ParaProf: A Portable, Extensible, and Scalable Tool for Parallel Performance Profile Analysis

Robert Bell, Allen D. Malony, and Sameer Shende

University of Oregon, Eugene, OR 97403 USA
{bertie,malony,sameer}@cs.uoregon.edu

Abstract. This paper presents the design, implementation, and application of *ParaProf*, a portable, extensible, and scalable tool for parallel performance profile analysis. ParaProf attempts to offer "best of breed" capabilities to performance analysts – those inherited from a rich history of single processor profilers and those being pioneered in parallel tools research. We present ParaProf as a parallel profile analysis framework that can be retargeted and extended as required. ParaProf's design and operation is discussed, and its novel support for large-scale parallel analysis demonstrated with a 512-processor application profile generated using the TAU performance system.

1 Introduction

Perhaps the best known method for observing the performance of software and systems is profiling. Profiling techniques designed over thirty years ago [11], such as prof [17] and gprof [5] for Unix, are still apparent in the profiling approaches for modern computer platforms (e.g., vprof [9] and cvperf [20]). While ideological differences exist in profiling instrumentation and data collection, most notably between sample-based versus measurement-based approaches, profiling is the most commonly used tool in the performance analyst's repertoire. Unfortunately, despite the ubiquitous nature of profiling, profile analysis tools have tended to be system specific, proprietary, and incompatible. Not only does this pose difficulties for cross-platform performance studies, but the lack of reusable profile analysis technology has slowed the development of next-generation tools. With the general availability of hardware counters in modern microprocessors, the complexity of performance profile data will increase, further exacerbating the need for more robust profile analysis tool support.

Parallel software and systems introduce more difficult challenges to profile analysis tools. All the concerns for single processor performance analysis are present in parallel profiling, except now the profile data size is amplified by the number of processors involved. Profile analysis scalability is important for large-scale parallel systems where each thread of execution may potentially generate its own profile data set. Parallel execution also introduces new performance properties [1] and problems that require more sophisticated analysis and interpretation, both within a single profiling experiment and across experiments. The

identification of parallel inefficiencies and load imbalances requires analysis of all execution threads in a profile, whereas pinpointing reasons for poor scalability, for example, must combine detailed analysis across profiles generated with different numbers of processors. Lastly, the performance problem solving provided by the parallel profile analysis tool should reflect the system architecture as well as the model of parallelism used in the application.

This paper presents the design, implementation, and application of *ParaProf*, a portable, extensible, and scalable tool for parallel performance profile analysis. ParaProf attempts to offer "best of breed" capabilities to performance analysts – those inherited from a rich history of single processor profilers and those being pioneered in parallel tools research. However, ParaProf should be regarded not as a complete solution, but rather a parallel profile analysis framework that can be retargeted and extended as needed. Thus, in this paper we emphasize, in equal measure, the design of ParaProf and its support for customizability, in addition to its application in the context of the TAU performance system.

In the sections that follow, we first relate ParaProf to selected profiling tools, sequential and parallel, that highlight important features that ParaProf incorporates, or aspires too. Section §3 describes the ParaProf architecture and section §4 goes into more detail of the operation of core components. While ParaProf is being applied in many applications, for this paper, we focus on its novel support for large-scale profile analysis. Section §5 demonstrates ParaProf's core capabilities on a highly-parallel application developed with the SAMRAI framework [8], one of several large-scale parallel environments where ParaProf is being deployed. In conclusion, we remark on how we see ParaProf evolving and its integration in a parallel performance diagnosis environment.

2 Related Work

Performance profiling characterizes the execution behavior of an application as a set of summary statistics associating performance data and metrics with program structure and semantics.[1] Profiling tools are distinguished by two aspects: what performance information is being analyzed, and how profile results are mapped to the program and presented to the user. Since *prof* [17] and *gprof* [5], execution time has been the standard performance data profiled, and the dominant mapping of execution time has been to program source statements. The inclusion of hardware performance counts in profile data (e.g., SGI's *ssrun* [20]) has significantly increased the insight on processor and memory system behavior. The performance API (*PAPI* [2]) provides a common interface to hardware performance counters across microprocessors and is in use by most current profiling tools. The types of program mapping to source statements include statement-, loop-, and routine-level mapping. Callgraph profiling, pioneered in *gprof*, has also been extended to callpath profiling [6].

[1] We can also speak of profiling the performance of a system, but to simplify the discussion, we will focus on application performance.

All of the above profiling features can be found in various forms in sequential profile analysis tools (e.g., cvperf [20], DynaProf [14], and vprof [9]). The HPCView tool [13] best exemplifies the integration of sequential analysis capabilities. Profile data from multiple sources can be input to HPCView, including performance data sets from hardware counters and different program executions. Internally, an extensible profile data format allows different data to be associated with program sites, each identified by a unique name and source mapping information. HPCView can compute derived performance statistics from mathematical expressions that involve performance data variables. The user interface allows navigation, grouping, and sorting of the profile data to assist in results analysis. However, HPCView is not a parallel profile analysis tool, per se. ParaProf can support many of HPCView's features, as well as provide scalable parallel profile analysis.

Parallel profile analysis tools often target specific parallel programming models. The GuideView [10] tool analyzes performance of multi-threaded OpenMP applications. It has sophisticated analysis functions, including handling of multiple experiments and relating performance metrics to ideal values. VGV [7] extends GuideView to hybrid OpenMP and MPI applications. Unfortunately, neither GuideView nor VGV are open systems (now Intel proprietary), and, thus, are able to accommodate more general parallel profile data. The HPM Toolkit [3] also targets OpenMP and MPI applications, with emphasis on the analysis of hardware performance monitor data and derived statistics, but it only runs on IBM platforms. Aksum [4] handles OpenMP and MPI applications run on Linux clusters and can analyze profiles across multiple experiments. SvPablo [15] processes profiles captured for multiple processes on several platforms, and like HPCView, presents performance results in a source-level view. Expert [19] analyzes more extensive profile information to associate performance properties and problems to different program and execution views. With its general event representation and programmable analysis, ParaProf is able process the profile information for many of the scenarios handled by these tools. Moreover, although none of these tools is specifically focussed on large-scale parallelism, ParaProf's profile data management and analysis is scalable to thousands of processors.

3 ParaProf Architecture

Software reuse and componentization lies at the heart of much current research in software engineering. Our goal in the ParaProf project is to apply these design principles to performance profile analysis and visualization. Given the commonalities of profile data and semantics, the opportunity is there to develop a framework for analysis and visualization that can be specialized for the parallel profiling problem. To this effect, we have abstracted four key components in the design of ParaProf: the *Data Source System* (DSS), the *Data Management System* (DMS), the *Event System* (ES), and the *Visualization System* (VS). Each component is independent, and provides well-defined interfaces to other components in the system. The result is high extensibility and flexibility, enabling

us to tackle the issues of re-use and scalability. The remainder of this section describes each these components.

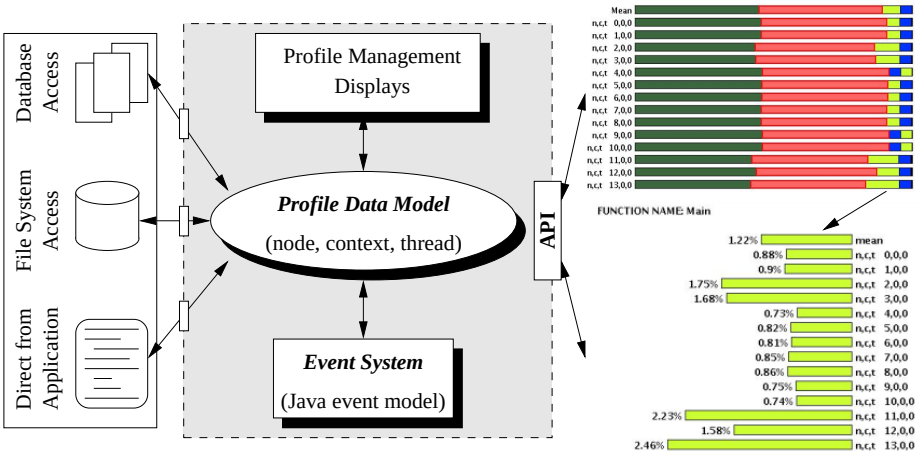

Fig. 1. ParaProf Architecture.

Current performance profilers provide a range of differing data formats. As done in HPCView [13], external translators have typically been used to merge profile data sets. Since much commonality exists in the profile entities being represented, this is a valid approach, but it requires the adoption of a common format. ParaProf's DSS addresses this issue in a different manner. DSS consists of two parts. One, DSS can be configured with profile input modules to read profiles from different sources. The existing translators provides a good starting point to implement these modules. An input module can also support interfaces for communication with profiles stored in files, managed by performance databases, or streaming continuously across a network. Two, once the profile is input, DSS converts the profile data to a more efficient internal representation.

The DMS provides an abstract representation of performance data to external components. Its supports many advanced capabilities required in a modern performance analysis system, such as derived metrics for relating performance data, cross experiment analysis for analyzing data from disparate experiments, and data reduction for elimination of redundant data, thus allowing large data sources to be tolerated efficiently. The importance of sophisticated data management and its support for exposing data relationships is an increasingly important area of research in performance analysis. The DMS design provides a great degree of flexibility for developing new techniques that can be incorporated to extend its function.

The VS components is responsible for graphical profile displays. It is based on the Java2D platform, enabling us to take advantage of a very portable development environment that continues to increase in performance and reliability. Analysis of performance data requires representations from a very fine granular-

ity, perhaps of a single event on a single node, to displays of the performance characteristics of the entire application. ParaProf's current set of displays range from purely textual based to fully graphical. Significant effort has been put into making the displays highly interactive and fast to draw. In addition, it is relatively easy to extend the display types to better show data relations.

Lastly, in the ES, we have provided a well-defined means by which these components can communicate various state changes, and requests to other components in ParaProf. Many of the display types are hyper-linked enabled, allowing selections to be reflected across currently open windows. Support for runtime performance analysis and application steering, coupled with maintaining connectivity with remote data repositories has required us to focus more attention on the ES, and to treat it as a wholly separate component system.

4 Operation

Let us now examine a typical usage scenario. A ParaProf session begins with a top-level profile management window that lets the user decide what performance experiments they want to analyze. An *experiment* is generally defined by a particular application and its set of associated performance profiles coming from experiment *trials*. As shown in Figure 2, the profile manager provides access to different profile sources (e.g., file system, performance database, or online execution) and to different profile experiments. Several profile data sets can be active within ParaProf at the same time. These may be from different experiments, allowing the user to compare performance behavior between applications, or from multiple runs of the same application where each profiled a different performance value. Note that the profile management window is where the user can also specify performance data calculations involving profile data values to derive new performance statistics. We discuss this further below.

Once performance profiles have been selected, ParaProf's view set then offers means to understand the performance data at a variety of levels. The global profile view is used to see profile data for all application events across all threads[2] of execution in a single view. Of course, not all of the profile data can be shown, so the user has control over what performance metrics to display through menu options. The global view is interactive in the sense that clicking on various parts of the display provides a means to explore the performance data in more detail through other views. For example, the global view of a 20-processor VTF [18] parallel profile is shown in Figure 3. The two other views show 1) the performance of one event across all threads, gotten from clicking one color segement, and 2) the performance of all events across a single thread, gotten from clicking on the *node/context/thead* identifier. The full profile data for any thread can be shown in an ASCII view at any time.

[2] We use the term "thread" here in a general sense to denote a thread of execution. ParaProf's internal profile data is organized based on a parallel execution model of shared-memory computing *nodes* where *contexts* reside, each providing a virtual address space shared by multiple threads of execution.

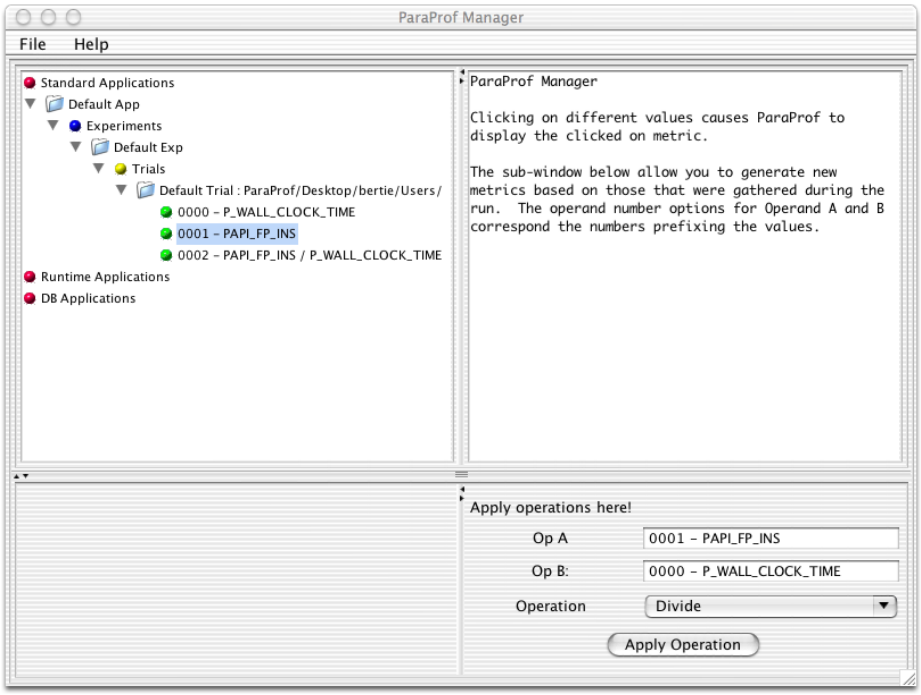

Fig. 2. ParaProf Profile Management Window.

All of the profile views are scrollable. For large numbers of events or large numbers of threads, scrolling efficency is important and we have optimized this in our Java implementation. We have also provided support for sizing down the display bars so that larger numbers of threads can be visualized. Nevertheless, for applications with high levels of parallelism, the two-dimensional profile displays can become unwieldy when trying to understand performance behavior. We have recently implemented histogram analysis support to determine the distribution of data values across their value range. A histogram display shows the distribution as a set of value bins (the number of bins is user defined) equally spaced between minimum and maximum. The number of profile values falling in a bin determines the height of the bin bar in the display.

Concurrent graphical and textual data representation at every stage provides for a seamless translation between levels of data granularity. In addition, the DMS controllers enable easy navigation between data sources thus facilitating understanding of data relationships in the AE. To highlight the importance of this last point, let us consider the following important analysis requirement. Most modern CPUs provide support for tracking a variety of performance metrics. The TAU performance system [16], for example, can simultaneously track CPU cycles, floating point instructions, data cache misses, and so on, using the PAPI library. A consistent problem in performance analysis has been how to examine on a large scale other useful performance metrics that can be derived from

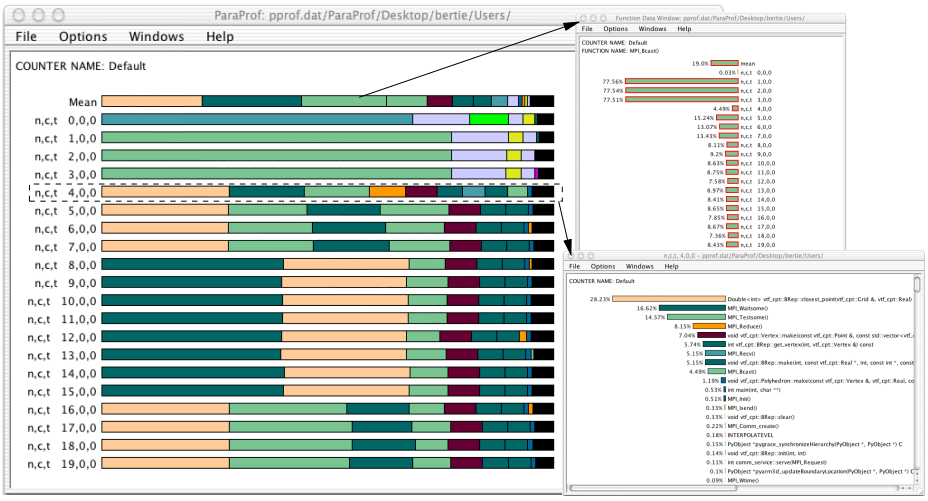

Fig. 3. ParaProf Display of VTF Profile.

measured base values. For example, to determine how efficiently the pipelines of modern super-scalar architectures are being used, our performance system might have access to both CPU cycles, and instruction counts, but gives us no correlation between the two. It is cumbersome to identify performance bottlenecks in CPU pipe utilization across thousands of events or threads if one has to make visual comparisons. To solve such problems, ParaProf's DMS can apply mathematical operations to the performance metrics gathered, thus obtaining more detailed statistics. These operations can be applied to single executions, to executions in different runs, and even across experiments, thus allowing a complex range of derived metrics to be gathered. When examining data across experiments, the DMS can tolerate disparities in data sources (an event might be present in one source, but not another) by examining the commonalities that do exist, and presenting only those. Data sources are treated as operands in the system, and the DMS allows a user to compose operations to produce a variety of derived statistics. Results from this analysis can then be saved for future reference.

To aid in the visual process, the ES passes user activity between windows so that areas of interest highlighted in one window are propagated to all related windows. This greatly reduces the time spent correlating data shown in different displays. Another feature of ParaProf (not show here) is its ability to recognize and create event groupings. Events can be grouped either at runtime by the performance system (supported in TAU), or post-runtime by ParaProf. Displays can then be instructed to show only events that are members of particular groups. This provides another mechanism for reducing visual complexity, and focusing only on points of interest or concern. ParaProf's DMS demonstrated its ability to simultaneously handle the large quantity of data comfortably. Any data redundancies present in the source data were eliminated as expected, and

we showed (via duplication and renaming) that the only practical limits to ParaProf's operation are the memory limitations of the platform. Even these can be alleviated to a great extent by the use of a central repository of data (such as a database) to which ParaProf can be directed to simply maintain links.

5 ParaProf Application

ParaProf, in its earlier incarnation as *jRacy*, has been applied in many application performance studies over the last several years. The additions we have made for profile management, multi-experiment support, derivative performance analysis, performance database interaction, and large-scale parallel profiles, plus overall improvements in efficiency, have elevated the tool to its new moniker distinction. Here we would like to focus on ParaProf's support for profile analysis of large-scale parallel applications. This will be an important area for our future work with the DOE laboratories and ASCI platforms.

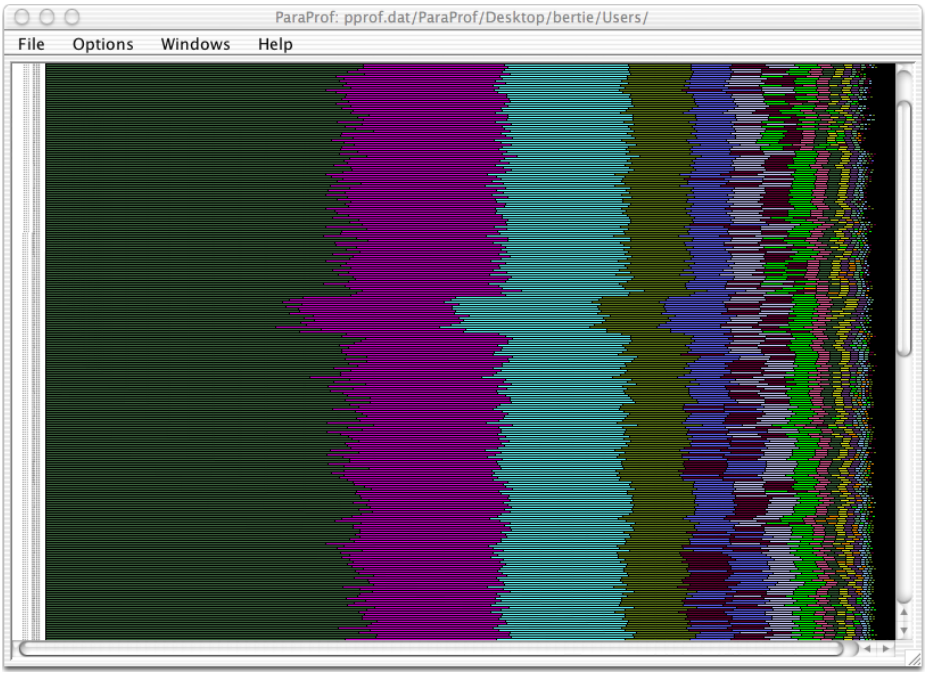

Fig. 4. Scalable SAMRAI Profile Display.

To demonstrate ParaProf's ability to handle data from large parallel applications in an informative manner, we applied ParaProf to TAU data obtained during the profiling of a SAMRAI [8] application run on 512 processor nodes. SAMRAI is a C++ class library for structured adaptive mesh refinement. Figure

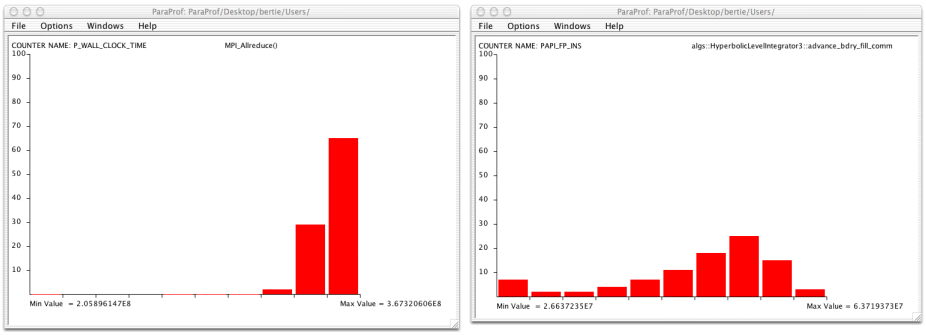

Fig. 5. SAMRAI Histogram Displays.

4 shows a view of exclusive wall-clock time for all events. The display is fully interactive, and can be "zoomed" in or out to show local detail. Even so, some performance characteristics can still be difficult to comprehend when presented with so much visual data. Figure 5 shows one of ParaProf's histogramming options that enable global data characteristics to be computed and displayed in a more concise form. In this case, we see binning of exclusive time data across the system for the routine whose total exclusive time is largest, `MPI_Allreduce()`. We have included another histogram showing the floating point instructions of a SAMRAI module; this data was obtained using TAU's interface to PAPI. Clearly, the histogram view is useful for understanding value distribution.

6 Conclusion

The ParaProf profile analysis framework incorporates important features from a rich heritage of performance profiling tools while addressing new challenges for large-scale paralle performance analysis. Although ParaProf was developed as part of the TAU performance system, our primary goals in ParaProf's design were flexibility and extensibility. As such, we are positioning ParaProf to accept profile data from a variety of sources, to allow more complete performance analysis. We will also continue to enhance ParaProf's displays to aid in performance investigation and interpretation. In particular, there are opportunities to apply three-dimensional graphics to visualize profile results for very large processor runs. Currently, we are developing an advanced visualization library based on OpenGL that can be used by ParaProf.

There are two areas where we want to improve ParaProf's capabilities. First, other parallel profile tools provide linkage back to application source code. The information needed for this is partly encoded in the profile event names, but ParaProf needs to have a standard means to acquire source mapping metadata (e.g., source files, and line and column position) to associate events to the program. We will apply our PDT [12] source analysis technology to this problem, and also hope to leverage the work in HPCView. In addition, source text display and interaction capabilities are required.

The second area is to improve how performance calculations are specified and implemented in ParaProf. Our plan is to develop an easy to use interface to define analysis formalae, whereby more complex expressions, including reduction operations, can be created. These can then be saved in an analysis library for reuse in future performance profiling studies.

References

1. APART, IST Working Group on Automatic Performance Analysis: Real Tools. See http://www.fz-juelich.de.
2. S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A Portable Programming Interface for Performance Evaluation on Modern Processors," *International Journal of High Performance Computing Applications*, **14**(3):189–204, Fall 2000.
3. l. DeRose, "The Hardware Performance Monitor Toolkit," *Euro-Par 2001*, 2001.
4. T. Fahringer and C. Seragiotto, "Experience with Aksum: A Semi-Automatic Multi-Experiment Performance Analysis Tool for Parallel and Distributed Applications," *Workshop on Performance Analysis and Distributed Computing*, 2002.
5. S. Graham, P. Kessler, and M. McKusick, "gprof: A Call Graph Execution Profiler," *SIGPLAN '82 Symposium on Compiler Construction*, pp. 120–126, June 1982.
6. R. Hall, "Call Path Profiling," *International Conference on Software Engineering*, pp. 296–306, 1992.
7. J. Hoeflinger et al., "An Integrated Performance Visualizer for MPI/OpenMP Programs," *Workshop on OpenMP Applications and Tools (WOMPAT)*, July 2001.
8. R. Hornung and S. Kohn, "Managing Application Complexity in the SAMRAI Object-Oriented Framework, *Concurrency and Computation: Practice and Experience*, special issue on Software Architectures for Scientific Applications, 2001.
9. C. Janssen, "The Visual Profiler." `http://aros.ca.sandia.gov/ cljanss/perf/vprof/`.
10. KAI Software, a division of Intel Americas, "GuideView Performance Analyzer," 2001. `http://www.kai.com/parallel/kapro/guideview`.
11. D. Knuth, "An Empirical Study of FORTRAN Programs," *Software – Practice and Experience*, **1**:105–133, 1971.
12. K. Lindlan, J. Cuny, A. Malony, S. Shende, B. Mohr, R. Rivenburgh, C. Rasmussen, "Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates," *Proc. Supercomputing 2000*, November, 2000.
13. J. Mellor-Crummey, R. Fowler, and G. Marin, "HPCView: A Tool for Top-down Analysis of Node Performance," *The Journal of Supercomputing*, **23**:81–104, 2002.
14. P. Mucci, "Dynaprof." `http://www.cs.utk.edu/ mucci/dynaprof`
15. D. Reed, L. DeRose, and Y. Zhang, "SvPablo: A Multi-Language Performance Analysis System," *10th International Conference on Performance Tools*, pp. 352–355, September 1998.
16. TAU (Tuning and Analysis Utilities). `http://www.acl.lanl.gov/tau`.
17. Unix Programmer's Manual, "**prof** command," Section 1, Bell Laboratories, Murray Hill, NJ, January 1979.
18. VTF, Virtual Test Shock Facility, Center for Simulation of Dynamic Response of Materials. `http://www.cacr.caltech.edu/ASAP`.
19. F. Wolf and B. Mohr, "Automatic Performance Analysis of SMP Cluster Applications," Technical Report IB 2001-05, Research Centre Jülich, 2001.
20. M. Zagha, B. Larson, S. Turner, and M. Itzkowitz, "Performance Analysis Using the MIPS R10000 Performance Counters," *Supercomputing '96*, November 1996.