Journal of
open research software

Löffler, F et al 2014 Cactus: Issues for Sustainable Simulation Software. *Journal of Open Research Software,* 2(1): e12, pp. 1-6, DOI: http://dx.doi.org/10.5334/jors.au

ISSUES IN RESEARCH SOFTWARE

Cactus: Issues for Sustainable Simulation Software

Frank Löffler*, Steven R. Brandt†, Gabrielle Allen‡ and Erik Schnetter§

The Cactus Framework is an open-source, modular, portable programming environment for the collaborative development and deployment of scientific applications using high-performance computing. Its roots reach back to 1996 at the National Center for Supercomputer Applications and the Albert Einstein Institute in Germany, where its development started. Since then, the Cactus framework has witnessed major changes in hardware infrastructure as well to its community. This paper describes its growth through these changes and, drawing upon lessons from its past, also discusses future challenges for Cactus.

Keywords: frameworks; scientific computing; software sustainability

1. Introduction

Motivated by the needs of the numerical relativity research community and stemming from earlier efforts at the National Center for Supercomputer Applications in the U.S., the design and development of the Cactus framework [1, 2] began at the Albert Einstein Institute, a Max Planck Institute for Gravitational Physics in 1996. The component-based architecture of Cactus was inspired by experiences of physicists and computer scientists who had previously worked together in the USA Binary Black Hole Alliance Grand Challenge. This NSF-funded collaboration (1993–1999) involved over eleven groups, working with a variety of independent code bases on a set of different projects with the aim of modeling the inspiral collision of two black holes using then-state-of-the-art supercomputers[1]. Even within a single group, multiple codes were used, often with multiple versions of each code. Contrary to the spirit of collaboration, advances in research methods or computing technologies were re-implemented, debugged, and verified in each code, thus duplicating effort, hampering communication, and slowing scientific progress.

The vision of the Cactus team was to provide an organic, community-oriented framework that would allow researchers to easily work together with reusable and extensible software elements.

From the beginning, the Cactus framework followed a modular design. It features a comparably small core (named the "flesh") which provides the compile-time and run-time interfaces between modules. The Cactus modules (called "thorns") use these APIs to specify inter-module dependencies and connections, e.g. whether to share or extend configuration information, to use common variables or run-time parameters. Modules compiled into an executable can remain dormant at run-time. Based on user-specified parameters and simulation data itself, the flesh decides when and in which order to call functions in different modules, assembling them into a coherent simulation.

This usage of modules and a common interface between them enables researchers to 1) easily use modules written by others without the need to understand all details of their implementation; 2) write their own modules without the need to change the source code of other parts of a simulation in the (supported) programming language of their choice; and 3) easily communicate the scientifically relevant ideas behind the module without involving the infrastructural details. The number of active modules within a typical Cactus simulation ranges from tens to hundreds and often has an extensive set of inter-module dependencies.

2. An evolving framework community

The accelerating growth and diversity of the Cactus community reinforced the modular development of both the physics-based and computational infrastructure. Examples of modules include the evolution equations for General Relativity, radiation or reflective boundary conditions, MPI parallelization, parameter parsing, and output routines. The value of this modularization is hard to overstate.

This design was of tremendous help to the motivating science problem, namely numerical relativity. It also

* Center for Computation and Technology, Louisiana State University, Baton Rouge, Louisiana, USA
knarf@cct.lsu.edu

† Center for Computation and Technology and Department of Computer Science, Louisiana State University, Baton Rouge, Louisiana, USA

‡ University of Illinois, Urbana-Champaign, Illinois, USA

§ Perimeter Institute for Theoretical Physics, Waterloo, Canada, Department of Physics, University of Guelph, Guelph, Canada, and Center for Computation and Technology, Louisiana State University, Baton Rouge, Louisiana, USA

Corresponding author: Frank Löffler

became clear that with such a design, the modules could easily be purposed for other science problems. PDE problems are especially well-suited for use with the Cactus framework, however, since Cactus was initially written for numerical relativity, which solves a set of complex, partial differential equations (PDEs).

The straightforward reusability of existing components spurred development in other areas of science-usually numerically similar-in particular solving sets of PDEs, e.g., coastal simulations of storm surges. However, most users of the Cactus framework are interested in numerical relativity, or more generally, relativistic astrophysics. Cactus development within this field is coordinated within the Einstein Toolkit [3, 4]. This focus stems, at in least part, from the fact that during most of Cactus development, interaction between developers and users was tight. In fact, most users became developers to some degree relatively quickly, so that users and developers were never truly distinct categories. One of the main reasons for this is the modular plug-in nature of Cactus, allowing end users to add new thorns to their application.

The majority of developers are and have been motivated by the study of numerical relativity rather than the broadening of the user base of the framework itself to other areas of science. However, owing to the modular nature of the framework, such a broadening was brought about as a result.

Today the Cactus and the Einstein Toolkit communities are still strongly interlinked through individuals who are active in both communities; but as different subdomains expand their usage of Cactus, these two groups have polarized into different roles. Currently, most users of the Einstein Toolkit are not active developers of the underlying framework itself, but rather merely use it to create and extend modules within the Einstein Toolkit. While there are exceptions, most new infrastructure is carried out by a few core developers and is based on feedback from the user community.

2.1. Related Work

A multitude of simulation packages exist that offer similar capabilities. Even limiting this to parallel AMR codes produces a list too long to discuss here. Therefore, we select a couple of cases which we believe to be most relevant.

Paramesh [5, 6] is a package of Fortran 90 subroutines designed to provide an application developer with an easy way to extend an existing serial code. Other interesting examples are Enzo [7, 8], BoxLib [9], Chombo [10], Uintah [11, 12], Jasmine [13] and SAMRAI [14, 15], but many more exist.

Likewise, DSL's are not unusual here. For example, Flash uses a DSL for configuration, and Chombo has ChomboFortran, a special macro language extension to ease portability, maintainability, dimension independence, and parallelism [16].

Many of these tools employ some form of modularization, but we believe that the way this is done in Cactus is unique, especially the use of self-describing modules.

3. Software Sustainability Issues

There are many issues connected to the sustainability of software. Some of these are important for almost all software projects, but some aspects are especially relevant for scientific projects. Out of the latter the authors picked four of the most relevant to discuss in more detail.

3.1. Modular Design

One of the key properties of a long-term sustainable scientific software project is a modular design. Cactus chose a unique method of enabling modularity that went beyond the usual notions of APIs, standard data structures, and coding conventions. Cactus uses a small set of domain specific languages (DSLs) to describe its distributed data structures and scheduling [17]. These DSLs enable Cactus modules to do run-time reflection, in both Fortran and C/C++, on the grid variables being evolved.

This may not sound like a revolutionary idea, but its consequences were far-reaching. Because of this simple design decision, several things became possible. First, Cactus was able to completely decouple I/O from science code. Unlike many scientific codes that have calls to I/O routines interspersed with the program logic, science modules in Cactus are only concerned with what variables they read and write. The I/O module(s) can take field variable names as a parameter, look them up with run-time reflection, and write them out as text, HDF5, JPEG and other formats. The DSL describing scheduling identifies when this I/O will be performed, and can be set at startup or steered at run-time by the user. Even the most important form of I/O, checkpoint/restart, is enabled and modularized by this design. Every variable known to the framework, and not explicitly excluded, will automatically be check-pointed and re-read when necessary.

Second, Cactus was able to abstract the time-integration method (e.g. Runge-Kutta, Iterative Crank-Nicholson, etc.) from the time evolution equations. The DSL describing scheduling, combined with the list of variables to be evolved, was sufficient for this task. The value of this module by itself is significant. The ability to avoid subtle coding bugs or to try out diverse integrators, without cluttering the codebase, is of great value. This time integration module can also easily be used to couple separate physics modules, such as e.g. the Einstein and Hydrodynamics equations.

Third, Cactus was able to create a web browser interface, interrupting the schedule tree at key places and allowing variables to be inspected and modified during an execution. This particular form of parameter steering was naturally enabled by the key module design decisions.

Fourth, an adaptive mesh refinement module named Carpet [18, 19] was added. Before Carpet was available, only uniform, Cartesian grids could be used for spatial decomposition within Cactus, and a lot of modules expected to get such rectangular meshes as input. It was possible to integrate Carpet into Cactus with little change to the science modules. Later, a multi-block mesh capability [20] was added with similarly little disruption to existing codes.

There are many other capabilities enabled by the unique modularization decisions of the Cactus Framework. Other special modules include those for debugging, e.g. NaNChecker, those enabling unusual IO, e.g. Twitter, generic grid modules, generic boundary conditions, timers, interfaces to PAPI counters, analysis modules, etc.

The usage of modules within the Cactus framework can be compared to the use of interfaces in modern object-oriented languages. Cactus modules not only provide a new "object" for the framework to work with, they also encapsulate a self-describing interface to that object within the Cactus framework.

These unique types of interfaces allow different groups to efficiently work on one common project, towards one common goal, avoiding unnecessary conflicts or duplication of effort. Usually, this modularization does not cause performance problems. Most of the time in our simulations is spent in scientific kernels, and only a small percentage is required for the infrastructure.

3.2. Growing Collaborative Community

In contrast to commercial products, academic scientific software like Cactus is usually developed in a university setting. Most of the actual development work is performed by graduate students and postdoctoral researchers who are focused on science. This poses a threat for long-term stability of any project, because these developers are typically not very interested in contributing to infrastructure, and frequently leave their research groups after three to four years, taking all their knowledge and experience with them. New members of research groups first need to be trained, and while this time can be shortened by creating respective courses and documentation, e.g., [21–23], the constant flux of developers continues to be a struggle.

The infrastructure development problem is handled within the Cactus community by connecting its development so closely to a specific science problem that publications that describe both are possible. This is the way many of the publications using and extending the Einstein Toolkit are written, containing a description of the new infrastructure while mainly representing a publication in physics. On the other hand there are examples of pure computational science publications as well, e.g., [24–29]. These sorts of projects are enabled by the ability to provide immediate benefits for the substantial number of physics problems described by the Einstein Toolkit.

The retention problem is handled by the Cactus community automatically, by creating an enjoyable programming and development experience. Many students continue to be interested in and develop for Cactus after they move on to new positions. They are attracted by the ability to leverage the work of other physicists and computer scientists in the community, to see their code re-used, and to collaborate on new research. The ability of Cactus to enable collaboration, based on its unique modular design and the use of open-source licensing, make it an attractive tool for students to use in their continuing research. Thus, rather than really leaving, many serve to expand the collaboration. Not all students continue to use Cactus, but enough are inspired by it to create the kind of stable contributions needed to maintain the health of the project.

3.3. Career paths

The third issue connects to the motivation of the workforce as well, but also affects the group leaders, especially if they are young faculty or trying to become faculty. In contrast to commercial products, the principle motivation of developers in academia is *credit.* Credit is obtained through publications and citations thereof, which are then used to quantify the scientific impact of an individual's work, forming the basis for future career plans as well as promotion and tenure for faculty. The most severe problem for developers in most computational sciences, currently, is that while most of the work is done creating well-written, sustainable software, the academic success is often exclusively tied to the solution of the scientific problem the software was designed for. Tasks that are essential from a software engineering standpoint, e.g., high usability, clean code, updated documentation, or porting infrastructure to new platforms, are not rewarded within this system.

While computations of some form are required for almost every aspect of academic research, the criteria for tenure positions focus solely on scientific results and fail to appreciate the infrastructural development that was necessary to achieve these results. Developers of scientific software are often experts across disciplines, and thus valuable as team members, group leaders, and lecturers. However, the scope of open tenure positions is often only very limited to single specific traditional science, resulting in a lack of career opportunities and thus a lack of student motivation to fully participate in these cross-discipline activities.

3.4. Credit

One way that credit is given to software developers is through citation of a paper describing the code. Widely used codes can lead to highly cited papers, which can help the author's academic career. Unfortunately, it is not uncommon that relevant citations are missing despite the usage of a specific software package. Enforcement of citation through licenses is possible to some degree, but such requirements tend to limit the desired level of openness, and because there are legitimate cases where such citations are not possible, e.g., because of space limitations.

Both the Cactus framework and its largest user group, the Einstein Toolkit, have a few general publications that are requested (but not required) to be cited whenever the software is used, and especially the Einstein Toolkit also requests additional citations when some of its modules are used. Examples of such modules are the Carpet mesh-refinement infrastructure, or the black hole horizon finder [30]. Handling citations on a module-level like this enables individual developers to receive credit for their work, especially if they entered the group only after the main publications had already been written. A complete list of these publications can be found on the Einstein Toolkit web site [4].

4. Automated Code Generation

Computational science is always evolving, and at an ever increasing pace. Changes in hardware and progress in numerical methods have been important factors in the quest to solve increasingly complex problems within a relatively constant time. The Cactus group is looking for ways to use modularization to attack these new types of problems.

Prior to the discovery of a stable set of numerical techniques for evolving black holes, it was necessary to experiment with the form of the Einstein equations. Unfortunately, these equations contain many hundreds of terms and transforming them into code was daunting, tedious and error-prone. Sascha Husa and Ian Hinder developed a tool called Kranc [31, 32] to mitigate this problem. Kranc takes a tensorial equation written in Mathematica and generates a complete Cactus module in C++ to evaluate it, enabling scientists to more easily experiment with the form of equations. Kranc turned out to be a much more powerful and useful tool than anticipated.

Modern architectures increasingly rely on vectorization to achieve performance, many of them now perform eight floating point instructions per clock cycle. Unfortunately, few compilers can vectorize the Einstein equations because of the sheer number of terms they contain, leading to a large performance penalty.

Recent work shows that Mathematica's pattern matching ability allows us to transform equations as Kranc generates them, inserting explicit vector instructions and side-stepping compiler limitations, and thereby achieving higher performance.

5. Newer Challenges

Until recent times, the modular structure of Cactus has allowed developers to easily make use of new ideas, algorithms, and technologies. To some extent, OpenMP style parallelism disrupted that modularity by intermixing parallel logic and code development, but the problems caused by this change have been relatively minor.

More recent challenges posed by accelerators have proved to be a more challenging disruption. Because Kranc isolates the high-level representation of the equations from their implementation in code, it makes it possible to generate code for multiple languages (such as CUDA, and OpenCL), it makes it possible to isolate the high level equations from the low level code implementation details required for high performance.

6. Conclusion

The key to Cactus' previous success on every level has been a result of its unique choices for modularization. The ability for the code to reflect on itself has enabled a number of valuable infrastructure features.

Recent challenges in hardware design have disrupted that modular design, but equation generation appears to provide a possible path forward, separating high level equations from their low level representation in code. This present direction of research is being studied by the Chemora [28] project, and work in this area is ongoing.

Acknowledgments

The authors would like to thank Edward Seidel, for his inspiration and vision, his support and guidance over many years of development of the framework. We would like to thank the early pioneers of Cactus, including Paul Walker, Joan Massó, Tom Goodale, and Thomas Radke. More recently, we have to especially thank developers like Ian Hinder and Roland Haas, whose contributions have been invaluable for the community. We are also grateful to all the other people who contributed to Cactus via ideas, code, documentation, and testing; without these contributions, this framework would not exist today.

Cactus was and is developed with support from a number of different sources, including support by the US National Science Foundation under the grant numbers 0903973, 0903782, 0904015 (CIGR) and 1212401, 1212426, 1212433, 1212460 (Einstein Toolkit), 0905046, 0941653 (PetaCactus), 1265449 (Chemora/EAGER), 1047956 (Eclipse/PTP), a Deutsche Forschungsgemeinschaft grant SFB/Transregio 7 (Gravitational Wave Astronomy), and a Canada NSERC grant.

Notes

[1] The first accurate black hole inspiral was finally modeled in 2007.

References

1. Cactus Computational Toolkit. Available at: http://www.cactuscode.org/
2. **Goodale, T, Allen, G, Lanfermann, G, Massó, J, Radke, T, Seidel, E,** et al 2003 The Cactus Framework and Toolkit: Design and Applications. In: Vector and Parallel Processing - VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science. Berlin: Springer. Available at: http://edoc.mpg.de/3341
3. **Löffler, F, Faber, J, Bentivegna, E, Bode, T, Diener, P, Haas, R,** et al 2012 The Einstein Toolkit: A Community Computational Infrastructure for Relativistic Astrophysics. *Classical and Quantum Gravity*, 29(11): 115001. DOI: http://dx.doi.org/10.1088/0264-9381/29/11/115001
4. Einstein Toolkit: Open software for relativistic astrophysics. Available at: http://einsteintoolkit.org/
5. **MacNeice, P, Olson, K M, Mobarry, C, de Fainchtein, R** and **Packer, C** 2000 PARAMESH: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications,* 126(3): 330–354. DOI: http://dx.doi.org/10.1016/S0010-4655(99)00501-9
6. Parallel Adaptive Mesh Refinement PARAMESH. Available at: http://www.physics.drexel.edu/~olson/paramesh-doc/Users_manual/amr.html
7. **The Enzo Collaboration, Bryan, G L, Norman, M L, O'Shea, B W, Abel, T, Wise, J H,** et al 2013 Enzo: An Adaptive Mesh Refinement Code for Astrophysics. ArXiv e-prints, July 2013.
8. Enzo astrophysical AMR code. 2013 Available at: http://enzo-project.org/
9. BoxLib 2011. Available at: https://ccse.lbl.gov/BoxLib
10. **Colella, P, Graves, D, Keen, N, Ligocki, T, Martin, D, McCorquodale, P,** et al 2009 Chombo Software

Package for AMR Applications Design Document. In: *Applied Numerical Algorithms Group*. Computational Research Division: Lawrence Berkely National Laboratory.

11. **Parker, S G** 2006 A component-based architecture for parallel multi-physics PDE simulation. *Future Generation Computer Systems*, 22(1-2): 204–216. DOI: http://dx.doi.org/10.1016/j.future.2005.04.001
12. **Parker, S G, Guilkey, J** and **Harman, T** 2006 A component-based parallel infrastructure for the simulation of fluid-structure interaction. *Engineering with Computers*, 22: 277–292. DOI: http://dx.doi.org/10.1007/s00366-006-0047-5
13. **Mo, Z, Zhang, A, Cao, X, Liu, Q, Xu, X, An, H,** et al 2010 JASMIN: a parallel software infrastructure for scientific computing. *Frontiers of Computer Science in China*, 4(4): 480–488. DOI: http://dx.doi.org/10.1007/s11704-010-0120-5
14. **CASC** 2007 SAMRAI Structured Adaptive Mesh Refinement Application Infrastructure. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory. Available at: https://computation.llnl.gov/casc/SAMRAI/
15. **Hornung, R D** and **Kohn, S R** 2002 Managing application complexity in the SAMRAI object-oriented framework. *Concurrency and Computation: Practice and Experience*, 14(5): 347–368. DOI: http://dx.doi.org/10.1002/cpe.652
16. **Christen, M** 2012 Automatic thread-level parallelization in the chombo amr library. Available at: http://escholarship.org/uc/item/6xm611zp
17. **Allen, G, Goodale, T, Löffler, F, Rideout, D, Schnetter, E** and **Seidel, E L** 2010 Component Specification in the Cactus Framework: The Cactus Configuration Language. In: Grid2010: Proceedings of the 11th IEEE/ACM International Conference on Grid Computing.
18. **Schnetter, E, Hawley, S H** and **Hawke, I** 2004 Evolutions in 3-D numerical relativity using fixed mesh refinement. *Classical and Quantum Gravity*, 21: 1465–1488. DOI: http://dx.doi.org/10.1088/0264-9381/21/6/014
19. Carpet: Adaptive Mesh Refinement for the Cactus Framework. Available at: http://www.carpetcode.org/
20. **Schnetter, E, Diener, P, Dorband, E N** and **Tiglio, M** 2006 A multi-block infrastructure for three-dimensional time-dependent numerical relativity. Classical and Quantum Gravity, 23: S553–S578. DOI: http://dx.doi.org/10.1088/0264-9381/23/16/S14
21. **Allen, G, Benger, W, Hutanu, A, Jha, S, Löffler, F** and **Schnetter, E** 2011 A practical and comprehensive graduate course preparing students for research involving scientific computing. In: Proceedings of the International Conference on Computational Science, ICCS 2011. Procedia Computer Science, pp. 1927–1936.
22. **Löffler, F, Allen, G, Benger, W, Hutanu, A, Jha, S** and **Schnetter, E** 2011 Using the Ter-aGrid to teach scientific computing. In: Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery, TG'11. New York, NY, USA: ACM, pp. 55:1–55:7. DOI: http://doi.acm.org/10.1145/2016741.2016800
23. **Zilhão, M** and **Löffler, F** 2013 An Introduction to the Einstein Toolkit. *arXiv*, 1305.5299 [accepted to IJMPA].
24. **Hutanu, A, Schnetter, E, Benger, W, Bentivegna, E, Clary, A, Diener, P,** et al 2010 Large-scale Problem Solving Using Automatic Code Generation and Distributed Visualization. *Scalable Computing, Practice and Experience Scientific International Journal for Parallel and Distributed Computing,* 11(2): 205–220. Available at: http://www.scpe.org/vols/vol11/no2/SCPE_11_2_10.pdf
25. **Seidel, E L, Allen, G, Brandt, S, Löffler, F** and **Schnetter, E** 2010 Simplifying complex software assembly: the component retrieval language and implementation. In: Proceedings of the 2010 TeraGrid Conference. TG'10. New York, NY, USA: ACM, pp. 18: 1–18:8. *arXiv*, 1009.1342. DOI: http://doi.acm.org/10.1145/1838574.1838592.
26. **Allen, G, Goodale, T, Löffler, F, Rideout, D, Schnetter, E and Seidel, E L** 2010 Component Specification in the Cactus Framework: The Cactus Configuration Language. In: Grid2010: Proceedings of the 11th IEEE/ACM International Conference on Grid Computing. *arXiv*, 1009.1341.
27. **Zebrowski, A, Löffler, F** and **Schnetter, E** 2011 The BL-Octree: An Efficient Data Structure for Discretized Block-Based Adaptive Mesh Refinement. In: ParCo2011: Proceedings of the 2011 International Conference on Parallel Computing. Prof. Dr Frans J. Peters, Poolsterlaan 6, 5632 AN Eindhoven, The Netherlands: ParCo Conferences.
28. **Blazewicz, M, Brandt, S R, Diener, P, Koppelman, D M, Kurowski, K, Löffler, F,** et al 2011 A Massive Data Parallel Computational Framework for Petascale/Exascale Hybrid Computer Systems. In: ParCo2011: Proceedings of the 2011 International Conference on Parallel Computing. Prof. Dr Frans J. Peters, Poolsterlaan 6, 5632 AN Eindhoven, The Netherlands: ParCo Conferences.
29. **Blazewicz, M, Hinder, I, Koppelman, D M, Brandt, S R, Ciznicki, M, Kierzynka, M,** et al 2013 From physics model to results: An optimizing framework for cross-architecture code generation. *Scientific Programming*, 21(1–2): 1–16. *arXiv*, 1307.6488. DOI: http://dx.doi.org/10.3233/SPR-130360.
30. **Thornburg, J** 2004 A Fast Apparent-Horizon Finder for 3-Dimensional Cartesian Grids in Numerical Relativity. *Classical and Quantum Gravity,* 21: 743–766. DOI: http://dx.doi.org/10.1088/0264-9381/21/2/026
31. **Husa, S, Hinder, I** and **Lechner, C** 2006 Kranc: a Mathematica application to generate numerical codes for tensorial evolution equations. *Computer Physics Communications*, 174(12): 983–1004. DOI: http://dx.doi.org/10.1016/j.cpc.2006.02.002
32. Kranc: Kranc Assembles Numerical Code. Available at: http://kranccode.org/

How to cite this article: Löffler, F, Brandt, S R, Allen, G and Schnetter, E 2014 Cactus: Issues for Sustainable Simulation Software. *Journal of Open Research Software,* 2(1): e12, pp. 1-6, DOI: http://dx.doi.org/10.5334/jors.au

Published: 9 July 2014

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press.

OPEN ACCESS