Ookami: An A64FX Computing Resource

A C Calder[1,2], E Siegmann[2], C Feldman[1,2], S Chheda[2], D C Smolarski[3], F D Swesty[1], A Curtis[2], J Dey[2], D Carlson[2], B Michalowicz[4], and R J Harrison[2]

[1] Department of Physics and Astronomy, Stony Brook University, Stony Brook, NY 11794-3800, USA
[2] Institute for Advanced Computational Science, Stony Brook University, Stony Brook, NY 11794-5250, USA
[3] Department of Mathematics & Computer Science, Santa Clara University, Santa Clara, CA 95053, USA
[4] Department of Computer Science and Engineering, The Ohio State University, Columbus, 0H 43210, USA

E-mail: alan.calder@stonybrook.edu

Abstract. We present a look at Ookami, a project providing community access to a testbed supercomputer with the ARM-based A64FX processors developed by a collaboration between RIKEN and Fujitsu and deployed in the Japanese supercomputer Fugaku. We provide an overview of the project and details of the hardware, and describe the user base and education/training program. We present highlights from previous performance studies of two astrophysical simulation codes and present a strong scaling study of a full 3D supernova simulation as an example of the the machine's capability.

1. Introduction

In this paper, we describe Ookami, a project providing access to a testbed computing system featuring the ARM-based A64FX processor developed by a collaboration between RIKEN and Fujitsu and deployed in Fugaku, what was, until June of 2022, the world's fastest supercomputer [1]. The project is run by Stony Brook University (SBU) in cooperation with the University at Buffalo [2] and provides open access along with training and resources to effectively use such hardware. Users have been able to port, analyze, and optimize the performance of many applications [3]. Below, we describe the hardware, our user base and education program, and provide highlights from previous performance studies of two astrophysical simulation codes, the astrophysical radiation hydrodynamics code V2D [4] and multi-application package FLASH [5, 6], here applied to thermonuclear supernovae. We also present a strong scaling study of a production 3D supernova simulation that demonstrates the capability of the hardware.

2. Overview of Project

The primary goal of the Ookami project is to provide access to a platform with the A64FX processors for testing and development. These processors offer an alternative to graphical programming units (GPUs) that power many contemporary supercomputers but may require considerable code development to make good use of. The expectation is that the reduced instruction set A64FX processors will provide high performance and reliability for

applications with common programming models, particularly memory-intensive applications, while maintaining a good performance-to-power ratio similar to GPUs. Users with a variety of applications are able to explore and evaluate this new hardware and its potential for use in settings from a local cluster to the extreme scale of leadership-class supercomputing centers.

The Ookami project started in 2019 and will run for six years, providing researchers worldwide with access to this cutting-edge computing technology. For the first few years, access to Ookami was allocated via SBU. Researchers submitted allocation requests directly to the Ookami team, who reviewed them. Approved projects were granted access to the cluster for testbed projects, and once users could prove that their application performs well on this novel architecture, they could advance their projects to production status. In October 2022 Ookami became an ACCESS [7] resource provider, and now 90% of its resources are allocated via ACCESS. Users can submit requests to ACCESS and, if approved, get credits, which they can then exchange for resources (e.g. node hours, GPU hours, storage) on one or multiple of the ACCESS resource providers.

2.1. The A64FX Hardware

Ookami is an HPE (formerly Cray) Apollo 80 system with 174 + 2 debug Fujitsu A64FX-FX700 compute nodes, each of which has 48 cores divided into 4 core memory groups (NUMA regions), each with 8 GB of high-bandwidth memory (HBM) for a total of 32 GB per node. Each core has a 64 KB L1 cache, and an 8 MB L2 cache shared between the cores in each core memory group, and runs at 1.8 GHz. These processors use the ARMv8.2–A Scalable Vector Extension (SVE) SIMD instruction set with a 512-bit vector implementation, allowing for vector lengths from 128 to 2048 bits (in 128-bit increments) and enabling vector length agnostic programming [8].

The operating system for each processor resides on a node's local 512 GB SSD and processors communicate via an Infiniband HDR100 fat tree interconnect with 200 Gb/s switches. A high-performance Lustre file system provides about 800 TB of storage. At present, Ookami is running Rocky Linux 8.4, and compilers and toolchains from GNU, LLVM, ARM, HPE/Cray, NVIDIA, LLVM, and Fujitsu are installed.

2.2. User Base

Since Ookami was opened to researchers, 358 users have been onboarded. Those users work on 125 projects, 31 of which have been allocated via ACCESS. The experience of the users is very diverse as they range from undergrad students to professors and professionals with decades of High Performance Computing (HPC) experience. Figure 1 illustrates the distribution of users.

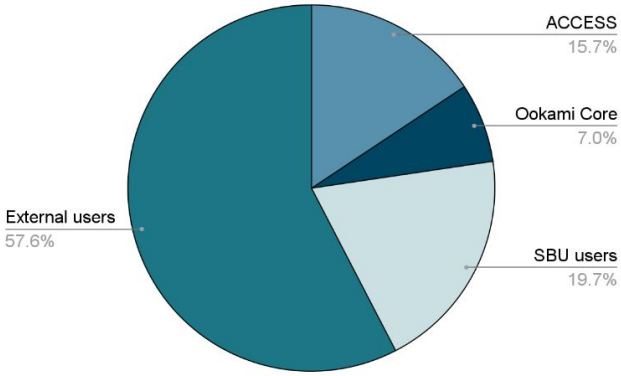

Figure 1. Pie chart with the distribution of users on Ookami, 358 as of November, 2023. SBU users are researchers from on-campus. External users are from other academic institutions. The core team is the personnel working on the project, including computational scientists, grad students, and system administrators. ACCESS users are the recent users added via the ACCESS program.

2.3. Education and Training

To cover the complex needs of the user community, the Ookami project has established a multi-modal support approach [9]. The extensive documentation, FAQ section, and getting-started

guide on the project's website [2] allow users to learn everything about the system and its efficient use via self-study. However, because user support is especially important for novel technology, Ookami also offers a Slack channel, to which all new users are automatically added. There they can ask questions, get feedback, or just start an informal conversation with other users as well as with the project team. The average response time is normally within a few minutes, making this the perfect tool for quick help. Office hours, offered twice a week, enable users to ask more detailed questions. These are virtual meetings open to all users, led by Ookami team members. This is the perfect opportunity to do an interactive debugging or profiling session, or to just show applications and get feedback. On average these calls are attended by six to seven persons. Ookami also offers a traditional ticketing system, which is mainly used for installation requests, new user or project requests, or used by users who prefer this approach over the Slack channel.

Regular webinars spanning various topics, e.g. profilers, debuggers, and programming languages, are held to support the user community holistically and to provide updated input for the usage of HPC tools. The Ookami team also reaches out to communities that have potential interest in the cluster, e.g. Campus Champions [10], Science Gateways [11], and computational researchers with suitable applications, and introduces them to A64FX and the opportunity to use Ookami for their work. Other outreach activities include an introduction to HPC for high school students, who come to the institute without prior experience, learn the basics of HPC, get up and running on Ookami, and depart with a self-reported positive experience.

3. Performance Studies

Guided by the bountiful support of the Ookami team and user community, we were able to port and evaluate the performance of two different astrophysical simulation codes on Ookami: V2D and FLASH. Below, we report the results of our efforts including comparing the performance of different available compilers and considering the capacity for full production runs.

3.1. V2D

The V2D code solves the Euler equations of inviscid hydrodynamics and multi-species flux-limited diffusive radiation transport in two spatial dimensions via finite-difference methods. It was originally designed for the core-collapse supernovae problem of astrophysics but may be applied to other radiation-hydrodynamic problems. Details related to the underlying numerical methods can be found in [4]. V2D is written in modern Fortran, employs MPI for communication, and uses the the HDF5 library for parallel I/O.

The problem we chose for our study with V2D is a the propagation of a 2D Gaussian pulse of radiation, a diffusive radiation transport problem. The computational effort is in the solution of a large, sparse, memory-bandwidth-limited linear system that describes the time evolution of the radiation distribution. The linear system consists of $x1 \times x2 \times 2$ coupled linear equations, with $x1 = 200$, and $x2 = 100$ zones in the spatial dimensions, and there are two radiation species. Each time step thus requires the solution of a unique $x_1 \times x_2 \times 2$ linear system, and the solution is obtained via the BiCGSTAB algorithm. This is a relatively small test problem chosen to explore the performance of SVE optimization. Also, we make only limited use of the parallel capability of V2D when we vary the process topology to adjust the problem size on/ each processor. The linear system is sparse, but it has a regular structure. The method is matrix-free, but if the matrix corresponding to this system were stored with a dictionary-like ordering it would form a banded matrix with five bands.

We tested combinations of compilers and MPI implementations. As a sample of results, those presented below indicate test runs using the GNU (ver. 11.1.0), Fujitsu (ver. 4.5), and HPE/Cray (ver. 21.03) compilers. We compared the CPU times of simulations compiled with different compilers, both with and without SVE optimization. The Linux `perf stat` command of the Linux kernel performance monitor, with the `-e duration_time` flag, was used to measure

the time of the simulations. This command measures the entire CPU time of the process. Each configuration (of the total number of processors used and the process topology in the $x1$- and $x2$- directions, determining how the linear system was partitioned) was run several times to confirm the timing results. Also, each test problem was run for 100 time steps.

All compilers available on Ookami are able to make use of SVE capabilities. In our tests, the SVE and optimization features were used, but we also turned off the SVE and other optimization features on some of the HPE/Cray tests for comparison purposes. When using a single processor, the executable compiled by GNU took the longest to complete the test run, around 363.91 seconds, while that compiled by the Fujitsu compiler, around 252.31 seconds, and the executable compiled using the HPE/Cray compiler (with optimization), around 181.26 seconds. Using more processors, however, with different domain decomposition to CPU core topologies, the executable compiled by Fujitsu performed better than the one compiled by HPE/Cray.

The following chart presents these results. The values in the N_p column refer to the total number of processors used for the run, with the numbers in the "Direction" `NX1` column indicating the number of domain decomposition tiles in the $x1$ direction and similarly for the numbers in the `NX2` column. Thus the product of the two values equals the total number of processors requested. The column labeled HPE/Cray (opt) indicates results obtained with an executable compiled both with both `-O3` optimization and SVE optimization enabled.

N_p	**Direction**		**Times by Compiler (seconds)**	
	`NX1`	`NX2`	Fujitsu	HPE/Cray (opt)
40	40	1	13.97	19.12
40	20	2	12.96	17.37
40	10	4	13.04	17.16
50	50	1	13.05	25.56
50	25	2	12.09	24.07
50	10	5	11.40	23.51

We note that the 50 core runs necessitated going to cores on a second node. The results varied by compiler, but in the case of the Cray compiler the "plateau" in the performance may follow from the overhead of communication across nodes. Further details may be found in [12].

3.2. FLASH

3.2.1. Overview and Initial Experiences FLASH is a simulation software package for addressing multi-scale, multi-physics applications. Initially developed at the University of Chicago to address thermonuclear flashes, stellar explosions powered by a thermonuclear runaway occurring on the surface or in the interiors of compact stars, FLASH continues to be developed for astrophysics [13] and high-energy-density physics [14], and a new code, FLASH-X, derived from FLASH and with a completely new infrastructure, is under development and will allow addressing more general problems [15].

At its heart, FLASH is a hydrodynamics plus additional physics (e.g. a stellar equation of state) method. FLASH uses the PARAMESH library to implement adaptive mesh refinement (AMR) to address problems with a wide range of physical scales on a block-structured mesh [16, 17]. FLASH is written primarily in modern Fortran and is parallelized primarily through MPI, although some solvers have been modified to take advantage of threaded approaches to parallelization [18] and development continues toward a more general design for better thread support [19–21].

PARAMESH manages a block-structured adaptive mesh, with the data typically in $16\times16\times16$ zone blocks (16×16 in 2D). Each block also includes four ghost zones in each direction around the block, and communication between blocks occurs through exchange of data in the ghost zones. Variables like density, temperature, internal energy, etc., are stored in

a data container, `unk`, a Fortran array in the form `unk(nvar, il_bnd, iu_bnd, jl_bnd, ju_bnd, kl_bnd, ku_bnd, maxblocks)`, where `nvar` is the number variables, `il_bnd:iu_bnd, jl_bnd:ju_bnd, kl_bnd:ku_bnd` are the x, y, and z zone limits, and `maxblocks` is the maximum number of blocks allowed for a given processor element. Accordingly, block data is typically accessed block-by-block with the result being that there is a stride in memory for addressing variables in different zones or blocks.

As it was one of the marquee applications for the Ookami project, FLASH was ported as soon as Ookami was up and available. Sorting out the compilers and their options, versions of MPI, and requisite packages like the HDF5 library took some effort, but our initial experience with FLASH and other applications was overwhelmingly positive [3]. FLASH ran "right out of the box" with several compilers and MPI implementations, scaling reasonably well with no tuning. Profiling with Linaro (née Arm) MAP [22] indicated that thermonuclear supernova simulations, our problem of interest, spent considerable time in the hydrodynamics and equation of state (EOS) routines so we decided to focus our analysis on those routines while running 2D supernova simulations. We investigated use of SVE with the HPE/Cray, Fujitsu, and GNU compilers, but vectorization proved difficult due to significant branching in the main loops of the EOS routines. Details of our exploration with scaling results and our attempt to utilize the A64FX's SVE instructions and NUMA architecture may be found in [23].

To investigate other areas where we could improve performance, we instrumented the code with the Performance Application Programming Interface (PAPI) [24], and found that the number of data translation lookaside buffer (DTLB) misses were exceptionally high. The DTLB is a special cache that manages the mapping of virtual to physical memory. Given the stride in memory of the data layout of PARAMESH, a logical choice was to investigate memory management to study the high number of DTLB misses and we chose the use of Huge Pages.

3.2.2. Huge Pages and Compiler Comparison Huge Pages (hp) are a Linux kernel feature that allow blocks of memory larger than the default 64 KB page to be used when managing virtual memory, reducing the number of pages managed by the operating system (OS). This is useful for simulations with a large memory footprint. Hp come in two types, standard (explicitly managed) and transparent (automatically managed by the OS). Transparent hp are by default disabled on Ookami, so the results presented here use standard 2 MB hp.

For our hp study, we ran 2D supernova simulations and 3D pure hydrodynamics simulations of the Sedov explosion problem, one of the standard test problems provided with FLASH. We dubbed these "EOS" and "3-d Hydro" because the equation of state and 3D hydrodynamics routines were the parts of the code instrumented for performance testing. Details of the EOS and the Sedov explosion problem may be found in the original FLASH paper [5]. For the hp study, the EOS test ran a $\sim$ 1 GB 2D SN Ia simulation for 50 time steps and one refine ending with 624 blocks and the 3D Hydro test ran a $\sim$ 9 GB Sedov explosion simulation for 2 time steps and one refine ending with 3337 blocks. Both tests were run on 1 and 12 cores. FLASH Morton orders the blocks to be spatially located together, so to spread the blocks as much as possible the runs on 12 cores used the round robin distribution of processors. Details of the simulations, science, and results may be found in [25].

We began our study of hp with the Fujitsu compiler because it readily made use of hp (by default) and needed just a compiler flag to toggle between using hp or not. Subsequent investigation allowed us to utilize hp with other compilers by linking to the Fujitsu library `libmpg`. Using PAPI, we collected hardware counter data over the course of the run, and then used it to calculate derived quantities such as bandwidths and latencies following the calculations in the A64FX manual [26]. We refer to these counters and their derived metrics as "performance measures". The performance measures were then used to compare runs with hp, without hp, with the Fujitsu compiler, and with the GCC compiler, to better understand how different settings

and compilers took advantage of the architecture. Runs on 1 core and 12 cores exhibited similar patterns in their performance measures, so we show results for only the 1 core runs below.

Figure 2 shows a comparison between simulations of the two test problems compiled with the Fujitsu 4.5 and GCC 12.2.0 compilers. Shown are ratios with and without hp of runtime, CPU cycles, DTLB misses, L1 and L2 DTLB misses, L1D and L2 cache misses, cycles waiting for memory, Bandwidth between L1D and L2 caches and L2 and HBM, and Latency for L1D cache misses and L2 cache misses.

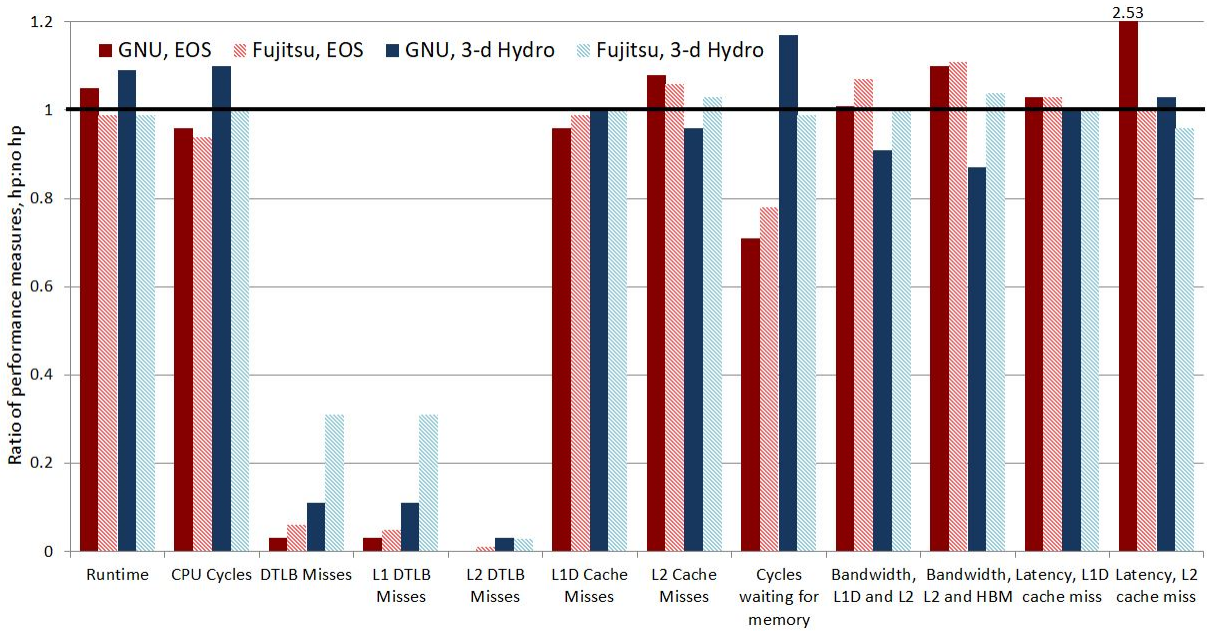

Figure 2. Bar chart showing the ratios of performance measures with and without hp for simulations of the two test problems with the Fujitsu and GNU compilers. Each bar is the ratio of each metric for simulations with and without hp. The DTLB misses show a drastic decrease when hp is enabled, regardless of compiler or test problem. For reasons unknown, the GNU executable for the EOS problem had 2.53× higher latency with hp enabled than disabled. The rest of the measures remain fairly close to 1.

Figure 3 presents the results of performance tests of FLASH for the two problems with and without hp. Shown are the same performance measures as the compiler test of Figure 2, except the ratios compare the compiler performance of the Fujitsu to the GNU compiler.

The results for the hp study may be summarized as follows. The use of hp decreased DTLB misses as expected, but this decrease did not affect performance as we expected. We attribute this finding to the performance of the translation table cache, but further research is needed to fully understand this result. The difference in compilers, however, had a large effect on runtime without making any changes to the code. FLASH performed faster with the Fujitsu compiler, requiring half the runtime and half the number of CPU cycles as with the GNU compiler. Fujitsu-compiled executables can access HBM 1.5 - 3× faster, so even though a similar number of cache misses as with the GNU compiler were seen, misses are not as expensive. Using the Fujitsu compiler, the code spends half as much time waiting for memory than when compiled with the GNU compiler. But because we also found that memory access is only 20-40% of the runtime (not illustrated in the plots), we conclude that the increased bandwidth can't completely account for the speedup. Complete results of our exploration of hp may be found in [27, 28].

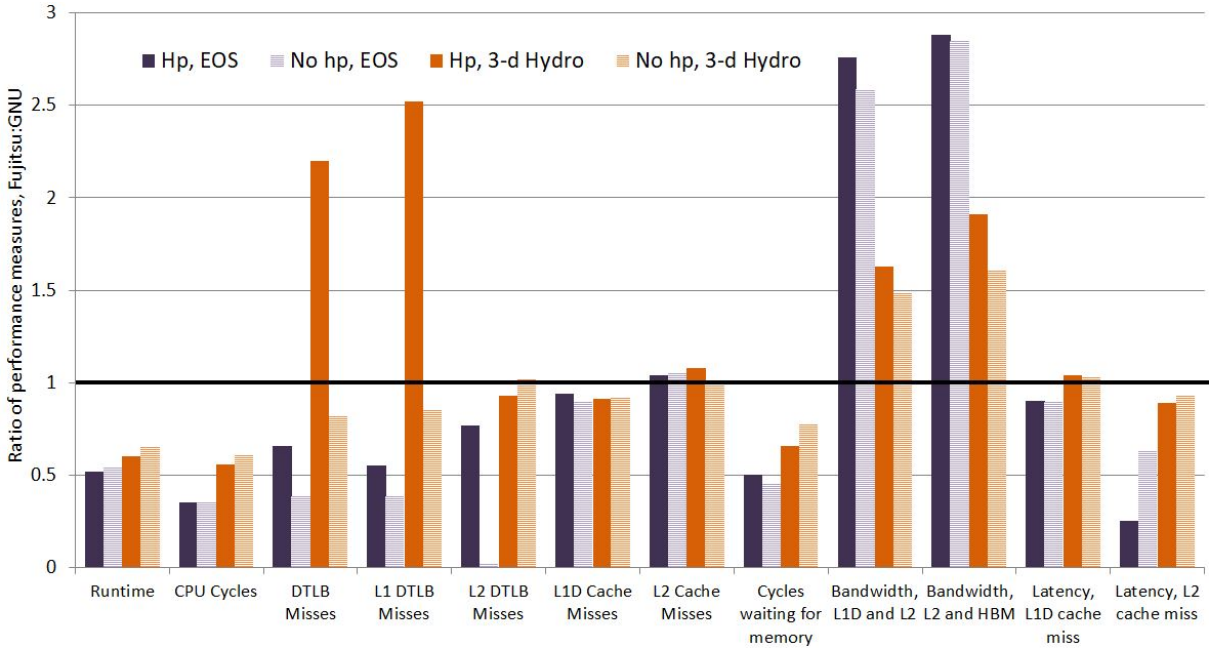

Figure 3. Bar chart showing the ratios of performance measures for Fujitsu and GNU compilers for simulations of the two test problems with (Hp) and without (No hp) hp. Each bar is the ratio of each metric for simulations compiled with the Fujitsu to the GNU compiler.

3.2.3. 3D Production Test As a way of testing the efficacy of Ookami for production runs, for this work we performed a scaling study with a full 3D simulation of a thermonuclear supernova, one of a suite of simulations from an ongoing science investigation. In this case, FLASH was compiled for $16 \times 16 \times 16$ zones per block and a maximum of 50 blocks per core. The simulations included the full physics of the flame model, equation of state, and multipole Poisson solve, and were run under production conditions in which the simulation was restarted from a checkpoint file and the mesh adapted as dictated by the fluid flow, typically every two time steps. FLASH was compiled with the GNU compiler v12.1.0 and were linked to the OpenMPI v4.1.4 MPI library and parallel hdf5 v1.12.1.

The simulations for the scaling study were started "in medias res" at about 0.5 s into the explosion when the flame is very active. The simulations consisted of approximately 33,000 blocks and ran for 20 time steps with 10 mesh refinements. The restart file was 30 GB, and the simulation required a minimum of 220 GB to run. The simulations were run on 21, 32, 64, and 128 nodes 7 times each, removing the minimum and maximum runtimes and averaging the remaining 5. The simulation ran on 36 of the 48 cores on each node, with 9 MPI ranks per NUMA region. This choice follows from experimentation with running simulations to fill much of the memory on a node. We discuss this issue in more detail below. The results of the scaling study are illustrated in Figure 4 and demonstrate reasonable strong scaling through 128 nodes. Also, the figure presents curves for the the total run time and the evolution time. The difference is the time required for initialization, reading the checkpoint file, and saving the final output, about 9 - 33 % of the run time. Although the input file is the same size in all cases, this percentage increases with the number of nodes as the amount of communication increases.

The choice of running on 36 of 48 possible cores follows from limitations of both FLASH and the A64FX memory. The number of blocks allocated must fit within the effective 27 GB node memory limit of Ookami. This is complicated by the adaptive mesh which needs more space to store its tree structure as the number of blocks grows, and the memory used by MPI which

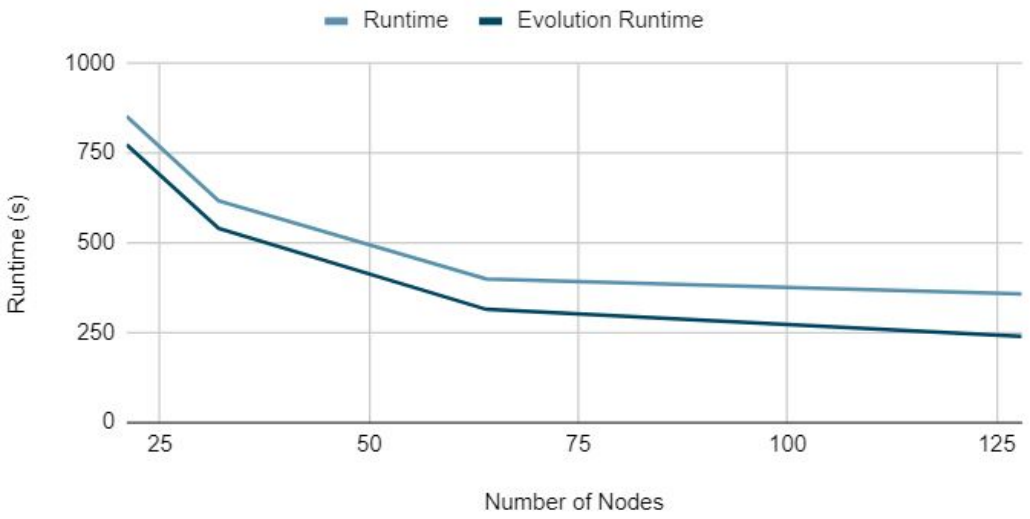

Figure 4. Strong scaling study for a full 3D simulation of a thermonuclear supernova with FLASH. The simulation had $\sim$ 33,000 blocks, and ran for 20 timesteps and 10 grid refinements. Shown are the full runtime and the evolution runtime which excludes I/O and initialization.

increases with the number of nodes in use. This sets an upper limit on the number of blocks that can be allocated to each processor. At the same time, for reasons we are still investigating, FLASH has issues with its guard blocks if too few blocks are allocated per processor. These opposing requirements meant that we needed to maximize the number of processors per node, and minimize the number of blocks allocated per process. We found a balance by allocating 50 blocks per process using 36 of the 48 available cores, which uses 26 GB of memory per node.

Using this configuration, we would be able to allocate a maximum of $\sim$316,000 blocks, or 4.7 TB, if we used the full machine. This limit is reached quickly in 3D, especially when we include the effects of a background turbulence field, a current work in progress, in our simulation. However, knowledge of this machine can lead to success on its larger counterpart, the supercomputer Fugaku, which has nearly 1000 times the number of nodes and memory than Ookami, along with a custom interconnect and additional assistant cores. This makes testing code on Ookami even more valuable, as understanding this testbed can lead to successful runs on more powerful machines with the same architecture.

4. Summary and Conclusions

Judging from the machine's use, the activity in Office Hours and the Slack channel, and the attendance at webinars, we conclude that the project has been successful in that it has indeed allowed many users with a variety of applications to meaningfully explore the Fujitsu A64FX processor. Our education and outreach efforts have been effective and have created a community of users that work well together.

Our performance studies for the two astrophysics codes produced mixed results. Both codes were readily ported to Ookami and ran with minimal adjustment. The use of SVE improved the performance of V2D but branching in the material EOS routines of FLASH prevented meaningful vectorization and thus there was no performance improvement. The use of hp with FLASH produced a dramatic decrease in DTLB misses but did not improve the performance of FLASH. This result suggests that DTLB misses do not adversely affect the performance of FLASH. We attribute this finding to the A64FX's translation table cache, which decreases the latency of virtual to physical address translation [26]. Finally, our results show that the best

performance for V2D and FLASH applications on the A64FX architecture of Ookami occurs with the use of the Fujitsu compiler.

While the relatively small available memory per node of Ookami presented some challenges, our strong scaling study with FLASH under production conditions demonstrated reasonable scaling. We conclude that with experimentation and tuning, the machine is able to perform production simulations for multi-scale, multi-physics applications with a variety of solvers and provide valuable experience for porting larger and more detailed simulations to the supercomputer Fugaku.

Acknowledgments
Ookami is supported by the US NSF grant #1927880, and this research was supported in part by the US DOE under grant DE-FG02-87ER40317. FLASH was developed in part by the US DOE NSA-ASC and OSC-ASCR-supported Flash Center for Computational Science at the University of Chicago. The authors gratefully acknowledge the generous support of the Ookami community. The authors also thank Jens Domke at RIKEN for very helpful suggestions.

References

[1] Matsuoka S 2021 Fugaku and a64fx: the first exascale supercomputer and its innovative arm cpu *2021 Symposium on VLSI Circuits* pp 1–3

[2] IACS 2020 Ookami homepage URL `https://www.stonybrook.edu/commcms/ookami/`

[3] Burford A, Calder A, Carlson D, Chapman B, Coskun F, Curtis T, Feldman C, Harrison R, Kang Y, Michalowicz B, Raut E, Siegmann E, Wood D, DeLeon R, Jones M, Simakov N, White J and Oryspayev D 2021 Ookami: Deployment and initial experiences *Practice and Experience in Advanced Research Computing* PEARC '21 (New York, NY, USA: Association for Computing Machinery) ISBN 9781450382922 URL `https://doi.org/10.1145/3437359.3465578`

[4] Swesty F D and Myra E S 2009 *ApJS* **181** 1–52

[5] Fryxell B, Olson K, Ricker P, Timmes F X, Zingale M, Lamb D Q, MacNeice P, Rosner R, Truran J W and Tufo H 2000 *The Astrophysical Journal Supplement Series* **131** 273–334

[6] Dubey A, Antypas K, Calder A, Fryxell B, Lamb D, Ricker P, Reid L, Riley K, Rosner R, Siegel A, Timmes F, Vladimirova N and Weide K 2013 The software development process of flash, a multiphysics simulation code *2013 5th International Workshop on Software Engineering for Computational Science and Engineering (SE-CSE)* pp 1–8

[7] 2023 NSF ACCESS program URL `https://access-ci.org`

[8] Mann B 2017 Arm architecture - armv8.2-a evolution and delivery URL `https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/arm-architecture-armv8-2-a-evolution-and-delivery`

[9] Siegmann E, Calder A, Feldman C and Harrison R J 2021 Educating hpc users in the use of advanced computing technology *2021 IEEE/ACM Ninth Workshop on Education for High Performance Computing (EduHPC)* pp 16–23

[10] 2023 Campus Champions URL `https://campuschampions.cyberinfrastructure.org/`

[11] 2023 Science Gateways URL `https://sciencegateways.org/`

[12] Smolarski D C, Swesty F and Calder A C 2022 Performance of an astrophysical radiation hydrodynamics code under scalable vector extension optimization *2022 IEEE International Conference on Cluster Computing (CLUSTER)* (Los Alamitos, CA, USA: IEEE Computer Society) pp 545–548 URL `https://doi.ieeecomputersociety.org/10.1109/CLUSTER51413.2022.00071`

[13] Townsley D M, Calder A C and Miles B J 2019 Modeling subgrid combustion processes in simulations of thermonuclear supernovae *Journal of Physics Conference Series* (*Journal of Physics Conference Series* vol 1225) p 012004 (*Preprint* 1908.06176)

[14] 2023 The Flash Center for Computational Science URL https://flash.rochester.edu

[15] O'Neal J, Weide K and Dubey A 2018 Experience report: Refactoring the mesh interface in flash, a multiphysics software *WSSSPE6.1, colocated with eScience 2018, Amsterdam, Netherlands*

[16] MacNeice P, Olson K M Mobarry C, de Fainchtein R and Packer C 1999 *NASA Tech. Rep. CR-1999-209483*

[17] MacNeice P, Olson K M Mobarry C, de Fainchtein R and Packer C 2000 *Comput. Phys. Commun.* **126** 330–354

[18] Dubey A, Calder A, Fisher R, Graziani C, Jordan G, Lamb D, Reid L, Townsley D and Weide K 2013 *International Journal of High Performance Computing Applications* **27** 360–373 ISSN 1094-3420

[19] Daley C, Bachan J, Couch S, Dubey A, Fatenejad M, Gallagher B, Lee D and Weide K 2012 Adding shared memory parallelism to FLASH for many-core architectures *TACC-Intel Highly Parallel Computing Symposium* poster

[20] Dubey A 2019 Programming abstractions for orchestration of hpc scientific computing https://chapel-lang.org/CHIUW2019.html keynote, Chapel User's Group Meeting

[21] Dubey A 2019 Dynamic resource management, an application perspective https://project.inria.fr/resourcearbitration/program/ invited talk, RADR, co-located with IPDPS

[22] Linaro 2021 Linaro map URL https://www.linaroforge.com/linaroMap/

[23] Feldman C, Michalowicz B, Siegmann E, Curtis T, Calder A and Harrison R 2022 Experiences with porting the flash code to ookami, an hpe apollo 80 a64fx platform *International Conference on High Performance Computing in Asia-Pacific Region Workshops* HPCAsia 2022 Workshop (New York, NY, USA: Association for Computing Machinery) p 72–77 ISBN 9781450395649 URL https://doi.org/10.1145/3503470.3503478

[24] 2022 Performance Application Programming Interface URL http://icl.cs.utk.edu/papi/

[25] Feldman C, Gutierrez N, Eisenberg E, Willcox D E, Townsley D M and Calder A C 2023 *ApJ* **959** 112 (*Preprint* 2309.07283)

[26] Fujitsu 2023 A64fx microarchitecture manual URL https://github.com/fujitsu/A64FX/blob/master/doc/A64FX_Microarchitecture_Manual_en_1.3.pdf

[27] Calder A C, Feldman C, Siegmann E, Dey J, Curtis A, Chheda S and Harrison R J 2022 On using linux kernel huge pages with flash, an astrophysical simulation code *2022 IEEE International Conference on Cluster Computing (CLUSTER)* (Los Alamitos, CA, USA: IEEE Computer Society) pp 539–544 URL https://doi.ieeecomputersociety.org/10.1109/CLUSTER51413.2022.00070

[28] Feldman C, Chheda S, Calder A, Siegmann E, Dey J, Curtis T and Harrison R 2023 A further study of linux kernel hugepages on a64fx with flash, an astrophysical simulation code *Practice and Experience in Advanced Research Computing* PEARC '23 (New York, NY, USA: Association for Computing Machinery) ISBN 9781450382922 URL https://doi.org/10.1145/3569951.3597583