Vol.2011-HPC-130 No.51
2011/7/28

Intuitive Performance Visualization Techniques for Topological Analysis on Capability Machines

Todd Gamblin,[†1] Martin Schulz,[†1] Timo Bremer,[†1]
Joshua A. Levine[†2] and Valerio Pascucci[†2]

The largest modern supercomputers are increasingly built using high-diameter networks with scalable topologies such as meshes and tori. These networks are cost effective in that they reduce the cost of switching hardware and are easily expandable. However, these topologies also make application performance sensitive to the placement of application processes on the physical network. Existing techniques to optimize applications for networks such as these do not take complex internal communication structure or application phases into account. However, these features could significantly reduce the complexity of the search for an optimal mapping. Moreover, they allow for more intuitive attribution of performance data to application constructs.

This paper describes a communication measurement framework to record performance data pertaining to application phases and application communication structure using instrumentation provided by application developers. We show that visualizing data collected using our framework can provide valuable insights into the performance of large-scale application codes, and that such data may eventually be used to guide automatic node-mapping for future petascale and exascale applications.

1. Introduction

Modern supercomputing systems have high-diameter interconnection networks connecting tens to hundreds of thousands of nodes. To build networks of this size affordably, hardware vendors have adopted scalable network topologies such as meshes and tori. Compared to commodity network topologies, such as fat trees, this reduces the cost of switching hardware and allows the network to be expandable. However, these topologies also increase the network diameter, and they make application communication performance sensitive to the layout of application processes on the physical network.

Finding an optimal node mapping for a given interconnection is an NP-complete problem, and it has been studied extensively since the early days of parallel, distributed-memory computers[6)–8),13)]. Recently, the trend towards high diameter networks and the trend towards on-node NUMA communication topologies has once again brought the node mapping problem to the forefront of high performance computing, and several models have been devised for application node mapping[3)–5),11)].

Several key problems remain to be addressed:

(1) Existing node mapping techniques typically use application-agnostic heuristics based on a simple communication graph, and they attempt to globally optimize this structure[11)]. Many HPC applications have distinct communication phases, and optimizing an aggregate communication graph overestimates the necessary bandwidth because not all phases' communication takes place at the same time.

(2) Solving the general graph problem does not take advantage of communication structure that may be known *a priori* by the application developers, further limiting potential optimizations.

(3) While machines such as the IBM BlueGene series present complete, regular meshes/tori to the application, other large supercomputers do not not necessarily give each job a complete mesh. On these machines, there may be holes in the network topology, or the application may be given an irregularly shaped partition, further complicating the mapping problem.

(4) As we move towards larger petascale and exascale machines, vendors are beginning to introduce networks with more complex topologies. Machines such as the newly released K supercomputer[1)] and the soon-to-be-released IBM BlueGene/Q use higher-dimensional mesh/tori to reduce diameter, but mapping applications to these topologies is not well understood.

The first two issues are measurement-related. Attempts to address the node mapping problem have measured running applications, but they have done so across entire runs, aggregating communication that occurs in separate phases. Real applications often have many distinct communication phases and commu-

†1 Center for Applied Scientific Computing, Lawrence Livermore National Laboratory
7000 East Avenue, Livermore, CA, USA 94550
†2 Scientific Computing & Imaging Institute (SCI), University of Utah
Salt Lake City, UT, USA 84112

nication structures, and developers must expose these, to allow tools to measure and present network performance data in an intuitive manner. The third and fourth issues are analysis problems. Application developers have devised "simple" schemes to map their applications onto regular topologies, but mapping onto high-dimensional, possibly incomplete meshes will require automated, dynamic analysis to achieve optimal performance.

This paper describes measurement tools and visualization techniques that we are currently developing to help application developers understand how their applications are affected by the above problems. We have developed:

(1) a framework to measure and display application-semantic communication on the physical network topology, enabling intuitive visualization of network performance data across phases and communication structures;
(2) techniques to predict network bandwidth consumption and to visualize hot links in the network for poorly laid out applications;
(3) visualization techniques using our tool data that display particular application communication patterns in their host network domain, allowing us to differentiate between slow and fast communicators; and
(4) we are currently developing techniques for more detailed attribution and scalable collection of communication performance data.

The remainder of this paper is organized as follows. In Section 2 we discuss our modular measurement framework and how we collect network measurements from running MPI applications. Section 3.1 describes how this instrumentation is applied to the AMG solver code and shows clearly how it aids in visualizing phases. Section 3.2 describes our instrumentation of the pF3D Code to make it aware of network topologies. Finally, in Section 4 we state our conclusions and outline our plans for finer-grained attribution of performance to application data.

2. Application Semantic Measurement

Traditional performance analysis tools consider performance measurements in terms of processes, source code, and sometimes time. Communication profilers and tracers such as mpiP[18] and Vampir NG[12] attribute time spent in communication routines to particular call sites and processes within the MPI rank domain. While this approach is general and can be applied easily to any MPI application, such tools do not allow the user to attribute performance measurements to particular aspects of application structure or particular parts of the network hardware. This loses potential insight into communication costs and congestion on the network. Figure 1 shows a trace collected from LLNL's Algebraic Multigrid (AMG) solver. Communication patterns are visible, but it is not clear how they relate to the network hardware, because only the MPI rank space is shown on the left axis. We are developing new measurement schemes that attempt to address these problems.

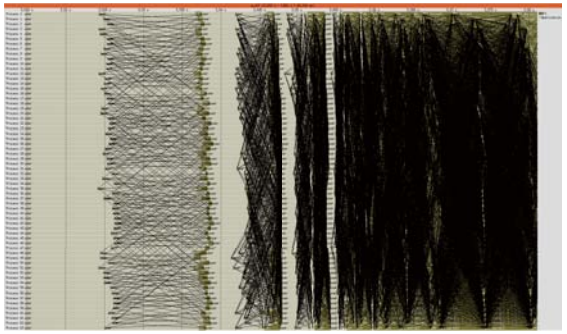


Fig. 1 Vampir trace showing communication as lines between MPI ranks.

In accordance with commonly available performance tools, existing approaches typically approach the node mapping problem using aggregate communication measurements over time. This aggregated, full-run data is then fed to graph partitioning algorithms[11),14)] or other heuristic optimizers. Aggregating communication over time can lose details about which communication patterns are temporally separate from one another, and summing many temporally disjoint communication structures into a single graph can also imply congestion where there is none. This can degrade performance by moving load away from links that are heavily used in all phases, but never saturated.

Other techniques take the application communication structure into account by attempting to use simple generalized heuristics to detect communication patterns in structured graphs[3),5)], but this approach can detect only regular patterns such as stencils in the application's communication structure. More complex communication patterns such as FFTs and multi-node collective algorithms are not yet

detectable using these schemes.

To detect the subtleties of MPI application behavior, tools must take into account underlying communication groups and analyze their behavior independently. The MPI applications that most frequently run on large-scale supercomputers often have many distinct parallel phases that execute in sequence. These may be numerical solvers, I/O schemes, or communication-intensive transform algorithms. For many applications, there are also repeated communication patterns within phases that execute concurrently on separate parts of the network. Without insight into the application, automatic detection of such behaviors is infeasible, and we can only apply a conservative set of optimization techniques on the application to optimize its communication layout.

In contrast, by analyzing and optimizing phases and repeated structures independently, we can both reduce the size of the communication graph to be analyzed (as each phase and/or structure can now be considered separately) and we can present this data more intuitively to tool users. Reducing the communication graph size reduces the number of combinatoric possibilities we must consider for node layouts, so this approach has obvious benefits for analysis. Considering phases and substructures separately also makes things more understandable for application developers. Typically, developers understand the phases of their own application, and displaying performance data in terms of a domain that they understand is intuitive.

We have found that application developers at LLNL are very willing to expose information about their application's particular phases to performance tools, especially if it results in higher-quality performance measurements. However, there is no standard mechanism for exposing phases to performance tools, and performance tool developers often work in isolation from application developers.

To address this issue, we have designed measurement tools using the PMPI tool interface provided by MPI that allow us to achieve three new types of attribution. Figure 2 shows a diagram of our framework.

At the top of the stack, we allow developers to specify semantic information inside of applications in a form accessible to tools. Working with application developers, we place `MPI_Pcontrol(int)` instrumentation calls in in application source code. These calls are then intercepted by our tools, and the single integer passed to `MPI_Pcontrol(int)` is used by the tools to determine that a new phase is beginning. We can then use this information to account for each phase separately.

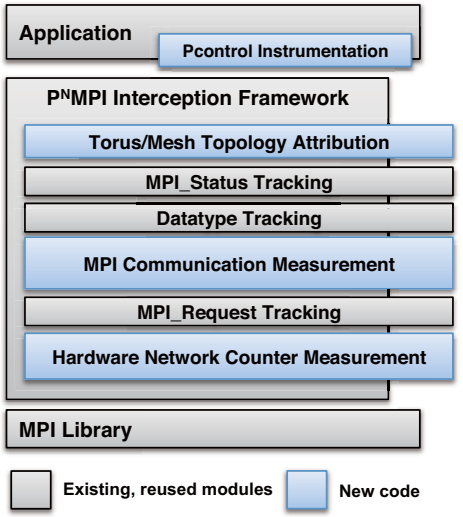


Fig. 2 P^{N}MPI tool architecture showing reused and new components.

Next, our PMPI tools track communication structures by examining MPI communicators as they are created within application. We allow performance data to be attributed to separate communicators, allowing for a more detailed accounting for total time spent executing the same code on different subgraphs of the full network.

Our tools provide a layer that allows us to attribute measurements not only to particular MPI processes but also directly to particular coordinates on the physical network. This allows us to associate communication within the application (*i.e.*, between MPI processes through messages) with actual packet data moving over the network. Where performance counters are available, we can measure the network traffic directly with a final hardware counter module.

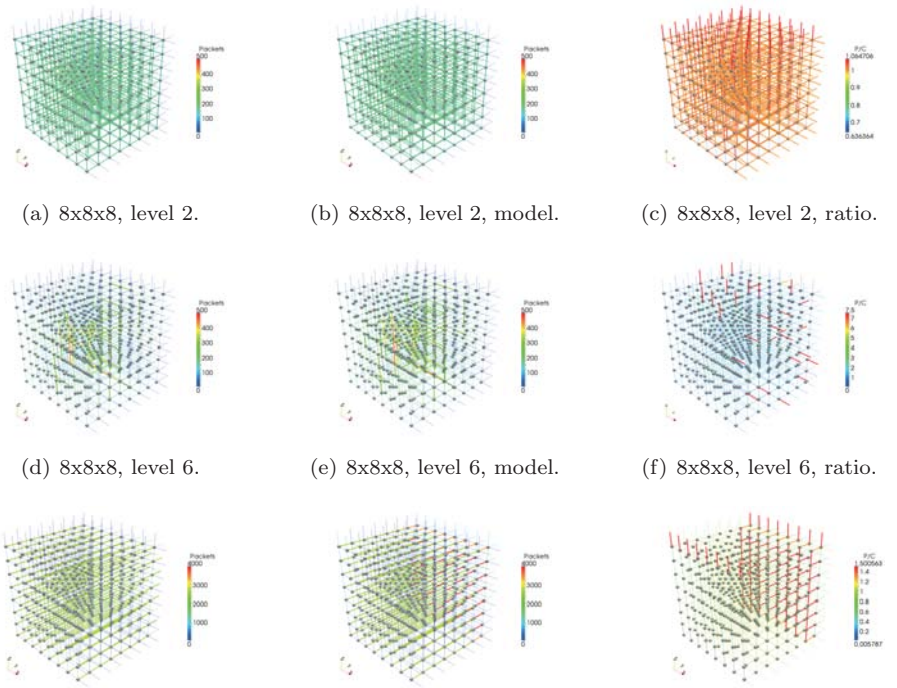
Our measurement techniques are implemented modularly using the P^{N}MPI tool virtualization environment[16]. P^{N}MPI allows multiple PMPI tools to be run

together by intercepting calls from the application to MPI and then delegating to multiple tools organized as a stack before routing the function call to MPI. Using this approach, we are able to combine a rich set of modules that collect different sets of performance data depending on the application used and the host architecture. In the next section, we detail how we have used this framework on two applications, and we detail the performance visualizations possible with our approach.

3. Case Studies

In this section, we describe the application of our measurement system to two large-scale parallel applications at LLNL. We describe the instrumentation we added to these applications and how it relates their semantics, and we show visualizations that enable us to better understand the communication behavior of these applications and its relationship with network topology on large machines. In both cases, our visualizations revealed key performance characteristics of the applications.

We use two machines in our case studies. The first is a single-rack, 4,096-processor IBM BlueGene/P system called *dawndev*. Its network has a 1,024-node torus topology, of which 512 nodes were available to us for these experiments. Each processor in the network contains 4 PowerPC 450 cores running at 850Mhz. On BlueGene/P machines, partitions of 512 or more nodes comprise a fully connected torus, and all nodes in the partition must be operational for the partition to boot. The second machine is *Cielo*, a Cray XE6 system at Los Alamos National Laboratory (LANL). Each of Cielo's nodes offers higher performance than a node of the BlueGene/P machine, with 2 8-core AMD Magny-Cours chips and a higher bandwidth network. Cielo has a total of 6,704 compute nodes in a torus/mesh configuration, for a total of 107,264 compute cores. However, the Cielo scheduler does not always give jobs regular partitions. Jobs may be allocated to irregularly shaped slabs of processors depending on what is already running on the machine. Further, the network is shared between jobs on the Cielo machine, so there can be network contention between traffic from applications running concurrently on the machine.



(a) 8x8x8, level 2. (b) 8x8x8, level 2, model. (c) 8x8x8, level 2, ratio.

(d) 8x8x8, level 6. (e) 8x8x8, level 6, model. (f) 8x8x8, level 6, ratio.

(g) 2x4x64, level 1. (h) 2x4x64, level 1, model. (i) 2x4x64, level 1, ratio.

Fig. 3 AMG network traffic measured in HW (left), MPI (middle), and MPI/HW (right).

3.1 Algebraic Multigrid Solver

Algebraic Multigrid (AMG)[15] methods are used to solve large, sparse linear systems in a number of production applications at LLNL. Here, we examine the *hypre*[9] library's BoomerAMG[10] benchmark, part of the Sequoia benchmark suite, a set of acceptance tests for the upcoming BlueGene/Q system at LLNL. This code will need to run efficiently on high-dimensional torus networks, so we are particularly interested in its performance on current capability-class machines.

The AMG solver is structurally interesting because it has multiple *levels* with very different communication patterns. Coefficients of AMG's system of equa-

tions are distributed over all processors. As the solve progresses, the coefficients are coarsened so that the next level is consolidated to a smaller set of processes. The larger the problem, the more levels of coarsening may be necessary. When a the coarse-grained solve completes, its results are interpolated back to the finer levels, and the solve progresses at the fine grain again. This process of iterative coarsening and refinement is called the *V cycle*, because the solver's progression through the levels and back resembles a 'V'.

AMG's levels can be considered as separate phases with very different communication patterns. At fine grain, AMG communication is essentially nearest-neighbor, as nodes message their immediate neighbors to get nearby fine-grained coefficients. As the data is coarsened, however, AMG's data is confined to a smaller set of processors, and communication becomes sparser. Typically, nodes have more neighbors at coarse granularity, so per-node bandwidth requirements increase.

We worked with the AMG developers to mark the code for each level of AMG's V-cycle using our phase-marking `MPI_Pcontrol` calls. We then measured communication at the MPI level by recording the total number of bytes sent over the network by each communication call intercepted by PMPI. Concurrently, we recorded the low-level BlueGene/P hardware counters. On BlueGene/P, the hardware counters give us the number of packets sent in each direction on each of the 6 torus links out of every node.

To predict network traffic from our MPI message trace data, we used a simple, congestion-free routing model, and we assumed an infinite amount of available bandwidth per link. Clearly, this is not the *correct* model for the BlueGene/P network. In reality, the links have a limited amount of bandwidth, and the machine will dynamically route packets around congestion. However, this model *will* tell us how much bandwidth would pass through each link in an ideal scenario, and we can use this in conjunction with hardware counter measurements to detect where adaptive routing is happening, and to discover "hot" links for the application. To do this, we simply divide the modeled bandwidth by the measured bandwidth, and the resulting plot clearly shows which links are bottlenecks in the MPI application. Saturated links will show higher modeled than measured bandwidth in spots where traffic has been routed away.

We ran the test AMG problem on an $8 \times 8 \times 8$ 512-process 3D torus partition of the dawndev machine. This test had 32.7 million coefficients, and the dimensions of the AMG domain decomposition wee set to $8 \times 8 \times 8$ to match the host partition on the machine. For this problem size, there are a total of 8 levels of refinement. We then ran the same problem with a $2 \times 4 \times 64$ domain decomposition projected onto the same $8 \times 8 \times 8$ torus for comparison. Figure 3 shows these results for selected levels of AMG.

In Figures 3(a-c), we see the fine-grained communication at level 2 of the AMG solver. The communication is well balanced throughout the torus, as the only communication is nearest-neighbor within the fine grid. Our traffic predictions match our measurements very closely, as we see that the value on the ratio plot is uniformly 1.

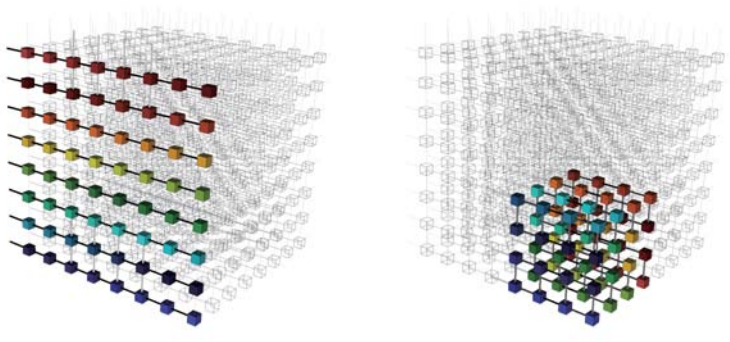
In Figures 3(d-f), we show the much coarser traffic at level 6 of AMG. At this level, the number of nodes actually communicating has become sparse. We can see this clearly on the torus, as most of the outer links have very little traffic. Traffic volume increases as we move towards the center of the partition, and we can clearly see a hot spot in the middle, along one of the vertical links within the torus. At this level our predicted bandwidth values are very close to the measured values, as well.

Finally, in Figures 3(g-i), our modeled and measured performance diverge, indicating saturated links that the machine corrected for through dynamic routing. These differences are small, but they clearly show the path that packets take in the BlueGene/P network to achieve higher bandwidth when links become saturated due to poor embedding of our problem in the host partition.

In all of these cases, application developers can clearly see where in the torus hot links are, and they can use this insight to create better node mappings for coarse levels of the AMG problem.

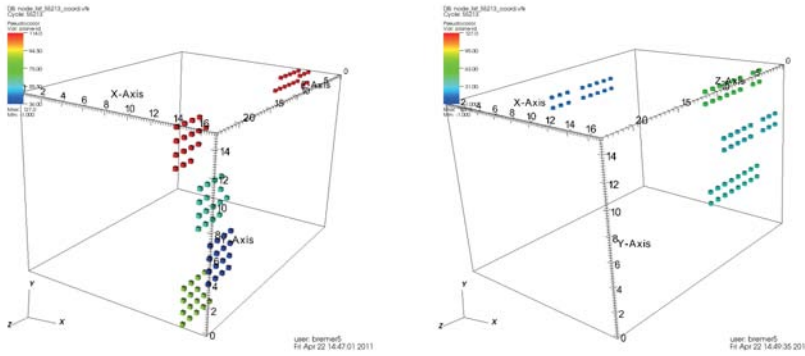
3.2 pF3D

pF3D[2),17)] is a laser-plasma interaction code used at LLNL's National Ignition Facility (NIF). The code is frequently run on hundreds of thousands of processors on BlueGene/P machines at LLNL and LANL, with excellent weak scaling. pF3D has also been run on the Cielo machine on up to 64K processes, but scaling there is not as efficient as on BlueGene machines.



(a) XYZT node mapping (b) Block mapping

Fig. 4 In (a), the torus is filled with planes but only links in a single direction will be used for the FFT. In (b), tiling smaller boxes allows use of links in multiple dimensions.



(a) Fastest Communicators (b) Slowest Communicators

Fig. 5 Layout within the Cielo network of the fastest and slowest 2D FFT planes for pF3D. Slower FFT planes to be longer and have more holes.

pF3D's simulated domain is decomposed into $x \times y \times z$ grid of processes, each of which models an equally sized chunk of the problem grid. In the simulated space, a laser runs along the z axis, and large numbers of 2D FFTs are computed within the xy planes transverse to the beam. Each 2D FFT in pF3D is entirely independent of the others, and `MPI_Alltoall` communication occurs only within these xy planes. The 2D FFTs can consume up to 30% of the runtime of pF3D, so it is important that they are mapped efficiently to the network.

On BlueGene machines, a simple layout of FFTs on the hardware maps xy planes in the simulated application domain to xy planes in the BG/P torus. Figure 4(a) shows one such plane in this configuration for the x direction of the FFT, with communicators denoted by color. Here, only links in the x dimension are used by the transform. When the slab of processes performs the y direction of the transform, the directions flip, and only the y links are used for message passing.

To better exploit the BlueGene/P network, pF3D employs another mapping scheme with each simulated plane folded into a block on the BlueGene/P network. Figure 4(b) shows this configuration. Again, communicators are highlighted by color, but with the remapping, multiple dimensions are used for communication within the transform. Instead of stacking xy planes along the z axis, blocks are tesselated within the torus.

On Cielo, static node mappings like these are not as effective because processes are not allocated in regular partitions. Rather, the lowest-ranked processors in the system according to a linear node order are assigned to jobs as they are allocated. There is thus no guarantee that all of pF3D's planes will have the same node mapping, because it is no longer possible to tesselate them within an allocation. Further, in a BlueGene/P system, we could assume that all FFTs would finish in the same amount of time due to their identical messaging rates within the torus. On Cielo, some planes may take longer than others. Since pF3D is a bulk synchronous application, this causes a load imbalance at the end of the FFT phase, and fast planes may sit idle while slow planes finish running. A simple mapping will thus not suffice for Cielo, because the shape of a job of a certain size may change on each run.

We used our visualization techniques to verify this load imbalance between differently shaped planes on the Cielo machine. We recorded the maximum time spent in the FFT phase on all processes within the same FFT plane, and we then ranked these times and plotted the shapes of the fastest and slowest FFT planes

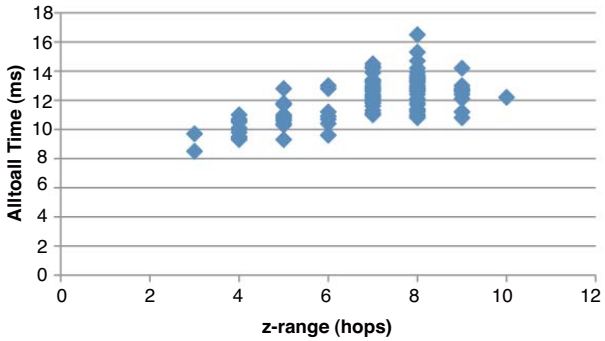


Fig. 6 Correlation of elapsed time to Z-range for FFT Alltoalls on Cielo

in their positions on the Cielo network. Figure 5(a) shows the fastest planes, differentiated by coloring, and Figure 5(b) slows the slowest FFT planes. In the figures, it is easy to see that the faster communicators have tight aspect ratios and very few holes. Conversely, the slowest FFT planes were elongated along the Z-axis, which allows fewer links to be used for all to all communication. Some of the slow planes also displayed large holes, which would serve to increase the time for messages within the plane to reach their destinations.

Having established this trend using visualization, we verified it by correlating maximum time spent in `MPI_Alltoall` with the maximum hops between any two z coordinates within the communicator. Figure 6 shows a clear positive trend between z elongation and time. The trend is more rigid among lower ranks, but there is noticeably more variability in `MPI_Alltoall` timing up to a z-range of around eight. After eight hops, `MPI_Alltoall` calls are slightly faster, but not as fast as z-ranges of six or fewer.

4. Conclusions and Future Work

We have presented a framework that allows attribution of performance data to application semantics. Our framework allows application developers to instrument their code and to gather performance data describing its communication patterns and structure. It also allows this data to be visualized on the same topology as the host machine's network, enabling developers to discover hot links and slow structures in their application's layout.

These are exploratory techniques, and we have developed these visualizations to gain understanding of a complex domain. Currently, we have two future directions. First, we are developing techniques for more fine-grained attribution of performance data. This paper presented methods for attributing communication costs and bandwidths to inter-node links in network topologies, but we would like to know more in particular about the types of communication going over those links. Specifically, we would like to attribute communication to source code regions and to parts of the application's data model, and visualize this information on the network topology. We are currently investigating techniques to preserve this information scalably and to display it intuitively. Second, we are developing techniques to automate this type of analysis so that it can be applied to high-dimensional networks on future supercomputers. This will involve taking the lessons learned from visualization of lower-dimensional topologies and developing general approaches for more complex network topologies.

Acknowledgments Part of this work was authored by a contractor of the U.S. Department of Energy under contract DE-AC52-07NA27344 (LLNL-CONF-481092). Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

References

1) Ajima, Y., Sumimoto, S. and Shimizu, T.: Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers, *Computer*, Vol.42, No.11, pp.36 –40 (online), DOI:10.1109/MC.2009.370 (2009).
2) Berger, R.L., Lasinski, B.F., Langdon, A.B., Kaiser, T.B., Afeyan, B.B., Cohen, B.I., Still, C.H. and Williams, E.A.: Influence of spatial and temporal laser beam smoothing on stimulated brillouin scattering in filamentary laser light, *Physics Review Letters*, Vol.75, No.6, pp.1078–1081 (1995).
3) Bhatele, A. and Kale, L. V.: Application-specific Topology-aware Mapping for Three Dimensional Topologies, *Proceedings of Workshop on Large-Scale Parallel Processing (IPDPS '08)* (2008).
4) Bhatelé, A. and Kalé, L.V.: Benefits of Topology Aware Mapping for Mesh Interconnects, *Parallel Processing Letters (Special issue on Large-Scale Parallel Processing)*, Vol.18, No.4, pp.549–566 (2008).
5) Bhatelé, A., Kalé, L.V. and Kumar, S.: Dynamic Topology Aware Load Balancing

Algorithms for Molecular Dynamics Applications, *23rd ACM International Conference on Supercomputing* (2009).
6) Bokhari, S.H.: On the Mapping Problem, *IEEE Trans. Computers*, Vol.30, No.3, pp.207–214 (1981).
7) Bollinger, S.W. and Midkiff, S.F.: Processor and Link Assignment in Multicomputers Using Simulated Annealing, *International Conference on Parallel Processing (ICPP)* (1988).
8) Ercal, F., Rmanujam, J. and Sadayappan, P.: Task Allocation Onto a Hypercube by Recursive Mincut Bipartitioning, *Proceedings of the 3rd Conference on Hypercube concurrent computers and applications*, ACM Press, pp.210–221 (1988).
9) Falgout, R., Jones, J. and Yang, U.: The Design and Implementation of hypre, a Library of Parallel High Performance Preconditioners, *Numerical Solution of Partial Differential Equations on Parallel Computers* (Bruaset, A. and Tveito, A., eds.), Vol.51, Springer-Verlag, pp.267–294 (2006).
10) Henson, V.E. and Yang, U.M.: BoomerAMG: a Parallel Algebraic Multigrid Solver and Preconditioner, *Applied Numerical Mathematics*, Vol.41, pp.155–177 (2000).
11) Hoefler, T. and Snir, M.: Generic Topology Mapping Strategies for Large-scale Parallel Architectures, *Proceedings of the 2011 ACM International Conference on Supercomputing (ICS'11)*, ACM, pp.75–85 (2011).
12) Knüpfer, A., Brunst, H. and Nagel, W.E.: High Performance Event Trace Visualization, *Euro Workshop on Parallel, Distributed and Network-Based Processing (PDP 2005)* (2005).
13) Lee, S.-Y. and Aggarwal, J.K.: A Mapping Strategy for Parallel Processing, *IEEE Trans. Computers*, Vol.36, No.4, pp.433–442 (1987).
14) Pellegrini, F. and Roman, J.: Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs, *High-Performance Computing and Networking*, Lecture Notes in Computer Science, Vol.1067, Springer Berlin / Heidelberg, pp.493–498 (1996).
15) Ruge, J. and Stüben, K.: Algebraic Multigrid (AMG), *Multigrid Methods* (McCormick, S., ed.), Frontiers in Applied Mathematics, Vol.3, SIAM (1987).
16) Schulz, M. and de Supinski, B.R.: P^NMPI Tools: A Whole Lot Greater Than the Sum of Their Parts, *Supercomputing 2007 (SC'07)*, Reno, NV (2007).
17) Still, C.H., Berger, R.L., Langdon, A.B., Hinkel, D.E., Suter, L.J. and Williams, E.A.: Filamentation and forward brillouin scatter of entire smoothed and aberrated laser beams, *Physics of Plasmas*, Vol.7, No.5, p.2023 (2000).
18) Vetter, J. and McCracken, M.O.: Statistical Scalability Analysis of Communication Operations in Distributed Applications, *ACM SIGPLAN Notices*, Vol.36, No.7, pp. 123–132 (2001).