Running AMD GPUs and APUs on Alliance clusters

Pawel Pomorski ppomorsk@sharcnet.ca

July 15, 2026

Introduction

The Alliance has been using NVIDIA accelerator hardware over the last decade.

Small AMD accelerator hardware installations done mostly for experimental purposes over the years.

Since 2025 the Nibi cluster has 6 nodes with AMD MI300A accelerators.

We are likely to have more AMD accelerators in the future.

Users will need help to switch from NVIDIA to AMD accelerators.

AMD and NVIDIA accelerator models

Model	Vendor	Launch
H100	NVIDIA	Q1 2022
MI300A,MI300X	AMD	Q4 2023
H200	NVIDIA	Q4 2024
B200	NVIDIA	Q4 2024
MI350X,355X	AMD	Q2 2025
B300	NVIDIA	Q3 2025
MI430X,440X,455x	AMD	H2 2026
R100	NVIDIA	H2 2026
MI500 line	AMD	2027
Rubin Ultra line	NVIDIA	2027

MI430X is for conventional HPC (has FP64)

MI440X, MI455X are for AI (have low precision only)

No MI300A successor

ROCM - AMD's equivalent of CUDA

Released in 2016

ROCm is a stack of about 60-100 packages. Its structure is continually evolving.

HIP compiler hipcc can build both for AMD and NVIDIA, syntax is a clone of CUDA

CUDA vs. HIP

```
// CUDA code
#include <cuda.h>
cudaMalloc((void **) &x_dev, memsize)
cudaMemcpy(x_dev, x_host, memsize, cudaMemcpyHostToDevice)
cudaFree(x_dev);

// HIP code
#include "hip/hip_runtime.h"
hipMalloc((void **) &x_dev, memsize)
hipMemcpy(x_dev, x_host, memsize, hipMemcpyHostToDevice)
hipFree(x_dev);
```

Convert from CUDA to HIP with:

```
hipify-perl code.cu > code.cpp
```

Challenges with converting from CUDA to HIP

If using libraries, a ROCM version must exist.

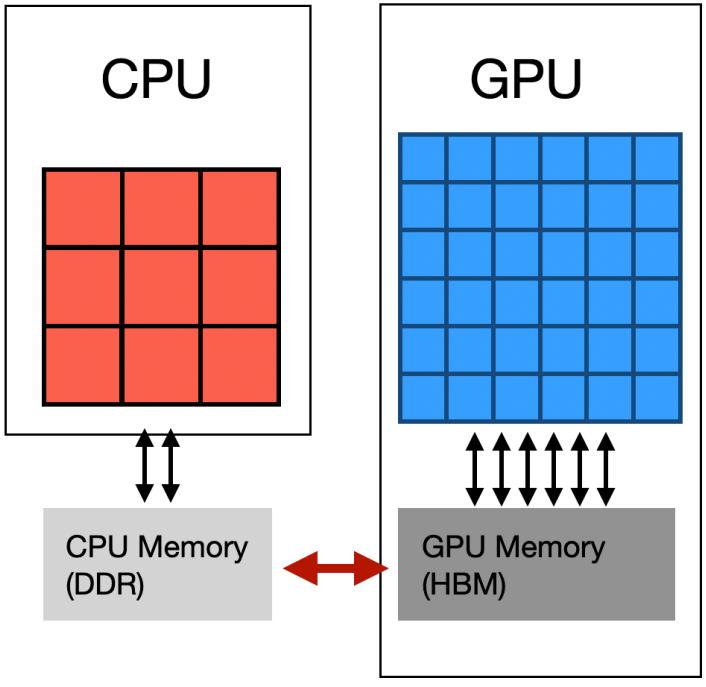
Latest CUDA features might not be supported by HIP.

For complex packages, build system conversion needed as well.

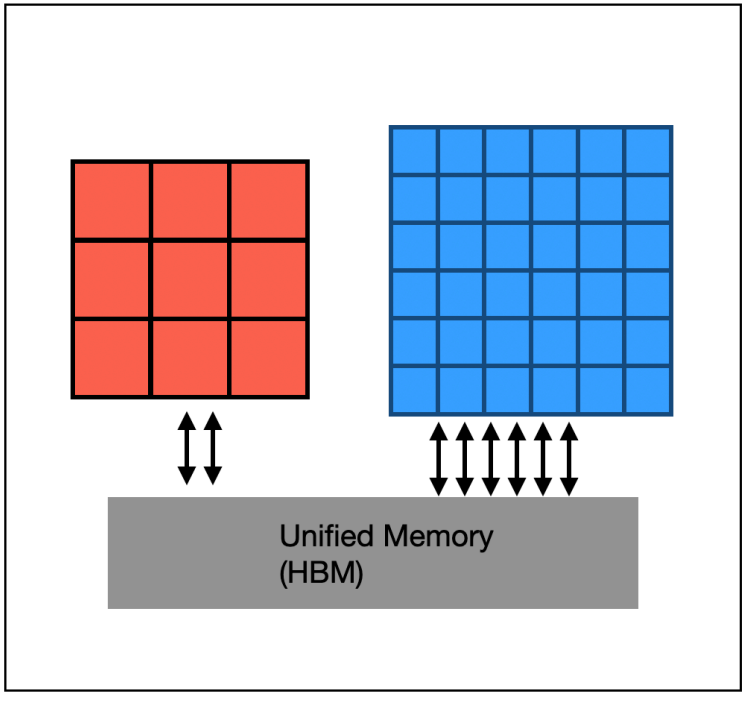
Architecture assumptions that are different for AMD must be changed manually.

Resulting code not necessarily optimized.

CPU+GPU vs. APU



Discrete CPU and GPU



APU

MI300A vs. H100 peak theoretical performance

	MI300A	H100 SXM
CPU cores	24	none
memory	128GB	80GB
bandwidth	5.3 TB/s	3.35 TB/s
FP32	122.6 TF	67 TF
FP64	61.3	34
FP64 matrix	122.6	67
TF32 (sparse)	980.6	989
FP16 (sp)	1961	1979
BFLOAT16 (sp)	1961	1979
INT8 (sp)	3922	3958
FP8 (sp)	3922	9358

Nibi MI300A nodes

6 nodes, each with 4 MI300A

Submit full node jobs with:

```
...
#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --cpus-per-task=24
#SBATCH --gpus=mi300a:4
...
```

Full node jobs leave it up to the user to control memory usage using numactl.

Job submission with single APU ensuring only the memory of that APU is used has not been implemented yet.

Example memory issue with MI300A

As memory is unified, must always leave some memory for the operating system.

Some programs test memory by malloc until failure so will fill it up

This can be prevented by defining variables:

GPU_MAX_ALLOC_PERCENT

GPU_SINGLE_ALLOC_PERCENT

What are the benefits of unified memory?

Can CPU-only programs benefit since they can use HMB memory?

When do GPU programs benefit? Are changes to the code required?

Stream Benchmark on MI300A

```
/opt/software/aocc-compiler-5.1.0/bin/clang -O3 \
-mcmodel=large -DSTREAM_TYPE=double -mavx2 \
-DSTREAM_ARRAY_SIZE=268435456 -DNTIMES=10 \
-ffp-contract=fast -fnt-store -lomp -fopenmp \
-mprefer-vector-width=256 -mllvm \
-force-vector-interleave=2 stream.c -o stream

OMP_NUM_THREADS=24 numactl --cpunodebind=0 --membind=0 \
./stream

Function  Best Rate MB/s  Avg time    Min time    Max time
Copy:        305039.3     0.014093    0.014080    0.014116
```

At about 0.3 TB/s this falls well short theoretical peak of 5.3 TB/s

Need to use GPU threads to achieve theoretical peak, so CPU only programs don't benefit.

SAXPY example

Add two vectors $y = \alpha x + y$

- Allocate and initialize x,y on CPU. (use CudaMallocHost)
- Allocate x_dev,y_dev on GPU (use CudaMalloc)
- Copy x,y data to x_dev, y_dev
- Perform the operation on GPU using x_dev,y_dev
- Copy y_dev to y

Results (vector length $N = 32 * 1024 * 1024$)

```
H100

saxpy kernel on gpu: 0.172 ms
saxpy kernel + memcopy operations: 8.15 ms
cpu kernel: 14.19 ms

MI300A

saxpy kernel on gpu: 0.143 ms
saxpy kernel + memcopy operations: 7.39 ms
cpu kernel 10.77 ms
```

On MI300A memory copies still take time even though data is copied within unified memory.

In both cases using the GPU is not worthwhile due to memory copy overhead.

SAXPY example with unified memory

Add two vectors $y = \alpha x + y$

- Allocate and initilize x,y on CPU (use CudaMallocHost)
- Perform the operation on GPU using data x,y

No memory copies are performed, saving time.

Results (vector length $N = 32 * 1024 * 1024$)

```
MI300A

saxpy kernel on gpu: 0.143 ms
saxpy kernel + memcopy operations: 7.39 ms
cpu kernel 10.77 ms

MI300A with unified memory

saxpy kernel on gpu: 0.276 ms
cpu kernel 10.13 ms
```

Useful speedup achieved with modified code that takes advantage of unified memory.

The kernel working on arrays allocated with CudaMallochost is less efficient that on those allocated with CudaMalloc, even though they both reside in the same unified memory.

GROMACS benchmark

ROCM version for AMD was built from source in a container since some of the required dependencies were missing in /opt/rocm.

STMV benchmark results (higher is better)

H100 42.445 ns/day

MI300A 35.304 ns/day

H100 theoretical peak for FP32 was 67 TF, for MI300A it is 122.6 TF.

This could suggest MI300A performance falls short, possibly due to suboptimal compilation, or AMD GPU specific optimizations not implemented in source code.

NAMD benchmark

Using NAMD module in cvmfs.

STMV benchmark results for single GPU (higher is better)

H100 14.1565 ns/day

MI300A 12.2954 ns/day

MI300A seems to underperform like it did for GROMACS.

FlashAttention benchmark

FlashAttention is a more efficient implementation of the attention algorithm, introduced in 2022.
It improves performance by reducing the number of HBM memory accesses by a factor of 10.

FlashAttention-2 representative results

H100 310.33 TFLOPs

MI300A 233.42 TFLOPs

Compiling code from source takes about 24 hours due to the high amount of optimization done.

The successor FlashAttention-3 was rewritten to be specifically optimized for H100, using Hopper's asynchronous features not available yet in AMD GPUs, achieving further speedup of 1.5-2.0x.

Software availability for AMD accelerators

Some popular packages have been ported to use AMD GPUs but far from all.

Popular machine learning software (eg. PyTorch) has been ported to AMD GPUs.

CUDA is still the default for latest machine learning research codes and bleeding edge tools, so there may be a lag before those are ported to AMD GPUs.

AMD software on Alliance clusters

Through multiple years of effort the Alliance has a rich software stack for CUDA software.

In comparison the software stack for AMD GPU software is in very early stages. Efforts could pick up if more AMD GPU hardware is acquired by the Alliance.

Installing ROCm into /opt

Simplest option, standard way to install on Linux, using standard packages

Currently available on Nibi

Users can use this to build software from source. Requires knowledgeable users.

Issues with tracking multiple versions. Need modules once this becomes sufficiently complex.

Containers

Good for complex installs like PyTorch. Might be needed for some software even in fully developed Alliance software stack.

ROCm releases many versions per year (12 major and minor versions in 2025 alone) and it might be impractical to install a module for each one. Using containers to accommodate diverse version needs might be the solution.

Containers are readily available for download for machine learning software. For some software the container might be out of date.

PyTorch from container

```
module load apptainer
apptainer pull \
docker://rocm/pytorch:rocm7.2_ubuntu24.04_py3.12\
_pytorch_release_2.7.1

# SIF file is 9.6G

apptainer shell --rocm \
pytorch_rocm7.2_ubuntu24.04_py3.12\
_pytorch_release_2.7.1.sif

python
>>> import torch
>>> print(torch.cuda.is_available())
True
```

Spack

An alternative to easybuild, gives more user control, enables installs in user space without root.

AMD GPU support is variable.

Alliance software stack uses EasyBuild

EasyBuild v5.3.0 (10 Apr 2026) added support for ROCm toolchains.

That enabled efforts to install first ROCm modules and software.

Umbrella ROCm module to load multiple dependent modules with simple command could be useful.

Support for AMD GPU software dependent on upstream efforts.

Current ROCm support in Alliance software stack

```
[user@l3.nibi ~]$ module spider rocm | grep ": rocm"
  rocm-cmake: rocm-cmake/0.14.0
  rocm-compilers: rocm-compilers/19.0.0
  rocm-smi: rocm-smi/7.6.0
  rocminfo: rocminfo/1.0.0

[user@l3.nibi ~]$ module spider namd-multicore/3.0.2
...
      StdEnv/2023  gcc/12.3
      StdEnv/2023  gcc/12.3  cuda/12.6
      StdEnv/2023  rocm-compilers/19.0.0
```

Conclusion

As we are likely to acquire more AMD accelerators hardware in the future, we will need to develop our ROCm software stack and provide more packages for users.

Users may need help porting their software to run on AMD GPUs.

Providing capabilities equal to those provided by CUDA will be a challenge.