

Common Computational Frameworks as Benchmarking Platforms

John Shalf,^{1,*} Erik Schnetter,^{2,†} Gabrielle Allen,^{2,‡} and Edward Seidel^{2,§}

¹*NERSC Division, Mail Stop 50F, Lawrence Berkeley Laboratory,
The University of California, Berkeley, CA 94720, USA[¶]*

²*Center for Computation & Technology, 302 Johnston Hall,
Louisiana State University, Baton Rouge, LA 70803, USA^{**}*

(Dated: September 15, 2005)

Computational Frameworks, supporting multiple algorithms and applications with common parallel, I/O, and other computational routines, can provide an excellent substrate for application-scale benchmarks. Such benchmarks are essential for estimating the effective performance of HPC systems for the purpose of system procurements. Cactus is extremely portable and its modularity supports a variety of communication substrates and numerical methods — offering wide-coverage of the HPC applications domain.

HPC procurements require more sophisticated methods to gauge the effectiveness of the system than just the speeds and feeds that are typically supplied by hardware vendors. HPC systems are typically presented based on raw hardware characteristics — the bandwidth of the interconnect, the peak floating point performance of the processors, the number of processors, the memory bandwidth, etc. . .). Without additional information, the systems may end up being compared based on their peak performance even though they are likely to deliver only a fraction of their peak performance in real use. Even references to SpecFP benchmarks provide little indication of the systems' effectiveness for parallel jobs in the context of a production supercomputing center because SpecFP does not exercise the interconnect, nor does the SpecFP workload have any guaranteed relationship to an HPC center's workload.

The overriding question for an HPC procurement team is: "what performance will this system deliver for our workload?" A procurement team needs a normalizing metric that tells them how efficiently the architecture can execute their typical workload given the proposed system's peak performance so that different systems can be compared on an equal footing. This need and the general methodology for approaching the problem was outlined in a 1982 technical report by Ingrid Bucher and Joanne Martin entitled "Methodology for Characterizing a Scientific Workload". These principles are embodied in the construction of the NAS kernel benchmarking program (Bailey & Barton, 1985), and form a consistent path to the construction of the Sustained System Performance metric used by NERSC for its procurements. Their method is founded on using workload characterization to select a representative set of codes to evaluate the effective performance of the system for real application workloads. The metric for the "effective system

*Electronic address: jshalf@lbl.gov

†Electronic address: schnetter@cct.lsu.edu

‡Electronic address: gallen@cct.lsu.edu

§Electronic address: eseidel@cct.lsu.edu

¶URL: <http://www-vis.lbl.gov/>

**URL: <http://www.cct.lsu.edu/>

performance” is based on the weighted harmonic mean of the execution time of the representative set of codes.

Implementing such an approach can be difficult in practice. All too often, scientific codes are very complex and difficult to port to a wide variety of platforms. Often, assumptions about the communication layer and language model are deeply embedded in the construction of the code — for instance, the use of MPI messaging as opposed to shared memory. Using an individual code-base for testing may offer far too narrow a view of the algorithms as the implementation and algorithms are often inextricably tied together. Under such conditions, it can be very difficult to offer a maintainable benchmarking code base that offers wide coverage of algorithms and implementations. We propose using the Cactus framework as a substrate for more effective HPC system benchmarking and comparisons.

I. CACTUS

The Cactus framework is a modular, open source, highly portable programming environment for collaborative HPC computing, primarily developed at LSU. Cactus has a generic parallel computational toolkit with modules providing parallel drivers, coordinates, boundary conditions, elliptic solvers, interpolators, reduction operators, and efficient I/O in different data formats. Generic interfaces are used, (e.g. an abstract elliptic solver API) making it possible to use external packages (e.g. PETSc) and improved modules, which are immediately available to its users.

In numerical relativity, Cactus is used to build codes for solving Einstein’s equations by over two dozen groups around the world. Cactus is also increasingly used in areas such as CFD, climate modeling, astrophysics, biological computing, chemical engineering, and others; and is a driving framework for a number of computing infrastructure projects, particularly in Grid Computing, e.g. GrADS, GridLab, GriKSL, the ASC, and others, bringing a rich set of computational science capabilities, such as robust checkpoint/restart, remote parameter steering and interaction, and high-performance I/O back to its application domain users. Scientists in each of these domains chose Cactus also because of its portability and scalability on high performance computing architectures.

II. PORTABILITY

Cactus is actively ported to a wide variety of platforms from Linux clusters to exotic platforms like the Cray X1, BG/L and the Hitachi SR/8000. Cactus was one of a handful of US codes that were ported and benchmarked at large scale on the Earth Simulator system in Japan. Because it is the framework for a number of active research projects around the world, the code base is actively developed and constantly ported and tested on emerging computing systems.

The entire Cactus build infrastructure automates the compilation of very complex codes in an architecture-independent manner without any complicated makefile editing. It allows scientists to write their routines in languages of their choice (C, C++, Fortran 77, or Fortran 90), while Cactus uses the GNU autoconf mechanism to automatically detect compiler and operating system properties. Cactus has built in a knowledge database

of today's compilers to handle inter-language calls and other compiler idiosyncracies. The Cactus developers maintain also in conjunction with selected Cactus power users so-called "Cactus configurations", which are descriptions of how to access third-party libraries (such as LAPACK, HDF5, PETSc etc.) on various machines, since each super-computing center follows its own guidelines for installed scientific software.

III. MODULARITY

Although the root of the Cactus lies historically in the numerical relativity community, Cactus itself is only a framework, and the numerical relativity application routines are only found in certain physics modules which are not part of Cactus proper. One can swap in a variety of different applications and numerical methods into Cactus. Stand-alone codes like Zeus or Flow-er have been ported to Cactus, and none of them have any physics code in common with the numerical relativity codes — but they can share the same computational infrastructure.

The original numerical relativity codes were based on second order finite differencing. Since then, scientists at LSU have implemented in Cactus higher order finite differencing operators, and cooperation partners in Tübingen (Germany) have provided modules for adaptive mesh refinement. These sets of modules can now be used by existing codes. A second mesh refinement driver based on SAMRAI is currently being developed at LSU, which uses a different AMR algorithm.

IV. PLUGGABLE COMMUNICATION MODEL

In addition to supporting modular replacement/substitution of applications and numerical methods, the Cactus infrastructure allows the communication model to be replaced at runtime. Vendors of Shared Memory systems complain that their results are handicapped by benchmarks that require MPI even though lighter weight communication and synchronization methods are available. Cactus offers considerable segregation of the parallel/communication layer from the serial/computation layer. This enables the user to change from a purely SHMEM based communication layer to a purely MPI based layer using only a single-line change in the configuration file. The numerical methods and application codes remain unchanged, thereby offering a fairer platform for comparing interconnect architecture.

V. ALGORITHM ADAPTABILITY

Different unigrid and mesh refinement drivers can coexist in the same executable, and can be chosen at run time through parameter files. The very same application codes can thus be run under these different drivers, which makes it easy to study the impact of the different grid structures and inter-grid boundary operators onto the performance on a particular hardware. One can also study the correlation between different orders of accuracy on the required minimum grid sizes to achieve a certain solution accuracy, thus comparing science output instead of floating point performance.

Especially the ability to use the very same executable for all benchmarks, and the ability to use a standard set of parameter files for the various combinations of algorithms

and application modules make Cactus a versatile platform for benchmarking real applications. One can also consider using realistic parameter files for integration benchmarks, which would test not only one system component, but test floating point performance, communication throughput, and I/O scalability at the same time in a realistic manner.

VI. CONCLUSION

Cactus would make an excellent benchmarking platform for future NSF HPC procurements. Its extreme portability to literally all current HPC platforms and its proven scalability to thousands of nodes make it ideally suited. It is currently used in production mode in several application fields, such that there already exist efficient codes that can form the basis of a standardized set of benchmarks. Taken as a platform, it allows current and future kernels to be implemented within its framework, such that a single executable could be used for all tests, additionally easily enabling integration tests of the machine as a whole instead of only its components. We believe that using Cactus as benchmarking platform would thus, in the end, also lead to more realistic and more reliable benchmarking results.