In-Situ I/O Processing: A Case for Location Flexibility

Fang Zheng, Hasan Abbasi, Jianting Cao, Jai Dayal, Karsten Schwan,
Matthew Wolf, Scott Klasky*, Norbert Podhorszki*
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{fzheng, habbasi, jcao32, jdayal3, schwan, mwolf}@cc.gatech.edu
*National Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, TN 37831
{klasky, pnorbert}@ornl.gov

ABSTRACT
Increasingly severe I/O bottlenecks on High-End Computing machines are prompting scientists to process output data during simulation time, "in-situ", and before placing data on disks. This paper argues for flexibility in the implementation of such in-situ data analytics, using measurements and a performance model that demonstrate the potential advantages and limitations of performing analytics at different levels of the I/O hierarchy, including on a machine's compute nodes vs. on separate "staging" nodes dedicated to analysis tasks. Model and measurement results are guided by realistic large-scale applications running on leadership class machines, and I/O and analytics actions are described as computational dataflow graphs - termed I/O graphs - that combine data movement with 'in transit' operations on data as it is being moved across the I/O hierarchy. Results demonstrate the importance of flexibility in analytics placement and characterize the attributes of analytics operations that lead to different placement decisions.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: *Performance attributes.*

General Terms
Performance, Experimentation, Measurement.

Keywords
I/O, In-Situ Processing, Staging, Placement, Analytics.

1. INTRODUCTION
The need for in-situ processing to cope with the extreme volumes of output data generated by scientific simulations (and beyond) has become an almost inevitable fact for current petascale and next generation high end machines. Tasks performed by enhanced output pipelines include data reduction and filtering prior to data movement, data reorganization for improved storage access or to enable useful analysis, and data processing and analysis to better understand simulation progress or status and to provide end users with rapid insight into scientific outcomes produced by simulations.

Work on in-situ I/O processing has been motivated by the increasingly severe I/O bottlenecks faced by large-scale scientific simulations and associated data analyses and visualizations. This is because high end simulations are generating ever larger data volumes to be analyzed/visualized in order to gain scientific insights, while there is an increasingly large disparity between the I/O and computational capabilities on most High-End Computing (HEC) machines[23]. Factors such as contention on shared resources[21] and complicated I/O patterns[19] further exacerbate the attainable I/O performance on those platforms. The combined effect results in suboptimal situations where a substantial portion of the simulation runtime is spent in writing data to the storage system[23]. Furthermore, scientifically important analysis and visualization on the output data incurs considerable I/O overhead (e.g., it has been reported that data read times from storage can consume up to 98% of the total runtime for large-scale visualizations[9]). Another important issue is the undue power consumption of large and/or repeated data movements into and out of storage[26].

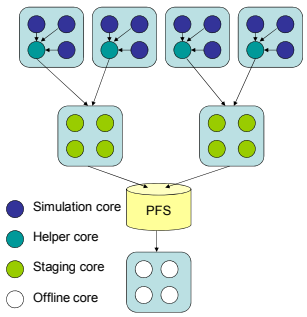

Figure 1. Different In-Situ I/O Processing Techniques.

In-situ techniques for I/O processing have shown promise in addressing I/O bottlenecks on high end machines. Below, we summarize representative techniques developed over the last few years, using a classification derived from [32].

PDSW'11, November 13, 2011, Seattle, Washington, USA.

Inline Processing: Analysis/visualization routines are synchronously performed by the simulation. ParaView's co-processing library[12], VisIt's remote visualization[31] and other in-situ visualization work[32] fall into this category.

Helper cores: some cores on the compute nodes where simulation runs are dedicated to perform select analysis actions. Examples include Functional Partitioning[18] and Software Accelerator[27].

Staging Area Processing: on its way from computation to storage, data is routed via an additional set of compute nodes on which it can be temporarily buffered, analyzed, and visualized. Our previous work in PreDatA[34] and Data Services[2], as well as DataSpaces[11], GLEAN[16], and HDF5/DSM[6], follows this approach.

Active Storage: certain computational routines are deployed in I/O nodes and triggered to operate on data written and/or read by the simulation[25][28].

Offline Processing: data written to storage is read back for additional or long term analysis or visualization[14], typically assisted by workflow tools[22].

The key factor distinguishing the above data processing techniques is "where" analytics computations are placed along the I/O path. Such placements determine which resource is allocated to which computations and how/when data is moved. Consequently, computation placement along the I/O path has significant impact on both the performance (e.g., runtime) and the cost (e.g., CPU hours) of the resulting coupled simulation and analysis codes. Further and as quantitatively demonstrated below, there are trade-offs among different placement strategies in terms of performance and cost, and these trade-offs vary across different analytics codes, data volumes, and scales. Therefore, flexibility in analytics placement is a key requirement for in-situ I/O processing.

The main technical contribution described in this paper is a quantitative formulation of the performance/cost trade-offs seen for different placement strategies. We also present useful metrics that capture and measure trade-offs. This is useful for scientific end users choosing placements for specific applications and analytics, provides insights to system/tool developers, and constitutes a first step toward developing automated functionality for future in-situ I/O processing systems. The work is novel in that it generalizes from specific measurements and costs reported in prior work and permits quantitative comparisons between different approaches and technique. This supplements other efforts[8] focused on the different usage models and software engineering implications associated with various in-situ processing methods.

The remainder of the paper is organized as follows. Section II defines performance and cost metrics and formulates performance models to characterize the trade-offs in in-situ I/O processing. Section III presents our middleware system's support for flexible placement. Section IV reports performance results on constructing and accelerating I/O processing for a large-scale scientific application on Oak Ridge National Laboratory's Cray XT5 platform, which demonstrates the benefits of flexible placement and empirically verifies the proposed performance model. Section V concludes the paper and discusses future work.

2. MODELING IN-SITU I/O PROCESSING

In order to quantitatively evaluate various in-situ I/O processing approaches, we define a set of performance and cost metrics. We then derive simple performance models to compare the Inline vs. Staging approaches, both of which have been actively developed and successfully applied in practice.

2.1 Performance and Cost Metrics

Total Execution Time: the time from the start of simulation and analysis to the completion of both, also termed "Time to Insight" in our prior work[1]. This performance metric measures how long it takes to finish both the simulation and analysis, the idea being that it is the scientific insights derived from data analysis that drive science end users[5].

Total CPU hours: the total nodes used multiplied by the total execution time (in units of hours). This metric measures the cost of a run, as supercomputing centers commonly charge end users with the total number of CPU hours consumed by their jobs. Another useful cost metric is a run's total power consumption[13], which we will consider in our future work.

2.2 Comparison between Inline and Staging Approaches

For modeling purpose, we consider the simple but common usage scenario of in-situ I/O processing (shown in Figure 2). In this case, the simulation periodically generates output data and passes the data to an analysis component, which then immediately performs certain processing actions. The term “analysis” is used to denote actions ranging from simply writing data to storage, to data analytics such as feature extraction, indexing, compression, to the processing needed for coupling scientific codes[33], to data conversions for storage, and data visualization. We assume a processing model in which such actions are arranged as computational dataflow graphs, where each such directed graph describes the inputs/outputs of individual, indivisible analysis actions and the data movements between them. The formulation shown below may be applied to any bi-section cut across this graph to evaluate the placement of all computations before and/or after the cut.

Table 1. Major notations used in model

$Psim$	Total number of nodes on which simulation is run
Pa	Total number of nodes in staging area (if present)
$Tsim(P)$	Simulation’s wall-clock time between two consecutive I/O actions when running on P nodes
$Ta(P)$	Analysis’ wall-clock time for processing one simulation output step when running on P nodes
K	Total number of I/O dumps
α	$\alpha = Pa/Psim$
β	$\beta = Ta(Psim)/Tsim(Psim)$
$Tsend$	Simulation-side visible data movement time
$Trecv$	Staging node-side visible data movement time
s	Slowdown factor of simulation

We denote the *Total Execution Time* using Inline and Staging approaches as *Tinline* and *Tstaging*, respectively. Figure 2(b) and

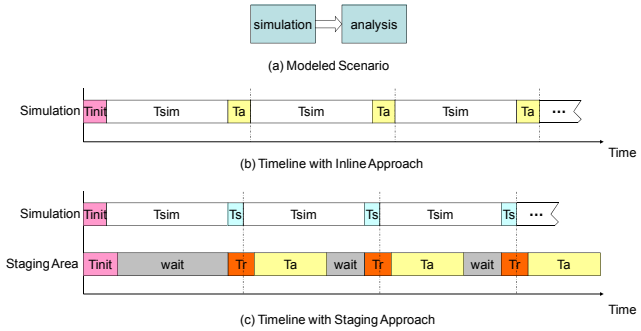

Figure 2. Timeline for Inline and Staging Approaches.

2(c) separately show the timeline of simulation and analysis with the Inline and Staging approaches. Omitting the initialization and finalization phases of long running simulation, *Tinline* and *Tstaging* can be calculated as follows:

$$Tinline = K \times [Tsim(Psim) + Ta(Psim)]$$

$$Tstaging = K \times \max\{Tsim(Psim) \times s + Tsend, Trecv + Ta(Pa)\}$$

Define the performance speedup of using Staging over Inline:

$$Speedup = \frac{Tinline}{Tstaging}$$

And let $\alpha = Pa / Psim$(size of staging area as percentage of total simulation nodes), and $\beta = Ta(Psim) / Tsim(Psim)$. This results in:

$$Speedup = \frac{Tsim(Psim)(1+\beta)}{\max\{Tsim(Psim) \times s + Tsend, Trecv + Ta(Psim \times \alpha)\}}$$

An upper bound on Speedup can be derived as:

$$Speedup < (1+\beta)/s$$

In the formula above, β denotes analysis time as percentage of simulation time at the scale of *Psim* nodes, and *s* is the slowdown factor of simulation time due to staging (s≥1) (e.g., slowdown due to network contention caused by additional data movements needed for staging[3], if any).

Since the Staging approach uses additional *Pa* nodes to offload analysis and may improve the total execution time through pipelining effect, it is interesting to understand the conditions under which Staging can achieve the maximum speedup with some associated cost. Figure 3 shows three different possible relationships between staging area size (α) and Speedup. In each figure, there are three regions: inefficient region (or sub-linear speedup region, colored yellow), efficient region (or super-linear speedup region, colored blue), and over-provisioned region (where no more speedup could be gained by increasing size of staging area, colored purple).

Figure 3(a) shows a case where Staging can outperform the Inline approach in both performance (Speedup>1) and Cost (Parallel Efficiency>1). The conditions are: (i) no slowdown, i.e., slowdown factor *s*=1; (ii) no additional delay due to data movement to staging: *Tsend*=0; (iii) simulation time between successive output steps is larger than the time required to receive

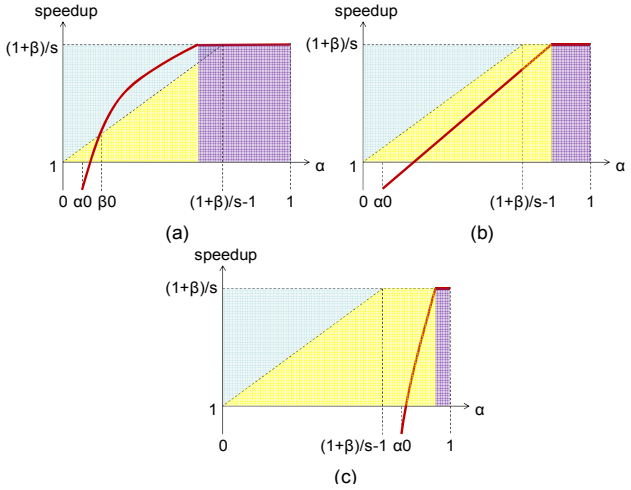

Figure 3. Speedup of Staging vs. Inline.

and analyze data: *Tsim*(*Psim*)>*Trecv*+*Ta*(*Pa*); and (iv) *Ta*(*P*) scales sub-linearly with *P*. Note that if analysis is sub-linear, then when scaling it down to run on some smaller number of nodes, the cost (*Ta*(*P*)×*P*) is reduced. This may create a "sweet-spot" region, shown as [β_0, (1+β)/*s*-1]in Figure 3(a), where α% of additional nodes as staging area can speedup the total execution time by more than α%!

Figure 3(b) shows a case for linear-scalable analysis. As can be seen, if analysis scales linearly, i.e., can be performed locally on compute nodes with no communication, then there is no savings in CPU hours by offloading it to a staging area (since the product *Ta*×*Pa* is constant), but offloading will only introduce additional costs for data movement.

Figure 3(c) demonstrates a case where the minimum size of the staging area (α_0), determined by the memory requirement to accommodate simulation output data plus the analysis code/data, is larger than (1+β)/*s*-1. In this case, the Staging approach with any staging area size $\alpha > \alpha_0$ will always falls into the inefficient region.

2.3 Summary of Performance Modeling

We use the performance model to review previous work in In-situ I/O processing by our group and others and draw the following conclusions.

Firstly, the staging approach can benefit non-scalable analysis actions. An interesting property of staging is that the performance improvement is more evident with less scalable analysis. Our work with GTC and Chimera applications[34][2] evaluated the feasibility of offloading various operations to a separate staging area (e.g., file writing, format conversion, array layout re-organization, histogram calculation, indexing, and sorting), and achieved the "sweet-spot" region shown in Figure 3(a) for both applications at large scale. Section V will provide results with Pixie3D application, which also benefits from placing non-scalable analysis into the staging area.

Secondly, placing linear-scalable analysis in the staging area is less cost-effective than placing it inline, since there will be additional costs for data movement but no reductions in total CPU usage. Data filtering[24], sampling[1], in-situ compression[17]

and visualization (such as slicing, isosurface, and PCA)[12] fall into this category. Those operations can scale to large core counts, extract subsets or features of raw data and hence reduce data volume, and sometimes share large amount of input and/or metadata with simulation, all of which makes it beneficial to place them inline with the simulation.

Thirdly, it is important for the Staging approach to move data in a way such that (i) simulation-side visible data movement latency (*Tsend*) is minimized; (ii) the slowdown factor (s) is minimized; (iii) receiver-side data movement latency (*Trecv*) is reduced to leave sufficient time for analysis to complete before the next I/O action. It is feasible to meet those conditions in practice: (i) by using middleware that provides specializable data copying and marshalling mechanisms to achieve very low simulation-side visible data movement latency (*Tsend*)[1]; (ii) by using contention-aware scheduling for asynchronous data movement to mitigate the slowdown factor (*s*)[3]; (iii) by leveraging a chunk-based processing model to overlap receiver-side data movement latency (*Trecv*) with analysis computation and reduce the memory requirements of the staging area (α_0)[34].

Fourthly, the applicability of staging is constrained by memory availability. At simulation side, extra memory space is needed for asynchronous data movement; the staging area should contain sufficient nodes (α_0) to accommodate the simulation output and all other data and code to run analysis functions. We therefore, promote a chunk-based stream processing model[2]. It is useful for many types of scientific data analytics [15][30], but is limited by the memory constraint to preclude certain time-based analyses that require large data sets across many simulation time steps.

It should be noted that our model assumes a simulation-driven, per-timestep analysis scenario. There are other cases where in-situ I/O processing may also be applied (e.g., computational steering and visual debugging). Supporting those other usage cases at large scale is an active R&D topic, and we are planning to refine our metrics definition and model to cover them in future work.

3. PREDATA MIDDLEWARE

Our previous work on the PreDatA[34] middleware has been extended with support for dynamically deploying computational functions into the simulation and the staging area, as shown in [1]. Although we initially adopted a streaming map-reduce processing model for PreDatA, we have generalized the approach to the "Computational I/O Graph" model. In this more general dataflow model, the computation units, which can be either sequential or parallel programs, communicate through the ADIOS I/O API[20]. Utilizing the componentization of the ADIOS framework, PreDatA allows the data movement to use either direct memory to memory transport[3] or the ADIOS/BP file(s) as the communication channel. The Computational I/O Graph model naturally fits the workflow structure of in-situ I/O processing, eases integrating simulation and analysis codes through use of their common I/O interfaces, and enables data and pipeline parallel execution and parallel data movement.

The PreDatA middleware includes a meta-data service that understand the structure and composition of I/O graphs, provides membership services for connection establishment and teardown, and offers a publish/subscribe mechanism for interacting computation units for exchanging the metadata necessary for MxN parallel data re-distribution [4].

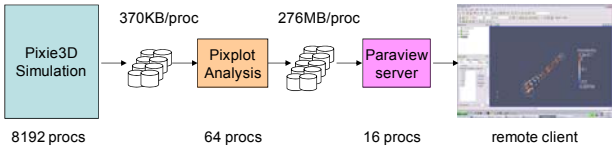

Figure 4. Pixie3D In-Situ I/O Processing.

Placement flexibility is supported at both compile/link time and runtime. For analysis which is known apiori, users can link analysis code either within the simulation program or within the staging area code, or both. Runtime computation placement flexibility is enabled through migratable codelets implemented in C-o-D. As detailed in [1], such codelets can be compiled and deployed into a program's address space at runtime in an efficient and secure manner. This feature is particularly useful for runtime flexible placement. Two common usage scenarios are: (i) offloading non-scalable analytics to downstream staging area, and (ii) uploading filters to upstream simulation nodes.

4. APPLICATION CASE STUDY

Experimental results concerning in-situ I/O processing are obtained for the Pixie3D[7] application. Pixie3D is a 3-Dimensional extended MHD code which solves the extended MHD equations in 3D arbitrary geometries using fully implicit Newton-Krylov algorithms. As illustrated in Figure 4, Pixie3D I/O processing uses a computational I/O graph structured as a three-stage pipeline. The first stage is the Pixie3D simulation, generating output data that consists of eight 3D arrays that represent mass density, linear momentum components, vector potential components, and temperature, respectively. The second stage is an analysis code called "Pixplot", which performs various diagnostic routines on Pixie3D output data to generate derived quantities, such as curl, gradient, flux, and divergence. The third stage uses the Paraview visualization tool to read the derived quantities generated by Pixplot for visual data exploration.

We run Pixie3D and Pixplot on ORNL's Jaguar Cray XT5 and use the performance model in Section II to reason about the best placement strategy for Pixie3D I/O processing pipeline. We first find that the Inline approach will perform poorly for the Pixie3D case at large scale. Figure 5 shows the weak scaling of time for Pixie3D simulation (*Tsim*) and Pixplot analysis (*Ta*). The time to write the output of Pixplot from simulation nodes is also shown in the figure. As seen, both Pixplot and I/O have worse scalability than Pixie3D. If Pixplot analysis is performed inline, the time portion spent in Pixplot analysis will increase from 0.06% of Pixie3D simulation time on 512 cores, to 35.6% on 8192 cores. Also, the time to write analysis results increases from 0.39% of Pixie3D simulation time on 512 cores to 3.8% on 8192 cores. The insufficient scalability of Pixplot is due to its intensive use of MPI collective communications and "aggregating-to-one-process" style computation which is not uncommon in analysis codes.

We then run Pixie3D simulation on 8192 cores with Pixplot placed in a staging area of 64 cores. Data movement between Pixie3D and Pixplot is via DataTap, a RDMA-based contention-aware transport[3]. The detailed timing in Figure 6 shows that placing Pixplot in staging area can effectively overlap analysis and writing (*Ta/compute* and *Ta/write*, respectively) with simulation (*Tsim*). The simulation-side visible send time (*Tsend*) is 1.3% of *Tsim*, and the slowdown factor *s* of *Tsim* due to staging is 1.008. The staging area side data movement time (*Trecv*) is less

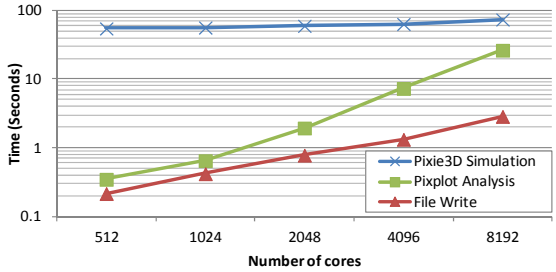

Figure 5. Pixie3D In-Situ I/O Processing.

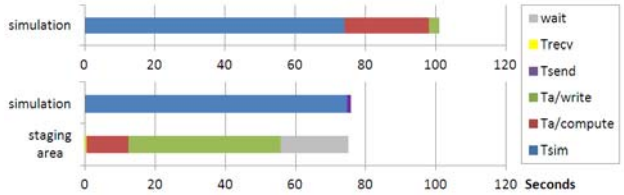

Figure 6. Timeline of Inline and Staging approaches, shown in upper and lower charts, respectively. In both cases Pixie3D runs on 8192 cores.

than 0.5 seconds. Overall, with a staging area which is of 0.78% the simulation nodes (α=*Pa/Psim*=0.78%), the measured speedup of Staging over Inline is 1.333, which is within 96% of the upper bound ((1+β)/s=(1+35.6%+3.8%)/1.008=1.383) given by the performance model. When running Pixie3D on 8192 cores, the minimal size of staging area (α_0), determined by the minimal memory space required to run Pixplot, is 64 cores (meaning α_0=0.78%). Figure 6 shows that even when staging area is of the minimal feasible size (α_0), the staging area still spends 25% of time waiting for output data from simulation, indicating that staging area is already over-provisioned. Also note that scaling down Pixplot from 8192 to 64 cores actually reduces its runtime by half. This is due to the communication-bound nature of Pixplot and improved locality when data are aggregated onto a smaller number of cores.

When compared to the Offline approach, by which Pixie3D writes output into a BP file using MPI-IO and Pixplot reads data from the file for analysis, the Staging approach hides file write latencies almost completely and improves Pixie3D's total execution time by 2.3% to 16.5% among 5 test runs at scale of 8192 cores. This validates the applicability of in-situ I/O processing to mitigate the I/O bottleneck on high end machines.

5. CONCLUSIONS AND FUTURE WORK

This paper discusses the significance of flexible placement for in-situ I/O processing. A simple performance model is used to quantitatively describe the trade-offs in placement. Based on the model, we compare two different placement strategies for in-situ processing and reveal their relative strengths and limitations. Performance results with a large-scale scientific application empirically verify our performance model and the argument for flexible placement.

Our future work is twofold: (1) we are working on runtime resource management and pipeline balancing for I/O graphs and (2) we are investigating automatic placement of analysis computations based on runtime performance monitoring[29].

6. ACKNOWLEDGMENTS

The authors thank Berk Geveci, Sebastien Jourdain, and Pat Marion from Kitware Inc. and Kenneth Moreland from Sandia National Laboratory for integrating ADIOS with ParaView and aid in implementing Pixie3D I/O processing pipeline. This work was funded in part by Sandia National Laboratories under contract DE-AC04-94AL85000, by the DOE Office of Science, Advanced Scientific Computing Research, under award number DE-SC0005505, program manager Lucy Nowell, and by the Department of Energy under Contract No. DEAC05- 00OR22725 at Oak Ridge National Laboratory. Additional support came from the resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, a grant from NSF as part of the HECURA program, a grant from the Department of Defense, a grant from the Office of Science through the SciDAC program, and the SDM center in the ASCR office.

7. 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 HPDC, 2011.

[2] H. Abbasi, J. Lofstead, F. Zheng, K. Schwan, M. Wolf, S. Klasky. Extending I/O through high performance data services. In Proc. of Cluster 2009.

[3] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan. And F. Zheng. DataStager: Scalable Data Staging Services for Petascale Applications. in Proceedings of HPDC 2009.

[4] H. Abbasi, M. Wolf, K. Schwan, G. Eisenhauer, and A. Hilton. Xchange: coupling parallel applications in a dynamic environment. in CLUSTER, 2004, pp. 471-480.

[5] A. Ailamaki, V. Kantere, and D. Dash. Managing scientific data. Commun. ACM, 2010.

[6] 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 EGPGV, 2011, pp. 91-100.

[7] L. Chacón. A non-staggered, conservative, $\nabla\cdot\rightarrow\beta$= 0, finite-volume scheme for 3D implicit extended magnetohydrodynamics in curvilinear geometries. Computer Physics Communications, 163:143.171, Nov. 2004.

[8] H. Childs. Architectural challenges and solutions for petascale postprocessing. J. Phys.: Conf. Ser., vol. 78, 2007.

[9] H. Childs, D. Pugmire, S. Ahern, B. Whitlock, M. Howison, Prabhat, G. H. Weber, and E. W. Bethel. Extreme scaling of production visualization software on diverse architectures. IEEE Computer Graphics and Applications, vol. 30, no. 3, pp. 22-31, 2010.

[10] C. Docan, M. Parashar, J. Cummings, and S. Klasky. Moving the code to the data - dynamic code deployment using activespaces. in IPDPS, 2011, pp. 758-769.

[11] C. Docan, M. Parashar, and S. Klasky. Dataspaces: an interaction and coordination framework for coupled simulation workflows. in HPDC 2010.

[12] N. Fabian, K. Moreland, D. Thompson, A. Bauer, et. al. The ParaView Coprocessing Library: A scalable, general purpose in situ visualization library. in LDAV 2011.

[13] W.-C. Feng and T. Scogland. The Green500 List: Year One. in 5th IEEE Workshop on High-Performance, Power-Aware Computing (in conjunction with IPDPS 2009), May 2009.

[14] A. Gerndt, B. Hentschel, M. Wolter, T. Kuhlen, and C. Bischof. Viracocha: An efficient parallelization framework for large-scale cfd post-processing in virtual environments. SC 2004.

[15] C. Herath and B. Plale. Streamflow -Programming Model for Data Streaming in Scientific Workflows. in CCGrid 2010.

[16] M. Hereld, M. E. Papka, V. Vishwanath. Toward Simulation-Time Data Analysis and I/O Acceleration on Leadership-Class Systems. Preprint ANL/MCS-P1929-0911, September 2011.

[17] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data. in Euro-Par, 2011, pp. 366-379.

[18] M. Li, S. S. Vazhkudai, A. R. Butt, F. Meng, X. Ma, Y. Kim, C. Engelmann, and G. Shipman. Functional partitioning to optimize end-to-end performance on many-core architectures. SC10.

[19] J. F. Lofstead, M. Polte, G. A. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu. Six degrees of scientific data: reading patterns for extreme scale science io. in HPDC2011.

[20] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata rich io methods for portable high performance io. In Proceedings of IPDPS'09, May 25-29, Rome, Italy, 2009.

[21] J. F. 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 SC 2010.

[22] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, et. al. Scientific workflow management and the kepler system. CCPE, vol. 18, no. 10, pp. 1039-1065, 2006.

[23] R. A. Oldfield, S. Arunagiri, P. J. Teller, S. Seelam, M. R. Varela, R. Riesen, and P. C. Roth. Modeling the impact of checkpoints on next-generation systems. in MSST '2007.

[24] R. Oldfield, D. Kotz. Armada: a parallel I/O framework for computational grids. Future Generation Comp. Syst. 18(4): 501-523, 2002.

[25] J. Piernas, J. Nieplocha, and E. J. Felix. Evaluation of active storage strategies for the lustre parallel file system. in SC2007.

[26] J. Shalf, S. S. Dosanjh, and J. Morrison. Exascale computing technology challenges. in VECPAR, 2010, pp. 1-25.

[27] A. Singh, P. Balaji, and W.-c. Feng. Gepsea: A general-purpose software acceleration framework for lightweight task offloading. in Proc. of ICPP'09, 2009, pp. 261-268.

[28] S. W. Son, S. Lang, P. Carns, R. Ross, R. Thakur, B. Ozisikyilmaz, P. Kumar, W.-K. Liao, and A. Choudhary. Enabling active storage on parallel i/o software stacks. in MSST'10.

[29] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, et. al. A flexible architecture integrating monitoring and analytics for managing large-scale data centers. in Proceedings of ICAC 2011.

[30] K. R. Wheeler, M. Allan, C. Curry. Lessons Learned From Developing A Streaming Data Framework for Scientific Analysis. http://ti.arc.nasa.gov, Septermber 2011.

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

[32] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma. In situ visualization for large-scale combustion simulations, IEEE Computer Graphics and Applications, 2010.

[33] F. Zhang, C. Docan, M. Parashar, and S. Klasky. Enabling multiphysics coupled simulations within the pgas programming framework. in CCGRID, 2011, pp. 84-93.

[34] F. Zheng, H. Abbasi, C. Docan, J. F. Lofstead, et. al. Predata -preparatory data analytics on peta-scale machines. in IPDPS 2010.