Linking Performance Data into Scientific Visualization Tools

Kevin A. Huck [*] Kristin Potter [*] Doug W. Jacobsen [†]
Hank Childs [*,▽] Allen D. Malony [*]
[*] University of Oregon {khuck,kpotter,hank,malony}@cs.uoregon.edu
[†] Los Alamos National Laboratory, douglasj@lanl.gov
[▽] Lawrence Berkeley National Laboratory, hchilds@lbl.gov

Abstract— **Note : The abstract.**

I. Introduction

Performance data refers to the data collected to describe program execution. It includes quantities such as performance counters, time elapsed in subroutines, etc. Significant analysis of this performance data is commonplace when running simulations on supercomputers, as the gains from such analysis can lead to significant improvements in execution time, and thus increases in the capacity of the supercomputer itself.

Performance data can be hard to interpret, and significant effort has gone into finding insightful ways to present it. Visualization is an obvious approach, since the human visual system is able to absorb information at very high rates. The visualization field is frequently segmented into two sub-disciplines — scientific visualization and information visualization — with visualization of performance data often utilizing information visualization techniques. The primary distinction between the two sub-disciplines is that scientific data normally has an implied spatial layout, while information visualization data does not. With scientific visualization, techniques can exploit the spatial properties of the information (meshes, grids, vectors, tensors, etc.) and utilize the three-dimensional capabilities of todays graphics engines to help visually present their analysis. The types of data associated with scientific visualization are often "scientific" in nature: engineering, climate, and medical data, etc. With information visualization, the data has no accompanying spatial information, and so the resulting visualizations have more flexibility in how to arrange data. The types of data associated with information visualization are diverse: tax records, twitter feed information, network usage statistics, etc. For both scientific visualization and information visualization, however, the goal is to enable insight into data.

With this work, we explore the topic of scientific visualization combined with performance data. The intuition behind the work is that a strictly information visualization-based approach can sometimes fail to convey important information about context. By complementing traditional approaches with scientific visualization techniques that show performance data in the spatial context of the simulation, additional insights can be gained that are difficult to infer otherwise.

The contribution of this paper is two-fold:

- to add to the evidence that scientific visualization can be helpful in understanding performance data, and
- to explore the costs and approaches of linking performance data with scientific visualization tools.

Regarding the linking, we are motivated by the costs involved with building a system. The cost to extract and categorize performance data is high, as is the cost to implement scientific visualization systems. Therefore, we view an approach that can leverage existing systems as paramount. Further, our findings demonstrate that linking these systems together can be done in a straight-forward way, requiring only modest effort and yielding significant benefit.

II. A Historical Perspective

There has been interest in the use of graphical and visualization techniques to understand parallel performance data since scalable parallel systems began to appear [1], [2]. Many of the motivations for performance visualization then – increasing performance data size, more complex performance phenomena, systems scalability, power of visual analysis – are the same ones that drive the research today. The early work in the field pioneered approaches with respect to both effective presentation of performance information [3], [4], [5] as well as how to develop tools incorporating visualization capabilities [6], [7], [8]. While the challenges facing parallel performance analysis now are arguably more daunting than 20+ years ago, there are useful concepts that carry forward to help inform the research today.

A conceptual design architecture for a parallel performance visualization system is shown in Figure 1 [9]. The main idea is that performance visualizations are a product of two key components. The first component is focused on defining the performance *analysis abstractions* that are of interest in being portrayed using performance *visual abstractions* where the association between the two are linked by *semantic* relationships. These abstractions are then translated into specifications for performance *views* and *displays* and the mappings between the two. The second component is the instantiation of the joint specification into a working visualization tool that leverages alternative technologies where available. The separation of

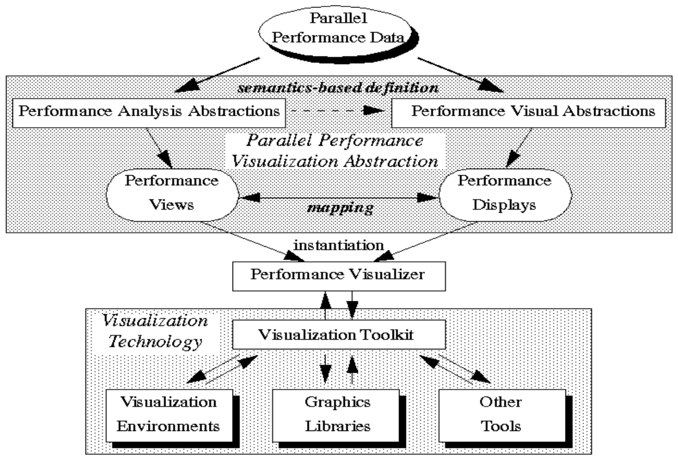

Parallel Performance Data
semantics-based definition
Performance Analysis Abstractions
Performance Visual Abstractions
Parallel Performance Visualization Abstraction
Performance Views
mapping
Performance Displays
instantiation
Performance Visualizer
Visualization Technology
Visualization Toolkit
Visualization Environments
Graphics Libraries
Other Tools

Fig. 1. Design architecture for parallel performance visualization systems.

concerns between the two components is important for evolving the performance visualization methods and improving the development of visualization techniques in real tools.

Good performance visualization is a design process. The goal is to associate certain properties of the performance analysis (as understood by the user) with graphical representations that have both good visual form and effective presentation. General visualization principles and graphics techniqes can help, but it does not directly solve the performance view creation problem. Early research experience was a product of experimenting with different parallel performance analysis and visualization abstractions and then attempting to extract constructive guidelines for good performance visualization development [10]. Although they may seem intuitive, it is instructive to consider them in the context of current requirements:

Multiple views. A visualization should provide different levels of observation and different perspectives (a set of views). Multiple dimensions of performance information may require a variety of visual perspectives.

Semantic context. The intent of visualization is to improve performance data understanding. Semantic context (e.g., program control, data abstractions, programming model, system mapping, runtime operation, computational behavior) can be important in visualizations with respect to how the structure, graphics, interactions, correlation to other displays, and visual features are interpreted.

User interaction. Involving the user in selection, control of detail, changing of parameters, and other interaction can be important to explore performance aspects in a visualization.

In addition to guidelines for good visualization design, there were recommendations for developing scalable visualization methods that resulted from former research projects. These included [4], [8], [10]:

Adaptive graphical representation. Present the performance to reveal detail, but prevent visual complexity from interfering with the perception of that detail.

Reduction and filtering. Use a summarized version of the raw data to show less detailed information as a way to reduce the complexity of the visualization at the level of graphical representation.

Spatial organization. Arrange graphical elements spatially so that as a dataset scales, the display size and/or complexity increase at a much slower rate, using other graphical techniques (e.g., coloring, annotation) to portray performance data characteristics.

Generalized scrolling. Present a continuous, localized view of a much larger mass of information to allow viewing of greater, while maintaining continuity (temporal and spatial) with nearby performance information.

Prior work also raised issues of building general visualization solutions versus the usability of any particular performance visualization outcome. Miller addressed some of these in his essay “What to Draw? When to Draw? An Essay on Parallel Program Visualization” where he attempted to define general requirements that will help guide the visualization designer to create effective visual displays [5].

III. Background

A. VisIt

Hank: Text about VisIt.

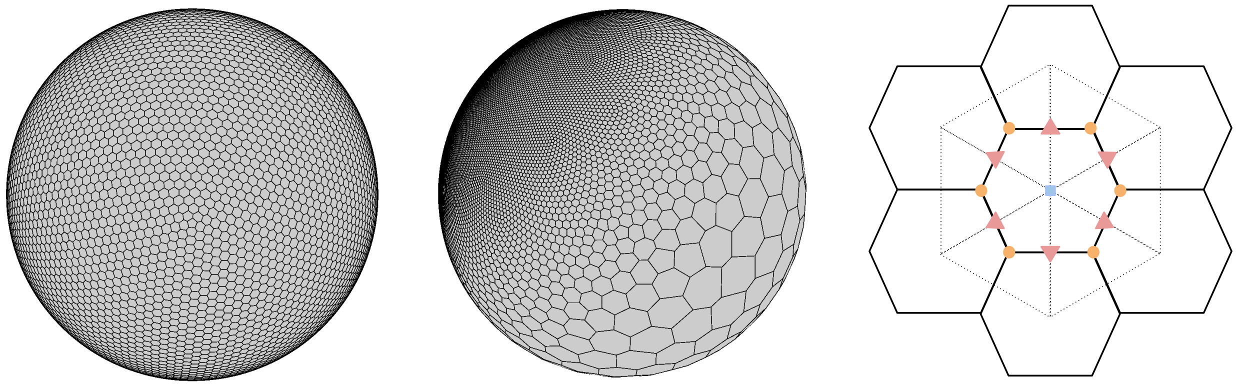
B. TAU

The performance tool used for this approach is the TAU Performance System® [11]. TAU is a portable profiling and tracing toolkit for performance analysis of parallel programs written in Fortran, C, C++ , Python and Java, running on a variety of shared and distributed memory parallel architectures including message passing libraries (i.e. MPI) and Partitioned Global Address Spaces (PGAS, co-arrays). TAU provides different means for instrumentation (source, library, binary) and is capable of gathering performance information (time, hardware counters) about functions, methods, basic blocks and statements as well as periodic sampling and metadata collection. TAU is distributed with profile and trace analysis tools, a performance database, and a multi-experiment data mining package called PerfExplorer [12]. PerfExplorer includes a Python interpreter for automating analysis, custom post-processing of data and other transformations including exporting data from the profile. Kevin: need more text about TAU and PerfExplorer?

C. MPAS-Ocean

The Model for Prediction Across Scales (MPAS) [13] is a framework project jointly developed by the National Center for Atmospheric Research (NCAR) and Los Alamos National Lab (LANL) in the United States. The framework is designed to perform rapid prototyping of single-component climate system models. Several models have been developed using the MPAS framework. MPAS-Ocean [14] is designed to simulate the ocean system for a wide range of time scales and spatial scales from less than 1 km to global circulations.

The MPAS framework utilizes unstructured meshes that can be created in a variety of ways but are typically Spherical



(a) Regular SCVT example. (b) Variable resolution SCVT example. (c) MPAS data layout.

Fig. 2. Spherical Centroidal Voronoi Tesselations.

Centroidal Voronoi Tessellations (SCVTs). The mesh cells are a combination of mostly hexagonal cells with a few pentagonal cells that decompose the data in 2 dimensions, as shown in Figures 2(a) and 2(b). Figure 2(c) shows the two cell meshes with overlapping spatial layouts: the *Primary Mesh* (Voronoi, hexagons), and the *Dual Mesh* (Delaunay, triangles). Within each cell region, edge and vertex lie *Scalar Quantities* (Blue Squares), *Vector Quantities* (Red Triangles) and *Vorticity Quantities* (Orange Circles). Each cell has regularly structured data, such as temperature, salinity, velocity and vorticity. After the mesh is partitioned, a typical block has 40 vertical levels, 200–500 cells, 400–1000 verticies and 600–1500 edges explictly assigned to the block. During simulation initialization, Each block is also assigned a halo region that typically consists of 3 layers of cells from neighboring blocks. A vast majority of the communication in MPAS consists of the halo exchanges between neighboring blocks during each phase of each timestep.

MPAS-Ocean is developed in Fortran using MPI for large scale parallelism. The application had previously been instrumented with internal timing functions. TAU was integrated by supplementing the timing infrastructure with TAU timers. The application instrumentation was mostly at a high level, encapsulating key phases in the simulation model. Linking with the TAU library also provides measurement of MPI communication through the PMPI interface and hardware counter data using PAPI.

D. MPAS-Ocean Load Imbalance

Kevin: Explain partitioning problem - the load balancing issue, the 3 layer halos. Show view of poorly balanced 60km example with 256 processes on Edison, both with ParaProf 3D views. (and with VisIt?)

The performance problem that we are addressing with this approach is that of load balancing and optimization due to the elimination of redundant computation. For example, a 256 process run was executed on Edison, an Intel-based Cray ??? system at NERSC. Kevin: list Edison's hardware properties here The simulation ran for about 5 minutes, executing 1080

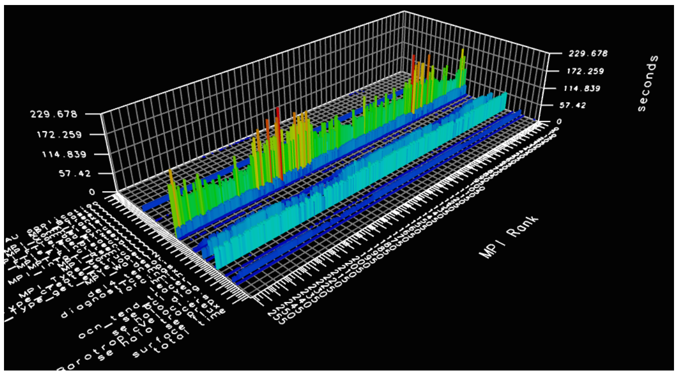
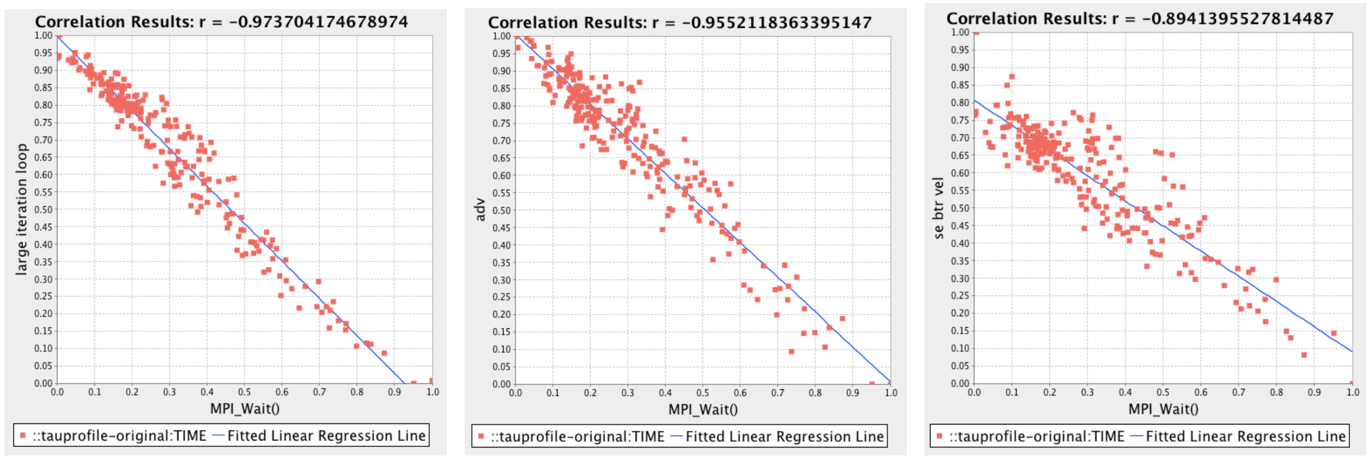


Fig. 3. 256 process performance profile of MPAS-ocean using K-way data partitioning, element decomposition and the RK4 solver on Edison. The height and color of the bars represents the time spent in each timed code region, the highest of which is `MPI_Wait()`.

timesteps to simulate 30 days. A TAU profile was collected, and the result is shown in Figure 3. The center ridge of counters is the amount of time spent in `MPI_Wait()`, which dominates the execution time. `MPI_Wait()` is used after asynchronous sends and receives between two processes to ensure the communication has completed. Later experiments showed that the number of neighbors each process communicates with is correlated with the number of calls to `MPI_Wait()`, but the cumulative time spent in `MPI_Wait()` is not correlated with the number of calls. In addition, as shown in Figure 5, the time in `MPI_Wait()` is negatively correlated with main computation routines. These observations all indicate that there is a load balance issue in the code. This result seemed counterintuitive, as the domain was statically decomposed using the k-way stragegy in gpetis [15] which reported an imbalance of less than 1%. Kevin: validate this percentage

Additional metadata was collected to compute exactly how much work each process was assigned to compute, including the number of cells explicitly assigned to each block of the partition (`nCellsSolve`), the total cells including halo cells (`nCells`), the total levels computed per block



(a) Correlation with `large iteration loop`. (b) Correlation with `adv`. (c) Correlation with `se btr vel`.

Fig. 5. Correlation of time spent in `MPI_Wait()` with computation regions.

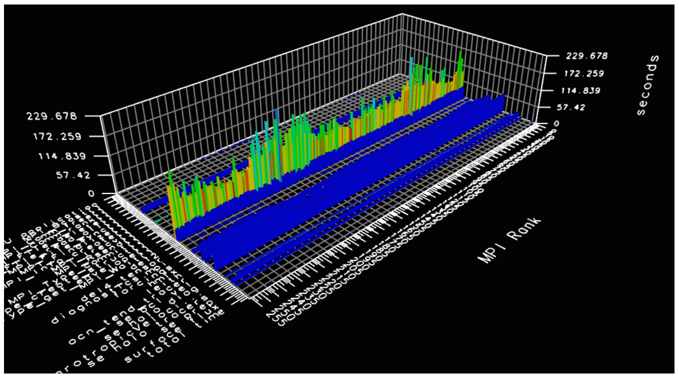


Fig. 4. 256 process performance profile of MPAS-ocean using K-way data partitioning, element decomposition and the RK4 solver on Edison. The height of the bars represents the time spent in each timed code region, the highest of which is `MPI_Wait()`. The colors represent the number of times each region was executed. `MPI_Wait()` is the event with the highest variability.

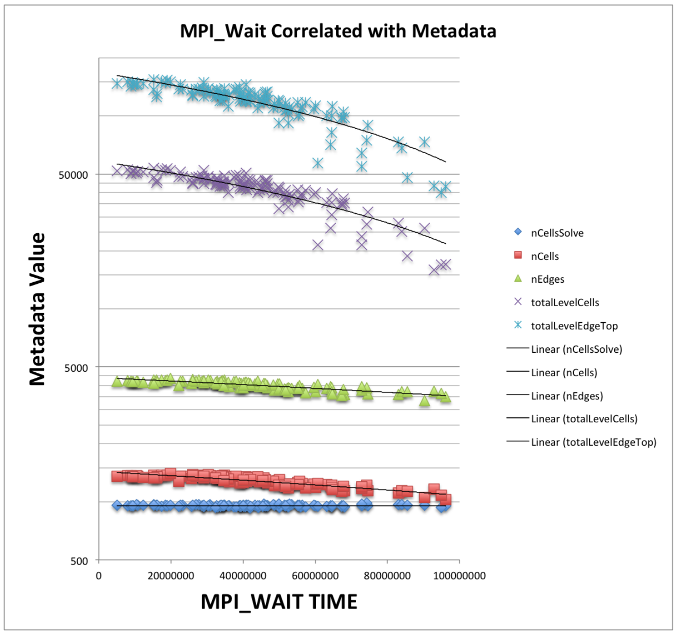


Fig. 6. Correlation of `MPI_Wait()` time with metadata values.

(`totalLevelCells`, `totalLevelEdgeTop`), the number of edges on each cell in the block (`nEdges`). The metadata values were correlated with the performance from each process, and the results are shown in Figure 6. While it appears that the `totalLevelEdgeTop`, `totalLevelCells` and even `nEdges` are highly correlated, they are all dependent on the truly correlated value, `nCells`. In fact, it is also obvious that the number of cells explicitly assigned to each block (`nCellsSolve`) is well balanced and uncorrelated with performance at all. To understand the difference between `nCells` and `nCellsSolve` requires a deeper understanding of the *halo* or *ghost* cells associated with each block of the partition.

As described earlier, each block in the partition requires halo cells from neighboring blocks in order to compute all phases of computation within a timestep with meaningful values, a common problem in all stencil decompositions. This problem is made worse because the computation performed with monotonic advection requires three layers of halo cells from all neighboring blocks. Each process is required to compute all of the cells explicitly assigned to the block (`nCellsSolve`), as well as the *implicitly* assigned halo cells, yeilding the `nCells` value. As a further complication, blocks that are less spherically shaped can have significantly more halo cells than spherical blocks with the same number of cells.

Different operators available within MPAS require different numbers of halos. For example, the monotonic advection is the only one that requires 3, but because that method is the default, all of the simulation options have a default number of halos of 3. If you have a del2 operation you need 1 halo, and if you have a del4 operation you need 2 halos. Other operators might require more or less halos, depending on their stencil. Almost all of the MPAS loops compute over all cells within a block, including the 3 layers of halo cells. When performing the barotropic solves within the *split_explicit* time stepper a 2D system is solved, and halo updates are performed each iteration to make the solver is not using "corrupted" data.

The number of cells in the halo regions for each block can vary significantly from one block to the next. This varation in the number of halo cells is correlated with the amount of time spent in computation for each block, and negatively correlated with the time spent in `MPI_Wait()`. Unfortunately, the number of cells in the halo region is a partitioning parameter that is not known until after the partitioning is complete, and to our knowledge there are no partitioners that are capable of generating a balanced partition weighted by an unknown value. The profiles also suggest that the variability in the depth of cells (and therefore amount of data to compute per cell) as well as the number of neighbors for each block also contribute to the load imbalance, but the most significant property is the halo region size. We also theorized that a majority of the underutilized processes were assigned either a coastal or poorly connected block with a considerably smaller halo region that does not fully enclose the block. However, at that time we did not have a method for visualizing the metadata nor performance properties associated with each block.

IV. Approach

Kevin : Discuss how performance data, metadata are collected by TAU at runtime.

As mentioned in Section III-D, the MPAS application is instrumented with application timers that are mapped to TAU timers, and MPI timers are collected by TAU using the standard PMPI interface. In addition, application properties that potentially have an effect on performance are also collected as metadata values using manual instrumentation. Some such properties include those discussed in Section III-D as well as other partition related variables such as how many neighbors a process has to communicate with in the mesh.

Kevin : Discuss PerfExplorer script to post-process the TAU profiles and export properties for each partition block.

The TAU profiles are then loaded into the TAUdb database [**?**], and post-processed by a PerfExplorer script to extract and export the performance measurements and metadata values per process, which maps directly to the block assigned to the process.

Hank : Discuss MPAS file format, and how NetCDF data is parsed and interpreted by VisIt.

Kevin : Discuss the hindsight partitioner?

V. Experiments

Kevin : revisit partitioning problem - the load balancing issue, the 3 layer halos. Show view of poorly balanced 60km example with 256 processes on Edison, both with ParaProf 3D views and with VisIt. Explain how VisIt view helps us understand the true nature of the imbalance.

Kevin : Briefly describe the hindsight partitioning approach. Show view of better balancing, shorter computation, shorter communication. Show with ParaProf and VisIt.

Kevin : Briefly describe the ghost partitioning approach. Show view of better balancing, but longer computation, more communication. Show view of per-block neighbor counts for each block, and number of calls to MPI_Wait(). Briefly explain ghost and hindsight approach, show results.

VI. Related

ParaProf 3D views [16].

BoxFish [17], [18].

Cube 3D [19].

Kevin : Other examples of performance data mapped to a domain? There are obvious ones, like all the usual performance data visualizations, but are they relevant?

Kevin : Other examples of visualization tools like VisIt?

VII. Discussion

Note : Rewrite this to make sense in the context of the paper!

During the SUPER engagement with the MULTISCALE project to help optimize the MPAS-Ocean application, the performance measurement and analysis process revealed interesting properties related to how the irregular mesh application data was decomposed prior to execution. The domain decomposition, previously considered to be balanced, was used in ways that lead to a workload imbalance due to the significant number of halo (ghost) cells shared between partitions.

SUPER team members from the University of Oregon and SDAV team members are collaborating to integrate the visualization of performance data and metadata within the application domain. Visualizing TAU performance measurements in the context of the application domain has already led to a better understanding of the partitioning challenges, and why attempted optimizations were or were not successful. The information gained will assist in developing new partitioning strategies where the block properties are unknown until after partitioning is complete. These visualizations could also be helpful in placing the application processes on the allocated hardware to improve communication locality.

VIII. Conclusion

Kevin : wrap it up

IX. Acknowledgements

This work is supported by DOE SciDAC grant DE-SC0006723. Allen : should we call out SUPER and SDAV explicitly, to make the partnering clear? Doug Jacobsen was supported by the US DOE Office of Science, Biological and Environmental Research program. Hank : Funding sources? Kristin : Funding sources?

Note : Make sure the missing references are resolved!

References

[1] A. Malony and D. Reed, "Visualizing parallel computer system performance," in *Instrumentation for Future Parallel Computer Systems*, M. Simmons, R. Koskela, and I. Bucher, Eds. ACM Press, 1989, pp. 59–90.

[2] A. Couch, "Problems of scale in displaying performance data for loosely coupled multiprocessors," in *Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, March 1989.

[3] M. Heath and J. Etheridge, "Visualizing the performance of parallel programs," pp. 29–39, September 1991.

[4] A. Couch, "Categories and context in scalable execution visualization," vol. 18, pp. 195–204, June-July 1993.

[5] B. Miller, "What to draw? when to draw? an essay on parallel program visualization," vol. 18, no. 1, pp. 265–269, June 1993.

[6] D. J. A. Malony, D. Hammerslag, "Traceview: A trace visualization tool," pp. 29–38, September 1991.

[7] A. Couch, "Massively parallel performance analysis," vol. 81, no. 8, pp. 1116–1125, August 1993.

[8] B. M. S. Hackstadt, A. Malony, "Scalable performance visualization for data-parallel programs," May 1994.

[9] D. R. M. Heath, A. Malony, "Parallel performance visualization: From practice to theory," vol. 3, no. 4, pp. 44–60, Winter 1995.

[10] ——, "The visual display of parallel performance data," vol. 28, no. 11, pp. 21–28, November 1995.

[11] S. Shende and A. D. Malony, "The TAU Parallel Performance System," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, Summer 2006.

[12] K. A. Huck, A. D. Malony, S. Shende, and A. Morris, "Knowledge support and automation for performance analysis with PerfExplorer 2.0," *Scientific Programming, special issue on Large-Scale Programming Tools and Environments*, vol. 16, no. 2-3, pp. 123–134, 2008, http://dx.doi.org/10.3233/SPR-2008-0254.

[13] LANL and NCAR, "MPAS," http://mpas-dev.github.io, 2014.

[14] T. Ringler, M. Petersen, R. L. Higdon, D. Jacobsen, P. W. Jones, and M. Maltrud, "A multi-resolution approach to global ocean modeling," *Ocean Modelling*, vol. 69, no. 0, pp. 211 – 232, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1463500313000760

[15]

[16] W. Spear, A. Malony, C. Lee, S. Biersdorff, and S. Shende, "An approach to creating performance visualizations in a parallel profile analysis tool," in *Euro-Par 2011: Parallel Processing Workshops*, ser. Lecture Notes in Computer Science, M. Alexander, P. DAmbra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. Di Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. Scott, J. Traff, G. Valle, and J. Weidendorfer, Eds. Springer Berlin Heidelberg, 2012, vol. 7156, pp. 156–165. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29740-3_19

[17] K. Isaacs, A. Landge, T. Gamblin, P.-T. Bremer, V. Pascucci, and B. Hamann, "Abstract: Exploring performance data with boxfish," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, Nov 2012, pp. 1380–1381.

[18] M. Schulz, A. Bhatele, P.-T. Bremer, T. Gamblin, K. Isaacs, J. Levine, and V. Pascucci, "Creating a tool set for optimizing topology-aware node mappings," in *Tools for High Performance Computing 2011*, H. Brunst, M. S. Mller, W. E. Nagel, and M. M. Resch, Eds. Springer Berlin Heidelberg, 2012, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31476-6_1

[19] D. Lorenz, D. Bhme, B. Mohr, A. Strube, and Z. Szebenyi, "Extending scalascas analysis features," in *Tools for High Performance Computing 2012*, A. Cheptsov, S. Brinkmann, J. Gracia, M. M. Resch, and W. E. Nagel, Eds. Springer Berlin Heidelberg, 2013, pp. 115–126. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37349-7_8