Experiences with Porting the FLASH Code to Ookami, an HPE Apollo 80 A64FX Platform

CATHERINE FELDMAN, BENJAMIN MICHALOWICZ, EVA SIEGMANN, TONY CURTIS, ALAN C. CALDER, and ROBERT J. HARRISON, Institute for Advanced Computational Science, Stony Brook University, USA

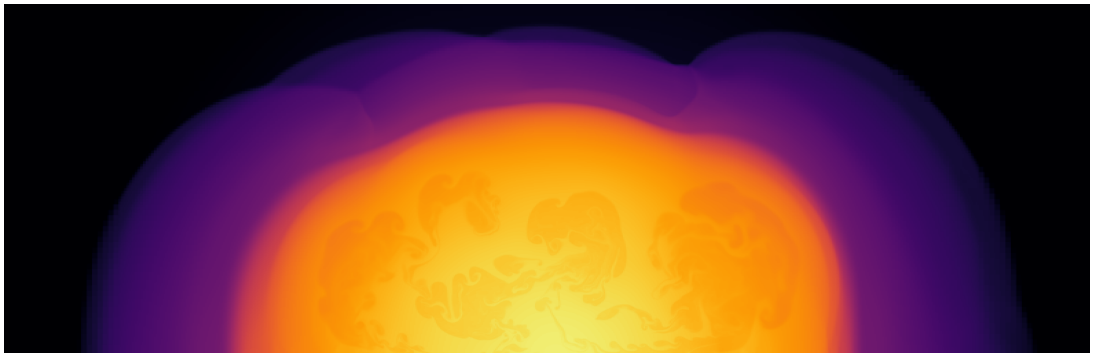

Fig. 1. Density distribution from a FLASH simulation showing a deflagration developing inside a white dwarf.

We present initial experiences with running the community simulation code FLASH, developed at the University of Chicago for multi-scale multi-physics applications, on Ookami, a technology testbed featuring the A64FX processor developed by Fujitsu. Our effort focused largely on running FLASH "right out of the box" to see which combinations of compilers and software implementations (e.g. MPI) allowed the code to run with minimal modification. FLASH was one application in a larger effort to deploy Ookami; it served as a test for different versions of newly installed software, and as a cornerstone for the FAQ page of the Ookami website.

We report on our results with different compilers and other software, along with our initial scaling results and attempts to utilize the A64FX's SVE instructions and NUMA architecture. We found that FLASH readily ran with different compilers and MPI implementations, and showed the expected good scaling with no turning. However, more work must be done to fully take advantage of the A64FX's architectural features and produce a significant speedup for FLASH on Ookami.

CCS Concepts: • **General and reference**; • **Hardware**; • **Applied computing** → **Physical sciences and engineering**; **Astronomy**;

Additional Key Words and Phrases: high-performance computing, A64FX, porting software, astrophysics

ACM Reference Format:

Catherine Feldman, Benjamin Michalowicz, Eva Siegmann, Tony Curtis, Alan C. Calder, and Robert J. Harrison. 2022. Experiences with Porting the FLASH Code to Ookami, an HPE Apollo 80 A64FX Platform. In *International Conference on High Performance Computing in Asia-Pacific Region Workshops (HPCAsia 2022 Workshop), January 11–14, 2022, Virtual Event, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3503470.3503478

1 INTRODUCTION

1.1 The Ookami Testbed

Ookami [3] is a testbed supported by the United States National Science Foundation (NSF) and run by teams at Stony Brook University and the University at Buffalo that provide access to state-of-the-art scientific computing technology. The system has 174 A64FX Fujitsu compute nodes, each with 32GB high-bandwidth memory and a 512GB SSD. These processors promise to combine familiar and successful programming models with very high performance for a wide range of applications [16]. They provide 2–4 times better performance on memory-intensive applications.

Ookami is the first open machine outside of Japan that uses the A64FX processor, the same processor deployed in the world's fastest machine: Fugaku at the RIKΞN Center for Computational Science in Japan [25]. Fugaku is not only the number one in the Top-500 ranking, but also tops the charts in the major benchmarks HPCG, HPL-AI, and Graph-500 [24], and ranks in the top 10 on the Green 500 [28]. The A64FX promises to have high performance and reliability as well as a good performance to power ratio. The A64FX 700 series, used in Ookami, consists of four core memory groups (CMGs) each with 12 cores, 64KB L1 cache, and 8MB L2 cache shared between the cores and runs at 1.8 GHz. The processor uses the ARMv8.2–A Scalable Vector Extension (SVE) SIMD instruction set with 512 bit vector implementation, allowing for vector lengths anywhere from 128–2048 bits and enabling vector length agnostic programming.

The aim of the Ookami project is to explore this technology and demonstrate its potential. For this purpose, various codes are being ported and tuned on the machine, including our application, the FLASH code. More details on the system can be found in the PEARC 2021 paper "Ookami: Deployment and Initial Experiences" [4].

1.2 The FLASH Code

The FLASH code is a component-based scientific simulation software package with a wide user base addressing a variety of multi-scale, multi-physics applications [10–12, 15]. FLASH was developed at the Flash Center for Computational Science at the University of Chicago and has a long history of deployment on advanced computing platforms. Originally funded by the Department of Energy's Accelerated Strategic Computing Initiative [22], the Flash Center had access to unclassified versions or partitions of the platforms installed at the Defense Program National Laboratories as they became available [26]. Highlights include winning the SC2000 Gordon Bell Prize, Special Category, for reactive fluid flow simulations using AMR that achieved 238 GFlops on 6420 processors of ASCI Red at LANL [5], simulating weakly compressible stirred-turbulence at an unprecedented resolution on the IBM BG/L machine commissioned at LLNL in 2005 when it was briefly available for unclassified use [1, 13], and serving as one of the formal acceptance tests for Intrepid, the IBM BG/P, and MIRA, the IBM BG/Q machines at the Argonne Leadership Computing Facility [12].

The FLASH code was originally written to address astrophysical problems involving 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 both for high-energy-density physics and more general problems as FLASH-X, a new code derived from FLASH with a completely new infrastructure, being funded by the U.S. Department of Energy's Exascale Computing Project [23].

FLASH uses Adaptive Mesh Refinement (AMR) to address problems with a wide range of physical and temporal scales by placing higher resolution only where needed. The current release of FLASH (version 4.6.2) implements AMR using the PARAMESH library [20, 21], which uses a block-structured mesh. FLASH is parallelized primarily through

MPI, although some solvers have been modified to take advantage of threaded approaches to parallelization [12] and development toward a more general design for better allowing threading continues [7–9].

Our application is an explosive astrophysical event known as a thermonuclear/Type Ia supernova, events thought to result from a thermonuclear explosion disrupting a compact star known as a white dwarf. Our study investigates one variation, a very dim event known as a Type Iax that is thought to be produced by a pure deflagration (subsonic burning front) occurring in a special “hybrid" white dwarf [14, 19]. The vast disparity between the radius of the white dwarf ($\sim 10^9$ cm) and the width of the laminar nuclear flame at high densities (< 1 cm) presents a significant challenge because even with many orders of AMR, it is not possible to resolve the physical flame in a whole-star simulation. Modeling these events thus necessitates the use of a model flame. The extended version of FLASH we use propagates an artificially broadened flame with an advection-diffusion-reaction (ADR) scheme [31] that propagates reaction progress variables describing the stages of the flame. Flame speeds are from previously-calculated tabular results [6, 27] and also include enhancement to the burning rate from unresolved buoyancy and background turbulence [17, 18, 29]. The results we describe here were for preliminary two-dimensional simulations allowing for a relatively inexpensive exploration of both the Ookami platform and the parameter space of the astrophysics problem.

2 METHODS

Using our supernova simulation as a test problem, our first task was to identify the prerequisites of FLASH, e.g. MPI and HDF5, and determine how those are accessed and configured on Ookami. We tested combinations of compilers and MPI implementations to see if FLASH could be ported to Ookami and which combination produced the best performance.

Both the GCC 10.3.0 and ARM 21.0 compilers have flags that allow users to specifically target the A64FX processor. SVE can be enabled by adding the `-O3` flag, which enables loop optimization options that are necessary for SVE instructions, and `-mcpu=a64fx`, which tells the compiler to use specific hardware features of the specified architecture, e.g., SVE, gather/scatter, and fast partial dot product. The Cray compilers, on the other hand, have separate module files for choosing between non-SVE and SVE instructions. Loading the `CPE` module on Ookami loads the A64FX-targeted Cray compiler and dependencies, which enables SVE instructions automatically. To compile without including SVE instructions, we instead load the `CPE-nosve` module, which does not target the A64FX hardware, and generates ARM-Neon code. These options are summarized below in Table 1. Note that this table reflects the state of software installations on Ookami at the time when experiments were conducted.

Table 1. Summary of compilers, compiler flags, MPI implementations, and additional flags required to generate SVE instructions with the FLASH code. Compiler flags were chosen so that reals were promoted to double-precision, doubles were kept at double-precision, and integers remained at single precision, options which are necessary for FLASH.

Compiler	Compiler Flags	MPI Implementation	Additional SVE Flags
GCC 10.3.0	`-fdefault-real-8 -Wuninitialized -fdefault-double-8 -fallow-argument-mismatch`	MVAPICH 2.3.5 Open-MPI 4.0.5	`-O3 -mcpu=a64fx`
Cray 10.0.3	`-O3 -h vector3 -s real64 -s integer32`	MVAPICH 2.3.5	Load the Cray 10.0.1 SVE module
ARM 21.0	`-r8 -armpl`	MVAPICH 2.3.5 Open-MPI 4.0.5	`-O3 -mcpu=a64fx`

The memory per core is indirectly tuned by setting the maximum number of grid blocks that can be allocated per core. Due to the relatively small 32 GB of memory per node, it was necessary to adjust this from the 10,000 blocks per

core that we use on Stony Brook's Intel-based cluster, SeaWulf [30], to 2,500 on Ookami. The GCC compiler proved to be the easiest to use and the least sensitive to memory constraints. Running with too high a setting of blocks per core produced incredibly slow runs with the GCC compiler, but simply crashed with the Cray and ARM compilers.

After fitting our problem into memory, we performed runtime comparisons using the available compilers and MPI implementations, compared performance for different MPI bindings and layouts, performed a strong scaling study, profiled our code, and looked into the challenge of introducing SVE instructions into FLASH. To check for accuracy, the final integrated quantities of our simulation (mass, element yields, energy, etc.) were compared to those of an initial run, to ensure that any changes we made did not affect the results of the simulation.

3 RESULTS

3.1 Compiler Comparison

Figure 2 compares the runtime of our supernova simulation between the compilers described in Table 1. Our simulation was run on 5 nodes at full subscription (48 cores per node) for 4 seconds of simulation time. The Cray compiler with enabled SVE instructions produced the fastest runtime, with the GCC compiler producing the second-fastest running code. The ARM compiler produced runs almost three times as long as the others. In general, runs using MVAPICH are slightly faster. All three compilers successfully produced an executable with SVE instructions, however the difference in runtime is not significant as we would hope. This indicates that our code must be manually tuned to include SVE instructions at the bottlenecks.

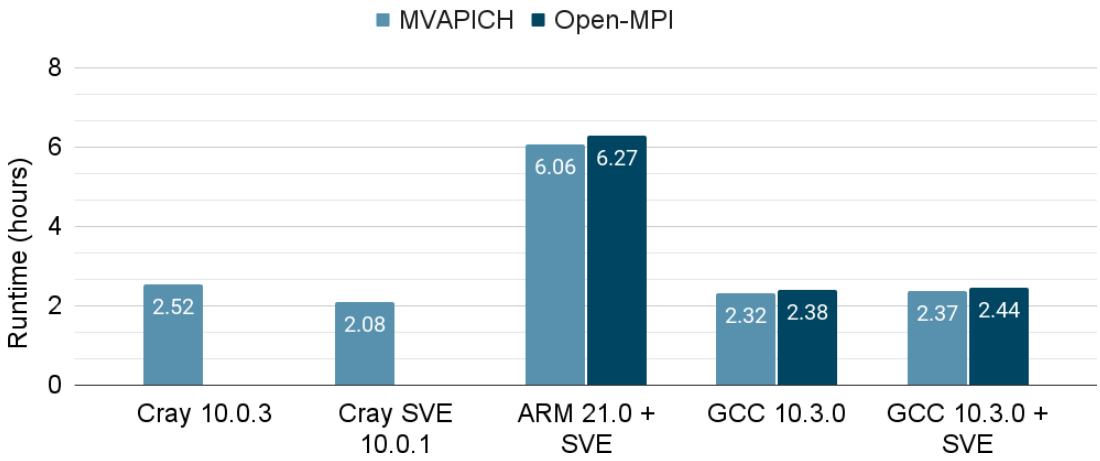

Fig. 2. Runtime comparison of our supernova simulation using the Cray, GCC, and ARM compilers, the two MPI implementations MVAPICH and Open-MPI, and SVE instructions.

3.2 Strong Scaling Study

We performed a strong scaling study of our supernova simulation using the GCC 10.3.0 compiler with GCC-compiled MVAPICH 2.3.5. We ran our problem for 4 seconds of simulation time on 12 and 24 cores of a single node and 1-10 nodes at full subscription (48 cores per node).

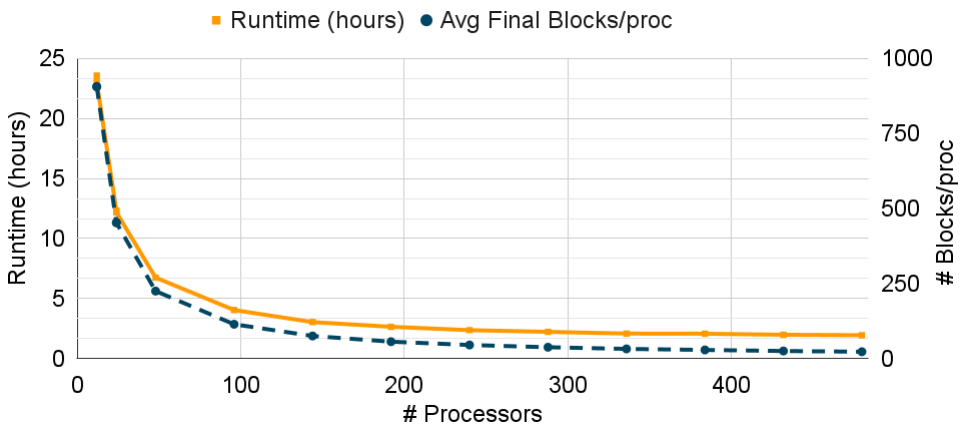

Fig. 3. Runtime vs number of cores on Ookami is shown in the solid orange line. Number of blocks per core is shown in the dotted blue line. The simulation has not been optimized - this serves as a demonstration that parallelism is feasible.

As shown in Figure 3, increasing the number of cores decreases the total simulation time as expected. However, this speedup tapers off. From runs using 12 to 96 cores, doubling the number of cores cuts runtime in half, demonstrating almost perfect scaling. Increasing from 12 to 96 cores decreases the runtime by 82.4%. Adding more cores after this point (up to 480) decreases the runtime by a further 10%, showing that each subsequent increase in the number of cores contributes less and less to this percentage.

While increasing the number of cores used, the number of grid blocks handled by each core decreases. While this decreases computation time, we will see an increase in communication. However, we do not see turning in our scaling graph, indicating that the decrease in computation time still outweighs the increase in communication time.

An important note is that increasing the number of cores used increases the time needed to write to an output file. However, FLASH allows adjusting the cadence of its output files, and we can take care to only save what we need.

3.3 MPI binding and placement

Because FLASH primarily uses point-to-point communication through `MPI_Send` and `MPI_Recv` calls, the way in which MPI processing elements (PE) are placed influences communication overhead and runtime. When possible, specific mapping flags were used at runtime to control PE placement—on the A64FX, these can be placed by core, by node, or by CMG, either in a cyclic (evenly fill CMGs) or block (fill one CMG at a time) fashion. To test this, we ran our supernova simulation with the GCC 10.3.0 compiler on 4 nodes and 96 cores, with different placements of 24 cores per node.

Table 2. Runtime of our simulation for different MPI rank placements.

MPI Implementation	Nodes	Cores	MPI Placement	Runtime (hours)
MVAPICH 2.3.5	4	96	Block	2.79
	4	96	Cyclic	2.80
Open-MPI 4.1.0	4	96	Block	2.86
	4	96	Cyclic	2.73
MVAPICH 2.3.5	2	96	Block	4.05
	4	192	Block	2.65

Initial results, shown in Table 2, show that the runs using Open-MPI show a greater difference due to a change in mapping than those using MVAPICH. It is also interesting to note that using 96 cores of 4 partially-filled nodes is much faster than using 96 cores of 2 completely filled nodes, and also only slightly slower than using all of the 192 cores on 4 full nodes. More study is needed in order to fully optimize communication pathways.

3.4 Profiling and attempt at using SVE

We profiled our code to find bottlenecks that could potentially be addressed with the use of SVE instructions through ARM MAP [2]. We found that our code spends half of its time in the hydrodynamics routines, and within that, 20% of the total computation time is spent calculating the equation of state (EOS) for each cell in a grid block. This became our first spot to try and improve through SVE instructions.

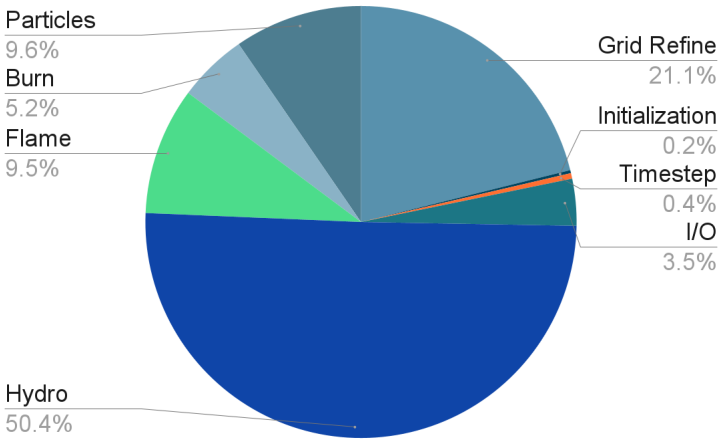

Fig. 4. Runtime breakdown of our FLASH supernova simulation. Half of the time spent is within the hydrodynamics routines.

The vectorization of the EOS routine remains a work in progress. The EOS implements the physics of the stellar plasma by calculating derived thermodynamic quantities from those evolved by the hydrodynamics, e.g. temperature as a function of density and composition. Accordingly, the routine contains math functions, nested if statements, tabular lookups, and calls to condition-checking and I/O routines. The latter needed to be taken out of the loop as they checked variables for physical significance and aborted the code if necessary, an action that cannot be vectorized. As the GCC 10 compiler is unable to vectorize certain math functions including `exp` and `pow` on its own, our code will have to either be linked to the ARM performance libraries (`-L<ARMPL_install_dir>/lib -lamath -larmpl_lp64 -lm`) or use the Cray compiler to fully vectorize. We are working on rewriting the loop into a vectorizeable form, an arduous process.

3.5 Comparison to SeaWulf

The SeaWulf cluster sports 328 Intel Xeon E5-2683v3 CPUs that operate at 2.0 GHz and have 128 GB of DDR4 memory (16GB reserved for the system). The system is configured with 2 CPUs per node for a total of 24 cores per node. Our problem was compiled on Seawulf with the GCC 10.2.0 compiler using MVAPICH 2.3.5, and run for four seconds of simulation time on 10 nodes, for a total of 240 cores. SeaWulf produced code that ran almost three times as fast (0.77

hours) than that on Ookami using a similar compiler and the same number of cores (2.3 hours). The difference in clock speeds (1.8 GHz on Ookami; 2.0 GHz on SeaWulf) is not enough to account for this difference, suggesting that FLASH must be tuned to use the A64FX's hardware features, especially HBM, to their full potential.

4 SUMMARY

Optimizing the FLASH code for use on Ookami's A64FX architecture will allow us to achieve our science goal — a full three-dimensional simulation of a Type Iax supernova. The two-dimensional simulation used for this study showed us what improvements we need to make, and what areas require additional study.

Without making any adjustments to the code, our FLASH supernova problem successfully compiled and ran with three different compilers using two MPI implementations. Use of SVE requires further experimentation, especially within the EOS routine that takes up 20% of the total computation time. To vectorize this routine, we will use what we have learned to tune the code, and test both the easy-to-use GCC compiler with the ARM performance libraries as well as the the Cray SVE compiler.

FLASH demonstrates strong scaling across nodes; however, as more nodes are added, this speedup rate decreases. Our problem also shows a sensitivity to how MPI PEs are mapped to the A64FX's CMGs. A closer examination of both FLASH's communication pattern and memory use will allow us to find the optimized MPI PE mapping for both intra/inter-node communication, as well as tune the code to take advantage of Ookami's HBM. An advantage of FLASH is that a run can be paused and then restarted on a different number of nodes; this allows us to both increase the number of nodes as the problem grows in size as the star explodes, and adjust the mapping as needed throughout a run.

ACKNOWLEDGMENTS

Ookami is a computer technology testbed supported by the National Science Foundation under grant OAC 1927880. The authors are grateful to the Ookami team for their efforts procuring and deploying the machine, and for the arduous process of setting up the software used. The authors would like to thank Research Computing and Cyberinfrastructure, and the Institute for Advanced Computational Science, at Stony Brook University for access to the high-performance SeaWulf cluster, which was made possible by a $1.4M National Science Foundation grant (#1531492). The FLASH code was developed in part by the DOE NNSA ASC- and DOE Office of Science ASCR-supported Flash Center for Computational Science at the University of Chicago. Work involving supernovae research was supported in part by the US Department of Energy under grant DE-FG02-87ER40317.

REFERENCES

[1] Katie Antypas, Alan C. Calder, Anshu Dubey, Robert T. Fisher, Murali K. Ganapathy, Brad Gallagher, Lynn B. Reid, Katherine Riley, Daniel J. Sheeler, and Noel T. Taylor. 2006. Scientific Applications on the Massively Parallel BG/L Machine. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications & Conference on Real-Time Computing Systems and Applications, PDPTA 2006, Las Vegas, Nevada, USA, June 26-29, 2006, Volume 1*, Hamid R. Arabnia (Ed.). CSREA Press, 292–298.

[2] ARM. 2021. *ARM Forge/MAP.* https://www.arm.com/products/development-tools/server-and-hpc/forge/map

[3] IACS Stony Brook. 2020. *Ookami.* https://www.stonybrook.edu/commcms/ookami/

[4] A. Burford, A. Calder, D. Carlson, B. Chapman, F. Coskun, T. Curtis, C. Feldman, R. Harrison, Y. Kang, B. Michalowicz, E. Raut, E. Siegmann, D. Wood, R. DeLeon, M. Jones, N. Simakov, J. White, and D. Oryspayev. 2021. Ookami: Deployment and Initial Experiences. In *Practice and Experience in Advanced Research Computing* (Boston, MA, USA) *(PEARC '21)*. Association for Computing Machinery, New York, NY, USA, Article 9, 8 pages. https://doi.org/10.1145/3437359.3465578

[5] A. C. Calder, B. C. Curtis, L. J. Dursi, B. Fryxell, G. Henry, P. MacNeice, K. Olson, P. Ricker, R. Rosner, F. X. Timmes, H. M. Tufo, J. W. Truran, and M. Zingale. 2000. High-Performance Reactive Fluid Flow Simulations Using Adaptive Mesh Refinement on Thousands of Processors. In *Proceedings of Supercomputing 2000*. http://sc2000.org.

[6] David A. Chamulak, Edward F. Brown, and Francis X. Timmes. 2007. The Laminar Flame Speedup by ^{22}Ne Enrichment in White Dwarf Supernovae. *The Astrophysical Journal* 655 (February 2007), L93. arXiv:astro-ph/0612507

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

[8] A. Dubey. 2019. Dynamic Resource Management, An Application Perspective. https://project.inria.fr/resourcearbitration/program/. Invited talk, RADR, co-located with IPDPS.

[9] Anshu Dubey. 2019. Programming Abstractions for Orchestration of HPC Scientific Computing. https://chapel-lang.org/CHIUW2019.html. Keynote, Chapel User's Group Meeting.

[10] Anshu Dubey, Katie Antypas, Alan C. Calder, Chris Daley, Bruce Fryxell, J. Brad Gallagher, Donald Q. Lamb, Dongwook Lee, Kevin Olson, Lynn B. Reid, Paul Rich, Paul M. Ricker, Katherine M. Riley, Robert Rosner, Andrew Siegel, Noel T. Taylor, Klaus Weide, Francis X. Timmes, Natasha Vladimirova, and John Zuhone. 2014. Evolution of FLASH, a multi-physics scientific simulation code for high-performance computing. *International Journal of High Performance Computing Applications* 28, 2 (2014), 225–237. https://doi.org/10.1177/1094342013505656

[11] A. Dubey, K. Antypas, A. Calder, B. Fryxell, D. Lamb, P. Ricker, L. Reid, K. Riley, R. Rosner, A. Siegel, F. Timmes, N. Vladimirova, and K. Weide. 2013. The Software development process of FLASH, a multiphysics simulation code. In *2013 5th International Workshop on Software Engineering for Computational Science and Engineering (SE-CSE)*. 1–8. https://doi.org/10.1109/SECSE.2013.6615093

[12] Anshu Dubey, Alan C. Calder, Robert Fisher, Carlo Graziani, G. C. Jordan, Donald Q. Lamb, Lynn B. Reid, Dean M. Townsley, and Klaus Weide. 2013. Pragmatic optimizations for better scientific utilization of large supercomputers. *International Journal of High Performance Computing Applications* 27, 3 (2013), 360–373. https://doi.org/10.1177/1094342012464404

[13] R. Fisher, S. Abarzhi, S. M. Antypas, K.and Asida, A. C. C alder, F. Cattaneo, P. Constantin, A. Dubey, I. Foster, J. B. Gallagher, M. K. Ganapath, C. C. Glendenin, L. Kadanoff, D. Q. Lam b, S. Needham, M. Papka, T. Plewa, L.B. Reid, P. Rich, K. Riley, and D. Sheeler. 2008. Tera-scale Turbulence Computation on BG/L Using the FLASH3 Code. *IBM J. Res. & Dev.* 52 (jan 2008), 127–136. https://doi.org/10.1147/rd.521.0127

[14] Ryan J. Foley, P. J. Challis, R. Chornock, M. Ganeshalingam, W. Li, G. H. Marion, N. I. Morrell, G. Pignata, M. D. Stritzinger, J. M. Silverman, X. Wang, J. P. Anderson, A. V. Filippenko, W. L. Freedman, M. Hamuy, S. W. Jha, R. P. Kirshner, C. McCully, S. E. Persson, M. M. Phillips, D. E. Reichart, and A. M. Soderberg. 2013. TYPE Iax SUPERNOVAE: A NEW CLASS OF STELLAR EXPLOSION. *The Astrophysical Journal* 767, 1 (mar 2013), 57. https://doi.org/10.1088/0004-637x/767/1/57

[15] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo. 2000. FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. *The Astrophysical Journal Supplement Series* 131 (2000), 273–334.

[16] Fujitsu. 2021. *A64FX*. https://www.fujitsu.com/global/products/computing/servers/supercomputer/a64fx/

[17] A. P. Jackson, D. M. Townsley, and A. C. Calder. 2014. Power-law Wrinkling Turbulence-Flame Interaction Model for Astrophysical Flames. *The Astrophysical Journal* 784, Article 174 (April 2014), 174 pages. https://doi.org/10.1088/0004-637X/784/2/174 arXiv:1402.4527 [astro-ph.SR]

[18] A. M. Khokhlov. 1995. Propagation of Turbulent Flames in Supernovae. *The Astrophysical Journal* 449 (1995), 695.

[19] M. Kromer, S. T. Ohlmann, R. Pakmor, A. J. Ruiter, W. Hillebrandt, K. S. Marquardt, F. K. Röpke, I. R. Seitenzahl, S. A. Sim, and S. Taubenberger. 2015. Deflagrations in hybrid CONe white dwarfs: a route to explain the faint Type Iax supernova 2008ha. *Monthly Notices of the Royal Astronomical Society* 450, 3 (05 2015), 3045–3053. https://doi.org/10.1093/mnras/stv886 arXiv:https://academic.oup.com/mnras/article-pdf/450/3/3045/18513746/stv886.pdf

[20] P. MacNeice, C. Olson, K. M. Mobarry, R. de Fainchtein, and C. Packer. 1999. PARAMESH: A Parallel Adaptive Mesh Refinement Community Toolkit. *NASA Tech. Rep. CR-1999-209483* (1999).

[21] P. MacNeice, C. Olson, K. M. Mobarry, R. de Fainchtein, and C. Packer. 2000. PARAMESH: A parallel adaptive mesh refinement community toolkit. *Comput. Phys. Commun.* 126 (2000), 330–354.

[22] USDOE Office of Defense Programs. 2000. Accelerated Strategic Computing Initiative (ASCI) Program Plan [FY2000]. (1 2000). https://doi.org/10.2172/768266

[23] Jared O'Neal, Klaus Weide, and Anshu Dubey. 2018. Experience Report: Refactoring the Mesh Interface in FLASH, a Multiphysics Software. In *WSSSPE6.1, colocated with eScience 2018, Amsterdam, Netherlands*. https://doi.org/10.6084/m9.figshare.7093199

[24] RIKEN. 2020. *Fugaku performance award*. https://www.riken.jp/en/news_pubs/news/2020/20200623_1/

[25] RIKEN. 2021. *Fugaku supercomputer*. https://www.r-ccs.riken.jp/en/fugaku

[26] R. Rosner, A. C. Calder, L. J. Dursi, B. Fryxell, D. Q. Lamb, J. C. Niemeyer, K. Olson, P. Ricker, F. X. Timmes, J. W. Truran, H. Tufo, Y. Young, M. Zingale, E. Lusk, and R. Stevens. 2000. Flash Code: Studying Astrophysical Thermonuclear Flashes. *Computing in Science and Engineering* 2 (March 2000), 33.

[27] F. X. Timmes and S. E. Woosley. 1992. The Conductive Propagation of Nuclear Flames. I - Degenerate C + O and O + Ne + Mg White Dwarfs. *The Astrophysical Journal* 396 (1992), 649.

[28] TOP500.org. 2020. *Green500 November 2020*. https://www.top500.org/lists/green500/2020/11/

[29] D. M. Townsley, A. C. Calder, S. M. Asida, I. R. Seitenzahl, F. Peng, N. Vladimirova, D. Q. Lamb, and J. W. Truran. 2007. Flame Evolution During Type Ia Supernovae and the Deflagration Phase in the Gravitationally Confined Detonation Scenario. *The Astrophysical Journal* 668 (Oct. 2007), 1118–1131. arXiv:arXiv:0706.1094

[30] Stony Brook University. 2016. *Understanding SeaWulf*. https://it.stonybrook.edu/help/kb/understanding-seawulf

[31] N. Vladimirova, G. Weirs, and L. Ryzhik. 2006. Flame capturing with an advection-reaction-diffusion model. *Combust. Theory Modelling* 10, 5 (2006), 727–747.