A Scientific Application Benchmark using the Cactus Framework

Gabrielle Allen, Elena Caraba, Tom Goodale, Yaakoub El Khamra, Erik Schnetter
Center for Computation & Technology
Louisiana State University
Baton Rouge, LA 70803, USA
Email: {gallen,elena,goodale,yye00,schnetter}@cct.lsu.edu

Abstract—**Applications based upon the Cactus Framework have been used for benchmarking for machine procurement and testing, for understanding performance characteristics, and for profiling and optimization studies for many years. Cactus is an open source problem solving environment designed for scientists and engineers with a modular structure which easily enables parallel computation across different architectures and collaborative code development between different groups. It has been developed and used over many years by a large international collaboration of physicists and computational scientists and is now used by a diverse set of applications including numerical relativity, quantum gravity, environmental modeling and computational fluid dynamics.**

This paper describes a Cactus benchmark, named `BSSN_PUGH`, based on the kernel of a numerical relativity application. The significance and profile of `BSSN_PUGH` is described, along with performance and scaling results with various architectures.

The `BSSN_PUGH` benchmark is distributed under an open source consistent with the GNU general public license, is extremely portable and can be used on a wide range of architectures and environments, and if fully documented on the Cactus web pages.

I. Introduction

Computer based simulation is vital for modern scientific and engineering research and development. These simulation codes can use large amounts of memory and disk space and require many days or weeks to run. Developing software to meet these demands is challenging — algorithms must scale to large numbers of processors or cores, efficiency is crucial with simulations running for days and weeks, highly optimized I/O is needed to handle the terabytes of data which must be written to disk, and the software must be well written and highly portable to handle new architectures. Flexibility, stability, portability, and ease-of-use are crucial features.

To reduce the development time for creating simulation codes and encourage code reuse, researchers have created computational frameworks such as the Cactus Framework [1], [2]. Such modular component-based frameworks allow scientists and engineers to develop their own application modules and use them in conjunction with existing modules to solve computational problems. Cactus provides tools ranging from basic computational building blocks to complete toolkits that can be used to solve a range of application problems. Cactus runs on a wide range of hardware ranging from desktop PC's, large supercomputers, to 'grid' environments. The Cactus Framework and core toolkits are distributed with an open source license from the Cactus website [1], are fully documented, and are maintained by active developer and user communities.

This paper describes the Cactus Framework and how scientific application codes developed within Cactus are being used for a range of benchmarking related activities. One specific benchmark, `BSSN_PUGH`, is described along with details about results for the benchmark with different architectures and with various numbers of processors. This paper compliments the `BSSN_PUGH` benchmark webpage [3], which contains complete information about obtaining, compiling, and running the benchmark, along with analysis of current results.

II. Cactus

The Cactus Framework consists of a central part ("flesh") and components ("thorns"). The flesh has minimal computational functionality and serves as a module manager, coordinating the flow of data between the different components to perform specific tasks. The components or "thorns" perform tasks ranging from setting up a computational grid, decomposing the grid for parallel processing, setting up coordinate systems, boundary and initial conditions, communication of data from one processor to another, solving partial differential equations, to input and output and streaming of visualization data. One standard set of thorns is distributed as the Cactus Computational Toolkit to provide basic functionality for computational science [4].

Cactus was originally designed for scientists and engineers to collaboratively develop large-scale, parallel scientific codes which would be run on laptops and workstations (for development) and large supercomputers (for production runs). Cactus provides the basic parallel framework supporting several different codes in the numerical relativity community used for modeling black holes, neutron and boson stars and gravitational

[1]Goodale is also affiliated with Cardiff School of Computer Science, The Parade, Cardiff, CF24 3AA, UK

waves. This has lead to over 150 scientific publications in numerical relativity which have used Cactus and the establishment of a Cactus Einstein Toolkit of shared community thorns. Other fields of science and engineering are also using the Cactus Framework, including Quantum Gravity, Computation Fluid Dynamics, Computational Biology, Coastal Modeling, Applied Mathematics etc, and in some of these areas community toolkits and shared domain specific tools and interfaces are emerging.

Cactus provides a range of advanced development and runtime tools including; an HTTPD thorn that incorporates a web server into the simulation allowing for realtime monitoring and steering through any web interface; a general timer infrastructure for users to easily profile and optimize their codes; visualization readers and writers for scientific visualization packages; and interfaces to Grid Application Toolkits for developing new scientific scenarios taking advantage of distributed computing resources.

Cactus is highly portable. Its build system detects differences in machine architecture and compiler features, using automatic detection where possible and a database of known information where auto-detection is impractical — e.g. which libraries are necessary to link Fortran and C code together. Cactus runs on all variants of the Unix operating system and on the Windows platform Codes written using Cactus have been run on some of the fastest computers in the world, such as the Japanese Earth Simulator and the IBM Bluegene/L.

Cactus has generally been used for calculations based upon explicit finite difference methods. Each simulation routine is called with a block of data — e.g. in a 3-dimensional simulation the routine is passed a cuboid, in a 2-dimensional simulation a rectangle — and integrates the data in this block forward in time. In a single processor simulation the block would consist of all the data for the whole of the simulation domain, however in a multi-processor simulation the domain is decomposed into smaller subdomains and each processor computes the block of data from a subdomain. In a finite difference calculation the main form of communication between these subdomains is on the boundaries, and this is done by *ghost-zone exchange* whereby each subdomain's data-block is enlarged by the nearest boundary data from neighbouring blocks — so called *ghost-zones*; the data from these ghost-zones is then exchanged once per iteration of the simulation.

Parallelisation and over all flow of control is controlled by by a specific component, called the *driver*. There are many drivers available for Cactus; the `BSSN_PUGH` benchmark described here makes use of the *PUGH* driver which has been shown to scale to many thousands of processors [5]. PUGH uses the standard Message Passing Interface (MPI) [6] for the ghost-zone exchange.

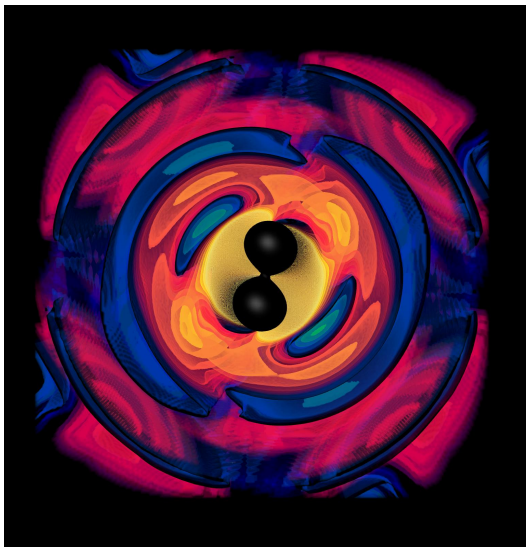

Fig. 1. Visualization of data from a Cactus simulation of two orbiting black holes which shows the radiation emitted as gravitational waves. The benchmark kernel described in this paper is the core computational contribution in the simulation. [Image credits: Werner Benger AEI/LSU/ZIB]

III. Cactus Benchmarking and Profiling

Cactus was designed for application communities solving large scale computational problems using large supercomputers. One such user community are numerical relativists solving Einstein's equations for gravity to investigate the physics of black holes and neutron stars (Fig. 1). Such problems require machines with extreme performance and memory; scientists typically have access to several machines provided by supercomputing centres around the world and are involved in procurement exercises to specify and purchase new machines.

This need has led to a sustained effort using Cactus for optimization and benchmarking studies [7], and in particular Cactus applications have been used for performance measurements to evaluate new and existing compute architectures [8], [9], [10], [11], [12], storage architectures [13], petascale (or "leadership-class") computing platforms [14], [5]. Cactus is used for prototype examples in investigating and understanding general application classification and performance [15], e.g. Cactus illustrates the structured grid "dwarf" in the Berkeley view of the future of high performance computing [16].

Cactus was recently benchmarked on the IBM BlueGene/L at IBM's TJ Research Center and showed excellent scaling, for the benchmark discussed in this paper, to over 32,000 processors (see Fig. 2).

Cactus benchmarks include

- `BSSN_PUGH`: The focus of this paper, this benchmark is an application of a numerical relativity code using finite differencing on a uniform structured grid. It uses the CactusEinstein infrastructure to evolve a vacuum spacetime, using the BSSN [17], [18] for-

mulation of the Einstein equations. It employs finite differencing in space, an explicit time integration method. MPI is used for message passing.

- `BSSN_Carpet`: This benchmark is similar to `BSSN_PUGH` but instead of a uniform structured grid it uses the Carpet driver [19], [20] to provide mesh refinement. MPI is used for message passing.
- `Whisky_Carpet`: This benchmark is relativistic hydrodynamics code with mesh refinement.It is similar to `BSSN_Carpet`, but it solves the relativistic Euler equations in addition to the Einstein equations. This requires more memory and more floating point operations per grid point. The relativistic Euler equations are implemented in the Whisky code [21].
- `IO_HDF5`, `IO_FlexIO`: These measure the speed of writing simulation data to disk using the HDF5 I/O method and IOFlexIO I/O method respectively from Cactus. In the default configuration, each processor writes 352 MBytes to disk.
- `Bench_ADM`: A historical benchmark that has been replaced by `BSSN_PUGH`. This benchmark uses a different numerical formulation of the equations of general relativity (ADM) which is not predominantly used today, although there is much similarity in the two kernels. The `Bench_ADM` benchmark forms part of the SPEC CPU2006 [22] benchmark (under the name CactusADM). The Cactus web pages include many results from `Bench_ADM`.

IV. `BSSN_PUGH` Benchmark

The `BSSN_PUGH` benchmark is based on the kernel of the *Ccatie* numerical relativity code which solves Einstein equations in vacuum, using the BSSN formulation [17], [18]. This formulation of the Einstein equations consists of 25 nonlinear, coupled, hyperbolic, partial differential equations. Although numerical formulations of these equations can involve solving four coupled elliptic constraints, `BSSN_PUGH` uses a "free" evolution where the constraints are not imposed during time evolution, and no elliptic equations are solved. The BSSN equations are discretised with fourth order accurate finite differences[1] on a uniform grid, and integrated using an explicit third order accurate Runge-Kutta time integrator.

A typical unigrid black hole simulation requires grid sizes of approximately 600^3 grid points and 1200 time steps. (These requirements can be reduced by using mesh refinement or similar methods [19]). A realistic black hole simulation also includes additional analysis thorns which are not part of this benchmark, e.g. to locate black hole horizons, compute emitted gravitational waveforms, result output, and periodic checkpoints. These additional analysis modules and IO make a substantial contribution to the total runtime.

[1]This requires two ghost-zones to be communicated from each face as described earlier.

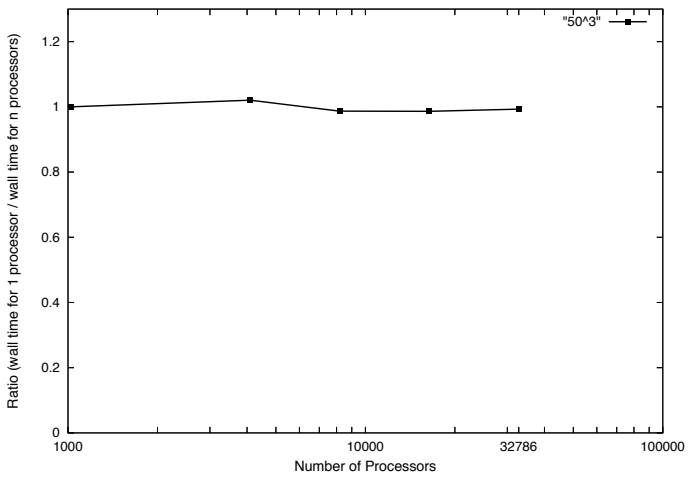

Fig. 2. Scaling efficiency of the `BSSN_PUGH` benchmark on the BlueGene/L architecture. The benchmark size was reduced to $50 \times 50 \times 50$ to fit the available memory of the processors. `BSSN_PUGH` scales well up to 32768 processors without loss of efficiency.

The `BSSN_PUGH` benchmark is distributed in two configurations, with three-dimensional grids of size (i) $80 \times 80 \times 80$ (with 80 time-step iterations) and (ii) $100 \times 100 \times 100$ (with 40 time-step iterations). These configurations use around 400 MB and 800 MB of main memory per CPU, and consume approximately 512 GFlop and 508 GFlop, respectively. On machines where memory is a restriction (for example the BlueGene/L for which results are shown in Fig. 2), it is necessary to decrease the size of the problem, and this can be achieved easily by changing a parameter in the input parameter file.

The Appendix provides basic information on obtaining, configuring, compiling, and running the `BSSN_PUGH` benchmark.

V. Benchmarking Metrics and Issues

The `BSSN_PUGH` benchmark is an application benchmark that characterises single-processor performance and parallel scalability of a machine for an application class typical in the world of high-performace computing. These are two of the most important metrics for the Cactus user community. The other important metric — I/O performance — is not measured by this benchmark.

A. Single-Processor Performance

When porting a code to a new machine or evaluating a new architecture, single processor performance is one of the first investigations. The coarsest, and for procurement purposes the most important, metric is the wall-clock time to complete a given simulation. This gives a realistic comparison of how different machines compare.

Wall clock time is directly related to the number of floating point operations performed and so another metric of the machine speed is the total number of floating

point operations performed per second — Flop/sec. Another interesting metric is the ratio of the Flop/sec of the code and the processor's theoretical peak performance. This provides an indication of whether the code could be restructured to make better use of the architecture-specific details, such as cache layout.

In general, the computation time is time spent performing floating point operations, memory access operations (load-store), and integer operations. Flop/sec is a measure of the first of these, however when optimising code it is essential to examine the other metrics, in particular load-store metrics such as cache hit ratio.

The BSSN_PUGH benchmark only provides wallclock time for the code as a whole and for routines provided by the components. However it is possible to use tools such as IPM [23] or PAPI [24] to determine other metrics.

B. Scalability

Scalability is usually discussed in terms of either weak or strong scaling. Weak scaling involves running in parallel with a constant problem size per processor — as the number of processors increases, the global problem size increases. Strong scaling involves running in parallel with a constant global problem size — as the number of processors increases, the local problem size decreases. With an infinite number of processors, the time to completion of a strong scaling problem will essentially be the communication time between the processors.

Both metrics of scalability are important for characterizing performance. Strong scaling information allows users to predict the cost-time benefits of using a larger number of processors for a given simulation. Weak-scaling information allows the user to decide the largest problem size (and hence highest accuracy simulation) which can be done.

Scaling is normally measured by its "efficiency" which measures how quickly a code would run if it had perfect scaling comapred to how quickly it runs in practice. For strong scaling perfect scaling would give run times inversely proportional to the number of processors, i.e.,

$$E_S = T_1/nT_n \tag{1}$$

where E_S is strong scaling efficiency, T_1 is the time to run on a single processor, n is the number of processors and T_n is the time to run on n processors. For weak scaling the time to run on n processors should be the same as on one processor, giving a weak scaling efficiency, E_W of

$$E_W = T_1/T_n \tag{2}$$

In general efficiency is less than one, although in some rare cases, such as reducing the problem size on a processor such that more efficient use is made of the cache, it is possible to get efficiencies greater than one — "superlinear" speedup.

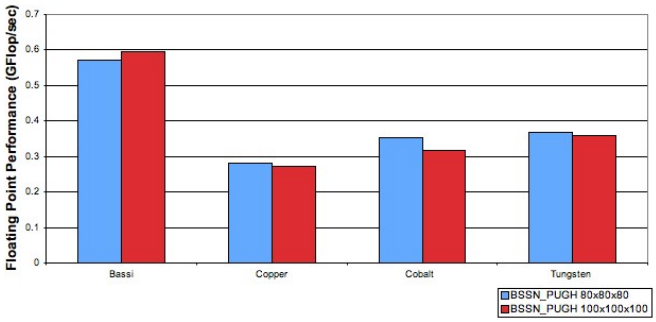

Fig. 3. Absolute single processor floating point performance of the BSSN_PUGH benchmark, measured in GFlop/sec. The graph shows the actual floating point performance that BSSN_PUGH achieves on different architectures.

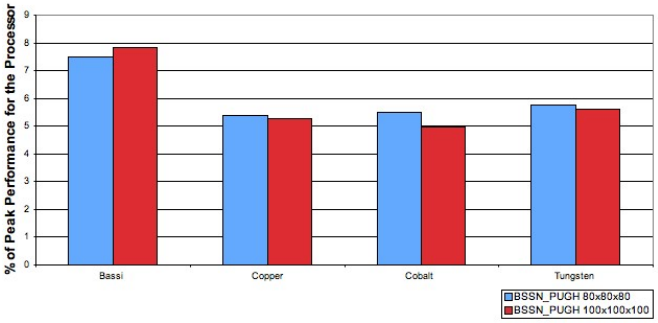

Fig. 4. Relative single processor floating point performance of the BSSN_PUGH benchmark. The graph shows the percentage of the architecture's peak performance that BSSN_PUGH achieves. As all realistic benchmarks, BSSN_PUGH is far from using 100% of the available floating point performance.

In the numerical relativity community scientists are generally interested in running the largest possible problem size, and thus the BSSN_PUGH benchmark is a formulated as weak scaling benchmark. An equivalent strong scaling benchmark is easy to produce by a small change to the parameter file.

VI. Results and Analysis

This section presents the results of running the BSSN_PUGH benchmark on the machines described in Table I. These four machines have been chosen to provide a spread of processors, operating systems and parallel interconnects: Bassi is an IBM p575 Power 5 machines running the AIX operating system; Copper is an IBM pSeries p690 machine with Power 4 chips; Tungsten is an Intel Xeon based cluster running the Linux operating system and using the Myrinet interconnect between the cluster nodes; and Cobalt is an SGI Altix (Linux with patches to enable a large single system image) using Intel Itanium 2 processors and Infiniband. The table also includes the architecture details of the BG/L system at the IBM T.J.Watson research centre which was used for the scaling runs previously shown in Fig. 2.

Machine	Bassi	Cobalt	Copper
Site	NERSC	NCSA	NCSA
Proc	Power 5	Itanium 2	Power 4
Procs/node	8		8
Clockspeed	1.9 GHz	1.6 GHz	1.3 GHz
Peak Perf	7.6 GFlop/sec	6.4 GFlop/sec	5.2 GFlop/sec
Memory	32 GB	2 GB	64 GB
OS	AIX	Linux	AIX
Interconnect	GigE	Infiniband	GigE

Machine	Tungsten	BGW
Site	NCSA	IBM TJ Watson
Proc	Intel Xeon	Power 440
Procs/node	2	2
Clockspeed	3.2 GHz	700 MHz
Peak Perf	6.4 GFlop/sec	2.8 GFlop/sec
Memory	3 GB	512 MB
OS	Linux	Custom
Interconnect	Myrinet	Custom torus

TABLE I
MACHINE CONFIGURATIONS FOR BENCHMARK RESULTS, INCLUDING; PROCESSOR TYPE, CLOCKSPEED, PEAK PERFORMANCE, MEMORY; MACHINE NAME, LOCATION, OPERATING SYSTEM, INTERCONNECT.

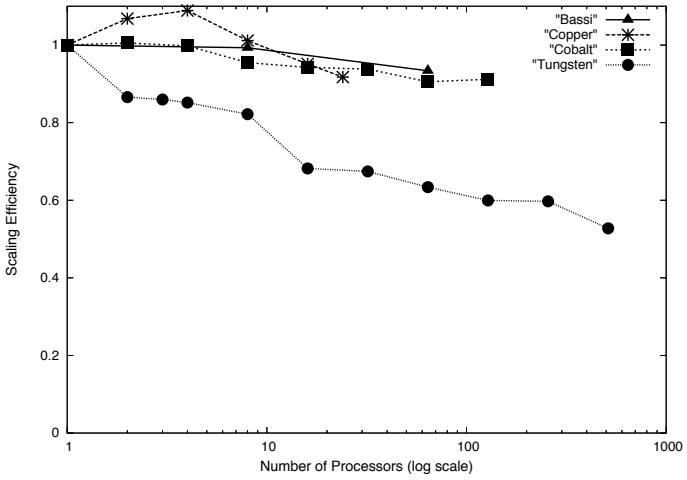

Fig. 5. Scaling efficiency of the BSSN_PUGH benchmark on different architectures for the $80 \times 80 \times 80$ problem size.

A. Single-processor Performance

Fig. 3 and 4 show single-processor floating point performance of BSSN_PUGH on the different architectures. Fig. 3 shows the value in GFlops, which provides a comparison of different architectures, whereas Fig. 4 shows it as a percentage of the theoretical peak performance of the machine, which provides a measure of how efficiently the code is using the features of the processor and interconnect; a lower than normal number for this generally indicates that it may be possible to restructure code to make better use of the processor.

B. Scaling

Fig. 5 and 6 present the scaling efficiencies (as defined in Equation 2) for the benchmark for the $80 \times 80 \times 80$ and $100 \times 100 \times 100$ problem sizes respectively.

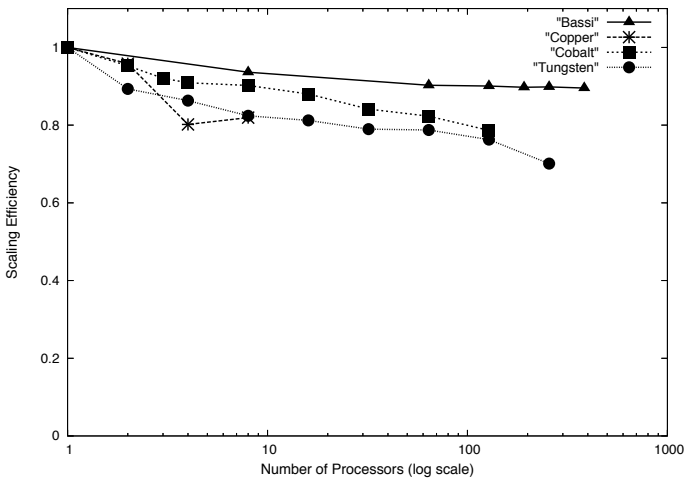

Fig. 6. Scaling efficiency of the BSSN_PUGH benchmark on different architectures for the $100 \times 100 \times 100$ problem size.

The worst scaling by far is seen in the Copper cluster, where there is dramatic fall-off of parallel efficiency and sharp changes when new processor communication pathways come into play due to the internal topology of the machine. The other machines are custom supercomputers and show a fairly slow drop in efficiency for the smaller problem size. Of the four machines presented here it is clear that Bassi — the IBM Power 5 system — is the most suited for running problems of this nature. None of the systems show the same level of scaling as the BGW — the BlueGene/L system at IBM T.J.Watson research centre — however this is somewhat offset by the greater memory per processor of these machines.

VII. CONCLUSION

The paper described the BSSN_PUGH benchmark, which provides performance and scaling efficiency for a real world scientific application in the field of numerical relativity. The benchmark uses the Cactus Framework, a popular parallel framework for a range of applications using high performance computing resources. BSSN_PUGH and other Cactus benchmarks are used extensively in studies of application performance and machine architectures. Performance and scaling results were presented and discussed on a range of architectures.

VIII. ACKNOWLEDGMENTS

We gratefully acknowledge the contributions of Thomas Radke, Ed Seidel, John Shalf and Sasanka Madiraju. This work used the computational resources at NCSA, NERSC, PSC, LSU, LONI and IBM TJ Watson.

APPENDIX

The BSSN_PUGH distribution can be downloaded from the Cactus website [3] where there are complete instructions on compiling and running the benchmark. This Appendix provides a brief overview of the compilation and

running procedure, however users are recommended to read the fuller description on the Cactus web pages.

1) Download the source code tarball from the Cactus web pages [3]. The current version (1.0) was distributed on 7th October 2005 and contains the complete source code required to build the benchmark and the two parameter files needed to run the benchmark: `Bench_BSSN_PUGH_80l.par` and `Bench_BSSN_PUGH_100l.par`.
2) Extract the contents of the tarball, and move to the main Cactus directory.
3) Create a benchmark configuration. It is possible to compile Cactus configurations with different compiler flags and options, for example specifying different sets of thorns, or whether on not the executable will use MPI (that is, whether it will suitable for a single processor or parallel machine). For a standard compilation for a single processor study, the command

   ```
   > gmake BSSN_PUGH_Single-config
   ```

   will create a Cactus configuration using the correct set of thorns called BSSN_PUGH_Single . For configurations using specific compiler options or MPI refer to the Cactus web pages and User Guide [25].
4) Compile the configuration to create an executable `./exe/cactus_BSSN_PUGH_Single`:

   ```
   > gmake BSSN_PUGH_Single
   ```
5) Run the benchmark. The benchmark is available in two sizes (80x80x80 or 100x100x100 problem size). To run for the chosen size use

   ```
   > ./exe/cactus_BSSN_PUGH_Single
             Bench_BSSN_PUGH_80l.par
   ```

   or

   ```
   > ./exe/cactus_BSSN_PUGH_Single
             Bench_BSSN_PUGH_100l.par
   ```
6) Extract the benchmark results from the standard output to screen. The results include the wall time usage and the rusage (resource usage). The time used for benchmarking purposes is the wall clock time from the last line "Total time for simulation."

REFERENCES

[1] Cactus Framework, "http://www.cactuscode.org."

[2] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf, "The Cactus framework and toolkit: Design and applications." in *High Performance Computing for Computational Science - VECPAR 2002, 5th International Conference, Porto, Portugal, June 26-28, 2002*. Berlin: Springer, 2003, pp. 197–227.

[3] "Cactus `pugh_bssn` benchmark." [Online]. Available: http://www.cactuscode.org/Benchmarks/bench_bssn_pugh

[4] G. Allen, T. Goodale, G. Lanfermann, T. Radke, D. Rideout, and J. Thornburg, *Cactus Thorn Guide*, 2005. [Online]. Available: http://www.cct.lsu.edu/~gallen/Reports/Cactus_ThornGuide.pdf

[5] L. Oliker, A. Canning, J. Carter, C. Iancu, M. Lijewski, S. Kamil, J. Shalf, H. Shan, E. Strohmaier, S. Ethier, and T. Goodale, "Scientific Application Performance on Candidate PetaScale Platforms," in *International Parallel and Distributed Processing Symposium (IPDPS)*, Long Beach, Ca., March 24-30 2007.

[6] "The message passing interface (mpi) standard." [Online]. Available: http://www.ncs.anl.gov/mpi/

[7] S. Madiraju, "Performance profiling with cactus benchmarks," Master's thesis, Louisiana State University - Baton Rouge, 2006.

[8] A. S. Bland, J. J. Dongarra, J. B. Drake, T. H. Dunigan, Jr., T. H. D. Jr., G. A. Geist, B. Gorda, W. D. Gropp, R. J. Harrison, R. Kendall, D. Keyes, J. A. Nichols, L. Oliker, H. Simon, R. Stevens, J. B. W. III, P. H. Worley, and T. Zacharia, "Cray X1 Evaluation," Oak Ridge National Laboratory, Oak Ridge, TN, Tech. Rep. ORNL/TM-2003/67, March 2003.

[9] L. Oliker, A. Canning, J. Carter, J. Shalf, D. Skinner, S. Ethier, R. Biswas, J. Djomehri, and R. V. der Wijngaart, "Evaluation of cache-based superscalar and cacheless vector architectures for scientific computations," *Supercomputing 03*, 2003. [Online]. Available: http://crd.lbl.gov/~oliker/papers/SC03_SX6.pdf

[10] D. Bader, A. Maccabe, J. Mastaler, J. M. III, and P. Kovatch, "Design and Analysis of the Alliance / University of New Mexico Roadrunner Linux SMP Super," in *First IEEE Computer Society International Workshop on Cluster Computing (IWCC)*, Melbourne, Australia, December 1999.

[11] D. Bader, "High-Performance Algorithms and Applications for SMP Clusters," in *NASA High Performance Computing and Communications Computational Aerosciences Workshop (CAS 2000)*, NASA Ames Research Center, 15-17 February 2000.

[12] L. Oliker, A. Canning, J. Carter, J. Shalf, and S. Ethier, "Scientific computations on modern parallel vector systems," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2004, p. 10.

[13] M. Pillai and M. Lauria, "RAAC: An Architecture for Scalable, Reliable Storage in Clusters ," in *IEEE Int. Conference on Cluster Computing (Cluster '04)*, San Diego, CA, 20-23 September 2004.

[14] J. Shalf, E.Schnetter, G. Allen, and E. Seidel, "Common computational frameworks as benchmarking platforms," 2005. [Online]. Available: http://www.cactuscode.org/Articles/Common_Frameworks.pdf

[15] L. Oliker, "Large-scale performance analysis using the bips application benchmark suite," 2005.

[16] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, December 18 2006. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[17] M. Alcubierre, B. Bruegmann, T. Dramlitsch, J. A. Font, P. Papadopoulos, E. Seidel, N. Stergioulas, and R. Takahashi, "Towards a stable numerical evolution of strongly gravitating systems in general relativity: The conformal treatments," *Physical Review D*, vol. 62, p. 044034, 2000. [Online]. Available: http://arxiv.org/abs/gr-qc/0003071

[18] M. Alcubierre, B. Bruegmann, P. Diener, M. Koppitz, D. Pollney, E. Seidel, and R. Takahashi, "Gauge conditions for long-term numerical black hole evolutions without excision," *Phys. Rev. D*, vol. 67, p. 084023, 2003. [Online]. Available: http://arxiv.org/abs/gr-qc/0206072

[19] E. Schnetter, S. H. Hawley, and I. Hawke, "Evolutions in 3D numerical relativity using fixed mesh refinement," *Class. Quantum Grav.*, vol. 21, no. 6, pp. 1465–1488, 21 March 2004. [Online]. Available: http://arxiv.org/abs/gr-qc/0310042

[20] "Adaptive mesh refinement with Carpet." [Online]. Available: http://www.carpetcode.org/

[21] "Whisky, gr hydrodynamics code." [Online]. Available: http://www.whiskycode.org/

[22] "Standard performance evaluation corporation," `http://www.spec.org/cpu/`.

[23] "Integrated performance monitoring web page , url = http://www.nersc.gov/nusers/resources/software/tools/ipm.php,."

[24] "The performance api (papi) home page:." [Online]. Available: http://icl.cs.utk.edu/papi

[25] G. Allen, T. Goodale, G. Lanfermann, T. Radke, D. Rideout, and J. Thornburg, *Cactus Users Guide*, 2005. [Online]. Available: http://www.cactuscode.org/Guides/Stable/UsersGuide/UsersGuideStable.pdf