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*—Understanding the performance of program execution is essential when optimizing simulations run on high-performance supercomputers. Instrumenting and profiling codes is itself a difficult task and interpreting the resulting complex data is often facilitated through visualization of the gathered measures. However, these measures typically ignore spatial information specific to a simulation, which may contain useful knowledge on program behavior. Linking the instrumentation data to the visualization of performance within a spatial context is not straightforward as information needed to create the visualizations is not, by default, included in data collection, and the typical visualization approaches do not address spatial concerns. In this work, we present an approach that links the collection of spatially-aware performance data to a visualization paradigm through both analysis and visualization abstractions to facilitate better understanding of performance in the spatial context of the simulation. Because the potential costs for such a system are quite high, we leverage existing performance profiling and visualization systems and demonstrate their combined potential on climate simulation.**

I. Introduction

Performance data measured from program execution reflects information about where time elapsed relative to the code, how the processor and memory were utilized, and what communications behavior resulted. 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.

However, performance data can be hard to interpret, and great 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, such as with engineering, climate, and medical data. 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, such as tax records, twitter feed information, and network usage statistics. Both scientific visualization and information visualization have as their goal the enabling of insight into the 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

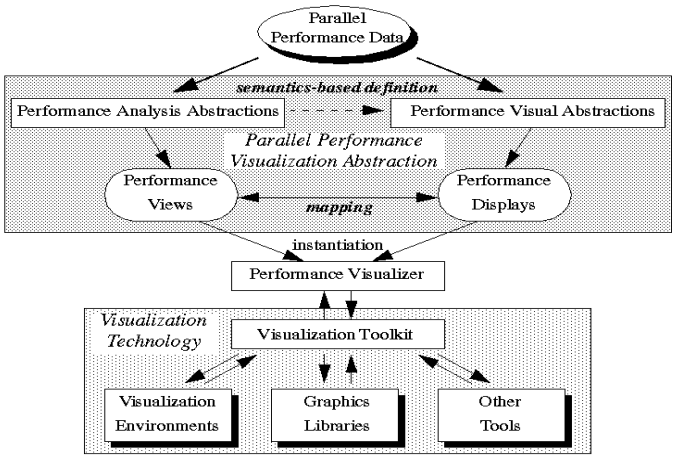


Fig. 1. Design architecture for parallel performance visualization systems.

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 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 [4], [8], [10]. These included:

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 a greater context, 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].

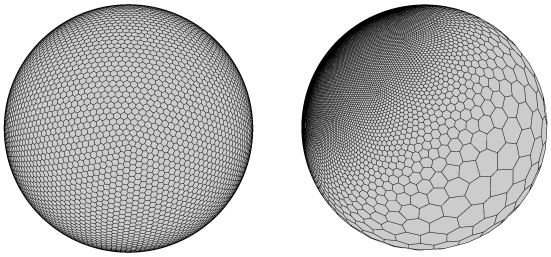
III. Background

A. VisIt

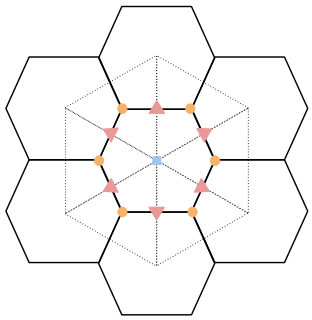
The VisIt open-source tool [11] was used to generate the visualizations in this work. VisIt is an open-source visualization, animation, and analysis toolkit that provides an interactive user interface and support for over 120 different scientific data formats. The customizable plugin design allows for user-developed modifications to accomodate a large variety of scientific visualization data sets and display methods. Many traditional visualization techniques are supported including methods for 2D data (pseudo-color, scatterplots, etc), 3D data (volume rendering, isocontours, etc), and vector data (streamlines, glyphs, etc). VisIt runs on a variety of platforms and can be configured to post-process in parallel and scale from desktop machines to large-scale high performance computers.

B. TAU

The performance tool used for this approach is the TAU Performance System® [12]. TAU is a portable profiling and tracing toolkit for performance analysis of parallel programs



(a) Regular SCVT. (b) Variable resolution SCVT.



(c) MPAS data layout.

Fig. 2. Spherical Centroidal Voronoi Tesselations.

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 [13]. PerfExplorer includes a Python interpreter for automating analysis, custom post-processing of data and other transformations including exporting data from the profile.

C. MPAS-Ocean

The Model for Prediction Across Scales (MPAS) [14] 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 allow the rapid prototyping of single-component climate system models. Several models have been developed using the MPAS framework. MPAS-Ocean [15] 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 Centroidal Voronoi Tessellations (SCVTs). The mesh cells are made up of arbitrary polygons, though the majority tend to be hexagons. The cells 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 vertices and 600–1500 edges explicitly 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

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, a Cray XC30 "Cascade" system at NERSC. Each Edison has two sockets, each populated with a 12-core Intel "Ivy Bridge" processor, providing 24 cores per node. The allocation was for 11 nodes, placing 24 processes per node, with the last node not fully utilized. The simulation ran for about 5 minutes, executing 1080 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

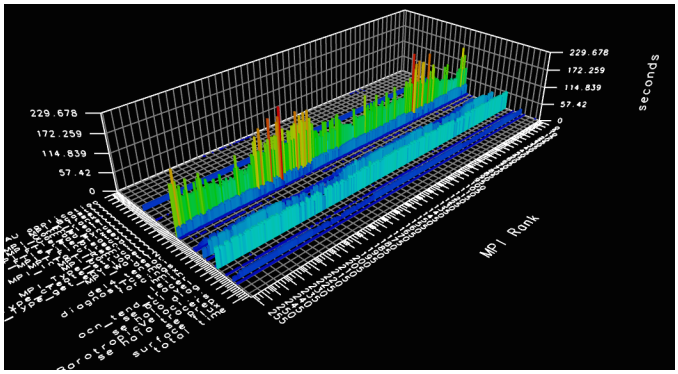
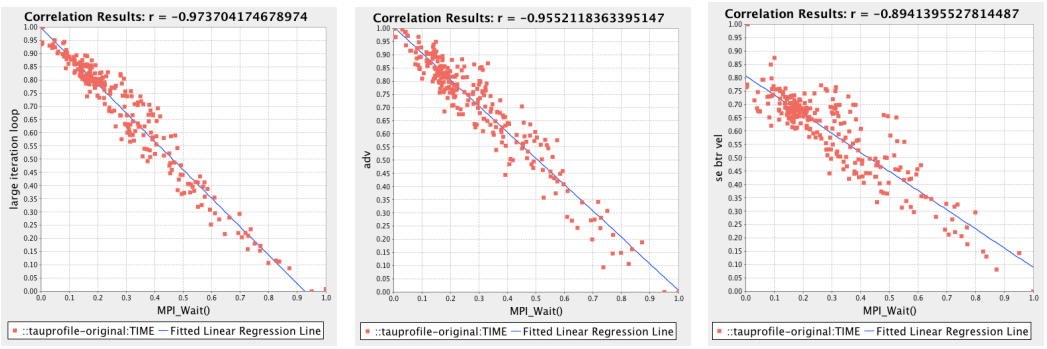


Fig. 3. 256 process performance profile of MPAS-ocean using K-way data partitioning, element decomposition and the RK4 solver on Edison. The height and color of the bars represents the time spent in each timed code region, the highest of which is `MPI_Wait()`.



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

Fig. 4. Correlation of time spent in `MPI_Wait()` with computation regions.

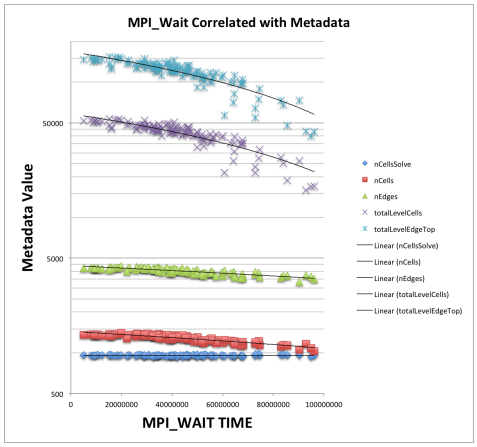


Fig. 5. Correlation of `MPI_Wait()` time with metadata values.

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 4, the time in `MPI_Wait()` is negatively correlated with main computation routines. These observations all indicated that there was a load balance issue in the code. This result seemed counterintuitive, as the domain was statically decomposed using the k-way strategy in gpmetis [16] which reported an imbalance 1.028

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 (`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 5. 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`, a deeper understanding of the *halo* or *ghost* cells associated with each block of the partition is needed.

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 must compute all of the cells explicitly assigned to the block (`nCellsSolve`), as well as the *implicitly* assigned halo cells, yielding 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. For example, a del2 operation requires only a single halo layer, while a del4 operation requires two halo layers. 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 for each iteration to ensure 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 variation 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. While the most significant property regarding load imbalance is the halo region size, 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. We 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, we could not confirm this since at that time we did not have a method for visualizing the metadata nor performance properties associated with each block.

IV. Approach

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. 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.

As with most simulation codes, MPAS already could output time slices of its state, storing its mesh layout and fields on that mesh. The file format used for MPAS was NetCDF. Although VisIt natively supports NetCDF, it was not, by default, able to interpret MPAS's Voronoi layout. So, to support MPAS, a new VisIt reader was written, leveraging heavily from an existing VTK-based MPAS reader. Further, the VisIt reader was augmented to accept additional performance data from TAU. All of the performance data from TAU was defined on partitions of the MPAS mesh; VisIt presented this data to the user alongside the fields from the simulation (e.g., temperature). To sum, the experience of using VisIt on combined MPAS and performance data was substantially similar to that of using VisIt on any simulation code, with the only difference being extra information (performance data) that could be displayed on the mesh. Further, the linking between the two was achievable by writing a new file format reader, a relatively modest task.

The TAU profiles are loaded into the TAUdb database [17], 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. The script iterates over all processes, generating a data file for each selected property to visualize. In our example, we generated files for the relevant load balance metadata fields (nCells, nCellsSolve, nodeid, number of neighbors) as well as the relevant performance measurements (`MPI_Wait()` time, aggregated computation time). In order to generate the MPAS VisIt files, the application data, partition data file and performance data file names are listed in an index file with ".mpas" as the suffix. The MPAS file lists the NetCDF data file, the Metis partition output file, and the metrics exported by PerfExplorer.

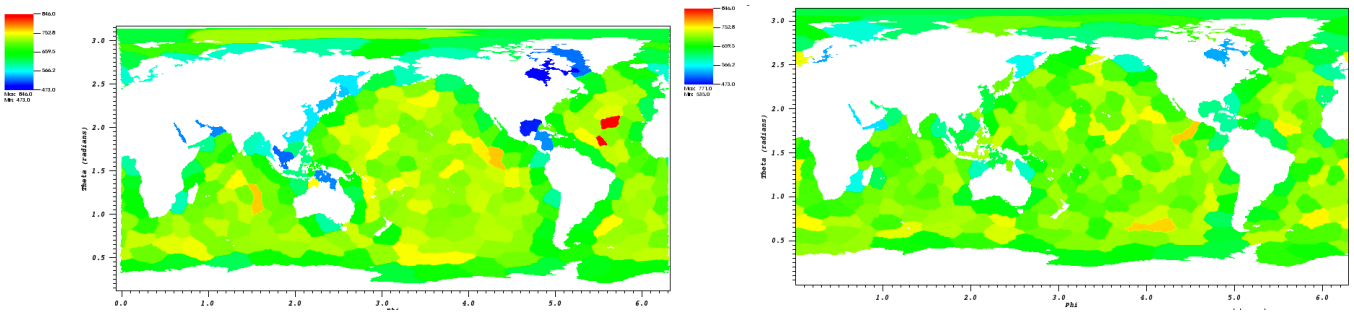
V. Experiments

As an experiment to test the TAU and VisIt integration, we evaluated the effectiveness of an experimental iterative, optimizing partitioning strategy. The strategy is called *Hindsight*, and it has one primary objective, with a secondary objective that happens to be a convenient side effect. The primary objective of Hindsight is to reduce the number of cells in the largest block of the partition, while the secondary objective is to balance the load among blocks.

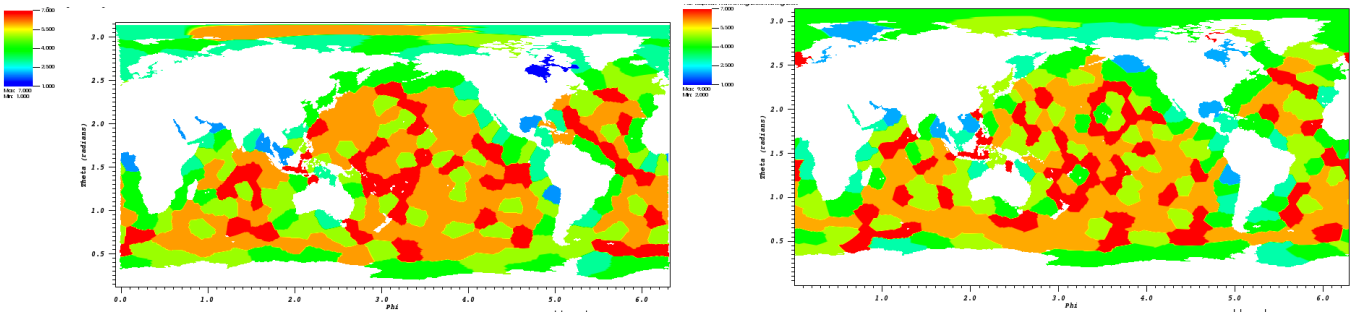
Current partitioners are very successful in decomposing meshes and grids into well balanced blocks. However, they rely on independent properties of the mesh or grid that are known prior to partitioning, such as node and edge weights. Unfortunately, the number of cells in the halo region of a block is unknown until the block is created by the partitioner. Because of the significant halo regions used by MPAS, the partitioner should ideally balance on the total amount of work assigned to a block, rather than just the number of cells explicitly assigned to the block. The Hindsight partitioner attempts to do just that.

The Hindsight partitioner executes the following algorithm. First, the mesh is partitioned by gpmetis, using the *kway* partitioner and default settings. Hindsight then parses the original partitioning generated by gpmetis, computing statistics such as the total cells explicitly assigned to each partition. Hindsight then examines each block to find the cells on the border of the block. Using those cells, Hindsight will find the cells that belong in the n layers of the halo region for the block, in this case 3. For each block, Hindsight then sums up the total number of cells explicitly assigned to the block as well as the number of cells in the halo region for that block. A new graph file is then formatted, assigning a weight to each cell. The weight for a cell is the *total* number of cells in the block to which it was assigned, including the halo. Hindsight then uses gpmetis to repartition the weighted mesh, using the output from that execution as input for the next iteration. This process iterates until the partitioning cannot be improved, or until a maximum number of iterations are executed. "Improvement" is measured by computing the number of cells assigned to the largest block, including its halo cells.

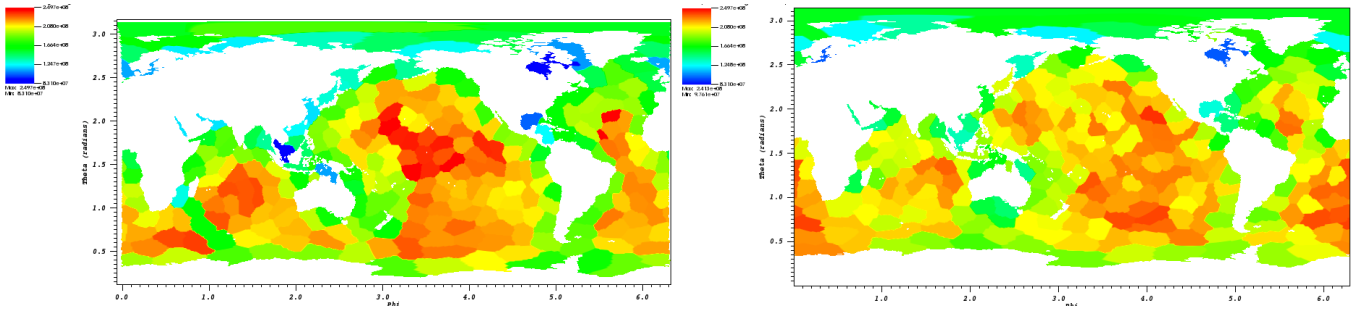
The optimization is not monotonic – the imbalance frequently immediately gets better and then worse, as the weights on the cells change on each iteration, sometimes considerably. For the "optimized" version of the 256 block version of the 60km dataset, the total number of cells in the largest block is reduced from 846 to 771 after 18 iterations. Experimental parametric variants of the algoritm are able to reduce it down as low as 743, but are not as broadly applicable to other data sets. The improvement generated by Hindsight is largely dependent on how good the original partition is – other MPAS data sets are not improved as much if there are not many "large" block outliers in the distribution. The Hindsight partitioner is still experimental, but shows promise in reducing the computational load for the processes that are assigned the largest blocks.



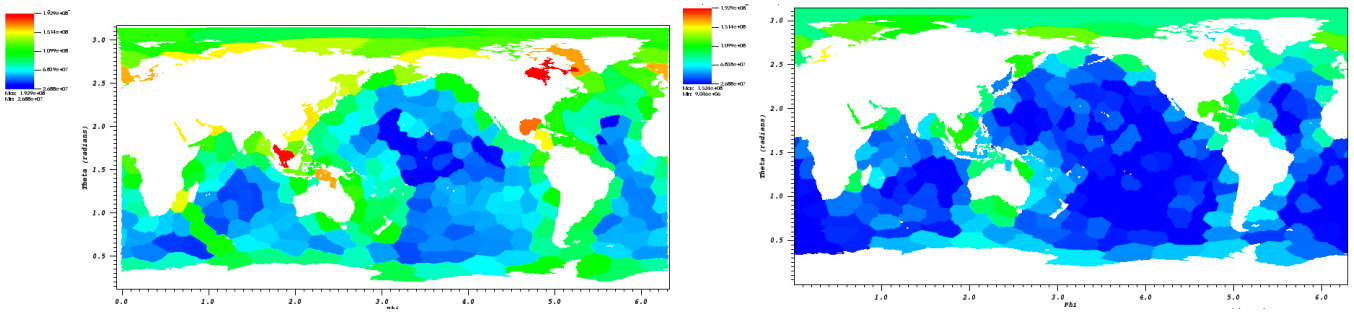
(a) Total number of cells per block.



(b) Total number of neighbors per block.



(c) Time spent in computation.



(d) Time spent in `MPI_Wait()`.

Fig. 6. 2D projections of TAU metadata and timer measurements for the original (left) and hindsight (right) partition strategy.

VI. Problem Visualization and Solution Validation

To test the integration, we ran an experiment similar to the one described in Section III-D, using the default data decomposition for 256 processes and collecting a TAU profile. We post-processed the TAU profile to generate the VisIt input data files. A view of the data in VisIt is shown in Figure 6 on the left column. As can be seen in the visualization, the partition blocks with less total cells spend less time computing and therefore arrive at the end of the timestep early. They subsequently spend more time waiting for their neighbors at the halo exchange phases. Even worse are the two blocks in the Atlantic Ocean that are clearly outliers in the other direction – the remaining 254 processes will wait for them to complete on each iteration and sub-iteration of the MPAS simulation. The visualization also confirmed our intuition with respect to coastal blocks as well as poorly connected blocks, in that they arrived at the synchronization points first, and the deep ocean blocks were those that arrived last. The data can also be rendered as a 3D planetary view for interactive exploration, but the 2D projections are easier to comprenend in print.

The right column of Figure 6 shows the effect of repartitioning with the Hindsight optimizer. The subfigures in Figure 6 have had their color ranges scaled in VisIt to match those of the original test case. The two blocks that previously dominated computation in the center of the Atlantic have had their computational load distributed to the surrounding blocks. In addition, more of the total computational load is borne by the previously underutilized coastal blocks. Overall, the computational load was reduced slightly, but more importantly the time spent in `MPI_Wait()` was reduced by 40%, leading to an overall reduction in execution time of about 10%. `MPI_Wait()` no longer dominates the execution time of the code, but the mean time spent in the key phases of the application, before and after Hindsight is applied. The slight increase in time spent in the main iteration loop is more than offset by the decreases found in other halo exchange phases. Clearly the optimization is not perfect - the block representing Hudson Bay north of Canada is still underutilized, mostly due to the lower computational and communication demands for that region. In that region, the water is not as deep (less computation), and the block has only one neighbor (less communication). Perhaps those features could be integrated into the weights of cells generated by Hindsight.

VII. Ensemble Visualization

During experimentation with the hindsight partitioner, an ensemble of data was created, with each member of the ensemble being a proposed partition. Since visualization tools like VisIt are able to cope with ensemble data (often via mechanisms for variation in time), we were able to visualize and compare this ensemble, as shown in the small multiples plot in Figure 7. We include this figure and expound on the point of ensembles because they help support an overarching point, namely that scientific visualization tools are flexible and can be used in many different ways, and that this in turn can add insight into performance data in many ways.

VIII. Related Work

Virtually all performance measurement and analysis tools provide some form of visualization. Most of the tools show performance information with respect to paralel entities (e.g., processes, threads), program structure (e.g., routines, call paths), and certain actions that take place. There has been a lot of work on the presentation of message communication performance and especially static and dynamic patterns that can be discerned especially from trace-based measurement.

The research work related to the results we present here looks at how performance data is mapped to a physical or logical domain. Topological-based mapping uses a topology specification to establish a coordinate system upon which performance data is shown. Scalasca is a powerful performance system that has extended support in it Cube 3D analysis [18] to show how performance data is distributed across a parallel execution using a computational topology base on a cube topology. They also employ a hardware layout topology for the Blue Gene/Qs 5D hardware topology to map performance information. Spear et al. [19] investigated extensions to TAU's ParaProf 3D visualizer to allow layout and mapping to be specified more generally. In particular, they showed how to create different topological-based performance displays based on both logical and physical characteristics. The BoxFish [20], [21] visualization tool has a similar capability, in that performance data is mapped to the hardware domain and either rendered in 3D or projected to various 2D domains. It provides a flexible framework for programming how the views are created.

However, there have been fewer instances of using the application domain itself as a target for rendering the performance information. The research most closely related to our work is reported in Böhme's thesis [22] which considers the problem of characterizing load and communication imbalance in parallel applications. In studying scaling issues due to load imbalance of selected modules of the Community Earth System Model (CESM), they developed a performance visualization within Cube 3D which mapped MPI waiting times onto physical earth coordinates to show severe load imbalances between regions of sea ice and open ocean. This is similar to what we are doing with MPAS. The main difference is that we are utilizing a sophisticated scientific visualization system to produce the end result, thereby gaining flexibility and visualization power via an integrated performance analysis workflow.

IX. Conclusion

In efforts to optimize the MPAS-Ocean application, the performance measurement and analysis process revealed interesting properties related to how the unstructured mesh application data was decomposed prior to execution. The domain decomposition, previously considered to be balanced, was used in the MPAS-O simulation in ways that led to

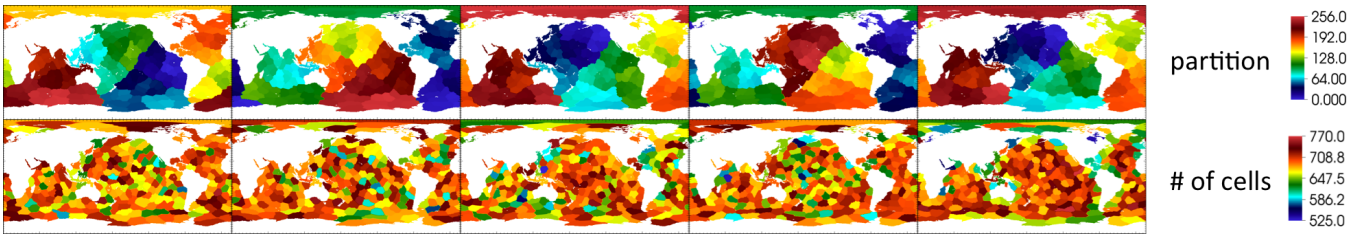


Fig. 7. This figure shows five outputs from the hindsight partitioner. The figure is arranged in a 2x5 layout, with the top row colored by partition, and the bottom row colored by the number of cells. Each of the five columns corresponds to a member of the ensemble of outputs from the partitioner.

a workload imbalance due to the significant number of halo (ghost) cells shared between partitions. By integrating the TAU performance system with a scientific visualization toolkit, such as VisIt, we were able to visualizate the performance data and metadata within the application domain itself, leading to a better understanding of the partitioning challenges, and why attempted optimizations were or were not successful. The integration process was straightforward and opened up opportunities for domain-specific mapping. In particular, spatial representation allowed partitioning strategies to be compared both in respect to their blocking properties (which are unknown until after partitioning is complete) and the resulting effect performance behavior across the physical model space. These visualizations could also be helpful in placing the application processes on the allocated hardware to improve communication locality. In future work, we will use the visualizations to improve the Hindsight partitioner. The partitioner is currently only using the block cell count to assign weights to cells, but that weight could also include other parameters that effect computation load and communication overhead, such as the number of vertical levels in the cell as well as the number of neighbors for a block. Both properties have been visualized with this framework, and could contribute to a better load balance.

X. Acknowledgements

This work is supported by the Department of Energy SciDAC SUPER and SDAV Institutes. Doug Jacobsen was supported by the US DOE Office of Science, Biological and Environmental Research program.

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] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, and P. Navrátil, "VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data," in *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, Oct 2012, pp. 357–372.

[12] 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.

[13] 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.

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

[15] 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

[16] G. Karypis and V. Kumar, "Metis - unstructured graph partitioning and sparse matrix ordering system, version 2.0," University of Minnesota, Tech. Rep., 1995, http://glaros.dtc.umn.edu/gkhome/views/metis.

[17] K. Huck, A. Malony, R. Bell, and A. Morris, "Design and implementation of a parallel performance data management framework," in *Proceedings of the International Conference on Parallel Processing (ICPP2005)*, Oslo, Norway, 2005, pp. 473–482, *(Chuan-lin Wu Best Paper Award)*, http://dx.doi.org/10.1109/ICPP.2005.29.

[18] 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

[19] 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

[20] 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.

[21] 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

[22] D. Böhme, "Characterizing load and communication imbalance in parallel applications," Ph.D. dissertation, RWTH Aachen University, 2014.