2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing

Evaluation of In-Situ Analysis Strategies at Scale for Power Efficiency and Scalability

Ivan Rodero, Manish Parashar
Rutgers Discovery Informatics Institute,
Rutgers University
New Brunswick, NJ, USA
{irodero, parashar}@rutgers.edu

Aaditya G. Landge, Sidharth Kumar, Valerio Pascucci
SCI Institute, University of Utah
Salt Lake City, UT, USA
{aaditya, sidharth, pascucci}@sci.utah.edu

Peer-Timo Bremer
Lawrence Livermore National Laboratory
Livemore, CA, USA
bremer5@llnl.gov

***Abstract*—The increasing gap between available compute power and I/O capabilities is resulting in simulation pipelines running on leadership computing facilities being reformulated. In particular, in-situ processing is complementing conventional post-process analysis; however, it can be performed by using the same compute resources as the simulation or using secondary dedicated resources.**

In this paper, we focus on three different in-situ analysis strategies, which use the same compute resources as the ongoing simulation but different data movement strategies. We evaluate the costs incurred by these strategies in terms of run time, scalability and power/energy consumption. Furthermore, we extrapolate power behavior to peta-scale and investigate different design choices through projections. Experimental evaluation at full machine scale on Titan supports that using fewer cores per node for in-situ analysis is the optimum choice in terms of scalability. Hence, further research effort should be devoted towards developing in-situ analysis techniques following this strategy in future high-end systems.

I. Introduction

Large scale scientific simulations using high performance computing resources are instrumental in understanding complex scientific phenomenon. Over the years, the available compute power for performing these simulations has been increasing enabling simulations of higher spatial and temporal resolutions thereby generating enormous amounts of data. Unfortunately, the I/O capabilities are not increasing at a similar rate becoming a bottleneck in current simulations. As a result, storing data at an effective temporal frequency for post process analysis is becoming increasingly infeasible. To overcome these challenges, in-situ analysis is becoming a favorable solution, where the analysis is performed concurrently with the simulation and only the analysis results are stored to disk. Since the analysis results are typically orders of magnitude smaller than the entire data, the I/O overhead is significantly reduced, allowing the analysis to be performed at a much higher temporal frequency. With the advent of exascale systems, these techniques are expected to be an integral part of the simulation pipeline [9].

Recent work has addressed in-situ analysis in various ways. Most of the efforts have been targeted towards visualization techniques [3], [24], [29], [30] but other types of analysis have been also carried out in-situ [6], [12], [19], [20]. These techniques follow a co-processing model where the analysis is performed using all the compute resources allocated to the simulation. Several frameworks [1], [2], [10], [25], [31] have been developed, where the data is transferred to a subset of the compute resources or to dedicated secondary compute resources over the network for analysis purposes before being stored to disk. The combination of the above two approaches has also been already explored [4].

As several approaches for performing in-situ analysis are available, it is important to make a careful evaluation of these approaches. In this work, we focus on three strategies in which the analysis can be performed on the same compute resources with the simulation. First, we use all of the compute resources of the simulation for the in-situ analysis [6], [12], [20]. As the data is already present in memory there is no data movement involved in this approach. The second strategy is to use a fewer number of cores from every node for performing in-situ analysis. The data from all the cores of the nodes could be moved to these cores by a shared memory or *on-node* data transfer. Lastly, a fewer subset of nodes from the entire machine could be used for analysis by moving the data over the network from the rest of the nodes i.e., *off-node* data transfer [5], [32].

Given the above strategies, making an optimum choice is non-trivial as each strategy has different behaviors in terms of performance and scalability. Furthermore, the choice of the strategy has a direct impact on the energy and power consumption as each strategy has different compute and data transfer characteristics. As we move towards exascale and tighter power budgets, it is crucial to understand and take into account the power behavior of these strategies and the trade-offs with performance and scalability.

In this paper, we evaluate the costs incurred by the three different in-situ analysis strategies mentioned above in terms of execution time, scalability and power/energy consumption. We make use of the parallel merge-tree computation [20] as a representative case for a general class of global feature detection approaches such as identifying connected components, vortex detection, clustering, etc. This computation has the typical characteristic of a global reduction common in most analysis algorithms that try to identify global features and hence becomes a suitable choice for this evaluation. We present the scalability and execution time behavior of this com-

978-1-5090-2453-7/16 $31.00 © 2016 IEEE
DOI 10.1109/CCGrid.2016.95

putation by performing in-situ analysis on Titan, a leadership class supercomputer at Oak Ridge National Laboratory, at full machine scale under the following scenarios:

- Using all the compute cores of the simulation.
- Using a fewer number of cores per node and on-node data movement.
- Using a subset of the nodes and off-node data movement.

As opposed to existing work that addresses energy efficiency and co-design issues at extreme scale by comparing different architectural alternatives [11], [18], we explore power-related issues of in-situ analytics at peta-scale (i.e., on Titan) in the above mentioned scenarios by conducting equivalent empirical experiments on CAPER, an instrumented platform for energy efficiency research. Fine-grained power measurements allow us to capture the behavior of in-situ analytics and extrapolate its power requirements at peta-scale and explore the design space by making projections.

Our evaluation and findings provide a baseline for making optimum choices for the in-situ analysis strategies in terms of performance, scalability and power efficiency.

II. Methods and Strategies

To perform the evaluation of the various in-situ scenarios we make use of the following methodology. First, we need to select an in-situ analysis technique that can serve as a good representative for a broader class of techniques. As feature detection is crucial in gaining scientific insights, we focus on one of the in-situ feature detection techniques. The following provides a description of each of the in-situ strategies along with on overview of how it can be evaluated.

A. Feature Detection Algorithms

In order to gain insight from scientific simulations, scientists are particularly interested in features of interest. These could be burning cells in turbulent combustions [7], [21], eddies in the ocean [28], halos in cosmology [27], etc. Conventionally, feature extraction has been carried out in post-process by analyzing the simulation dumps, but as we move towards exascale, feature detection is moving to in-situ to overcome the I/O bottlenecks as well as to capture features at a high temporal frequency. In this regard, many of the feature detection algorithms have been parallelized [14], [15], [22], [26] and some of them have also been deployed in-situ [20]. All these techniques exhibit a common pattern in terms of design. Typically, a set of local computations is performed on the distributed data set followed by an exchange of data along the boundaries of a block decomposition. Another set of computations then takes into account this neighborhood information. Multiple iterations of these steps are performed until a solution is obtained. Conceptually, this results in a reduction-like pattern, which can also be a limiting factor in scaling such analysis, as the later stages of the reduction typically causes load imbalance problems.

In this paper, we make use of the distributed merge tree computation [20], [22] as a representative for the feature detection algorithms. The merge tree encodes the evolution of connected components of the super-level sets of a given scalar function defined on the given domain, where the super-level set is the region of the domain above a certain function value. The geometric descriptions of the super-level sets are often needed for analysis, for example, to track features, to determine their volumes and shapes, and for visualization, which have been found to be useful in a number of scientific applications [21], [28]. An in-situ implementation of the computation of merge trees on tens of thousand of cores was provided in [20]. Due to its diverse applications, large scale in-situ capability, and characteristic reduction-based framework of the merge tree computation makes it a suitable choice for our evaluation. In this work, we make use of an updated version of the implementation presented in [20]. The next section gives a brief overview of this algorithm in terms of both the computation and the communication pattern involved in the computation.

1) In-situ Merge Tree Computation: The distributed merge tree computation involves three stages. The first stage involves computing the merge trees for the individual blocks of data that are distributed across the compute cores by the on-going simulation. We refer to these individual trees as *local trees*. The second stage then joins subsets of these individual local trees to form the merge trees of the joined blocks. This resultant tree is then given to the participating local trees in the third stage, so that they can correct themselves based on the new information received after joining with the neighboring blocks. The second and third stages are successively performed, every time adding information of the increasing boundary, until the entire domain is covered. This resembles a global merge pattern as shown in Figure 1. The join can be performed with a k-way fan-in that gives rise to an interesting communication pattern where data is not only sent down the join hierarchy but also upwards to the leaves of the hierarchy after every join to the correction phase. The algorithm is implemented using C++ and MPI. We refer the reader to [20] for a detailed description of this algorithm.

In this work, we use an updated implementation of the same algorithm but with a few changes. The earlier implementation computed the merge tree for data points lying within a given function range (thresholds) of the domain. Instead, in this work we perform the computation of the merge tree for the full range of function values and all data points in the domain. Although, since thresholds are data dependent, we decided to not use them in this context as we are using this algorithm as a proxy for a broader class of algorithms. Also, computing the tree for the entire domain is more computationally involved and hence is a good candidate for evaluating the scalability in the various scenarios.

2) Data Sets: We demonstrate our results using two datasets generated by S3D [8], a large-scale direct numerical simulation code that models turbulent combustions. The HCCI data set is a $560 \times 560 \times 560$ simulation of a homogeneous charge compression ignition process in which a lean, premixed fuel-air mixture is compressed until it ignites spontaneously in many separate locations. The HCCI data was generated

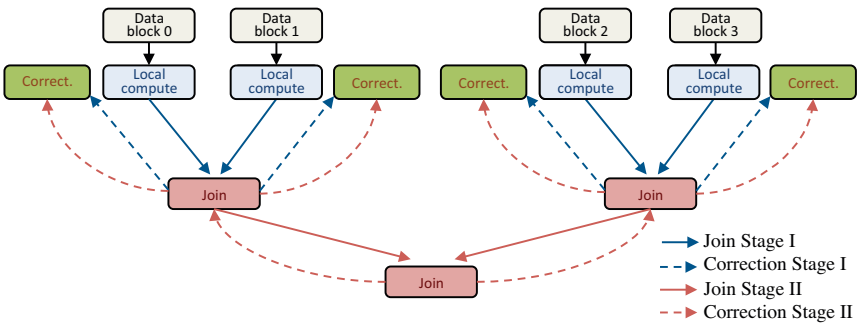


Fig. 1: Dataflow diagram for the binary reduction type merge tree computation. The arrows depict the communication involved.

on Titan at the Oak Ridge Leadership Computing Facility (OLCF). To conduct a scaling study and simulate the exascale work flow, we constructed a larger version by repeating the periodic HCCI data eight times to form a $1120 \times 1120 \times 1120$ volume. The second dataset is the Lifted Flame dataset which is a $1600 \times 400 \times 2025$ volume used to investigate turbulent lifted flames with the goal of better understanding direct injection stratified spark ignition engines for commercial boilers, as well as fundamental combustion phenomena. The Lifted Flame data was also generated on Titan. We doubled the size of this data set along its periodic boundary to create a volume of $1600 \times 800 \times 2025$ for the scaling study.

These two datasets are different in terms of the feature distribution and hence make a good choice for this study to understand the behavior of the analysis under different types of workloads. The HCCI dataset has features distributed in the entire volume, whereas the Lifted Flame has two jets of a flame entering the domain from one side and spanning the center of the domain. All the features are concentrated in the central part of the domain.

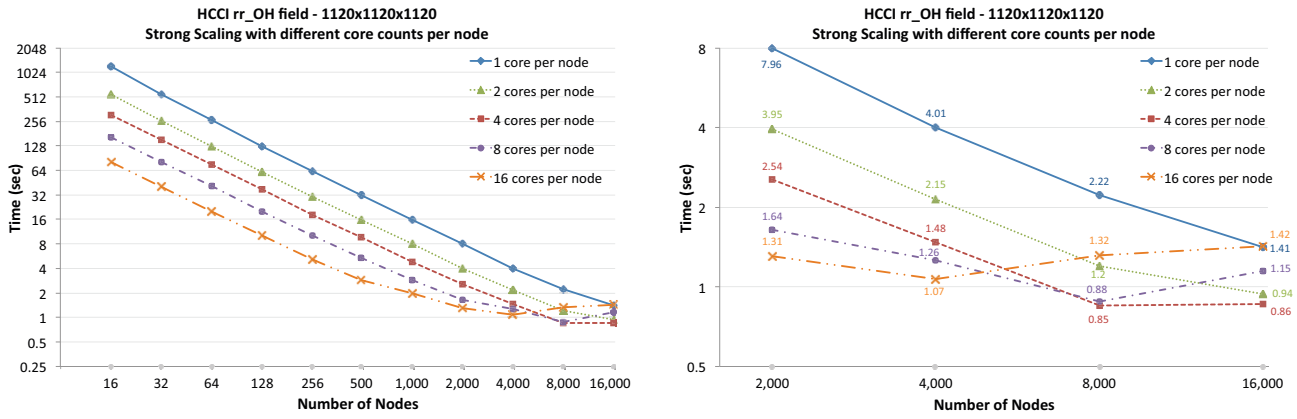
B. In-situ Strategies

As mentioned in Section I, various strategies can be adopted for in-situ analysis. Here, we elaborate on those strategies and how they were evaluated.

1) No Data Movement: In this strategy, the in-situ analysis is performed by all the compute cores of the simulation. As the simulation already has the data in memory, this strategy does not involve any data movement. Since there is no a priori data movement cost, ideally, this strategy should give best performance, but in practice, there are several factors that make this strategy difficult to implement. Firstly, the analysis algorithms have to work within constraints of the simulation in terms of its domain decomposition, core counts, node mappings, etc. These may not always be suitable to the analysis algorithms, thus hindering its performance and scalability. For example at very high core counts, the analysis may not strong scale losing scaling efficiency and sometimes even taking longer than smaller core counts. For our evaluation, we make use of DIY [23] to decompose the datasets and distribute them to the MPI processes. This acts as a scenario where the simulation data is already in memory. The merge tree computation is then performed on this data decomposition using all the MPI processes. Since DIY is used only for loading the data into memory, we do not measure its performance or power consumption.

2) On-Node Data Movement: To mitigate the drawbacks of the above strategy, data movement strategies within the node are employed. In this strategy, we explore the option of using only few cores from each compute node for the analysis. This requires moving or aggregating the data from all the cores of the node to a fewer cores on the node, which would participate in the analysis. In [17], it is shown that on-node data transfer between MPI ranks can be done efficiently using shared memory in a fast manner. So in our scalability experiments on Titan, we consider the data movement to have negligible execution time. Instead of performing the on-node data movement, we decompose the data into parallelepipeds using DIY [23], in a fashion that already represents the aggregated data within the node. We then allocate only 1, 2, 4 or 8 MPI processes per node on Titan while doing the scaling studies. For the power analysis, the data movement cost cannot be ignored. So, a prototype data movement scheme was implemented on CAPER to get accurate power behavior.

3) Off-Node Data Movement: In this strategy, the data is moved to a subset of nodes in the system. All the cores from these nodes are then used to perform the analysis. The drawback of this approach is that it requires the data to be moved from all of the compute nodes to the selected set of nodes over the system interconnect. This adds a significant overhead both in terms of time as well as energy spent in moving the data. Although, there are disadvantages, this technique works well when integrated with the I/O frameworks [25] and if the data movement is done in an asynchronous fashion [1], [2], [5], [10], [31], [32]. However, this inhibits the analysis frequency as data is moved to the staging area less frequently due to the significant data movement costs. Also, even though successful deployments of such strategies have been done, they do not take into account the power impact of this strategy. In our evaluation, we make use of our prototype data movement implementation on CAPER, to move the data to a subset of nodes for the power study. For the large scale runs on Titan, we expect the analysis to perform in the same fashion as



(a) (b)

Fig. 2: Time taken by the analysis on Titan for (a) the HCCI data set on various node counts by utilizing 1, 2, 3, 4, 8 or 16 cores per node. (b) A zoomed in view with only the higher range of node counts. We see that using fewer cores per node gives better performance.

the strategy discussed in Section II-B1 with an extra cost in terms of data movement. We assume that such a transfer can be performed with any of the existing infrastructures in an asynchronous fashion. Upon completion of the data transfer, the situation is same as in Section II-B1 of running the analysis using all the cores on the node, but only this time on a smaller number of nodes. As a result, there is no need to perform separate set of experiments to understand the scaling and performance of this strategy as it would be the same as in the case of Section II-B1.

III. Experimental Evaluation

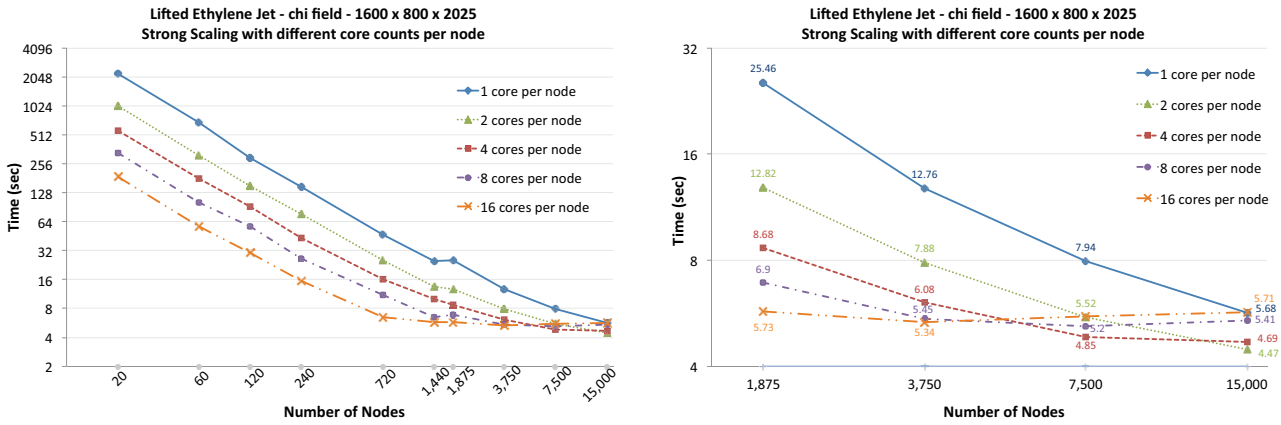
A. Performance and Scalability Analysis

1) Hardware Setup: For our evaluation, we performed the scalability and performance experiments on Titan by strong scaling both the HCCI and Lifted Flame datasets. Titan is a peta-flop Cray XK7 system with 18,688 nodes each with a 16-core AMD Opteron 2.2 GHz processor for a total of 299,008 compute cores.

HCCI dataset. Figure 2 shows the execution time and scaling behavior for this dataset using varying number of cores per node. At the lower node counts the per process block decomposition size is large and the computation is dominated by the compute intensive, yet data parallel, local computation phase. As a result, having more cores involved in the analysis at overall lower core counts gives best performance. In fact, at the lowest node count using all the cores is almost $15\times$ faster than using only a single core. We see that more number of cores per node gives the best performance up to 4,000 nodes at which point the local block size has become $14 \times 28 \times 56$, while using all the cores on the node. As the work load per process has reduced, the scaling efficiency of using all the cores in the node decreases significantly after this point and the execution time is dominated by the communication costs. In fact, the execution time for using all the cores increases from 1.07s at 4,000 nodes to 1.42s at 16,000 nodes. By doing on-node data aggregation and using fewer cores, we increase the work per process and hence achieve better performance. In Figure 2b, we see that after 4,000 nodes its beneficial to use fewer cores per node. The best performance after 4,000 nodes is achieved by using 4 cores per node, computing the tree in 0.85s at 8000 nodes, but even this has poor scaling behavior and the execution time tends to rise at 16,000 nodes to 0.86s. In general, we see a trend that using lower number of cores per node after 4,000 nodes gives better performance, but at the same time loses scaling efficiency quite fast when the node count is further increased as seen in the case for 8 cores per node and 4 cores per node.

Lifted Flame dataset. As shown in Figure 3 we see similar behavior as in the HCCI dataset but in this case the scaling efficiency drops rapidly past 720 nodes when using all the cores on the node. The execution time reduces by just 11% going from 6.49s at 720 nodes(11,520 cores) to 5.78s at 1,440 nodes (23,040 cores). As this dataset has a concentration of features at the center of the domain, there is an inherent load imbalance and hence we experience poor scalability as compared HCCI. As seen in Figure 3b, the execution time while using all the cores in the nodes reduces marginally after $1,875$ from 5.73s to 5.34s at 3,750 nodes, but slowly tends to rise again and eventually matches the execution time of a single core run at machine scale. In this dataset we observe that using 4 cores per node also loses scaling efficiency and the best performance at scale is achieved by using 2 cores per node computing the tree in 4.47s.

Discussion. Based on the performance and scalability behavior, we see that using all cores of a node results in good performance initially but we reach a point where the performance for this strategy starts to dip after which, using



(a) (b)

Fig. 3: Time taken by the analysis on Titan for (a) the lifted data set on various node counts by utilizing 1, 2, 3, 4, 8 or 16 cores per node. (b) A zoomed in view with only the higher range of node counts. We see that using fewer cores per node gives better performance.

fewer cores per node and on-node data movement provides better performance. Extrapolating this plot, for higher node counts, one would use fewer and fewer cores per node until even using only a single core would exhibit poor scalability. At this point one would have to use off-node data movement to further reduce the core count by aggregating the data onto a smaller subset of nodes. However future high-end systems are expected to have a constant or even decreasing number of nodes compared to existing systems with respect to the core count [9]. As a result it seems unlikely that in the future an off-node data transfer would outperform intra-node aggregation. This is especially true as off-node communication would further increase the power consumption as well as the software complexity when compared to the shared memory exchanges within a node.

B. Power Analysis and Study of Trade-offs

1) Hardware Setup: The power-centric evaluation has been conducted on the NSF-funded research instrument "Computational and dAta Platform for Energy efficiency Research" (CAPER). This is an eight-node cluster based on SuperMicro SYS-4027GR-TRT system, which is capable of housing concurrently, in one node up to eight general-purpose graphical processing units (GPGPU), or eight Intel many-integrated-core (MIC) coprocessors – or any eight-card combination of the two; and up to 48 hard disk drives (HDD), or solid-state drives (SSD). However, the configuration used in this experimental evaluation features servers with two Intel Xeon Ivy Bridge E5-2650v2 (16 cores/node), 128GB of DRAM, and Infiniband FDR network connectivity. Accelerators were not deployed. This platform also mirrors key architectural characteristics of high-end system, which allow us to extrapolate our models to larger systems and make projections towards exascale co-design. Furthermore, CAPER is instrumented with both coarse- and fine-grained power metering at server level – an instrumented Raritan PDU provides power measurements at 1Hz, and a Yokogawa DL850E ScopeCorder data acquisition recorder provides power measurements from current and voltage modules. Our experimental evaluation was conducted using the fine-grained instrumentation system at server level with power readings at 50Hz. Power readings were obtained integrating current and voltage readings at 500KHz, which allowed a high accuracy level. The fine-grained metering capability is essential in this work because the execution time of the studied data analysis is very short and power/energy cannot be measured appropriately with typical PDU-level power metering capability. Network power measurements were based on server-level power measurements as the infiniband FDR switch didn't show power variability (i.e., can be considered static power) as existing studies [16] already pointed out.

2) "Power-friendly" Problem Implementation: In our experiments we used an appropriately downscaled version of the problem to understand its power behavior. We also had to take into account power considerations to let us capture the power behavior associated to the data restructuring and in-situ merge-tree computation.

Figure 4 shows the system power dissipation over time for two implementations of the downscaled version of the problem with HCCI data set, and using 128 MPI processes and 1 core per node. As detailed in the right of Figure 4, we introduced a one-second sleep call before and after the data restructuring to better differentiate the different phases of the program. The sleep call is performed by the master MPI rank and the rest of MPI processes are on a barrier.

While the figure in the left uses traditional synchronous MPI collective operations, the figure in the right uses asynchronous MPI collective operations available in MPI 3.0. The figure in the left illustrates how power dissipation is maximized not only

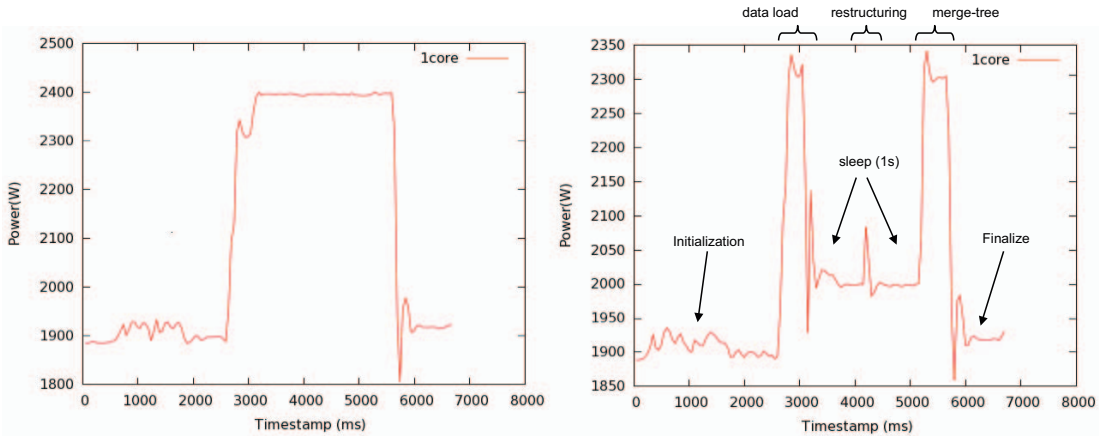


Fig. 4: System power dissipation over time for HCCI using 128 MPI processes and 1 core per node. Note that Y axes of the figures are in different scales. The figure in the left is obtained with the original implementation with synchronous MPI collective operations. The figure in the right is obtained with the energy friendly implementation, which shows that asynchronous MPI collective operations (e.g., during 1 second sleeps) require lower power.

during data load, data restructuring and merge-tree phases but also during the sleep periods. This is explained due to the fact that synchronous collective operations are designed for performance (i.e., using a busy waiting mechanism) therefore require a significant amount of power. However, the figure in the right shows that using asynchronous MPI collectives the power is significantly reduced during idle periods (i.e., sleep calls).

Our implementation based on asynchronous MPI collectives uses *MPI_Ibarrier*, *MPI_Iallgather* and *MPI_Iallreduce* operations and (nano) sleep periods of *500ns* in a *MPI_Test* loop. There is no significant performance impact as these collective operations are not used intensively; however, it significantly impacts energy consumption, but more importantly, power dissipation which is a critical concern in current and next-generation high-end systems.

This observation clearly supports that "power-friendly" implementations of both MPI codes and runtimes need to be considered at scale.

3) Power Behavior Analysis: In order to understand the power behavior of the in-situ analysis at scale, we simulated an equivalent workload on CAPER as Titan by providing each MPI process the same amount of data as it would be allocated at large scale on Titan.

Figures 5 and 6 display the execution time required for the data restructuring and in-situ merge-tree computation, the system energy consumption (i.e., the energy consumed by all CAPER nodes) and average active power (also known as dynamic power). Active power does not consider the idle power, which is the nominal power dissipation when the system is idle. We use active power to isolate the power requirements of the in-situ analysis and let us make projections to larger scale. The figures present the results for different configurations and both HCCI and Lifted Flame data sets. The "Base" configuration represents in-situ analysis using 16 cores per node. The "1 node (8 core/node)" and "2 nodes (4 cores/node)" configurations use a reduced number of nodes (one and two, respectively) with higher core counts. The variability in the measurements was not significant as most of the metrics are based on averages and/or numerous data points.

In general, the results show that larger core counts per node reduces the execution time and energy consumption, as energy is correlated with execution time; however, the power dissipation is higher. Since future-generation systems are expected to be constrained by power, configurations with highest power requirements might not be able to be run together with simulation codes at scale. As a result, in-situ analysis may have to tolerate some level of performance degradation and therefore higher energy consumption to maintain simulation-analysis workflows running in leadership-class supercomputers within their power budgets. Both "1 node (8 core/node)" and "2 nodes (4 cores/node)" configurations consider the static energy consumption of all nodes in the system. This is based on the assumption that in-situ analysis will run across the platform together with the simulation. Further, while it significantly impacts energy consumption, average active power is comparable (or lower) to the other configurations. The figures also show that "1 node (8 core/node)" and "1 core/node" (using 8 nodes) configurations present similar results. These two observations present opportunities for exploiting available cores when the cost of data movement is not in the critical path or the available resources have advanced capabilities (e.g., closer to fast data storage such as flash-based burst buffers). Figures 2 and 3 showed that, at scale, the on-node gathering might not only be fastest; however, Figures 5 and 6 support that it is always more power efficient. Furthermore, the off-node aggregation is at best equal in power to on-node and that is without considering the power for data transfer, which supports our argument that off-node is not a viable solution.

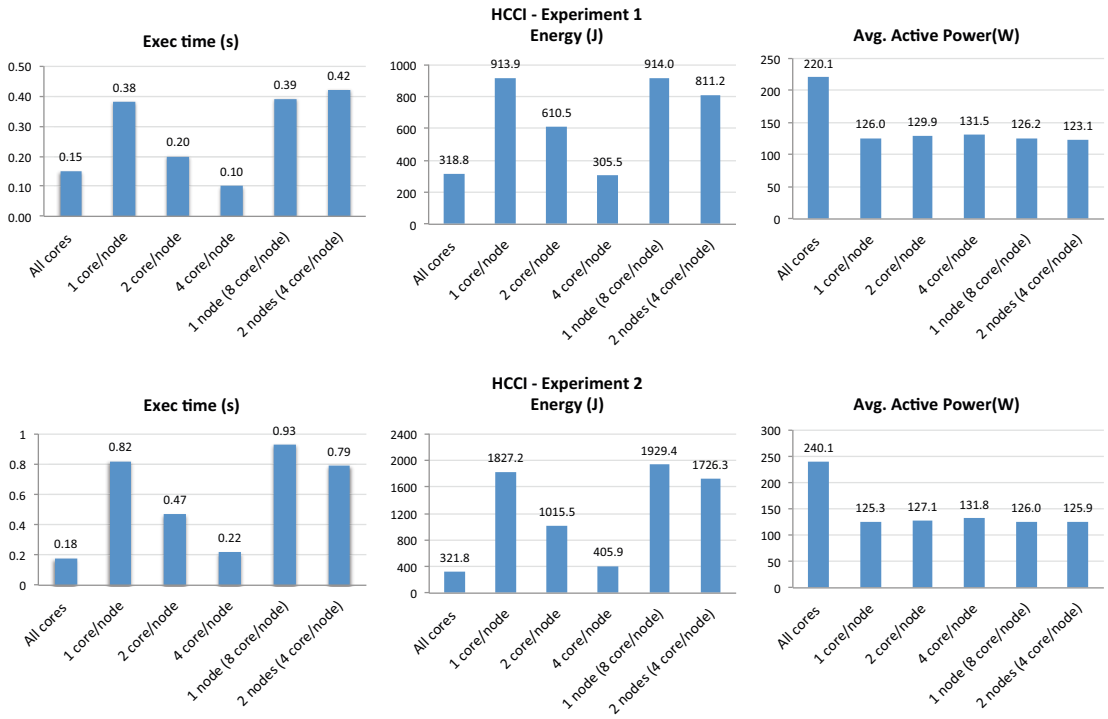


Fig. 5: Execution time, energy consumption and active power dissipation on CAPER for the HCCI data set and the data configurations associated to the two different Titan sizes.

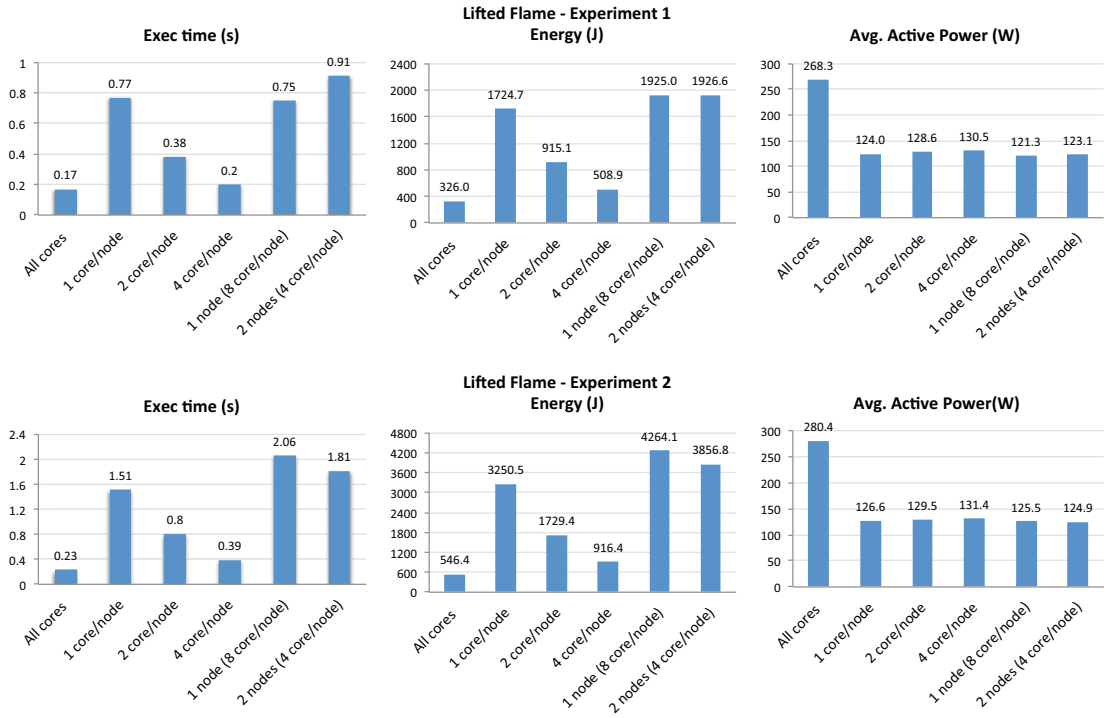


Fig. 6: Execution time, energy consumption and active power dissipation on CAPER for the Lifted flame data set and the data configurations associated to the two different Titan sizes.

4) Extrapolation to Titan Scale: We extrapolate the measurements obtained with CAPER to Titan based on its known power requirements and hardware vendor specifications. More specifically, we estimate the static system power from the power model used in [13], which is based on the following reasoning. Titan drains 8,209KW at full capacity, which results 439W per node (from a total of 18,668 nodes). Considering that both dynamic and static power for the processor, memory and GPU are well known, we estimate that Titan nodes' static power is ∼61W. The active power is scaled from CAPER to Titan based on the processors and memory vendor specifications.

Figure 7 illustrates the estimated power dissipation of Titan at full-system scale for the Lifted Flame data set. Note that power dissipation in this case is based on maximum power dissipation from experiments in CAPER because we are interested in the peak power and not in the nominal power. The plot shows the estimation and also projections using 16 cores per node and 1 core per node as they represent the upper and lower bounds. The figure shows a difference of 50KW between the two configurations; the rest of configurations are comprised between these two. Taking into account that the data analysis will run in-situ with CPU-intensive computations (i.e., simulation codes) we might not be able to use larger core counts due to the power constraints.

Figure 7 also illustrates different projections with the assumption that future systems will be more energy efficient and therefore their static power will be reduced. Figure 7 displays projections assuming systems with 2-10% lower static power dissipation than Titan. The figure shows that 4% more energy efficient hardware would allow using 16 cores at the same power budget that 1 core/node in the current Titan architecture. This observation give us a baseline to understand what level of energy efficiency improvement is required to execute the in-situ analysis without performance degradation and how much degradation has to be tolerated given specific design choices.

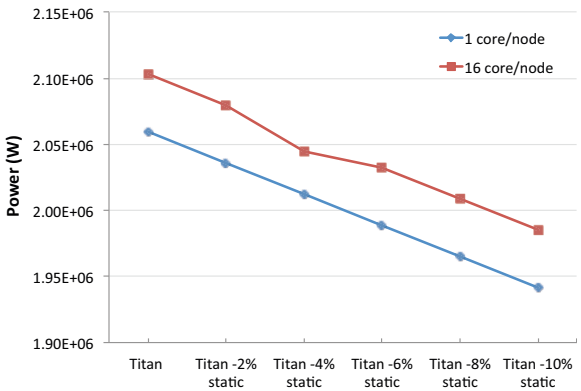


Fig. 7: Power extrapolation to Titan-scale based on executions in CAPER and power projections for different energy efficiency system design choices (i.e., reduced static power dissipation in 2-10%).

IV. Conclusion

This paper evaluated three in-situ analysis strategies depending on the way the compute resources are used for the analysis: i) using all the compute cores for the analysis without any a priori data movement, ii) on-node data movement and using fewer cores from each of the compute nodes, and iii) off-node data movement and using a subset of the compute nodes for performing the analysis. We evaluated the costs incurred by each of the above strategies in terms of performance, scalability and power. For our experimental evaluation, we deployed the merge-tree computation in-situ using the above mentioned strategies at full machine scale (16,000 nodes) on Titan supercomputer for the performance and scalability analysis. We also performed scaled but equivalent experiments on CAPER with fine-grained power metering to understand the power behavior of these strategies. Using the obtained power behavior, we extrapolated the power requirements in Titan and made projections based on energy efficiency design choices.

Based on our analysis, we conclude that up to a certain number of nodes using all the cores for analysis gives best performance and consumes lower energy; however, it requires significantly higher power. Furthermore, we observed that using all cores per node at full machine scale is not scalable hence we use fewer cores with on-node data movement which gives better performance, scalability and power efficiency. Based also on our findings and extrapolating our results to even larger node counts than the available in Titan, we expect this strategy to also stop scaling. In this case, making use of a subset of the compute resources would be a viable choice. However, future high-end systems are expected to have nodes with larger core counts and fewer nodes with respect to the core count. In this case, using fewer cores per node for analysis purposes is expected to be an optimum choice and more research effort should be devoted towards developing in-situ analysis techniques following this strategy.

Our ongoing work also includes the study of deep memory hierarchies using different memory devices (e.g., NVRAM) as the analysis is expected to be co-located with simulation codes on the same physical resources thus main memory capacity and bandwidth may be very limited.

Acknowledgments

The research presented in this work is supported in part by National Science Foundation (NSF) via grants numbers ACI-1339036, ACI-1310283, ACI-1464317, CNS-1305375, and DMS-1228203, by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the US Department of Energy Scientific Discovery through Advanced Computing (SciDAC) Institute of Scalable Data Management, Analysis and Visualization (SDAV) under ward number DE-SC0007455, the Advanced Scientific Computing Research and Fusion Energy Sciences Partnership for Edge Physics Simulations (EPSI) under award number DE-FG02-06ER54857, the ExaCT Combustion Co-Design Center via subcontract number 4000110839 from UT Battelle, and by an IBM Faculty Award.

The research at Rutgers was conducted as part of the Rutgers Discovery Informatics Institute (RDI2).

REFERENCES

[1] H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan, and S. Klasky. Just In Time: Adding Value to The IO Pipelines of High Performance Applications with JITStaging. In *Proc. of 20th International Symposium on High Performance Distributed Computing (HPDC'11)*, June 2011.

[2] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. Datastager: scalable data staging services for petascale applications. In *Proc. of 18th International Symposium on High Performance Distributed Computing (HPDC'09)*, 2009.

[3] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. H. Rogers, and M. Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 424–434, Piscataway, NJ, USA, 2014. IEEE Press.

[4] J. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proc. ACM/IEEE Conference on Supercomputing (SC12)*, 2012.

[5] J. Biddiscombe, J. Soumagne, G. Oger, D. Guibert, and J.-G. Piccinali. Parallel computational steering and analysis for hpc applications using a paraview interface and the hdf5 dsm virtual file driver. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, EGPGV '11, pages 91–100, Aire-la-Ville, Switzerland, Switzerland, 2011. Eurographics Association.

[6] J.-M. F. Brad Whitlock and J. S. Meredith. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proc. of 11th Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'11)*, April 2011.

[7] P.-T. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. B. Bell. Interactive exploration and analysis of large scale simulations using topology-based data segmentation. *IEEE Trans. on Visualization and Computer Graphics*, 17(99), 2010.

[8] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorski, R. Sankaran, S. Shende, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using s3d. *Computational Science and Discovery*, 2:1–31, 2009.

[9] A. Choudhary, T. Critchlow, S. Klasky, K.-L. Ma, and V. Pascucci. Scientific discovery at exascale: Report from the doe ascr 2011 workshop on exascale data management, analysis, and visualization. 2011.

[10] C. Docan, M. Parashar, and S. Klasky. DataSpaces: An Interaction and Coordination Framework for Coupled Simulation Workflows. In *Proc. of 19th International Symposium on High Performance and Distributed Computing (HPDC'10)*, June 2010.

[11] D. Donofrio, L. Oliker, J. Shalf, M. F. Wehner, C. Rowen, J. Krueger, S. Kamil, and M. Mohiyuddin. Energy-efficient computing for extreme-scale science. *Computer*, 42(11):62–71, Nov. 2009.

[12] N. Fabian, K. Moreland, D. Thompson, A. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *Proc. of IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 89 –96, October 2011.

[13] M. Gamell, I. Rodero, M. Parashar, J. Bennett, H. Kolla, J. Chen, P.-T. Bremer, A. Landge, A. Gyulassy, P. McCormick, S. Pakin, and V. Pascucci. Exploring power behaviors and trade-offs of in-situ data analytics. In *Proc. ACM/IEEE Conference on Supercomputing (SC13)*, 2013.

[14] A. Gyulassy, T. Peterka, R. Ross, and V. Pascucci. The parallel computation of Morse-Smale complexes. *IEEE International Parallel and Distributed Processing Symposium, to appear*, 2012.

[15] W. Hendrix, D. Palsetia, M. Patwary, A. Agrawal, W.-K. Liao, and A. Choudhary. A scalable algorithm for single-linkage hierarchical clustering on distributed memory architectures. In *Proceedings of 3rd IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, 2013.

[16] T. Hoefler. Software and hardware techniques for power-efficient hpc networking. *Computing in Science Engineering*, 12(6):30–37, Nov 2010.

[17] T. Hoefler, J. Dinan, D. Buntinas, P. Balaji, B. Barrett, R. Brightwell, W. Gropp, V. Kale, and R. Thakur. Mpi+ mpi: a new hybrid approach to parallel programming with mpi plus shared memory. *Computing*, 95(12):1121–1136, 2013.

[18] J. Krueger, D. Donofrio, J. Shalf, M. Mohiyuddin, S. Williams, L. Oliker, and F.-J. Pfreund. Hardware/software co-design for energy-efficient seismic modeling. In *Proc. of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, pages 73:1–73:12, 2011.

[19] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, H. Kolla, S.-H. Ku, S. Ethier, J. Chen, C. Chang, S. Klasky, R. Latham, R. Ross, and N. Samatova. Isabela-qa: Query-driven analytics with isabela-compressed extreme-scale scientific data. In *Proc. of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1 –11, November 2011.

[20] A. G. Landge, V. Pascucci, A. Gyulassy, J. C. Bennett, H. Kolla, J. Chen, and P.-T. Bremer. In-situ feature extraction of large scale combustion simulations using segmented merge trees. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 1020–1031, Piscataway, NJ, USA, 2014. IEEE Press.

[21] A. Mascarenhas, R. W. Grout, P.-T. Bremer, E. R. Hawkes, V. Pascucci, and J. H. Chen. Topological feature extraction for comparison of terascale combustion simulation data. In V. Pascucci, X. Tricoche, H. Hagen, and J. Tierny, editors, *Topological Methods in Data Analysis and Visualization*, Mathematics and Visualization, pages 229–240. Springer Berlin Heidelberg, 2011.

[22] D. Morozov and G. H. Weber. Distributed merge trees. In A. Nicolau, X. Shen, S. P. Amarasinghe, and R. Vuduc, editors, *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '13, Shenzhen, China, February 23-27, 2013*, pages 93–102. ACM, 2013.

[23] T. Peterka, R. Ross, W. Kendall, A. Gyulassy, V. Pascucci, H.-W. Shen, T.-Y. Lee, and A. Chaudhuri. Scalable parallel building blocks for custom data analysis. In *Proceedings of Large Data Analysis and Visualization Symposium LDAV'11*, Providence, RI, 2011.

[24] T. Tu, H. Yu, L. Ramirez-Guzmanz, J. Bielak, O. Ghattas, K.-L. Ma, and D. R. O'Hallaron. From Mesh Generation to Scientific Visualization: An End-to-End Approach to Parallel Supercomputing. In *Proceedings of ACM/IEEE Supercomputing Conference*, 2006.

[25] V. Vishwanath, M. Hereld, and M. Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In *Proc. of IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, October 2011.

[26] B. Welton, E. Samanas, and B. Miller. Mr. scan: Extreme scale density-based clustering using a tree-based network of gpgpu nodes. In *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*, pages 1–11, Nov 2013.

[27] W. Widanagamaachchi, P.-T. Bremer, C. Sewell, L.-T. Lo, J. Ahrens, and V. Pascucci. Data-parallel halo finding with variable linking lengths. In *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*, pages 27–34. IEEE, 2014.

[28] S. Williams, M. Petersen, P.-T. Bremer, M. Hecht, V. Pascucci, J. Ahrens, M. Hlawitschka, and B. Hamann. Adaptive extraction and quantification of atmospheric and oceanic vortices. *IEEE Trans. Vis. Comp. Graph.*, 17(12):2088–2095, 2011.

[29] H. Yu, T. Tu, J. Bielak, O. Ghattas, J. C. López, K.-L. Ma, D. R. O'Hallaron, L. Ramirez-Guzmanz, N. Stone, R. Taborda-Rios, and J. Urbanic. Remote Runtime Steering of Integrated Terascale Simulation and Visualization. In *ACM/IEEE Supercomputing Conference HPC Analytics Challenge*, 2006.

[30] H. Yu, C. Wang, R. Grout, J. Chen, and K.-L. Ma. In Situ Visualization for Large-Scale Combustion Simulations. *IEEE Computer Graphics and Applications*, 30(3):45–57, 2010.

[31] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. PreDatA - preparatory data analytics on peta-scale machines. In *Proc. of 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10)*, April 2010.

[32] F. Zheng, H. Zou, G. Eisenhauer, K. Schwan, M. Wolf, J. Dayal, T.-A. Nguyen, J. Cao, H. Abbasi, S. Klasky, N. Podhorszki, and H. Yu. Flexio: I/o middleware for location-flexible scientific data analytics. *Parallel and Distributed Processing Symposium, International*, 0:320–331, 2013.