Scalable Techniques for Performance Analysis

Chee Wai Lee and Laxmikant V. Kalé

Department of Computer Science

University of Illinois at Urbana-Champaign

cheelee@uiuc.edu, kale@cs.uiuc.edu

Abstract

The scalability of performance tools in high performance computing has been lagging behind the growth in the sizes of supercomputers and the applications that run on them. It is recognized that performance event traces still provide the best and most useful source of information for analysis. However, the volume of performance trace data generated easily becomes unmanagable without appropriate controls as we scale upwards. At the same time, the amount of information that has to be presented to a human analyst can also become overwhelming.

We present techniques used to address the above problems and enhance the scalability of Projections, a performance instrumentation and visualization framework for the migratable object programming model CHARM++. *Projections provides multiple resolutions of performance data. We couple this feature with the use of heuristics and clustering algorithms at application runtime to provide powerful mechanisms to reduce performance data volume and analysis time while preserving data relevance. We employ similar heuristics and algorithms for enhanced interactive post-mortem visualization and analysis assistance to a human expert.*

To demonstrate the continued effectiveness of the analysis process using the above techniques while showing that the data volume is significantly reduced, we present experimental results on simulation benchmarks based on NAMD*, a popular molecular dynamics application.*

1 Introduction

A large amount of trace data may be generated each time an instrumented application is run for the purpose of studying its parallel performance. As an application scales in size (e.g. running larger scientific simulations) and runs on more processors, performance tools must scale accordingly to handle increases in the amount of performance data that can be generated.

Although aggregated or sampled profile data can be useful, the use of detailed event traces from instrumented parallel applications is considered the best and most useful source of information for studying their performance [25, 15]. At the same time, a performance tool must also be scalable as well as effective in presenting information to a human analyst. In today's scientific codes running on existing high performance systems, event trace logs will typically run into the gigabyte range. Dealing with data from large simulations running on thousands of processors has already become a reality in our collaborations and research and has strained the abilities of our performance analysis tools. We believe that for performance studies to be scalable and productive, the volume of data generated as well as the information presented to a human analyst must be kept low, relevant and concise.

Our approach is centered on the use of heuristics for the identification of extrema data and the identification of the representative and outliers of equivalence classes of processors through clustering algorithms. We will show in this paper that an *online* application of the approach during an application's execution, coupled with different performance data resolutions, can allow us to generate less data and yet capture its core performance characteristics in high detail. At the same time, similar *offline* approaches can be interactively used by an analyst to visually identify bad behavior over more specific performance space domains and quickly switch to more detailed representations of the performance domain.

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 our ADAPTIVE MPI applications, most notably the popular molecular dynamics application NAMD. 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.

Our experiments are conducted on the Cray XT3 supercomputer installed at Pittsburgh Supercomputing Center with NAMD simulation benchmarks instrumented for over 200 simulation timesteps.

2 CHARM++ and its Applications

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 [11], 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. NAMD [17] is a well-known and widely used parallel classical molecular dynamics application for biomolecular systems which has won a Gordon Bell Award [18], and scaled to at least $16,000$ processors. LEANCP [22] is a collaborative effort using CHARM++ to gain unprecedented scalability for the Car-Parrinello ab initio molecular dynamics method. CHANGA (CHARM N-BODY GRAVITY SOLVER) [7] is a scalable code performing collisionless N-body simulations. It can perform cosmological simulations with periodic boundary conditions in comoving coordinates or simulations of isolated stellar systems.

3 Projections Performance Analysis Framework

The *Projections Analysis Framework* [13] consists of an instrumentation component and a visualization/analysis tool. 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).

Unlike MPI, in CHARM++ we can retrieve the idle time from the RTS. In MPI when a processor waits at a barrier or a receive call, the time spent is considered a part of the communication overhead. However, this often includes idle time, because another processor has not arrived at the barrier (or has not sent the message). CHARM++ RTS can cleanly separate communication overhead from such idle time. This prevents users from making erroneous conclusions that the performance is poor due to "the slow barrier operations", when it may be due to load imbalances.

The log data (see section 3.1) is typically written out at the end of the run. However, if it uses 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.

The *Projections visualization component* supports multiple views:

1. a **Summary Graph** view is quickly loaded at Projections startup which shows a sketch of average processor utilization over the entire run. Time interval granularity is limited to whatever is provided by the summary module. This fast automatic loading is performed if the user provides the location of the log files to Projections at startup;

2. an **Overview** graph shows slightly more detail, displaying utilization as a color-intensity value in a plot of processors against time intervals;

3. the **Profile** view shows a stacked column bar for each selected processor, for a selected time interval. The time spent by each processor in various activities is shown within each bar. This view clearly separates idle time and communication overhead;

4. our **Histogram** views are very useful for revealing grainsize issues for computation units as well as communication.

One such view shows the frequency of entry method calls binned according to that particular call's execution time. It also shows the frequency of messages sent binned according to the size of the message;

5. the **Time Profile** view allows a user to divide an execution time range into a number of time-interval bins. It shows the time spent by each CHARM++ entry method within each time-interval bin aggregated across all processors of interest. This information is displayed against time on the x-axis. Figure 2(b) shows how this view used to visualize the overlapping activities in a NAMD execution;

6. the **Timeline** view displays a chronological sequence of entry methods for each processor of interest and is similar to other timeline tools such as Jumpshot [27], and Paragraph [8]. However, it is a highly sophisticated view which presents additional detailed information about events not normally available in other MPI-based tools via simple mouse clicks. This is shown in Figure 1 where each row corresponds to a specific processor with colored bars representing the entry method that was executed;

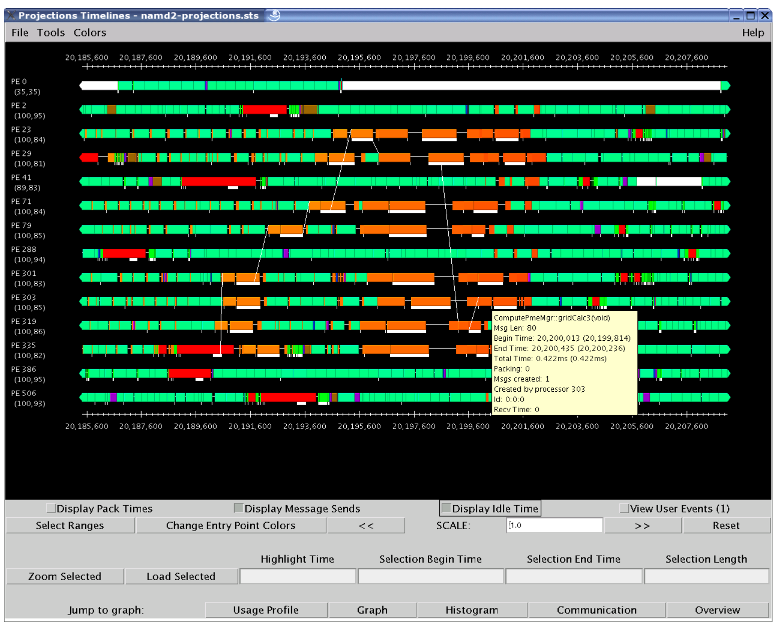
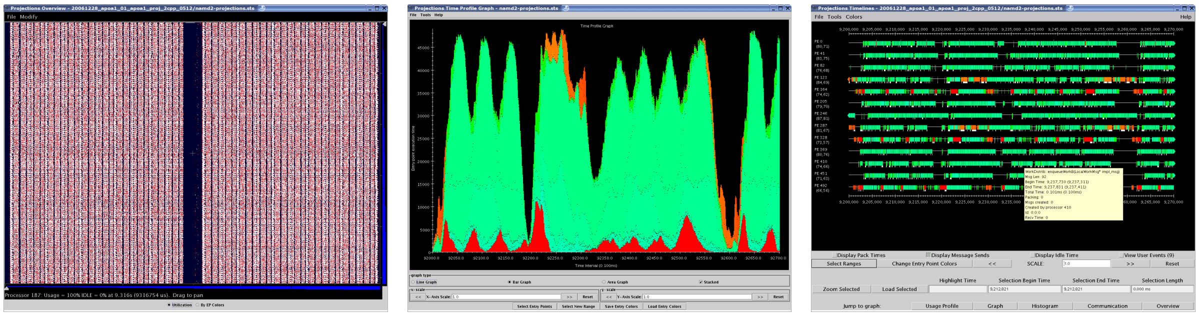


Figure 1. Timeline events showing entry method activities in NAMD. Mouse-overs give detailed information about events and users may right-click on events to show white communication lines that may reveal critical path issues.

Performance analysis is human-centric. As illustrated in Figure 2, starting from visual distillations of overall application performance characteristics, the analyst employs a mixture of application domain knowledge and experience with visual cues expressed through Projections in order to identify general areas (e.g. over a set of processors and time intervals) of potential performance problems. The analyst then zooms in for more detail and/or seeks additional perspectives through the aggregation of information across data dimensions (e.g. processors). The same process is repeated, usually with higher levels of detail, as the analyst homes in on a problem or zooms into another area to correlate problems. The richness of information coupled with the tool's ability to provide relevant visual cues contribute greatly to the effectiveness of this analysis process.



(a) Overview of 512-processor run of NAMD over several seconds.

(b) A time profile of 512 processors over a 70ms range of interest.

(c) Detailed timeline of events on sample processors of the same 70ms range.

Figure 2. Projections performance analysis, zooming to different levels of detail and viewing different data perspectives.

A use-case from our tuning of NAMD on ASCI RED [12] illustrates the above discussion. The benchmark took 57 seconds on a sequential processor, but performance was unexpectedly poor on 2000 processors. A *time profile* view (like that of Figure 2(b)) showed a characteristic load imbalance pattern. This was verified through a *profile* view from which heavily loaded processors were manually identified. The type of objects revealed to have dominant activities on those heavily loaded processors were analyzed via a grainsize histogram. This showed that the largest object of this type had activities with a duration of 40 ms! That clearly precluded speedups near 2000, even under ideal conditions. The same histogram view illustrated a bi-modal distribution of activity duration for these objects. This helped NAMD developers identify which objects to decompose further, leading to a better grainsize distribution. This was a significant step in the scaling of NAMD to 8192 processors on BlueGene/L [14].

3.1 Multiple Levels of Log Resolution

Projections supports three different levels of resolution for performance data. All three are ASCII formats and store each processor's generated data into a different file.

Our *summary* format is the most compact data format. It records processor utilization for each processor over k time intervals where k is fixed. The duration, d, of these time intervals is adjusted dynamically, doubling each time the application's execution goes past $k \times d$. The size of each file is dependent only on the desired number of time-intervals k. Therefore, the total size summed across all files grows in $O(P)$ where P is the number of processors. We use runlength encoding as an additional measure to reduce the size of each file. We observe typical file sizes to be about 3 kB given default settings for the number of intervals.

Projections also supports an intermediate *summary detail* format. It employs the same dynamically adjusted time interval scheme for recording data as described above for the *summary* format. However, we record attribute information for each time interval such as the time spent or number of messages received separatedly for each CHARM++ entry method. Such a format is reasonably rich and useful for analysis, allowing a quick way to zoom in on a potential problem area. The size of each file is of the order of $k \times E$, where E is the number of CHARM++ entry methods in the application. However, E is relatively small, around 220 for NAMD, and depends on the static design of the application. As a result, E can be considered in practice to be a constant factor for all applications. So, the total data size grows in $O(P)$ similar to *summary* files. Like the *summary* format, runlength encoding is used to reduce the size of the file. The size of each file for NAMD is typically between 30–50 kB.

For our event log format, 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 access to the instrumentation log buffers. Any perturbation effects depend on the granularity of CHARM++ entry method work. Table 1 shows the effects of Projections instrumentation on the simulation time-step time of two NAMD benchmarks.

nCPUs	**apoa1 projections s/timestep**	**apoa1 no projections s/timestep**	**% diff**
512	0.00655327	0.00677664	+3.29
1024	0.00447543	0.00430876	-3.86
2048	0.00375795	0.00358866	-4.71
nCPUs	**f1atpase projections s/timestep**	**f1atpase no projections s/timestep**	**% diff**
512	0.01807300	0.01791210	-0.89
1024	0.00972048	0.00980697	+0.88
2048	0.00651015	0.00634311	-2.60

Table 1. Overhead of Projections instrumentation on NAMD execution time per simulation timestep. apoa1 simulates a system with 92k atoms while f1atpase simulates a system with 327k atoms.

The numbers show in Table 1 are based on results from a single data point because of time constraints. We believe the data instances where instrumented code runs faster than uninstrumented code are the result of noise. The important thing to note is that the differences in time are within 5% of each other despite the fact that in the worst scenario shown in the table, all of NAMD work in a single simulation timestep have to be instrumented within about 3.5 ms.

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.

This mechanism for controlling the volume of performance data generally helps to contain the total data volume to the order of $O(P \times C)$ where C is the overhead due to additional communication events. However, this will not address the scalability of data generation due to the application. For instance, simulating a system of 327,000 atoms generates a larger total number of events over 200 simulation timesteps than for a system of 92,000 atoms. Table 2 shows how the volume of event data grows as the number of processor increases for two different NAMD simulations.

nCPUs	**apoa1 max filesize kilobytes**	**apoa1 total data size megabytes**
512	3,500	827
1024	3,400	938
2048	3,360	1,200
nCPUs	**f1atpase max filesize kilobytes**	**f1atpase total size megabytes**
512	6,910	1,800
1024	5,580	2,200
2048	5,440	2,800

Table 2. The maximum sizes of each generated event log files as well as the total volume of data summed across all files are tabulated here. apoa1 simulates a system with 92k atoms while f1atpase simulates a system with 327k atoms.

4 Scalability Enhancements to Projections

Our approach enhances the scalability of Projections in two ways, data reduction; and visualization techniques that do not require a human analyst to have to unnecessarily browse through large portions of performance space in detail. The approach employs two key techniques, clustering algorithms to identify representatives; and heuristics to identify outliers and extrema either globally or locally. We will describe how we can achieve good data reduction through the use of these techniques at the end of a CHARM++ application execution and also how we have enhanced a visualization tool that employs similar techniques at analysis time to achieve the other scalability goal.

4.1 Locating Extrema/Outlier Processors

Extrema processors can be identified by methods as simple as sorting. For example, in CHARM++, processors with the least idle time are candidates for being overloaded and hence very helpful for identifying possible load imbalance.

More sophisticated heuristics can be used. For example, we can acquire $E \times P$ data points for E CHARM++ entry methods and P processors where T_ep is the execution time spent by the entry method e on a processor p. A global operation computes the average execution time and standard deviation of each entry method across all P processors. Each processor can now take these standard deviations summed across all E entry methods as a reasonable heuristic measure to rank the processor, and hence determine if a processor is an extrema. As an additional measure to avoid bias, we have implemented a version of this heuristic which takes the sum of standard deviations weighted by the average significance of each entry method (e.g. an entry method that, on average, takes up 50% of all computation has its standard deviation measure multiplied by a factor of 0.5).

These heuristics can be applied globally to all processors, or to individual equivalence classes identified through the clustering algorithms described in Section 4.2. In other words, P can be the total number of processors in that execution instance, or P can be the number of sample-points associated with one of the clusters. In the case when clustering is employed, the distance measure used to determine clusters may also be used as a heuristic for locating extrema values.

4.2 Identifying Representative Processors

A natural division of performance space data is that of processr association. With the exception of cross-processor communication, it is reasonably self-contained. As a result, our approach makes use of the k-Means clustering [9] algorithm to generate equivalence classes of processors and from these classes, select representative processors.

The clustering is performed over E dimensions, where E is the number of CHARM++ entry methods. We define a processors sample-point to be a vector of E dimensions, where the coordinate along each dimension is given by the 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 3 shows an example of the final result from using k-Means clustering with 3 clusters (equivalence classes), over 2 dimensions (labelled entry method X and entry method Y) with an arbitrary number of processor sample-points.

4.3 Data Reduction

k-means clustering and our heuristics are applied at the end of a CHARM++ application's execution to identify a set of representative and extrema processors. We have implemented these to take advantage of the parallel machine on which the application is run, using the same process space in order to perform this task, ensuring its own scalability. Unlike prior work [19], the instrumentation of the application is not dynamically affected by the application of these techniques. Since we

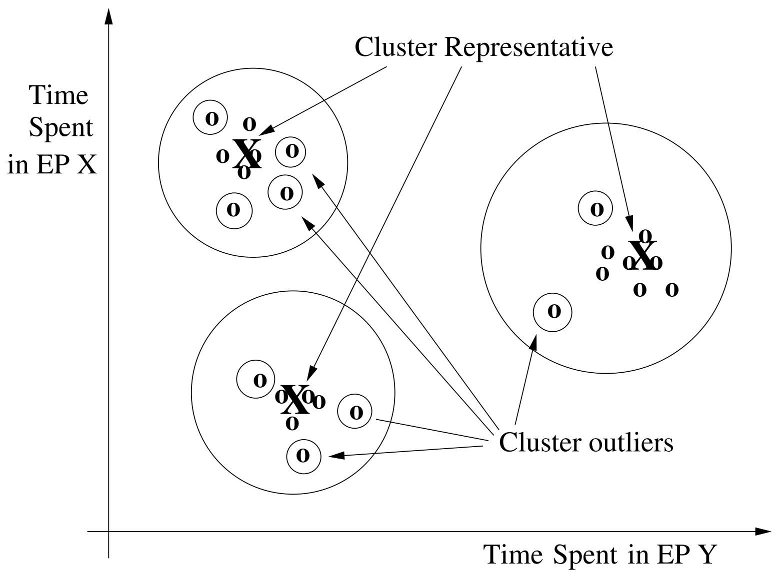


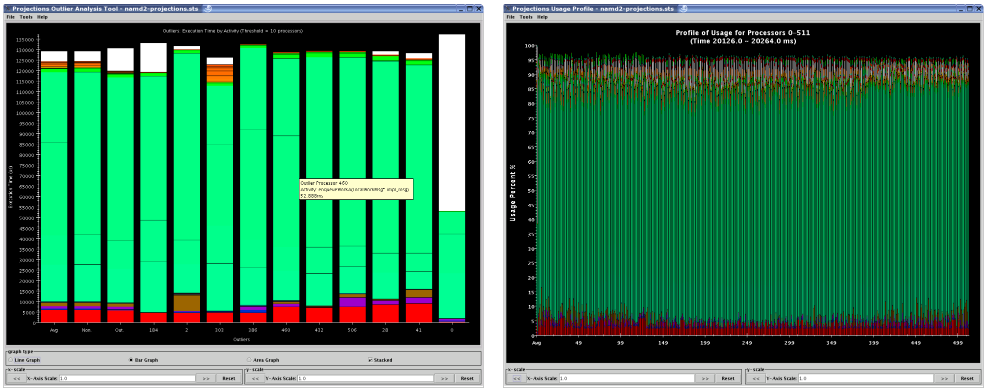
Figure 3. An example of clustering of processor sample points over 2 entry methods X and Y.

are capturing the full trace logs, we feel that a human analyst can better apply them offline to arbitrary phases of the execution as described in section 4.4.

After identifying the representatives and extrema, each processor is then instructed to either write out its full event trace log, or performance data at lower resolution or no data at all. Since the number of processor equivalence classes is relatively constant across runs, and given an upper bound on the number of extrema processors, the total amount of data due to full event trace logs generated can be significantly reduced while ensuring the data is relevant for the subsequent study of performance problems.

4.4 Effective Analysis and Visualization

The techniques of clustering and extrema discovery heuristics are extremely useful when applied offline at analysis time. We employ a simple modification of the *Profile* view described in section 3. Figure 4(a) shows the x-axis representing processors, each with processor-bars that displays the total time spent by entry methods. Different entry methods are marked with a different color. Starting from the left side of the screen, the processor-bar representing the global average; the average over all non-extrema processors; the average over all extrema processors; and the individual extrema processors sorted by significance. Each extrema processor-bar can be selected by the user to load a detailed *Timeline* view. This is a massive improvement over a manual hunt for mis-behaving processors and allows analysts to quickly load detailed performance information, over the appropriate time-range, to seek out sources of performance bottlenecks. Figure 4(b) shows a *Profile* view of the equivalent time range over all 512 processors. Analysts would have had to manually eyeball through the information, seeking processor bars that stood out as “unusual”. This process, of course, becomes impossible for runs involving thousands of processors.



(a) Modified *Profile* view presenting extrema performance data of the 10 most significant outliers out of 512 processors.

(b) *Profile* view showing all 512 processors. Note that the clarity of the individual processor-bars may be degraded in print due to aliasing.

Figure 4. Visualization enhancements contrasted against using the older *profile* view

It is important to note that this view can be employed interactively by the analyst. The information is generated on-the-fly by Projections using available data, given the analyst's preferences of execution time-range, partial processor selection and the number of extrema processors to display. This is done independently of any data reduction techniques described in section 4.3.

5 Validation

We have performed a preliminary test of our approach on a 256-processor execution of the NAMD apoa1 benchmark, instructing the system to restrict full event log generation to just the top 10% of processors identified as outliers. As expected, the total volume of data generated was around 83 MB down from 740 MB which is typical of instrumented logs for that execution configuration. The logs were fed into the visualization component of Projections and analyzed as usual with the only difference in experience being the inability of the *timeline* view to load any processor's data that were not full event traces. From our experience, there was no need to since those logs were categorized as non-outliers anyways.

6 Related Work

The Projections performance analysis framework is unique in that it is aimed at studying applications based on the migratable object model where CHARM++ objects (and ADAPTIVE MPI threads) are first class entities. Other tools and frameworks like Vampir [16] and Vampir NG [2], Cray Apprentice[2] [4], TAU [21], svPablo [5], jumpshot [27, 26] and Kojak [24] are replete with MPI-specific or architecture-neutral features but lack support for the performance semantics suitable for migratable objects.

An increasing body of work [3, 6, 23, 20, 15, 19] has recognized the scalability of performance analysis tools as a serious problem, especially recently. Many take the approach of automating the analysis process, either for dynamically adjusting the rate of instrumentation in some scalable fashion; or making the automatic tools parallel in order to scalably handle the increased volume of performance information. The use of clustering algorithms for performance bottleneck detection, visualized using scatterplots, have been explored in TAU [21] but not for data reduction. Pablo [19] demonstrated an old prototype very similar to our approach for data reduction and goals. However, it attempts to apply clustering dynamically over multiple phases of an application's execution, adding additional dimensions to the difficulties of using clustering such as clustering frequency and the computation costs of clustering. Our approach avoids those problems and added techniques for outlier/extrema discovery on top of clustering.

7 Conclusion and Future Work

To conclude, this paper's contribution to the body of research on scalability centered on the adaptation of clustering algorithms and the use of heuristics for extrema identification for data reduction and to provide quick assistance for bottleneck discovery during analysis. We have studied the nature of event trace data growth in NAMD and identified the processor-data association as a natural division for data reduction purposes. We showed a way of visualizing information gained through these techniques that allowed a human-analyst to quickly focus attention on details with potential performance bottlenecks. Finally, we have performed a simple validation through our implementation of the techniques in Projections.

Future work will focus on better use of the k-Means clustering algorithm the behavior of which is dependent on the value of k as well as the initial seeding of the k clusters. A natural option is to try several values of k and using the result that shows the best fit. Similarly, using several random starting points and choosing the best fit may prove a better option than uniformly distributed starting points. We also intend to show tangible improvements to the performance analysis process through these techniques. This can be measured in the form of time-to-bottleneck discovery by the users of Projections. Another interesting measure would be the number of times a user needs to use *timeline* to locate a bottleneck on a processor which did not have its detailed event log generated because it was considered "uninteresting".

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] H. Brunst and B. Mohr. Performance Analysis of Large-scale OpenMP and Hybrid mpi/openmp Applications with VampirNG. In *Proceedings of the International Workshop on OpenMP (IWOMP)*, Eugene, OR, June 2005.

[3] I-Hsin Chung, Robert E. Walkup, Hui-Fang Wen, and Hao 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.

[4] Luiz DeRose, Bill Homer, Dean Johnson, and Steve Kaufmann. Performance Tuning and Optimization with CrayPat and Cray Apprentice2. In *Cray User Group Meeting 2006*, Lugano, Switzerland, 2006.

[5] Luiz DeRose and Daniel A. Reed. Svpablo: A multi-language architecture-independent performance analysis system. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, 1999.

[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] Filippo Gioachin, Amit Sharma, Sayantan Chakravorty, Celso Mendes, Laxmikant V. Kale, and Thomas R. Quinn. Scalable cosmology simulations on parallel machines. In *VECPAR 2006, LNCS 4395, pp. 476-489*, 2007.

[8] M.T. Heath and J.A. Etheridge. Visualizing the Performance of Parallel Programs. *IEEE Software*, September 1991.

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

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

[11] Laxmikant 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.

[12] Laxmikant V. Kalé, Sameer Kumar, Gengbin Zheng, and Chee Wai Lee. Scaling molecular dynamics to 3000 processors with projections: A performance analysis case study. In *Terascale Performance Analysis Workshop, International Conference on Computational Science(ICCS)*, Melbourne, Australia, June 2003.

[13] Laxmikant V. Kale, Gengbin Zheng, Chee Wai Lee, and Sameer 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.

[14] Sameer Kumar, Chao Huang, Gheorghe Almasi, and Laxmikant V. Kalé. Achieving strong scaling with NAMD on Blue Gene/L. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2006*, April 2006.

[15] 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.

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

[17] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kalé, and Klaus Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005.

[18] James C. Phillips, Gengbin Zheng, Sameer Kumar, and Laxmikant 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.

[19] 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.

[20] Philip C. Roth and Barton P. Miller. On-line automated performance diagnosis on thousands of processes. In *PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 69–80, New York, NY, USA, 2006. ACM Press.

[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] Ramkumar V. Vadali, Yan Shi, Sameer Kumar, L. V. Kale, Mark E. Tuckerman, and Glenn 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] 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.

[24] F. Wolf and B. Mohr. KOJAK - A Tool Set for Automatic Performance Analysis of Parallel Applications. In *Proc. of the European Conference on Parallel Computing (Euro-Par)*, volume 2790 of *Lecture Notes in Computer Science*, pages 1301–1304, Klagenfurt, Austria, August 2003. Springer. Demonstrations of Parallel and Distributed Computing.

[25] Felix Wolf and Bernd 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.

[26] C. Eric Wu, Anthony Bolmarcich, Marc Snir, David Wootton, Farid Parpia, Anthony Chan, Ewing Lusk, and William Gropp. From Trace Generation to Visualization: A Performance Framework for Distributed Parallel Systems. *sc*, 00:50, 2000.

[27] Omer Zaki, Ewing Lusk, William Gropp, and Deborah Swider. Toward scalable performance visualization with Jumpshot. *The International Journal of High Performance Computing Applications*, 13(3):277–288, Fall 1999.