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

<table>
<thead>
<tr>
<th>GPU family</th>
<th>year</th>
<th>arch</th>
<th>instruction set</th>
<th>ROCm support*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arctic Islands/</td>
<td>2016</td>
<td>gfx8..</td>
<td>GCN 4</td>
<td>Yes</td>
</tr>
<tr>
<td>Polaris</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vega</td>
<td>2017</td>
<td>gfx9..</td>
<td>GCN 5</td>
<td>Yes</td>
</tr>
<tr>
<td>Navi..</td>
<td>2019</td>
<td>gfx10..</td>
<td>RDNA</td>
<td>No* (2022?)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gfx11..</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arcturus</td>
<td>2020</td>
<td>gfx908</td>
<td>CDNA</td>
<td>Yes</td>
</tr>
<tr>
<td>(MI100)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Radeon Instinct - AMD’s Tesla

<table>
<thead>
<tr>
<th>Model</th>
<th>Release</th>
<th>Cores</th>
<th>arch</th>
</tr>
</thead>
<tbody>
<tr>
<td>MI6</td>
<td>2016</td>
<td>2304 (SP only)</td>
<td>gfx803</td>
</tr>
<tr>
<td>MI8</td>
<td>2016</td>
<td>4096 (SP only)</td>
<td>gfx803</td>
</tr>
<tr>
<td>MI25</td>
<td>2017</td>
<td>4096 (SP only)</td>
<td>gfx901</td>
</tr>
<tr>
<td>MI50</td>
<td>2018</td>
<td>3840: 60 CU</td>
<td>gfx906</td>
</tr>
<tr>
<td>MI60</td>
<td>2018</td>
<td>4096: 64 CU</td>
<td>gfx906</td>
</tr>
<tr>
<td>MI 100</td>
<td>2020</td>
<td>7680: 120 CU matrix cores</td>
<td>gfx908</td>
</tr>
</tbody>
</table>

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.
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**

```bash
[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
==70== test comparison shows 0 errors
==70== Profiling application: ./a.out
==70== Profiling result:

<table>
<thead>
<tr>
<th>Type</th>
<th>Time(%)</th>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU activities:</td>
<td>52.43%</td>
<td>59.747ms</td>
<td>2</td>
<td>29.873ms</td>
<td>29.842ms</td>
<td>29.905ms</td>
<td>[CUDA memcpy</td>
</tr>
<tr>
<td>HtoD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[CUDA memcpy</td>
</tr>
<tr>
<td></td>
<td>46.78%</td>
<td>53.304ms</td>
<td>1</td>
<td>53.304ms</td>
<td>53.304ms</td>
<td>53.304ms</td>
<td>[CUDA memcpy</td>
</tr>
<tr>
<td></td>
<td>0.79%</td>
<td>901.02us</td>
<td>1</td>
<td>901.02us</td>
<td>901.02us</td>
<td>901.02us</td>
<td>saxpy_gpu(float*, float*, float, int)</td>
</tr>
</tbody>
</table>
```
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

```c
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

```c
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

ROCm kernels exactly the same as in CUDA

! identical in both CUDA and HIP

```c
__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

\[ A \times B = C \]
Simple matrix multiplication

```c
__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

```c
__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

<table>
<thead>
<tr>
<th>GPU</th>
<th>simple</th>
<th>shared memory</th>
<th>BLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radeon Pro 520V</td>
<td>4.98 s</td>
<td>1.34 s</td>
<td>not available</td>
</tr>
<tr>
<td>2304 cores</td>
<td>220 GFLOPS</td>
<td>824 GFLOPS</td>
<td></td>
</tr>
<tr>
<td>max 7373 GFLOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tesla P100</td>
<td>5.21 s</td>
<td>0.502s</td>
<td>0.144 s</td>
</tr>
<tr>
<td>3584 cores</td>
<td>211 GFLOPS</td>
<td>2188 GFLOPS</td>
<td>7593 GFLOPS</td>
</tr>
<tr>
<td>max 9526 GFLOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>GPU</th>
<th>s/step</th>
<th>days/ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD MI50</td>
<td>0.051</td>
<td>0.59</td>
</tr>
<tr>
<td>AMD MI100</td>
<td>0.038</td>
<td>0.44</td>
</tr>
<tr>
<td>Tesla P100</td>
<td>0.050</td>
<td>0.58</td>
</tr>
<tr>
<td>Tesla T4</td>
<td>0.054</td>
<td>0.63</td>
</tr>
</tbody>
</table>

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 NVIDIA’s tensor cores