ROCm: AMD's platform for GPU computing

Pawel Pomorski, *HPC Software Analyst*

SHARCNET, University of Waterloo

ppomorsk@sharcnet.ca

What is ROCm?

AMD's open-source software development platform for HPC GPU computing

Launched as "Boltzmann Initiative" in 2015

Alternative to NVIDIA's CUDA

Key goal is easy transfer of code between GPU architectures

Easier alternative to OpenCL

ROCm components include

Runtime and kernel driver (can also use upstream drivers)

OpenCL driver and compiler

HIP compiler and tools

ROCm libraries: BLAS, RAND, FFT, SPARSE

Debugger

Profiler

AMD GPU current generations

GPU family	year	arch	instruction set	ROCm support*
Arctic Islands/ Polaris	2016	gfx8..	GCN 4	Yes
Vega	2017	gfx9..	GCN 5	Yes
Navi..	2019	gfx10.. gfx11..	RDNA	No* (2022?)
Arcturus (MI100)	2020	gfx908	CDNA	Yes

Radeon Instinct - AMD's Tesla

Model	Release	Cores	arch
MI6	2016	2304 (SP only)	gfx803
MI8	2016	4096 (SP only)	gfx803
MI25	2017	4096 (SP only)	gfx901
MI50	2018	3840: 60 CU	gfx906
MI60	2018	4096: 64 CU	gfx906
MI 100	2020	7680: 120 CU matrix cores	gfx908

Upcoming: MI200, server APUs

Experimental AMD GPU node on graham

8 x Radeon RX 5700 XT

Released July, 2019

Current price: about $1500

Cores: 2560

Architecture: Navi 10, RDNA 1.0, gfx1010

ROCm support: Not officially supported. But partial support seems to be available.   Two possible drivers: ROCm and upstream.

AMD GPUs on AWS cloud

Radeon Pro V520

Released December, 2020, cloud only offering

Cores: 2304

Architecture: Navi 12, RDNA 1.0, gfx1011

ROCm support: Not officially supported. But partial support seems to be available. AWS driver bundle possibly more reliable.

Cloud offers more flexibility for debugging than HPC cluster.

ROCm with Singularity

It would be difficult to build the many packages of ROCm from source with the Compute Canada environment.

Script for building from source within various distributions exist, but would have to be modified extensively for our environment

Hence it is much more convenient to create a singularity image with a standard Linux distribution.

A single image can be configured to work on machines with NVIDIA and with AMG GPUs.

Working GPU drivers must be installed on host machine

ROCm with singularity

```
[ppomorsk@gra1339 scratch]$ singularity run -C --app rocm -B/home --rocm rocm4.3-
cuda10.1-centos7.sif /bin/bash -i

Singularity> hipify-perl saxpy_cuda.cu > saxpy_cuda.cpp
Singularity> hipcc saxpy_cuda.cpp
Singularity> ./a.out
```

```
[ppomorsk@gra1340 scratch]$ singularity run -C --app cuda -B/home --nv rocm4.3-cuda10.1-
centos7.sif /bin/bash -i
Singularity> hipify-perl saxpy_cuda.cu > saxpy_cuda.cpp
Singularity> hipcc saxpy_cuda.cpp
Singularity> nvprof ./a.out
==70== NVPROF is profiling process 70, command: ./a.out
test comparison shows 0 errors
==70== Profiling application: ./a.out
==70== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   52.43%  59.747ms         2  29.873ms  29.842ms  29.905ms  [CUDA memcpy
HtoD]
                   46.78%  53.304ms         1  53.304ms  53.304ms  53.304ms  [CUDA memcpy
DtoH]
                    0.79%  901.02us         1  901.02us  901.02us  901.02us
saxpy_gpu(float*, float*, float, int)
```

HIP

HIP is a C++ dialect similar in syntax to CUDA that can be compiled to run on both NVIDIA and AMD GPUs.

HIP provides a “strong subset” of functionality in CUDA, but some features are (currently) not supported
Examples: dynamic parallelism, float 16

HIP code provides the same performance as native CUDA code, plus the benefit of running on AMD platforms

Tools provided for automatic conversion from CUDA to HIP

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

HIP vs CUDA syntax

```
! HIP code
hipMalloc((void **) &x_dev, memsize);

hipMemcpy(x_dev, x_host, memsize, hipMemcpyHostToDevice);

hipLaunchKernelGGL(saxpy_gpu, dim3(nBlocks),
dim3(blockSize), 0, 0, y_dev, x_dev, alpha, n);

hipDeviceSynchronize();
```

```
! CUDA
cudaMalloc((void **) &x_dev, memsize);

cudaMemcpy(x_dev, x_host, memsize, cudaMemcpyHostToDevice);

saxpy_gpu<<<nBlocks, blockSize>>>(y_dev, x_dev, alpha, n);

cudaDeviceSynchronize();
```

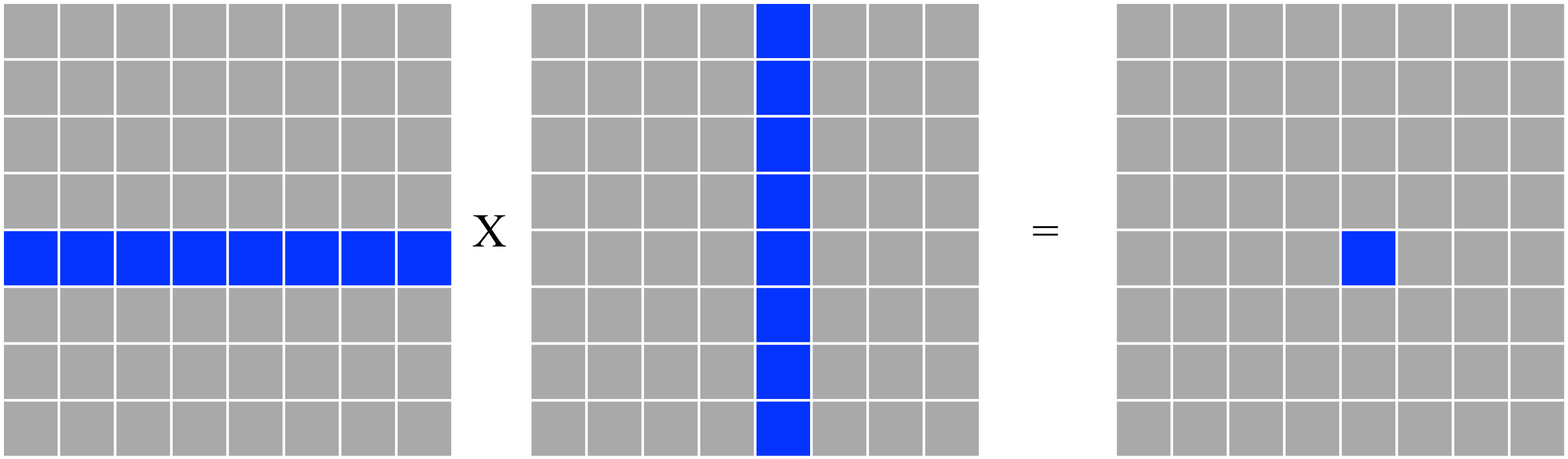
ROCm kernels exactly the same as in CUDA

```
! identical in both CUDA and HIP

__global__ void saxpy_gpu(float *vecY, float *vecX,
float alpha ,int n) {
    int i;

    i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i<n)
        vecY[i] = alpha * vecX[i] + vecY[i];
}
```

Note: Deprecated hipThreadIdx_x etc. still found in old documentation.

Matrix multiplication

A x B = C



Simple matrix multiplication

```
__global__ void simpleMultiply_gpu(float *a, float
*b, float *c ,int N)
{

  int row=blockIdx.y*blockDim.y + threadIdx.y;
  int col=blockIdx.x*blockDim.x + threadIdx.x;

  float sum = 0.0f;

  for (int i=0; i < N; i++) {
      sum+= a[row*N+i]*b[i*N+col];
  }
  c[row*N+col]=sum;
}
```

Shared memory matrix multiplication

```
__global__ void simpleMultiply_o2_gpu(float *a, float *b, float *c ,int N){
  __shared__ float atile[TILE_DIM][TILE_DIM], btile[TILE_DIM][TILE_DIM];
  int row=blockIdx.y*blockDim.y + threadIdx.y;
  int col=blockIdx.x*blockDim.x + threadIdx.x;
  int row_in_tile = threadIdx.y;
  int col_in_tile = threadIdx.x;
  float sum = 0.0f;

  for(int t=0; t<N/TILE_DIM; t++) {
    atile[row_in_tile][col_in_tile]=a[row*N+t*TILE_DIM+col_in_tile];
    btile[row_in_tile][col_in_tile]=b[(t*TILE_DIM+row_in_tile)*N+col];
    __syncthreads();

    for (int i=0; i < TILE_DIM; i++) {
      sum+= atile[row_in_tile][i]*btile[i][col_in_tile];}
    __syncthreads();
  }
  c[row*N+col]=sum;}
```

Matrix multiplication performance N=8192

GPU	simple	shared memory	BLAS
Radeon Pro 520V 2304 cores max 7373 GFLOPS	4.98 s 220 GFLOPS	1.34 s 824 GFLOPS	not available
Tesla P100 3584 cores max 9526 GFLOPS	5.21 s 211 GFLOPS	0.502s 2188 GFLOPS	0.144 s 7593 GFLOPS

rocprof profiler

rocprof -i input.xml --timestamp on ./a.out

input.xml contains detailed specification of metrics to be collected

output in CSV format

Metrics not available if GPU not supported by ROCm

NAMD

NAMD provides version that supports ROCm (binary + source)

STMV benchmark, using 16 CPU cores, 1 GPU

GPU	s/step	days/ns
AMD MI50 (2018)	0.051	0.59
AMD MI100 (2020)	0.038	0.44
Tesla P100 (2016)	0.050	0.58
Tesla T4 (2018)	0.054	0.63

Other software ported to ROCm, see AMD Infinity Hub

Machine learning with ROCm

PyTorch and TensorFlow for ROCm available, with upstream support for easy installation

MIOpen libraries for high performance machine learning primitives

Running with OpenCL only also a possibility

Newly introduced MI100 GPU with matrix cores analogous to NVDIA’s tensor cores