Damaris: Efficiently Leveraging I/O Cores for Scalable Post-Petascale HPC Simulations

Research overview for consideration to ACM SRC Grand Finals 2012
An extensive study including the results presented here is available as a research report at
`http://hal.inria.fr/inria-00614597`

Matthieu Dorier
ENS Cachan Brittany, IRISA, Rennes, France
`matthieu.dorier@irisa.fr`

1 Problem and motivation

Many science domains, such as Earth science, life science, physics, and chemistry, rely largely on computationally intensive simulations. Such simulations generate tremendous quantities of information. Conventional practice consists in storing data on disk, moving it off-site, reading it into a workflow, and analyzing it. This becomes increasingly harder because of the large data volumes generated at fast rates, in contrast to a slower increase in the performance of storage systems. Rapidly storing this data, protecting it from loss, and analyzing it to understand the results are significant challenges. Typically, I/O is periodically and concurrently performed by all processes, which leads to I/O bursts. On current petascale systems, resource contention and substantial variability of I/O performance significantly impact both the overall application performance and its predictability. As the community considers designs for exascale systems, there is a growing consensus in the international community that revolutionary new approaches are needed in computational science storage.

Motivated by these challenges in the context of the NCSA's Blue Waters supercomputer project [16], we propose Damaris, an approach that efficiently leverages a subset of cores on each multicore SMP node to act as a data management service. Damaris has been implemented as an open-source middleware. As opposed to other "space-partitioning" approaches, Damaris makes an *efficient use of intra-node shared memory to avoid costly copies or movements of data* and provides a plugin system to bridge simulations with any I/O or data analysis library. It efficiently removes I/O related costs and is able to *gather, compress and store data in an overhead-free, asynchronous manner* for subsequent offline analysis.

We evaluate our approach on up to 9216 cores on the Kraken Cray XT5 supercomputer [17], with the CM1 atmospheric model, one of the target HPC applications for the Blue Waters project. By overlapping I/O with computation and by gathering data into large files while avoiding synchronization between cores, our solution brings several benefits:

1. It fully hides I/O variability and all I/O-related costs, which makes simulation performance predictable.
2. It increases the sustained write throughput by a factor of 15 compared to traditional approaches.
3. It allows almost perfect scalability of the simulation where other I/O approaches fail to scale.
4. It enables a 600% compression ratio without any additional overhead, leading to a major reduction of storage requirements.

2 Background and related work

2.1 Stat-of-the-art I/O techniques

A typical behavior in large-scale simulations consists of alternating computation phases and write phases. As a rule of thumb, it is commonly accepted that a simulation spends at most 5% of its run time in I/O phases. Two main approaches are typically used for performing I/O in HPC simulations:

The *file-per-process* approach This approach consists of having each process write in a separate, relatively small file. Whereas this avoids synchronization between processes, parallel file systems are not well suited for this type of load when scaling to hundreds of thousands of files: special optimizations are then necessary [4]. File systems using a single metadata server, such as Lustre [9], suffer from a bottleneck: simultaneous creations of so many files are serialized, which leads to immense I/O variability. Moreover, reading such a huge number of files for post-processing and visualization becomes intractable.

Collective I/O All processes synchronize together to open a shared file, and each process writes particular regions of this file. This approach requires a tight coupling between MPI (the Message Passing Interface) and the underlying file system [18]. Algorithms termed as "two-phase I/O" [8, 19, 7] enable efficient collective I/O implementations by aggregating requests and by adapting the write pattern to the file layout across multiple data servers [6]. Collective I/O avoids metadata contention, but imposes additional process synchronization, leading to potential loss of efficiency. Our own experiments with a real application (see Section 4) have shown that Collective I/O can consume 45% of the overall run time at a small 576 cores scale.

The aforementioned approaches create periodic peak loads in the file system and suffer from contention at several levels. Such a contention can happen at the level of each multicore SMP node, as concurrent I/O requires all cores to access remote resources (networks, I/O servers) at the same time. Optimizations in Collective I/O implementations are provided to avoid this first level of contention. "Two-phases" I/O algorithms add a communication step before a subset of processes actually perform the I/O. Whereas this allows to aggregate data and issue bigger writes, the underlying communications impact there scalability. Additionally the synchronous nature of these optimizations forces many processes to remain idle while other are performing I/O, leading to a waste of computation time.

2.2 Coprocessing and dedicated cores

A second problem appears when it comes to read back, analyze and visualize the large amount of produced data. New approaches try to tightly integrate post-processing phases within the I/O path of the application, in order to reduce the size of the data or to produce immediately valuable results.

To solve this problem, a first solution consists in reserving additional computation nodes to better handle the flow of data. In PreDatA [21] for instance, a staging area is deployed on a set of node that process the data prior to effective, asynchronous I/O. Forwarding nodes [2] is another solution using dedicated nodes to buffer and schedule the writes before reaching the file system. This kind of approach is limited by the low amount of memory on dedicated nodes, and the intensive use of network which may interfere with the application.

Another solution consists in avoiding I/O by tightly integrating synchronous post-processing capabilities within simulations. Visualization software such as VisIt [20] or ParaView [12] provide libraries to perform *in-situ* visualization in a time-partitioning manner, i.e. the simulation periodically stops to produce images. While this potentially decreases the time-to-insights from running simulations, the obvious downside is the synchronous nature, which degrades the application's performance and makes it unpredictable.

Both solutions are orthogonal to our approach. Coupling these approaches with our solution based on dedicated cores could bring them what they are missing: more resources and less network traffic to staging areas, asynchrony to in-situ analysis.

A few approaches using dedicated cores have been already proposed to offer asynchronous data-processing and I/O capabilities. Yet, these approaches rely on transport devices such as the network, FUSE interfaces [13] or RDMA [15], which involves keeping multiple copies of data in a context where wasting local memory cannot be afforded. Our approach leverages shared memory and a zero-copy strategy to more efficiently use resources on SMP nodes. Besides, we demonstrate the efficiency of our approach at much larger scale than demonstrated by these previous efforts.

3 The Damaris framework: approach and uniqueness

As the first level of I/O contention occurs when several cores in a single SMP node try to access the same network interface, it becomes natural to work at the node level.

On each SMP node, we propose to delegate I/O operations to a subset of cores (possibly a single one): computation cores write data to the node's shared memory, then the I/O cores asynchronously perform writes for the whole data while the computation cores go on with the next iteration. The I/O cores are dedicated to I/O (i.e. do not run the simulation code) in order to overlap I/O with computation and avoid contention for accesses to the file system. We call this approach Damaris (Dedicated Adaptable Middleware for Application Resources Inline Steering): this section introduces its design, implementation and API.

3.1 Design principles

While most attempts to improve I/O performance rely on a *write* abstraction that is supposed to copy, process and ship data to the right persistent device, our approach is different. We propose that each core running the simulation directly writes its data in a pre-allocated shared-memory buffer. This buffer being accessible by dedicated cores, these cores have immediate access to the data and can read it without the need for extra copies, communication or synchronization. The write-once nature of simulation's data helps us defining a simple API that informs dedicated cores of the moment when data is ready

to be processed.

The second strength of Damaris consists in a plugin system that lets developers provide their own post-processing tasks. 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. In the experiments presented in this paper, we used this plugin system to design a persistency layer that stores data using the HDF5 format [5].

3.2 Implementation and API

Damaris is implemented in C++ and can be interfaced with any MPI application written in C, C++ or Fortran. In a way directly inspired by ADIOS [14] and also present in EPSN [11], it relies on an external XML description of data. This avoids passing many arguments to function calls, and, more importantly, lets dedicated cores manage their buffer in an efficient way. This is implemented through three functions which indicate *when* a particular variable can be consistently read or written by asynchronous processing tasks. Figure 1 visually summarizes these functions.

- `dc_alloc("variable",iteration)` replaces *malloc*. It retrieves a pointer from shared memory to hold the *variable* for a given iteration. Only the simulation is aware of the data, Damaris cannot access it.
- `dc_commit("variable",iteration)` is called when the current buffer associated with the *variable* won't be written anymore by the simulation. A notification is sent to Damaris through a shared queue. Both the simulation and Damaris have read access to the data.
- `dc_clear("variable",iteration)` replaces *free*. Instead of freeing the data, it notifies Damaris that it won't be accessed anymore by the simulation. The dedicated core can safely perform transformations, remove them, or write them persistently.

4 Experimental results

In this section we evaluate Damaris on the Kraken Cray XT5 supercomputer (featuring 12 cores per SMP node) on up to 9216 cores, with the CM1 [3] atmospheric simulation, one of the target Blue Waters applications. CM1 originally uses the two state-of-the-art I/O approaches using the HDF5 and Parallel-HDF5 formats. We use a weak scaling configuration, i.e. the problem size per node is fixed, the total problem size increases with the number of cores used. Each process outputs about 28 MB.

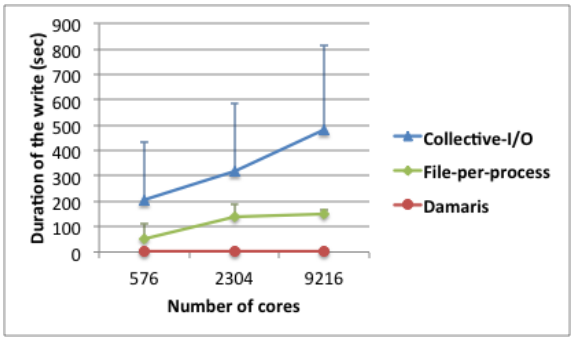


Figure 2: Duration of a write phase on Kraken (average and maximum). Damaris completely removes the I/O cost while f-p-p and collective I/O have a big impact on the run time predictability.

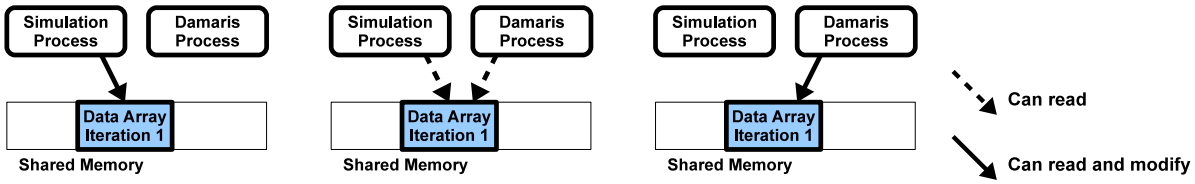
4.1 Damaris removes I/O costs

Figure 2 shows the average and maximum duration of an I/O phase on Kraken from the point of view of the simulation. This time is extremely high and variable with Collective-I/O, achieving up to 800 sec on 9216 processes. The average of 481 sec represents about 70% of the overall simulation's run time. In other words, as we plan a 30-days run on the future Blue Waters system, the simulation would spend 21 days writing and only 9 days computing, which is simply unacceptable. Additionally, the write time went up to 1600 sec with Collective-I/O when setting the stripe size to 32 MB instead of 1 MB in Lustre, exemplifying the fact that bad choices of file system's configuration can lead to extremely poor I/O performance. Unexpectedly, the file-per-process approach seemed to lead to a lower variability, especially at large process count. Yet the problem arising with this approach is the intractability of subsequent file manipulations. Moving these millions of files to an analysis cluster or simply deleting them takes up to an entire day, putting an extreme pressure on the file system and the frontend.

Using Damaris, one core out of 12 on each SMP node is dedicated to I/O, thus potentially reducing the computation power for the benefit of I/O efficiency. As a means to reduce I/O costs and variability, this approach is clearly effective: the time to write from the point of view of the simulation is cut down to the time required to notify the dedicated cores and re-allocate new variables in shared-memory, which represents about 0.2 sec and does not depend anymore on the number of processes. The variability is in order of 0.1 sec (too small to be represented here).

4.2 Damaris scales well

CM1 exhibits very good weak scalability and very stable performance when no I/O is performed. Thus as we increase the number of cores, the scalability



(a) After dc_alloc (b) After dc_commit (c) After dc_clear

Figure 1: Semantics of the three functions: **(a)** at iteration 1, an array is allocated through `dc_alloc`, the simulation holds it, **(b)** eventually (e.g. before starting the next iteration), a call to `dc_commit` notifies the dedicated core of the location of the data. The buffer can be read by both processes, they agree not to write in it. Finally **(c)** a call to `dc_clear` at (e.g. iteration 3) indicates that the simulation does not need the old buffer anymore, Damaris can modify it or move it to a persistent storage.

becomes mainly driven by the I/O phases. To measure the scalability of an approach, we define the scalability factor as $S = N * \frac{C}{T_N}$ where N is the number of cores considered. C is the computation time on one core only for 50 iterations without I/O. T_N is the time to perform 50 iterations plus one I/O phase (which is a typical setting for CM1) on N cores. A perfect scalability factor on N cores should be N. The scalability factor on Kraken for the three approaches is given in Figure 3 (a). Figure 3 (b) provides the associated application run time for 50 iterations plus one write phase. As we can see, thanks to hiding the I/O costs and using shared memory, Damaris shows a nearly perfect scalability where other approaches fail to scale.

4.3 Damaris improves I/O throughput

Figure 3 (c) shows the effective throughput as seen by the writers in the different approaches. Damaris achieves an aggregate throughput about 6 times higher than the file-per-process approach, and 15 times higher than Collective I/O. Note that in the case of Damaris, this throughput is only seen by the dedicated cores. Figure 3 (d) presents the time these cores spare for other usage. As the amount of data on each node is the same whatever the scale (weak scaling configuration), the only explanation for the dedicated cores to take more time at larger process count is the access contention for the network and the file system. Scheduling techniques, not presented here for space constraints, have been provided to further reduce this contention by leveraging the spare time. They are presented in [10].

In conclusion, reducing the number of writers while gathering data into bigger files also has an impact on the throughput that the simulation can achieve.

Similar results have been achieved on other platforms: the French Grid'5000 with PVFS on 672 cores, JaguarPF at ORNL with Lustre on 16K cores, and a Power5 cluster on 1024 cores with GPFS. On all these platforms, state-of-the-art approaches have shown their limits, whereas Damaris is able to perfectly hide all I/O costs, improve applications scalability, and substantially increases throughput, thus making a more efficient use of the file system.

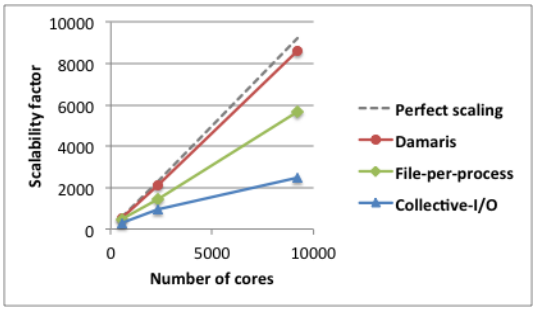
4.4 Damaris saves time for post-processing

We noticed that Damaris can fully overlap writes with computation and still remain idle 75% to 99% of time. Consequently, without impacting the application execution time, the I/O cores could be leveraged for data post-processing or in-situ data analysis, one of our current research directions.

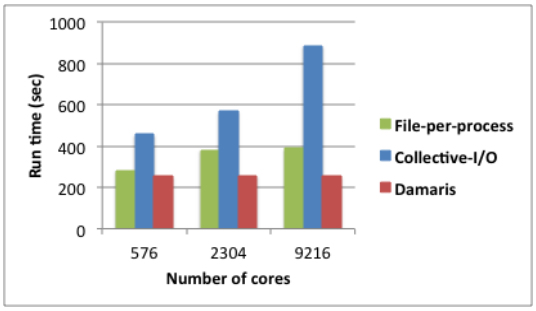
Using gzip compression and reducing the precision of the floating point values to 16 bits can drastically reduce the size of the output data, and thus the storage requirement. This compression comes at the price of an increased write time and variability when using a file-per-process approach, and simply cannot be done using Collective-I/O. In contrast, the spare time in the Damaris approach can be leveraged for this purpose. In addition, by aggregating data into bigger files, we reduce the total data entropy, thus increasing the achievable compression ratio. Enabling compression on dedicated cores allowed us to achieve a 600% compression ratio without any additional cost for the simulation. Table 1 provides the average write time with and without compression from the point of view of the dedicated cores, exemplifying the tradeoff between reducing the size of the data or the time to write it.

Without compression	6.3 sec
Using compression	13.6 sec

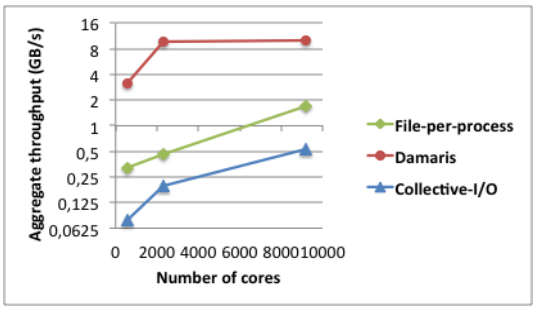
Table 1: Average write duration in dedicated cores when using different optimizations on 2304 cores.



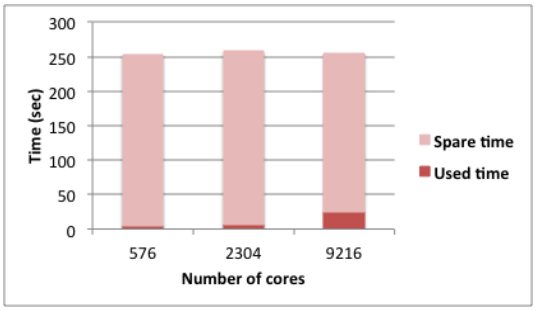
(a) Scalability factor $S = N\frac{C_1}{T_N}$



(b) Overall run time (50 iterations + 1 output)



(c) Effective I/O throughput



(d) Average spare time in dedicated cores

Figure 3: Results obtained on the Kraken Cray XT5 supercomputer with the CM1 tornado simulation, using the three approaches: File-per-process, Collective I/O and Damaris.

5 Conclusion

As the performance of I/O systems lag behind increasing numbers of cores on today's HPC supercomputers, the scalability of scientific simulations becomes highly dependent on the performance of data output phases. With hundreds of thousands of cores that keep generating larger and larger data, it becomes more and more difficult to move, read back and process these data. New approaches to I/O, using in-transit processing capabilities within the simulation resources, must then be considered.

We introduce Damaris, an approach using dedicated cores to address the above I/O issues. It leverages one or a few cores in each multi-core SMP node to enable asynchronous data processing and movements. As opposed to other similar approaches subject to research efforts parallel to ours, Damaris makes a very efficient usage of local memory on SMP nodes. Data is directly allocated in shared memory and written by the computation cores, then asynchronously transferred by the I/O cores to the storage system. This avoids (costly!) data copies and substantially reduces the need for intra-node and inter-node transfers.

Results obtained with one of the challenging target applications of the Blue Waters post-petascale supercomputer (now being delivered at NCSA), clearly demonstrate the benefits of Damaris in experiments with up to 9216 cores performed on the Kraken supercomputer (ranked 11th in the Top500 list). Damaris completely hides all I/O-related costs and variability, and achieves a throughput 15 times higher than standard existing approaches. Besides, it reduces application execution time by 35% compared to the conventional file-per-process approach. Execution time is even reduced by 70% compared to approaches based on collective-I/O! Moreover, it substantially reduces storage requirements, as the dedicated I/O cores enable overhead-free data compression with up to 600% compression ratio. To the best of our knowledge, no concurrent approach demonstrated such improvements in all these directions at such scales. The high *practical* impact of this promising approach has recently been recognized by application communities expected to benefit from the Blue Waters supercomputer and, for these reasons, Damaris was formally validated to be used by these applications on Blue Waters.

Our current work aims to connect visualization software such as VisIt in order to offer asynchronous, overhead-free in-situ visualization capabilities. Preliminary experiments on up to 16K cores of the Jaguar supercomputer (ranked 3rd in the Top500 list) have already shown that Damaris can be used to produce images from our tornado simulation in an asynchronous, straightforward manner.

References

[1] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. DataStager: scalable data staging services for petascale applications. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, HPDC '09, pages 39–48, New York, NY, USA, 2009. ACM.

[2] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan. Scalable I/O forwarding framework for high-performance computing systems. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pages 1–10, September 2009.

[3] G. H. Bryan and J. M. Fritsch. A benchmark simulation for moist nonhydrostatic numerical models. *Monthly Weather Review*, 130(12):2917–2928, 2002.

[4] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig. Small-file access in parallel file systems. *International Parallel and Distributed Processing Symposium*, pages 1–11, 2009.

[5] C. Chilan, M. Yang, A. Cheng, and L. Arber. Parallel I/O performance study with HDF5, a scientific data package, 2006.

[6] A. Ching, A. Choudhary, W. keng Liao, R. Ross, and W. Gropp. Noncontiguous I/O through PVFS. page 405, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[7] P. Dickens and J. Logan. Towards a high performance implementation of MPI-I/O on the Lustre file system. *On the Move to Meaningful Internet Systems OTM 2008*, 2008.

[8] P. Dickens and R. Thakur. Evaluation of Collective I/O Implementations on Parallel Architectures. *Journal of Parallel and Distributed Computing*, 61(8):1052 – 1076, 2001.

[9] 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, 2003.

[10] 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, INRIA, Dec. 2011.

[11] A. Esnard, N. Richart, and O. Coulaud. A steering environment for online parallel visualization of legacy parallel simulations. In *Distributed Simulation and Real-Time Applications, 2006. DS-RT'06. Tenth IEEE International Symposium on*, pages 7–14. IEEE, 2006.

[12] 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.

[13] 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*, pages 1–12. IEEE Computer Society, 2010.

[14] 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*, CLADE '08, pages 15–24, New York, NY, USA, 2008. ACM.

[15] X. Ma, J. Lee, and M. Winslett. High-level buffering for hiding periodic output cost in scientific simulations. *IEEE Transactions on Parallel and Distributed Systems*, 17:193–204, 2006.

[16] NCSA. BlueWaters project, `http://www.ncsa.illinois.edu/BlueWaters/`.

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

[18] J.-P. Prost, R. Treumann, R. Hedges, B. Jia, and A. Koniges. MPI-IO/GPFS an Optimized Implementation of MPI-IO on Top of GPFS. page 58, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

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

[20] 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.

[21] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. PreDatA – preparatory data analytics on peta-scale machines. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010.