Interpreting Performance Data Across Intuitive Domains

Martin Schulz[1], Joshua A. Levine[2], Peer-Timo Bremer[1,2], Todd Gamblin[1], Valerio Pascucci[2]
[1]*Lawrence Livermore National Laboratory, Livermore, CA, USA*
[2]*Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT*
{*schulzm, ptbremer, tgamblin*}*@llnl.gov,* {*jlevine, pascucci*}*@sci.utah.edu*

***Abstract*—To exploit the capabilities of current and future systems, developers must understand the interplay between on-node performance, domain decomposition, and an application's intrinsic communication patterns. While tools exist to gather and analyze data for each of these components individually, the resulting information is generally processed in isolation and presented in an abstract, categorical fashion unintuitive to most users. In this paper we present the *HAC* model, in which we identify the three domains of performance data most familiar to the user: (i) the application domain containing the application's working set, (ii) the hardware domain of the compute and network devices, and (iii) the communication domain of logical data transfers.**

We show that taking data from each of these domains and projecting, visualizing, and correlating it to the other domains can give valuable insights into the behavior of parallel application codes. The *HAC* abstraction opens the door for a new generation of tools that can help users more easily and intuitively associate performance data with root causes in the hardware system, the application's structure, and in its communication behavior, and by doing so leads to an improved understanding of the performance of their codes.

***Keywords*-Performance Analysis and Visualization**

I. Motivation

By the end of this decade, exascale machines are expected to support 500 million to 4 billion concurrent tasks. At the same time the complexity of both architectures and applications will increase significantly, leading to often unexpected and unpredictable interactions between applications, middleware, and underlying hardware. In order to exploit such machines fully, programmers must be able to measure, analyze, and understand the behavior of their applications and its corresponding impact on the overall performance in great detail.

Most current tools gather and present performance data in the native hardware or MPI rank domain, relying on the user to infer causal relationships and interdependencies. However, neither of these two domains is intuitive or familiar to the users. As a result, data appear as abstract or highly discrete, and domain and performance experts must expect considerable effort to analyze it. At exascale the quantity and complexity of performance data will grow by orders of magnitude, rendering this traditional, manual performance analysis infeasible. Traditional techniques for analyzing scientific data have dealt with scale using approaches on continuous data. In this paper, we borrow from these techniques and show that they can be used to present performance data in more intuitive ways by projecting it, e.g., to the simulated application domain. Doing so makes performance data easier to visualize and analyze, and it allows us to discover correlations of features across domains that would otherwise not be apparent.

We introduce the *HAC* model (illustrated in Figure 1), which breaks performance data into three distinct categories. These are the ***H***ardware domain consisting of network nodes and physical links between them, the simulated ***A***pplication domain, understood by application scientists and mathematicians and designed to represent the underlying problem (e.g., matrices, unstructured grids, or 3D volumes), and the ***C***ommunication domain, which captures the communication patterns of the application.

Using this basic *HAC* abstraction, we define and implement mappings between the domains that enable us to project data from one domain to another, emphasizing correlations between performance data gathered in any two of the three domains. Further, by projecting data from one

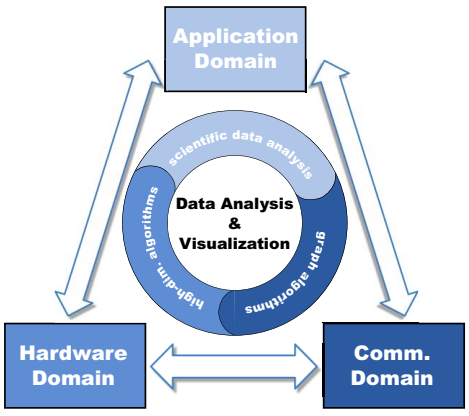


Figure 1. Interplay between the three critical domains for performance analysis.

 This work was partially performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. ICPP September 2011, Taipei, Taiwan.

0190-3918/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPP.2011.60

domain to another, we can then use the set of data analysis techniques available in the destination domain to extract and compare features. For example, we might project floating point counts from the hardware domain into the application domain and then analyze them by applying feature-finding algorithms there. The correlation of hardware, communication, and application data allows application developers to understand how structures in their simulated domains correspond to real performance problems, and it also gives systems engineers a new, more intuitive perspective on how simulations map to physical hardware. This correlation is not possible with existing approaches.

This paper makes the following contributions:

- We introduce the *HAC* model to describe three of the most relevant data domains for performance analysis;
- We define and implement projections between these domains that allow us to project data to new domains and analyze it there;
- We present a measurement framework to allow concurrent, consistent data collection from multiple domains;
- We show how correlating data across domains provides new insight into application performance; and
- We demonstrate our approach with three case studies, each highlighting the use of one of the three inter-domain mappings.

The remainder of this paper is structured as follows: Section II briefly discusses the state of the art in performance analysis tools, Section III introduces the *HAC* model and corresponding data mappings in detail. Section IV presents our experimental setup and Section V discusses the three case studies and illustrates projections between domains. In Section VI, we state our conclusions.

II. Background and Related Work

There is a large body of existing research on performance tools for parallel programs. Tools such as HPCToolkit/HPCviewer [22], SCALASCA [38], its predecessor Kojak [24], Paradyn [23], and Open|SpeedShop [32] perform a wide range of performance measurements including tracing application behavior over time and low overhead statistical profiling. However, performance tools do not typically tie their data directly to the application or communication domains. Rather, they report performance measurements on a per-source region or per-rank/process basis, leaving the user to associate the reported data to application semantics.

Several tools provide sophisticated displays, allowing a deeper insight into the performance an application. The Vampir [26] and Vampir Next Generation (VNG) [5] trace viewers offer a multi-scale display for large message traces. HPCToolkit can scalably display large callpath traces [34]. Other analyses allow developers to associate scaling and load balance problems with particular code regions [11], [35], [36]. TAU's PerfExplorer directly compares performance

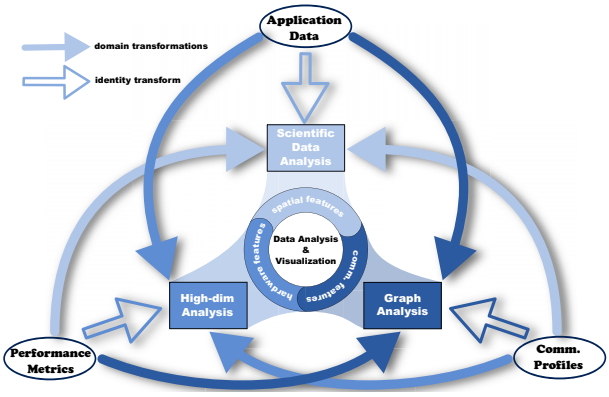


Figure 2. Mappings between domains in the *HAC* model.

metrics in 3D plots [17]. However, these tools restrict performance data to its own domain and do not allow deeper correlation with application or hardware data.

Kojak's/SCALASCA's Cube browser is closest to the work presented here. It presents projections of performance data onto the physical network topology of supercomputers. Extensions of this work also explore opportunities to explain performance results in application space [2] by exploiting adjacency of processes in the application domain.

SvPablo was one of the earliest tools [27] to offer performance visualization, with visualizations of communication data in an animated tunnel.

Overall, existing tools have limited capabilities for correlating performance data from different domains, and this limits the insights that can be derived from visualizations.

III. The *HAC* Model

Most traditional scientific data analysis techniques are concerned with many quantities of interest (temperature, pressure, etc.) defined on a single common domain such as physical space. Performance data has many "natural" domains describing different aspects of application or system behavior. For example, hardware counter metrics are measured on a per-core or per-socket basis, and communication patterns are tracked within the MPI rank space abstraction. Understanding the meaning of these metrics typically requires relating them to other domains. For example, the performance of a particular core provides little information without a description of the part of the application domain to which it is assigned, or information about the portion of global communication it performs.

The *HAC* model allows tools to account for the multi-domain nature of this problem by identifying three key domains for performance measurements and by defining projections between them. Figure 2 illustrates this approach: we gather data in each of the three domains in the form of application data, performance metrics and communication profiles, and we define projections of this data to the

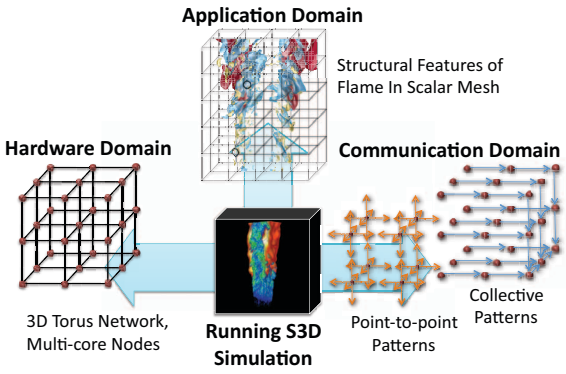


Figure 3. Domain decomposition of S3D.

other two. This not only allows us to directly compare the data across domains, but also opens the door to using data visualization and analysis tools available in the other domains. We can now apply scientific visualization tools to performance data projected into the application domain, or we can apply graph-based techniques to the same data projected into the communication domain.

The *HAC* model provides a structured characterization of common performance analysis and optimization challenges in terms of relevant source data/domains, the necessary mappings, and the subsequent analysis requirements. This structure will lead to new insights, more focused research directions, and a common framework to describe, compare, and combine different approaches. In the following we briefly describe the three data domains and a set of mappings among them using, as a reference, the combustion code S3D [16] (see Figure 3).

Application Domain: The application domain represents the physical or other simulated system modeled by an application code. Often it is a piece of physical space with materials represented by some grid or mesh, but more abstract domains such as sparse matrices or large, abstract graphs are also possible. This is the domain most familiar to domain scientists, and it generally provides the mental reference frame during the code design stage. The application is ideal for exploring performance problems related to specific physical phenomena of the simulation. As an example, consider S3D, which describes a flame with some features of interest, as shown at top in Figure 3. Correlating features of the flame with performance data, such as unusually high cache miss counts, may highlight problems with a specific chemistry kernel or with some data structure in the application. There exist a large number of scientific visualization and analysis tools for application domains that can be exploited for performance analysis once the data is mapped into this space.

Hardware Domain: The hardware domain describes the physical hardware of a computer system in terms of computing cores and network links. A common configuration in today's integrated HPC systems is a mesh or torus network of multi-core compute nodes as shown on the left in Figure 3. Commodity clusters often rely on Infiniband (IB) networks with fat tree topologies. Each of these networks has its own graph of hardware links connecting nodes.

Data analysis in the hardware domain poses interesting challenges. Clearly, large scale graph algorithms such as clustering can be helpful. However, many of the largest supercomputers use regular grids connected into three-dimensional tori. Future machines will use higher-dimensional mesh/torus topologies. Given the large number of nodes and the Cartesian scaling properties, it is reasonable at scale to treat such a regular grid as a discrete representation of a continuous space. This allows the use of new classes of algorithms such as local correlation analysis [18], [29] or topological techniques [15], [19] to detect and extract features from high dimensional manifolds.

Mapping between hardware and application domains is application-specific, and it is typically done manually. Mesh-based simulations, for example, distribute partitions of a mesh across nodes, and particle systems group sets of neighboring particles on each node. However, this mapping does not need to be static. Adaptive simulation frameworks, such as SAMRAI [37], dynamically create and destroy mesh partitions and rebalance computational load by moving partitions among cores.

In practice, mapping from the application to the hardware domain will result in a downsampling or averaging of information since there are fewer cores than mesh elements. For the reverse mapping one typically works with two representations, the high resolution application domain and a lower resolution representation carrying hardware information, as shown in Figure 4.

Communication Domain: The communication domain consists of a general graph. For MPI programs, such as the one studied here, nodes correspond to MPI processes and edges correspond to message exchanges between them. During execution, most MPI applications will exhibit multiple distinct communication patterns. For example, as shown on the right of Figure 3, the S3D code uses primarily a stencil-based neighbor exchange pattern resulting in a grid that is almost identical to the hardware domain. This is interleaved with global collective patterns, such all-to-alls and the reduction shown in the figure.

Large-scale simulation codes explicitly decompose the application domain over MPI processes. Typically, it is this process decomposition that is specified in the application code, rather than a hardware-specific decomposition over the processors themselves. Together with a communication to hardware mapping specified at runtime, these define the application to hardware mapping. The mapping itself is of particular interest, because it can severely affect performance if it is constructed in a way that slows down inter-node com-

munication. The problem of choosing the best-performing mapping of MPI processes to physical processors is referred to as the *node mapping problem*. The full search problem is NP-hard, but given a mapping, we can embed the edges of the communication graph into the hardware domain, and we can use this to predict the expected network traffic on each link in the hardware domain. This can help with locating performance problems.

IV. Experimental Setup

In the remainder of this paper, we present three case studies showing how inter-domain projections can be defined, implemented and used to gain new insight into application performance.

Experimental Platforms: For our experiments, we use two systems at Lawrence Livermore National Laboratory: *Hera* and *DawnDev*. *Hera* is an 864 node multicore cluster with an Infiniband interconnect. Each node consists of four AMD Quadcore 2.3 GHz processors (8,356 total), and each core has its own L1 and L2 cache. The four cores share a 2 MB L3 cache. Each node runs CHAOS 4, a high performance Linux variant based on RedHat Enterprise Linux. We rely on MVAPICH as our MPI implementation.

DawnDev is a Blue Gene/P system consisting of a single rack with 1,024 compute nodes, of which 512 were available to us at a time. Each node contains a quad-core 850 MHz PowerPC 450 Processor and the nodes are connected by a 3D torus network. BG/P systems use a custom compute node kernel and rely on IBM's optimized MPICH2 implementation for communication.

Measurement System: To obtain data suitable for analysis within the *HAC* model, we require a measurement system that is capable of gathering data from multiple sources and domains concurrently. This is necessary to guarantee consistency between the different data sources and avoid differences caused when measuring multiple executions.

We achieve this by layering the *HAC* measurement system on top of P^{N}MPI [31]. This infrastructure, once linked with the code, allows the dynamic loading of multiple PMPI profiling tools in a single run. Each measurement tool is a separate component, loaded by P^{N}MPI at program start time. When necessary, these modules can leverage common resources or services, such as request tracking or datatype walking, keeping the implementation effort for each module focused on the targeted data source only.

V. Case Studies

We use the *HAC* model in three case studies from the areas of fluid mechanics, multigrid solvers, and molecular dynamics, each highlighting a different mapping in our three domain *HAC* model.

A. Performance Data in the Application Domain ($H \rightarrow A$)

Performance counters and other measurement devices are present on most modern microprocessors, and these provide measurements of architectural features, caches, on-chip networks, and other components of computer systems. This data is collected in the hardware domain, as it pertains to system resources such as nodes, networks, and cores. For parallel applications, the data is typically associated with the rank of the MPI process taking these measurements, as this is a readily available process identifier and MPI ranks roughly correspond to measurements for single cores.

The MPI rank space is unintuitive, since it is a flat namespace with no inherent locality semantics. MPI ranks are distributed arbitrarily among cores on many systems, and it is not always possible for tool users to obtain precise network topology information easily.

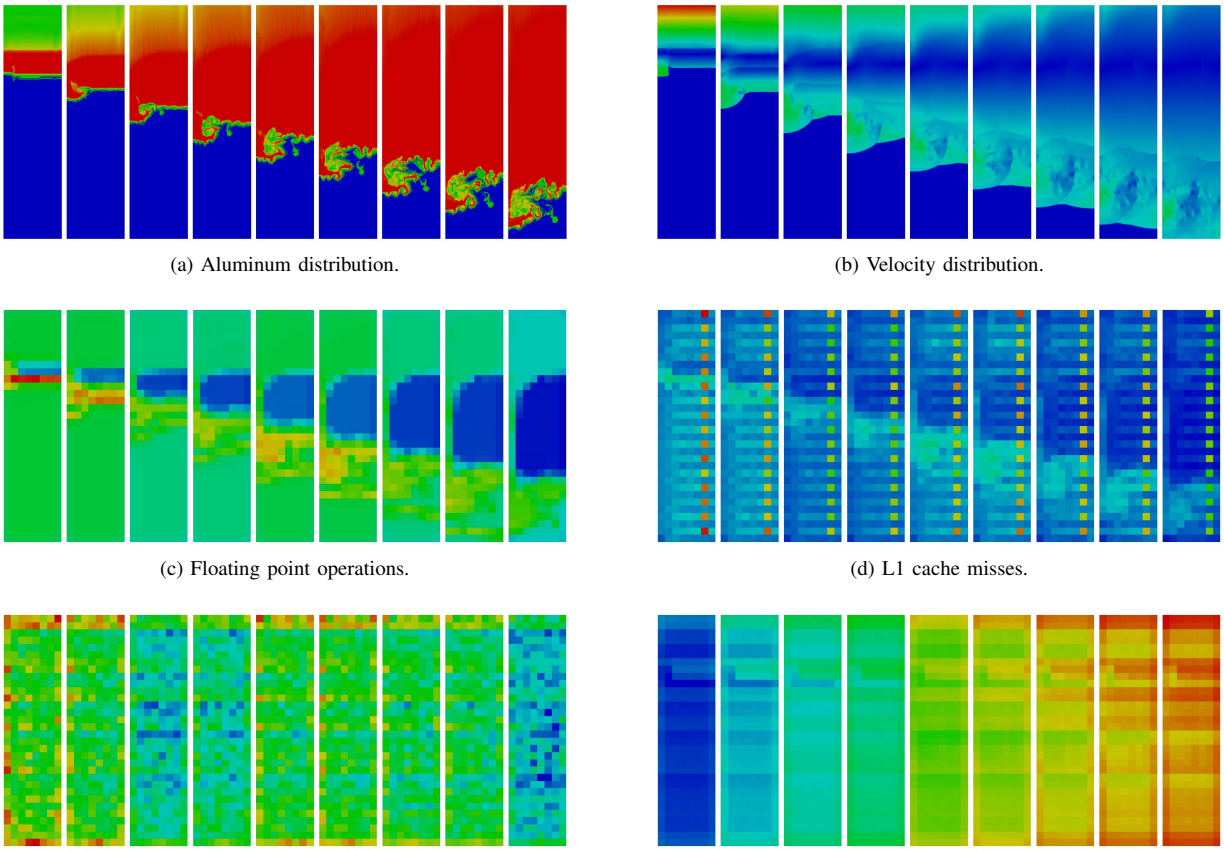
In order to provide better attribution of hardware measurements to application-domain structures, this first case study maps performance data from the hardware domain to the application domain. Here our hardware domain is the machine structure of *Hera*, and the application domain is the physical system simulated by a fluid dynamics code. We correlate the features in the simulation with our performance observations.

Application: We use Miranda [7], [30], a higher order hydrodynamics code for computing fluid instabilities and turbulent mixing in 2D and 3D domains. It is primarily used to study Rayleigh-Taylor (R-T) and Richtmyer-Meshkov (R-M) instabilities, which occur in supernovae and inertial confinement fusion. The application uses a massively parallel spectral/compact solver for variable-density incompressible flow, including viscosity and species diffusivity effects [6].

In our experiments, we simulate an ablator driving a shock into an aluminum section containing a void and producing a jet of aluminum. The simulation results of a 12 hour run using a 2D grid running on 256 cores of *Hera* are depicted in Figures 4a and 4b using nine selected timesteps in regular intervals sorted from left to right. The figures clearly show the aluminum jet on the left side as well as the created shockwave traveling from top to bottom.

Measurement Setup: For this first experiment we concentrate on per-core performance information. We measure the total compute time spent per timestep as well as three marquee performance counters: L1 miss rate, number of floating point operations, and number of branch miss-predictions. We measure this data using PAPI [25] from within a P^{N}MPI module loaded transparently at application start. We take advantage of the application's built-in visualization capability, and we write out our performance data as additional fields within periodic visualization dumps.

Projection ($H \rightarrow A$): The simulation data is organized in a 2D regular, dense grid, which is split into equal chunks in all dimensions and distributed among the processors in row major order. In our case study we use 256 cores split



(a) Aluminum distribution.

(b) Velocity distribution.

(c) Floating point operations.

(d) L1 cache misses.

(e) Branch miss-predictions.

(f) Compute time per timestep.

Figure 4. Simulation results (top row) and performance data mapped to the application domain (middle and bottom row) — left to right: simulation progress, blue shows low and red high values.

into an 8 × 32 grid. We extract the mapping of grid cells to processor cores from the application, and we store the matching processor grid coordinates in the output files. Then, we use this data to determine which parts of the simulation grid are computed by which cores, and this provides us with the necessary projection function between the hardware and application domains.

Evaluation and Results.: Plots of these four performance metrics are shown Figure 4c)-f). Despite the simplicity of the experiment, the use of the *HAC* model provides valuable insight not possible using MPI rank space alone. Figure 5 shows a simple plot of FP counts per MPI rank, but it provides little insight into the application's behavior.

In contrast, the projections of both FP operation counts and L1 cache misses clearly show features that mirror the movement of the physical shockwave through 2D space. By simple visual inspection we see that the shockwave itself requires a higher number of FP operations to compute, but in the wake of the wave we see significantly fewer operations per iteration. Furthermore, while the areas not affected by

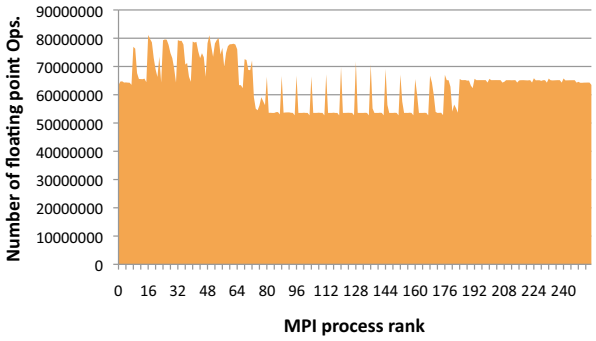


Figure 5. MPI rank space visualization of number of floating operations for rightmost timestep in Figure 4c.

the shockwave show a very steady number of operations, the computation of the wave clearly shows more variation. Similar observations, although with smaller differences, can be made for the L1 cache miss counts. Combined, these observations allow predictions of load imbalance.

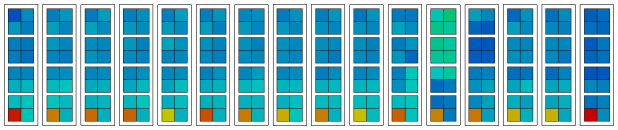


Figure 6. L1 cache data for leftmost timestep in Figure 4d) mapped to the architecture of the machine (2x2 blocks indicate a Quadcore socket, columns 4-way SMP nodes).

The number of branch mispredictions shows a different picture. We see higher numbers on the boundary of the domain caused by specialized code to handle boundary conditions. Over time, this effect is reduced, showing that the branch predictor was able to learn the application's behavior.

The actual compute time per iteration (which includes time spent blocked within an iteration and hence is not a good measure for load balance) also shows increased computation time at the top and bottom boundary conditions. More importantly, though, we clearly see a gradual increase in runtime per iteration caused by the increasing complexity of the simulated problem.

In addition to these application-related observations, which enable users to quickly reason about application-dependent performance, the L1 miss rates show a secondary effect. Mapping this information into and visualizing it in the architectural space of the hardware domain (Figure 6), we see that one fixed core per node has a significantly higher L1 miss rate as indicated by the series of red dots. This stems from processing overhead within MPI on that core for collective operations, in particular from large message broadcasts. As additional experiments verified, this core acts as a local leader for optimized collectives in MPI, which first gather on-node data to the leader and then use this buffer for broadcast [21].

This example shows how important it is to look at the data from different perspectives in multiple domains. It not only enables users to fully understand application-dependent overheads, identify correlations between application behavior and observed performance metrics, and with that establish a realistic baseline to detect anomalous behavior, but it also provides the necessary information to distinguish system overheads due to their *lack* of correlation with application data.

B. Logical vs. Physical Message Flow ($C \rightarrow H$)

Communication is a critical part of every parallel application. The rising complexity of both the application's communication patterns and the topologies of the underlying networks makes a proper understanding of the impact of application communication on physical network structure essential. This forms the foundation for a wide range of optimizations in the application, the network architecture, and mappings of processes to nodes.

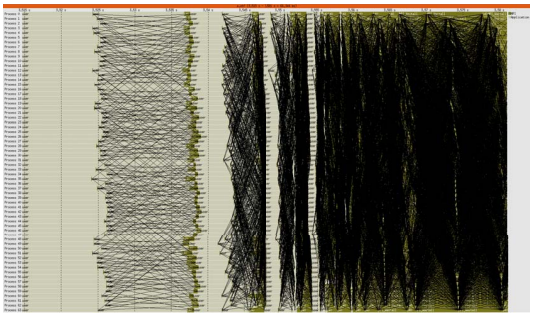


Figure 7. MPI trace using Vampir of the coarsening phase of a single AMG V cycle: fine levels are computation bound and can overlap computation and communication (left), coarser levels are dominated by communication (right).

Application: Algebraic multigrid (AMG) methods [4], [28], [33] are popular in scientific computing due to their robustness when solving large, unstructured sparse linear systems. In particular, *hypre*'s BoomerAMG [1], [8], which we use in our experiments and which is also part of the Sequoia benchmark suite [20], plays a critical role in a number of diverse simulation codes, such as elastic and plastic deformations of explosive materials and structural dynamics codes.

AMG methods operate on a V or W cycle, which means they start with a fine grid, which is iteratively coarsened through multiple levels until it can be solved directly. After this step, the result is successively interpolated through the same levels back to the fine grid. This process is then repeated until the system converges.

While finer levels of the AMG computation are compute-dominated and have regular communication structures with a limited set of neighbors, coarser levels are communication bound and exhibit more random and input-dependent communication pattern with a large set of neighbors [10]. This can be seen in traditional MPI traces, such as the one shown in Figure 7. However, MPI traces do not show the impact of such communication on the network hardware.

Measurement Setup: To provide more insight into the communication structure of AMG we perform a series of experiments on *DawnDev*, our Blue Gene/P system, and contrast measurements from the hardware and the communication domain. We use a complete midplane of 512 nodes, which are connected in an $8 \times 8 \times 8$ torus. The Blue Gene/P provides network performance counters that can be used to measure the number of packets sent to each of the six outgoing links of a torus node. This includes packets stemming from messages initiated on the local node as well as packets from messages passing through the node. We measure the actual communication in the AMG application by intercepting all MPI calls and aggregating the communication volume into a communication matrix covering all communication pairs.

Both measurement approaches are implemented as separate P^{N}MPI modules and loaded concurrently at the start of the execution. Additionally, we added minimal instrumentation to the core solver of AMG to distinguish the coarsening levels and partition our measurements in one group per level. For our particular example problem size of $N = 320^3$, AMG required eight levels of coarsening.

Projection ($C \rightarrow H$): In order to compare the measurements in the hardware and network domains, we must first project the information onto a common domain. In this case, the data in the hardware domain lacks the necessary information to be mapped to the communication domain since bandwidth on individual links is shared by several communication paths. We therefore project the communication onto the hardware domain.

To accomplish this, we first studied the behavior of the Blue Gene/P network for various torus locations and messages sizes and, based on this data, developed an accurate model of the network communication between two arbitrary node pairs under the assumption of no contention. We apply this model individually to all data transfers recorded in the communication domain, i.e., the communication matrix introduced above, and use the sum of the modeled traffic between all communication pairs as an estimate of the complete communication traffic. Once complete, we can directly compare the estimate data with the measurements.

Evaluation and Results: This estimate will not be fully accurate, particularly in high traffic scenarios, since it assumes infinite bandwidth links. However, it provides us a good estimate how the traffic *should* flow through the interconnect. Further, by comparing the estimate with the measurements on each links, we can now identify links that are saturated beyond capacity and we can also analyze which communication pairs contribute to this overload and hence cause bottlenecks. We further illustrate this with three visualizations: the measured network counters, the estimated traffic, and the ratio between the two.

Figure 8 shows the results of an AMG run with an $8 \times 8 \times 8$ and a $2 \times 4 \times 16$ processor grid in the application projected onto *DawnDev*'s $8 \times 8 \times 8$ torus. For the $8 \times 8 \times 8$ topology (Figure 8a-d)), we can clearly see the regular neighbor or stencil communication in AMG's finer levels. The ratio graphs show that communication on z links is more likely overloaded (and hence appears hotter in the graph) than in the other two dimensions, although the difference is small.

The behavior becomes more irregular and sparse for coarser levels until we hit an interesting point in level 6. Here, we begin to see potential performance penalties from hot links. However, we also see that these hot links exist in both the estimate and the measurements, meaning that the hardware still satisfied the communication requirements. Only a few edges show a difference between simulated and measured traffic, in this case mostly due to inaccuracies in our network estimation for links with low utilization.

The situation is different for the execution of the same problem split into a $2 \times 4 \times 64$ processor grid, as shown in Figure 8e-f). Here the estimate shows a serious of hot links (red), while the measurements do not expose a hot edge. In this case, the machine's router automatically adjusted the paths of messages to correct this problem, but with a potential performance penalty. The ratio graph shows that these differences are rather small and appear mostly for high y coordinates.

In all cases we can retain the projection information itself and use it to determine which node pairs use this edge for their communication. This will allow us to modify the mapping of processes to nodes such that such hotspots are no longer present.

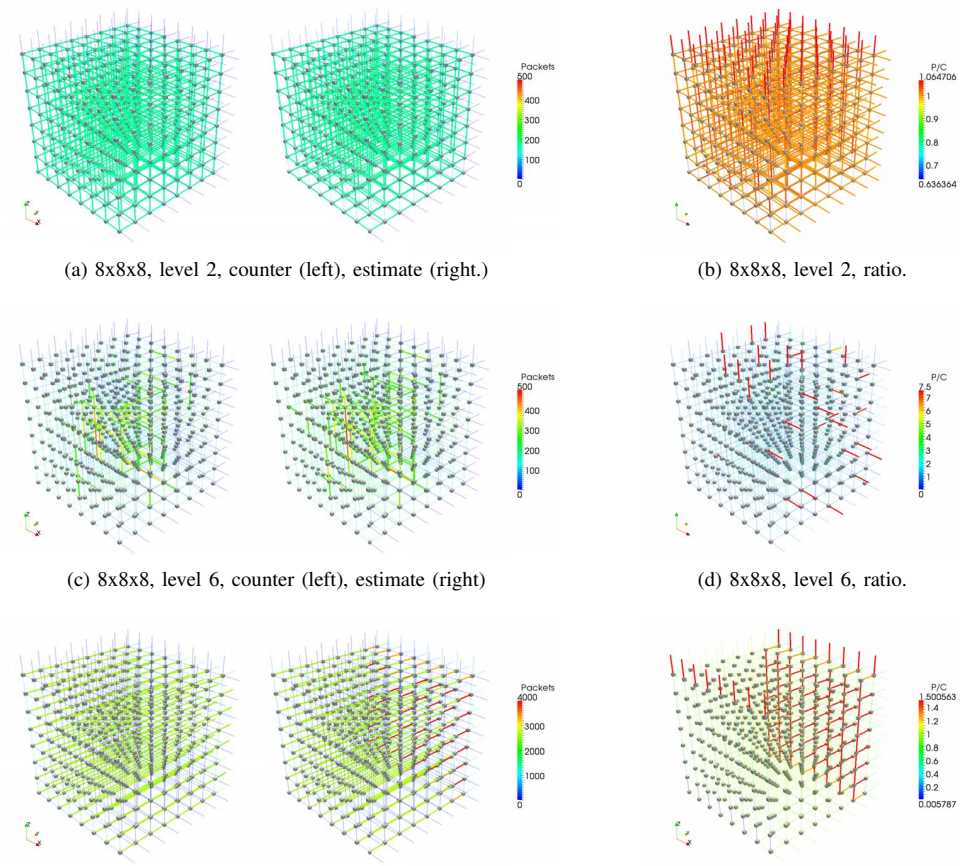
C. Identifying Communication Groups on Application Structures ($C \rightarrow A$)

Our experience with AMG illustrates the potential complexity of communication patterns in HPC applications. Many developers struggle with mapping their own applications' communication to high performance networks. In many cases, they could also benefit greatly by optimizing the communication patterns of third-party libraries. AMG, for example, is used as a solver library by many application codes, and its performance can dominate that of an entire run. Other codes make extensive use of parallel numerical linear algebra libraries, such as SCALAPACK [3].

Typically, application developers are not familiar with the communication patterns of the sophisticated algorithms used in parallel numerical libraries, and this poses a problem as networks become larger and have higher long-distance hop latencies. Meshes and tori are very scalable, but this comes at the cost of nonuniform latency and bandwidth. It has been shown that performance improvements of up to 65% are possible by remapping processes on a torus network [14], but this was achieved only through lengthy manual analysis.

In this section, we describe our mapping from the communication domain to the application domain. We have developed a P^{N}MPI tool to extract the major communication patterns of MPI applications in order to project them onto the application domain. This allows developers to tailor their own node mappings based on the network behavior of code they depend on, without having to understand all of the details of its communication.

Application: Our experiments in this section use QBall, an implementation of First-Principles Molecular Dynamics (FPMD), an accurate atomistic simulation approach used to predict the properties of materials without experimental data or variational parameters. QBall is widely used in many areas, including solid-state physics, chemistry, biochemistry, and nanotechnology. The FPMD approach combines a quantum mechanical description of electrons with a classical



(a) 8x8x8, level 2, counter (left), estimate (right.)

(b) 8x8x8, level 2, ratio.

(c) 8x8x8, level 6, counter (left), estimate (right)

(d) 8x8x8, level 6, ratio.

(e) 2x4x64, level 1, counter (left), estimate (right)

(f) 2x4x64, level 1, ratio.

Figure 8. Network traffic measured in HW domain (left), estimate (middle), and ratio (right) for AMG.

Figure 9. Communicators detected in QBox projected onto the application domain.

description of atomic nuclei. To reduce the exponential complexity of the many-body Schrödinger equation to $O(N^3)$ scaling, a plane wave, pseudopotential density functional theory (DFT) formalism is used [12], where N is the number of chemically-active valence electrons.

QBall inherits its structure from QBox [14], its predecessor that won the Gordon Bell award for peak performance in 2006 partly due to heavy node mapping optimizations. It is written entirely in C++, and it uses the FFTW library [9] as well as the ScaLAPACK and BLACS libraries for the bulk of its computation and communication [13].

Measurement Setup: For QBall, as for QBox, the key to a good node mapping is the structure of row and column communicators and the tradeoff between nearest-neighbor and local collective communication. QBall makes heavy use of `MPI_Comm_split` to create communicators. We use our P^{N}MPI tool to intercept `MPI_Comm_split` and `MPI_Comm_create` and to record new communication configurations.

One problem with existing profiling tools is that they are not able to effectively track identical communicators created at different times. Parallel numerical libraries often create special communicators that live for a single function call. The function is then called again later, and the communicator is re-created for the same communication patterns. A traditional PMPI tool attempting to track communication patterns by MPI communicator identifiers would not be able to detect this, as communicators are transient.

Our P^NMPI tool creates canonical identifiers for communication patterns created by `MPI_Comm_split` and identifies identical communicator groups at runtime, enabling their profile information to be grouped together. This allows us to detect the structure of application communication and to present it intuitively to application developers.

Projection ($H \rightarrow A$): Figure 9 shows two automatically detected communication configurations in QBall. These patterns arise from the behavior of the SCALAPACK and BLACS libraries that QBall uses to do its computation. In the figure, the data is projected onto QBall's application domain, which is a matrix of particle wave functions where each column contains the full set of coefficients for a particle.

Evaluation and Results: Unlike traditional molecular dynamics, where performance and communication have a strong spatial correlation with the application domain, QBall's behavior more precisely mimics its solvers.

To better optimize solver performance, in addition to the communicator structure we show here, we can also record, per unique communicator set, a profile of the amount of time spent in particular MPI operations within these configurations (e.g., the rows or the columns). We have omitted these profiles due to space restrictions, but this information is particularly useful for optimizing the node layout of QBall, as it tells us not only the communication neighborhoods, but also what type of communication is typically executed within these neighborhoods.

VI. Conclusions and Future Work

This paper presents the *HAC* model, which provides a structured approach to leverage the multi-domain nature of performance data. By collecting performance data in three key domains and describing projections of data between them, our framework provides new intuitive ways to analyze and visualize performance data. By extending and combining maps and analysis techniques in different domains, we showed that the *HAC* model enables users to differentiate between application-specific and system-specific performance problems, to detect hot links in network profiles, and to automatically detect communication patterns for use in node mapping optimizations.

We have described a structured framework to guide future research and highlight underdeveloped or missing components of a complete performance analysis toolkit. We are currently working to develop new types of scalable, automated analysis techniques using the foundational techniques presented here.

Acknowledgements

Part of 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-CONF-476091).

References

[1] A. H. Baker, T. Gamblin, M. Schulz, and U. M. Yang. Challenges of Scaling Algebraic Multigrid across Modern Multicore Architectures. In *25th IEEE Parallel and Distributed Processing Symposium*, Anchorage, AK, May 2011.

[2] N. Bhatia, F. Song, F. Wolf, J. Dongarra, B. Mohr, and S. Moore. Automatic experimental analysis of communication patterns in virtual topologies, Sept. 2005.

[3] L. Blackford, J. Choi, A. Cleary, E. D¡D5¿Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walke, and R. Whaley. ScaLAPACK User's Guide. SIAM, Philadelphia, 1997.

[4] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for sparse matrix equations. In D. J. Evans, editor, *Sparsity and its Applications*, pages 257–284, Cambridge, 1984. Cambridge University Press.

[5] H. Brunst, D. Kranzlmüller, and W. Nagel. Tools for Scalable Parallel Program Analysis - Vampir NG and DeWiz. *The International Series in Engineering and Computer Science, Distributed and Parallel Systems*, 777:92–102, 2005.

[6] W. Cabot and A. Cook. Reynolds number effects on Rayleigh-Taylor instability with possible implications for t ype-Ia supernovae. *Nature Physics*, 2:562–568, Aug. 2006.

[7] A. Cook, W. Cabot, M. Welcome, P. Williams, B. Miller, B. de Supinski, and R. Yates. Tera-scalable Algorithmms for Variable-Density Elliptic Hydrodynamics with Spectral Accuracy. In *Proceedings of IEEE/ACM Supercomputing '05*, Nov. 2005.

[8] R. Falgout, J. Jones, and U. Yang. The design and implementation of hypre, a library of parallel high performance preconditioners. In A. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume 51, pages 267–294. Springer-Verlag, 2006.

[9] M. Frigo and S. G. Johnson. FFTW for version 3.0. http://www.fftw.org/fftw3.pdf, 2003.

[10] H. Gahvari, A. Baker, M. Schulz, U. M. Yang, K. Jordan, and W. Gropp. Scalable Fine-grained Call Path Tracing. In *Proceedings of the International Conference on Supercomputing*, June 2011.

[11] T. Gamblin, B. R. de Supinski, M. Schulz, R. J. Fowler, and D. A. Reed. Scalable load-balance measurement for SPMD codes. In *Supercomputing 2008 (SC'08)*, pages 46–57, Austin, Texas, November 15-21 2008.

[12] F. Gygi. Architecture of Qbox: A scalable first-principles molecular dynamics code. *IBM Journal of Research and Development*, 52, January/March 2008.

[13] F. Gygi, E. Draeger, B. de Supinski, R. Yates, F. Franchetti, S. Kral, J. Lorenz, C. Überhuber, J. Gunnels, and J. Sexton. Large-Scale First-Principles Molecular Dynamics simulations on the BlueGene/L Platform using the Qbox code. In *Proceedings of IEEE/ACM Supercomputing '05*, Nov. 2005.

[14] F. Gygi, E. Draeger, M. Schulz, B. de Supinski, J. Gunnels, V. . Austel, J. Sexton, F. Franchetti, S. Kral, J. Lorenz, and C. Überhuber. Large-Scale Electronic Structure Calculations of High-Z Metals on the BlueGene/L P latform. In *Proceedings of IEEE/ACM Supercomputing '06*, Nov. 2006.

[15] A. Gyulassy, P.-T. Bremer, V. Pascucci, and B. Hamann. A practical approach to Morse-Smale complex computation: Scalability and generality. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1619–1626, 2008.

[16] E. R. Hawkes and J. H. Chen. Direct numerical simulation of hydrogen-enriched lean premixed methane–air flames. *Combustion and Flame*, 138:242–258, 2004.

[17] K. Huck and A. Malony. PerfExplorer: A Performance Data Mining Framework For Large-Scale Parallel Computing. In *Proceedings of IEEE/ACM Supercomputing '05*, Nov. 2005.

[18] H. Jaenicke, A. Wiebel, G. Scheuermann, and W. Kollmann. Multifield visualization using local statistical complexity. *IEEE Transactions on Visualization and Computer Graphics*, 13:1384–1391, 2007.

[19] D. Laney, P.-T. Bremer, A. Mascarenhas, P. Miller, and V. Pascucci. Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities. *IEEE Trans. Vis. and Comp. Graphics*, 12(5):1052–1060, 2006.

[20] Lawrence Livermore National Laboratory. NNSA awards IBM contract to build next generation supercomputer. Press Release, available at https://publicaffairs.llnl.gov/news/news_releases/2009/NR-09-02-01.html, Feb. 2009.

[21] A. Mamidala, R. Kumar, D. De, and D. Panda. MPI Collectives on Modern Multicore Clusters: Performance Optimizations and Communication Characteristics. In *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 130–137, Washington, DC, USA, 2008. IEEE Computer Society.

[22] J. Mellor-Crummey, R. Fowler, and G. Marin. HPCView: A tool for top-down analysis of node performance. *The Journal of Supercomputing*, 23:81–101, 2002.

[23] B. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. Irvin, K. Karavanic, K. Kunchithapadam, and T. Newhall. The Paradyn Parallel Performance Measurement Tool. *IEEE Computer*, 28(11):37–46, Nov. 1995.

[24] B. Mohr and F. Wolf. KOJAK - A Tool Set for Automatic Performance Analysis of Parallel Programs. In *Proceedings of the International Conference on Parallel and Distributed Computing (Euro-Par 2003)*, pages 1301–1304, Aug. 2003.

[25] P. J. Mucci, S. Browne, C. Deane, and G. Ho. PAPI: A portable interface to hardware performance counters. In *Proc. Department of Defense HPCMP User Group Conference*, June 1999.

[26] W. E. Nagel, A. Arnold, M. Weber, H. C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, 1996.

[27] L. D. Rose, Y. Zhang, and D. A. Reed. Svpablo: A multi-language performance analysis system, Sept. 1999.

[28] J. Ruge and K. Stüben. Algebraic multigrid (AMG). In S. McCormick, editor, *Multigrid Methods*, volume 3 of *Frontiers in Applied Mathematics*. SIAM, 1987.

[29] D. Schneider, A. Wiebel, H. Carr, M. Hlawitschka, and G. Scheuermann. Interactive comparison of scalar fields based on largest contours with applications to flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14:1475–1482, November 2008.

[30] M. Schulz, A. Cook, W. Cabot, B. de Supinski, and W. Krauss. On the Performance of the Miranda CFD code on Multicore Architectures. In *Proceedings of ParCFD*, May 2009.

[31] M. Schulz and B. R. de Supinski. $P^N MPI$ Tools: a Whole Lot Greater than the Sum of their Parts. In *Proceedings of SC07*, 2007.

[32] M. Schulz, J. Galarowicz, D. Maghrak, W. Hachfeld, D. Montoya, and S. Cranford. Open|SpeedShop: An open source infrastructure for parallel performance analysis. *Scientific Programming*, 16(2-3):105–121, 2008.

[33] K. Stüben. An introduction to algebraic multigrid. In U. Trottenberg, C. Oosterlee, and A. Schüller, editors, *Multigrid*, pages 413–532. Academic Press, London, 2001.

[34] N. Tallent, J. Mellor-Crummey, M. Franco, R. Landrum, and L. Adhianto. Scalable Fine-grained Call Path Tracing. In *Proceedings of the International Conference on Supercomputing*, June 2011.

[35] N. R. Tallent, L. Adhianto, and J. M. Mellor-Crummey. Scalable identification of load imbalance in parallel executions using call path profiles. In *Proceedings of IEEE/ACM Supercomputing '10*, Nov. 2010.

[36] N. R. Tallent, J. M. Mellor-Crummey, L. Adhianto, M. W. Fagan, and M. Krentel. Diagnosing performance bottlenecks in emerging petascale applications. In *Proceedings of IEEE/ACM Supercomputing '09*, Nov. 2011.

[37] A. Wissink, R. Hornung, S. Kohn, S. Smith, and N. Elliott. Large scale parallel structured AMR calculations using the SAMRAI framework. In *Proceedings of IEEE/ACM Supercomputing '01*, Nov. 2001.

[38] F. Wolf, B. Wylie, E. Abraham, D. Becker, W. Frings, K. Fuerlinger, M. Geimer, M.-A. Hermanns, B. Mohr, S. Moore, and Z. Szebenyi. Usage of the SCALASCA Toolset for Scalable Performance Analysis of Large-Scale Parallel Applications. In *Proceedings of the 2nd HLRS Parallel Tools Workshop*, Stuttgart, Germany, july 2008.