Application Experiences on a GPU-Accelerated Arm-based HPC Testbed

Wael Elwasif
William Godoy
Nick Hagerty
J. Austin Harris
Oscar Hernandez
Balint Joo
Paul Kent
Damien Lebrun-Grandié
Elijah MacCarthy
Verónica G. Melesse Vergara
Bronson Messer
Ross Miller
Sarp Oral
elwasifwr@ornl.gov
godoywf@ornl.gov
hagertynl@ornl.gov
harrisja@ornl.gov
oscar@ornl.gov
joob@ornl.gov
kentpr@ornl.gov
lebrungrandt@ornl.gov
maccarthyea@ornl.gov
vergaravg@ornl.gov
bronson@ornl.gov
rgmiller@ornl.gov
oralhs@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, TN, USA

Sunita Chandrasekaran
schandra@udel.edu
University of Delaware
Newark, DE, US

Sergei Bastrakov
Michael Bussmann
Alexander Debus
Klaus Steiniger
Jan Stephan
René Widera
s.bastrakov@hzdr.de
m.bussmann@hzdr.de
a.debus@hzdr.de
k.steiniger@hzdr.de
j.stephan@hzdr.de
r.widera@hzdr.de
Helmholtz-Zentrum
Dresden-Rossendorf
Dresden, Saxonia, Germany

Spencer H. Bryngelson
Henry Le Berre
Anand Radhakrishnan
Jeffrey Young
shb@gatech.edu
hberre3@gatech.edu
aradhakr@gatech.edu
jyoung9@gatech.edu
Georgia Institute of Technology
Atlanta, GA, US

Florina Ciorba
Osman Simsek
florina.ciorba@unibas.ch
osman.simsek@unibas.ch
University of Basel
Basel, Basel-City, Switzerland

Kate Clark
Filippo Spiga
Jeff Hammond
John E. Stone
mclark@nvidia.com
fspiga@nvidia.com
jehammond@nvidia.com
jstone@nvidia.com
NVIDIA Corporation
Santa Clara, CA, USA

David Hardy
dhardy@illinois.edu
University of Illinois at
Urbana-Champaign
Urbana, IL, USA

Sebastian Keller
Jean-Guillaume Piccinali
keller@cscs.ch
jgp@cscs.ch
Swiss National Supercomputing Center
Lugano, Ticino, Switzerland

Christian Trott
crtrott@sandia.gov
Sandia National Laboratories
Albuquerque, NM, USA
ABSTRACT
This paper assesses and reports the experience of ten teams working to port, validate, and benchmark several High Performance Computing applications on a novel GPU-accelerated Arm testbed system. The testbed consists of eight NVIDIA Arm HPC Developer Kit systems, each one equipped with a server-class Arm CPU from Ampere Computing and two data center GPUs from NVIDIA Corp. The systems are connected together using InfiniBand interconnect. The selected applications and mini-apps are written using several programming languages and use multiple accelerator-based programming models for GPUs such as CUDA, OpenACC, and OpenMP offloading. Working on application porting requires a robust and easy-to-access programming environment, including a variety of compilers and optimized scientific libraries. The goal of this work is to evaluate platform readiness and assess the effort required from developers to deploy well-established scientific workloads on current and future generation Arm-based GPU-accelerated HPC systems. The reported case studies demonstrate that the current level of maturity and diversity of software and tools is already adequate for large-scale production deployments.

ACM Reference Format:
Wael Elwaif, William Godoy, Nick Hagerty, J. Austin Harris, Oscar Hernandez, Balint Joo, Paul Kent, Damien Lebrun-Grandie, Elijah MacCarthy, Verónica G. Melesse Vergara, Bronson Messer, Ross Miller, Sarp Oral, Sergei Bastrakov, Michael Bussmann, Alexander Debus, Klaus Steiniger, Jan Stephan, René Widera, Spencer H. Bryngelson, Henry Le Berre, Anand Radhakrishnan, Jeffrey Young, Sunita Chandrasekaran, Florina Corb, Osman Simsek, Kate Clark, Filippo Spiga, Jeff Hammond, John E. Stone, David Hardy, Sebastian Keller, Jean-Guillaume Piccinali, and Christian Trott. 2023. Application Experiences on a GPU-Accelerated Arm-based HPC Testbed. In International Conference on High Performance Computing in Asia-Pacific Region Workshops (HPCASIAWORKSHOP 2023), February 27-March 2, 2023, Raffles Blvd, Singapore. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3581576.3581621

1 INTRODUCTION
Deploying new supercomputers requires continuous evaluation of novel platforms and understanding of the trade-offs in porting existing applications to different architectures. With many of the HPC technology players building general-purpose or specialized accelerators, it is increasingly important to have a concrete understanding of the level of human-time investment required to make applications production-ready on any of these accelerated platforms, as well as the expected performance benefits to be gained with such effort.

Since the introduction of Arm Neoverse IP by Arm Ltd, we have witnessed a steady adoption and an increasing number of CPU products based on the Arm Instruction Set Architecture (ISA). Noticeable deployments include Sandia Astra (first petascale-class system deployed in 2018) and the RIKEN R-CCS Fugaku (first exascale-class system deployed in 2020). Fugaku, based on Fujitsu’s A64FX Arm-based CPU1 was also the first systems with a SIMD-capable CPU via the Arm Scalable Vector Extension (SVE) [30].

Looking at cloud deployments, the Graviton processor2 provides a significant portion of computational resources provisioned by Amazon Web Services. Now in its 3rd generation, the Graviton CPU is based on Arm Neoverse V1 core IP and supports Arm SVE SIMD instructions. AWS is not the only hyperscaler interested in deploying Arm CPUs; others, like Microsoft and Oracle, have started to offer Arm-based instances primarily based on Ampere Computing Altra and Altra Max CPUs.

In the very early days of the Arm journey into HPC, Arm systems were often custom-built and of limited scale (tens of nodes). The Mont–Blanc [27] project and the UK Catalyst initiative have paved the way to more robust and accessible systems, no longer experimental testbeds. In recent years hybrid CPU–GPU systems are becoming the dominant choice for large-scale leadership-class facilities (above ~100 PFlops) due to their performance and power efficiency. As we advance, platforms combining a modern Arm-based CPU with an energy-efficient high-performance GPU appear to be a natural choice to tackle future computing and computational challenges.

In collaboration with NVIDIA, Oak Ridge National Laboratory pioneered the combined use of Arm CPU and NVIDIA GPU in 2019. The NVIDIA Arm HPC Developer Kit3 represents a modern Arm-based GPU-accelerated platform. The upcoming NVIDIA Grace Superchip4 marks a step further in the platform design where CPU and GPU are tightly integrated into a “superchip” with enhanced I/O capabilities.

In this fast-paced evolving landscape of accelerators and heterogeneous systems, assessing as early as possible the viability of any technology and its impact on software maturity, code portability, and developer productivity remains a must. This paper presents an application-focused assessment of a multi-node NVIDIA Arm HPC Developer Kit test bed used primarily to validate software and ecosystem readiness. These systems are part of an experimental HPC cluster facility called Wombat, which is discussed in Section 2.

This study makes the following contributions: 1) The first thorough collaborative investigation of a modern GPU-accelerated Arm-based system using production applications; 2) Readiness analysis of those software tools required to compile the selected applications with and without GPU support; 3) preliminary performance results

Notice: This manuscript has been authored in part by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

HPCASIAWORKSHOP 2023, February 27-March 2, 2023, Raffles Blvd, Singapore © 2023 Copyright held by the owner/authors(s). Publication rights licensed to ACM. ACM ISBN 978-1-4503-9988-0/23/02... $15.00 https://doi.org/10.1145/3581576.3581621

1 https://www.fujitsu.com/global/products/computing/servers/supercomputer/a64fx/
2 https://aws.amazon.com/ec2/graviton/
3 https://developer.nvidia.com/arm-hpc-devkit
4 https://www.nvidia.com/en-us/data-center/grace-hopper-superchip/
compared to ORNL’s Summit system; and 4) overall general assessment of the software ecosystem readiness for GPU-accelerated Arm-based platforms.

A handful of production-ready HPC applications has been selected for this evaluation. Table 1 reports a simplified classification of the selected applications. Since the primary goal is to assess porting feasibility and obtain an initial performance baseline, we decided to use the selected applications as-is without investing any extra tuning efforts apart from adapting compiler flags or linking vendor-provided optimized libraries. Due to the breadth of the study and space constraints, some details are not included in this work but are available in [11], and references to pertinent sections are used where appropriate.

2 WOMBAT TESTBED

2.1 Background

Wombat is a small HPC cluster which has been equipped since 2018 with various Arm-based platforms from different vendors. The cluster is deployed and managed by The Oak Ridge Leadership Computing Facility (OLCF) and is freely accessible to users and researchers. The purpose of the cluster is to serve as a testbed for Arm-based AArch64 processors and related technologies within a close-to-production environment. Users who request access can use the system to port and validate their applications. Platform engineers at OLCF have been using Wombat to experiment and compare end-to-end integration and configuration aspects of Arm-based HPC systems.

2.2 Hardware

Currently the Wombat cluster consists of three sets of compute nodes:

1. **HPE Apollo 70** (4 nodes), each equipped with dual-socket Marvell ThunderX2 CN9980 processors and two NVIDIA V100 GPUs, connected via PCIe Gen 3.
2. **HPE Apollo 80** (16 nodes), each equipped with a single-socket Fujitsu A64FX processor.
3. **NVIDIA Arm HPC Developer Kit** (8 nodes), each equipped with a single-socket Ampere Computing Altra Q80–30 CPU (based on Arm Neoverse N1 IP) and two NVIDIA A100 GPUs - connected via PCIe Gen 4.

All nodes share a CPU-only login node based on dual-socket Marvell ThunderX2. All nodes are connected via either InfiniBand EDR or HDR to the same Infiband network.

2.3 Programming Environment

The programming environment and system software has been maintained as-is for the entire duration of the evaluation (April and May 2022). We consciously decide not to constantly vary the environment and create a fixed baseline. Wombat nodes boot their OS from the network, and all nodes are provisioned with the same pre-built compute image based on CentOS 8.1 with kernel 4.18. Job submission and execution are orchestrated using SLURM. The compilers and interpreters available include NVIDIA HPC SDK (NVHPC) 22.1, Arm Compiler for HPC 22, CUDA 11.5.1, GNU 11.1, LLVM 15.0.0 with OpenMP offload enabled, Python 3.9.0, and Julia 1.7.0. Networking support is provided by OFED 5.4 and UCX 1.11.1 and although most experiments are single node, OpenMPI 4.1.2a1 is installed for multi-node jobs. NSight Compute SDK, Allinea Forge, and Score-P are available for profiling purposes.

We use Spack [15] for additional third party scientific libraries and tools, including for example HDF5, OpenBLAS, and Score-P. We did not manually modify any compiler optimization flags used by Spack, aiming for an unfiltered “out-of-the-box” experience. Packages that did not have working Spack recipes were installed by Spack, aiming for an unfiltered “out-of-the-box” experience.

Each application team was responsible for building their respective application, installing extra dependencies, and linking the appropriate libraries.

3 EVALUATION METHODOLOGY

By definition, any testbed may lack some features found in final production systems. This fact should be taken into consideration when analyzing the performance results obtained.

For the purpose of this evaluation the most common performance score used in HPC, the Time-to-Solution, is not the primary Figure of Merit. Rather then perform a deep dive into the performance characteristics of each application, we perform a breadth-first study to assess platform’s software ecosystem readiness. This approach sets the stage for further improvements on system setup and tuning, aiming to increase robustness. Moreover enhancements in system architecture can be identified.

Following a call of contributions, 13 application teams agreed to participate in the evaluation process and 10 teams carried out the evaluation work until completion. Table 1 summarizes the final list of applications and their key characteristics. The list covers eight different scientific domains and includes codes written in Fortran, C, and C++. The parallel programming models used were MPI, OpenMP/OpenACC, Kokkos, Alpaka, and CUDA. We did not include changes to the application codes in the porting activities.

| App. Name     | Science Domain(s) | Language    | Parallel Programming Model(s) |
|---------------|-------------------|-------------|------------------------------|
| EvaStar       | Stellar Astrophysics | Fortran     | OpenACC, OpenMP offload      |
| CPU-1-TASSER  | Biotechnology     | C           | OpenACC                      |
| LAMMPS        | Molecular Dynamics | C++         | MPI, OpenMP, KOKKOS          |
| MFC           | Fluid Dynamics    | Fortran     | MPI, OpenACC                  |
| MILC          | QCD               | C/C++       | CUDA                         |
| NAMD/VMD      | Molecular Dynamics | C++         | Charm++, CUDA                 |
| PIC/hp100     | Plasma Physics    | C++         | Alpaka, CUDA                  |
| QMCPACK       | Chemistry         | C++         | OpenMP offload, CUDA         |
| SPEC/PC2021   | Variety of applications | C/C++, Fortran | OpenMP offload, OpenMP   |
| SPI-EAX2      | Hydrodynamics     | C++         | MPI, OpenMP, CUDA, HIP       |

Table 1: Applications evaluated on the Wombat testbed.

The evaluation process primarily focuses on application porting and testing, with less emphasis on absolute performance in light of the experimental nature of the testbed. Application teams were responsible for the basic configuration and build management for their respective application with support for installing needed system-wide packages using Spack as needed. The evaluation took place over two months spanning April and May 2022. Application
teams were free to choose the particular use cases to be evaluated for usability and performance on the testbed and to compare such performance with other platforms where the respective codes are regularly deployed.

### 3.1 Porting for functionality and correctness

The porting process for the applications used in this study was fairly straightforward. While applications use different programming languages, offloading approaches, and third party libraries, such factors did not pose a challenge for initial application porting and functionality. The improved maturity of the Arm software ecosystem, the general availability of the NVHPC toolkit for Arm, and improved support in the Spack package management system for Arm were all factors that contributed to a seamless porting process. Minor modifications to respective application build systems were however required as is typical when moving to a new HPC platform, though no major obstacles were encountered in this phase. In the following section, we do not report porting experience for each application, unless noteworthy issues were encountered regarding the usability of existing toolchains on the Wombat testbed.

### 4 APPLICATIONS

#### 4.1 ExaStar

4.1.1 Background. The toolkit for high-order neutrino-radiation hydrodynamics (thornado) [21] is a Fortran code (F2008) written as a stand-alone module that can be incorporated into ExaStar simulations [16] using the Flash-X multi-physics code. Thornado is used to compute the neutrino radiation field with a two-moment model for spectral neutrino transport that evolves moments of the neutrino phase-space distribution function representing spectral energy and momentum densities. In this study, we use two stand-alone thornado benchmarks as a tool for evaluating node-level performance: Streaming Sine Wave and Relaxation.

4.1.2 Performance and comparisons. As a baseline, we ran both benchmarks on a single node of the Summit computer at the Oak Ridge Leadership Computing Facility (OLCF). Each Summit node has 2 IBM POWER9 CPUs and 6 NVIDIA Volta GPUs, but for comparisons to the NVIDIA Arm HPC Dev Kit, we limit comparisons to a single CPU or single GPU. For the CPU runs with POWER9, we also test different configurations of Simultaneous Multithreading (SMT). The total number of OpenMP threads is set by the product of the number of cores and hardware threads available. To demonstrate the parallel efficiency of our OpenMP implementation, we also report serial execution times for each CPU. On both systems, we use standard -O2 optimizations and -tp for the target CPU. For benchmarks that report using the GPU, all computation is done on the GPU; the CPU thread is only used to launch kernels and manage data transfer. In both cases, the salient Figure of merit is wall-time (lower is better).

**Streaming Sine Wave.** We report the total wall-time to evolve ten timesteps of the Streaming Sine Wave benchmark for each hardware configuration in Table 2. The serial CPU comparison shows a speedup factor of 1.3x (2.5x) for the Ampere Altra relative to the POWER9 (ThunderX2). This single-core performance gain is also realized for the multi-core comparison, where we find speedup by a factor of 2.2x (2.8x) for Altra relative to POWER9 (ThunderX2). However, we find poor strong scaling of Altra (18% parallel efficiency with 80 threads) relative to POWER9 (42% efficiency with 21 threads). We speculate that this is rooted in the introduction of OpenMP overhead stemming from many small loop nests used in the streaming advection operation. This is further supported by the drop in performance on POWER9 for increasing SMT levels. The Altra+A100 results also exhibit a speedup factor of 1.3x (1.9x) relative to the POWER9+V100 (ThunderX2+V100) and a factor of 2.3x relative to the Altra CPU-core multi-core result. Further analysis of the contributions of different components to the overall performance on different platforms can be found in [11], Section 4.1.

**Relaxation.** We report the total wall-time to evolve 10 timesteps of the Relaxation benchmark for each hardware configuration in Table 3. We measure the improved serial performance of 1.2x (2.2x) for Altra relative to POWER9 (ThunderX2), though it is a smaller improvement than the previous benchmark. The Relaxation benchmark exhibits similar strong scaling efficiency for multi-core performance of Altra, and we find a speedup factor of 1.6x (3.2x) relative to POWER9 (ThunderX2). The GPU results are also favorable for the Altra+A100 configuration; we find a 1.7x (1.9x) speedup relative to POWER9+V100 (ThunderX2+V100) and a 21.5x speedup relative to the Altra CPU-only multi-core case. Further analysis of the performance across the different platforms can be found in [11].

#### 4.2 GPU-I-TASSER

4.2.1 Background. GPU-I-TASSER is a GPU-capable bioinformatics method for protein structure and function prediction. It is developed from the Iterative Threading ASsembly Refinement (I-TASSER) method [39]. The I-TASSER suite predicts protein structures through four main steps. These include threading template identification, iterative structure assembly simulation, model selection, and refinement, and the final step being structure-based

| CPU          | GPU      | Cores:SMT:Thrds. | Prog. Model | Time (sec) |
|--------------|----------|-----------------|-------------|------------|
| Power9       | None     | 1:1:1           | OpenMP      | 129        |
| ThunderX2    | None     | 1:1:1           | OpenMP      | 244        |
| Ampere Altra | None     | 1:1:1           | OpenMP      | 99.0       |
| Power9       | None     | 21:1:21         | OpenMP      | 14.5       |
| Power9       | None     | 21:2:42         | OpenMP      | 17.0       |
| Power9       | None     | 21:4:84         | OpenMP      | 21.3       |
| ThunderX2    | None     | 28:1:28         | OpenMP      | 18.6       |
| ThunderX2    | None     | 28:2:56         | OpenMP      | 17.8       |
| ThunderX2    | None     | 28:4:112        | OpenMP      | 18.5       |
| Ampere Altra | None     | 80:1:80         | OpenMP      | 6.72       |
| Power9       | V100     | 1:1:1           | OpenACC     | 3.75       |
| ThunderX2    | V100     | 1:1:1           | OpenACC     | 5.54       |
| Ampere Altra | A100     | 1:1:1           | OpenACC     | 2.96       |

Table 2: Comparison of thornado wall-clock times on each platform for the Streaming Sine Wave test problem. All runs used the ifortran compiler. Green rows indicate NVIDIA ARM HPC Development Kit hardware.
function annotation. The structure folding and reassembling stage is conducted by replica-exchange Monte Carlo simulations.

I-TASSER has predicted protein structures over the last decade with high accuracy. Thus, it has been ranked as the first automated server for protein structure prediction, according to the critical assessment of structure prediction (CASP) experiments, CASP7 through CASP13 [22].

Despite the robustness of I-TASSER in predicting protein structures with high accuracy, it takes considerably longer to predict some proteins’ structures. GPU-I-TASSER has therefore been developed to utilize the efficient GPU in predicting the structure of proteins. GPU-I-TASSER is developed by targeting bottleneck replica-exchange Monte Carlo regions of the protein structure prediction method and porting those to the device. The ported replica-exchange Monte Carlo regions utilize the GPU to optimize the application. The GPU optimization is based on OpenACC parallelization of bottleneck regions with extensive data management.

### 4.2.2 Performance and comparisons.

Performance gains across the testbed are compared to the performance from running the same benchmark dataset of proteins on Summit. For details regarding the hardware and software specs of Summit, please refer to [38]. To ensure that both systems are on the same level regarding performance comparison, we used the same GPUs. For the initial comparison, we assess the average runtime in seconds for both serial and GPU runs on Wombat using one ThunderX2 processor and one NVIDIA V100 GPU. We observe an average speedup of 7.68x using V100 GPUs on Wombat.

We further compare the performance across V100 GPUs to A100 GPUs on Wombat. We use one A100 and one V100 GPU in this case. We record an average of 7.35x speedup on A100 GPUs compared to the 7.68x on V100 GPUs on Wombat. We should note that the A100 runs were in-comparison to Ampere Computing Altra processors, whereas the V100 performance was relative to ThunderX2 processors. Also, we took the average runtimes against the number of cycles of simulations within a Monte Carlo run.

Finally, we compare the performance of GPU I-TASSER on Wombat to Summit using NVIDIA V100 GPUs. An average speedup of 6.92x is recorded using 1 V100 GPU on Summit. Comparing individual runs on Summit to Wombat, we can observe that Summit performed slightly better than Wombat across GPU and serial runs. Specifically, average serial and GPU runtimes per cycle of simulations measured in seconds are 1669.57 and 217.52, respectively, on Wombat, whereas on Summit, those are 1498.70 and 216.64, respectively.

Figure 1 shows the performance of Wombat’s ThunderX2 and Ampere Altra processors and NVIDIA A100 and V100 GPUs relative to the POWER9 processor on Summit. We record a slowdown of an average of 0.9x comparing ITASSER run on Wombat’s ThunderX2 processor to Summit’s POWER9 processor. For Ampere Altra (CPU-only), NVIDIA V100, and A100, we record positive speedups of 1.8x, 6.9x, and 13.3x, respectively.

### 4.3 LAMMPS and Kokkos

#### 4.3.1 Background.

The Kokkos C++ Programming Model is one of the leading ways of writing performance portable single source code for current and future HPC platforms [37]. It is widely used in the HPC community, particularly within the US National Laboratories and their partners. The programming model is implemented as a C++ abstraction layer on top of vendor-specific programming models such as CUDA, HIP, OpenMP, and SYCL. It is funded by the DOE Exascale Computing Project and developed by a multi-institutional team spanning several DOE laboratories.

LAMMPS is a widely used molecular dynamics application that one can use to simulate a wide range of materials, including condensed matter, gases, and granular materials [36]. It can leverage a wide array of architectures via Kokkos.

#### 4.3.2 Performance and comparisons.

We decided on four benchmarks that stress host-device interactions to investigate the impact of using Arm CPU as host. Generally, we do not expect code mainly bound by GPU execution time to show different behavior based on the host CPU.

---

| CPU       | GPU | Cores:SMT:Thrds. | Prog. Model | Time (sec) |
|-----------|-----|------------------|-------------|------------|
| Power9    | None| 1:1:1            | OpenMP      | 199        |
| ThunderX2 | None| 1:1:1            | OpenMP      | 374        |
| Ampere Altra | None| 1:1:1            | OpenMP      | 167        |
| Power9    | None| 21:1:21          | OpenMP      | 24.5       |
| Power9    | None| 21:4:84          | OpenMP      | 26.3       |
| ThunderX2 | None| 28:1:28          | OpenMP      | 48.9       |
| ThunderX2 | None| 28:2:56          | OpenMP      | 46.4       |
| ThunderX2 | None| 28:4:112         | OpenMP      | 44.3       |
| Ampere Altra | None| 80:1:80          | OpenMP      | 15.3       |
| Power9    | V100| 1:1:1            | OpenACC     | 1.21       |
| ThunderX2 | V100| 1:1:1            | OpenACC     | 1.32       |
| Ampere Altra | A100| 1:1:1            | OpenACC     | 0.71       |

Table 3: Comparison of thornado wall-clock times on each platform for the Relaxation test problem. All runs used the nvfortran compiler. Green rows indicate NVIDIA ARM HPC Development Kit hardware.
As comparison systems, we used one with an NVIDIA A100 GPU, an AMD EPYC (Milan) x86 CPU, and a system with NVIDIA V100 GPUs and an IBM POWER9 CPU. The latter system connects the GPU and CPU via NVLink. The measured performance numbers are given in Table 4.

Kokkos Kernel Latency. The Kokkos Programming model provides many different parallel operations, such as parallel_for and parallel_reduce, which come with different latencies. Overall, the Wombat system has latencies that fall between the x86 and the IBM POWER-based systems. While the pure launch latencies are comparable to x86, subsequent fences take longer. That, in turn, is reflected in higher latencies for reductions.

System Atomic Throughput. To measure the throughput of system atomics, we ran a benchmark distributed as part of the Kokkos repository, which emulates three common atomic access patterns. However, we modified the benchmark to perform the updates into host pinned memory, emulating scenarios where the host and the GPU work on some data collaboratively. The Wombat system performs similarly to the x86 system. The IBM system with NVLink interconnect is significantly faster.

Host-Device Data Transfer. We investigate three common host-device data transfer scenarios: transferring data to the device from regular and pinned host allocations and relying on page faults with managed memory.

For regular allocations, all systems perform similarly. With host pinned allocations, Wombat performs 3.5x worse than the IBM system with NVLink, and 25% worse than the x86 system. For managed allocations, the transfer rates depend significantly on the copy direction. Wombat beats the other systems for host-to-device transfers while being the slowest for device-to-host transfers.

LAMMPS. LAMMPS demonstrates the impact the observed behavior in the previous micro-benchmarks has on real applications. Often users run small problem sizes per GPU to achieve high simulation rates, making the code kernel latency sensitive. Furthermore, LAMMPS will be impacted by host device data transfer rates due to necessary MPI halo exchanges.

We chose a simple Lennard Jones type simulation with two different problem sizes (32k atoms and 256k atoms per GPU) to demonstrate this sensitivity. We only ran with one and two MPI ranks to avoid conflating the scaling behavior of LAMMPS into the data.

As the micro-benchmark would suggest, the most latency-sensitive scenario (single rank, 32k atoms) performs worse on Wombat than on the x86 system. The larger—less latency sensitive—system performs similarly on Wombat and the x86 system while being slower on the IBM machine due to its older GPU.

When running with two ranks, the total number of kernels increases, resulting in more latency overhead and significant host-device transfers. The data shows that Wombat performs fairly similarly to the x86 system. The IBM system does not seem to benefit from its NVLink connection, indicating that LAMMPS likely uses regular allocations in its non-GPU-aware MPI code path.

4.4 MFC

4.4.1 Background. MFC (Multi-component Flow Code) is an open-source fluid flow solver available at https://mflowcode.github.io [4].

Table 4: Performance of Kokkos-based benchmarks on different platforms. Latencies are measured in microseconds (μs), atomic throughput in billion updates per second (GUps), transfer rates in GB/s, and LAMMPS performance in million atomsteps per second (MAS/s). Except for latencies, higher is better.

| Benchmark | Arm+A100 | x86+A100 | P9+V100 |
|-----------|----------|----------|---------|
| latency par_for (μs) | 2.1 | 2.3 | 6.3 |
| latency par_for+fence (μs) | 10.0 | 8.7 | 15.0 |
| latency par_red (μs) | 2.3 | 2.7 | 6.2 |
| latency par_red+fence (μs) | 16.0 | 13.0 | 19.0 |
| atomic histogram (GUps) | 0.030 | 0.035 | 0.048 |
| atomic force update (GUps) | 0.150 | 0.170 | 0.470 |
| atomic mat.-assembly (GUps) | 0.150 | 0.170 | 0.470 |
| transfer h-d regular (GB/s) | 12 | 11 | 12 |
| transfer d-h regular (GB/s) | 11 | 11 | 11 |
| transfer h-d pinned (GB/s) | 18 | 25 | 62 |
| transfer d-h pinned (GB/s) | 15 | 21 | 60 |
| transfer h-d managed (GB/s) | 17 | 11 | 8 |
| transfer d-h managed (GB/s) | 12 | 17 | 26 |
| LAMMPS 1-MPI 32k (MAS/s) | 122 | 148 | 125 |
| LAMMPS 2-MPI 32k (MAS/s) | 95 | 89 | 98 |
| LAMMPS 1-MPI 256k (MAS/s) | 420 | 404 | 320 |
| LAMMPS 2-MPI 256k (MAS/s) | 201 | 201 | 139 |

It provides high-order accurate solutions to a wide variety of physical problems, including multi-phase compressible flows [29] and sub-grid dispersions [3]. MFC employs a finite volume shock and interface capturing scheme via weighted essentially non-oscillatory (WENO) reconstruction, HLL-type approximate Riemann solvers, and total variation diminishing time steppers. Quadrature moment methods handle the sub-grid closures [7].

The MFC codebase is written in Fortran with MPI (and CUDA-aware) capabilities for distributed parallelism. OpenACC provides GPU offloading capability for all compute kernels A Python front-end handles input data, execution, and metaprogramming for compiler optimizations. The FFTW package provides access to fast Fourier transforms for computing derivatives in cylindrical coordinates. HDF5 and Silo handle I/O and post-processing.

4.4.2 Performance and comparisons. We next investigate the performance of MFC on NVIDIA Arm HPC Development Kits, stressing both the Ampere CPUs and the NVIDIA A100 GPUs. A three-dimensional, two-phase, 16 million grid point fluid dynamics problem served this purpose, representing a typical multiphase flow workload. The performance metric of interest is the average execution wall-clock time over 10 time steps (excluding the first five steps). We tested performance on several available CPUs: Ampere Altra Q80-30, Fujitsu A64FX, Cavium ThunderX2, Intel Xeon Gold Cascade Lake (SKU 6248), and IBM POWER9. Both NVHPC and GCC v11.1 compilers were tested with -fast and -Ofast compiler optimization flags, respectively. GPU performance was analyzed for the NVIDIA V100 (accessible on Summit) and A100 (accessible on Wombat) using the NVHPC v22.1 compiler with the -Ofast flag. All computations are double precision.

5Access provided by Pittsburgh Supercomputing Center
Table 5: Comparison of wall-clock times per time step on various architectures. All comparison use either the NVHPC v22.1 or GCC v11.1 compilers as indicated. Highlighted rows indicate NVIDIA Arm HPC Development Kit hardware.

| Architecture       | # Cores | Compiler | Time [s] | Slowdown |
|--------------------|---------|----------|----------|----------|
| NVIDIA A100       | —       | NVHPC    | 0.28     | Ref.     |
| NVIDIA V100       | —       | NVHPC    | 0.50     | 1.7      |
| 2xXeon 6248       | 40      | NVHPC    | 2.7      | 9.6      |
| 2xXeon 6248       | 40      | GCC      | 2.1      | 7.5      |
| Ampere Altra      | 40      | NVHPC    | 3.9      | 14       |
| Ampere Altra      | 40      | GCC      | 2.7      | 9.6      |
| 2xPOWER9          | 42      | NVHPC    | 4.4      | 16       |
| 2xPOWER9          | 42      | GCC      | 3.5      | 12       |
| 2xThunderX2       | 64      | NVHPC    | 21       | 75       |
| 2xThunderX2       | 64      | GCC      | 5.4      | 19       |
| A64FX             | 48      | NVHPC    | 4.3      | 15       |
| A64FX             | 48      | GCC      | 13       | 46       |

Figure 2: Cost breakdown of different MFC subroutines on various architectures. Cases V100 and A100 have all compute kernels on the respective GPUs, so the associated CPU architecture is not meaningful. Numbers above the bars indicate the absolute wall-clock time (in seconds) as shown in table 5.

4.5 MILC

4.5.1 Background. MILC is an application package concerned with the simulation of Lattice Quantum Chromodynamics (LQCD) to further the study of the (sub-)nuclear physics. MILC handles the generation of gauge field configurations (sampling of the partition function) using Markov Chain Monte Carlo methods, most commonly RHMC [8], and analyzes those configurations to generate physics observables. For both, the dominant algorithm is the iterative linear solver, stemming from the discretized Dirac equation on a 4-d spacetime, giving rise to a sparse matrix, or stencil, one must repeatedly solve. Conjugate Gradient is the solver of choice for the commonly used HISQ discretization [13] employed by MILC practitioners.

While popular in the LQCD community, MILC is also often used as a benchmark for HPC systems. Node-level performance is usually dictated by memory bandwidth or, in the case of multi-node scaling, the network bandwidth. Specifically, the inter-process bandwidth must be fast enough to overlay the stencil halo communication with the local stencil application.

MILC runs on GPUs via QUDA library. Given the propensity for high memory bandwidth on GPUs relative to CPUs, offloading the iterative solver to the GPU dramatically increases the inter-process (GPU) memory bandwidth required to successfully strong scale.

4.5.2 Performance and comparisons. To probe performance, we utilize the NERSC Medium benchmark and look at performance on one and two GPUs on the same node, comparing performance to a platform with AMD EPYC 7742 Rome CPUs and identical A100 GPUs. This platform is similar because it lacks the NVLink interconnect and has the same PCIe gen4 capability. However, critically it supports the peer-to-peer PCIe protocol allowing for inter-GPU communication without staging in CPU memory. We also include measurements taken on the ThunderX2 system compared to Summit, with the latter notably supporting peer-to-peer communication.

---

1. https://github.com/milc-qcd/milc_qcd
2. https://github.com/lattice/quda/wiki/Running-the-NERSC-MILC-Benchmarks
3. While NVSHMEM is supported on Rome, we chose to make a more direct comparison by deploying MPI exclusively as the communication protocol.
using NVLink. Due to memory footprint size, we include only 2 GPU results.

Table 6 breakdowns the benchmark run times. We note the following key results:

- Single GPU performance is roughly equivalent between Wombat and Rome (2650 s vs. 2705 s), with a slight advantage over Wombat.
- For Dual GPU performance, we see Rome does significantly better (1684 s vs. 1548 s), with the primary deficit arising due to the "compute".
- The non-GPU accelerated computation "host" shows that Wombat is more than competitive with Rome.
- The raw copy bandwidth between host and device seems to favor the Altra, regardless of the direction of the copy.
- Summit performs significantly better overall than ThunderX2 (2645 s versus 3186 s), with the primary deficit being due to compute.

To better understand the poor scaling of Wombat on two GPUs, in Figure 3 we plot the performance of the HISQ stencil for the three precisions, the application of which is responsible for the bulk of the time spent in the mixed-precision solver. Without communication, we see performance parity between the two platforms. However, when we include communication overhead, we see that Wombat’s performance is severely impacted. In particular, we note that half-precision on 2 GPUs is 45% slower on Wombat versus Rome. We do not include the ThunderX2 and Summit results here for brevity, but we note that a similar picture is painted: with ThunderX2 having a 54% performance deficit for the half-precision stencil.

### 4.6 NAMD and VMD

#### 4.6.1 Background

NAMD [26] and VMD [17] are biomolecular modeling applications for molecular dynamics simulation (NAMD) and for preparation, analysis, and visualization (VMD). Researchers use NAMD and VMD to study biomolecular systems ranging from individual proteins, large multi-protein complexes, photosynthetic organelles, and entire viruses. Both programs support hardware platforms ranging from personal laptops, workstations, and clouds, up to the largest parallel supercomputers [1]. NAMD and VMD are written in C++, C, CUDA, and some platform-specific SIMD vector intrinsics and assembly language for specific performance-critical routines. NAMD is based on the Charm++ parallel runtime system [18], which provides an adaptive, asynchronous, distributed, message-driven, task-based parallel programming model using C++. NAMD and VMD incorporate built-in interpreters for Tcl and Python to provide easy-to-use scripting.

#### 4.6.2 Notes on porting for functionality and correctness experience

The first adaptations of NAMD and VMD to Arm hardware were performed with SoC on-chip GPU embedded system platforms (NVIDIA CArmA, KAYLA, Jetson TK1, and Jetson TX1), or PCIe-attached GPU (Applied Micro X-Gene/ThunderX + Tesla K20c) system [31]. Wombat presented no compilation barriers for NAMD or VMD, but some minor issues are noted. The Charm++ parallel runtime system used by NAMD did not compile cleanly with GCC 11.0.1 did not generate correct results for SVE, so the newer version 11.1.0, so GCC 10.2 was used to compile NAMD and its associated system [31]. Wombat presented no compilation barriers for NAMD or VMD, but some minor issues are noted. The Charm++ parallel runtime system used by NAMD did not compile cleanly with GCC 11.0.1 did not generate correct results for SVE, so the newer version 11.1.0, so GCC 10.2 was used to compile NAMD and its associated components. Besides the CUDA toolkit, NAMD also requires FFTW and Tcl libraries, which were easily built on Wombat. Performance results for GPU-resident NAMD are reported in Table 7 and Table 8.

VMD used a new startup query of CPU SIMD vector instruction set extensions for runtime dispatch of performance-critical loops to hand-vectorized CPU kernels. VMD was extended to query Arm64 CPU vector instruction availability using the Linux kernel getauxval() API, enabling runtime detection and kernel dispatch for Arm64 NEON and VMD vector instructions. New hand-vectorized data-parallel NEON and VMD kernels were developed for key atom selection operations and for molecular orbital analysis and visualization, with performance reported in [11]. The new NEON and VMD molecular orbital kernels are direct mathematical and algorithmic descendants from previous CPU and GPU kernels [25, 31–35].

Testing of SVE vector instructions on Fujitsu A64fx nodes demonstrated that two recent versions of the Arm compiler toolchain (21.1 and 22.0) and LLVM (Clang) 10.0.1 generated incorrect code for particular SVE vector intrinsics used in the VMD molecular orbital kernel. As such, the older Arm HPC toolkit version 20.3 was used for the reported results. Similarly, LLVM/Clang versions older than 11.0.1 did not generate correct results for SVE, so the newer version was used for reported results.
A100–PCIe

1.58

GCC

128.2

2

4.6.3 NAMD performance and comparisons. Benchmarks are shown for the new GPU-resident code path in NAMD [26], which is able to fully utilize an A100 GPU. Although GPU-resident NAMD scales across multiple GPUs on a single node, it depends on high-performance peer-to-peer GPU communication through NVLink using relatively fine-grained load-store operations within CUDA kernels. The lack of this capability on ORNL Wombat limited this study to single GPU performance and the best use of the Ampere Altra.

| CPU : Cores:SMT:Threads | GPU | Comp. (ns/day) |
|--------------------------|-----|---------------|
| ThunderX2 : 32:4:2      | V100-PCie | GCC | 124.9 |
| 2xPower9 : 42:4:7       | V100-NVLINK | XLC | 125.7 |
| 2xXeon 6134 : 16:2:4    | A100-PCie | ICC | 181.4 |
| Ampere Altra : 80:1:4   | A100-PCie | GCC | 182.2 |
| DGX-A100 : 128:2:2      | A100-SXM4 | GCC | 187.5 |

Table 7: NAMD single-GPU performance for 92K-atom ApoA1 simulation, NVE ensemble with 12a cutoff, rigid bond constraints, multiple time stepping with 2fs fast time step, and 4fs for PME. Green rows indicate development kit hardware.

| CPU : Cores:SMT:Threads | GPU | Comp. (ns/day) |
|--------------------------|-----|---------------|
| ThunderX2 : 32:4:8      | V100-PCie | GCC | 9.43  |
| 2xPower9 : 42:4:7       | V100-NVLINK | XLC | 10.26 |
| 2xXeon 6134 : 16:2:8    | A100-PCie | ICC | 14.52 |
| Ampere Altra : 80:1:40  | A100-PCie | GCC | 15.09 |
| DGX-A100 : 128:2:8      | A100-SXM4 | GCC | 15.87 |

Table 8: NAMD single-GPU performance for 1M-atom STMV simulation, NVE ensemble with 12a cutoff, rigid bond constraints, multiple time stepping with 2fs fast time step, and 4fs for PME. Green rows indicate development kit hardware.

4.7 PIconGPU

4.7.1 Background. PIconGPU [5] is a C++ application that is a scalable, heterogeneous, and fully relativistic particle-in-cell (PIC) code and provides a modern simulation framework for laser-plasma physics and laser-matter interactions suitable for production-quality runs. The code is used to develop advanced particle accelerators for cancer radiation therapy, high-energy physics, and photon science. PIconGPU utilizes the alpaka [19, 23] abstraction layer and the particle-in-cell algorithm for its science case simulations.

For this work, we use a configuration of PIconGPU that simulates a Weibel instability in a plasma of electrons and positrons, i.e., where all particle species have equal mass. Three variations with different computational intensity are considered: one with a cubic-spline particle shape using single-precision floating point and two with quadratic-splines using single- and double-precision, respectively.

Structurally, PIconGPU is a stencil code with spatial domain decomposition. To facilitate scaling benchmarks, automatic estimation of suitable buffer sizes for particle exchange was introduced into PIconGPU. Each MPI rank exchanges boundary/ghost values and particles passing the boundaries with its spatial neighbors using asynchronous point-to-point communication. The particle-grid operations are spatially local and so fit in this scheme.

For the following performance evaluation, we used the aforementioned configuration and verified the correctness of the results by comparing them to previous benchmark results we have collected on other systems.

4.7.2 Performance and comparisons. Our main analysis focus was execution on Wombat’s Ampere nodes since PIconGPU is not yet a fully heterogeneous code, we did separate runs for the CPUs and the A100 GPUs. Additionally, we evaluated both single precision and double precision data. For all benchmarks, we used the Triangular Shape Cloud (TSC) particle form factor. Variation across multiple grid dimensions would result in more MPI overhead, so we restricted the benchmark variants to the z dimension.

Experimental setup. For the CPU runs, we used one MPI rank per node. Each MPI utilized 80 OpenMP threads. From PIconGPU’s perspective, this constitutes a single CPU device per node. For the GPU runs, we used two MPI ranks per node with one rank per A100 GPU. Both configurations maximise the use of the available resources.

Weak scaling. For the weak scaling analysis, we used a base problem size of 100 time steps and 256 × 256 × 256 cells per computation device. Then we added another 256 cells to the z dimension for any additional device. Table 14 in [11] shows the setup per node in more detail. The results of the weak scaling benchmarks are shown.
in Table 9. With the efficiency staying above 90% for all cases, it can be demonstrated that PIConGPU scales well across multiple Ampere compute nodes – on a previously unknown HPC system and equally unfamiliar hardware – with minimal porting effort.

However, there are also significant differences between CPU and GPU efficiency. This can be explained by the absolute runtime required for the computation as shown in Table 11. The GPUs perform the computations much faster than the CPUs. In turn, the GPU weak scaling efficiency is affected by MPI communication overhead much more than the CPU efficiency, likely due to GPU to host data transfer.

| Nodes | Scaling | Altra SP | Altra DP | A100 SP | A100 DP |
|-------|---------|----------|----------|---------|---------|
| 1     | Weak    | 1.000    | 1.000    | 1.000   | 1.000   |
| 2     | Weak    | 0.998    | 0.997    | 0.992   | 0.986   |
| 4     | Weak    | 0.995    | 0.994    | 0.982   | 0.970   |
| 8     | Weak    | 0.992    | 0.989    | 0.930   | 0.911   |

Table 9: Weak Scaling Efficiency for PIConGPU (where ideal = 1.000). Problem size per device: $256 \times 256 \times 256$ and 100 timesteps using TSC Particle form factor (SP: single precision, DP: double precision).

| Nodes | Scaling | Altra SP | Altra DP | A100 SP | A100 DP |
|-------|---------|----------|----------|---------|---------|
| 1     | Strong  | 1        | 1        | 1       | 1       |
| 2     | Strong  | 2.00     | 2.04     | 1.89    | 1.92    |
| 4     | Strong  | 3.99     | 4.08     | 3.28    | 3.48    |
| 8     | Strong  | 7.94     | 8.09     | 4.73    | 5.20    |

Table 10: Strong Scaling Factors for PIConGPU (where ideal = 1). Problem size per device: $256 \times 256 \times 256$ and 100 timesteps using TSC Particle form factor (SP: single precision, DP: double precision).

Table 11: Total computation times for PIConGPU’s weak scaling benchmark. Problem size per device: $256 \times 256 \times 256$ and 100 timesteps. Particle form factor: TSC: SP: single precision, DP: double precision.

### 4.8 QMCPACK

#### 4.8.1 Background
QMCPACK[20] is an open-source, high-performance Quantum Monte Carlo (QMC) package that solves the many-body Schrödinger equation using a variety of statistical approaches. The few approximations made in QMC can be systematically tested and reduced, potentially allowing the uncertainties in the predictions to be quantified at a trade-off of the significant computational expense compared to more widely used methods such as density functional theory. Applications include weakly bound molecules, two-dimensional nanomaterials, and solid-state materials such as metals, semiconductors, and insulators.

The present study’s goal is to evaluate the performance of the Diffusion Monte Carlo (DMC) algorithm on NVIDIA A100 GPUs and Arm Ampere CPUs using QMCPACK’s standard performance tests. They consist of short DMC calculations of variously sized supercells of bulk nickel oxide, $\text{NiO}$. The computational cost of these calculations formally scales cubically with the total electron count, which in turn is determined by the atoms in the supercell and their elemental composition.

#### Performance and comparisons
We set up a set of problem sizes in the $\text{NiO}$ supercell benchmark characterized by the number of electrons in the system. Memory usage is formally quadratic in the electron count. As memory requirements increase, the number of potential “walkers” that can fit in the GPU or on-node memory reduces. Because the GPU implementation batches work over the number of walkers, the achievable efficiency can be limited if the batch size can not be large enough before the GPU memory is exhausted.

Performance is measured using a throughput metric. As defined in (1), throughput is measured as the computational cost associated with a single DMC simulation yielding to the frequency of advancing walkers in the DMC simulation, with higher values indicating better performance. The cost is cubic in the electron count and linear in the walker count. Thus the throughput drops dramatically at large electron counts.

$$\text{Throughput} = \frac{\text{walkers} \times \text{blocks} \times \text{steps}}{\text{DMC time}}$$

**GPU-only Results.** The initial focus on targeting Wombat’s NVIDIA’s A100 GPUs on Ampere nodes is to understand the number of possible “walker count per GPU device” for the $\text{NiO}$ supercell benchmark for different system sizes. Walker counts in QMCPACK are equivalent to the “batch size” for GPU computation, finding the maximum number of walkers also allows for efficient use of each available GPU. We apply a bisectional search to find the maximum walker count limits due to memory limitations within a single walker count range for accuracy ($\pm 1$ walkers). The resulting walker count limits per A100 GPU (40 GB) are given in Table 12 which also provides this information for reference on the V100 GPU, offering 16 GB of memory, from our experiments on Summit. As the system size increases, the benefits of the A100 memory become larger, with the largest measured system size of 6144 electrons surpassing the simple memory ratio between A100 and V100 of 2.5x by a factor of...
max walkers

719
818
240
1244
174
32797
12554
Summit V100
2047
423
785
2047
1
96
65535
322
32
65535
Wombat A100
65534
35419

Figure 4: QMCPACK DMC throughput for Wombat and Summit nodes as a function of the number of electrons in the NiO benchmark from Table 13.

32 due to the significant additional memory overheads in storing wavefunctions used in the calculation.

Table 13: The maximum number of walkers (batch size) on a single Wombat A100 and Summit V100 GPU.

| NiO supercell electrons | max walkers (batch size) | max walkers (batch size) |
|-------------------------|--------------------------|--------------------------|
|                         | Summit V100              | Wombat A100              |
| 48                      | 633535                   | 65535                   |
| 96                      | 35419                    | 65534                   |
| 192                     | 12554                    | 32797                   |
| 384                     | 818                      | 2047                    |
| 768                     | 785                      | 2047                    |
| 1152                    | 423                      | 1244                    |
| 1536                    | 240                      | 719                     |
| 2304                    | 96                       | 322                     |
| 3072                    | 43                       | 174                     |
| 6144                    | 1                        | 32                      |

We use the walker count on Table 13 on each system to compare the DMC performance throughput on (1) ranging from 1 GPU to the maximum limit using Summit’s 6 V100 GPUs and Wombat’s 2 A100 GPUs per node. Results are illustrated in Figure 4 showing the results obtained on Wombat using the NVHPC compiler and on Summit. As expected, single A100 GPU runs on Wombat outperform those on V100s, with significantly larger throughput for nearly all problem sizes. When using all the available GPUs per node on each system, we observe that for smaller cases, Summit 6 V100 GPUs outperform in terms of throughput per node. However, Wombat’s A100 GPUs are significantly more performant for the largest and most computationally challenging case. For these system sizes, greater GPU memory is the biggest factor in increased performance.

In addition to the study using GPU offloading, we performed an assessment using CPU only configuration for QMCPACK. Those results can be found in Section 4.8 in [11].

4.9 SPEC HPC 2021

4.9.1 Background. SPECpc 2021 is a benchmark suite comprised of real-world application codes designed for portable performance across heterogeneous CPU and GPU architectures [2]. SPECpc provides C/C++ and Fortran codes, accelerated by OpenMP, OpenMP Offloading, OpenACC, and CUDA programming models. On Wombat, we utilized SPECpc 2021 to evaluate single-node performance using one to two NVIDIA A100 GPUs while varying the number of cores bound to each GPU.

4.9.2 Performance and comparisons. We ran the SPECpc 2021 suite on Wombat comparing the results to ORNL’s Summit. The compilers used on Wombat were NVHPC 22.1 using OpenMP target offloading (NVHPC-TGT) and OpenACC offloading (ACC), and LLVM v15.0.0 using OpenMP target offloading (LLVM-TGT). POT3D, SOMA, and Weather benchmarks data is not provided since LLVM is not built with Fortran support. Three iterations of the tiny benchmark were performed on Wombat. On Wombat, we tested with combinations of one and two NVIDIA A100 GPUs. We ran the benchmark suite using one and two ranks per GPU for a total of four data points for each acceleration model. On Summit, we tested the use of six V100 GPUs with one iteration using one rank per GPU. Summit displays several runtime errors while running on one V100 GPU because the SPECpc tiny benchmark targets about 40 GB of memory usage, which exceeds the V100 limit of 16 GB.

Figure 5 and Figure 6 show the performance (measured as wall-time) of the OpenMP target offloading implementations of NVHPC and LLVM on Wombat and Summit, respectively, relative to NVHPC OpenACC. A 19x speed-up difference in runtime is observed in Minisweep from NVHPC-ACC to NVHPC-TGT on Wombat using a single GPU, one rank per GPU, and a 14x difference is observed when using both A100 GPUs. This behavior is not limited to Wombat, as Summit also observed an 8x slowdown from NVHPC-ACC to NVHPC-TGT when using all 6 GPUs, one rank per GPU. This behavior is also not limited to NVHPC’s OpenMP offloading, as LLVM-TGT demonstrates a 4-6x slowdown on Minisweep on both Summit and Wombat.

Using one GPU on Wombat, five of the six codes that complete with NVHPC-TGT are slower than when using NVHPC-ACC, and all three of the codes that complete for LLVM-TGT are slower than when using NVHPC-ACC. On all GPUs, 7 of the 9 codes run faster using ACC than TGT on Wombat, and 5 of the 7 codes that complete for LLVM-TGT demonstrate a 4-6x slowdown on Minisweep on both Summit and Wombat.

4.10 SPH-EXA2

4.10.1 Background. The SPH-EXA2 project is a multidisciplinary effort that extends the SPH-EXA [6] project and aims to scale the Smoothed Particle Hydrodynamics (SPH) method to enable exascale hydrodynamics simulations for Cosmology and Astrophysics. On Wombat, we used the Sedov-Taylor blast wave explosion test [14] to simulate a spherical shock generated by the instantaneous injection of thermal energy at a single point in a static uniform background. This test requires the code to simulate shock-fronts while correctly maintaining spherical symmetry and conservation laws. SPH-EXA2 [13] is open source, written in C++17, parallelized with MPI and OpenMP, and accelerated with CUDA and HIP.

12https://www.spec.org/hpc2021/
13https://github.com/unist-dmi-hpc/SPH-EXA
4.10.2 Performance and comparisons. To investigate the impact of using the Arm CPU on SPH-EXA2, we conduct tests on three different systems within the Wombat platform (described in Section 2.2) and two x86_64 non-Arm systems (described in [11, Table 20]). We report and compare the performance results of a CPU-only run and a CPU+GPU run using a single node executing the Sedov–Taylor blast test case with 200$^3$ particles for 800 time-steps.

**CPU-only Results.** Figure 7 shows the results for the MPI+OpenMP code version of SPH-EXA2 on CPU only setup. The average time in seconds per time-step of the simulation is shown on the top chart (lower is better), and the achieved iteration throughput per minute of the simulation is shown on the bottom chart (higher is better). On Wombat, the best performance is obtained with the GNU compiler on the Ampere N1 CPU, while the overall best performance is achieved on x86_64 CPUs. Systems with fewer cores per socket lead to lower overall performance than those with higher core counts. Additionally, the results on Marvel ThunderX2 and Fujitsu A64FX systems show that the SPH-EXA2 code compiled with the GNU compiler outperforms the Arm compiler.

Further code profiling using the Arm Performance Reports tool allowed us to identify the cause of the performance difference between Ampere N1 and Fujitsu A64FX CPUs since the former has fewer cores but performs better in our tests. Profiling showed that a higher number of L2 cache misses and stalled cycles on the Fujitsu

A64FX CPUs cause performance degradation. We believe this is due to the Ampere N1 having only 1 NUMA node compared to the 4 NUMA nodes of Fujitsu A64FX. Further analysis is needed to use the vectorization support (SVE) better and increase compute performance.

**CPU+GPU Results.** Figure 8 shows the execution times of the MPI+OpenMP+CUDA version of the SPH-EXA2 code. The Ampere N1 system on Wombat slightly outperforms the x86_64 reference system. The difference in performance is caused by the Ampere N1 having PCIe 4.0 compared to the x86_64 reference system’s PCIe 3.0 port, which creates the difference between data transfer rates between the CPU and the GPU. The size and speed of CUDA memcpy operations reported in Table 14 show that the same amount of data was transferred between host (H) and device (D) on both systems, with higher transfer rates on Wombat’s Ampere N1.
Prior work has primarily focused on the evaluation of HPC applications on the Arm Cavium ThunderX2 with the Aries interconnect as part of the Isambard supercomputer [24] and the A64FX processor with TOFU interconnect in the Fugaku system [28] and with InfiniBand interconnect [12] on the Okami system. Other related work has looked at Arm-based performance portability with ThunderX2 and previous generation Ampere nodes [9] and concludes that Kokkos and OpenMP provide performance portability across several HPC applications running on Arm, including a full set of mini-apps and applications [15], but it did not include any accelerated nodes. This work complements other HPC application efforts on the NICs on the node. In the near future, more tightly integrated

In terms of more cloud-HPC-focused efforts, a recent hackathon run by the non-profit Arm HPC User Group, AWS, and Arm supported the testing and development of HPC codes on AWS’s custom Graviton2 instances. This event, the AHUG Hackathon: Cloud Hackathon for Arm-based HPC [14], supported 30 teams to investigate the top HPC applications used on AWS and helped test Spack packages with flags for the Graviton2 setup as well as Reframe testing scripts for Arm and x86 platforms. The effort focused on porting several HPC applications running on Arm, including a full set of mini-apps and applications [15], but it did not include any accelerated nodes. This work complements other HPC application efforts on AWS, including Nalu [16], a CFD modeling code, and NWChem [17], a widely used quantum chemistry code.

6 CONCLUSIONS

In this work, we used the Wombat testbed at the Oak Ridge Leadership Computing Facility (OLCF) to study the readiness and usability of a modern GPU-accelerated Arm-based HPC platform, the NVIDIA Arm HPC Developer Kit. Ten representative applications from different scientific domains, and using a variety of programming models and languages were selected, built on the platform and tested for correctness. Wherever possible, performance was compared with other leading HPC platforms used for production science, as well as other Arm-based platforms that are part of the Wombat system.

As seen from the various application experiences, the porting process was straightforward and mostly required minor modifications to the build systems to compile and run on the target platform. The availability of a fairly mature set of compilers that cover the gamut of used programming models was crucial in achieving this seamless porting process. Of particular note, the availability of the NVIDIA HPC SDK facilitated the porting process or those applications that currently use this tool-chain on other GPU-accelerated supercomputers, such as Summit. Furthermore, the maturity of Arm support in the Spack package management system greatly facilitated the deployment of third-party tools and libraries needed by the various application teams.

While exhaustive performance optimization was not a primary goal of this work, we carried out preliminary performance measurements to assess the overall platform readiness. For applications considered GPU-dominant, performance improvements were commensurate with the hardware capabilities of the NVIDIA Ampere GPU (A100) relative to the previous generation NVIDIA Volta GPU (V100), and using an Arm-based CPU did not adversely impact the outcome. We carried out several CPU-only experiments for a subset of the applications where the code can be configured to run only on the CPU. We observed that the Ampere CPU’s performance was generally competitive with leading x86-64 and Power9 CPUs. It should be noted that the lack of an appropriate fast and fully RDMA-capable CPU-GPU bus in the Wombat testbed (similar to NVIDIA NVLink on POWER9 CPU in Summit or AMD’s xGMI in the newly installed Frontier supercomputer at OLCF) and the lack of NVLink across the CPU and GPUs adversely impacted performance for applications that require fast data movement across the different processing elements in the platform. Exploiting these features requires a holistic design that combines needed system software with a hardware design that adopts a GPU-centric platform design. Such a design can be found in systems such as NVIDIA DGX [18] or Frontier [15], where the GPUs are connected directly to the NICs on the node.
of the Platform for Advanced Scientific Computing Conference (Geneva, Switzerland) (PASC ’20). Association for Computing Machinery, New York, NY, USA, Article 11, 11 pages. https://doi.org/10.1145/3394277.3401855

[7] A. Charalampopoulos, S. H. Bryngelson, T. Colonius, and T. P. Sapsis. 2022. Hybrid quadrature moment method for accurate and stable representation of non-Gaussian processes applied to bubble dynamics. Philosophical Transactions of the Royal Society A (2022).

[8] M. A. Clark and A. D. Kennedy. 2007. Accelerating staggered-Fermion dynamics with the rational hybrid Monte Carlo algorithm. Physical Review D 75, 1 (2007). https://doi.org/10.1103/physrevd.75.011502

[9] Tom Deakin, Simon McIntosh-Smith, James Price, Andrei Poenaru, Patrick Atkinson, Codrin Pops, and Justin Solomon. 2019. Performance Portability across Diverse Computer Architectures. In 2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC). 1–13. https://doi.org/10.1109/P3HPC51976.2019.00006

[10] Wael Elwisaf, William Godoy, Nick Hagerty, J. Austin Harris, Oscar Hernan, Dezlan Joo, Paul Kent, Damien Lebrun-Grandie, Eliaj Maccarty, Veronica G. Melesse Vergara, Bronson Messer, Ross Miller, Sarp Ompl, Sergey Bastrakov, Michael Bussmann, Alexander Debus, Klaus Steinger, Jan Stephani, Rene Fonder, Spencer H. Bryngelson, Henry Le Berre, Anand Radhakrishnan, Jeffery Young, Sunita Chandrasekaran, Florina Corba, Osman Simsek, Kate Clark Filippi Spiga, Jeff Hammond, John E. Stone, David Hardy, Sebastian Keller, and Jean-Guillaume Picard. 2022. Application Experiences on a GPU-Accelerated Arm-based HPC Testbed. https://doi.org/10.48550/ARXIV.2209.09791

[11] Catherine Feldman, Benjamin Michalowicz, Eva Siegmann, Tony Curtis, Alan Calder, and Robert Harrison. 2022. Experiences with Porting the FLASH Code to Oakami, an HPE Apollo 80 A6EX Platform. HPCAsia 2022 (to appear) (2022).

[12] E. Follana, Q. Mason, C. Davies, K. Hornbostel, G. P. Lepage, J. Shimigens, H. Trottier, and K. Wong. 2007. Highly improved staggered quarks on the lattice with applications to charm physics. Physical Review D 75, 5 (mar 2007). https://doi.org/10.1103/physrevd.75.054502

[13] Nicholas Frontierr, J. D. Embsomber, Michael Buehlmann, Joseph Adamo, Salam Habah, Katrin Heitmann, and Claude-André Faucher-Giguère. 2022. Simulat- ing Hydrodynamics in Cosmology with CRK-HACC. https://doi.org/10.48550/ARXIV.2209.02840

[14] Todd Gamblin, Matthew P. Lengendre, Richard Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and Wes Futral. 2015. The Spack Package Manager: Bringing order to HPC software chaos. In Supercomputing 2015 (SC’15). Austin, Texas.

[15] J. Austin Harris, Ran Chu, Sean M Couhc, Anshu Dube, Eirik Endeve, Antigoni Giagiodi, Rajery Jain, Daniel Kasen, M P Laiu, OE B Messer, Jared Oöté-Meal, Michael A. Sandoval, and Klaus Weide. 2022. Exascale models of stellar explosi- ons: Quintessential multi-physics simulation. The International Journal of High Performance Computing Applications 36, 1 (2022), 59–77. https://doi.org/10.1177/10943420219725667

[16] F. Schmitt, U. Schramm, J. Schuchart, and R. Widera. 2013. Article 5, 12 pages. https://doi.org/10.1145/2503210.2504564

[17] Jeffrey Kelling, Sergei Bastrakov, Alexander Debus, Thomas Khue, Matt Leinhauer, Richard Fausch, Klaus Steinger, Jan Stephani, René Widera, Jeff Young, et al. 2022. Challenges Porting a C++ Template-Metaprogramming Abstraction Layer to Directive-based Offloading. arXiv preprint arXiv:2110.08650 (2021).

[18] R. P. C. Kent, Abdullah Hannaberdiev, Anouar Benali, M. Chandler Bennett, Edgar Josebachvili, Linzda Borda, Peter Dosk, Hongxia Hao, Kenneth D. Jordan, Jaron T. Krogel, Ikka Kyömpäjärvi, Joonho Lee, Ye Liao, Fionn D. Malone, Cody A. Melton, Lulu Mitas, Miguel A. Morales, Eric Neuscamman, Fernando A. Reboredo, Brenda Rubenstein, Kayahan Saritas, Shiv Upadhyay, Guangming Wang, Shuai Zhang, and Lunung Zhao. 2020. QMCPACK: Advances in the develop- ment, efficiency, and application of auxiliary field and real-space variational and diffusion quantum Monte Carlo. The Journal of Chemical Physics 152 (2020). 174105. https://doi.org/10.1063/5.0004860

[19] M. Paul Lau, Eirik Endeve, Ran Chu, J. Austin Harris, and E. O. Bronson Messer. 2021. A DG-IMEX Method for Two-moment Neutrino Transport: Non-linear Solvers for Neutrino-Matter Coupling. Astrophys. J., Suppl. Ser. 253, 2, Article 52 (April 2021), 52 pages. https://doi.org/10.3847/1538-4365/abae78

[20] Eliaj A Maccarty, Chengxin Zhang, Yang Zhang, and KC Dukka. 2022. GPU-TAS- SER: a GPU accelerated I-TASSER protein structure prediction tool. Bioinfor- matics (2022).

REFERENCES

[1] BügeAcun, David J. Hardy, LuximantKale, Ke Li, James C. Phillips, and John E. Stone. 2018. Scalable Molecular Dynamics with NAMD on the Summit System. IBM Journal of Research and Development 62, 6 (2018), 41–4:9. https://doi.org/10.1147/rd.2018.2889866

[2] Holger Brustein, Sunita Chandrasekaran, Florina Ciorba, Nick Hagerty, Robert Henschel, Guido Jackelund, Junjie Li, Veronica G. Melesse Vergara, Sandra Wienke, and Miguel Zavala. 2022. First Experiences in Performance Benchmarking with the New SPECpouc 2021 Suites. https://doi.org/10.48550/ARXIV.2203.00751

[3] S. H. Bryngelson, K. Schmidmayer, and T. Colonius. 2019. A quantitative comparison of phase-averaged models for bubbly, cavitating flows. International Journal of Multiphase Flow 115 (2019), 137–143. https://doi.org/10.1016/j.ijmultiphaseflow.2019.03.028

[4] Spencer H. Bryngelson, Kevin Schmidmayer, Vedran Coralic, Jomela C Meng, Kazuki Maeda, and Tim Colonius. 2021. MFC: An open-source high-order multi- component, multi-phase, and multi-scale compressible flow solver. Computer Physics Communications 266 (2021), 107396.

[5] M. Bussmann, H. Buring, T. E. Cowan, A. Debus, A. Huebl, G. Jackelund, T. Kluge, W. E. Nagel, R. Pausch, F. Schmitt, U. Schramm, J. Schuchart, and R. Widera. 2013. Radiative Signatures of the Relativistic Kelvin–Helmholtz Instability. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (Denver, Colorado) (SC ’13). ACM, New York, NY, USA, Article 5, 12 pages. https://doi.org/10.1145/2503210.2504564

[6] Aurélien Cavelan, Rubén M. Caberoz, Michel Grabarczyk, and Florina M. Ciorba. 2020. A Smoothed Particle Hydrodynamics Mini-App for Exascale. In Proceedings
[23] Alexander Mattes, René Widera, Erik Zenker, Benjamin Worpitz, Axel Huebl, and Michael Bussmann. 2017. Tuning and Optimization for a Variety of Many-Core Architectures Without Changing a Single Line of Implementation Code. Using the Alpaka Library. In High Performance Computing. Julian M. Kunkel, Ryo Yokota, Michela Tauber, and John Shalf (Eds.). Springer International Publishing, Cham, 496–514. https://doi.org/10.1007/978-3-319-67630-2_36

[24] Hans-Christian McIntosh-Smith, James Price, Andrei Ponnuru, and Tom Deakin. 2020. Benchmarking the first generation of production quality Arm-based supercomputers. *Concurrency and Computation: Practice and Experience* 32, 20 (2020), e5369.

[25] Marcelo C. R. Melo, Rafael C. Bernardi, Till Budack, Maximilian Scheuerer, Christoph Riplinger, James C. Phillips, Julio D. C. Maia, Gerd B. Rocha, João V. Ribeiro, John E. Stone, Frank Nese, Klaus Schulten, and Zaida Luthey-Schulten. 2018. NAMD goes quantum: An integrative suite for hybrid simulations. *Nature Methods* 15 (2018), 351–354.

[26] James C. Phillips, David J. Hardy, Julio D. C. Maia, John E. Stone, João V. Ribeiro, Rafael C. Bernardi, Ronak Buch, Giacomo Fiorin, Jérôme Hénin, Wei Jiang, Ryan McGreavy, Marcelo C. R. Melo, Brian Radak, Robert D. Skeel, Abhishek SinghaRoy, Yi Wang, Benoit Roux, Aleksei Aksimentiev, Zaida Luthey-Schulten, Laxmikant V. Kale, Klaus Schulten, Christophe Chipot, and Emad Tadjkhorshid. 2020. Scalable molecular dynamics on CPU and GPU architectures with NAMD. *Journal of Chemical Physics* 153 (2020), 044130. https://doi.org/10.1063/5.0014475

[27] Nikola Rajojevic, Alejandro Rico, Nikola Puzovic, Chris Adeniyi-Jones, and Alex Ramirez. 2014. Tiledahpe: Making the case for an ARM-based HPC system. *Future Generation Computer Systems* 36 (2014), 322–334.

[28] Mitsuhisa Sato, Yutaka Ishikawa, Hirofumi Tomita, Yuetso Kodama, Tetsuya Oda-jima, Miwako Tsuji, Hisashi Yashiro, Masaki Aoki, Naoyuki Shida, Isao Miyoshi, Kouichi Hira, Atsushi Furuya, Akira Asato, Kuniki Morita, and Toshiyuki Shimizu. 2020. Co-Design for A64FX Manycore Processor and "Fugaku". In SC20: *International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15. https://doi.org/10.1109/SC41.405.2020.00051

[29] K. Schmidtmark, S. H. Bryngelson, and T. Colonius. 2020. An assessment of multicomponent flow models and interface capturing schemes for spherical bubble dynamics. *J. Comput. Phys.* 402 (2020), 109080. https://doi.org/10.1016/j.jcp.2019.109080

[30] N. Stephens, S. Biles, M. Boettcher, J. Eapen, M. Eoyele, G. Gabrielli, M. Hornsme, G. Magklis, A. Martinez, N. Premillieu, A. Reid, A. Rico, and P. Walker. 2017. The ARM Scalable Vector Extension. *IEEE Micro* 37, 02 (mar 2017), 26–39. https://doi.org/10.1109/MM.2017.35

[31] John E. Stone, Michael J. Hallock, James C. Phillips, Joseph R. Peterson, Zaida Luthey-Schulten, and Klaus Schulten. 2016. Evaluation of Emerging Energy-Efficient Heterogeneous Computing Platforms for Biomolecular and Cellular Simulation Workloads. 2016 IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW) (2016), 89–100. https://doi.org/10.1109/IPDPSW.2016.130

[32] John E. Stone, David J. Hardy, Jan Saam, Kirby L. Vandivort, and Klaus Schulten. 2011. GPU-Accelerated Computation and Interactive Display of Molecular Orbitals. In *GPU Computing Gems*; Wen-mei Hwu (Ed.). Morgan Kaufmann Publishers, Chapter 1, 5–18.

[33] John E. Stone, David J. Hardy, Ivan S. Ulfimtsev, and Klaus Schulten. 2010. GPU-Accelerated Molecular Modeling Coming of Age. *J. Molecular Graphics and Modelling* 29 (2010), 116–125.

[34] John E. Stone, Antti-Pekka Hynninen, James C. Phillips, and Klaus Schulten. 2016. Early Experiences Porting the NAMD and VMD Molecular Simulation and Analysis Software to GPU-Accelerated OpenPOWER Platforms. *International Workshop on OpenPOWER for HPC (IWOHPF)* (2016), 188–206.

[35] John E. Stone, Jan Saam, David J. Hardy, Kirby L. Vandivort, Wen-mei W. Hwu, and Klaus Schulten. 2009. High-Performance Computation and Interactive Display of Molecular Orbitals on GPUs and Multi-core CPUs. In *Proceedings of the 2nd Workshop on General-Purpose Processing on Graphics Processing Units, ACM International Conference Proceeding Series*, Vol. 383. ACM, New York, NY, USA, 9–18.

[36] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in’t Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton. 2022. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Comp. Phys. Comm.* 271 (2022), 108171. https://doi.org/10.1016/j.cpc.2021.108171

[37] Christian R. Trott, Damien Lebrun-Grandi, Daniel Arndt, Jan Ciesko, Vinh Dan, Nathan Ellingwood, Rahul Kumar Gayatri, Evan Harvey, Daisy S. Hollow, Dan Ibanez, Nevin Liber, Jonathan Madsen, Jeff Miles, David Polakoff, Amy Powell, Sivasankaran Rajamanickam, Mikael Simberg, Dan Sunderland, Bruno Turckcin, and Jeremiah Wilke. 2022. Kokkos 3: Programming Model Extensions for the Exascale Era. *IEEE Transactions on Parallel and Distributed Systems* 33, 4 (2022), 805–817. https://doi.org/10.1109/TPDS.2021.3097283

[38] Verónica G Vergara Larrea, Wayne Joubert, Michael J Brim, Reuben D Budiardja, Don Maxwell, Matt Ezell, Christopher Zimmer, Swen Boehm, Wael Elwaisif, and Sarp Oral, et al. 2019. Scaling the summit: deploying the world’s fastest supercomputer. In *International Conference on High Performance Computing*. Springer, 330–351.

[39] Wei Zheng, Chengxin Zhang, Eric W Bell, and Yang Zhang. 2019. I-TASSER gateway: a protein structure and function prediction server powered by XSEDE. *Future Generation Computer Systems* 99 (2019), 73–85.