Early Experiences Evaluating the HPE/Cray Ecosystem for AMD GPUs

Verónica G. Vergara Larrea, Reuben D. Budiardja, Wayne Joubert
National Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, TN, USA
Email: {vergaravg,reubendb,joubert}@ornl.gov

***Abstract*—Since deploying the Titan supercomputer in 2012, the Oak Ridge Leadership Computing Facility (OLCF) has continued to support and promote GPU-accelerated computing among its user community. Summit, the flagship system at the OLCF — currently number 2 in the most recent TOP500 list — has a theoretical peak performance of approximately 200 petaflops. Because the majority of Summits computational power comes from its 27,972 GPUs, users must port their applications to one of the supported programming models in order to make efficient use of the system. Looking ahead to Frontier, the OLCF's exascale supercomputer, users will need to adapt to an entirely new ecosystem which will include new hardware and software technologies. First, users will need to familiarize themselves with the AMD Radeon GPU architecture. Furthermore, users who have been previously relying on CUDA will need to transition to the Heterogeneous-Computing Interface for Portability (HIP) or one of the other supported programming models (e.g., OpenMP, OpenACC). In this work, we describe our initial experiences in porting three applications or proxy apps currently running on Summit to the HPE/Cray ecosystem to leverage the compute power from AMD GPUs: minisweep, GenASiS, and Sparkler. Each one is representative of current production workloads utilized at the OLCF, different programming languages, and different programming models. We also share lessons learned from challenges encountered during the porting process and provide preliminary results from our evaluation of the HPE/Cray Programming Environment and the AMD software stack using these key OLCF applications.**

***Keywords*-GPU accelerated computing; HIP; CUDA; OpenMP**

I. Introduction

Since deploying the Titan supercomputer in 2012, the Oak Ridge Leadership Computing Facility (OLCF) has continued to support and promote GPU-accelerated computing among its user community. Summit, the flagship system at the OLCF and currently number 2 in the most recent TOP500 list, has a theoretical peak performance of approximately 200 petaflops. Because the majority of Summits computational power comes from its 27,972 GPUs, users must port their applications to one of the supported programming models in order to make efficient use of the system.

Looking ahead to Frontier, the OLCFs exascale supercomputer, users will need to adapt to an entirely new ecosystem which will include new hardware and software technologies. First, users will need to familiarize themselves with a new GPU architecture, the AMD Radeon GPU. In addition, users that have been relying on NVIDIA's CUDA API on OLCFs Titan and Summit will need to transition to AMD's Heterogeneous-Computing Interface for Portability (HIP) or another supported programming model (e.g., OpenMP with target offload, OpenACC, Kokkos). To facilitate this transition, the OLCF is working closely with HPE/Cray and AMD to evaluate the programming environment for AMD accelerators. This includes evaluation of new compilers and tools as well as math libraries and supporting packages.

In this work, we describe our initial experiences in porting three proxy applications, currently running on Summit, to the HPE/Cray ecosystem to leverage the compute power from AMD GPUs. These applications include Minisweep [1], GenASiS [2], and Sparkler [3]. Minisweep is a mini-application developed to simulate the computational patterns used in the Denovo radiation transport application. It is an MPI application written in C++ and has been ported to several programming models and interfaces including: CUDA, OpenMP (threaded and target offload), and OpenACC. GenASiS (*Gen*eral *A*strophysics *Si*mulation *S*ystem) is a developing multi-physics code for large-scale simulations of astrophysical phenomena. Its example problem in GenASiS Basics has been used to experiment with different programming models includingOpenMP (threaded and target offload), OpenACC, and CUDA. Sparkler is a mini-application developed to mimic the computation performed by the CoMet comparative genomics application [4]. Sparkler is an MPI application written in C++ that relies on CUDA and cuBLAS. These applications are representative of current production workloads utilized at the OLCF and span different programming languages as well as programming models.

In this work, we also share lessons learned from challenges encountered during the porting process. We provide preliminary results from our evaluation of the HPE/Cray Programming Environment and the AMD software stack using these key OLCF applications. The results presented will be useful for the user community working to prepare their applications for the arrival of the Frontier supercomputer.

II. Related work

When the OLCF deployed the Summit supercomputer, we started offering a mechanism for our users to take advantage of the OpenMP offload capabilities available in the OpenMP 4.5 specification. As a result, we began efforts to port and evaluate key kernels that would help us gather lessons learned that we could share with the user community. In [5], we ported two of the same codes used here, minisweep and GenASiS to a programming model new to the codes to compare results between OpenACC and OpenMP offload.

Several efforts to port applications to HIP are in active development as is the case for the Ginkgo linear algebra package [6].

III. Experimental Setup

For our experiments, we used the OLCF's flagship system, Summit, and one of the early access systems deployed in preparation for the arrival of Frontier [7], Spock.

A. Summit

Summit, deployed in 2018, is currently the number two fastest supercomputer in the world according to the most recent TOP500 list (Nov 2020) [8]. Summit is an IBM AC922 supercomputer comprised of 4,662 compute nodes, an increase of 54 nodes since it was first put in production.

Each compute node has two 22-core POWER9 CPUs and six NVIDIA Tesla V100 GPUs. The GPUs on the node are interconnected via NVLink 2.0 which provides 50GB/s of peak bandwidth between devices. As shown in Figure 1, each POWER9 CPU is directly connected via NVLink 2.0 to three of the six GPUs.

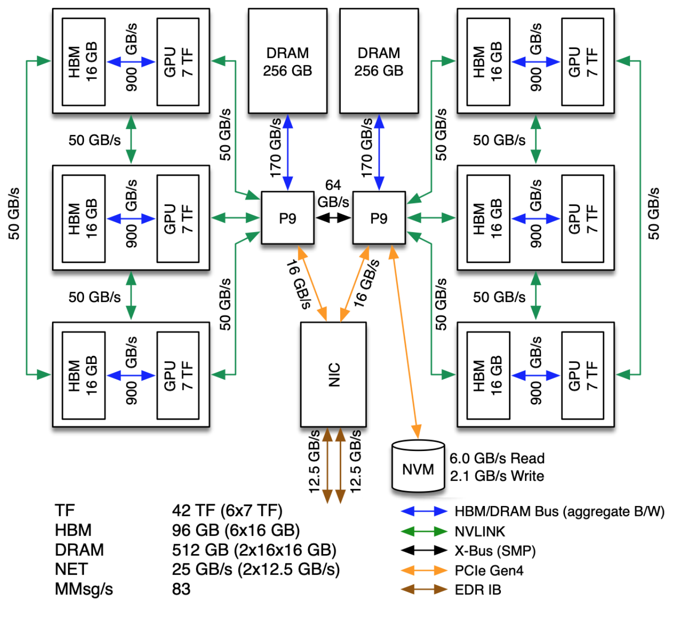


Figure 1. Summit node architecture.

Summit is equipped with the IBM HPC Software Stack which includes IBM Spectrum MPI as the MPI implementation, the IBM XL compiler, and IBM ESSL mathematics library. OLCF also provides the GNU Compiler Collection (GCC) compiler, the PGI compiler, the LLVM compiler, and the NVIDIA HPC Software Development Kit (NVHPC SDK) and corresponding toolchains with each one.

We used the following versions of these packages to collect the results presented in this work:

- IBM XL 16.1.1 PTF5 compiler
- IBM Spectrum MPI 10.3.1.2
- NVIDIA CUDA 10.1.243

B. Spock

Spock is a 36-node Cray EX supercomputer deployed at the OLCF to provide users with a platform similar in architecture to the upcoming exascale supercomputer, Frontier [7]. Each compute node on Spock has a single 64-core AMD EPYC 7662 Rome CPU with 256GB of DDR4 memory and four AMD MI100 GPUs with 32GB HMB2 memory per device. As shown in Figure 2, the GPUs are interconnected via AMD's Infinity Fabric technology which provides 46GB/s of peak bandwidth (unidirectional).

Spock compute nodes are connected with HPE's Slingshot 10 interconnect technology [9].

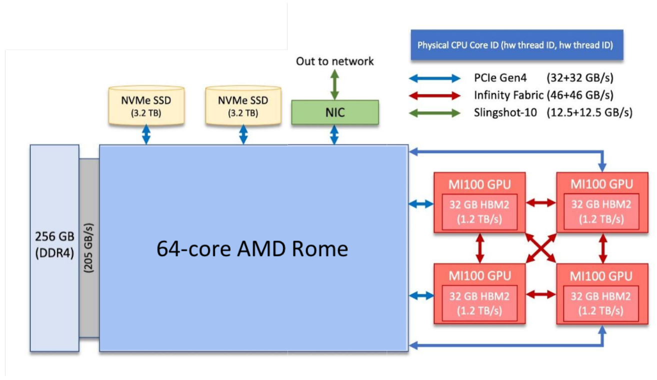


Figure 2. Spock node architecture.

Being an HPE/Cray system, Spock is equipped with the HPE/Cray Programming Environment (CPE) which, for this platform, includes support for two compilers: the Cray Compiling Environment (CCE), and the GNU Compiler Collection (GCC). The CPE also includes Cray MPICH as the MPI implementation that is optimized for the system.

In addition, AMD's Radeon Open Compute (ROCm) stack is available and includes the HIP programming framework as well as ROCm libraries such as rocBLAS, hipBLAS, rocFFT, RCCL, among others.

The OLCF also provides LLVM and the AMD AOMP compilers to complement the other compilers already available in CPE.

We used these specific packages to collect results on Spock for this paper:

- HPE/Cray Programming Environment 8.0.0 (CPE 21.04)
- Cray MPICH 8.1.4
- CCE 11.0.4
- ROCm 4.1.0
- AOMP 13

Table I provides a side-by-side comparison of the main differences between Spock and the previously described Summit supercomputer.

IV. Experimental Results

A. GenASiS

GenASiS uses OpenMP target offload (available since OpenMP 4.5 specifications) as its primary mechanism to exploit GPUs for its computation kernels [10]. The majority of its device memory management functionalities (allocation, data movement, mapping) are also implemented using OpenMP API, except for a small number of functionalities that are not yet implemented by current OpenMP compilers. These include allocating to the host page-locked (pinned) memory and getting device memory info.

Porting GenASiS to AMD GPU is therefore a relatively straightforward experience. For OpenMP offloaded kernels, technically there should not be any changes needed.

However, it turns out there are minor adjustments needed due to the way CCE Fortran compiler maps OpenMP construct to GPU. While IBM XL compiler maps `!$OMP parallel do` to Nvidia GPU warp and threads, CCE only maps `!$OMP simd` to AMD GPU's analogue (i.e. wavefront and work-item). The simplest solution is to add the `simd` construct for every `parallel do` construct.

Perhaps a more portable solution is to use OpenMP 5 metadirective feature to generate the appropriate construct based on compiler family. However, as of this writing, this feature is not yet widely implemented. As such, we fall back to use preprocessor macro to implement this change.

Certain kernels use `reduction` clause within its `target` region. While IBM XL compiler implements OpenMP 5 standard that automatically maps reduction variables as `tofrom`, the CCE compiler has yet implemented this (as of this writing). Adding the mapping explicitly was needed for the code to run correctly, at the cost of reduced portability with CPU multi-threading version (i.e. non-offload) of the code. Again in this case we fall back to employ preprocessor.

For this work, we use an example problem `RiemannProblem` in GenASiS Basics [11]. The Basics division of GenASiS implements the utilitarian functionality required for most large-scale physics simulations. They include physical units and constants, message passing, I/O, runtime parameter management, and exploitation of hardware accelerators. These classes are used in higher-level GenASiS divisions: Mathematics [12] and Physics. Example problems implemented in GenASiS Basics use these facilities with a more simplified distributed mesh and solvers. These solvers mimic the more sophisticated ones in higher level (Mathematics and Physics), allowing these example problems to be used as proxy applications.

For earlier performance study, we have also ported the kernels in GenASiS Basics to CUDA. This port allows us to compare CUDA and OpenMP performance on Summit. Porting these CUDA kernels to HIP is very straightforward. Other changing CUDA library calls to the corresponding HIP calls, the only other change is how to execute subroutine for kernel launch. CUDA uses the "triple-chevron" `<<< ... >>>` syntax while HIP initially used `hipLaunchKernelGGL()` function call. Since then, support for the "triple-chevron" has been added.

In some kernels, we have inadvertently relied on Summit unified shared memory feature for small arrays that are needed on the GPUs. Because this feature does not yet exist on Spock, we encountered page fault and crash for these kernels. The solution is to copy these arrays to device memory prior to kernel launch.

Figure 3 shows timings of kernels in GenASiS Basics `RiemannProblem`. The timings for OpenMP offload and CUDA/HIP versions of the kernels are shown. On Summit, we use the IBM XL compiler version 16 and NVIDIA CUDA compiler version 10 `nvcc`, respectively. On Spock, we use CCE compiler version 11 and ROCm compiler version 4.1.0 for OpenMP offload and HIP, respectively.

On Figure 3 we observe that for most kernels OpenMP and CUDA or HIP versions achieve near performance parity. For the `Difference` and `Fluxes` kernels, CCE OpenMP is noticeably slower than the HIP version. On Summit, this is correspondingly observed on `Fluxes` kernel only. On the other hand, we observe that CCE OpenMP perform better for `ApplyBoundaryConditions` than HIP version of the kernel. These differences in performance speak to optimization that can be in the compilers.

B. *Minisweep*

Minisweep is a miniapplication that models the performance characteritics of the Denovo S_n radiation transport code, part of the Exnihilo package[1], [13]. It is written in C and supports multiple programming models including OpenMP 3.1, CUDA, OpenACC and OpenMP offload.

In the present work we port Minisweep to HIP and evaluate on AMD GPUs. By construction, Minisweep attempts to isolate into individual source code files the parts of the code that are specific to a given programming model. In particular, this includes code for memory and thread management, transfers to and from the GPU, and kernel launch. Because of this, we did not use the AMD `hipify-perl` tool to

Table I
SIDE-BY-SIDE COMPARISON BETWEEN SPOCK AND SUMMIT.

Component	Spock	Summit
Number of nodes	36	4,662
Network	Slingshot 10	InfiniBand
Filesystem	GPFS	GPFS
Processors per Node	1 x 64-core AMD Rome CPU	2 x 22-core IBM POWER9 CPU
Memory per Node	256GB DDR4	256GB DDR4
Accelerators per Node	4 x AMD MI100 GPUs	6 x NVIDIA V100 GPUs
GPU-GPU Bandwidth	46+46 GB/s InfinityFabric	50 GB/s NVLink2
CPU-GPU Bandwidth	32+32 GB/s PCIe Gen4	50 GB/s NVLink2
Memory Bandwidth	205 GB/s DDR4	170 GB/s DDR4
Accelerator Memory	32GB HBM2	16GB HMB
Accelerator Memory Bandwidth	1.2 TB/s HMB2	900 GB/s HBM
Accelerator Programming Models	HIP / ROCm, OpenMP offload	CUDA, OpenMP offload, OpenACC

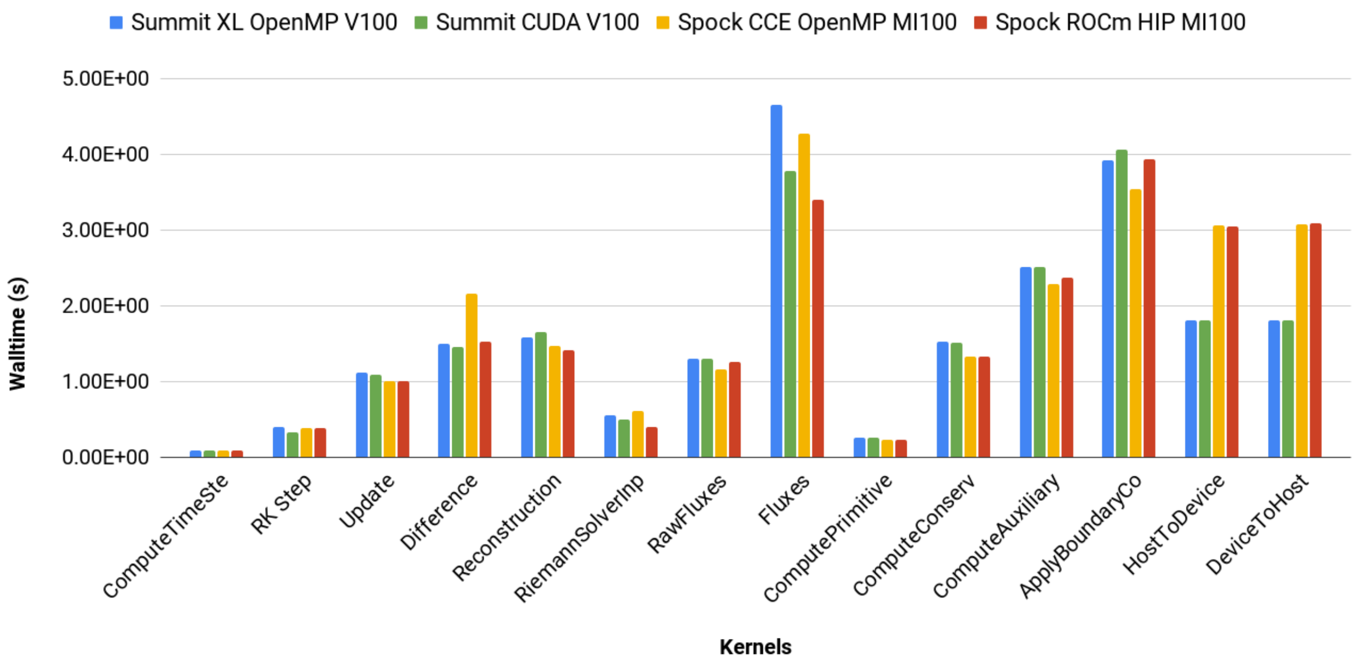


Figure 3. Kernel timings for GENASIS BASICS `RiemannProblem` with 256^3 cells per GPU evolved for 50 timesteps. The bars represent the timing with IBM XL compiler for OpenMP on Summit (blue), NVIDIA nvcc compiler for CUDA on Summit (green), CCE compiler for OpenMP on Spock (yellow), and ROCm compiler for HIP on Spock (red).

translate CUDA to HIP automatically, but instead translated the code manually. Owing to the near-isomorphism between the CUDA and HIP APIs, the the hand translation is straightforward and makes use of `ifdefs` to specify the programming model option at compile time. A few differences required specific attention, such as `__CUDA_ARCH__` vs. `__HIP_DEVICE_COMPILE__` and the kernel launch syntax. An early version of `HIPConfig.cmake` was installed to support the CMake configuration system. Overall, the porting process was straightforward and required less than two days of effort.

Preliminary performance results are shown in Figure 4. For this test, the code is run on one rank using one GPU, and the number of gridcells is varied. Performance rates are shown for one NVIDIA V100 GPU on Summit and one AMD MI100 GPU on Spock. The maximum performance of this code on either architecture is roughly 5% of double precision floating point peak, which is very typical for this algorithm insofar as it is inherently memory bandwidth limited. Performance on the MI100 and the V100 GPUs are qualitatively similar, with better performance for very large problem sizes, for which various overheads and latencies are amortized. The larger memory of the MI100 GPU allows for larger problems to be run, which further improves efficiencies. The raw performance of the V100 GPU for this problem is higher than the MI100 GPU. This is not surprising insofar as the Minisweep implementation has been heavily tuned to the NVIDIA architecture, and for these preliminary tests the code has not been significantly re-tuned for the AMD architecture. It is expected that further tuning effort in the future will improve performance for the MI100 GPU case.

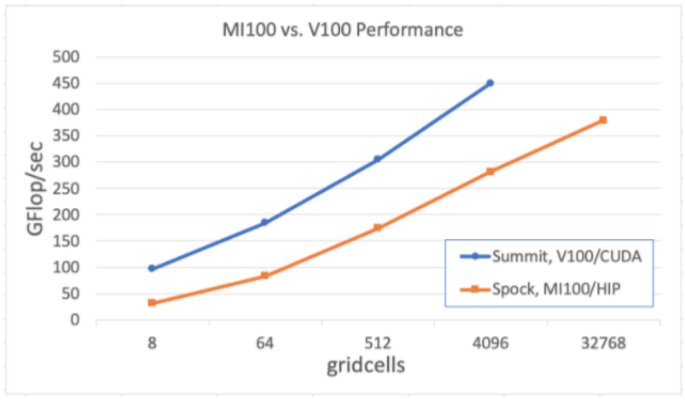


Figure 4. Minisweep performance results

C. Sparkler

Sparkler is a mini-application developed at ORNL to mimic the computations executed by the CoMet computational genomics code [4], [14]. Sparkler computes dense matrix-matrix multiplication for small integer elements. In particular, Sparkler computes the Custom Correlation Coefficient (CCC) metric. Like CoMet, Sparkler is written in C++ and uses MPI and CUDA.

Sparkler was also featured in 2019 as the mystery application for the Student Cluster Competition at the 2019 International Conference for High Performance Computing, Networking, Storage, and Analysis (SC19) held in Denver, CO [15].

Given that Sparkler already used CUDA, we used the `hipify-perl` script provided as part of the AMD ROCm stack to do the initial port. As shown in Listing 1, in three cases, an exact translation from CUDA to HIP was not available. For that reason, `CUBLAS_TENSOR_OP_MATH` and `cublasSetMathMode` were removed, and `CUBLAS_GEMM_ALGO4_TENSOR_OP` was replaced with `HIPBLAS_GEMM_DEFAULT`.

With those changes, we were able to successfully compile the HIP version of Sparkler on Spock using ROCm 4.1.0's `hipcc` and linking explicitly to the Cray MPICH libraries.

We then conducted experiments on a single node of Summit and a single node of Spock to compare the performance of both implementations. The first problem size chosen used 4,000 vectors with 90,000 fields and, as shown in Figure 5, we were able to obtain nearly identical performance on a single NVIDIA V100 GPU and a single AMD MI100 GPU. As we increased the number of processes to fully utilize a Spock node with 4 GPUs, we noticed that the performance obtained on Summit was approximately 21% higher than on the AMD platform.

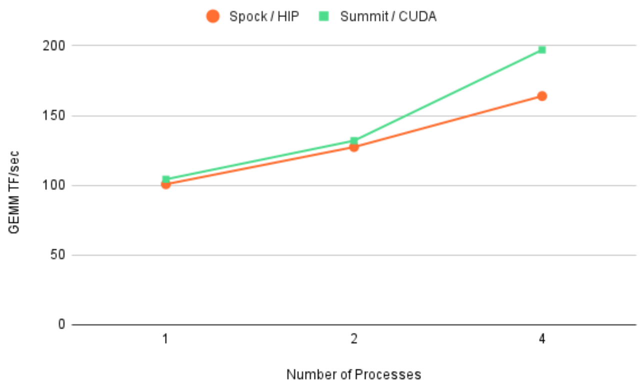


Figure 5. Sparkler performance on Summit and Spock using 4,000 vectors and 90,000 fields over 400 iterations.

A similar experiment conducted using 15,000 vectors demonstrated that on larger problems the overheads can be amortized and we can obtain better performance on Spock as shown in Figure 6.

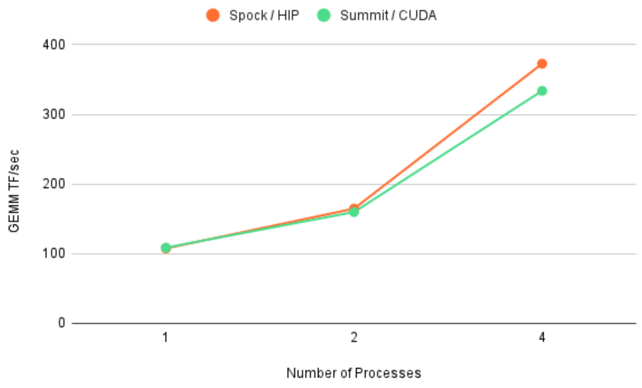
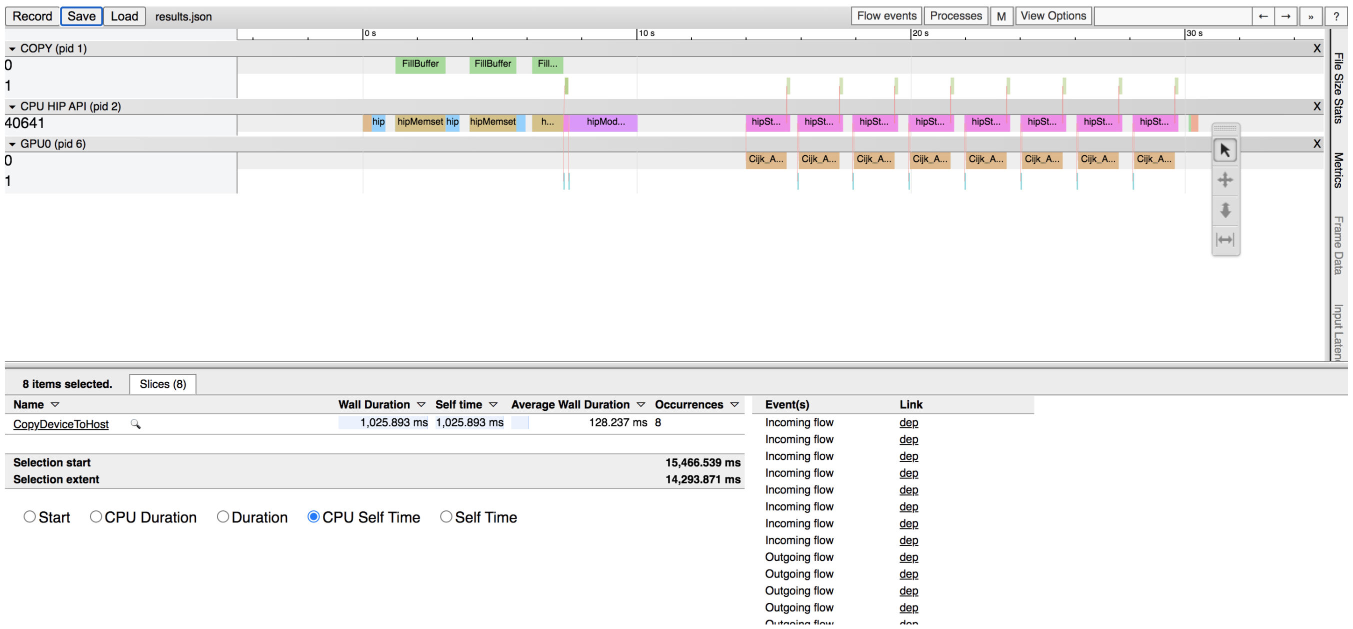


Figure 6. Sparkler performance on Summit and Spock using 15,000 vectors and 90,000 fields over 400 iterations.

To better understand the performance difference, we used `rocprof` to profile the code using a 8 iterations instead of 400. Unfortunately, we encountered runtime issues with the tool that prevented us from generating a trace using ROCm 4.1.0. The issues were due to a bug in the Spack [16] package provided for 'rocprof' and 'roctracer' when installed to a non-default location. These issues were corrected in ROCm 4.2.0, and once available on Spock, we were able to capture a profile and trace using `rocprof --timestamp on --hip-trace --stats ./sparkler`. The resulting JSON file can be displayed using Google Chrome's tracing tool (i.e., `chrome://tracing`) as shown in Figure 7. The trace shows nine different data transfers, one from the host to the device at the beginning of the application and eight transfers from the device to the host after every iteration.

```
$ hipify-perl main.cu
  warning: main.cu:398: unsupported identifier "CUBLAS_GEMM_ALGO4_TENSOR_OP"
  warning: main.cu:482: unsupported identifier "cublasSetMathMode"
  warning: main.cu:483: deprecated identifier "CUBLAS_TENSOR_OP_MATH" since CUDA 11.0
  warning: main.cu:483: unsupported identifier "CUBLAS_TENSOR_OP_MATH"
```

Listing 1. Sample output from running `hipify-perl` on Sparkler's `main.cu` file.



8 items selected. Slices (8)

Name	Wall Duration	Self time	Average Wall Duration	Occurrences
CopyDeviceToHost	1,025.893 ms	1,025.893 ms	128.237 ms	8

Selection start	15,466.539 ms
Selection extent	14,293.871 ms

Start CPU Duration Duration [✓ CPU Self Time] Self Time

Event(s)	Link
Incoming flow	dep
Incoming flow	dep
Incoming flow	dep
Incoming flow	dep
Incoming flow	dep
Incoming flow	dep
Incoming flow	dep
Incoming flow	dep
Outgoing flow	dep
Outgoing flow	dep
Outgoing flow	dep
Outgoing flow	dep
Outgoing flow	dep

Figure 7. Sparkler `rocprof` profile on Spock using 15,000 vectors and 90,000 fields over 8 iterations.

The version of `rocprof` available on Spock does not yet combine traces from ranks on different nodes. However, since we are running all processes on the same node for the Sparkler experiments, we are able to use the method recommended at [17] to profile the MPI job. Further improvements to this and the tracing tools are expected in future versions of ROCm that will allow users to more easily study their multi-node applications.

V. Lessons Learned

During this work, we encountered several issues that we think would be relevant to others currently porting applications to an AMD-based Cray EX supercomputer. In this section, we include a selection of these.

- As of this writing, the HPE/Cray PE does not include the ROCm stack and as a result, we have to install it on our facility maintained software stack. This process has not being as straightforward as we had hoped. We currently use Spack packages developed by AMD to install ROCm [18] on an NFS area. However, these installs all components in their own prefixes when ROCm and its tools expect all to be in the default location (i.e., `/opt/rocm-<version>`). These issues are currently being investigated by AMD and we expect updates to the Spack packages will eventually address them.
- The HPE/Cray PE 21.04, currently provides toolchains for CCE and GCC. However, other compilers like AOMP is in development.
- As identified by GenASiS the AMD `flang` compiler does not yet provide the require features needed by OpenMP codes.
- As seen in this work, while tools like `rocprof` can already be useful, because they are in active development, they are not as robust and fully featured as those available on NVIDIA platforms. Future releases of the tools will continue adding features and will provide support for future devices.
- Because ROCm is in active development, releases of new versions are frequent. At the time of this writing, the HPE/Cray PE currently does not yet support the latest ROCm release 4.2.0.
- As noted in the Sparkler experiments, SLURM binding and mapping for GPUs is a relatively new feature which

will require user education and also close collaboration with the SchedMD team.

- In this work, we found that porting codes that already support CUDA to be a simple and straightforward process. However, unsurprisingly, to get the most out of the new architecture additional tuning would be required.

VI. Conclusion

In this paper, we describe our first experiences porting key application kernels or mini-applications to a platform equipped with AMD GPUs using the HPE/Cray programming environment and the AMD ROCm stack.

Our contribution includes porting the three applications, GenASiS , Minisweep, and Sparkler, to HIP and evaluating their performance on the latest AMD GPU device available to the public.

As discussed here, porting the selected CUDA-based codes to Spock was fairly straightforward. Due to the fact that AMD ROCm is a younger platform than CUDA, we found a few cases in which exact matches were not available. However, we were able to successfully execute the three codes studied on the AMD MI100 GPUs. Although we were able to obtain comparable performance "out-of-the-box" for specific problem sizes, additional optimizations would be needed in order to fully utilize the compute power of the MI100 GPU.

Furthermore, as was shown in the GenASiS case, the AMD compiler, which is in active development, does not yet provide the same level of OpenMP support as more mature compilers like CCE or GCC.

The early experiences presented here could be useful for users that are starting to port codes to AMD accelerators.

Acknowledgment

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

References

[1] O. B. Messer, E. DAzevedo, J. Hill, W. Joubert, M. Berrill, and C. Zimmer, "Miniapps derived from production hpc applications using multiple programing models," The International Journal of High Performance Computing Applications, vol. 32, no. 4, pp. 582–593, 2018. [Online]. Available: https://doi.org/10.1177/1094342016668241

[2] C. Y. Cardall, R. D. Budiardja, E. Endeve, and A. Mezzacappa, "GENASIS: GENERAL ASTROPHYSICAL SIMULATION SYSTEM. i. REFINABLE MESH AND NONRELATIVISTIC HYDRODYNAMICS," The Astrophysical Journal Supplement Series, vol. 210, no. 2, p. 17, jan 2014. [Online]. Available: https://doi.org/10.1088%2F0067-0049%2F210%2F2%2F17

[3] "Sparkler," https://github.com/wdj/sparkler, 2019.

[4] W. Joubert, D. Weighill, D. Kainer, S. Climer, A. Justice, K. Fagnan, and D. Jacobson, "Attacking the opioid epidemic: Determining the epistatic and pleiotropic genetic architectures for chronic pain and opioid addiction," in SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, 2018, pp. 717–730.

[5] V. Melesse Vergara, R. Budiardja, R. Gayatri, C. Daley, O. R. Hernandez, and W. Joubert, "Experiences porting mini-applications to OpenACC and OpenMP on heterogeneous systems," 2019.

[6] Y. M. Tsai, T. Cojean, T. Ribizel, and H. Anzt, "Preparing ginkgo for amd gpus – a testimonial on porting cuda code to hip," 2020.

[7] "Frontier: Direction of discovery." https://www.olcf.ornl.gov/frontier/.

[8] "TOP500 Supercomputer Sites," https://www.top500.org.

[9] "Spock quickstart guide," https://docs.olcf.ornl.gov/systems/spock_quick_start_guide.html, 2021.

[10] R. D. Budiardja and C. Y. Cardall, "Targeting gpus with openmp directives on summit: A simple and effective fortran experience," Parallel Computing, vol. 88, p. 102544, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819119301358

[11] ——, "Genasis basics: Object-oriented utilitarian functionality for large-scale physics simulations (version 3)," Computer Physics Communications, vol. 244, pp. 483 – 486, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010465519301729

[12] C. Y. Cardall and R. D. Budiardja, "Genasis mathematics : Object-oriented manifolds, operations, and solvers for large-scale physics simulations," Computer Physics Communications, vol. 222, pp. 384 – 412, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010465517303429

[13] C. Baker, G. Davidson, T. M. Evans, S. Hamilton, J. Jarrell, and W. Joubert, "High performance radiation transport simulations: Preparing for titan," in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ser. SC '12. Washington, DC, USA: IEEE Computer Society Press, 2012.

[14] "Comet repository," https://github.com/wdj/comet, 2019.

[15] D. Olds, "SC19 Cluster Competition Efficiency Ratings!" https://www.hpcwire.com/2019/12/19/sc19-cluster-competition-efficiency-ratings/, 2019.

[16] "Spack," https://spack.io/, 2021.

[17] R. van Oostrum, S. Moe, D. McDougall, and P. Bauman, "Amd tools overview," https://www.olcf.ornl.gov/wp-content/uploads/2019/10/ORNL_Application_Readiness_Workshop-AMD_Tools_Overview.pdf, 2019.

[18] "Amd rocm spack," https://github.com/RadeonOpenCompute/rocm-spack-pkgs, 2021.