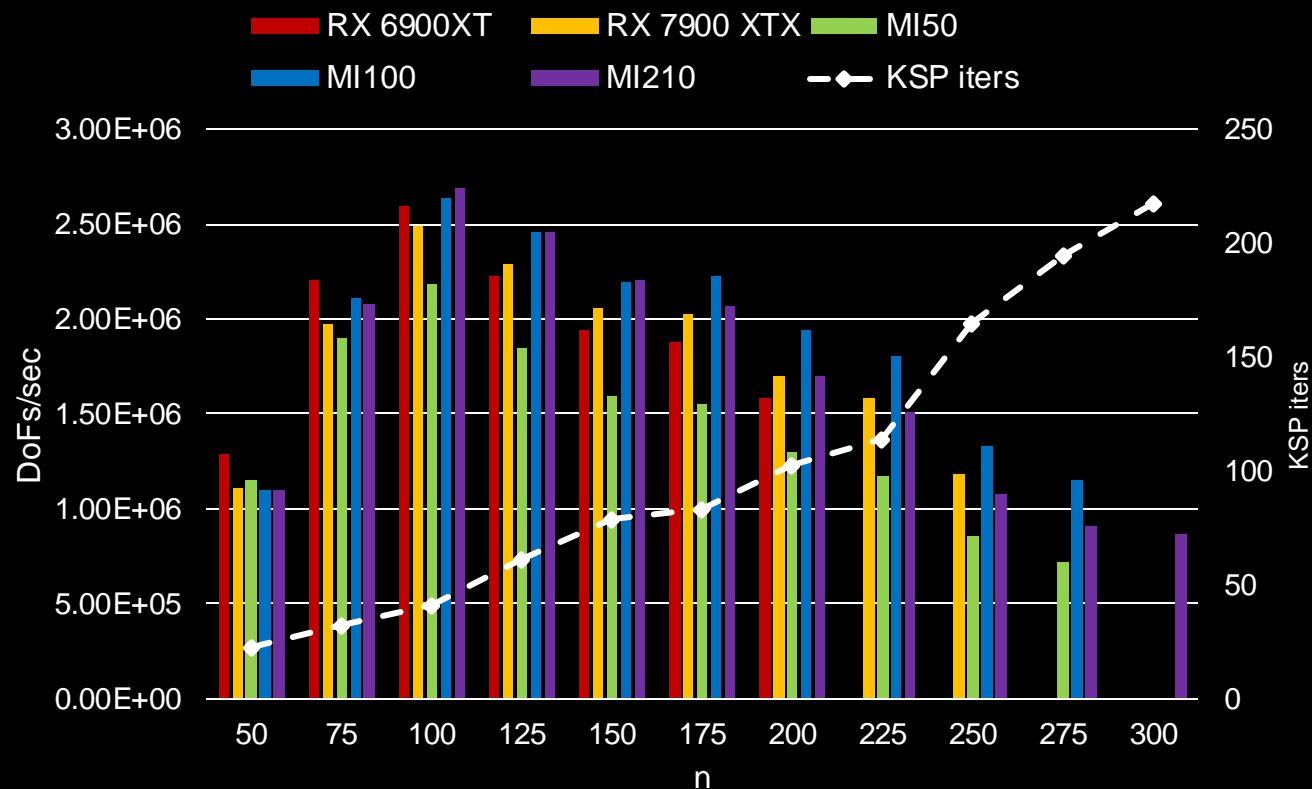
NOTE: Missing bars denote lack of GPU memory

```
mpirun -n 1 ./bench_kpsolve -matmult -mat_type aijhipsparse -n X
```

Testing conducted on single GPUs using ROCm version 5.4.0-72. The reported Gflops/sec are not validated performance numbers and are provided only as proof-of-concept. Actual Gflops/sec depend on multiple factors including system configuration and environment settings, reproducibility of the performance is not guaranteed.



# Work-time spectrum (KSPSolve)



```
mpirun -n 1 ./bench_kspsolve -matmult aijhipsparse -n X
```

- GMRES + ILU(0) solve (out-of-box configuration) is performed on the same problem sizes
- KSP iteration count grows with problem size
- Inverse correlation between DoFs/sec and KSP iterations
- If unusual trend in performance observed:
  1. `-log_view` to identify regions of interest
  2. `Omniperf` to identify why region performs the way it does

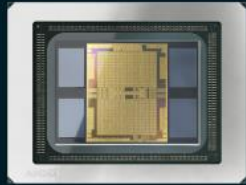
NOTE: Other solvers and ROCm versions may will exhibit different trends

Testing conducted on single GPUs using ROCm version 5.4.0-72. The reported DoFs/sec are not validated performance numbers and are provided only as proof-of-concept. Actual DoFs/sec depend on multiple factors including system configuration and environment settings, reproducibility of the performance is not guaranteed.

## Future plans for the PETSC GPU benchmark

- (Must have) Solve more than just the Poisson equation, possibly Q2 Elasticity
- (Must have) Allow users to provide their own matrix and vector
- (Nice to have) Reconsider leveraging PETSc DM objects
- (Nice to have) Incorporate into the PETSc library

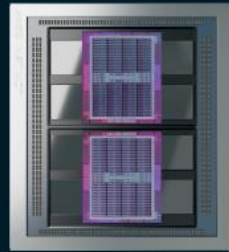
# AMD Data Center GPU Roadmap



AMD Instinct™ **MI100**  
AMD CDNA™

## Ecosystem Growth

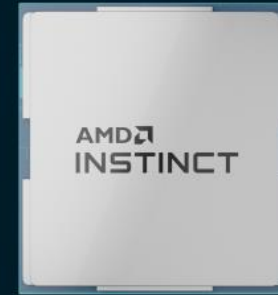
First purpose-built GPU architecture for the data center



AMD Instinct™ **MI200**  
AMD CDNA™ 2

## Driving HPC and AI to a New Frontier

First multi-die data center GPU expands scientific discovery and brings choice to AI training



AMD Instinct™ **MI300**  
AMD CDNA™ 3

## Data Center APU

Breakthrough architecture designed for leadership efficiency and performance for HPC and AI

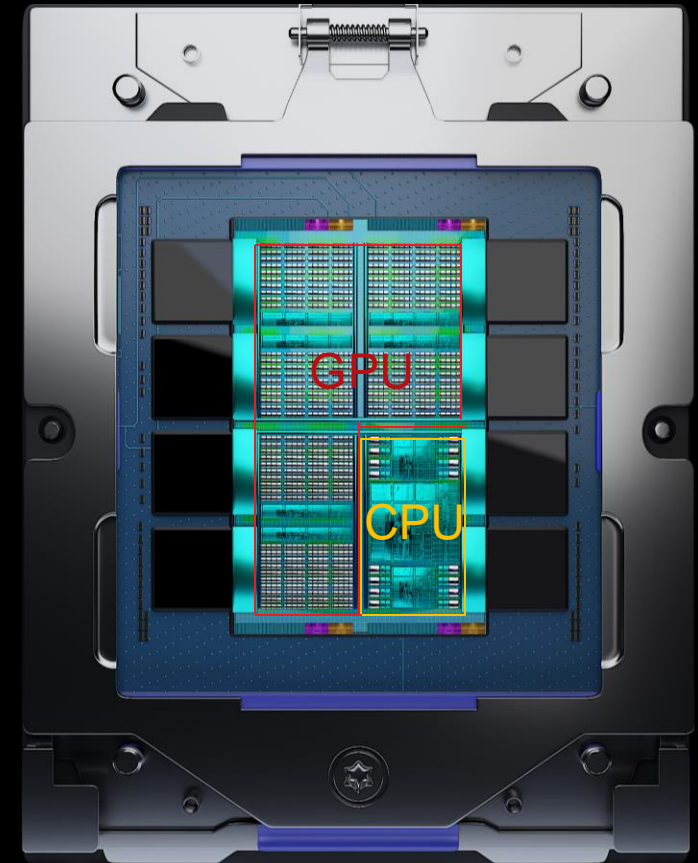
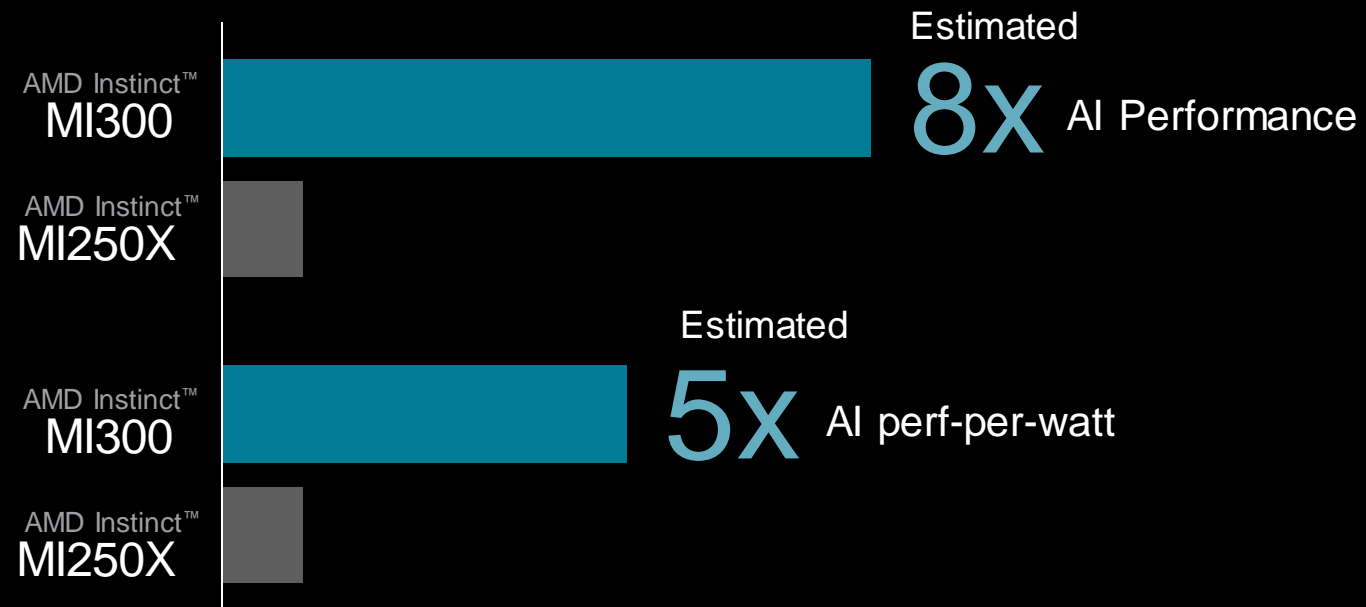
2020

2023

Source: <https://ir.amd.com/news-events/financial-analyst-day>

# MI300 Architectural Innovation at the Next Level

- 5nm process technology with 3D stacking
- Next-gen Infinity Cache™ and 4<sup>th</sup> Gen Infinity Fabric base die
- New Math formats
- Unified memory APU Architecture



# 3D CPU+GPU Integration for Next-Level Efficiency

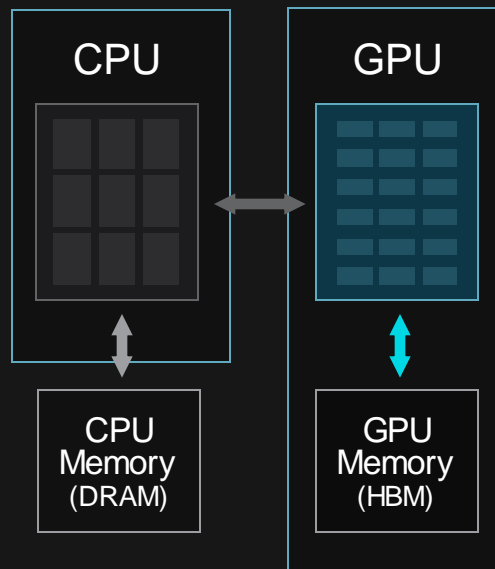
AMD CDNA™ 2 Coherent Memory Architecture



AMD CDNA™ 3 Unified Memory APU Architecture

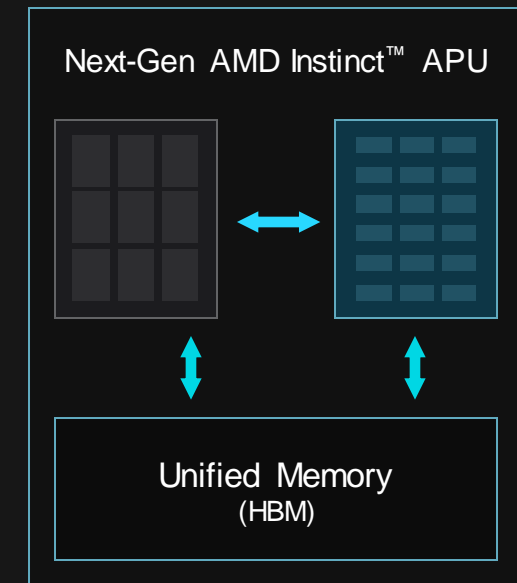
## AMD Instinct™ MI250 Accelerator

- Simplifies programming
- Low overhead 3<sup>rd</sup> Gen Infinity interconnect
- Industry standard modular design



## AMD Instinct™ MI300 Accelerator

- Eliminates redundant memory copies
- High bandwidth, low latency communication
- Low TCO with unified memory APU package



# APU programming model (HIP)

## CPU CODE

```
double* in_h = (double*)malloc(Msize);
double* out_h = (double*)malloc(Msize);
```

```
for (int i=0; i<M; i++) //initialize
    in_h[i] = ...;
```

```
cpu_func(in_d, out_d, M);
```

```
for (int i=0; i<M; i++) // CPU-process
    ... = out_h[i];
```

## GPU CODE

```
double* in_h = (double*)malloc(Msize);
double* out_h = (double*)malloc(Msize);
hipMalloc(&in_d, Msize);
hipMalloc(&out_d, Msize);
```

```
for (int i=0; i<M; i++) //initialize
    in_h[i] = ...;
hipMemcpy(in_d, in_h, Msize);
gpu_func<< >>(in_d, out_d, M);
hipDeviceSynchronize();
hipMemcpy(out_h, out_d, Msize);
```

```
for (int i=0; i<M; i++) // CPU-process
    ... = out_h[i];
```

## APU CODE

```
double* in_h = (double*)malloc(Msize);
double* out_h = (double*)malloc(Msize);
```

```
for (int i=0; i<M; i++) //initialize
    in_h[i] = ...;
```

```
gpu_func<< >>(in_h, out_h, M);
hipDeviceSynchronize();
```

```
for (int i=0; i<M; i++) // CPU-process
    ... = out_h[i];
```

- GPU memory allocation on Device
- Explicit memory management between CPU & GPU
- Synchronization Barrier

# APU programming model (OpenMP® Target Offloading)

## CPU CODE

```
double* in = (double*)malloc(Msize);
double* out = (double*)malloc(Msize);

for (int i=0; i<M; i++)
    in[i] = ...;

for (int i=0; i<M; i++)
    out[i] = ... in[i] ...;

for (int i=0; i<M; i++)
    ... = out[i];
```

## GPU CODE

```
double* in = (double*)malloc(Msize);
double* out = (double*)malloc(Msize);

for (int i=0; i<M; i++)
    in[i] = ...;

#pragma omp target data \
    map(to:in[0:Msize]) \
    map(from:out[0:Nsize])
{
#pragma omp target distribute parallel for
    for (int i=0; i<M; i++)
        out[i] = ... in[i] ...;
}

for (int i=0; i<M; i++)
    ... = out[i];
```

## APU CODE

```
double* in = (double*)malloc(Msize);
double* out = (double*)malloc(Msize);

for (int i=0; i<M; i++)
    in[i] = ...; //writes GPU mem directly

#pragma omp target distribute parallel for
for (int i=0; i<M; i++)
    out[i] = ... in[i] ...;

for (int i=0; i<M; i++)
    ... = out[i]; //reads GPU mem directly
```

- GPU memory allocation on Device
- Explicit memory management between CPU & GPU
- Synchronization Barrier

## What this APU means for PETSc

- Single set of memory, no more time spent on memory transfers or page migrations
- Continue using the same hip\*/roc\* math libraries with system memory
- Serial or smaller workloads may not need HIP/accelerated code
- Programming challenge: portability between an APU and traditional CPUs/GPUs
- AMD welcomes feedback on how we can help improve the ROCm ecosystem



# Concluding Remarks

- AMD has provided at least two mutual sources of documentation for ROCm needs
- The AMD lab notes is a blog series containing useful tips for both HPC and AI applications
- The PETSc HIP backend based on hip\* libraries was recently introduced for the Mat class
- A KSPSolve() benchmark was introduced to allow users to quickly compare the performance of different solvers, backends, and hardware
- The MI300 APU has a unified memory architecture and could offer PETSc tremendous improvements

**AMD** 

# Disclaimer and Attritions

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

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 MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY 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, AMD CDNA™, AMD Instinct™, AMD Radeon™, AMD RDNA™, ROCm, and combinations thereof, are trademarks of Advanced Micro Devices, Inc. Other names are for informational purposes only and may be trademarks of their respective owners. PCIe® is a registered trademark of PCI-SIG Corporation. GitLab is a registered trademark of GitLab, Inc. OPENFOAM® is a registered trademark of OpenCFD Limited, producer and distributor of the OpenFOAM software via [www.openfoam.com](http://www.openfoam.com). The OpenMP name and the OpenMP logo are registered trademarks of the OpenMP Architecture Review Board.

# Endnotes

End note MI300-003. Measurements by AMD Performance Labs June 4, 2022 on current specification and/or estimation for estimated delivered FP8 floating point performance with structure sparsity supported for AMD Instinct™ MI300 vs. MI250X FP16 (306.4 estimated delivered TFLOPS based on 80% of peak theoretical floating-point performance). MI300 performance based on preliminary estimates and expectations. Final performance may vary. MI300-003.

Measurements conducted by AMD Performance Labs as of Jun 7, 2022 on the current specification for the AMD Instinct™ MI300 APU (850W) accelerator designed with AMD CDNA™ 3 5nm FinFET process technology, projected to result in 2,507 TFLOPS estimated delivered FP8 with structured sparsity floating-point performance.

End note: MI300-04. Estimated delivered results calculated for AMD Instinct™ MI250X (560W) GPU designed with AMD CDNA™ 2 6nm FinFET process technology with 1,700 MHz engine clock resulted in 306.4 TFLOPS (383.0 peak FP16 x 80% = 306.4 delivered) FP16 floating-point performance. Actual results based on production silicon may vary. MI300-04