Compiling and running code on GPU nodes

Johan Hellsvik

Reference pages:

Building for AMD GPUs

Introduction to GPUs course, September and October 2023

Generalized programming for GPUs

Central processing units (CPU) and graphics processing units (GPU) do different work

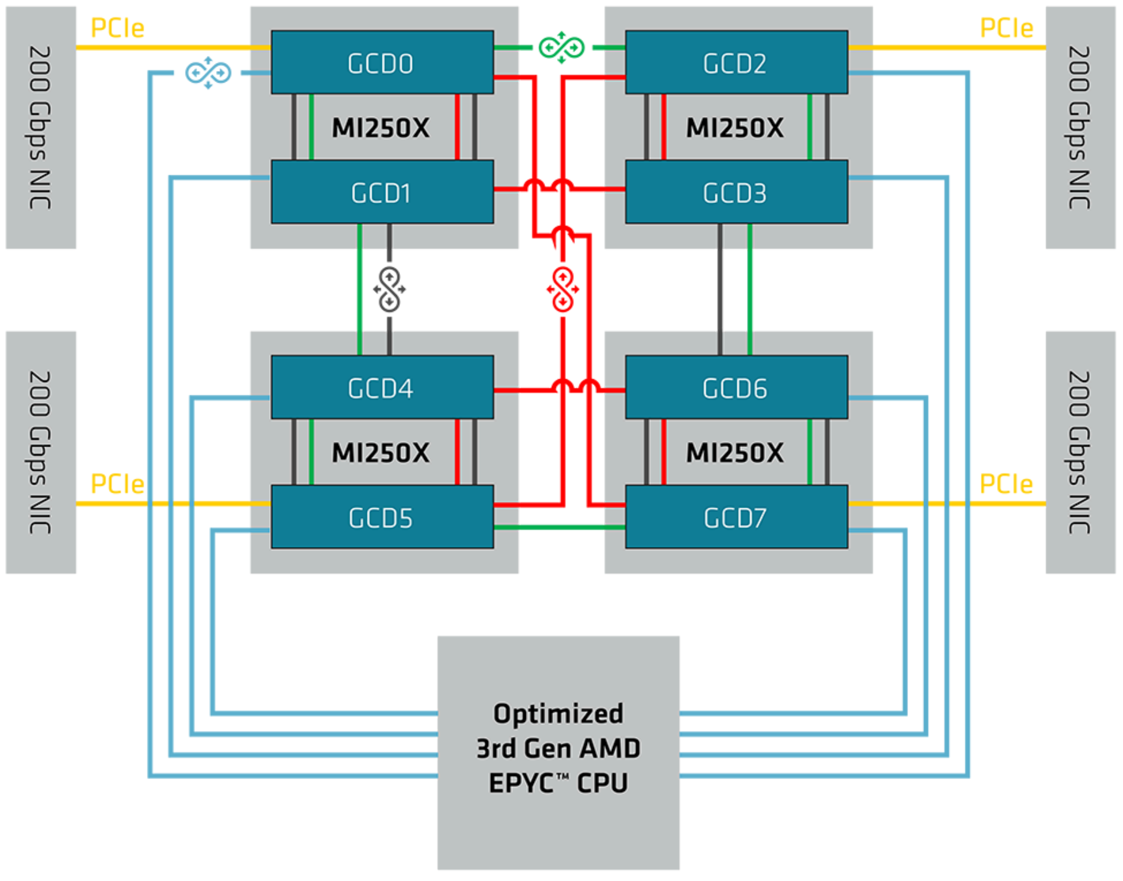
- CPUs have large instruction sets and execute general code.
- GPUs have smaller instructions sets. Runs compute intensive work in parallel on large number of compute units (CU).
- Code execution is started and controlled from the CPU. Compute intensive work is offloaded to the GPU.

Dardel GPU nodes

Dardel has 56 GPU nodes, each of which is equipped with

- One AMD EPYC™ processor with 64 cores
- 512 GB of shared fast HBM2E memory
- Four AMD Instinct™ MI250X GPUs (with an impressive performance of up to 95.7 TFLOPS in double precision when using special matrix operations)

Optimized 3rd Gen AMD EPYC™ Processor + AMD Instinct™ MI250X Accelerator



Green, Red, Gray, and Blue lines are AMD Infinity Fabric™ Links
Red and Green links can create two bi-directional rings
Blue Infinity Fabric Link provides coherent GCD-CPU connection

Orange lines are PCIe® Gen4 with ESM

AMD Radeon Open Compute (ROCm)

The AMD Radeon Open Compute (ROCm) platform is a software stack for programming and running of programs on GPUs.

- The ROCm platform supports different programming models
  - Heterogeneous interface for portability (HIP)
  - Offloading to GPU with OpenMP directives
  - The SYCL programming model
- AMD ROCm Information Portal

Setting up a GPU build environment

- Load the PDC/23.03 module and version 5.0.2 of ROCm with
  - `ml PDC/23.03`
  - `ml rocm/5.0.2`
- Set the accelerator target to **amd-gfx90a** (AMD MI250X GPU)
  - `ml craype-accel-amd-gfx90a`
- Choose one of the available toolchains (Cray, Gnu, AOCC)
  - `ml cpeCray/23.03`
  - `ml cpeGNU/23.03`
  - `ml cpeAOCC/23.03`

The ROCM info command

Information on the available GPU hardware can be displayed with the `rocminfo` command. Example output (truncated)

```
ROCk module is loaded
=====================
HSA System Attributes
=====================
Runtime Version:         1.1
System Timestamp Freq.:  1000.000000MHz

==========
HSA Agents
==========
*******
Agent 1
*******
  Name:                    AMD EPYC 7A53 64-Core Processor
  Uuid:                    CPU-XX
```

The CRAY_ACC_DEBUG runtime environment variable

For executables that are built with the compilers of the Cray Compiler Environment (CCE), verbose runtime information can be enabled with the environment variable `CRAY_ACC_DEBUG` which takes values 1, 2 or 3.

For the highest level of information

```
export CRAY_ACC_DEBUG=3
```

Offloading to GPU with HIP

The heterogeneous interface for portability (HIP) is a hardware close (low level) programming model for GPUs. Example lines of code:

- Include statement for the HIP runtime

```
#include <hip/hip_runtime.h>
```

- HIP functions have names starting with `hip`

```
// Get number of GPUs available
if (hipGetDeviceCount(&ndevices) != hipSuccess) {
    printf("No such devices\n");
    return 1;
    }
printf("You can access GPU devices: 0-%d\n", (ndevices - 1));
```

- Explicit handling of memory on the GPU

```
// Allocate memory on device
hipMalloc(&devs1, size);
hipMalloc(&devs2, size);
// Copy data host -> device
hipMemcpy(devs1, hosts1, size, hipMemcpyHostToDevice);
```

- Call to run the compute kernel on the GPU

```
// Run kernel
hipLaunchKernelGGL(MyKernel, ngrid, nblock, 0, 0, devs1, devs2);
```

Offloading to GPU with OpenMP

The OpenMP programming model can be used for directive based offloading to GPUs.

Example: A serial code that operates on arrays `vecA`, `vecB`, and `vecC`

```
! Dot product of two vectors
do i = 1, nx
   vecC(i) =  vecA(i) * vecB(i)
end do
```

Implement OpenMP offloading by inserting OpenMP directives. In Fortran the directives starts with `!$omp`

```
! Dot product of two vectors
!$omp target teams distribute map(from:vecC) map(to:vecA,vecB)
do i = 1, nx
   vecC(i) =  vecA(i) * vecB(i)
end do
!$omp end target teams distribute
```

Exercise 1: Hello world with HIP

Build and test run a Hello World C++ code which offloads to GPU via HIP.

- Download the source code
  - `wget https://raw.githubusercontent.com/PDC-support/introduction-to-pdc/master/example/hello_world_gpu.cpp`
- Load the ROCm module and set the accelerator target to amd-gfx90a (AMD MI250X GPU)
  - `ml rocm/5.0.2`
  - `ml craype-accel-amd-gfx90a`
- Compile the code with the AMD hipcc compiler on the login node
  - `hipcc --offload-arch=gfx90a hello_world_gpu.cpp -o hello_world_gpu.x`

Run the code as a batch job

- Edit job_gpu_helloworld.sh to specify the compute project and reservation
- Submit the script with `sbatch job_gpu_helloworld.sh`

with program output written to `output.txt`

```
You can access GPU devices: 0-7
GPU 0: hello world```
...
```

Exercise 2: Dot product with OpenMP

Build and test run a Fortran program that calculates the dot product of vectors.

- Activate the PrgEnv-cray environment `ml PrgEnv-cray`
- Download the source code
    - `wget https://github.com/ENCCS/openmp-gpu/raw/main/content/exercise/ex04/solution/ex04.F90`
- Load the ROCm module and set the accelerator target to amd-gfx90a
    - `ml rocm/5.0.2 craype-accel-amd-gfx90a`
- Compile the code on the login node
    - `ftn -fopenmp ex04.F90 -o ex04.x`

Run the code as a batch job

- Edit job_gpu_ex04.sh to specify the compute project and reservation
- Submit the script with `sbatch job_gpu_ex04.sh`
- with program output `The sum is:  1.25` written to `output.txt`

Optionally, test the code in interactive session.

- First queue to get one GPU node reserved for 10 minutes
  - `salloc -N 1 -t 0:10:00 -A <project name> -p gpu`
- wait for a node, then run the program `srun -n 1 ./ex04.x`
- with program output to standard out `The sum is:  1.25`

- Alternatively, login to the reserved GPU node (here nid002792) `ssh nid002792`.
- Load ROCm, activate verbose runtime information, and run the program
  - `ml rocm/5.0.2`
  - `export CRAY_ACC_DEBUG=3`
  - `./ex04.x`
- with program output to standard out

```
ACC: Version 5.0 of HIP already initialized, runtime version 50013601
ACC: Get Device 0
...
...
ACC: End transfer (to acc 0 bytes, to host 4 bytes)
ACC:
The sum is:  1.25
ACC: __tgt_unregister_lib
```