SoftwareX 19 (2022) 101168

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Flash-X: A multiphysics simulation software instrument

Anshu Dubey [a,h,*], Klaus Weide [h,a], Jared O'Neal [a], Akash Dhruv [a,f], Sean Couch [c], J. Austin Harris [b], Tom Klosterman [a], Rajeev Jain [a], Johann Rudi [a], Bronson Messer [b,l], Michael Pajkos [c], Jared Carlson [c], Ran Chu [l], Mohamed Wahib [n], Saurabh Chawdhary [a], Paul M. Ricker [d], Dongwook Lee [e], Katie Antypas [g], Katherine M. Riley [a], Christopher Daley [g], Murali Ganapathy [i], Francis X. Timmes [j], Dean M. Townsley [m], Marcos Vanella [k], John Bachan [g], Paul M. Rich [a], Shravan Kumar [h], Eirik Endeve [b,l], W. Raphael Hix [b,l], Anthony Mezzacappa [l], Thomas Papatheodore [b]

[a] *Argonne National Laboratory, Lemont, IL, 60439, USA*
[b] *Oak Ridge National Laboratory, Oak Ridge, TN, 37831, USA*
[c] *Michigan State University, USA*
[d] *University of Illinois, Urbana Champaign, USA*
[e] *University of California, Santa Cruz, USA*
[f] *George Washington University, USA*
[g] *Lawrence Berkeley National Laboratory, USA*
[h] *University of Chicago, USA*
[i] *Google Inc, USA*
[j] *Arizona State University, USA*
[k] *National Institute of Standards and Technology, USA*
[l] *University of Tennessee, Knoxville, TN, 37996, USA*
[m] *University of Alabama, Tuscaloosa, AL, 35487, USA*
[n] *RIKEN BNL Research Center, USA*

ARTICLE INFO

Article history:
Received 10 May 2022
Received in revised form 12 July 2022
Accepted 21 July 2022

Keywords:
Multiphysics
Simulation software
High-performance computing
Performance portability

ABSTRACT

Flash-X is a highly composable multiphysics software system that can be used to simulate physical phenomena in several scientific domains. It derives some of its solvers from FLASH, which was first released in 2000. Flash-X has a new framework that relies on abstractions and asynchronous communications for performance portability across a range of increasingly heterogeneous hardware platforms. Flash-X is meant primarily for solving Eulerian formulations of applications with compressible and/or incompressible reactive flows. It also has a built-in, versatile Lagrangian framework that can be used in many different ways, including implementing tracers, particle-in-cell simulations, and immersed boundary methods.

* Correspondence to: Argonne National Laboratory, 9600 S. Cass Ave, Lemont, IL, 60439, USA.

E-mail addresses: adubey@anl.gov (Anshu Dubey), kweide@uchicago.edu (Klaus Weide), joneal@anl.gov (Jared O'Neal), adhruv@anl.gov (Akash Dhruv), scouch@msu.edu (Sean Couch), harrisja@ornl.gov (J. Austin Harris), tklosterman@anl.gov (Tom Klosterman), jain@anl.gov (Rajeev Jain), bronson@ornl.gov (Bronson Messer), mapajkos@gmail.com (Michael Pajkos), jaredc.scholar@gmail.com (Jared Carlson), rchu@vols.utk.edu (Ran Chu), saurabh.chawdhary@gmail.com (Saurabh Chawdhary), pmricker@illinois.edu (Paul M. Ricker), dlee79@ucsc.edu (Dongwook Lee), kantypas@lbl.gov (Katie Antypas), riley@alcf.anl.gov (Katherine M. Riley), csdaley@lbl.gov (Christopher Daley), murali@google.com (Murali Ganapathy), fxt44@mac.com (Francis X. Timmes), dean.m.townsley@ua.edu (Dean M. Townsley), marcos.vanella@nist.gov (Marcos Vanella), john.bachan@gmail.com (John Bachan), richp@alcf.anl.gov (Paul M. Rich), shravan2915@gmail.com (Shravan Kumar), endevee@ornl.gov (Eirik Endeve), raph@ornl.gov (W. Raphael Hix), mezz@tennessee.edu (Anthony Mezzacappa), papatheodore@ornl.gov (Thomas Papatheodore).

https://doi.org/10.1016/j.softx.2022.101168

Code metadata

Current code version	1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00102
Code capsule	https://github.com/Flash-X/Workflows/tree/main/incompFlow/FlowBoiling
Legal Code License	Apache 2.0
Code versioning system used	git
Software code languages, tools, and services used	Fortran, C, C++, Python3, MPI, OpenMP, OpenACC, HDF5
Compilation requirements, operating environments and dependencies	Unix, Linux, OSX based compilers for languages and libraries mentioned above
Developer documentation/manual	https://flash-x.org/pages/documentation/
Support email for questions	flash-x-users@lists.cels.anl.gov

1. Motivation and significance

Flash-X [1] is a new incarnation of FLASH [2,3], a multiphysics software system that has been used by multiple science communities. Flash-X is meant for use beyond existing FLASH science communities. It is designed to be easily adaptable for use by any computational scientists who rely upon differential equations as their primary mathematical model with finite-volume or finite-difference discretization. FLASH was designed only for a homogeneous, distributed-memory parallel model with bulk-synchronism, which has rendered it unsuitable for use on many newer system architectures that are heavily reliant on disparate memory spaces (e.g., accelerators). This difficulty is further exacerbated by increasing heterogeneity in hardware as well as solvers within the code. Flash-X has a fundamentally redesigned architecture that uses abstractions and asynchronous operations for performance portability across a variety of platforms, both with and without accelerators. Our design is forward-looking in that it makes minimal assumptions about which parallelization or memory models are likely to be prevalent in future platforms. The design relies upon self-describing code components of varying granularity and a toolchain that can interpret the metadata of the code components to synthesize application instances. The synthesis is done partly through assembly, partly through code translation, and partly through code generation. Some code assembly features have been imported from FLASH, but have been significantly enhanced to discretize components at a finer scope than subroutines or functions. Tools for code translation and runtime management are new and will enable orchestration of computation and data movement between distinct compute devices on a node.

In addition to the new architecture, Flash-X has newer and higher-fidelity physics solvers. Most notable among these are Spark [4] for magnetohydrodynamics, XNet [5,6] for nuclear burning, thornado [7,8] for neutrino radiation transport, and WeakLib [9–11] for tabulated microphysics. Additionally, Flash-X can support multiphase flow through a level-set method, which did not exist in FLASH releases [12]. Flash-X has been exercised on small clusters at Argonne National Laboratory and on leadership-class machines at Oak Ridge National Laboratory and Argonne National Laboratory. Flash-X will showcase the key performance parameters of ExaStar [13], a project under the Exascale Computing Project [14,15] (ECP), through a core-collapse supernova (CCSN) simulation on exascale machines to be deployed by the US Department of Energy. To run effectively at scale, Flash-X will rely upon the toolchain described above. Some components of the toolchain are embedded in Flash-X, while others are encapsulated into independent libraries that can be used by other codes. Note that compilation and execution of the code do not require using these external libraries; they are used only to orchestrate data movement and computation for better performance.

Along with a new architecture, Flash-X also adopts a community-based, open development model. The stewardship of the code is guided by a Council representing all the major science communities of FLASH/Flash-X. More details of our community development model are available at https://flash-x.org.

2. Software description

The Flash-X code is a component-based software system for simulation of multiphysics applications that can be formulated largely as a collection of partial and ordinary differential equations (PDEs and ODEs), as well as algebraic equations. The equations are discretized and solved on a domain that can have uniform resolution (UG) or adaptive mesh refinement (AMR). In Flash-X, one can select between PARAMESH [16], an octree-based library written in Fortran, or AMReX [17,18], a highly-flexible, patch-based, C++ AMR library. Both AMR frameworks can interface to math libraries such as hypre [19] and PETSc [20], making those solvers available to Flash-X. Physics units are designed to be oblivious of domain decomposition. Bulk of their code is written for block-by-block update, interspersed with invocation of fine-coarse boundary resolution related API functions of the Grid unit as needed.

Hyperbolic equations are solved using explicit methods commonly used for compressible flows with strong shocks, described in Section 2.2. For elliptic equations, one can either use an included multipole solver [21], AMReX's multigrid solver, or an interface to one of the math libraries. For parabolic equations, one must rely upon library interfaces.

The maintained code components are written in a combination of high-level languages such as Fortran, C, and C++, with an embedded domain-specific configuration language (DSCL) that also supports Flash-X custom macros. The DSCL permits multiple alternative definitions of macros with a built-in arbitration mechanism to select the appropriate definition for an instance of code assembly. The accompanying configuration toolchain can translate and assemble different combinations of the components to configure a diverse set of applications. Flash-X has been designed from the outset to be performant with increasing heterogeneity of both the platforms and the solvers within the code.

The code uses the Message-Passing Interface (MPI) library for communication between nodes, though more than one MPI rank can also be placed on a node. HDF5 is the default mode for IO. Support for OpenMP, both for threading and for offloading to accelerators, is built into several, though not all, of the solvers.

2.1. Software architecture

Flash-X has composable components with accompanying metadata that can express, for example, inter-component dependency and exclusivity, necessary state variables, etc. The metadata is encapsulated within the code components by accompanying *config* files and is parsed and interpreted by the configuration tool, *Setup*. Setup parses config files recursively, aggregates requisite components, and assembles a complete application. It also assembles the compilation/make system and runtime parameters for each component included in the application. The Setup tool also implements code inheritance through a combination of keywords in the config files and the Unix directory structure instead of using programming language supported

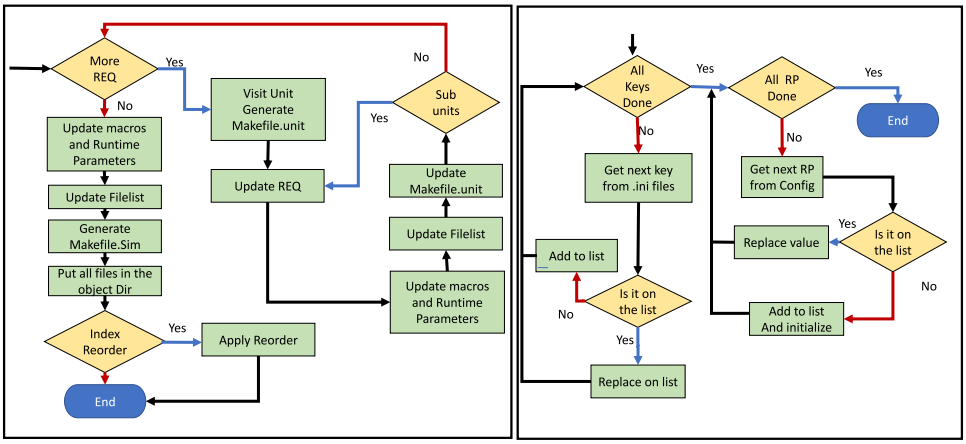

Fig. 1. Schematic for the implementation of inheritance in Flash-X. Handling of inheritance and variants assumes three lists: one for files, one for macros, and one for runtime parameters. The flowchart in the right box gives details of how keys and runtime parameters are updated as the source tree is traversed.

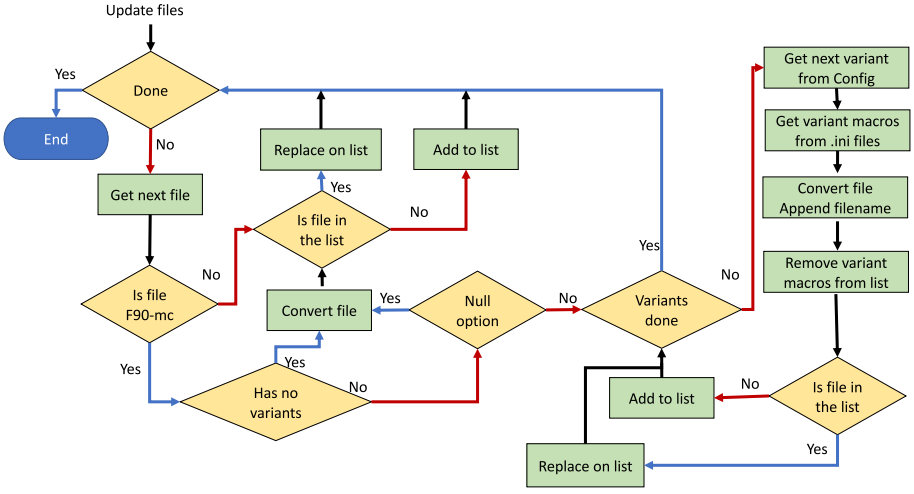

Fig. 2. Schematic for generating variants from single source using inheritance and macros.

inheritance mechanisms. When Setup parses the source tree, it treats each subdirectory as inheriting all of the files in its parent's directory. While source files at a given level of the directory hierarchy override files with the same name at higher levels, config files accumulate all definitions encountered. The schematic for inheritance is shown in Fig. 1.

In Flash-X parlance, the highest-level code component for a specific type of functionality is called a *unit*. Units can have subunits. A unit includes an API accessible to the whole code through which it interacts with other units and the driver. While each subunit can have its own sub-components with no restriction on how fine-grained they can become, the general rule of thumb is to keep them as coarse-grained as feasible for ease of maintenance. A unit can have multiple alternative implementations, one of which is required to be a null implementation. If a unit is not needed in a simulation, the null implementation is included. This feature facilitates maintaining very few implementations of the main driver while permitting many combinations of capabilities to be included in an application. Any code component can have multiple alternative implementations, though unlike the unit-level API, lower-level components do not require null implementations.

A different mechanism is used when a code component needs to become smaller than a function or a subroutine. Here, we rely on macros to implement alternative definitions of an operation, including the null case. The inheritance mechanism shown in Fig. 1 arbitrates on which definition to select. The macros may also have arguments, be inlined, and be recursive. This mechanism serves two purposes. The first is for developer convenience. Certain code patterns repeat often in the code — for example, invocation of iterators, bounds for loop-nests, and bounds for arrays. We have provided macros for such repeated patterns, and developers can use these at their discretion. Macros make the code compact, reduce cut-and-paste errors, and help to clarify the control flow and semantics of the code. The second, more powerful motivation is that with alternative definitions, we can generate many variants of a code component from the same source. This functionality is particularly useful when different control flow is more suitable for different compute devices. We can keep arithmetic expressions invariant while using macros for the control flow, or vice-versa, thus not only eliminating code duplication but also keeping the maintained code more compact. The schematic for generating variants from a single source where specializations are obtained through alternative macro definitions is shown Figure in 2.

2.2. Software functionalities

The Flash-X distribution includes solvers for compressible and incompressible fluids, several methods for handling equations of state (EOS), source terms for nuclear burning, several methods for computing effects of gravity, level-set methods for multiphase

flow, and several others. The primary formulation for PDEs in Flash-X is Eulerian, although a versatile Lagrangian framework is also included that can be configured to do computations such as tracers, particles-in-cell, immersed boundaries, etc. The vast majority of applications using Flash-X include some form of hydrodynamics or magnetohydrodynamics in their configuration. However, it is possible to configure applications that completely bypass those solvers.

Magnetohydrodynamics and Hydrodynamics: a compressible magnetohydrodynamics/hydrodynamics solver with second- or third-order strong stability preserving (SSP) Runge–Kutta (RK) time integration (Spark) [4], another compressible hydrodynamics solver with a predictor–corrector formulation [22,23], and an incompressible hydrodynamics solver with fluid–structure interaction [24] are included in the distribution. All of the solvers can be used in 1-, 2-, or 3- dimensional configurations.

Equations of State: the code supports several EOS versions suitable for a range of regimes in astrophysical flows. The simplest one is a perfect-gas EOS with a multispecies variant. Another implementation with two variants uses a fast Helmholtz free-energy table interpolation to handle degenerate relativistic electrons and positrons and also includes radiation pressure and ions (via the perfect gas approximation) [25].

Nuclear Burning: three nuclear reaction networks of varying numbers of species are included in the distribution. Approx-13 and approx-19 [26] are inherited from FLASH. XNet is a standalone code for evolving astrophysical nuclear burning and is generalizable to arbitrarily large networks as needed for improved physical fidelity of some applications.

Gravity: the gravitational potential can be treated very simply as constant, or through a Poisson solve using a multipole or multigrid method depending upon the symmetry of the density field.

Particles: this component of the code forms the basis for the Lagrangian framework [27]. Particles maintain their own spatial coordinates and are independently integrated in time. They interact with the Eulerian mesh either to obtain physical quantities needed for their advancement or to deposit quantities such as mass, charge, or energy to the mesh, depending on usage.

Incompressible Fluid Dynamics: this component of the code solves incompressible Navier–Stokes equations for single and multiphase flow simulations with options for heat transfer and phase transitions [12]. The Navier–Stokes solver is implemented using a fractional-step temporal integration scheme that uses a Poisson solver for pressure. Multiphase interfaces are tracked with a level-set function and use ghost-fluid methods to account for forces due to surface tension and mass transfer [28]. The effect of solid bodies on the fluid is modeled using an immersed boundary method that uses Lagrangian particles [29].

Importable Modules: Flash-X uses GitHub's submodules to import some capabilities that are independently developed and hosted in their own repositories. These include WeakLib for tabulated, nuclear EOS and neutrino–matter interaction rates, and thornado for spectral neutrino radiation transport.

3. Illustrative examples

We describe two example simulations using Flash-X from two different science communities. The first is a CCSN simulation that uses compressible hydrodynamics, nuclear EOS, neutrino radiation transport, and self-gravity solvers. The second is a subcooled flow boiling simulation that uses multiphase incompressible Navier–Stokes and heat advection diffusion solver.

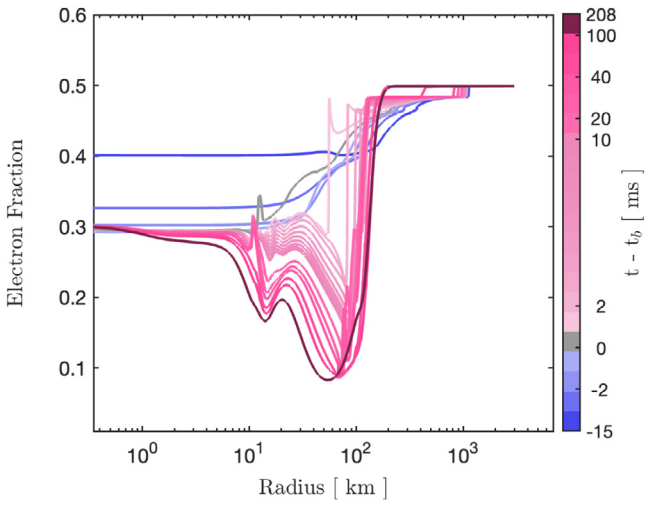

Fig. 3. The electron fraction is plotted versus stellar radius for various times (relative to the bounce-time, t_b) in a CCSN simulation.

We perform a CCSN simulation in spherical symmetry, initiated with a low-mass pre-collapse progenitor star previously modeled throughout all stages of stellar evolution [30]. Electron-type neutrinos and anti-neutrinos are evolved using thornado's two-moment neutrino transport solver and WeakLib's tabulated nuclear EOS [31] and neutrino–matter interaction rates [32]. Compressible hydrodynamics are evolved with Spark, and Newtonian self-gravity is computed using the multipole Poisson solver. For a more detailed description of the physics included, see [33]. Fig. 3 shows the evolution of the ratio of electrons to baryons (electron fraction) versus radius during a critical epoch in the simulation that spans the formation of the primary shock-wave during core "bounce" — the phenomena of infalling matter colliding with, and *bouncing* off of, the newly-formed neutron-star.

Fig. 4 provides details for the subcooled flow-boiling simulation which was designed to replicate experiments performed at different gravity levels by Lebon et al. [34]. These computations used the multiphase incompressible Navier–Stokes solver along with the phase transition capability, and were preformed at a resolution almost twice the previous state-of-the-art [28,35]. Liquid coolant flows over a heater surface with a mean velocity U_0, leading to phase-change and formation of vapor bubbles. These vapor bubbles grow, merge, and finally depart the heater surface due to buoyancy which introduces turbulence in the domain. The heat transfer associated with this turbulence is an important parameter in designing cooling systems for automotive and industrial components, but is difficult to quantify through experimental observations/measurements. With Flash-X we are able to address this challenge through targeted high-fidelity simulations to quantify the contribution of turbulent heat flux.

4. Impact

FLASH has been an influential code for computing astrophysical flows almost since its inception. FLASH's scientific impact is clearly demonstrated by the citation history of the original paper describing the code, as shown in Fig. 5. Analysis in [36,37] further quantifies the scientific significance and impact of the code on science. FLASH has not only been used extensively for science, it has also been among the pioneers in giving due importance to software quality and adopting rigorous auditing and productivity practices [38].

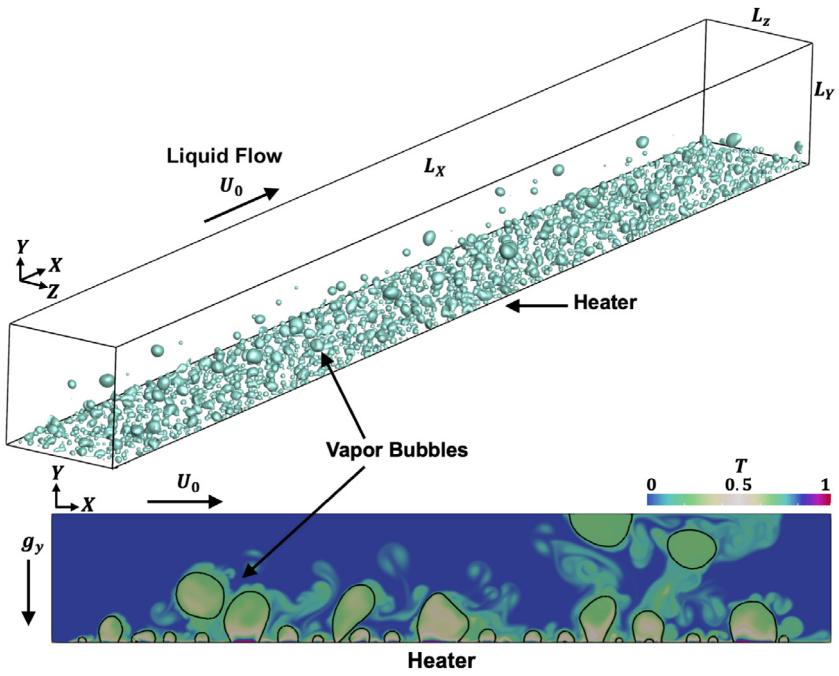

Fig. 4. Example of a flow boiling simulation with Flash-X. Liquid coolant flows over a heater surface with a mean velocity U_0, leading to phase-change and formation of vapor bubbles. The bubble dynamics introduce turbulence which enhances heat transfer between the heater surface and coolant, shown by temperature (T) contours.

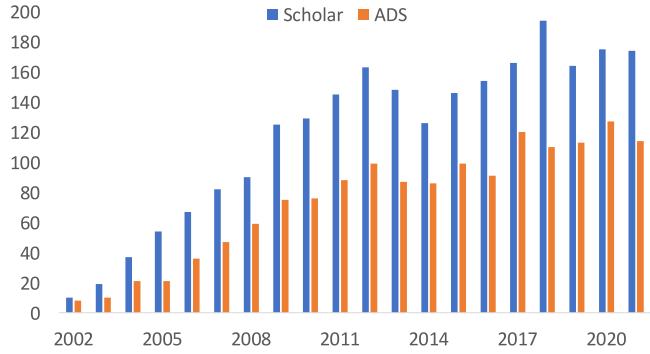

Fig. 5. Citation history of the original FLASH paper from Google Scholar and ADS.

In recent years, FLASH's use has been diminishing in several communities because of its inability to use accelerators effectively. Flash-X is designed to fill this gap and become a reliable multiphysics simulation code for the communities that earlier relied on FLASH. At least two major communities of FLASH users, stellar astrophysics [4,33] and fluid–structure interactions [12], are already transitioning to Flash-X, with some users now exclusively using Flash-X. These use cases have also reported on performance gains with the use of GPUs. Not all of FLASH's physics capabilities are available in Flash-X yet. However, since Flash-X is open source, it is expected that interested users will assist in transitioning their capabilities of interest to the Flash-X architecture and help grow the Flash-X community. Additionally, the new tool-chain for orchestration of data and work movement is still in the early stages of being exercised. Preliminary performance studies of the runtime tool have been very encouraging [39]. It is expected that full performance gains will have been realized by the next major release

5. Conclusions

Sustained funding under the ECP has permitted modernization of a highly capable community code for current and future platforms. With Flash-X, the FLASH science communities can embrace heterogeneity and use available hardware effectively. With the move to an open, community-based development model, users are assured of continuity and support for the code without depending on a single funding source. FLASH has had a long history of scientific discovery, and Flash-X aims to follow in that tradition. With more modern solvers and flexible architecture, Flash-X can continue to be a very useful resource for science domains that rely on modeling of partial differential equations.

CRediT authorship contribution statement

Anshu Dubey: Conceptualization, Methodology, Software, Validation, Writing – original draft, Supervision, Project administration, Funding acquisition. **Klaus Weide:** Conceptualization, Methodology, Software, Validation, Writing – review & editing. **Jared O'Neal:** Conceptualization, Methodology, Software, Validation, Writing – review & editing. **Akash Dhruv:** Conceptualization, Methodology, Software, Validation, Writing – original draft. **Sean Couch:** Methodology, Software, Validation, Writing – review & editing. **J. Austin Harris:** Methodology, Software, Validation, Writing – review & editing. **Tom Klosterman:** Conceptualization, Methodology, Software, Validation. **Rajeev Jain:** Conceptualization, Methodology, Software, Validation. **Johann Rudi:** Conceptualization, Writing – review & editing. **Bronson Messer:**

Methodology, Writing – review & editing, Supervision, Funding acquisition. **Michael Pajkos:** Software, Validation. **Jared Carlson:** Software, Validation. **Ran Chu:** Software, Validation, Writing – original draft. **Mohamed Wahib:** Conceptualization, Writing – review & editing. **Saurabh Chawdhary:** Software, Validation. **Paul M. Ricker:** Conceptualization, Methodology, Software, Validation, Writing – review & editing. **Dongwook Lee:** Conceptualization, Methodology, Software, Validation. **Katie Antypas:** Conceptualization, Methodology, Software, Validation, Writing – review & editing. **Katherine M. Riley:** Conceptualization, Methodology, Software, Validation, Writing – review & editing. **Christopher Daley:** Methodology, Software, Validation. **Murali Ganapathy:** Conceptualization, Software, Validation. **Francis X. Timmes:** Conceptualization, Methodology, Software, Validation, Writing – review & editing. **Dean M. Townsley:** Methodology, Software, Validation, Writing – review & editing. **Marcos Vanella:** Methodology, Software, Validation. **John Bachan:** Methodology, Software, Validation. **Paul M. Rich:** Software, Validation. **Shravan Kumar:** Methodology, Software, Validation. **Eirik Endeve:** Methodology, Software, Writing – review & editing, Supervision. **W. Raphael Hix:** Methodology, Software, Validation, Writing – review & editing. **Anthony Mezzacappa:** Writing – review & editing, Supervision. **Thomas Papatheodore:** Software, Validation.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Anshu Dubey reports financial support was provided by US Department of Energy.

Data availability

No data was used for the research described in the article.

Acknowledgments

The authors acknowledge all contributors to the Flash-X code, including contributors to the FLASH code from whose work Flash-X has inherited.

This work was supported by the U.S. Department of Energy Office of Science Office of Advanced Scientific Computing Research under contract number DE-AC02-06CH1137.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) that are responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.

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

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. 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.

References

[1] Flash-X. URL https://github.com/Flash-X/Flash-X.
[2] Dubey A, Antypas K, Ganapathy MK, Reid LB, Riley K, Sheeler D, et al. Extensible component-based architecture for FLASH, a massively parallel, multiphysics simulation code. Parallel Comput 2009;35(10–11):512–22. http://dx.doi.org/10.1016/j.parco.2009.08.001.
[3] Fryxell B, Olson K, Ricker P, Timmes F, Zingale M, Lamb D, et al. FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. Astrophys J Suppl Ser 2000;131:273. http://dx.doi.org/10.1086/317361.
[4] Couch SM, Carlson J, Pajkos M, O'Shea BW, Dubey A, Klosterman T. Towards performance portability in the spark astrophysical magnetohydrodynamics solver in the flash-x simulation framework. Parallel Comput 2021;108:102830.
[5] XNet, https://github.com/starkiller-astro/xnet (Apr. 2022).
[6] Hix WR, Thielemann FK. Computational methods for nucleosynthesis and nuclear energy generation. J Comput Appl Math 1999;109:321–51.
[7] Chu R, Endeve E, Hauck CD, Mezzacappa A. Realizability-preserving DG-IMEX method for the two-moment model of fermion transport. J Comput Phys 2019;389:62–93. http://dx.doi.org/10.1016/j.jcp.2019.03.037.
[8] Laiu MP, Endeve E, Chu R, Harris JA, Messer OEB. A DG-IMEX method for two-moment neutrino transport: Nonlinear solvers for neutrino-matter coupling. Astrophys J Suppl Ser 2021;253(2):52. http://dx.doi.org/10.3847/1538-4365/abe2a8.
[9] WeakLib, https://github.com/starkiller-astro/weaklib (Apr. 2022).
[10] Pochik D, Barker BL, Endeve E, Buffaloe J, Dunham SJ, Roberts N, et al. Thornado-hydro: A discontinuous galerkin method for supernova hydrodynamics with nuclear equations of state. Astrophys J Suppl Ser 2021;253(1):21. http://dx.doi.org/10.3847/1538-4365/abd700.
[11] Landfield RE. Sensitivity of neutrino-driven core-collapse supernova models to the microphysical equation of state [Ph.D. thesis], University of Tennessee; 2018, https://trace.tennessee.edu/utk_graddiss/5294.
[12] Dhruv A, Balaras E, Riaz A, Kim J. An investigation of the gravity effects on pool boiling heat transfer via high-fidelity simulations. Int J Heat Mass Transfer 2021;180:121826. http://dx.doi.org/10.1016/j.ijheatmasstransfer.2021.121826.
[13] Exastar, multi-physics stellar astrophysics at exascale. URL https://sites.google.com/lbl.gov/exastar.
[14] The exascale computing project. 2020, URL https://www.exascaleproject.org/.
[15] Alexander F, Almgren A, Bell J, Bhattacharjee A, Chen J, Colella P, et al. Exascale applications: skin in the game. Phil Trans R Soc A 2020;378(2166):20190056. http://dx.doi.org/10.1098/rsta.2019.0056.
[16] MacNeice P, Olson K, Mobarry C, de Fainchtein R, Packer C. PARAMESH: A parallel adaptive mesh refinement community toolkit. Comput Phys Comm 2000;126(3):330–54.
[17] Amrex. 2020, https://amrex-codes.github.io/.
[18] Zhang W, Almgren A, Beckner V, Bell J, Blaschke J, Chan C, et al. AMReX: A framework for block-structured adaptive mesh refinement. JOSS 2019;4(37):1370.
[19] Falgout R, Yang U. Hypre: A library of high performance preconditioners. Comput Sci-ICCS 2002;2002:632–41.
[20] Balay S, et al. Petsc web page. 2001.
[21] Couch SM, Graziani C, Flocke N. An improved multipole approximation for self-gravity and its importance for core-collapse supernova simulations. Astrophys J 2013;778(2):181.
[22] Lee D, Deane AE. An unsplit staggered mesh scheme for multidimensional magnetohydrodynamics. J Comput Phys 2009;228(4):952–75.
[23] Lee D. A solution accurate, efficient and stable unsplit staggered mesh scheme for three dimensional magnetohydrodynamics. J Comput Phys 2013;243:269–92.
[24] Vanella M, Rabenold P, Balaras E. A direct-forcing embedded-boundary method with adaptive mesh refinement for fluid–structure interaction problems. J Comput Phys 2010;229(18):6427–49.
[25] Timmes FX, Swesty FD. The accuracy, consistency, and speed of an electron-positron equation of state based on table interpolation of the Helmholtz free energy. Astrophys J Suppl Ser 2000;126(2):501–16. http://dx.doi.org/10.1086/313304.
[26] Timmes F. Integration of nuclear reaction networks. Astrophys J Suppl Ser 1999;124:241–63.
[27] Dubey A, Daley C, ZuHone J, Ricker PM, Weide K, Graziani C. Imposing a Lagrangian particle framework on an Eulerian hydrodynamics infrastructure in FLASH. Astrophys J Suppl Ser 2012;201:27. http://dx.doi.org/10.1088/0067-0049/201/2/27.
[28] Dhruv A, Balaras E, Riaz A, Kim J. A formulation for high-fidelity simulations of pool boiling in low gravity. Int J Multiph Flow 2019;120:103099. http://dx.doi.org/10.1016/j.ijmultiphaseflow.2019.103099.
[29] Vanella M, Balaras E. Short note: A moving-least-squares reconstruction for embedded-boundary formulations. J Comput Phys 2009;228(18):6617–28. http://dx.doi.org/10.1016/j.jcp.2009.06.003.

[30] Sukhbold T, Ertl T, Woosley S, Brown JM, Janka H-T. Core-collapse supernovae from 9 to 120 solar masses based on neutrino-powered explosions. Astrophys J 2016;821(1):38.
[31] Steiner AW, Lattimer JM, Brown EF. The equation of state from observed masses and radii of neutron stars. Astrophys J 2010;722(1):33.
[32] Bruenn SW. Stellar core collapse - numerical model and infall epoch. Astrophys J Suppl Ser 1985;58:771–841. http://dx.doi.org/10.1086/191056.
[33] Harris JA, Chu R, Couch SM, Dubey A, Endeve E, Georgiadou A, et al. Exascale models of stellar explosions: Quintessential multi-physics simulation. Int J High Perform Comput Appl 2021;10943420211027937.
[34] Lebon MT, Hammer CF, Kim J. Gravity effects on subcooled flow boiling heat transfer. Int J Heat Mass Transfer 2019;128:700–14. http://dx.doi.org/10.1016/j.ijheatmasstransfer.2018.09.011.
[35] Sato Y, Niceno B. Pool boiling simulation using an interface tracking method: From nucleate boiling to film boiling regime through critical heat flux. Int J Heat Mass Transfer 2018;125:876–90. http://dx.doi.org/10.1016/j.ijheatmasstransfer.2018.04.131.
[36] Dubey A, Tzeferacos P, Lamb DQ. The dividends of investing in computational software design: A case study. Int J High Perform Comput Appl 2019;33(2):322–31.
[37] Grannan A, Sood K, Norris B, Dubey A. Understanding the landscape of scientific software used on high-performance computing platforms. Int J High Perform Comput Appl 2020;34(4):465–77.
[38] Dubey A, Antypas K, Calder AC, Daley C, Fryxell B, Gallagher JB, et al. Evolution of FLASH, a multi-physics scientific simulation code for high-performance computing. Int J High Perform Comput Appl 2014;28(2):225–37.
[39] O'Neal J, Wahib M, Dubey A, Weide K, Klosterman T, Rudi J. Domain-specific runtime to orchestrate computation on heterogeneous platforms. In: Chaves R, Heras DB, Ilic A, Unat D, Badia RM, Bracciali A, Diehl P, Dubey A, Sangyoon O, Scott SL, Ricci L, editors. Euro-Par 2021: parallel processing workshops. Cham: Springer International Publishing; 2022, p. 154–65.