Towards Scalable Performance Analysis and Visualization through Data Reduction

Chee Wai Lee, Celso Mendes and Laxmikant V. Kalé
Department of Computer Science
University of Illinois at Urbana-Champaign
cheelee@uiuc.edu, cmendes@uiuc.edu, kale@cs.uiuc.edu

Abstract

Performance analysis tools based on event tracing are important for understanding the complex computational activities and communication patterns in high performance applications. The purpose of these tools is to help applications scale well to large numbers of processors. However, the tools themselves have to be scalable. As application problem sizes grow larger to exploit larger machines, the volume of performance trace data generated becomes unmanagable especially as we scale to tens of thousands of processors. Simultaneously, at analysis time, the amount of information that has to be presented to a human analyst can also become overwhelming.

This paper investigates the effectiveness of employing heuristics and clustering techniques in a scalability framework to determine a subset of processors whose detailed event traces should be retained. It is a form of compression where we retain information from processors with high signal content.

We quantify the reduction in the volume of performance trace data generated by NAMD*, a molecular dynamics simulation application implemented using* CHARM++*. We show that, for the known performance problem of poor application grainsize, the quality of the trace data preserved by this approach is sufficient to highlight the problem.*

1 Introduction

The next few years will see the availability of several very large machines with many thousands of processors. With these resources available, application developers are likely to have two desires. The first is the desire to take advantage of the larger total available memory and computational power to solve larger problems. The second is the desire to attempt to solve the same problems faster, by running them on more processors.

In both cases, it is important that parallel performance tools be available to study the application's performance as it is scaled to larger numbers of processors. Unfortunately, as we will show, the already large volume of performance data also grows with scaling, particularly for weak-scaling where larger problems are being solved. As a result, it is important that the parallel performance tools themselves become scalable in the face of increased volume of traced data.

In this paper, we focus our investigation on the approach of reducing the volume of event-based performance data retained for post-mortem analysis. While aggregated or sampled profile data can be useful, the use of detailed event traces from instrumented parallel applications has been documented to be a valuable source of information for studying their performance [11, 25, 16]. Our approach takes advantage of the fact that performance data are recorded in per-processor buffers in memory. We propose the retention of, by writing out to disk, only a subset of these per-processor buffers through a careful selection criterion applied online, making use of the parallel machine after the application has completed.

This selection criterion is centered on the use of heuristics for the identification of the representative and outliers within equivalence classes of processors discovered through clustering algorithms. We will show in this paper that an online application of the approach after an application's execution can generate much less data and yet capture its core performance characteristics in high detail.

We have implemented our approach to enhance the scalability of *Projections*, a performance tracing, visualization and analysis tool used for analyzing the performance of CHARM++ and ADAPTIVE MPI applications. Our experiments examine the approach's impact on performance logs generated for the popular molecular dynamics application NAMD which is written in CHARM++. Projections instrumentation is automatic by default, tracking task execution and communication events in the CHARM++ runtime. Event traces are generated and visualized/analyzed post-mortem through a visualization component. The experiments were conducted on the Cray XT3 supercomputer installed at Pittsburgh Supercomputing Center using three NAMD simulation benchmarks instrumented for over 200 simulation timesteps.

The organization of the rest of this paper is as follows. Section 2 provides a software context to the implementation of our approach and quantifies the volume of performance data generated. Section 3 describes the details of our approach to performance data reduction. Section 4 discusses our experimental methodology and studies results to investigate the degree of effectiveness our preliminary heuristics enjoy. We then discuss related work in scalable performance analysis in section 5 and finally draw conclusions on our contributions in this paper and discuss the rich area of future work that should be pursued in section 6.

2 Software Infrastructure

CHARM++ [10] is a portable C++ based parallel programming language based on the *migratable object programming model* and resultant *virtualization* of processors. In this approach [9], a programmer decomposes a problem into N *migratable objects* (MO) that will execute on P processors, where ideally $N >> P$. The application programmer's view of the program is of MOs and their interactions; the underlying runtime system keeps track of the mapping of MOs to processors and performs any remapping that might be necessary at run-time. In CHARM++, MOs are known as *chares*. Chares are C++ objects with special *entry methods* that are invoked asynchronously from other chares through messages. CHARM++ uses message-driven execution to determine which chare gets control of a processor. An advantage of this approach is that no chare can hold a processor idle while it is waiting for a message. Since $N >> P$, there may be other chares on the same processor that can overlap their computation with the communicating chare. CHARM++ is actively used in a number of major real-world scientific applications [18, 22, 7] that demand high scalability for some of the phenomenon the scientific community wishes to study.

The *Projections Analysis Framework* [11] consists of an instrumentation component and a visualization/analysis tool. It's features have been used extensively to tune many CHARM++ applications in order to enhance their scalability and performance, especially NAMD [19, 11, 15]. Projections instrumentation is fully automated by default since CHARM++ is a message driven system. Specifically, the runtime system (RTS) knows when it is about to schedule the execution of a particular method of a particular object (in response to a message being picked up from the scheduler's queue), and when an object sends a message to another object.

In most MPI-based tools when a processor waits at a receive call, the time spent is considered a part of the communication overhead associated with the actual call. However, this often includes idle time, which can be cleanly separated from communication overhead by the CHARM++ RTS. Similar useful information may be recorded by the proposed PERUSE [12] interface for MPI which provides access to performance data associated with the underlying MPI library implementation.

The log data (see section 2.1) is typically written out at the end of the run. If it exhausts all the memory space allocated to it, or if the user desires (e.g. at known global synchronization points), the data can be flushed to disk. However, in our experience, asynchronous flushing of log data causes such severe perturbation of application performance that the performance information after the first instance of such a flush becomes effectively useless.

2.1 Event Log Format and Data Volume

Our event logs are written in a text format with one event per line. Some events contain more details than others. For example, we record events like the start and end of each CHARM++ entry method, every time a message was sent and each time the runtime scheduler goes idle or returns from being idle. For each event, different types of attributes may be recorded. This includes the timestamp, size of a message, the CHARM++ object id, and any performance counter information (e.g. PAPI [1]) associated with the event. Recording performance counter information is optional and the default instrumentation policy has small overhead, involving a low-cost timestamp request and accessing the in-memory instrumentation log buffers.

It is clear that instrumenting an application for the entire run duration is unscalable and counter-productive. Projections has an interface for turning instrumentation on and off, allowing iterative applications like NAMD to instrument, for example, a sample of 200 simulation steps to capture performance data that represent a meaningful subset of the entire execution.

nCPUs	apoa1	f1atpase	stmv
512	827 MB	1,800 MB	2,800 MB
1024	938 MB	2,200 MB	3,900 MB
2048	1,200 MB	2,800 MB	4,800 MB
4096			5,700 MB

Table 1. Total volume of trace data summed across all files. apoa1 is a NAMD simulation with 92k atoms, f1atpase simulates 327k atoms while stmv simulates 1M atoms.

This mechanism for controlling the volume of performance data generally helps to contain the total data volume to the order of $O(E + P \times C)$ where E is the total number of computational events generated by an application given a fixed input, P is the number of processors on which the application is executed and C is the overhead due to additional communication events on each processor as a result of scaling. We quantify the volume of trace data generated for three different NAMD simulations in table 1. The table shows how data volume tends to grow for strong scaling as well as for weak scaling. It is worthwhile to note that the overhead factor C is dependent on E, making the problem of large trace data volume worse when we try to scale applications for larger problem sizes.

There are plans, in the coming years, to make simulations in the order of 100 million atoms on machines with at least a hundred thousand processors. These plans drive the urgency of research in scalable performance analysis approaches to deal with the expected increase in performance data volume.

3 Approach to Performance Data Reduction

Our approach to helping performance tools scale with the larger volume of generated event-based performance data is to simply trim the total volume of data fed post-mortem to these tools by only retaining performance data pertaining to a subset of processors on which the application is executed.

The adoption of this approach is based on two observations. The first is that in almost all implementations of event-based performance tracing, log buffers document the behavior and are stored in the memory of each processor. The second is that processors often exhibit performance behavior that is similar to a set of other processors, forming equivalence classes. These observations highlight the possibility that the online selection of performance data from an appropriate subset of outlier processors may be sufficient for the capture of details of a significant number of important bottlenecks and performance problems. Coupled with the selection of a subset of processors whose behavior are representitive of other processors in their equivalence class, this could then allow a reasonable reconstruction of the application's behavior for that run, but with significant reduction in data volume. This approach can take advantage of the parallel machine available to the application. It allows the analysis algorithms to be performed in parallel on the full traces held on each processor's process space within the original parallel application. We note, however, that this also requires the factoring of the additional analysis overhead into the time requested for the job.

The first step to this process is to discover the equivalence classes of processor behavior. Once a suitable partition of processor sets have been found, we employ heuristics to identify processors whose behavior are representative of its associated equivalence class, as well as a number of processors whose behavior exhibit extremal behavior with respect to its associated equivalence class. The following is a summary of the factors that affect the dataset. These are discussed in more detail in the subsequent subsections.

1. The performance attributes (e.g. execution time of CHARM++ entry methods) selected as the basis for equivalence class discovery.
2. The quality of equivalence classes formed.
3. The quality of the heuristic used to select representatives of an equivalent class.
4. The number of representatives selected from an equivalent class.
5. The quality of the heuristic used to select extremal processors from an equivalent class.
6. The number of extremal processors selected from an equivalent class.

3.1 k-Means Clustering for Equivalence Class Discovery

We make use of the k-Means clustering [8] algorithm to discover equivalence classes of processors and from these classes, select representative processors.

We choose to apply k-Means clustering over E dimensions, where E is the number of instrumented CHARM++ entry methods. Currently, we take the total execution time of an entry method as the primary attribute for the algorithm. We define a processor's sample-point to be a vector of E dimensions, where the coordinate along each dimension is given by the total execution time spent by each CHARM++ entry method on that processor. The distance metric between any two processors is then computed as an unweighted Euclidean distance given the two processors' sample-point vectors. The initial k cluster starting points for the algorithm are placed by uniformly spreading them across the E-dimensional bounding-box. The bounding-box is formed by the minimum and maximum values of each of the E entry method execution times over all P sample points where P is the number of processors. Figure 1 graphically shows an example of the final result from the hypothetical application of k-Means clustering with 3 clusters (equivalence classes), over 2 dimensions (labelled entry method X and entry method Y) with some arbitrary number of processor sample-points.

Several factors affect the accuracy and quality of k-Means clustering. One is the choice of k which tells the algorithm to locate k clusters in the sample-space, which

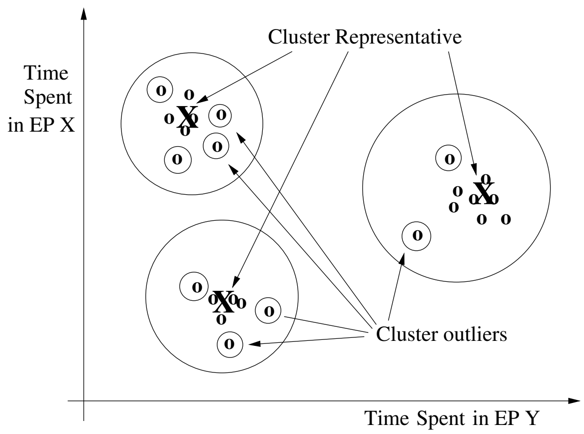

Figure 1. An example of clustering of processor sample points over 2 entry methods X and Y.

may not correspond to a more natural number of equivalence classes. Most tools making use of k-Means clustering either make use of domain knowledge or trial and error for the choice of k. Another factor is the initial seeding of the k starting points in the sample-space which affect which sample points eventually get placed in a cluster. Finally, the chosen set of performance attributes determines the usefulness of the partitioning.

The value of k is in fact the number of initial seeds used in the clustering algorithm. Depending on seeding policy, they may or may not ultimately represent non-empty clusters. We made some trial and error experiments with NAMD, varying the number of processors and k, summarized in table 2. The results for $k = 15$, we felt, appeared more or less consistent with the number of processor-classes we have observed in the past while studying the performance of NAMD. The observed classes include processor 0 which has to perform special tasks, as well as certain processors in NAMD which, when assigned certain work objects are not allowed to be assigned certain other types of objects. As a result, we chose to set k to 15 for the identification of clusters in our subsequent experiments. The initial seeding of the k starting points are, for now, uniformly distributed in the E dimensional space.

3.2 Choosing Representative Processors

Our selection of processor representatives given a set of equivalence classes is extremely simple. For each non-empty equivalence class C, we select exactly one representative R_C closest to the mean point according to the Euclidean distance measure described in section 3.1. Note that the mean point of a cluster need not necessarily represent a real sample point.

nCPUs	Number of non-empty clusters found with				
	5 seeds	10	15	20	25
512	1	4*	4*	6*	6*
1024	2	4	6*	6*	7*
2048	2	4*	5*	6*	7*
4096	3	6*	7*	9*	10*

Table 2. Number of non-empty clusters found by clustering algorithm by varying the number of initial seeds uniformly distributed in the sample space The * indicates that processor 0 was alone in its own cluster.

The quality of this selection scheme depends on the quality of the discovered clusters. For instance, in a simple dataset with 2 dimensions, this scheme functions poorly in the case of a cluster whose members' data points form part of a perfect circle around the cluster's mean point.

3.3 Choosing Extremal/Outlier Processors

The goal at this stage is to pick out processors whose behavior is most different from the norm within a given cluster. To select O outlier or extremal processors given the equivalence classes, where O is some number smaller than the number of processors P, we need to apply a selection heuristic as follows.

If cluster C has one or fewer sample points, then nothing needs to be done, as the representative selection scheme has already picked out that processor in section 3.2.

Every other clusters' member data point will have had their Euclidean distance from the cluster mean points computed as part of the clustering algorithm. We can now make use of these computed values. For each cluster C, we sort the member processors by distance computed. Let S_c be the number of processor sample-points in each cluster C. We now select approximately $\frac{O \times S_c}{P}$ processors from each cluster that are furthest from their cluster mean points.

Once again, this heuristic is dependent on the quality of the equivalence classes generated by the clustering algorithm in 3.1. More importantly though, the heuristic is very simple and currently does not take into account biases in cluster distributions. For instance, if one cluster is particularly tight relative to others, then perhaps the heuristic can be made to realize that this cluster is best captured by just the representatives selected in section 3.2.

4 Experimental Methodology

The goal of our experiments is to investigate the effectiveness of our processor selection heuristics. We determine effectiveness to comprise of two components. The first is a quantitative measure of how much data was reduced. The second is an evaluation of the quality of the reduced dataset with respect to performance problem discovery which, in our case, is performed using Projections.

Our experiments are based on the 1 million atom NAMD simulation of the complete satellite tobacco mosaic virus (**stmv** in table 1). We made the runs from 512 to 4096 processors on the Cray XT3 installed at Pittsburgh Supercomputing Center through Teragrid [3] resources.

To enable an assessment of our technique, we injected a known poor-grainsize performance problem as described in our case study paper [11] into the simulation. This would manifest itself as a bimodal "camel hump" in a histogram plot that would not show up otherwise when the same plot was made of performance data from a run without the problem injection.

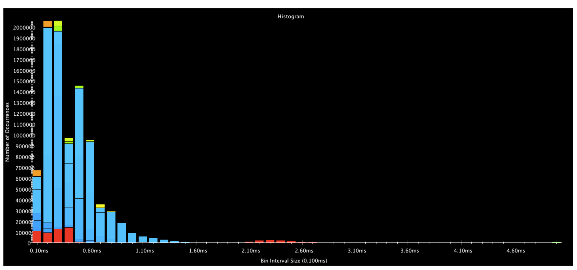

Figure 2. Histogram plot of stmv in Projections. The vertical bars show the number of occurrences of CHARM++ entry methods, distinguished by their colors, that took a certain amount of time to execute. The first bar shows the number of entry methods that executed for 0.1 ms to 0.2 ms, the second for 0.2 ms to 0.3 ms, etc

In our histogram plot, we display a stacked graph of occurrence counts of each instrumented CHARM++ entry method against the time it took. The histogram covers the occurrence of entry methods that range from 0.1 ms to 10.0 ms over 100 bins. The occurrence count data is summed across all processors over the 200 NAMD iterations. Figure 2 shows what the histogram looks like without the injected performance problem and the corresponding bimodal histogram in figure 3.

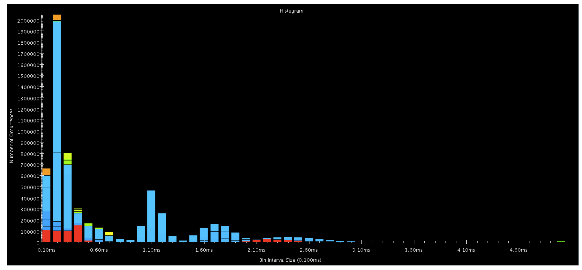

Figure 3. Histogram plot of stmv with poor-grainsize in Projections. Note the shift of the number of CHARM++ entry method occurrences rightward where the bins represent longer execution times.

Projections currently does not apply any proportional modifications to performance information of the processor representatives in order to extrapolate their contribution. As a result, we determine the quality of the reduced data set by two criteria. Let H_r^i be the total occurrence counts for the i-th histogram bar in the reduced data set. Let H_o^i be the total occurrence count for the i-th histogram bar in the original full data set. Let P_r be the number of processors in the reduced data set and let P_o be the number of processors in the original full data set. Our first criterion states that for each i, $\frac{H_r^i}{H_o^i}$ should be close to $\frac{P_r}{P_o}$ where $H_o^i \neq 0$. Our second criterion states that across all i where $H_o^i \neq 0$, $\frac{H_r^i}{H_o^i}$ should not vary by too much. We will refer to these two criteria as the *proportionality criteria*.

4.1 Results

We first show the reduction in data volume in table 3, through the selection of subsets of processors that number approximately 10% of the original dataset. As we can see, the reductions come as no surprise, although it is important to note that the number of traced events can vary significantly between processors. One cannot trivially expect to see a perfectly linear reduction in data volume.

For the quality measure, we applied the *proportionality criteria* to the reduced trace data generated from full datasets ranging from 512 to 4096 processors and with processor reduction ranging from approximately 5% to approximately 20% of the original number of processor logs. This is summarized in table 4. In this table, $\overline{H} = \frac{\sum_{i=0}^{n} \frac{H_r^i}{H_o^i}}{m}$ where $H_o^i \neq 0$. For our experiments, $n = 100$ and $m = n - k$ where k is the number of instances where

nCPUs	original size	reduced data
512	2,800 MB	275 MB
1024	3,900 MB	402 MB
2048	4,800 MB	551 MB
4096	5,700 MB	667 MB

Table 3. Reduction in total volume of trace data for stmv. The number of processors selected in the subsets are 51, 102, 204 and 409 for 512, 1024, 2048 and 4096 original processors respectively.

$H_o^i = 0$ (In other words, we ignore histogram bars where proportionality has no meaning). We use $\overline{H}$'s closeness to $\frac{P_r}{P_o}$ as the measure to satisfy the first criterion of the *proportionality criteria*. Likewise, the standard deviation σ over $\frac{H_r^i}{H_o^i}$ for all i where $H_o^i \neq 0$ is used as the measure to satisfy the second criterion for reduced data set quality. From the table, we see that with the exception of the data from 512 processors, the proportionality of the histogram bars generated from reduced data sets matches very well with what is expected of the data. In addition, the standard deviation values appear to be good. We believe the 512 processor data sets involve entry methods of larger grain size, making them more sensitive to variation across processors. In any event, the values are not terribly off-the-mark although improvements may be possible.

We have also visually confirmed the quality of the data using Projections on the partial processor logs generated by our approach.

P_o	$\frac{P_r}{P_o}$	$\overline{H}$	Standard deviation σ
512	0.0488	0.0641	0.00732
	0.0996	0.1180	0.00768
	0.1992	0.2237	0.00732
1024	0.0498	0.0511	0.00168
	0.0996	0.1008	0.00157
	0.1992	0.1921	0.00264
2048	0.0498	0.0487	0.00122
	0.0996	0.0977	0.00216
	0.1992	0.1883	0.00575
4096	0.0498	0.0501	0.00170
	0.0998	0.0981	0.00203
	0.1997	0.1975	0.00163

Table 4. Reduced dataset quality by proportionality based on total height of histogram bars.

5 Related Work

A large body of recent work tackles the scalability of performance analysis tools. Wolf et. al. surveyed [24] the issues, challenges and approaches to this problem in detail. The Scalasca project [6, 5] makes use of the parallel machine generating traces to also perform automated performance analysis of the trace data.

The use of clustering algorithms for performance bottleneck detection, visualized using scatterplots, have been explored in TAU [21]. They do not make use of clustering for data reduction but have highlighted the importance of dimensionality reduction and the removal of correlated performance metrics. Pablo [17, 20] demonstrated an old prototype similar to our approach for spatial data reduction. It retains trace data segments for processor representatives of clusters it discovers, reclustering as needed as performance data changes over time. Our approach currently ignores changes over time while focusing on processors' performance behavior across the entire duration of the traces recorded. At the same time, we are uncertain about the overheads that are involved in the constant testing of cluster quality and potential need to recompute clusters in their scheme. The main difference between our approaches is that we seek out outlier/extrema processors from clusters as our focus while still preserving the general performance profile with representatives. This difference is due to our belief that in performance analysis, the unusual should be sought out in addition to what is usual.

Performance data may also be reduced in the temporal dimension. Chung et. al. [4] sought repeated communication patterns in MPI codes as a source for compression, augmenting the visualization tool to re-generate the full details when needed. Knupfer and Nagel [14] shows great potential for temporal data reduction through the construction and subsequent compression of Complete Call Graphs. They further introduce a distributed architecture for performance analysis [13] that further enhances scalability by allowing parallel analysis. Knupfer's approach offers the potential to co-exist well with our approach, reducing data both spatially and in time while at the same time allowing for the possibility of parallel analysis of trace data on the same machine the application was executed. Casas et. al. [2] meanwhile applies signal processing techniques like non-linear filtering and spectral analysis directly on event traces in order to identify similar regions along the time dimension and achieve data reduction by removing multiple instances. Vetter and Reed [23] studies the reduction of performance data by removing uninteresting performance metrics through the technique of dynamic statistical projection pursuit.

6 Conclusion and Future Work

We have presented a way for reducing the volume of event trace data by retaining a subset of processor logs through the identification of representatives and extrema members after partitioning through k-means clustering. We have demonstrated the potential of this approach by quantifying the reduction in data volume as well as the quality of the retained data, for the specific performance problem of poor application grainsize, for a 1 million atom NAMD simulation from 512 to 4096 processors. We showed that the use of processor equivalence classes is important for the retention appropriate processors so that the grainsize profile of the performance data is not badly affected.

Unfortunately, we have not measured the time taken to apply the algorithms used in our approach. In the interest of time, most of our results were generated using an equivalent sequential code post-mortem rather than attempting to submit jobs for many experimental parameters that requires the entire machine at PSC. The k-means algorithm has been observed to converge quickly and the sequential code took less than 30 seconds to compute the results for 4096 processor logs on a workstation. We do not foresee the parallel version taking any longer.

Future work will focus on studying the quality of the approach with respect to other performance problems, other applications, other programming paradigms like MPI and even higher levels of processor scaling, up to tens of thousands of processors. This includes studying the use of other performance metrics like communication characteristics to be used as attributes for our approach. It is also unclear if this approach can be applied ubiquitously to all classes of performance problems and bottlenecks. If not, a technique for reasonbly combining multiple heuristics that each choose a different subset of processor trace data suitable to different classes of problems is needed. In particular, we hope to more clearly demonstrate the utility of selecting outlier/extremal processors from clusters. We see them as being capable of locating "rogue" processors, for example processors that experience very long computational stretches in some entry methods. These processors would still be members of the same clusters but we expect the computational spikes to cause them to veer off the cluster centroid more than other typical processor members. On the other hand, we also want to be able to identify possible classes of performance problems that cannot be effectively retained through data reduction by this approach. Problems that involve communicating critical paths come into mind as they would require the heuristic to have to pick the exact processors involved in the critical path. Finally, this approach is not intended to be standalone, we intend to study the quality of this approach when used in conjunction with temporal data reduction techniques possibly developed by others.

7 Acknowledgments

This research was supported in part by the National Science Foundation through TeraGrid resources (Grant Number: MCA93S028) provided by Pittsburgh Supercomputing Center and by the National Institute of Health (NIH PHS 5 P41 RR05969-04). TeraGrid systems are hosted by Indiana University, LONI, NCAR, NCSA, NICS, ORNL, PSC, Purdue University, SDSC, TACC and UC/ANL.

References

[1] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A Portable Programming Interface for Performance Evaluation on Modern Processors. *Int. J. High Perform. Comput. Appl.*, 14(3):189–204, 2000.

[2] M. Casas, R. M. Badia, and J. Labarta. Automatic Structure Extraction from MPI Application Tracefiles. *Lecture Notes in Computer Science*, 4641:3–12, August 2007.

[3] C. Catlett and et. al. TeraGrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications. In L. Grandinetti, editor, *HPC and Grids in Action*, Amsterdam, 2007. IOS Press.

[4] I.-H. Chung, R. E. Walkup, H.-F. Wen, and H. Yu. MPI tools and performance studies—MPI performance analysis tools on Blue Gene/L. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 123, New York, NY, USA, 2006. ACM Press.

[5] M. Geimer, F. Wolf, A. Knupfer, B. Mohr, and B. J. N. Wylie. A Parallel Trace-Data Interface for Scalable Performance Analysis. In *Proceedings of the Workshop on State-of-the-art in Scientific and Parallel Computing (PARA), Minisymposium Tools for Parallel Performance Analysis*, Umea, Sweden, June 2006.

[6] M. Geimer, F. Wolf, B. J. N. Wylie, and B. Mohr. Scalable Parallel Trace-Based Performance Analysis. In *Proceedings of the 13th European Parallel Virtual Machine and Message Passing Interface Conference*, Bonn, Germany, September 2006. Springer LNCS.

[7] F. Gioachin, A. Sharma, S. Chakravorty, C. Mendes, L. V. Kale, and T. R. Quinn. Scalable cosmology simulations on parallel machines. In *VECPAR 2006, LNCS 4395, pp. 476-489*, 2007.

[8] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[9] L. V. Kalé. Performance and productivity in parallel programming via processor virtualization. In *Proc. of the First Intl. Workshop on Productivity and Performance in High-End Computing (at HPCA 10)*, Madrid, Spain, February 2004.

[10] L. V. Kale and S. Krishnan. Charm++: Parallel Programming with Message-Driven Objects. In G. V. Wilson and P. Lu, editors, *Parallel Programming using C++*, pages 175–213. MIT Press, 1996.

[11] L. V. Kale, G. Zheng, C. W. Lee, and S. Kumar. Scaling applications to massively parallel machines using projections performance analysis tool. In *Future Generation Computer Systems Special Issue on: Large-Scale System Performance Modeling and Analysis*, volume 22, pages 347–358, February 2006.

[12] R. Keller, G. Bosilca, G. Fagg, M. Resch, and J. J. Dongarra. Implementation and Usage of the PERUSE-Interface in Open MPI. *Lecture Notes in Computer Science*, 4192:347–355, September 2006.

[13] A. Knupfer, H. Brunst, and W. E. Nagel. High performance event trace visualization. *Parallel, Distributed and Network-Based Processing, 2005. PDP 2005. 13th Euromicro Conference on*, pages 258–263, 2005.

[14] A. Knupfer and W. E. Nagel. Construction and compression of complete call graphs for post-mortem program trace analysis. *icpp*, 00:165–172, 2005.

[15] S. Kumar, C. Huang, G. Almasi, and L. V. Kalé. Achieving strong scaling with NAMD on Blue Gene/L. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2006*, April 2006.

[16] S. Moore, F. Wolf, J. Dongarra, S. Shende, A. Malony, and B. Mohr. A Scalable Approach to MPI Application Performance Analysis. *LNCS*, 3666:309–316, 2005.

[17] O. Y. Nickolayev, P. C. Roth, and D. A. Reed. Real-time statistical clustering for event trace reduction. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):144–159, Summer 1997.

[18] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005.

[19] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kalé. NAMD: Biomolecular simulation on thousands of processors. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–18, Baltimore, MD, September 2002.

[20] D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. W. Schwartz, and L. F. Tavera. Scalable Performance Analysis: The Pablo Performance Analysis Environment. In *Proc. Scalable Parallel Libraries Conf.*, pages 104–113. IEEE Computer Society, 1993.

[21] S. Shende and A. D. Malony. The TAU Parallel Performance System. *International Journal of High Performance Computing Applications*, 20(2):287–331, Summer 2006.

[22] R. V. Vadali, Y. Shi, S. Kumar, L. V. Kale, M. E. Tuckerman, and G. J. Martyna. Scalable fine-grained parallelization of plane-wave-based ab initio molecular dynamics for large supercomputers. *Journal of Comptational Chemistry*, 25(16):2006–2022, Oct. 2004.

[23] J. Vetter and D. Reed. Managing performance analysis with dynamic statistical projection pursuit. *sc*, 00:44, 1999.

[24] F. Wolf, F. Freitag, B. Mohr, S. Moore, and B. Wylie. Large Event Traces in Parallel Performance Analysis. In *Proceedings of the 8th Workshop Ṕarallel Systems and Algorithms(́PASA)*, Lecture Notes in Informatics, Gesellschaft fr Informatik, Frankfurt/Main, Germany, March 2006.

[25] F. Wolf and B. Mohr. EARL - A Programmable and Extensible Toolkit for Analyzing Event Traces of Message Passing Programs. In *HPCN Europe '99: Proceedings of the 7th International Conference on High-Performance Computing and Networking*, pages 503–512, London, UK, 1999. Springer-Verlag.