11th World Congress on Computational Mechanics (WCCM XI)
5th European Conference on Computational Mechanics (ECCM V)
6th European Conference on Computational Fluid Dynamics (ECFD VI)
E. Oñate, J. Oliver and A. Huerta (Eds)

THE IMPACT OF COMMUNITY SOFTWARE IN ASTROPHYSICS

A. Dubey[1], M.J. Turk[2] and B.W. O'Shea[3]

[1] Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, USA, adubey@lbl.gov and http://crd.lbl.gov/groups-depts/ANAG/about/staff/anshu-dubey/
[2] Astronomy and Astrophysics, Columbia University, Mail Code 5246, 550 W 120th St, New York, NY 10027, matthewturk@gmail.com
[3] Department of Physics and Astronomy, Lyman Briggs College, and the Institute for Cyber-Enabled Research, Michigan State University, East Lansing, MI 48824, USA, oshea@msu.edu

Key words: Community Astrophysics Codes, Impact on Communities

Abstract. Advances in mathematical models and numerical algorithms combined with increasing reliance on simulations for understanding multi-physics and multi-scale phenomena has made the task of software development for simulations a large and complex enterprise. Development and adoption of community codes is one way to address this challenge. The astrophysics community has been ahead of many other science communities in making research codes publicly available, and therefore in the development and adoption of community codes. ZEUS-2D was one of the earliest codes to become public, and it has been followed by several others. In this paper, we highlight the impact of three of these packages – FLASH, Enzo and yt – on the astrophysics community. These, and similar projects in other fields, show that if reliable and robust codes exist, and efforts are made to promote their use, they facilitate higher overall scientific output. They also help foster open science in their corresponding communities, which has wide-ranging benefits.

1 Introduction

With modeling and simulation gaining acceptance as necessary research tools in more research fields, there are rapid advancements in mathematical models and numerical algorithms. These advances have brought more complex non-linear problems within reach, but have correspondingly increased the challenge of developing scientific codes for simulation and analysis. The model of small groups developing their own software still applies to very specific limited phenomenon research, but increasingly, the research problems being pursued through simulations are multi-physics and multi-scale in nature. Such problems typically require several independent or interdependent solvers to properly model all phenomena of interest, and they need high performance computing (HPC) resources.

The solvers often have divergent requirements of data management, and place different demands on the computing platforms. However, for an integrated simulation, all the included solvers need to be able to interoperate with one another. When we take into account the diversity of computing platforms and their typical shelf life into account, and compare that with the number of person years it takes to build a reliable and efficient multi-physics software for one platform, it becomes obvious that the task of building highly capable scientific research software has gone beyond the resources of individual, or even small groups of researchers. Development and adoption of community codes is, therefore, of paramount importance in today's research.

The astrophysics community has been ahead of many other science communities in making research codes publicly available, and therefore in the development and adoption of community codes. ZEUS-2D [8] was one of the earliest codes to become public, and it has been followed by several others such as FLASH [3], Gadget [6], Enzo [9] and Athena [7]. This list is not exhaustive, but it provides a glimpse of the open software development that prevails in the astrophysics community. This culture of openness and sharing of code is further reflected in the development and rapid adoption of `yt` [10], an analysis and visualization package, which is proving to be an immensely useful resource to the community. In this paper, we highlight the impact of community codes in astrophysics research with focus on three of these packages: FLASH, Enzo and `yt`.

The remainder of the paper is organized as follows: section 2 gives a brief history of community codes in astrophysics, and development histories of the three codes that are the focus of this paper. Section 3 provides a survey of computing capabilities covered by the three codes, their user communities and their impact are described in section 4. Finally, in section 5 we give our conclusions.

2 History

2.1 FLASH

The Flash project started as part of the Department of Energy University Alliance program, under its Advanced Scientific Computing Initiative (ASCI, now ASC) program. One of the deliverables of the project was public release of the scientific software used for simulations in the project. This is how the FLASH code came into existence in 1999 and first became public in 2000. The first version of the code was an amalgamation of two already existing software packages, PARAMESH [5] an adaptive mesh refinement (AMR) library, and PROMETHEUS [4] for compressible hydrodynamics, and sections of several other codes that had been developed individually by various members of the team in their earlier research.

FLASH has had four major versions releases, where first release was provided within two years of the start of the Center. This version also laid the foundation of the code architecture with modularity and extensibility in mind. Version 1 had only one release, 1.6, before transitioning to Version 2 in 2003. The final release of version 2 was 2.5,

occurring in 2006. Version 3 was released in 2008, with the alpha and beta versions appearing in 2006 and 2007 respectively, and went up to 3.3 release. FLASH4.0 was released in 2012, without significant architectural or infrastructure change, instead with capabilities added that targeted a new research community (high energy density physics, or HEDP). FLASH has had over 50 significant developers during its lifetime, and for most of its history the development has been centralized at the University of Chicago's Flash Center. Though in the more recent years there has been a rapid increase in external contributions. FLASH development has been predominantly supported by DOE-NNSA, with some funding from NSF. The Flash Center at the University of Chicago distributes and supports the code with ongoing funding from DOE-NNSA. For more details on the history and evolution of the code see [2, 1].

2.2 Enzo

The Enzo code (`http://enzo-project.org/`) started as a graduate project of Greg Bryan in the mid-1990s at the University of Illinois, and was subsequently stewarded by the Laboratory for Computational Astrophysics (LCA) at the University of California at San Diego. Initially, Enzo was released to "friendly users," and then distributed via tar file as a public open-source code release in 2004 (as Enzo Version 1.0). Following the first public release, Enzo 1.5 was distributed in 2007 using the Subversion version control system. In 2009, many versions of Enzo (being maintained and modified by research groups at different universities) were merged together and were distributed as Enzo 2.0, which included substantially improved physics packages and infrastructure beyond the previous public release. Perhaps more importantly, Enzo 2.0 and following are distributed using the Mercurial distributed version control system (`http://mercurial.selenic.com/`), allowing users to easily use version control on their own version of the code, to share code modifications with each other, and to contribute these changes upstream to the Enzo community codebase (with quality control being maintained by rigorous peer review and an automated testing framework). Currently, Enzo is developed using the hosted source control platform BitBucket at http://bitbucket.org/enzo, and the "stable" version of Enzo is updated roughly annually as point releases (with Enzo 2.3 appearing in 2013). Version 3.0 of Enzo is currently under development, and will include significant infrastructure improvements. Enzo has had approximately 25 developers during its lifetime, spread across roughly a dozen institutions worldwide, and a large and enthusiastic user community. Funding for Enzo development has come from a variety of sources, including NSF, DOE, NASA, direct funding by universities, and foreign funding agencies in Canada, Japan, and the European Union.

2.3 yt

`yt` (`http://yt-project.org/`) is an analysis and visualization package for volumetric data. Originally created in 2006 by a single author, it was designed to conduct rela-

tively simple visualization of Enzo data. At this time, the development was funded as a direct side effect of the scientific research being conducted by the author of the code. As additional functionality was added to `yt`, the code base became more complex and was extended to support data formats from several different simulation codes including the Orion (boxlib) code and the FLASH code. Over time additional individuals, first from the Enzo community and then from additional communities, began to develop and contribute new functionality. As the developer community grew, the focus shifted from simple visualization to a more generic analysis framework, where visualization is regarded as a consequence of the data ingestion and management. This enabled deeper and more complex data representations, including volume rendering, which served to expand the user and developer community. Starting in 2011, the lead author of the code (co-author Turk) was the recipient of an NSF Fellowship in CyberInfrastructure for Transformative Computational Science (CI TraCS), which funded the development of additional functionality and an expansion of the indexing and data ingestion facilities to additional simulation platforms and methodologies. In late 2013, the project was the subject of a successful NSF SI2-SSE grant, designed to further develop its infrastructure, interface and instrumentation capabilities; this will lead to an expansion of the served communities, an investment in growth of functionality, and deeper ties to advanced computing resources. It is developed completely in the open on the hosted source control platform BitBucket (`http://bitbucket.org/yt_analysis/`), with continuous integration for automated testing of contributions, and has received contributions of all sizes from over 50 individuals over its lifetime.

3 Capabilities

3.1 FLASH

FLASH can solve a large class of problems in reactive compressible flows. It can model pure hydrodynamics as well as magnetohydrodynamics, and many flavors of gravity, including self-gravity. Support exists for various equations of states in fields such as supernovae and laser experiments, as well as the simplest ideal gas gamma law. Source terms include heating, cooling, laser drive, nuclear burning, conduction and diffusion. The code can model radiation with flux-limited-diffusion. The infrastructural capabilities include uniform mesh and AMR with parallel IO for both. A Lagrangian framework also exists in the code which support particles in passive and active modes. The active particles can be treated as massive or charged. Certain capabilities are not in public release at the time of this writing. They include an incompressible Navier-Stokes solver and fluid-structure interactions using immersed boundary methods. Significant external contributions include multigrid and tree solvers for self gravity, sink particles, primordial chemistry, and hybrid-PIC.

3.2 Enzo

Enzo was originally designed to study cosmological structure formation, but is generally useful for the study of self-gravitating compressible fluid dynamics. It supports N-body particle dynamics (typically, but not exclusively, used to simulate dark matter), a variety of solvers for the equations of hydrodynamics and magnetohydrodynamics (both ideal and non-ideal), radiation transport using either a ray-casting method or implicit flux-limited diffusion, and a variety of subgrid models for radiative cooling, gas chemistry, and the formation and feedback of stars and black holes. Enzo is a Cartesian grid code, and can be run in 1, 2 or 3 dimensions using either a grid of constant cell size or patch-based adaptive mesh refinement with grid patches of arbitrary size and aspect ratio. Adaptive time-stepping is used throughout the code, with each level of the grid hierarchy taking its own timestep – this is critical for the study of gravitationally-driven astrophysical phenomena, as the physical timescales of interest vary tremendously. The code is highly parallel (scaling well to $\simeq 10^5$ MPI tasks for non-AMR runs and $\simeq 10^4$ MPI tasks for AMR runs), and portions of it have been modified to run on graphics processing units.

3.3 yt

The analysis framework yt was instrumented in a similar fashion to FLASH; while initially designed to support only the Enzo simulation code, it has been designed to be modular in its intake of data while retaining a consistent public data access API and internal data representation. This enables identical analysis scripts to be applied to data produced by simulation codes such as FLASH, Enzo, and Athena. Recently added support for octree and particle-based codes such as RAMSES and Gadget has led to significant increase in the size of the community.

4 Impact

FLASH and Enzo together, annually account for the usage of several hundred million cpu hours on supercomputers run by DOE, NSF, NASA and other international agencies. FLASH has been downloaded more than $3,000$ times with several of the downloads by the same person or group, indicating continued use of the code by those users. Over 850 papers have been published that used FLASH to obtain all or part of their results. Enzo is somewhat less widely used, but has been used for substantially more than 100 peer-reviewed papers. Although almost all users need something different from, or over and above, the core capabilities provided by the public FLASH or Enzo distribution, they still benefit tremendously from having many of their needed capabilities available in an extensible and flexible framework into which they can plug in their additions. They have the added advantage of almost never having to concern themselves with the basic infrastructure services such as mesh management and IO. Many of these additions make their way into the public code base because the contributors find it rewarding to have a permanent place for the results of their development efforts, and they also become better

known in the community through their contributions.

Over the three years since the yt method paper [10] was published, it has been cited by approximately 105 different papers. During its lifetime, code contributions from nearly 50 different authors (ranging from very small to very large) have been accepted to the main code base and the user and developer mailing list have 220 and 70 members, respectively. On these mailing lists not only are questions related to the code discussed, but a number of related topics such as mechanisms for analysis, extracting meaningful quantities from data, and issues with software installation and management.

Community growth has been important to all three of these projects. An active user and developer community serves the extremely important function of reducing the barrier to entry for new researchers using HPC resources for scientific simulations. This has been a particularly high priority for yt, as it is segmented into both fundamental infrastructure (typically contributed by a core of individuals) and analysis modules and/or code front-ends, which are often contributed by individuals that are newer to the project or who have developed technology for their own use. The yt project itself is designed to remove barriers to direct technology sharing between researchers by providing a common analysis platform.

The impact has been most marked on the younger members of the community. Many graduate students and post-doctoral researchers in computational astrophysics do not start by writing their own code from scratch – they look through the available packages, select the one closest in capabilities to their needs and enhance and/or customize it for their own purposes. In the process they typically save tremendous amount of effort that would have gone toward replication of someone else's effort, and are instead able to focus on advancing their research. They have the added advantage of using tools that have been vetted by multiple researchers and are therefore less error-prone than their own necessarily less-tested code would have been. As a consequence, their research goes farther faster.

More generally, the widespread use of the open-source tools described in this paper has helped to hasten a change in attitude in the theoretical astrophysics community. In particular, over the past decade the concept of a "community code" has become gradually more accepted, and young scientists have seen how contribution to these open-source projects can have a positive impact on their scientific reputation and career prospects. Having completely open tool chains (e.g., Enzo or FLASH used for simulations, and yt for data analysis and figure production) also helps with the promotion of open, reproducible research, with some authors releasing simulation parameter files and yt analysis scripts along with their papers (with the Enzo method paper being a prime example of this – see `https://bitbucket.org/enzo/enzo-method-paper`). This makes it relatively straightforward to reproduce scientific results, supports federally-mandated goals regarding the sharing of scientific data, and assists with making "apples to apples" comparisons between different simulation tools.

Additionally, investment in a well-architected community code framework can sometimes have unexpected beneficial side effects. For example, because many needed capabil-

ities overlap between astrophysics and HEDP, FLASH has recently been adopted as the primary open code for the academic HEDP community. The impact was huge because the academic HEDP community has historically had very little access to codes capable of simulating experiments for design and analysis. The presence of an AMR framework with built-in IO, runtime management, many needed solvers and plugin-capable architecture has enabled a community code for an entirely new community in a short time (less than two years) with very modest investment.

5 Conclusions

These, and similar projects in other fields, show that if reliable and robust codes exist, and efforts are made to promote their use, they facilitate higher overall scientific output. This is because the community codes with their robust infrastructure and extensibility lead to better utilization of the most scarce resource in any research enterprise – human time. Researchers don't waste their precious time in developing infrastructural support and re-implementing well known solvers for their research codes if they can leverage needed capabilities already provided by one of the community codes. They can instead focus on either developing more innovative methods or using the code to obtain results. Open community codes also help foster open science in their corresponding communities, which has wide-ranging benefits mentioned above. Furthermore, involvement in these communities has had strong positive impact on the career trajectories of young scientists, and has contributed significantly to the cause of transparent and open science.

REFERENCES

[1] A. Dubey, K. Antypas, A. Calder, B. Fryxell, D.Q. Lamb, P. Ricker, L. Reid, K. Riley, R. Rosner, A. Siegel, F. Timmes, N. Vladimirova, and Klaus Weide. The software development process of flash, a multiphysics simulation code. In *SE-CSE San Francisco, USA*, 2013.

[2] A. Dubey, K. Antypas, A.C. Calder, C. Daley, B. Fryxell, J.B. Gallagher, D.Q. Lamb, D. Lee, K. Olson, L.B. Reid, P. Rich, P.M. Ricker, K.M. Riley, R. Rosner, A. Siegel, N.T. Taylor, F.X. Timmes, N. Vladimirova, K. Weide, and J. ZuHone. Evolution of FLASH, a multiphysics scientific simulation code for high performance computing. *International Journal of High Performance Computing Applications*, accepted, 2013.

[3] A. Dubey, K. Antypas, M.K. Ganapathy, L.B. Reid, K. Riley, D. Sheeler, A. Siegel, and K. Weide. Extensible component-based architecture for FLASH, a massively parallel, multiphysics simulation code. *Parallel Computing*, 35(10-11):512–522, 2009.

[4] B.A. Fryxell, E. Müller, and D. Arnett. *Numerical Methods in Astrophysics*, page 100. New York: Academic, 1989.

[5] P. MacNeice, K.M. Olson, C. Mobarry, R. de Fainchtein, and C. Packer. PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, 126(3):330–354, 2000.

[6] V. Springel. The cosmological simulation code GADGET-2. *MNRAS*, 364:1105–1134, December 2005.

[7] J. M. Stone, T. A. Gardiner, P. Teuben, J. F. Hawley, and J. B. Simon. Athena: A new code for astrophysical MHD. *APJS*, 178:137–177, September 2008.

[8] J. M. Stone and M. L. Norman. ZEUS-2D: A radiation magnetohydrodynamics code for astrophysical flows in two space dimensions. I - The hydrodynamic algorithms and tests. *APJS*, 80:753–790, June 1992.

[9] The Enzo Collaboration, G. L. Bryan, M. L. Norman, B. W. O'Shea, T. Abel, J. H. Wise, M. J. Turk, D. R. Reynolds, D. C. Collins, P. Wang, S. W. Skillman, B. Smith, R. P. Harkness, J. Bordner, J.-h. Kim, M. Kuhlen, H. Xu, N. Goldbaum, C. Hummels, A. G. Kritsuk, E. Tasker, S. Skory, C. M. Simpson, O. Hahn, J. S. Oishi, G. C So, F. Zhao, R. Cen, and Y. Li. Enzo: An Adaptive Mesh Refinement Code for Astrophysics. *ArXiv e-prints*, July 2013.

[10] Matthew J. Turk, Britton D. Smith, Jeffrey S. Oishi, Stephen Skory, Samuel W. Skillman, Tom Abel, and Michael L. Norman. yt: A multi-code analysis toolkit for astrophysical simulation data. *The Astrophysical Journal Supplement Series*, 192(1):9, 2011.