Installing and Developing GPU Software on Setonix

Setonix Phase 2 Migration

May 2023, Version 1.00

Focus for this Training

Target Audience: All Topaz users

Learning Outcomes:

- Learn about GPU programming technologies, such as ROCm, HIP
- Learn about GPU offloading technologies, such as OpenMP and OpenACC
- Be familiar with best practises for installing GPU-enabled software on Setonix
- Be aware of GPU development tools available on Setonix, including compilers, debuggers, and profilers

Section 1

GPU Programming Technologies

The ROCm platform

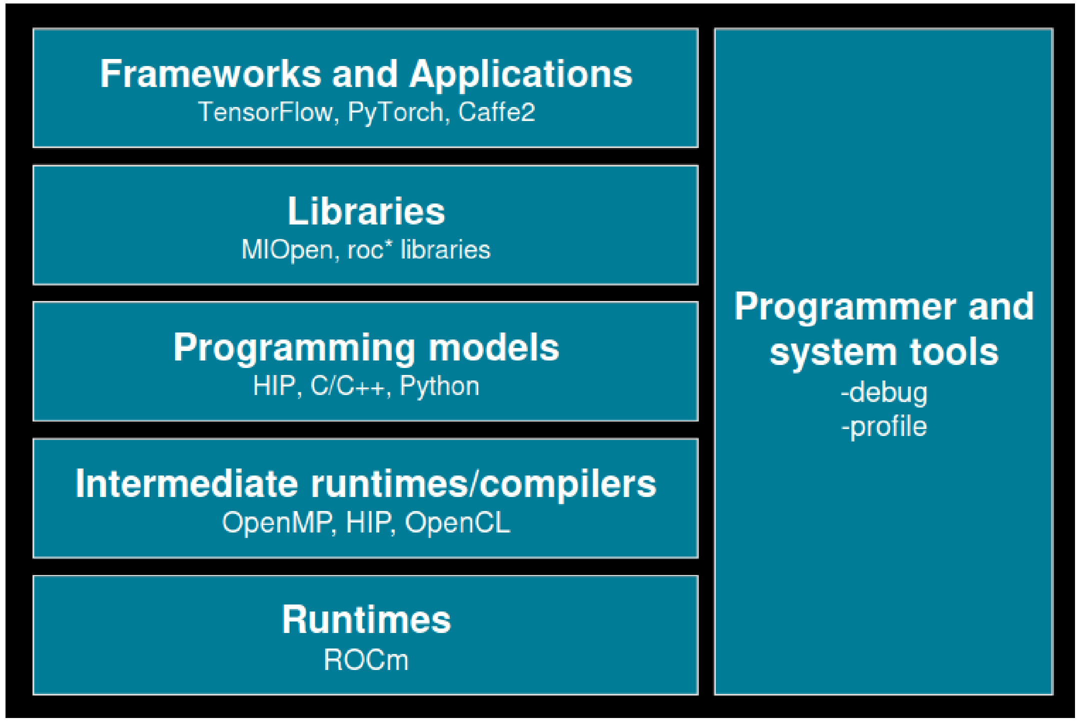
ROCm is the GPU software development and runtime environment provided by AMD. On NVIDIA GPUs, CUDA is equivalent.

On Setonix, ROCm is available through the module **`rocm/5.0.2`**. Once loaded, here is what you will have available for use:

- HIP, a C++ language extension and API calls equivalent to CUDA. And the **`hipcc`** compiler matching **`nvcc`**
- GPU-enabled numerical libraries, such as **`rocFFT`**, **`rocBLAS`** and **`rocRAND`**
- Debugging and profiling tools: **`rocgdb`**, **`rocprof`**, **`roctracer`**
- HIPIFY tools - **`hipify-perl`** and **`hipify-clang`** - to help port CUDA codes to HIP

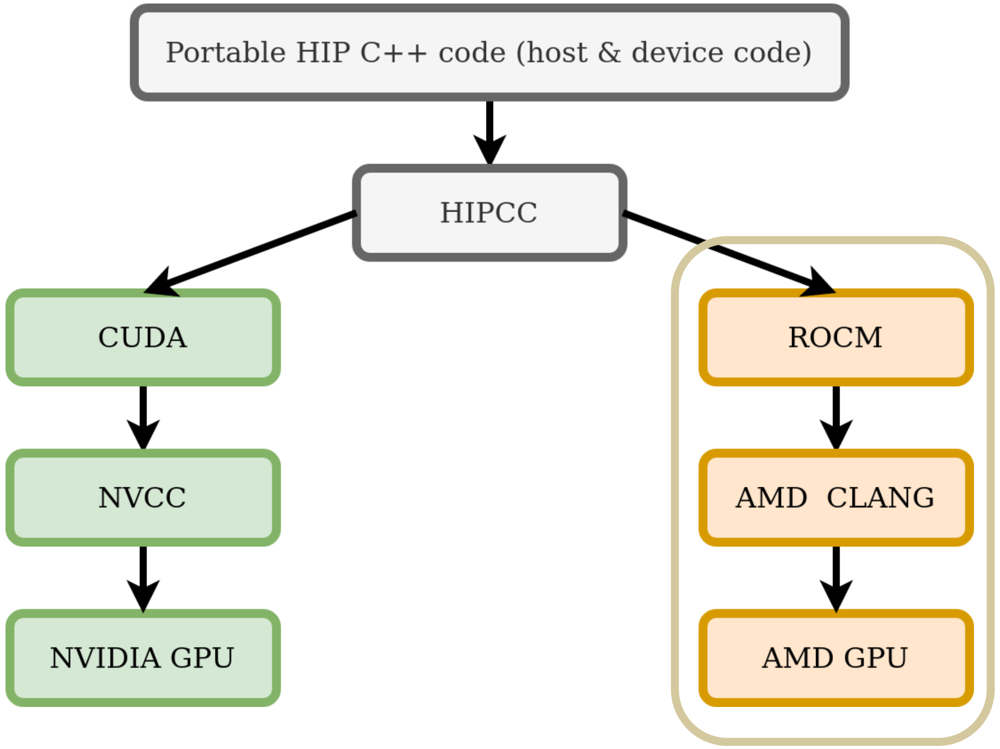
Additionally, there is an experimental installation of **`rocm/5.4.3`**. If you encounter an issue, submit a ticket to the Pawsey Help Desk.

A schematic representation of what ROCm provides.


Frameworks and Applications
TensorFlow, PyTorch, Caffe2
Libraries
MIOpen, roc* libraries
Programming models
HIP, C/C++, Python
Intermediate runtimes/compilers
OpenMP, HIP, OpenCL
Runtimes
ROCm
Programmer and system tools
-debug
-profile

The HIP Programming Language

- HIP (Heterogeneous Interface for Portability) is developed by AMD to program on AMD GPUs. It is a C++ language extension and runtime API similar to CUDA.
- Code written in HIP can be compiled with the **`hipcc`** compiler driver and run on either the NVIDIA or AMD hardware environment.
    - HIP provides the HIPIFY tool to translate CUDA code to HIP. Although it does not work in all instances, it does provide a good starting point.
- On AMD platforms, an AMD fork of LLVM is used. On NVIDIA platforms, HIP is a wrapper around CUDA.
- For more information see: https://support.pawsey.org.au/documentation/display/US/HIP



OpenMP

OpenMP is an API designed to provide parallelism on shared memory systems and for offloading compute to accelerators like GPUs. OpenMP API supports C/C++ and Fortran programming languages.

For more information, see:

- https://support.pawsey.org.au/documentation/display/US/OpenMP

- CPU parallelism is supported by all Cray Programming Environments (CPE), including the **`hipcc`** compiler provided by the **`rocm/5.0.2`** module.
- GPU offloading is currently available through
    - C/C++ **`hipcc`** provided by **`rocm/5.0.2`**
    - **`cc/CC/ftn`** compilers provided by the **`PrgEnv-cray`** CPE
    - C/C++ **`hipcc`** provided by **`rocm/5.4.3`** is currently NOT supported

OpenACC

OpenACC is an API parallel computing programming standard for open accelerators facilitating parallel programming of heterogeneous CPU/GPU systems. OpenACC API supports C/C++ and Fortran programming languages.

For more information, see:

- https://www.openacc.org/sites/default/files/inline-files/API%20Guide%202.7.pdf

GPU offloading is currently available through

- Fortran **`ftn`** compiler provided by the **`PrgEnv-cray`** CPE
- GNU compilers on Setonix do not support OpenACC offloading.

Section 2

Installing GPU-Enabled Software on Setonix

GPU-Enabled Software with Spack

- Pawsey recommends using Spack via the **`spack/0.17.0`** module to install CPU software. For more information, see:
  - Spack: https://support.pawsey.org.au/documentation/display/US/Spack
- However, installing GPU accelerated software with the current deployment of Spack is not well supported.
  - Pawsey is progressing with work on the next software stack deployment with a newer version of Spack that will enable users to GPU-enabled software in the future.

GPU-Enabled Containers-as-Modules

- Pawsey has provided some of these containers-as-modules on Setonix. Currently, most of the AMD GPU-enabled software is provided through containers-as-modules.

```
$ module avail gfx90a
------------- /software/setonix/current/containers/views/modules -------------
   cp2k-amd-gfx90a/8.2                        namd-amd-gfx90a/2.15a2-20211101
   gromacs-amd-gfx90a/2022.3.amd1_174         namd3-amd-gfx90a/3.0a9
   lammps-amd-gfx90a/2022.5.04_130
```

- However, these containers are still in early testing and are not recommended for production use yet.
  - Containers can currently scale only up to 1 node.
  - Some containers can currently use only a single GPU.
- Additional Containers-as-modules can be installed using SHPC through the **shpc** module.

```
$ module load shpc/0.0.57
$ shpc install nvcr.io/hpc/gromacs #install a gromacs container-as-module
```

For more information on SHPC, see:

- https://support.pawsey.org.au/documentation/pages/viewpage.action?pageId=116131369

GPU-Enabled Containers

- The AMD Infinity Hub has a collection of GPU containers for scientific software, AI and Machine Learning applications. Popular software containers available are OpenFOAM, Gromacs, LAMMPS, NAMD, and AMBER.
  - https://www.amd.com/en/technologies/infinity-hub
- The E4S initiative also provides containerised build tools and application stacks for HPC:
  - https://e4s-project.github.io
- These can be downloaded and used via the Singularity container engine:

```
$ module load singularity/3.8.6-nompi
$ singularity pull docker://amdih/gromacs:2022.3.amd1_174
$ srun singularity exec gromacs_2022.3.amd1_174.sif <command> <args>
```

- These containers are not yet ready for production runs and have similar caveats to the containers-as-modules.
- For a discussion of containers see:
  - https://support.pawsey.org.au/documentation/display/US/Containers
  - https://support.pawsey.org.au/documentation/display/US/Use+with+Singularity

GPU-Enabled Modules

- Module files follow a naming convention **`<software>-amd-gfx90a`**
- These modules can be loaded on both CPU and GPU nodes but will only work on GPU nodes.

```
$ module avail gfx90a

----- /software/setonix/current/custom/modules/zen3/gcc/12.1.0/custom -----
   nekrs-amd-gfx90a/22.0
```

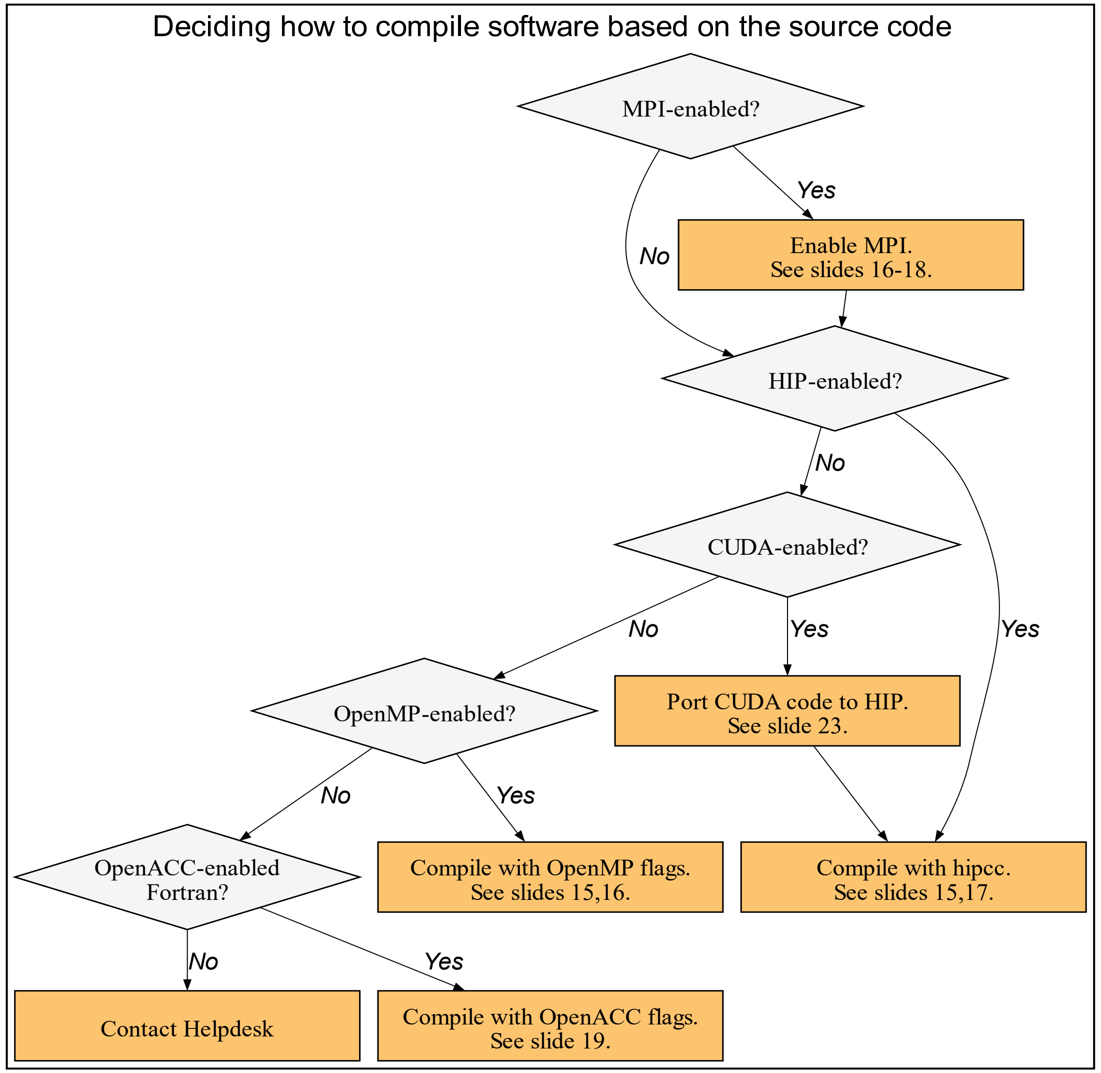
Navigating Manual Installation Process

- There are different approaches for compiling MPI-enabled codes, HIP-enabled codes, and OpenMP-enabled codes.

- CUDA-enabled codes require the additional step of porting to HIP before compilation.

- Not all GPU technologies are fully supported (OpenCL, Sycle, OpenACC for C/C++) but it is possible that workarounds or specific compilation steps might address issues encountered.

- If you encounter issues at any point, contact the Pawsey Help Desk.


Deciding how to compile software based on the source code
MPI-enabled?
Yes
No
Enable MPI. See slides 16-18.
HIP-enabled?
No
Yes
CUDA-enabled?
No
Yes
Port CUDA code to HIP. See slide 23.
OpenMP-enabled?
No
Yes
Compile with OpenMP flags. See slides 15,16.
Compile with hipcc. See slides 15,17.
OpenACC-enabled Fortran?
No
Yes
Contact Helpdesk
Compile with OpenACC flags. See slide 19.

Getting Started with Manual Installation

At early access, Pawsey-provided, GPU-enabled software will be limited. Therefore, we recommend manual builds if possible. First main steps are:

1. Check code/software is HIP-enabled and compatible with AMD GPUs (ROCm), is C/C++/Fortran, and uses OpenMP offloading, or is Fortran and uses OpenACC offloading.

2. Log into a GPU node and load appropriate modules.

```
$ salloc -N1 -p gpu-dev --gres=gpu:1 -A <project>-gpu
$ module swap PrgEnv-gnu PrgEnv-cray # only for directive based offloading
$ module load rocm/<version>
$ module load craype-accel-amd-gfx90a
```

3. Ensure GPU drive is present. By compiling on a GPU node with a GPU, you avoid issues arising from not having the driver for the GPU present. Additionally, any unit tests will pass, which are part of the installation process that may require a GPU to run.

4. Refer to documentation for general guides to compiling software from source: https://support.pawsey.org.au/documentation/display/US/How+to+Manually+Build+Software

Manual Installation of Non-MPI Software with HIP/ROCm

- Pure HIP codes require the use of the **`hipcc`** compiler provided by the **`rocm/<version>`** module. Once on a GPU node with the module loaded, compile the source code.

```
$ hipcc <source_code> -o <executable_name>
```

- OpenMP offloading codes can be compiled on a GPU node with the **`rocm/5.0.2`** module loaded. The compilation process requires extra flags.

```
$ hipcc --offload-arch=gfx90a -fopenmp -fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-
target=amdgcn-amd-amdhsa -march=gfx90a <source_code> -o <executable_name>
```

- The clang++ compiler provided by **`rocm/<version>`** can also compile OpenMP offloading code by using the same flags, though it requires additional flags to include the appropriate **`rocm`** libraries that are added by **`hipcc`**.

Manual Installation of MPI-Enabled Software with Cray Programming Environment Compilers

1. Update environment variables storing paths with the MPI path:

```
$ export PATH=$PATH:${CRAY_MPICH_DIR}/bin
$ export CPATH=$CPATH:${CRAY_MPICH_DIR}/include
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${CRAY_MPICH_DIR}/lib
```

2. Compile HIP-enabled source code, if any, to construct object files:

```
$ hipcc -c <source_code>
```

3. Compile OpenMP source code, if any, with the **`cc/CC/ftn`** CPE Cray compilers (**`PrgEnv-cray`**):

```
$ CC -c -fopenmp <source_code> # C++ code example
$ ftn –c –homp <source_code> # Fortran code example
```

4. Compile the MPI-enabled source with the **cc/CC/ftn** CPE Cray compiler (**`PrgEnv-cray`**) where there may be HIP references (such as in the include):

```
$ CC -c -D__HIP_PLATFORM_AMD__ <source_code> # C++ code example
```

5. Link object files to produce an executable:

```
$ CC <object_files> -o <executable_name>
```

Manual Installation of MPI-Enabled Software with HIP Compiler

1. Update environment variables storing paths with the MPI path:

```
$ export PATH=$PATH:${CRAY_MPICH_DIR}/bin
$ export CPATH=$CPATH:${CRAY_MPICH_DIR}/include
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${CRAY_MPICH_DIR}/lib
```

2. Compile MPI, HIP-enabled source code, if any:

```
$ hipcc -c <source_code>
```

3. Compile MPI, OpenMP source code, if any:

```
$ hipcc --offload-arch=gfx90a -fopenmp -fopenmp-targets=amdgcn-amd-amdhsa -Xopenmp-
target=amdgcn-amd-amdhsa -march=gfx90a -c <source_code>
```

4. Link object files to produce an executable:

```
$ hipcc -lmpi <object_files> -o <executable_name>
```

GPU-GPU MPI-Enabled Software with HIP Compiler

To build software that allows for GPU-to-GPU MPI communication, the steps are similar to building MPI-enabled code with the HIP compiler with the addition of the following:

1. Set the following environment variable when compiling and running:

```
$ export MPICH_GPU_SUPPORT_ENABLED=1
```

2. When linking object files to produce an executable, add the following flags:

```
$ hipcc -L/${CRAY_MPICH_ROOTDIR}/gtl/lib -lmpi_gtl_hsa –lmpi <object_files> -o
<executable_name>
```

Manual Installation of Fortran based OpenACC software

1. Compile OpenACC source code to construct object files.

```
$ ftn -hacc <source_code> <executable_name>
```

The Fortran compiler wrapper will also handle MPI-enabled code by adding the relevant MPI flags.

Manual Installation Linking to GPU-Enabled Libraries

ROCm provides rocFFT (Fourier transforms), rocBLAS (BLAS, linear algebra), rocRAND (random numbers), rocSOLVER (LAPACK) which should be accessed through the HIP wrappers, hipFFT, hipBLAS, hipRAND, hipSOLVER, etc.

1. To compile code, link to the appropriate library. Examples are:

```
$ hipcc -lhipfft <source_code> -o <executable_name> # for FFT
$ hipcc –lhipblas <source_code> -o <executable_name> # for BLAS
$ hipcc –lhipsolver <source_code> -o <executable_name> # for hipSolver
```

2. For CUDA code that used libraries, such as cuFFT, source code will require porting to HIP.
    - An FFT example is where **`cufft_plan_create`** becomes **`hipfft_plan_create`**.

Things to be aware of when compiling from source

Build systems:

CMake

- Cmake >=3.21 has functionality to autodetect HIP/ROCm, but earlier versions do not.

```
$ module load cmake/3.21.4
```

- Source code may require an update to the CMakeLists.txt to make use of the newer functionality to automatically discover and add hooks to the ROCm libraries provided through HIP.

Libraries:

- The default behaviour on the HPE Cray EX systems is to provide math libraries, such as BLAS and LAPACK, through the **`cray-libsci`** module, which is automatically added when using the compiler wrappers **cc/CC/ftn**. This may not be desirable and is in contrast to non-Cray systems. Therefore, we suggest removing this module explicitly before compiling code and running software.

```
$ module unload cray-libsci
```

Section 3

GPU Developer Tools on Setonix

Porting CUDA code with HIPIFY

- The **HIP/ROCm** packages provide a tool to port CUDA code to HIP called HIPIFY. There are two versions:
  - **`hipify-perl`**, a conversion script that will map CUDA calls to the corresponding HIP call.
  - **`hipify-clang`**, a compiler that not only converts the code but also tries compiling it.
- Due to the non-trivial nature of porting codes, we suggest using **`hipify-perl`** to begin the process and then proceed with trying to compile the code with **`hipcc`**.

```
$ module load rocm/<version>
$ hipify-perl <original_cuda_code>.cu > <new_hip_code>.cu
$ hipcc -c <new_hip_code>.cu
```

- It is quite likely the porting will require more modifications to get optimal performance on AMD GPUs and in some instances will be incomplete.
- For porting examples see:
  - https://docs.amd.com/bundle/HIP-Programming-Guide-v5.0/page/Transitioning_from_CUDA_to_HIP.html
- Contact the Pawsey Help Desk in case of questions.

Debuggers & Profilers

ROCgdb

- Command line profiler for AMD GPUs provided by AMD, similar to the CPU debugger, gdb
- Provided by **`rocm/<version>`** modules
- For more information, see:
  - https://docs.amd.com/bundle/ROCDubugger-User-Guide/page/index.html

ROCprof

- Command line profiling tools for AMD GPUs
- Provided by **`rocm/<version>`** modules
- For more information, see: https://docs.amd.com/bundle/ROCm-Profiling-Tools-User-Guide-v5.3/page/Introduction_to_ROCm_Profiling_Tools_User_Guide.html

Notes on Other Debugging Tools

- Perfetto (https://ui.perfetto.dev) is a useful tool for visualising profiling data produced by ROCprof.

- ArmFORGE DDT, a powerful MPI-enabled debugger, does not support AMD GPUs but can be used for parallel CPU-based debugging.

- Cray provides several tools for debugging (Cray CCDB provided by **`cray-ccdb/<version>`**) and profiling (CrayPAT and MAP provided by **`perftools/<version>`**), which support MPI. It currently does not fully support AMD GPUs but has plans to do so in the future.

- Currently, there is nothing equivalent to NVIDIA-Nsight, but software is being developed to provide similar functionality for AMD GPUs.

- We are working to provide additional developer tools, such as Omnitrace and Omniperf.

How do I get help?

Migration Documentation & Migration Guides

- Setonix GPU Partition Quick Start
- Setonix Migration Guide
- Setonix User Guide
- Supercomputing Documentation

Migration Training Materials & Video Recordings

- Upcoming Migration Training
- Recordings: Pawsey YouTube Setonix Migration Phase 2 Playlist
- Materials: Setonix Migration Training Materials (PDFs)

Help Desk

- Help Desk
- Email: help@pawsey.org.au

Thank you for attending!

Please complete this short survey:

https://www.surveymonkey.com/r/PLLNSQT