

Chapter 1

Creating a Tool Set for Optimizing Topology-Aware Node Mappings

Martin Schulz, Abhinav Bhatele, Peer-Timo Bremer, Todd Gamblin,
Katherine Isaacs, Joshua A. Levine, and Valerio Pascucci

Abstract Modern HPC systems, such as Cray’s XE and IBM’s Blue Gene line, feature sophisticated network architectures, often in the form of high dimensional tori. In order to fully exploit the performance of these systems, it is necessary to carefully map an application’s communication structure to the underlying network topology. In this step, both latency (i.e., physical distance between nodes) and bandwidth (i.e., number of concurrently used links) have to be taken into account, leading to mappings that are often non-intuitive. To help developers with this complex problem, we are developing a set of tools that aim at helping users understand the communication behavior of their codes, map them onto the network architecture, and create better-performing topology-aware node mappings. In this paper, we present initial steps towards this goal, including a measurement environment capturing both communication patterns and network metrics within the same run, a methodology to compare these measurements, and a visualization tool that helps users understand the impact of their application’s characteristics on the network behavior.

1.1 Motivation and Background

Modern HPC architectures typically feature high dimensional tori as their inter-connection network. Some examples are the IBM Blue Gene line, Cray’s XT/XE systems, and Fujitsu’s K computer. Because of their constant per-node link count,

M. Schulz (✉) · A. Bhatele · P.-T. Bremer · T. Gamblin · K. Isaacs
Center for Applied Scientific Computing, Lawrence Livermore National Laboratory,
Livermore, CA 94550, USA
e-mail: schulzm@llnl.gov; bhatele@llnl.gov; bremers5@llnl.gov; tgamblin@llnl.gov;
isaacs6@llnl.gov

J.A. Levine · V. Pascucci
Scientific Computing and Imaging Institute University of Utah, Salt Lake City, UT 84112, USA
e-mail: jlevine@sci.utah.edu; pascucci@sci.utah.edu

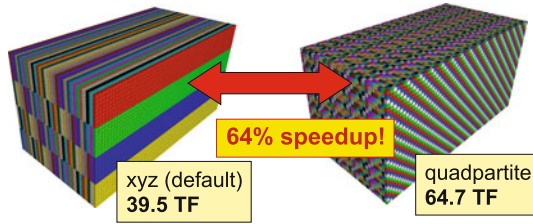


Fig. 1.1 Application performance of an FPMD application on 64 K nodes of Blue Gene/L using the default (*left*) and the optimized mapping (*right*). Blocks of the same color denote nodes assigned to the same row of the core data structure, a dense 2D matrix

these networks are relatively easy to deploy and scale. However, the resulting high network diameter creates performance and scaling problems for applications. Thus, it is critical to lay out the tasks of an application on the underlying network topology carefully, so that communication is efficient for the specific patterns the application exhibits.

The topology mapping problem is well-documented in the literature [2–5, 10, 18]. As one example, during the optimization process of a first principle molecular dynamics (FPMD) application [10], we found that the performance difference between the default and the best performing layout was 64 %. Further, the best mapping was contrary to prior expectations and non-intuitive (see Figure 1.1).

In order to enable such optimizations, we need tools that help users understand the communication patterns of their application, how they map onto the hardware topology, how this impacts performance, and ultimately provide efficient node mappings for their applications. To achieve this goal, we make use of performance data gathered in multiple independent domains to provide the user with informative perspectives on the performance of their code [15]. Further, we can map data gathered in one domain to another domain enabling us to compare and correlate the performance data and features found in different domains. In particular, we focus on data from the communication domain, representing the application’s communication pattern, and the hardware domain, in which we gather network statistics. This allows us to understand the impact of an application’s communication behavior on the network for a particular layout.

We present initial steps towards a novel tool set to explore and optimize topology-aware node mappings for HPC applications, using information mapped and correlated between multiple performance domains. We make the following contributions:

- We show the use of a three-domain model of performance data for studying node mappings in HPC systems.
- We present a flexible measurement tool set that can be used to concurrently track data from multiple measurement domains.
- We introduce a technique to correlate network measurements with application communication behavior.

- We show preliminary results of an Algebraic Multigrid (AMG) solver.

The remainder of this paper is organized as follows: Sect. 1.2 introduces the underlying domain model as well as the necessary data mappings. Section 1.3 discusses how we gather and interpret the data, Sect. 1.4 shows first preliminary results, Sect. 1.5 briefly discusses related work, and Sect. 1.6 contains concluding remarks and an overview of future work.

1.2 Gaining Insight from Multiple Perspectives

Performance data can be gathered and analyzed in multiple domains. In previous work [15], we provided a structured approach for the three most relevant performance data domains. In this section, we briefly introduce this model and show how it applies to the node mapping problem.

1.2.1 The HAC Model

Figure 1.2 shows the basic concept behind our domain model, the *HAC* model. We identify the three most relevant domains for performance analysis, namely the application, the hardware, and the communication domain. The application domain is used by the application scientist to express the actual problem being computed. For scientific simulations it often represents some physical simulation space on which the problem is defined, although more abstract representations are also possible (e.g., for graph algorithms or sparse matrix representations in solver libraries). The hardware domain expresses the structure of the underlying architecture, e.g., a torus with multicore nodes and is often used for hardware related measurements, such as hardware counters. The communication domain is used to express the code's communication pattern, often represented as a graph with nodes representing application tasks (e.g., MPI processes) and edges showing communication between nodes.

For each of the three domains we can use specialized analysis techniques that allow us to extract features from the respective data sets. Additionally, we can define mappings for data between domains allowing us to compare data from multiple domains for further analysis. For example, we can map performance data gathered in the hardware domain, such as hardware counter data, into the application domain and display it concurrently with simulated features from the application. In our previous work [15] we have demonstrated that this can show correlations between simulated features and measured performance data allowing us a clear attribution of performance data and establishing clear baselines.

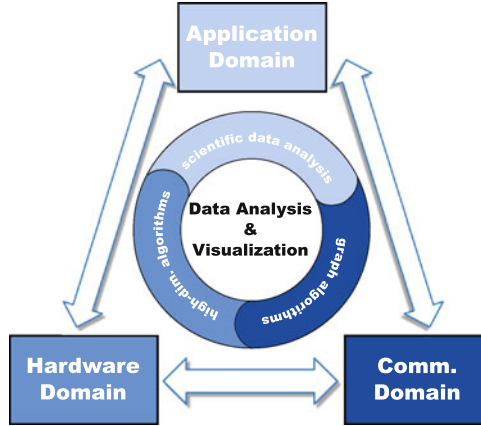


Fig. 1.2 Interplay between the three critical domains for performance analysis

1.2.2 Domains Relevant for Node Mappings

For the node mapping problem, we focus on two of the three domains: the communication domain showing an application's communication behavior and the hardware domain showing its network behavior. By contrasting the data from these two domains we can analyze the impact of the application's characteristics on the network behavior and gain the necessary insight to optimize performance.

In the communication domain we are interested in all communication executed by the application, in our case in the form of MPI messages. Conceptually, we gather the full communication graph, although in reality we restrict ourselves to a statistical summary of the data which is sufficient for our purposes. In the hardware domain we must acquire the data showing the actual communication executed in the system. For this work we focus first on Blue Gene systems, which provide a set of performance counters to monitor the actual number of packets exchanged on each hardware link in the system.

1.2.3 Mapping Communication to Hardware Data

We must establish a mapping between the two domains in order to allow a direct comparison of hardware and communication data. This mapping has two components: we first map MPI processes to physical nodes in the hardware domain. This is the actual node mapping we are aiming to optimize and hence we will evaluate how changing this mapping affects performance.

Second, for a concrete node mapping, we map the performance data from one domain to the other. Ideally, we would map the hardware domain data to the

communication domain, since this is the most relevant for the user. Unfortunately, though, this is impossible because each hardware link is potentially used by several MPI communication pairs, since links are shared for all packets passing through the connected nodes. Therefore, we cannot cleanly correlate the hardware data back to individual MPI message events.

We therefore reverse the mapping and map the MPI communication behavior to the network architecture. We do this by mapping each message individually, assuming no contention, interference, or bandwidth limits, and with that we emulate the hardware behavior under ideal load. As a result, we gain a best case view of how the application's communication under the given node mapping behaves on the system. We can contrast this best case estimate with the actual measurements showing where behavior deviates from the best case estimate and hence where bottlenecks manifest themselves during execution.

1.3 Creating a Flexible Measurement Environment

Our approach requires the ability to gather data from multiple domains concurrently, since only this keeps the data comparable and avoids cross-execution variations. We implement this on top of the P^N MPI infrastructure, and develop a series of plugin modules for the actual data acquisition. We further define structured, self-explanatory data formats to store the data for post-processing in our analysis and visualization tools.

1.3.1 Concurrent Measurements Using P^N MPI

As most tools targeting MPI, we rely on the MPI Profiling Interface (PMPI), which allows tools to transparently intercept invocations to MPI routines and with that to establish wrappers around MPI calls to gather execution information. However, the usage of this interface is limited to a single tool. We therefore rely on the P^N MPI infrastructure [14], which virtualizes this interface and enables the concurrent execution of an arbitrary number of PMPI tools, in the extreme case even without changes to the tool or the application. Figure 1.3 illustrates the principle behind P^N MPI.

In the following we refer to PMPI tools used within P^N MPI as tool modules, since they are used as plugin modules in the overall infrastructure. The modules used for a specific run are specified in an ASCII configuration file, which P^N MPI loads at startup and then uses to locate, load, and instantiate its modules.

For our experiments we implement the data acquisition in the two domains as separate P^N MPI modules and load them dynamically at runtime. Both are executed independently and can gather data in their respective domains.

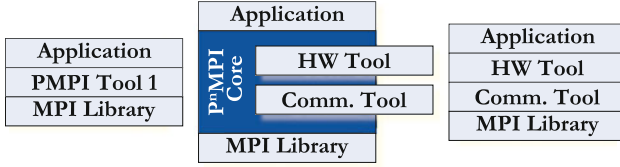


Fig. 1.3 Usage of the MPI profiling interface (*left*); combining two measurement tools using P^N MPI (*middle*); effective tool stack as seen by the application (*right*)

1.3.2 Gathering Data in the Hardware Domain

In the hardware domain we collect performance counter data from each node in the application's partition. In particular, we gather the number of packets sent in each direction of the torus from each node along with some non-network metrics such as cache misses or floating point operations. We implement this module on top of IBM's Universal Performance Counter (UPC) interface. At program termination, we gather the data to rank zero and store the result as a table with a row for each rank and a column for each hardware counter metric.

1.3.3 Gathering Data in the Communication Domain

The data in the communication domain represents the communication patterns of the application. We gather this information in the form of a communication matrix that stores aggregated information for all MPI rank pairs that exchange messages during the application's execution. In particular, we store the number of messages sent between two ranks as well as the total number of bytes. Further, we distinguish the data for three different messages sizes to capture data for different communication protocols on the Blue Gene systems. The latter is necessary, since different communication protocols lead to different network traffic and hence emulating this traffic requires separate handling for each protocol.

1.3.4 Phase Attribution

For both domains we further need to refine the data gathered by the two modules to expose phase behavior. For this we instrument the application with `MPI_Pcontrol` calls to specify application specific phase boundaries. We use the `level` argument to `MPI_Pcontrol` to indicate the ID of new phases and extend both modules to recognize these phase transitions and to annotate any performance data with the ID of the phase during which it was collected.

```

---
- !!python/tuple [mpirank, int32]
- !!python/tuple [phase, int32]
- !!python/tuple [x, int32]
- !!python/tuple [y, int32]
...
0 1 4 5
1 1 8 9
0 2 1 1
1 2 6 3

```

Fig. 1.4 Example of a YAML file (header and body) showing data for a 2D coordinates (x/y) with data for different ranks (0, 1) and application phases (with IDs 1, 2)

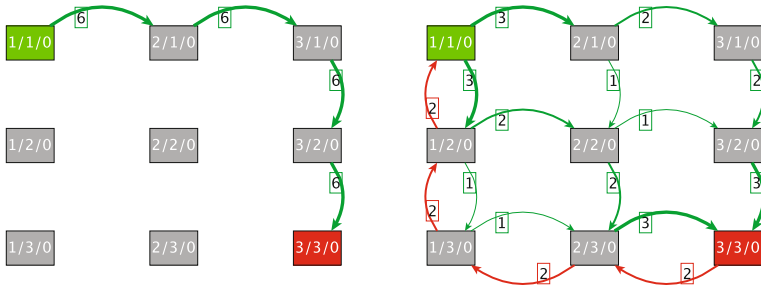


Fig. 1.5 Message traffic on a part of a plane of BG/P nodes for a short message (*left*) and a long message (*right*). *Green arrows* indicate packets sent in the positive direction of the torus network, *red arrows* packets in the negative

1.3.5 Data Storage in Structured YAML Files

The results from both modules are stored in the form of a structured file using ASCII together with a YAML header [7] describing the contents. An example is shown in Fig. 1.4. This enables our tools to interpret the data, opens our tools to multiple data sources, and allows us to implicitly document the data represented in the file.

1.3.6 Investigating Best Case Mappings

Once gathered, we need to map the data from one of the domains to another. As mentioned in Sect. 1.2.3, we do this by mapping the effect of all messages observed in the communication domain individually onto the hardware domain. To enable this, we first studied the network traffic for individual MPI messages (see Fig. 1.5) and used this data to construct an accurate model for individual messages. To capture the correct behavior we distinguish short and long messages and model them separately.

We then apply this model to every communication pair observed in the communication domain and recorded in the communication matrix during the

execution of our target application and add up the resulting number of modeled packets. This gives us an overall estimate of the total traffic in the system. While this estimate is not accurate since it does not take contention and overlapping and interfering traffic into account, it still gives a best case estimate of traffic as if the traffic was executed on a system with unbounded communication capabilities. Any difference between this estimate and the actual observed traffic is therefore caused by system limitations or bottlenecks and therefore gives an indication of problems with the underlying node mappings.

1.3.7 *Approximating Collectives*

The current approach only covers point to point messages, since we can isolate their behavior on the network. Collective operations are implemented as a set of point to point messages themselves, but the exact algorithm used depends on both the underlying network topology—and with that the node mapping—and the MPI implementation. In the current version of our tool set we use an idealized model to represent collectives showing the effectively communicated data rather than the data exchanges in a specific MPI implementation. While this is sufficient for most applications, it has the potential of decreasing the estimate’s accuracy. We are currently working on extending the approach using probabilistic models for collectives to avoid this drawback.

1.4 Preliminary Results

To demonstrate our techniques, we apply them to an Algebraic Multigrid (AMG) solver from the hypre library [8]. This codes solves a system of sparse linear equations using an iterative approach and operates on a V cycle, which means it starts with a fine grid, coarsens it through multiple levels until it can be solved directly, and interpolates the coarse grain solution back to the fine grain grid. This process is then repeated until the system converges to a stable solution.

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 [9].

We execute the AMG code on 512 nodes of a Blue Gene/P machine at LLNL, which are organized as a $8 \times 8 \times 8$ torus. We extract both communication and hardware counter data using our measurement infrastructure. The top row of graphs in Fig. 1.6 shows the measured link activity visualized on top of the underlying physical 3D torus structure for two node mappings and for specific AMG levels; the middle row of graphs shows the communication data mapped into the hardware domain using our best case network emulation approach introduced above, and

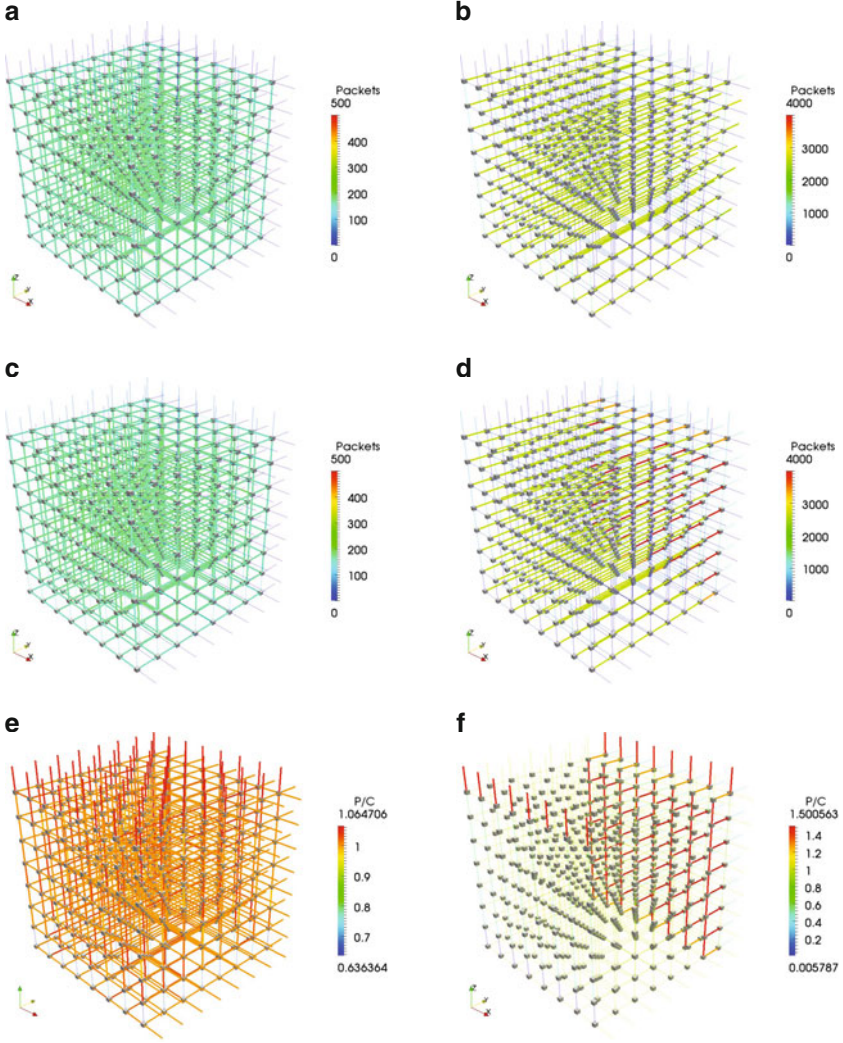


Fig. 1.6 Measured vs. emulated network traffic for two node mappings of AMG on 512 nodes of BG/P. (a) $8 \times 8 \times 8$, level 2, counter data. (b) $2 \times 4 \times 16$, level 1, counter data. (c) $8 \times 8 \times 8$, level 2, emulated data. (d) $2 \times 4 \times 16$, level 1, emulated data. (e) $8 \times 8 \times 8$, level 2, emulated/counter. (f) $2 \times 4 \times 16$, level 1, emulated/counter

the bottom row shows the ratio between the emulated data and the hardware measurements.

The left column of the data shows results from a run in which the data set is decomposed into an $8 \times 8 \times 8$ problem and hence matches the underlying topology. We can clearly see that the measured and estimated data are close to each other, indicating a mostly bottleneck free execution. However, looking at the ratio graph,

we can see a roughly 7% degradation of the actual measurement compared to the ideal best case model for links in the Z direction, while the measurements in the X and Y direction don't show a significant difference (red vs. orange links). This asymmetry is unexpected since neither the application nor the system favor any of the three dimensions.

The right column of the Fig. 1.6 shows a different picture. Here, the input problem is decomposed into $2 \times 4 \times 16$ blocks and hence does not match the underlying hardware topology. As a consequence, measured and emulated best case data no longer match up indicating messaging bottlenecks in the system. In particular, we see a significant difference on the right side of the torus, as indicated by the red links in Fig. 1.6f.

Overall, our measurement, analysis, and visualization approach enables users to understand the behavior of their code under different node mappings and based on that optimize the node mappings.

1.5 Related Work

A significant body of work on optimizing node mappings exists from the 1980s, which was targeted towards topologies such as hypercubes and two-dimensional grids [1, 6, 12]. Recent work has focused on three-dimensional tori, a popular topology for current supercomputers, and includes techniques to optimize communication for specific applications [4, 5, 10] as well as general frameworks [2, 3, 18]. However, there are few existing tools to map communication to hardware data and to visualize the mapping of tasks on processors.

Individual application developers have developed or exploited existing visualization frameworks such as ParaView to visualize task mapping on the physical network. SCALASCA [17] and its predecessor Kojak [13] feature a data browser that presents projections of performance data onto the physical network topology of supercomputers. The TAU tool set includes support to specify and visualize mappings within the hardware space using a simple mapping language [16], but it fails to exploit inter-domain mappings as we do with the *HAC* model. The Projections visualization framework [11] has capabilities to show the average number of hops/links that messages originating from a processor travel on a 3D torus/mesh. However, it does not have 3D views to visualize this data.

1.6 Conclusions and Future Work

Optimizing the mapping of application processes to nodes in the physical topology of a machine is a critical performance optimization step on today's HPC systems. Without it, applications might pay high messaging costs both in terms of latency and bandwidth. In this work, we presented an initial step towards a general tool set for

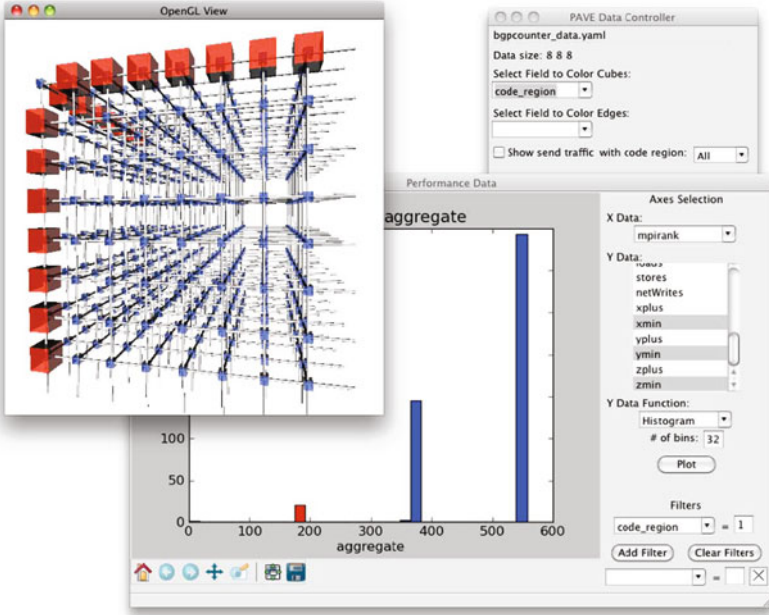


Fig. 1.7 Screenshot of the Boxfish tool showing the data control panel (*right*), statistical data on link usage (*middle*), and the 3D torus of Blue Gene/P with the nodes highlighted that correspond to the selected statistical data (*left*)

optimizing these mappings. Our approach enables users to correlate communication data with actual observations on the system leading to the detection of bottlenecks in the communication subsystem. We introduced the base measurement infrastructure and explored the overall approach using an Algebraic Multigrid solver as a case study.

We are currently in the process of extending this work in several directions. We are working on additional application studies using a 3D laser-plasma interaction code at LLNL as well as an adaptive mesh refinement library. In both cases, the communication, and consequently the node mappings, have been identified as one of the critical bottlenecks. We will study these codes both in a static scenario, where the partition size and shape is known a priori, and in a dynamic scenario, where the scheduling system decides on the partition shape at job startup. Further, we are currently developing a flexible data analysis and visualization tool, Boxfish (see Fig. 1.7), that will enable users to interactively analyze their codes' behavior and will guide mapping optimizations.

Acknowledgements 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 (LLNL-CONF-508551).

References

1. Aleliunas, R., Rosenberg, A.L.: On embedding rectangular grids in square grids. *IEEE Trans. Comput.* **31**(9), 907–913 (1982)
2. Bhanot, G., Gara, A., Heidelberger, P., Lawless, E., Sexton, J.C., Walkup, R.: Optimizing task layout on the Blue Gene/L supercomputer. *IBM J. Res. Dev.* **49**(2/3), 489–500 (2005)
3. Bhatele, A.: Automating topology aware mapping for supercomputers. Ph.D. thesis, Department of Computer Science, University of Illinois. <http://hdl.handle.net/2142/16578> (2010)
4. Bhatele, A., Bohm, E., Kale, L.V.: Optimizing communication for charm++ applications by reducing network contention. *Concurr. Comput. Pract. Exp.* **23**(2), 211–222 (2011)
5. Bhatele, A., Kalé, L.V., Kumar, S.: Dynamic topology aware load balancing algorithms for molecular dynamics applications. In: 23rd ACM International Conference on Supercomputing, Yorktown Heights, NY (2009)
6. Bokhari, S.H.: On the mapping problem. *IEEE Trans. Comput.* **30**(3), 207–214 (1981)
7. Evans, C.C.: The official YAML web site. <http://yaml.org/> (2011)
8. Falgout, R., Yang, U.: Hypre: a library of high performance preconditioners. In: Proceedings of the International Conference on Computational Science (ICCS), Amsterdam, The Netherlands. Part III, Lecture Notes in Computer Science, vol. 2331, pp. 632–641 (2002)
9. Gahvari, H., Baker, A., Schulz, M., Yang, U.M., Jordan, K., Gropp, W.: Scalable fine-grained call path tracing. In: Proceedings of the International Conference on Supercomputing, Tucson, AZ (2011)
10. Gygi, F., Draeger, E., Schulz, M., de Supinski, B., Gunnels, J., Austel, V., Sexton, J., Franchetti, F., Kral, S., Lorenz, J., Überhuber, C.: Large-scale electronic structure calculations of high-Z metals on the BlueGene/L platform. In: Proceedings of IEEE/ACM Supercomputing 2006 (SC06), Tamp, FL (2006)
11. Kale, L.V., Zheng, G., Lee, C.W., Kumar, S.: Scaling applications to massively parallel machines using projections performance analysis tool. *Future Gener. Comput. Syst.* **22**, 347–358 (2006). Special issue on: Large-scale system performance modeling and analysis
12. Lee, S.-Y., Aggarwal, J.K.: A mapping strategy for parallel processing. *IEEE Trans. Comput.* **36**(4), 433–442 (1987)
13. Mohr, B., Wolf, F.: 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), Klagenfurt, Austria, pp. 1301–1304 (2003)
14. Schulz, M., de Supinski, B.R.: P^N MPI Tools: a whole lot greater than the sum of their parts. In: Proceedings of Supercomputing 2007 (SC07), Reno, NV (2007)
15. Schulz, M., Levine, J.A., Bremer, P.T., Gamblin, T., Pascucci, V.: Interpreting performance data across intuitive domains. In: International Conference on Parallel Processing (ICPP), Taipei City, Taiwan (2011)
16. Spear, W., Malony, A.D., Lee, C.W., Biersdorff, S., Shende, S.: An approach to creating performance visualizations in a parallel profile analysis tool. In: Workshop on Productivity and Performance (PROPER), Bordeaux, France (2011)
17. Wolf, F., Wylie, B., Abraham, E., Becker, D., Frings, W., Fuerlinger, K., Geimer, M., Hermanns, M.A., Mohr, B., Moore, S., Szebenyi, Z.: Usage of the SCALASCA toolset for scalable performance analysis of large-scale parallel applications. In: Proceedings of the 2nd HLRS Parallel Tools Workshop, Stuttgart. http://www.cs.utk.edu/~karl/research/pubs/WOLF-2008_Scalasca.pdf (2008)
18. Yu, H., Chung, I.H., Moreira, J.: Topology mapping for blue Gene/L supercomputer. In: SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, p. 116. ACM, New York (2006). <http://doi.acm.org/10.1145/1188455.1188576>