ABHINAV BHATELE | TODD GAMBLIN | BRIAN T N GUNNEY | MARTIN SCHULZ | PEER-TIMO BREMER | KATHERINE E ISAACS

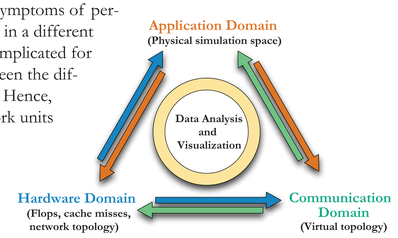
Revealing Performance Artifacts in Parallel Codes through Multi-domain Visualizations

Traditional methods of performance analysis can require significant efforts and time on the part of the application developer. Here, we present a case study to demonstrate the use of visualizations in multiple domains: application, hardware and communication, for discovering performance artifacts in a parallel application. Using a structured adaptive mesh refinement code, we present visualization and analysis techniques which make the process of performance debugging faster and more intuitive.

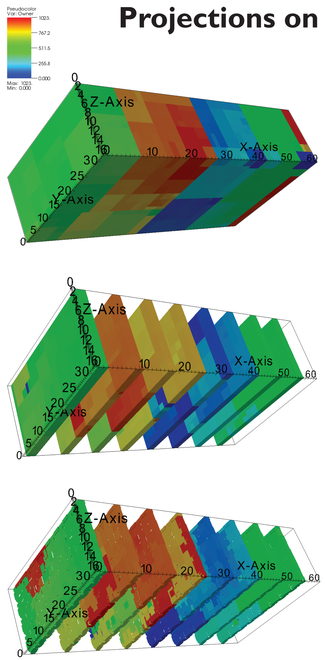
The HAC model

Performance data is measured in domains such as application, communication or hardware. The HAC model provides an intuitive way of analyzing performance by projecting data obtained from one domain in to another domain. By doing this, we can correlate symptoms of performance problems in a domain to their root causes in a different one. Projecting data across domains can be more complicated for adaptive applications because the relationships between the different domains change as the simulation progresses. Hence, we need to keep track of the movement of data/work units and changes in communication across phases.

In this poster, we present the projections of measured data across multiple domains to discover performance bottlencks in an adaptive code.



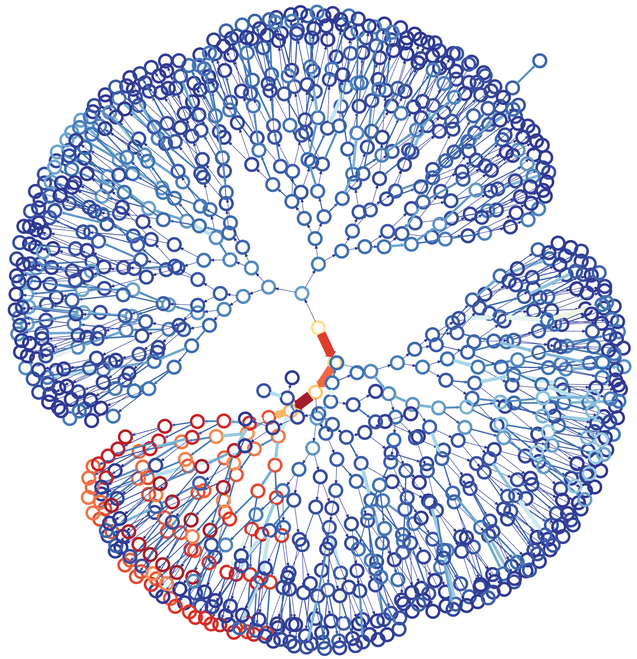
Projections on the application domain



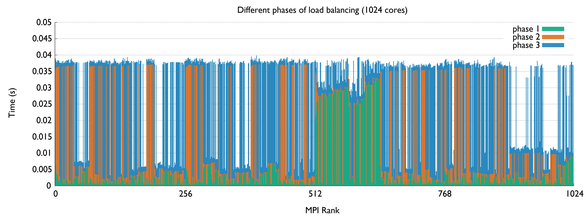
Structured adaptive mesh refinement (SAMR) is an application domain with challenges in scaling and performance analysis. We study a linear advection (LinAdv) benchmark that uses a popular SAMR library called SAMRAI. SAMR applications decompose the simulation space hierarchically into multiple mesh levels. Each mesh level is broken down into patches that contain cells in the areas where a mesh level has been refined. LinAdv has three patch levels, 0, 1 and 2 with increasingly finer resolution.

On the left, we use VisIt to visualize the application domain for a 1024-core run on Blue Gene/P. Instead of coloring the patches by their physical properties, we color them by the MPI rank of the process that owns them. This gives an indication of whether neighboring patches in the application domain get mapped to nearby physical processors on the hardware domain - in this case a three-dimensional torus network.

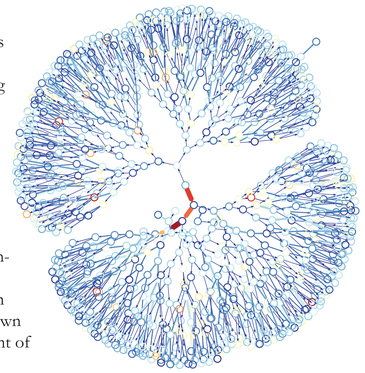
We can see that nearby patches are colored in similar shades of red/yellow, blue or green. Even across different levels, patches at the same cartesian coordinates have similar colors. We can also color the patches by the average or maximum number of hops traversed by messages from a patch to its neighbors. These mappings can be used to analyze if communication between patches is efficient.



Projections on the communication domain



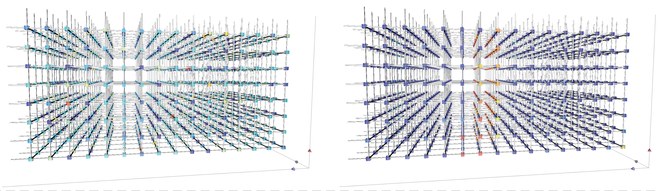
Load balancing in SAMRAI takes more and more time as the problem is scaled to a large number of processors. The times spent in the three phases within load balancing were recorded and plotted linearly by the MPI ranks (above). Phase 1 *i.e.*, load distribution appears to lead to longer times spent in other phases. Coloring the nodes in the communication domain, which happens to be a binary tree (right), by the amount of load on each processor did not provide any clues.



We then projected the timing data for phase 1 to the communication domain (figure on the left). This time, we clearly see that a part of the tree is significntly delayed (in red). Coloring the edges by the number of boxes sent down the tree shows a flow problem arising from the movement of load from the rest of the tree to this sub-tree.

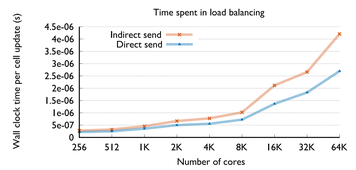
Projections on the hardware domain

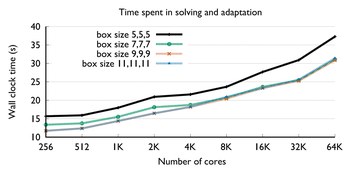
Communication is becoming the dominant bottleneck as we scale to a large number of cores. It becomes important to analyze communication in terms of contention on specific links and distribution of network traffic between various directions. Boxfish can be used to visualize such data. In the figures below, an 8 x 8 x 16 torus is shown with the nodes colored by their load on the left and by the time spent in a phase of load balancing on the right.



Performance improvements

We tried some preliminary solutions to mitigate the problem observed in the load balancing phase. We used two strategies to reduce the amount of data being moved: 1. Send the patch destinations to the source processor directly instead of sending the data over the tree network, and 2. Increase the box size which reduces the number of boxes. Both these measures reduce the time spent in load balancing and also the overall time. A scalable solution will be implemented in the future to compleletely eliminate this bottleneck.





Center for Applied Scientific Computing, Lawrence Livermore National Laboratory

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-POST-527971).