ALPACA: CACTUS TOOLS FOR APPLICATION LEVEL PERFORMANCE AND CORRECTNESS ANALYSIS

ERIK SCHNETTER$^{(1,2)}$, GABRIELLE ALLEN$^{(1,3)}$, TOM GOODALE$^{(1)}$, AND MAYANK TYAGI$^{(1,4)}$

ABSTRACT. Although the speed and performance of high end computers have increased dramatically over the last decade, the ease of programming such parallel computers has not progressed. The time and effort required to develop and debug scientific software has become the bottleneck in many areas of science and engineering. The difficulty of developing high-performance software is recognised as one of the most significant challenges today in the effective use of large scale computers.

Cactus is a framework for science applications which is used to simulate physical systems in many fields of science, such as black holes and neutron stars in general relativity. As in other software frameworks, applications are built from separately developed and tested components. Below we outline *Alpaca*, a concept and a project to develop high-level tools to allow developers and end-users to examine and validate the correctness of an application, and aid them in measuring and improving its performance in production environments. These tools are components themselves, built into the application and interacting with it. Alpaca's approach includes help to render applications tolerant against partial system failures, which is becoming a pressing need with tomorrow's architectures consisting of tens of thousands of nodes.

In contrast to existing debuggers and profilers, Alpaca's approach works at a much higher level, at the level of the physical equations and their discretisations which are implemented by the application, not at the level of individual lines of code or variables. It is not enough for only the main kernels to be correct and show good scalability – the overall application, which may contain many smaller modules, must perform. We assume that Alpaca's integrative ansatz will lead to well-tested and highly efficient applications which are developed in a shorter time scale and execute more reliably.

1. MOTIVATION

The application of high performance computing to solving increasingly complex problems in science and engineering, such as the nature of a gamma-ray burst or the function of a complete living organism, is at a critical crossroads in at least three areas: (*i*) *Hardware*: Radically new petascale architectures, built at unprecedented scales exceeding a million processors, 2-3 orders of magnitude larger than current large scale systems, are being designed for deployment; (*ii*) *Software*:

Date: December 30, 2007.

Email: schnetter@cct.lsu.edu.

Web: http://www.cct.lsu.edu/.

(1) Center for Computation & Technology, Louisiana State University, Baton Rouge, LA, USA.

(2) Department of Physics and Astronomy, Louisiana State University, Baton Rouge LA, USA.

(3) Department of Computer Science, Louisiana State University, Baton Rouge, LA, USA.

(4) Department of Petroleum Engineering, Louisiana State University, Baton Rouge, LA, USA.

Standard approaches to system software, debugging tools, domain decomposition, message passing, and execution models are outdated, and new approaches are being developed; (*iii*) *Complex applications*: Traditional, simplified, static applications, developed by single groups are evolving towards highly complex codes that aim to capture the complexity of nature, including and coupling myriad physical effects, and utilising adaptive data structures, that require teams of researchers and computer scientists to develop and use.

These developments create immense and critical problems for the application scientists and engineers who must develop these codes and use these environments for their science. There is a growing recognition that the community is facing a *software crisis*. It is already increasingly difficult for teams of computational scientists to develop comprehensive applications that capture the complexity of the natural systems under study on today's HPC systems; this problem may become crippling as petascale systems are introduced with potentially very different architectures, execution models, libraries, etc.

At the heart of this crisis is the dearth of software tools that aid in the development of complex, collaborative scientific applications, appropriate for highly scalable hardware architectures, providing fault tolerance, advanced debugging, and transparency against new developments in communication, programming, and execution models. Such tools are especially rare *at the application level*, where they are most critically needed.

We propose to address this set of problems through the development of powerful, platform independent, portable extensions to the well known Cactus Computational Toolkit, widely used by dozens of application and computer science groups around the world, in a diverse set of application areas from astrophysics to coastal modelling. We focus our attention on several key issues facing scientific and engineering application developers.

Specifically, below we will address the following issues, *at the application level*: (*i*) *fault tolerant capabilities* that will be needed for increasingly large scale machines, where system and hardware faults are likely to be much more common than they are at present (section 3.6); (*ii*) *performance monitoring capabilities*, which will make it much easier for application developers and users to determine how their more complex application codes perform on current and future hardware (section 3.4); (*iii*) *interactive debugging capabilities*, critical to locate and cure software or algorithmic errors in such complex applications which may be developed by international collaborations (section 3.3); and (*iv*) *integration with a common user interface*, such as Eclipse, the increasingly popular open source code development environment, making it much easier to develop code (sections 3.2 and 3.5). In addition, (*v*) such tools need to be developed *with full involvement from application developers* across a broad range of areas (section 3.1).

The Alpaca project is has received three years' funding from the NSF SDCI programme.

2. Cactus Framework

The Cactus Framework [30, 24] is an open-source, modular, and portable programming environment for collaborative HPC computing. It was designed and written specifically to enable scientists and engineers to develop and perform the large-scale simulations needed for modern scientific discovery across a broad range of disciplines. From the outset, Cactus has followed two fundamental tenets: *driven by user needs* and *embracing and exploiting new technologies to advance science*. As described below, Cactus is used by a wide and growing range of applications.

Development of Cactus and its associated components (modules) has been driven from the beginning by user requirements. This has been achieved by developing, supporting, and listening to a large user base. Among these needs have been ease of use, portability, support of large and geographically diverse collaborations, and the ability to handle enormous computing resources, visualisation, file I/O, and data management. Cactus must also support the inclusion of legacy code, as well as a range of programming languages. It is essential that any living framework be able to incorporate new and developing cutting edge computation technologies and infrastructure (e.g. the Cell processor or GPGPU accelerators [46]), with minimal or no disruption to its user base. Some of the key strengths of Cactus have been its portability and high performance, which led to it being chosen by Intel to be one of the first scientific applications deployed on the IA64 platform. For example, a Cactus application was recently benchmarked on the IBM BG/L system at IBM T. J. Watson and scaled well up to 32,768 processors [19].

2.1. **Application Users.** Work started on Cactus in 1997 at the Albert-Einstein-Institut in Potsdam, Germany, where it was developed with researchers at Washington University in St. Louis and at the NCSA as the core simulation code for numerical relativity for an international collaboration. An early version of this code was involved in the NASA grand challenge neutron-star modelling project and was delivered to NASA as the `GR3D` code [1]. Cactus is now used by over two dozen numerical relativity groups for their cutting edge research. The Cactus numerical relativity kernels have long been used as benchmarks for hardware architectures [21, 2]. A version has been incorporated into the SPEC CPU2006 benchmarking suite [16].

Although numerical relativity remains the main Cactus user base, the computational framework is generalised, and Cactus is now increasingly being used for scientific investigations in a wide range of other application areas including astrophysics, quantum gravity, chemical engineering, Lattice Boltzmann Methods, econometrics, computational fluid dynamics, and coastal and climate modelling [32, 31, 38, 34, 28, 44, 20, 25]. Consequently, the Alpaca project can draw on the expertise of developers and users in several of these areas, listed in table 1, and described in more detail in section 3.1 below.

Cactus is also seen as a prime enabling environment for petascale computing [42]. Its impressive scaling and portability provide an ideal base to develop petascale applications. The Alpaca project aims at providing debugging, profiling, and fault-tolerance features which are essential to effective development of such applications and effective use of petascale machines. The profiling infrastructure in particular needs to facilitate the development of new parallel driver components in order to allow the efficient use of multi-core processors and hardware architectures with a hierarchy of bandwidths and latencies.

2.2. **Current Cactus Development.** As with most frameworks, the Cactus code base is structured as a central part, called the *flesh* that provides core routines, and components, called *thorns*. The flesh is independent of all thorns and provides the main programme, which parses the parameters and activates the appropriate

Application	List of Key Computational Requirements
Numerical Relativity	Large scale simulations. Interest and involvement in new architectures and paradigms. Adaptive mesh refinement introduces new performance issues which are addressed by Alpaca tools.
Computational Fluid Dynamics	Multi-block simulations and unstructured meshes lead to difficulties in load balancing. Many existing packages need to be integrated. Using the Cactus CFD Toolkit as educational HPC tool.
Reservoir Simulations	High-throughput simulations. Complex geometries, elaborate physical models.
Coastal Modelling	Simulations require robustness & reliability. Long-term simulations (many time steps) on massively parallel computers.
Quantum Gravity	Young field, requires experimenting with a wide variety of algorithms, not necessarily PDE based. Performance crucial.
Astrophysics	Large scale simulations. Interest and involvement in new architectures and paradigms. Efficient solvers for elliptic equations which are addressed by Alpaca tools.

TABLE 1. A cross section of current Cactus use across a wide range of fields. Alpaca can rely on the expertise of Cactus application developers and users listed in this table to provide detailed requirements and use cases. This will help test and improve tool implementations, ensure that software works effectively on a wide range of architectures, and aid in providing documentation and support to a wider user community.

thorns, passing control to thorns as required. By itself, the flesh does very little science; to do any computational task the user must compile in thorns and activate them at runtime.

A thorn is the basic working component within Cactus. All user-supplied code goes into thorns, which are, by and large, independent of each other. Thorns communicate with each other via calls to the flesh API or, more rarely, custom APIs of other thorns. The Cactus component model is based upon tightly coupled subroutines working successively on the same data, although recent changes have broadened this to allow some element of spatial workflow. The connection from a thorn to the flesh or to other thorns is specified in configuration files that are parsed at compile time and used to generate glue code that encapsulates the external appearance of a thorn. At runtime, the executable reads a parameter file that details which thorns are to be active and specifies values for the control parameters for these thorns.

User thorns are generally stateless entities; they operate only on data which are passed to them. The data flow is managed by the flesh. This makes for a very robust model where thorns can be tested and validated independently, and can be combined at run-time in the manner of a functional programming language. Furthermore, thorns contain test cases for unit testing. Parallelism, communication, load balancing, memory management, and I/O are handled by a special component called *driver* which is not part of the flesh and which can be easily replaced. The flesh (and the driver) have complete knowledge about the state of the application, allowing inspection and introspection through generic APIs.

The current version of Cactus provides many computational modules for finite difference based methods, and has been very successful as indicated by the large number of scientific publications it has enabled (see above). There exist currently (December 2007) approximately 540 thorns in over 50 arrangements at the AEI and LSU, many of which are publicly available. We are currently developing a new version of the core (Cactus 5) which will extend key support for other application domains requiring particle methods, unstructured meshes, multi-patch, multi-physics, or multi-domain simulations. We have implemented proof of concepts of these with the current Cactus version, and will use this experience to design efficient generic interfaces for these.

All Cactus development will continue to be an open, community driven process and based on actual user needs.

3. Alpaca Research and Development

Petascale computing environments are inherently complex, and it is a tedious task to program for them. Current tools are not adequate; new programme development tools need to be developed to help meet this challenge. It is still an open question what these tools will look like in the end. As they are developed, new methodologies will need to be tried, and user experience needs to be fed back into the development process.

There exist many tools to measure performance and to debug applications. Most of these tools work today at a very low level; for example, parallel debuggers allow examining variables at the level of Fortran variables, and allow single stepping of individual routines or lines of code. While this is important for finding uninitialised variables and segmentation faults, it is not sufficient to ensure that the "physics" of the overall simulation is correct.

Similarly, performance measurement tools allow one to time routines, or to count the number of executed floating point instructions. This level of detail is important for certain kinds of performance tuning. However, the overall performance of a simulation depends also very much on the run-time parameters that are chosen by the end user, which e.g. decide what grid structure and what resolution is used. Thus the end user needs to be involved in the performance tuning process. We envision the scenario described in example 1 to be a showcase use of the Alpaca tools.

Without high-level tools such as envisioned by Alpaca, the user in this scenario would have had to submit many runs, repeatedly wasting time in the job queue. She would have had to add "print statements" (explicit calls to file I/O routines)

Gabriela, a young postdoc from Córdoba in Argentina, wants to perform the final tests for her new wave extraction module. She takes a set of well-tested components for binary black hole initial data, time evolution, boundary conditions, etc., and adds her new module to it. After building the application on the Tezpur supercomputer at LSU, she submits a job using a new parameter file she created.

Using the Alpaca debugger user interface, she watches the signal as the gravitational waves are detected by her module. She notices that the waveform amplitude increases with radius, which is unphysical. Still using the debugger, and still from within the same job, she walks through the individual algorithmic steps of her wave extraction module. She notices that the problem is caused by the lapse function, which has unexpectedly small values at small radii. Correspondingly, she switches to a different gauge condition, and after a few iterations the lapse starts to grow. Since this effect is only visible in binary black hole systems, she could not have detected it in a test run on a single-processor machine. After correcting this problem, she moves on to setting up a simulation with a higher resolution to reproduce a known published result.

While waiting for the results of this simulation, she notices that the simulation makes only slow progress. Using the same Alpaca user interface, she activates some interactive performance monitoring tools for this run. These tools profile the ongoing simulation, and then access a server with "performance experience" from earlier runs without her new module, showing her that her new simulation runs only half as fast as "it should". Having this background knowledge, she is able to pinpoint the problem to a recent change in the horizon finder – not in her own code, as she first assumed. She then sends an email to the horizon finder developer asking for advice.

Example 1: A typical envisioned use of the Alpaca tools.

to get access to intermediate data. Tracking down the source of the performance problem would have required her to submit several production jobs, modifying her parameter files to obtain timing data for comparison. This tedious procedure would have cost her at least a week of wrestling supercomputers and their job submission systems. By using high-level tools, and by interacting with a simulation instead of merely observing it, she was able to achieve her goals in a single day.

The Alpaca approach follows a multi-layered approach. Effective understanding of code and machine requires, on one hand, an interactive component (i.e., a user interface) allowing remote access to production level simulations on supercomputers. It requires on the other hand a software framework allowing interaction with a running simulation, to obtain data about it and to be able to steer it. We suggest to provide this through a special API, implemented for the Cactus framework, and also for science applications which do not use Cactus.

Today's large scale simulation codes are not homogeneous programmes any more. They consist of many components, have matured over many years, are written in multiple languages, and often use a software framework to hold things together. One can usually assume that each of these components has been tested in isolation on small machines. However, combining these components and using them on a supercomputer with a computing power that is many orders of magnitude larger can lead to unforeseen things occurring. And since the components are combined by the end user, not by an application developer, it is important to empower the end user to debug and profile the application.

In contrast to many existing free and commercial utilities, the Alpaca tools will not be external to the application, but will be built-in, so that they have direct high-level access to information about the running application, and can interact with the user on a correspondingly high level. We call this the *application level*, as opposed to debugging or profiling on the *code level*, which deals with individual Fortran variables or MPI calls. While it is important to deal with code-level problems to find e.g. uninitialised variables or segmentation faults, it is these days equally important to be able to deal with a complete application as a whole, and understand its behaviour on a global level. Instead of only *observing* a programme, it is also necessary to *query* it about its state, and to *interact* with it to modify this state.

Finally, successful simulations also depend on the reliability of the system, necessitating a mechanism where software can compensate for hardware errors. Alpaca will research improved checkpointing/recovery methods for fault tolerant computing, based on special MPI implementations. This will let the application recover after partial system failures, which are increasingly common in large machines containing thousands or tens of thousands of nodes. By dynamically adapting the simulation to problems in the computing environment, fault tolerant computing can increase throughput and productivity manifold.

3.1. **Application Communities.** The application communities described in table 1 are providing requirements to the Alpaca project:

Numerical Relativity: A large number of relativity codes exist which were designed for Cactus from the beginning, both at LSU and outside: The freely available AEI/LSU BSSN code [18, 17], the LSU GH (Generalized Harmonic) multi-block code [37], and the AEI GH excision code [43] are spacetime evolution codes, `Whisky` [22, 45] is a hydrodynamics module that works in conjunction with any spacetime code. Other numerical relativity groups have their own, independent spacetime codes. This list is not complete.

Numerical relativity continues to contribute major infrastructure improvements to Cactus, such as the addition of mesh refinement [41, 26] or multi-block methods [40]. The adaptive mesh refinement package `PARAMESH` [36, 33] was recently integrated with Cactus.

Astrophysics: The `FLOWER` code at LSU has already been ported to Cactus. This 3D hydrodynamics code is used for investigating secular bar-mode instabilities in rotating neutron stars and mass transfer in binary systems.

Reservoir Studies: A black oil simulator for Cactus has recently been developed [15], with an emphasis on being able to perform rapid simulations using large numbers of processors, to enable new types of reservoir studies. The capabilities of Cactus, such as parameter steering, checkpointing, and streaming file output, will enable new scenarios in this community.

Computational Fluid Dynamics (CFD): LSU is designing and implementing a CFD Toolkit [27] in Cactus. This toolkit will enable future Cactus-based

CFD applications to seamlessly interoperate, for example though the toolkit's common data models. This CFD toolkit will give students access to a collaborative and integrative problem solving environment for state-of-the-art high performance computing platforms.

Coastal and Environmental Modelling: LSU participates in a project to develop within Cactus the capability of modelling coastal circulation and nearshore surface waves in deltaic sedimentary and hydrodynamic environments in an integrated modelling framework. This work will extend the Boussinesq theory for nearshore hydrodynamics to muddy coasts and non-hydrostatic three-dimensional flow regimes with stratifications. Currently, a 2D finite difference model called `FUNWAVE` has been ported to Cactus.

Quantum Gravity: An extensible toolkit to enable the discrete quantum gravity community to perform a variety of high performance computations within the Cactus framework has been developed. The modularity and automatic parallelism provided by Cactus are for the first time opening this field to large scale simulation, and providing new insights into Causal Sets [39, 35] and Loop Quantum Gravity [23].

Alpaca software development will occur as part of the ongoing Cactus development and follow the established procedures, to ensure that the portability and reliability that people have come to expect from Cactus are not compromised. This will include producing documentation and example code, designing regression test cases and performing regular build tests on all major HPC platforms.

3.2. **User Interface.** Whilst the underlying technologies for debugging and profiling are crucial, the key component for effective and wide-scale use of the infrastructure is a well-designed and easy-to-use user interface. Such a user interface must allow remote access to supercomputers and interaction with jobs running on thousands of processors. Current supercomputers are used as batch systems with command line interfaces, and it is often impossible to interact with a running job. Naturally, this makes for very large turn-around times, especially since queue waiting times are often significant. The breadth of information that can be efficiently presented textually is also limited.

Part of the Alpaca project will be research for new ways to let the user interact with a simulation, taking the special circumstances of HPC computing into account, which require particularly remote access, scalability, and robustness. The user interface should be friendly to the science expert, without damaging the efficiency of the application code. The Alpaca tools are going to potentially generate large amounts of profiling data which cannot reasonably be displayed in a simple table. Building on existing domain-specific and generic visualisation tools like `paraprof` (for `TAU` [3]) or `VisIt` [4], Alpaca will research the optimal way to present these data to the user.

To have a clean separation of functionality, the user interface code should interact with the debugging and profiling infrastructure through a standard set of library calls. These should be designed such that they can not only be used by a compiled-in Cactus user-interface component, but also be exposed as network services. This will allow the easy construction of other user interfaces; e.g. a portal, a component within the Eclipse IDE (see below), or even components within visual workflow frameworks such as `Triana` [5] and the Swift virtual data system. Such

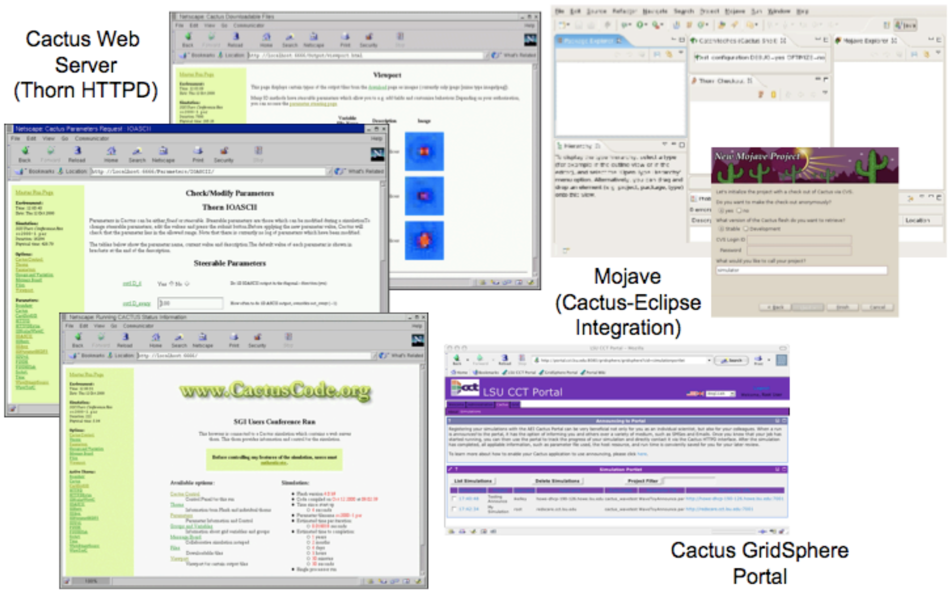

FIGURE 1. Alpaca will build on existing Cactus tools including a web interface which provides information from running simulations and providing steering and visualization interfaces, a GridSphere portal environment which displays information which is automatically collected from Cactus simulations, and Mojave [6] which is a prototype Cactus–Eclipse integration.

a separation into components interacting through defined calls will also facilitate the use of this infrastructure for applications which are not Cactus-based.

The compiled-in user-interface component should make use of the Cactus web server component (figure 1) which is already in wide use to interact with and examine running simulations.[1] This allows the debugging and profiling infrastructure to be always available (compiled in), without hindering the simulation, and be activated instantly when the user requires it, including for production simulations.

3.3. **Debugging.** Standard debuggers provide mechanisms to start a programme or interact with an already-running programme, examine programme data, examine process data (e.g. a stack-trace), watch and trace data and set conditional or unconditional break points, and step through the code on a line-by-line or subroutine basis. Parallel debuggers additionally provide mechanisms to switch between the processes which are being examined and examine data from all processors simultaneously. However, there is a deep gap between the level of abstraction at which a standard debugger like `gdb` or `TotalView` displays information to the user, and the level of the algorithm that is actually implemented in the application.

[1]See http://cactus.cct.lsu.edu:5555/ for a live demonstration.

The Alpaca project plans to bring traditional debugging functionality up to the level of the application. The core infrastructure of Cactus will be enhanced to allow single-stepping through the schedule. At any point in time, the infrastructure knows all application-level variables. This allows the examination or visualisation of such data through both the user interface or via a facility that allows external visualisation tools to connect to a simulation. Such a facility is already available through the HDF5 [7] and web server components in the framework and allows subsets of the data to be chosen.

An important feature not normally available in traditional debuggers, but possible within Alpaca, is the ability for a user to provide a routine which can be used to trigger a break-point. This is much more powerful than the simple arithmetic expression based conditional breakpoints normally available to date. This is possible because, withing Alpaca, a debugger is not an external programme observing an application, but is a component that is part of a framework. It would also be possible to trigger breakpoints based upon performance data as gathered by the profiling infrastructure discussed in the next section.

With traditional debuggers it is not often possible to back-track in time once one has stepped past a point in the code; generally one must restart the application and set a new breakpoint. This is true especially if the programme uses MPI or has open sockets connecting it to other applications. Cactus, however, possesses the ability to checkpoint and restart. This will allow Alpaca to incorporate this facility into the debugging infrastructure, allowing a user to trigger a checkpoint at any time and then to ask Cactus to roll back the simulation to that checkpoint.[2]

Whilst application-level debugging is important, it also remains important to be able to debug at the code level. Alpaca will provide a facility to launch a code-level debugger such as `TotalView`, `DDT`, or `gdb` from within the infrastructure; see section 3.5 below.

3.4. **Profiling.** The key features of a generalised, parallel, application-level profiling tool are the ability to collect any metric of interest, not just those defined by the developer, and easy selection, navigation and presentation of such data. The Alpaca tools should complement the many existing performance measurement libraries, such as e.g. `PAPI`, `mpiP`, or `TAU` [3], by using these libraries to collect data, then interpreting these data together with meta-data about the application that are known to the framework, and finally presenting these data to the user in an interactive manner.

There are many different metrics which can be used to measure performance or are of interest to code optimisers. These range from, at the coarsest level, the wall-clock time of the whole application, to, at the finest level, the number of L1 cache misses of a single machine instruction. In parallel applications there are many additional metrics, such as the time spent waiting for other processes to reach a barrier. Integration libraries providing new metrics are already available within Cactus to some extent, and this can for instance be used to use the `PAPI` performance library on hosts where it is available. The Alpaca tools need to enhance this infrastructure as necessary, including the ability to gather information from job queuing systems and hardware cluster monitors. They need furthermore to

[2]This on-line roll-back is not useful in the case of a critical error leading to the failure of a whole process, e.g. a segmentation fault. For this case it will need to be combined with a fault-tolerance mechanism such as that of FT-MPI and Open MPI (see below).

provide a framework to collect such performance information, to collect algorithmic meta-data (e.g. iteration numbers, numerical accuracy, total mass), and let the user interpret both kinds of data in conjunction with each other. This allows the user to examine the performance on the algorithmic level.

Cactus already places timing caliper points around each routine. Alpaca plans to enhance this infrastructure to distinguish calls from different contexts, utilising the information the flesh has about the application schedule. Additionally it will become possible to set caliper points between any two points in the schedule, e.g. by use of a graphical representation of the workflow; this will be possible at run time, without recompiling the application.

The Alpaca tools also need to provide features to aggregate data from all processors or sets of processors. This latter will enable the user to distinguish between parallel performance features at many levels of the hierarchy of current or envisioned high-end systems – e.g. a single process, processes running on several cores of the same processor, processors on the same SMP board or on the same interconnect, or processes running on different systems connected by a LAN or WAN.

In addition to access to the data though the user-interface for a currently running application, Alpaca also needs to develop a facility to store the profiling data to disk or stream it to a remote site for off-line analysis. This will eventually lead to a body of *performance experience*, i.e., a set of performance results documented by automatically collected meta-data describing the circumstances under which it was gathered.[3] An infrastructure providing a secure long-term storage for such data is currently being developed as part of the AstroGrid-D project [9] at the Albert-Einstein-Institut in Potsdam, Germany.

3.5. **Eclipse Integration.** The Eclipse framework [10] delivers "rich-client applications", i.e., it can be used as intelligent front-end to arbitrary servers. The Parallel Tools Platform (PTP) [11], implemented in Eclipse, provides a portable platform enabling the integration of scalable tools for parallel computing architectures. PTP is creating an environment which simplifies end-user interaction with parallel systems. For example, it is also planned that PTP will provide a parallel (low-level) debugger.

The Alpaca tools should be integrated with the Eclipse framework. This can build on a previous undergraduate project at LSU called Mojave [12]. On one hand, Eclipse is an ideal basis for a rich and portable front-end to the Cactus debugging and profiling infrastructure that the Alpaca project plans to develop. On the other hand, PTP will incorporate many tools that will be important for Cactus users, such as the above-mentioned debugger, which can be enhanced by providing to them meta-data about the application, via an interface between Cactus and Eclipse.

3.6. **Fault Tolerance.** Petascale computing environments will contain tens of thousands of nodes, and single node failures will need to be dealt with in the course of a simulation, without aborting the whole application and forcing it to restart from an old checkpoint after being manually re-queued by the user. To provide fault tolerance for MPI-based simulation codes, Alpaca is considering Open MPI [29, 13],

[3]The `FFTW` library [8] uses a similar concept, although only locally to a machine: It finds certain optimal parameter choices experimentally and stores these in a database which is specific to the machine on which it runs. Later, these data can quickly be used without requiring user action.

a new MPI implementation which incorporates features from Fault Tolerant MPI (FT-MPI) [14] which has additional error-recovery mechanisms. A code using Open MPI (or FT-MPI) has the capability to notice a failure, potentially respawn a failed process, and thus recover and continue a simulation.

The data for the new process could be taken from a checkpoint file on disk, or from an in-memory checkpoint coupled with a set of redundant checksum nodes. The Alpaca project will research the best way to use Open MPI for real world applications to provide file-based or in-memory checkpointing for scientific computing applications. This includes finding ways to dynamically change the number of processors available to the application. Incidentally, being able to change the number of processors can also provide important performance benefits if the communication pattern or memory requirements of a simulation have changed.

4. Conclusions

As the speed and performance of high end computers have increased over the last decade, it has also become increasingly more difficult to program such parallel computers. The difficulty of developing high-performance software is recognised today as one of the most significant challenges in the effective use of large computers. There is a lack of effective software tools that aid in the development of complex, collaborative scientific applications.

Above we have outlined Alpaca, a concept and a project to develop high-level tools allowing to examine and validate the correctness of an application, and aiding in measuring and improving its performance. Alpaca builds on Cactus, an HPC software framework for scientific applications. As with other software frameworks, applications are built from separately developed and tested components. The Alpaca tools are components themselves, so that they are built into the application and can interact with it.

Complementing existing debuggers and profilers, Alpaca's approach works at a much higher level, namely at the level of the components which encode the physical equations and their discretisations, not at the level of individual functions or variables. Specifically, Alpaca addresses *fault tolerance* that will be needed for increasingly large scale machines, *performance monitoring* to make efficient use of such machines, and *interactive debugging* to locate and remedy algorithmic errors. The Alpaca tools will be tied together by a *common user interface* to make them accessible to end users, and their development will involve a *broad range of application areas*.

In order to design, develop, and maintain complex applications, it does not suffice to focus on the correctness and performance of only the main kernels, although this remains nevertheless important. The overall application, which may consist of many components or modules that are assembled by the end user, must be correct and must perform. We assume that Alpaca's integrative and framework-based approach will lead to more reliable and highly efficient applications which can be developed at a shorter time scale.

Acknowledgements

We thank our colleagues and collaborators M. Brodowicz, T. Radke, E. Seidel, and C. White for their contributions to Alpaca. The Alpaca project is funded by the NSF SDCI programme under award number 0721915.

References

[1] GR3D: A multi-purpose 3D code for compuational general relativistic astrophysics, URL http://wugrav.wustl.edu/research/codes/GR3D.

[2] Cactus benchmarking results, URL http://www.cactuscode.org/Benchmarks/.

[3] TAU: Tuning and Analysis Utilities, URL http://www.cs.uoregon.edu/research/tau/home.php.

[4] VisIt Visualization Tool, URL https://wci.llnl.gov/codes/visit/.

[5] Triana, URL http://www.trianacode.org.

[6] Mojave, URL http://www.cactuscode.org/Community/Mojave/.

[7] HDF 5: Hierarchical Data Format Version 5, URL http://hdf.ncsa.uiuc.edu/HDF5/.

[8] FFTW Home Page, URL http://www.fftw.org/.

[9] AstroGrid-D: German Astronomy Community Grid (GACG), URL http://www.gac-grid.org/project-products/Software/InformationService/InformationProducer/CactusRDFProducer.html.

[10] Eclipse: An open development platform, URL http://www.eclipse.org/.

[11] PTP: Parallel Tools Platform, URL http://www.eclipse.org/ptp/.

[12] Mojave web page, URL http://www.cactuscode.org/Community/Mojave/.

[13] Open MPI: Open Source High Performance Computing, URL http://www.open-mpi.org/.

[14] FT-MPI: Fault Tolerant MPI, URL http://icl.cs.utk.edu/ftmpi/.

[15] *UCoMS: Ubiquitous computing and monitoring system*, URL http://www.ucoms.org/.

[16] *SPEC CPU2006 benchmark descriptions*, 2006, URL http://www.spec.org/cpu2006/publications/CPU2006benchmarks.pdf.

[17] Miguel Alcubierre, Bernd Brügmann, Peter Diener, Michael Koppitz, Denis Pollney, Edward Seidel, and Ryoji Takahashi, *Gauge conditions for long-term numerical black hole evolutions without excision*, Phys. Rev. D **67** (2003), 084023, eprint gr-qc/0206072.

[18] Miguel Alcubierre, Bernd Brügmann, Thomas Dramlitsch, José A. Font, Philippos Papadopoulos, Edward Seidel, Nikolaos Stergioulas, and Ryoji Takahashi, *Towards a stable numerical evolution of strongly gravitating systems in general relativity: The conformal treatments*, Phys. Rev. D **62** (2000), 044034, eprint gr-qc/0003071.

[19] Gabrielle Allen, Elena Caraba, Tom Goodale, Yaakoub El Khamra, and Erik Schnetter, *A scientific application benchmark using the Cactus framework*, Tech. report, Center for Computation & Technology, Louisiana State University, 2007, URL http://www.cactuscode.org/Articles/Cactus_Allen07a.pre.pdf.

[20] *Astrophysics Simulation Collaboratory (ASC) home page*, URL http://www.ascportal.org.

[21] David A. Bader, Arthur B. Maccabe, Jason R. Mastaler, John K. McIver III, and Patricia A. Kovatch, *Design and Analysis of the Alliance/University of New Mexico Roadrunner Linux SMP SuperCluster*, 1st IEEE Computer Society International Workshop on Cluster Computing, 1999, p. 9.

[22] Luca Baiotti, Ian Hawke, Pedro J. Montero, Frank Löffler, Luciano Rezzolla, Nikolaos Stergioulas, José A. Font, and Ed Seidel, *Three-dimensional relativistic simulations of rotating neutron star collapse to a Kerr black hole*, Phys. Rev. D **71** (2005), 024035, eprint gr-qc/0403029.

[23] Johannes Brunnemann and David Rideout, *Spectral analysis of the volume operator in loop quantum gravity*, Proceedings of the Eleventh Marcel Grossmann Meeting on General Relativity (Singapore) (H. Kleinert, R. T. Jantzen, and R. Ruffini, eds.), World Scientific, Singapore, 2007, eprint gr-qc/0612147.

[24] Cactus Computational Toolkit home page, URL http://www.cactuscode.org/.

[25] Karen Camarda, Yuan He, and Kenneth A. Bishop, *A parallel chemical reactor simulation using Cactus*, Proceedings of Linux Clusters: The HPC Revolution, NCSA, 2001, URL http://www.cactuscode.org/Articles/Camarda01.doc.

[26] Mesh Refinement with Carpet, URL http://www.carpetcode.org/.

[27] CFD Toolkit for Cactus, URL http://www.cactuscode.org/Community/CFDToolkit/.

[28] Fokke Dijkstra and Aad van der Steen, *Integration of Two Ocean Models*, Special Issue of Concurrency and Computation, Practice & Experience, vol. 18, Wiley, 2005, pp. 193–202.

[29] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall, *Open MPI: Goals, concept, and design of a next generation mpi implementation*, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 2004.

[30] Tom Goodale, Gabrielle Allen, Gerd Lanfermann, Joan Massó, Thomas Radke, Edward Seidel, and John Shalf, *The Cactus framework and toolkit: Design and applications.*, High Performance Computing for Computational Science - VECPAR 2002, 5th International Conference, Porto, Portugal, June 26-28, 2002 (Berlin), Springer, 2003, pp. 197–227.

[31] Jong G. Kim and Hyoung W. Park, *Advanced simulation technique for modeling multiphase fluid flow in porous media*, Computational Science and Its Applications - Iccsa 2004, LNCS 2004, by A. Lagana et. al., 2004, pp. 1–9.

[32] Soon-Heum Ko, Kum Won Cho, Young Duk Song, Young Gyun Kim, Jeong su Na, and Chongam Kim, *Lecture notes in computer science: Advances in grid computing - egc 2005: European grid conference, amsterdam, the netherlands, february 14-16, 2005, revised selected papers*, ch. Development of Cactus Driver for CFD Analyses in the Grid Computing Environment, pp. 771–777, Springer, 2005.

[33] Peter MacNeice, Kevin M. Olson, Clark Mobarry, Rosalinda de Fainchtein, and Charles Packer, *PARAMESH: A parallel adaptive mesh refinement community toolkit*, Comp. Phys. Comm. **126** (2000), no. 3, 330–354.

[34] Seth Major, David Rideout, and Sumati Surya, *Spatial hypersurfaces in causal set cosmology*, Class. Quantum Grav. **23** (2006), 4743–4752, eprint gr-qc/0506133.

[35] ———, *On recovering continuum topology from a causal set*, J. Math. Phys. (accepted) (2007), eprint gr-qc/0604124.

[36] PARAMESH: Parallel Adaptive Mesh Refinement, URL http://www.physics.drexel.edu/~olson/paramesh-doc/Users_manual/amr.html.

[37] Enrique Pazos, Ernst Nils Dorband, Alessandro Nagar, Carlos Palenzuela, Erik Schnetter, and Manuel Tiglio, *How far away is far enough for extracting numerical waveforms, and how much do they depend on the extraction method?*, Class. Quantum Grav. **24** (2007), S341–S368, eprint gr-qc/0612149.

[38] D. Rideout and S. Zohren, *Evidence for an entropy bound from fundamentally discrete gravity*, Class. Quantum Grav. (2006), eprint gr-qc/0606065.

[39] David Rideout and Stefan Zohren, *Evidence for an entropy bound from fundamentally discrete gravity*, Class. Quantum Grav. **23** (2006), 6195–6213, eprint gr-qc/0606065.

[40] Erik Schnetter, Peter Diener, Nils Dorband, and Manuel Tiglio, *A multi-block infrastructure for three-dimensional time-dependent numerical relativity*, Class. Quantum Grav. **23** (2006), S553–S578, eprint gr-qc/0602104.

[41] Erik Schnetter, Scott H. Hawley, and Ian Hawke, *Evolutions in 3D numerical relativity using fixed mesh refinement*, Class. Quantum Grav. **21** (2004), no. 6, 1465–1488, eprint gr-qc/0310042.

[42] Erik Schnetter, Christian D. Ott, Gabrielle Allen, Peter Diener, Tom Goodale, Thomas Radke, Edward Seidel, and John Shalf, *Cactus Framework: Black holes to gamma ray bursts*, Petascale Computing: Algorithms and Applications (David A. Bader, ed.), Chapman & Hall/CRC Computational Science Series, 2007, eprint arXiv:0707.1607 [cs.DC].

[43] Béla Szilágyi, Denis Pollney, Luciano Rezzolla, Jonathan Thornburg, and Jeffrey Winicour, *An explicit harmonic code for black-hole evolution using excision*, Class. Quantum Grav. **24** (2007), S275–S293, eprint gr-qc/0612150.

[44] B. Talbot, S. Zhou, and G. Higgins, *Review of the Cactus framework: Software engineering support of the third round of scientific grand challenge investigations, task 4 report - earth system modeling framework survey*, URL http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20020069014_2002111115.pdf.

[45] Whisky, EU Network GR Hydrodynamics Code, URL http://www.whiskycode.org/.

[46] Burkhard Zink, *A general relativistic evolution code on CUDA architectures*, CCT Technical Report Series (in preparation), 2007.