DOI : 10.1109/IPDPSW.2013.101

Efficient I/O using Dedicated Cores in Large-Scale HPC Simulations

Matthieu Dorier ***(2nd year PhD student)***
ENS Cachan Brittany, IRISA, Rennes, France – matthieu.dorier@irisa.fr
PhD advisors: Gabriel Antoniu (INRIA Rennes Bretagne-Atlantique – gabriel.antoniu@inria.fr),
Luc Bougé (ENS Cachan Brittany – luc.bouge@irisa.fr)

I. Introduction

Post-petascale HPC machines featuring now millions of cores offer new opportunities to simulate physical phenomenon at finer spacial and temporal grain. These simulations usually generate massive amounts of data that are conventionally stored, moved off-site to be later analyzed and visualized in order to produce scientific results.

A typical behavior of HPC simulations consists of alternating between computation phases that solve physical equations, and I/O phases during which all the processes synchronously write the output of the last simulated time step. Yet because of unmatched computation and I/O performance, the scalability of these simulations starts being driven by the performance of their I/O phases. Access contentions at the level of the network and the parallel file system produce substantial performance degradation as well as a high variability, leading to unpredictable performance.

Additionally, it becomes harder and harder to move huge amounts of data from a supercomputer to another, and analysis tasks becoming heavily data-parallel start to suffer from the same I/O bottleneck.

As the community considers designs for exascale systems, there is a growing consensus that revolutionary new I/O and data management approaches will be required. A promising direction consists in leveraging the multicore architecture of next-generation supercomputers to implement better I/O and data processing methods.

Motivated by the aforementioned challenges in the context of the NCSA's BlueWaters supercomputer project [1], this PhD research focuses on Damaris [2], an approach that efficiently leverages a subset of dedicated cores on each multicore SMP node to act as a data management service.

This paper summarizes design issues and results already obtained with Damaris on several platforms, and showing its efficiency in hiding the I/O-related costs as well as improving the performance and scalability of HPC simulations. We also present current work and future directions leveraging Damaris for in-situ data analysis, thus addressing several aspects of this "Big Data" challenge.

II. Background and related work

Two main approaches are traditionally implemented for performing I/O in large-scale HPC simulations. Most high-level I/O libraries such as HDF5 [3] or NetCDF provide both approaches on top of the MPI-I/O layer, which allows simulations to write their results in a format that can be natively read by many analysis tools.

The file-per-process approach consists of having each process write its data in an independent, relatively small file. Whereas this avoids synchronization between processes, it also introduces an important metadata management overhead. Parallel file systems are usually ill-suited for this type of load, in particular when they feature only one metadata server, as in Lustre [4]. Besides, dealing with datasets spread in many small files becomes a major issue when it comes to reload them from a different number of processes for analysis and visualization purposes.

The collective I/O approach introduces communication steps between processes in order to re-organize the data layout and better interact with the file system. Algorithms termed as "two-phases I/O" [5] allow processes to better aggregate data in order to match the file system's parameters or delegate actual writes to a subset of processes. This approach has the advantage of avoiding metadata redundancy and produces bigger, shared files that are easier to read from analysis software. Our experiments showed however that the performance of collective I/O is often worse than that of the file-per-process approach.

Yet both approaches create periodic peak loads in the file system and a high variability in the time taken by each process to complete its operations [6], [7]. This variability can be observed between different processes in the same I/O phase, but also for a given process between different I/O phases, where the observed performance gap between the slowest and the fastest processes can be of several orders of magnitude. Besides, other applications running on the platform and concurrently accessing the parallel file system further increase this variability, leading to degraded overall application performance.

III. The Damaris approach

In order to address the aforementioned I/O challenges, we proposed the Damaris approach [8]. Its main idea consists of dedicating one or a few cores to I/O and data processing tasks in each SMP node. These cores do not run the simulation's code, but handle asynchronous I/O operations on behalf of the other cores, which in turn hides the performance impact of these operations. Such an approach, though easy to express, raises many challenges as well as

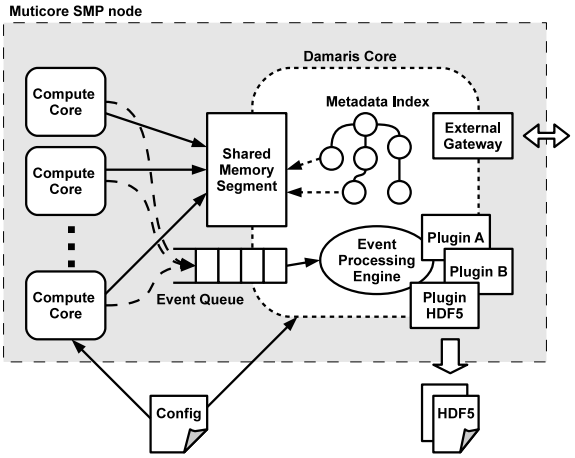

Figure 1. Design of the Damaris approach.

new opportunities that this thesis proposes to explore. This section summarizes the key design choices of the Damaris approach as well as implementation issues.

A. Design principles and challenges

Central to the Damaris approach is the use of shared memory to communicate data from the cores running the simulation to the cores running the data management service. This communication model distinguishes Damaris from other approaches that also use dedicated cores such as [9], relying on message passing or kernel functionalities and involving multiple copies of data. We attempt with Damaris to have a finer control on the memory usage and to avoid unnecessary copies.

The second strength of Damaris consists in a plugin system which makes the design of custom data management services straightforward. Plugins can be written in C or C++ as dynamic libraries, or even in Python scripts, thus leveraging high-level scientific libraries such as SciPy and NumPy. This plugin system may simply be used to forward I/O operations to the HDF5 [3] library, but it can also be (and has been) used to integrate statistical analysis using Python scripts, and visualization tasks using VTK or VisIt [10].

Finally, data management in Damaris is based on a high level description of the data, coming from an external XML file in a way similar to ADIOS [11]. This file contains the description of variables, along with their relationships such as dimension scales, meshes and data layouts. It also contains the configuration of the different plugins that constitute the data management service.

This design led us to investigate the efficient use of shared memory when multiple cores access the same pieces of data, the design of an API that is simple enough for users to build their processing tasks and the right representation of scientific datasets in an XML format.

B. Implementation

Damaris is implemented in C++ and is available as an open-source middleware [2] that can be integrated in existing C, C++ and Fortran simulations that uses MPI. Figure 1 shows the architecture of the middleware.

Its simulation-side API includes functions to directly access the shared memory segment and copy or allocate blocks of data. These blocks are identified by metadata including a *block identifier*, the writer's *process identifier* (usually its MPI rank), and the associated *time step* of the simulation. All data blocks are indexed in a metadata structure that helps searching for particular blocks from data management services.

A shared message queue is used for the simulation processes to send events to he dedicated cores. These events activate the user-provided plugins. The message queue is also used for sending events that inform dedicated cores of the state of the simulation, and help Damaris adapting its behavior.

IV. Evaluation and results

To show the effectiveness of the Damaris approach in addressing the I/O challenge of HPC simulations, we evaluated it with the CM1 atmospheric simulation [12], one of the target applications of the BlueWaters project [1]. CM1 periodically writes either one file per process, or a single shared file in a collective manner using Parallel HDF5. Experiments were carried on several platforms including the French Grid'5000 testbed [13] with 24 cores per nodes, the Kraken Cray XT5 supercomputer [14] with 12 cores per node, and a Power5 cluster featuring 16 cores per node. The results of these experiments are detailed in [8], and summarized below.

A. Making simulations scale

CM1 naturally achieves a very good weak scalability when no I/O is performed. Its overall scalability thus mainly depends on the I/O phase. The two state of the art approaches have a large impact on this scalability. Our experiments performed with up to 9216 processes on Kraken showed that the I/O phase can take up to 800 seconds when collective I/O is used, which represents 70% of the overall run time. The file-per-process approach on the other hand achieves better performance but leads to the creation of a huge amount of files that are simply impossible to post-process.

Damaris manages to achieve a nearly perfect scalability, with a slight impact du to the fact that some cores are not performing computation anymore. This scalability does not depend anymore and the I/O operations, since these operations are now asynchronous and overlap with computation. A speedup of 3.5 was achieved on Kraken compared to collective I/O.

B. Hiding the I/O variability

The computation phases in CM1 also have an extremely predictable run time, therefore the unpredictability in run time only comes from the effect of I/O variability. The unpredictability in the duration of I/O phases can be up to several hundreds of seconds, leading to several hours of unpredictability for a one-month run and forcing scientists to reserve more resources without the guaranty that their simulation will complete in the required time.

The Damaris approach, by moving I/O operations to dedicated cores, manages to perfectly hide this I/O variability, bringing back to the application its original run time predictability. The time to write from the point of view of the simulation is cut down to the time required to write in shared-memory, which is in the order of 0.1 seconds, and does not depend anymore on the scale. Besides, Damaris is able to group the output of multiple processes into bigger files without de communication overhead of a collective I/O approach. Thus the output of dedicated cores can be easily post-processed by analysis tools.

C. Increasing I/O throughput

By aggregating data into bigger files while avoiding communication between processes, Damaris makes a more efficient use of the storage system. While the aggregate throughput was as low as 0.5 GB/s with a collective I/O approach on Kraken, and less than 1.7 GB/s with a file-per-process approach, Damaris was able to achieve up to 10 GB/s.

D. Saving time

Since Damaris uses dedicated cores for I/O and achieves a very high throughput, these cores remain idle most of the time. We measured that this idle time ranges from 92% to 99% on Kraken with the CM1 application. Thus, not only does Damaris improve performance, throughput, scalability and hides the variability, it also leaves room for integrated data processing tasks.

In our previous work we used this spare time to add data compression in files, and achieved a 600% compression ratio without any overhead on the simulation. We also implemented a better I/O scheduling schema to further increase the throughput, achieving up to 12.7 GB/s of aggregate throughput on Kraken.

Our current work plan is to investigate practical uses of this spare time for in-situ data analysis and visualization, a promising direction that we present in the following section.

V. Development directions

As mentioned in the introduction, improving I/O at the simulation level partially addresses the "Big Data" challenge raised by post-petascale machines. Indeed the huge amounts of data generated still have to be read and processed in order to produce a scientific value. One promising trend consists in bypassing the file system and producing results "in-situ", i.e. directly from the simulation as it is running. This technique raises new challenges [15], [16], as it requires a tight coupling between simulations and analysis codes.

A. In-situ analysis challenges

Despite the limitations of the traditional offline approach to data management, it has been often pointed out that scientists are seldom accepting in-situ analysis approaches [17], [15]. This technique indeed often involves changing the code of the simulation and diving into the programming interface of analysis tools. Besides, the scalability and variability of analysis algorithms may have a larger impact that simple I/O. We here summarize four main challenges that have to be addressed when designing an in-situ analysis framework.

1) The programming interface should have a low impact on the code of the simulation. Besides, this interface should be easy enough to be understood and used without the assistance of a specialist.
2) The framework should be adaptable to different analysis and visualization tasks, and be generic enough to be plugged into any simulation.
3) It should have a low impact on the simulation's run time and offer predictable performance bounds as well as a good scalability.
4) Finally it should optimize resources utilization such as local memory and CPU resources, and also leverage GPUs when available.

Current analysis and visualization software offering in-situ capabilities such as VisIt or ParaView [18] often do not meet these requirements. They involve many code changes and perform synchronously by periodically stoping the application in order to compute images. Besides, their underlying algorithms still lake a good scalability.

B. Providing in-situ capabilities to Damaris

Having proven the effectiveness of the Damaris approach in improving I/O performance, this section presents how it can help addressing the challenges of in-situ analysis and visualization.

1) Through its plugin system and its simple API using external description files, Damaris can be easily integrated in existing applications to provide custom data analysis operations.
2) We have embedded the VisIt visualization software in Damaris and leveraged the high level description of data structures in the XML files to seamlessly connect any simulation to this visualization backend.
3) By using dedicated cores, all analysis and visualization operations run in parallel with the simulation without impacting it.
4) Finally the use shared memory in Damaris allows visualization and analysis tasks to work directly on the in-memory data. Besides, the presence of GPU in the nodes of more and more recent platforms such as BlueWaters can be leveraged when the simulation itself does not use them.

C. Preliminary evaluation

We evaluated the use of Damaris for in-situ visualization in several applications: in addition to the CM1 atmospheric model already presented, we used the Nek5000 CFD solver [19]. This experimental campaign aims at showing the effectiveness of Damaris around two aspects.

1) Performance impact: While the use a software like VisIt or ParaView requires the simulation to periodically stop in order to perform analysis tasks, Damaris hides these tasks in dedicated cores and has no performance impact on the simulation. Experimentally, we managed to perform runs of Nek5000 at a full cluster scale (800 cores) of Grid5000 with

Damaris while using VisIt in a synchronous manner did not scale that far.

A challenging problem arises when the analysis tasks take more than the duration of a simulation's time step to complete. In this case it may happen that the shared memory becomes full and blocks the simulation. Discussions with visualization specialists led us to the choice of accepting potential loss of data rather than blocking the simulation. We thus implemented in Damaris a way to automatically skip some iterations of data in order to keep up with the simulation's output rate. More elaborate techniques that will select portions of data carrying important scientific values are now being considered.

2) Usability: We compared the use of the Damaris interface to the code instrumentation required to perform in-situ visualization with VisIt. In order to do so, we took all the examples provided with the VisIt source code and rewrote them using Damaris. All these examples require more than a hundred lines of code with the VisIt API. Damaris only requires one line per data object that has to be shared with dedicated cores, along with some external XML descriptions, ending up with less than 10 lines of code changes in every examples.

Thus, Damaris proves to be very easy to use and a good candidate for the transparent integration of custom analysis and visualization tasks in existing HPC simulations.

VI. Conclusion and future work

As the number of cores in next-generation supercomputers increases, the huge amounts of data produced by HPC simulations running at large scale becomes problematic: it becomes harder and harder to efficiently write such amounts of data in an efficient manner and without an important impact on the simulation. Reading back this data offline for analysis and visualization is becoming intractable as well.

We summarized here our past and present research around these "Big Data" challenges. This PhD research started with the Damaris approach, which proposes to dedicate cores in multicore SMP nodes to I/O, first, and then more generally data management.

The Damaris middleware provides a good basis for the development and evaluation of various data management approaches that use dedicated cores. Its ease of integration in existing applications, coupled with its plugin system and an efficient use of shared memory so far allowed us to (1) provide efficient jitter-free I/O and (2) support in-situ asynchronous visualization with the VisIt software.

Our future work will focus on two directions: we plan to implement "smart" behaviors in Damaris, by asynchronously tracking and processing scientifically relevant subsets of data as the data is being generated by the simulation. We also plan to use Damaris to improve further the efficiency of asynchronous I/O through more elaborate scheduling strategies.

References

[1] NCSA, "BlueWaters project, http://www.ncsa.illinois.edu/BlueWaters/."

[2] KerData, IRISA, INRIA Rennes, "Damaris, http://damaris.gforge.inria.fr/."

[3] "Hierarchical Data Format HDF5, http://www.hdfgroup.org/HDF5/."

[4] S. Donovan, G. Huizenga, A. J. Hutton, C. C. Ross, M. K. Petersen, and P. Schwan, "Lustre: Building a file system for 1000-node clusters," Citeseer, 2003.

[5] R. Thakur, W. Gropp, and E. Lusk, "Data Sieving and Collective I/O in ROMIO," *Symposium on the Frontiers of Massively Parallel Processing*, p. 182, 1999.

[6] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf, "Managing Variability in the IO Performance of Petascale Storage Systems," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–12.

[7] M. Dorier, G. Antoniu, F. Cappello, M. Snir, and L. Orf, "Damaris: Leveraging Multicore Parallelism to Mask I/O Jitter," Rapport de recherche RR-7706, Nov 2011. [Online]. Available: http://hal.inria.fr/inria-00614597

[8] ——, "Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O," in *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, sept. 2012, pp. 155 –163.

[9] M. Li, S. Vazhkudai, A. Butt, F. Meng, X. Ma, Y. Kim, C. Engelmann, and G. Shipman, "Functional partitioning to optimize end-to-end performance on many-core architectures," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–12.

[10] B. Whitlock, J. M. Favre, and J. S. Meredith, "Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System," in *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*. Eurographics Association, 2011.

[11] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS)," in *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, ser. CLADE '08, 2008, pp. 15–24.

[12] G. H. Bryan and J. M. Fritsch, "A Benchmark Simulation for Moist Nonhydrostatic Numerical Models," *Monthly Weather Review*, vol. 130, no. 12, pp. 2917–2928, 2002.

[13] INRIA, "Aladdin grid'5000 http://www.grid5000.fr."

[14] NICS, "Kraken Cray XT5, http://www.nics.tennessee.edu/computing-resources/kraken."

[15] K.-L. Ma, "In Situ Visualization at Extreme Scale: Challenges and Opportunities," *Computer Graphics and Applications, IEEE*, vol. 29, no. 6, pp. 14 –19, nov.-dec. 2009.

[16] E. Bethel and H. Childs, *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. Chapman & Hall, 2012, vol. 16.

[17] K.-L. Ma, C. Wang, H. Yu, and A. Tikhonova, "In-situ processing and visualization for ultrascale simulations," *Journal of Physics: Conference Series*, vol. 78, no. 1.

[18] N. Fabian, K. Moreland, D. Thompson, A. Bauer, P. Marion, B. Geveci, M. Rasquin, and K. Jansen, "The ParaView Coprocessing Library: A Scalable, General Purpose In Situ Visualization Library," in *LDAV, IEEE Symposium on Large-Scale Data Analysis and Visualization*, 2011.

[19] J. W. L. Paul F. Fischer and S. G. Kerkemeier, "nek5000 Web page," 2008, http://nek5000.mcs.anl.gov.