BENCHMARKING PARALLEL I/O PERFORMANCE FOR A LARGE SCALE SCIENTIFIC APPLICATION ON THE TERAGRID

FRANK LÖFFLER[(1)], JIAN TAO[(1)], GABRIELLE ALLEN[(1,2)], AND ERIK SCHNETTER[(1,3)]

ABSTRACT. This paper is a report on experiences in benchmarking I/O performance on leading computational facilities on the NSF TeraGrid network with a large scale scientific application. Instead of focusing only on the raw file I/O bandwidth provided by different machine architectures, the I/O performance and scalability of the computational tools and libraries that are used in current production simulations are tested as a whole, however with focus mostly on bulk transfers. It is seen that the I/O performance of our production code scales very well, but is limited by the I/O system itself at some point. This limitation occurs at a low percentage of the computational size of the machines, which shows that at least for the application used for this paper the I/O system can be an important limiting factor in scaling up to the full size of the machine.

1. INTRODUCTION

The rapid increase of computational resources available for science and engineering provides a great opportunity as well as a great challenge for computational scientists. Highly efficient and scalable scientific applications that can fully leverage the computational power of the supercomputers are seen to be crucial for new scientific discoveries in many fields, for instance Gamma Ray Burst modeling [13] in astrophysics. Developing and optimizing such large scale scientific applications for current resources benefits from well-defined and standardized benchmarks that can provide a basis for measuring and understanding performance. Such benchmarks also importantly help define and describe the application scenario making it easier to engage computer scientists and HPC consultants in further optimization strategies. Such benchmarks for numerical relativity are being defined through the NSF XiRel project [16, 1].

Many existing benchmarks for HPC systems focus on measuring the performance and scaling of only the compute power provided (e.g. [12]), with only some exceptions (e.g. [11]). However, in practice most production codes also generate large amounts of data output that typically is written to a network filesystem. If this is not done efficiently the performance and scaling of the whole simulation suffers.

There are several reasons why large data output is needed. First, analysis and visualization of simulation data typically occurs after the simulation completes, and particularly for 3D-visualization the amount of data which has to be stored for this can already easily reaches hundreds of GiB. This type of output can typically be controlled by parameters and it might be possible to reduce the data size by e.g. down-sampling.

However, there is a second reason why most large simulations needs to write large amounts of data. It is common that simulations take many days, even weeks or months to run, even on large and powerful supercomputers. However, the maximum queue walltime is typically on the order of some days at most. This means that the simulation has to have a way to save all the necessary data (*checkpoint*) and then be able to continue later, potentially on a different machine, by reading in this data (*restart*). This paper focuses only on benchmarking I/O using the checkpoint scenario, since it is well-defined and practically unavoidable for large scale scientific applications.

This paper concentrates on understanding the time spent writing such checkpoint files within the typical environment of a production simulation. This does not necessarily aim for the largest possible files or the most optimized use of provided bandwidth, but focuses on the typical use of the facilities by scientists performing production simulations. The Cactus-Carpet computational infrastructure on which our application code is built in introduced in section 2. Section 3provides details on the Cactus I/O layer. The I/O benchmark specifications and computational resources are described in section 4 and 5 respectively. Finally the benchmark results are provided in section 6 and summarized in 7.

Date: June 02, 2009.

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

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

(3 Department of Physics & Astronomy, Louisiana State University, Baton Rouge LA, USA.

2. CACTUS-CARPET COMPUTATIONAL INFRASTRUCTURE

In the Cactus-Carpet computational infrastructure [9, 7, 15, 14, 8] the simulation domain is discretized using high order finite differences on block-structured grids employing a Berger-Oliger block-structured AMR method [6] which provides both efficiency and flexibility. The time integration schemes used are explicit Runge-Kutta methods. The basic recursive Berger-Oliger AMR algorithm is implemented by the Carpet library. The Cactus framework hides the detailed implementation of Carpet from application developers and separates application development from infrastructure development. Time integration is provided by the Method of Lines time integrator.

Cactus is an open source software framework consisting of a central core, the *flesh*, which connects many modules (*thorns* in Cactus terminology) through an extensible interface. Cactus is highly portable and runs on all variants of the Unix operating system as well as the Windows platform. Carpet acts as a driver layer of the Cactus framework providing adaptive mesh refinement, multi-patch capability, as well as memory management, parallelization, and efficient I/O.

3. CACTUS I/O LAYER

One of the aims of the Cactus computational toolkit is to provide common infrastructure that is needed by scientific simulation codes across different disciplines. This approach emphasizes code reusability, and leads naturally to well constructed interfaces, and well tested and supported software. By providing I/O infrastructure scientists can concentrate more on their science problems and at the same time it creates a de-facto standard which helps to compare or exchange data much more easily. The number of people involved in writing and using such a complex infrastructure is large which makes it necessary to divide it into separate thorns and a clearly specified interface between them.

Cactus itself does not include thorns which handle the low-level I/O itself because the flesh does not have access to information describing the actual data structures used to store the grid variables. However, Cactus does provides an interface between thorns that do provide this functionality and thorns which might want to use it. Because this paper focuses on simulations using the Carpet mesh refinement driver, the thorn which provides the low-level checkpoint routines is provided by the Carpet code.

This paper focuses on one of the typically largest I/O operations: writing checkpoint files. Because these data is usually quite large it is important to choose an appropriate file format. The most used format for this within Cactus is the Hierarchical Data Format, version 5 (HDF5) [10]. It was designed to store and organize large amounts of numerical data, stores these data in binary form and supports compression and parallel reading and writing.

The mesh refinement driver Carpet provides a thorn to read and write HDF5 data, called CarpetIOHDF5. This thorn can handle different ways to write checkpoint files, depending on parameters which are typically specified at startup time. One of these possible choices is to write from each process sequentially into one file in total or to write in parallel to one file per process. Because the second option is usually faster, this is used in production simulations which is why this setting was chosen for our tests.

4. BENCHMARK SPECIFICATIONS

The I/O benchmark was set up with an emphasis on the I/O operations of a typical production simulation. The chosen configuration evolves a stable Tolman-Oppenheimer-Volkoff (TOV) star together with the underlying relativistic spacetime. The focus of this paper lies primarily on the I/O scaling, which is why only one particular code for evolving the spacetime has been chosen, the CCATIE code [5, 4]. Using another code would change the number of variables which are checkpointed and thus, the size of the checkpoint files and the total time to write them. This is not expected to change the overall scaling much, but will be part of future investigations.

The fluctuation of the I/O results is quite high, because unlike for the CPU scaling runs, the benchmarked part of the system is fundamentally shared among all users. The only way to exclude fluctuations from this error source would be to run on the systems exclusively. Even if this would be easily possible, it would not reflect the typical environment scientists have to use. Our approach to minimize this error is to perform more than a single checkpoint in one simulation and to perform more than one such simulation for each core count, choosing the fastest of those simulations for the analysis of the results. For this paper each of three identical simulations for each core count wrote ten checkpoint files.

The chosen benchmark saves 53 three-dimensional variables (double precision) of which 49 have three time levels and 4 have one time level. This adds to a total of 151 three-dimensional arrays, 148 of double

	QueenBee	Ranger	Kraken
number of nodes	668	3,936	8,256
number of cores per node	8	16	8
total number of cores	5,344	62,976	66,048
peak performance (TFlop/s)	50.7	579.3	608.0
disk size (TiB)	192	1,700	2,400
number of OSSs	4	50	48
number of OSTs	16	300	336
default stripe count	1	4	4
expected peak I/O (GiB/s)	≈0.9	≈25	≈30

TABLE 1. This table shows some specifics of the tested clusters as of May 2009. It can be seen very clearly that Ranger and Kraken are roughly comparable in both the number of cores and the specifics of the I/O system, and that QueenBee is smaller by an order of magnitude. OSS and OST are functional units in Lustre file system, where OSS stands for the object storage server, and OST stands for object storage targets. In a given Lustre file system, more OSSs and OSTs give larger I/O bandwidth.

precision floating point type and 3 of integer type. In addition, various one-dimensional arrays and scalar values are saved, together with the complete set of parameters. The three-dimensional arrays are by far the largest part of the saved data. This amounts to an average size of a checkpoint file per core of 205 MiB to 255 MiB. For the largest simulations using 2048 cores this sums up to about 650 GiB per complete checkpoint and about 6.4 TiB in total (for 10 checkpoints).

5. COMPUTATIONAL RESOURCES

The benchmarks were run as specified above on three different clusters which the numerical relativity group at Louisiana State University use for production type simulations: QueenBee (LONI), Ranger (TACC) and Kraken (NICS). QueenBee is about an order of magnitude smaller than the other two clusters, in both the number of CPUs available as also in the size of the I/O system. Especially the different number of file servers (OSSs, OSTs) is clearly visible in the results.

All resources used for this paper utilize Lustre [2] as the parallel scratch filesystem. The performance for parallel operations on Lustre is not only limited by the type of used fileservers and the interconnect between them and the compute nodes, but also by the number of used fileservers.

Lustre distinguishes between two types of fileservers. Metadata servers (MDS) handle all the metadata attached to files or directories, like the name of files, their permission settings or where the actual file data is stored. These data is not handled by the MDS directly, but by another type of server, the object storage server (OSS). These OSSs have a number of physical devices attached which are used to store the data, the so called object storage targets (OST). Lustre can spread files across a number of those OST in a RAID 0 like fashion, which is called *striping*.

When using striping while writing a file, the client will register the (new) file first with the MDS and will then connect to a certain amount of OSSs to write the file in chunks in parallel. The number of OSTs used in this operation is called *stripe count* and the size of the chunks *stripe size*. Using this parallel operations can significantly increase performance. A typical value of a stripe size is 1 MiB and a typical setting for the stripe count is 4. The optimal settings of those parameters depend on the number of I/O processes, the size of the files and the specifics of the user cluster. Some important specifics of each used cluster can be found in table 1. The expected peak I/O bandwidth assumes a benchmark specifically designed to maximize the I/O throughput. The benchmarks done for this paper have not been designed with that goal in mind, but to measure the total time needed for the whole application to save its state. This also includes the time to structure the data in the right way for I/O and compressing the data for smaller files. This is why it is not expected to reach the expected peak I/O bandwidth of the clusters.

6. PARALLEL I/O BENCHMARK RESULTS

The benchmarks have been run from 1 to 2048 cores on Ranger and Kraken and up to 1024 cores on QueenBee. Higher core counts could not be used on Ranger and Kraken because the application itself

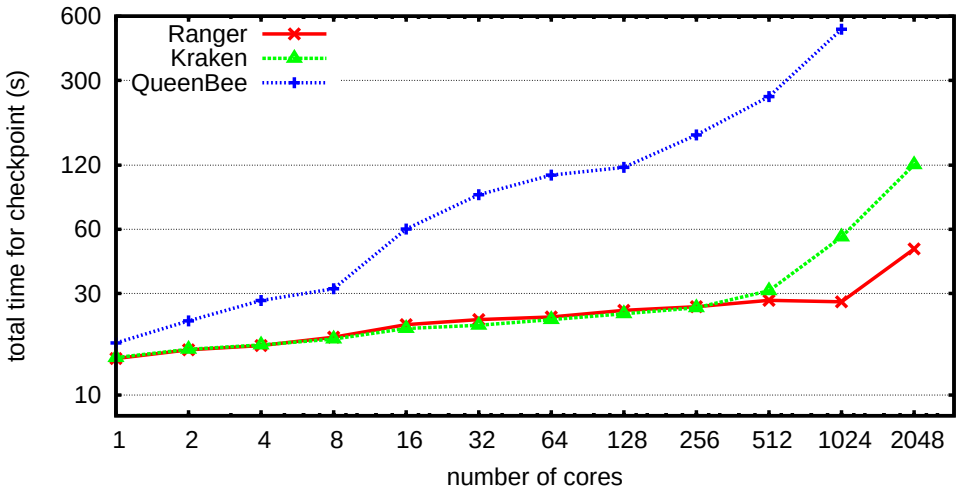

FIGURE 1. Plotted is the time which is needed to write one single checkpoint on all cores of the simulation. This time should stay below 10 minutes which is the case for Ranger and Kraken. On QueenBee, this is reached for a simulation using 1024 cores. The almost constant line for Ranger and Kraken for simulations up to 512 cores shows that the I/O system of these systems scales very well up to that point, while the I/O system of QueenBee does not scale well already beginning with 16 cores.

shows scaling problems at these sizes. On QueenBee the tests ran to 1024 cores only, since it was apparent from both the theoretical limits and the results for 1024 cores that the available I/O bandwidth is already saturated. Further increasing the benchmark runs would not give useful numbers for the scaling anymore but would seriously hinder the work of other users on QueenBee. Each simulation used the default stripe settings (see table 1). It is important that this amount of data can be written to disk in a time very short compared to the maximum wallclock time, which typically is one or two days. However, this is not the only limitation. The larger supercomputer systems grow, the more they are subject to failure. Even if the maximum walltime is set to more than a day it might be interesting to checkpoint much more often to avoid the risk of loosing a large amount of time of a large simulation. Practically this means that writing a checkpoint should not need more than $\approx$ 10 minutes. Reading of these checkpoint files at restart should also take not much longer than that, but testing this will be part of our future work.

Figure 2 plots the time which the different systems needed to write a complete set of checkpoint files for a given number of cores writing in parallel. As can be seen, the time for all three systems is well below 10 minutes, except for QueenBee and very large numbers of cores where is comes close already for 1024 cores. However, it is important to keep in mind that 1024 cores are already a substantial part of the whole QueenBee system, while the same amount of cores is still small for the Ranger and Kraken systems.

Figure 2 shows the total average bandwidth used during the write of the checkpoint files. The total bandwidth scales linear for Kraken and Ranger up to 512 cores and reaches its maximum on Ranger just above 10 TiB/s. The results on QueenBee seem to be worse, but they are just as good given that the whole system is much smaller than the other two and that the application still reaches about 50% of the expected maximum peak I/O bandwidth without optimizing the benchmark towards that goal. Another difference between the results on Ranger and Kraken on one side and QueenBee on the other side is the deviation from a linear scaling on QueenBee above 16 cores. The reason is here that the results were not measured on an empty system. The I/O system is a shared resource on a cluster and without an otherwise empty system the load of the I/O system is close to unpredictable to the common user. 16 to 64 cores on QueenBee can already be considered to be of medium size, whereas the same size on Ranger or Kraken is still negligible compares to the total size of the cluster and their I/O systems.

The results obtained so far indicate that the Cactus-Carpet I/O layer scales well for the size of current production runs (256-2048 cores). However, even for this scale of simulation a large percentage of the available I/O bandwidth is used, leading to potential contention for resources for other applications (e.g. running on

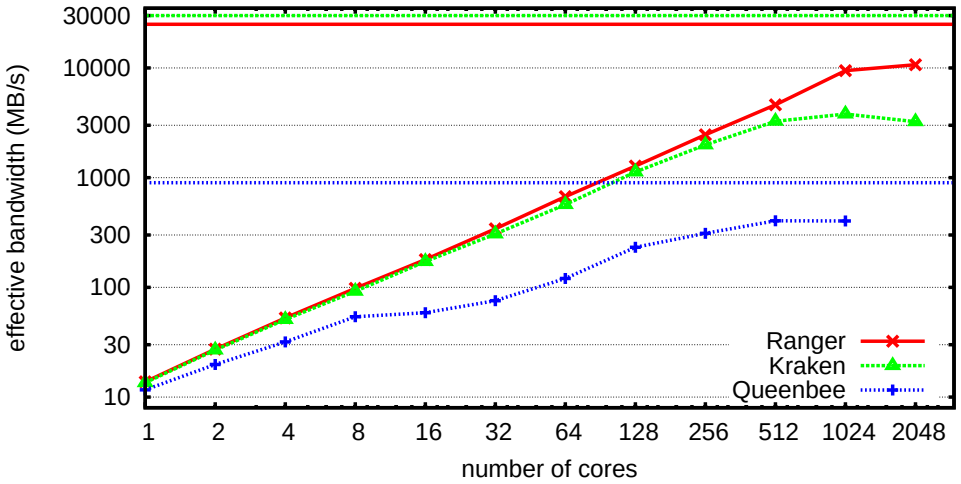

FIGURE 2. This shows the total bandwidth of all cores within one simulation, writing to the I/O system in parallel. This is measured as the total written size of the files divided by the time to finish the complete checkpoint and includes application overhead. For each cluster, the expected peak bandwidth is plotted as horizontal line.

the remaining 58,000 cores of Ranger). The benchmarking team were in close contact with the administrative personnel of all three sites to avoid impact on the work of other users as much as possible, which is of interest to both sides. Improvements which lead to less load of the I/O system ultimately lead to better performance for all users of the system.

Current work in increasing the CPU scaling of our parallel application code to around 16,000 cores for production simulations will require I/O capabilities that are one order larger than achieved in our results. This becomes harder and harder to achieve while retaining an acceptable level of availability. Another possibility would be to find better ways to use a given I/O system on the application level.

For example, potential ways to improve on that level include:

- Use downsampling (e.g. writing only every second data point) or hyperslabbing (e.g. write data only a smaller selected region) for regular output to fit within machine I/O limitations. This strategy would not however be possible for checkpointing where the complete data set is needed.
- Overlap the writing of data from the application with computation. This would require additional work in the Cactus framework to automatically describe the start and end times in the schedule tree where data can be written and will not be changed.
- Since checkpointing is usually performed regularly, both to be able to restart a simulation after the queue time has expired, but also to safe guard against machine or application failure, the absolute time that any checkpoint file is written is usually not important. To optimize the use of contented bandwidth, the I/O system could be monitored with Cactus choosing to perform checkpointing when the I/O subsystem is not loaded.
- Start to write I/O sequentially for simulations which otherwise would overload the I/O system.

7. CONCLUSIONS

I/O performance and scalability of a large scale scientific application has been tested, which is built upon the Cactus-Carpet computational infrastructure, on some of the leading computational facilities of the NSF Teragrid and LONI networks. It has been shown that the I/O component of the tested application scales very well but the overall I/O performance is limited by the I/O system itself. For production simulations on Ranger and Kraken which typically use less than 1024 cores no problems are expected in the immediate future, but our infrastructure will need to be improved for larger simulations. For this it will be important to have continuing direct access to system level experts on the relevant machines.

8. ACKNOWLEDGMENTS

This work is supported by the XiRel project via NSF awards 0701566/0653303, the CyberTools project via NSF award 701491, and the NSF Blue Waters project via NSF award 0725070, The development of performance measurement tools in Carpet is supported by the Alpaca project via NSF award 0721915.

The CCATIE code was developed by a collaboration between Louisiana State University and the Albert Einstein Institute. This work used the *SimFactory* [3] for job management.

This work used the computational resources Kraken at ORNL and Ranger at TACC via the NSF TeraGrid allocation TG-MCA02N014. It also used the computational resources QueenBee at LSU/LONI.

Thanks also has to go to the system level experts from each of the three TeraGrid centers used for this work for their support and advice: Ariel Martinez, Jr. for QueenBee, Yaakoub El Khamra for Ranger and Heo Junseong for Kraken.

REFERENCES

[1] XiRel: Next Generation Infrastructure for Numerical Relativity, URL http://www.cct.lsu.edu/xirel/.
[2] The Lustre file system, URL http://www.lustre.org.
[3] SimFactory, URL http://www.cct.lsu.edu/~eschnett/SimFactory/.
[4] 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.
[5] 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.
[6] Marsha J. Berger and Joseph Oliger, *Adaptive mesh refinement for hyperbolic partial differential equations*, J. Comput. Phys. **53** (1984), 484–512.
[7] Cactus Computational Toolkit home page, URL http://www.cactuscode.org/.
[8] Mesh Refinement with Carpet, URL http://www.carpetcode.org/.
[9] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf, *The Cactus framework and toolkit: Design and applications*, Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science (Berlin), Springer, 2003.
[10] Hierarchical Data Format Version 5 (HDF5) Home Page `http://hdf.ncsa.uiuc.edu/HDF5`.
[11] *Hpc challenge benchmark*, http://icl.cs.utk.edu/hpcc/.
[12] Cleve Moler Jack Dongarra, Jim Bunch and Pete Stewart, *Linpack*, http://www.netlib.org/linpack/index.html.
[13] C. D. Ott, E. Schnetter, G. Allen, E. Seidel, J. Tao, and B. Zink, *A case study for petascale applications in astrophysics: simulating gamma-ray bursts*, MG '08: Proceedings of the 15th ACM Mardi Gras conference (New York, NY, USA), ACM, 2008, pp. 1–9.
[14] 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, URL http://stacks.iop.org/CQG/23/S553.
[15] 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.
[16] Jian Tao, Gabrielle Allen, Ian Hinder, Erik Schnetter, and Yosef Zlochower, *XiRel: Standard Benchmarks for Numerical Relativity Codes Using Cactus and Carpet*, Tech. report, Louisiana State University, Baton Rouge, LA 70803, May 2008.