Chapter 1

Performance and Debugging Tools: A Research and Development Checkpoint

Daniel A. Reed[1]
University of Illinois
Urbana, Illinois 61801

Jeffrey S. Brown, Ann H. Hayes, and Margaret L. Simmons
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

"Here they are, boys; get your tools ready." ... As they ran, they pulled weapons from under their coats, hatchets, knuckle-dusters, hammers, and bars of iron.

F. D. Sharpe (1938)

[1]Supported in part by the Advanced Research Projects Agency under ARPA contracts DAVT63-91-C-0029 and DABT63-93-C-0040, by the National Science Foundation under grants NSF IRI 92-12976, NSF ASC 92-12369, ASC 93-08242, NSF CDA94-01124, and by the National Aeronautics and Space Administration under NASA Contract Numbers NGT-51023, NAG-1-613, and USRA 5555-22.

1.1 Introduction

For several years, it has been economically and technically feasible to build parallel systems that scale from tens to hundreds of processors. Recognition of this feasibility has fueled the national High-Performance Computing and Communications (HPCC) program and the fierce competition among new and old high-performance computing companies for a share of the massively parallel market.

Though vendor competition has led to rapid architectural innovation and higher peak hardware performance, it has stretched academic, laboratory, and vendor software tool groups to the limit, forcing them to continually create tools for changing programming models and new hardware environments. By necessity tools embody knowledge of the execution environment, identifying performance bottlenecks or logical program errors in terms of application code constructs and their interaction with the execution environment.

Because the root causes of poor performance or unexpected program behavior may lie with run-time libraries, compilers, the operating system, or the hardware, tools must gather and correlate information from many sources. Not only does this correlation require interfaces for information access, the need for such information is most often gained only from experience. Experience comes with time, as tool developers understand the common programming idioms, the interactions of application code and the underlying hardware and software, and the user interfaces best suited for relating these interactions in intuitive ways. Simply put, developing good tools takes time, experience and substantial effort; small profit margins and short product life cycles, both leading to small installed product bases, have made it difficult for tool developers to create and support effective tools.

The goal of this workshop (and of its predecessors) was to bring together vendor tool developers, academic and government laboratory tool researchers, and application scientists to discuss the current situation and outline research issues and technology transition remedies. New architectures pose important unresolved research problems for tool developers. Moreover, transferring previous research results to products clearly requires new mechanisms if tool developers are ever to catch the speeding train of architectural change.

In this light, the remainder of this introductory chapter is organized as follows. In Sec. 1.2, we describe the motivations for the latest workshop and

the broadening of its scope to include both performance and debugging tools. Based on this context, in Sec. 1.3, we describe the research issues raised by the participants and the implications of these issues for tool research. Finally, because tools lie at the nexus between system software and applications, tool developers must exploit the features of system software to support a user community. This necessitates investment of time and effort in many activities not normally associated with research and, as described in Sec. 1.4, poses a host of difficult problems. Finally, Sec. 1.5 concludes with a summary of recommendations for research and technology transition.

1.2 Workshop Motivations and Experiences

This book contains the papers and working group summaries from discussions during the fifth in a series of workshops on software tools for parallel computer systems. For the first time in the workshop series, developers of both debugging and performance analysis tools met together with application developers and vendors to discuss the technical and sociological problems facing the field. The goal of this combined workshop was the integration of both performance analysis and debugging tools, with the intent of maximizing the return from shared development.

As with the previous workshops in the series [30,33,31,32,40,41], sessions consisted of technical presentations, panels, and working group discussions. Each session of technical presentations included three different perspectives: academic software tool developer, application developer, and computer vendor. Our goal was a dialogue involving all three communities so that all might learn the others' needs and frustrations in building and using tools on parallel systems.

One lesson drawn from the workshop was that tool users and developers speak different languages, often failing to understand the needs or problems faced by the other group. As we will discuss in Sec. 1.4, the attendees concluded, as they did at the 1993 Performance Workshop [32], that a project combining vendors, academics, and users must be undertaken to develop techniques for testing and evaluating tools and encouraging the support and commercialization of effective tools.

A second lesson is that there are new and exciting problems to be solved. The World Wide Web (WWW), distributed metacomputing, and new programming models all pose unsolved problems for debugging and performance

analysis tools. A subset of these issues is summarized in Sec. 1.3 below, and papers by the workshop participants elaborate on these issues in subsequent chapters.

1.3 Research Issues

As with all workshops, the attendees discussed a wide variety of topics, both during the formal sessions and during the extended working groups. Despite the diversity, three major research themes emerged:

- tools for task and data parallel languages,
- techniques for real-time adaptive system control, and
- optimization of heterogeneous metacomputing applications.

All three issues are united by the need for greater access to system internals and data and by the need for standard interfaces, both internally and for users.

Historically, performance and debugging tools have been developed independently of system software and compilers. This separation reflects both the integration of software from disparate sources (i.e., third party compilers and operating system kernels) and limited support provided by software systems for tool development. For example, operating systems typically provide debugger developers little more than a `ptrace` system call for controlling process execution and performance tool developers only a mechanism for accessing a coarse-resolution system clock. Similarly, compilers provide simple symbol table information in object files. For single processor systems, these features are sufficient to build breakpoint debuggers and profilers, though they are far from optimal. For parallel systems and workstation clusters, they are woefully inadequate.

At present, few compilers provide access to program transformation data, though debugging and tuning the performance of programs written in task and data parallel languages (e.g., like High-Performance Fortran (HPF) [20]) requires both compile-time and run-time data. Relating the dynamic behavior of compiler-synthesized code to the user's source code requires knowledge of the program transformations applied by the compiler and the code generation model. For example, if an HPF compiler generates message passing code for a distributed memory system, understanding how messages relate to array

locality is a key to improving performance. Moving beyond tools for explicit parallelism (e.g., via message passing) will require far tighter integration of tools with compilers.

Likewise, few operating systems or run-time libraries provide mechanisms for selecting resource management policies or for configuring those policies based on knowledge of application resource demands or dynamic performance data. However, many experiments have shown that tuning policies to application behavior is key to improving performance for irregular applications with complex behavioral dynamics [17]. For example, Schwan et al. [8] have shown that allowing users to steer application load distribution and to automatically adjust thread locking policies based on expected synchronization delay can substantially improve performance for shared memory codes.

The explosion of WWW [3] use, together with rapidly expanding interests in distributed data mining and heterogeneous parallel computing, pose a different, though equally thorny, set of tool research problems. New types of debugging and performance tuning tools will be needed to create metacomputing applications that can exploit distributed computation resources (e.g., by distributing a computation across multiple, geographically dispersed parallel systems) and that can mine large data archives in response to complex queries. Measuring network latencies and bandwidths and adapting to changes in network loads will necessitate integration of "standard" tools for parallel systems with distributed network management mechanisms (e.g., like the Simple Network Management Protocol (SNMP) [34]).

Support for new programming models, tuning of resource management policies, and management of distributed metacomputations requires interfaces and access to data not readily available via present methods. Below, we briefly describe the data requirements for each of these three domains, with pointers to extended discussions by the workshop participants.

1.3.1 Task and Data Parallel Languages

High-level, data-parallel languages such as HPF [20] have attracted attention because they offer a simple and portable programming model for regular scientific computations. By allowing the programmer to construct a parallel application at a semantic higher level, without recourse to low-level message passing code, HPF is an effective specification language for regular, data parallel algorithms. Similarly, task parallel languages like Fortran M [9]

allow application developers to decompose less regular computations into a group of cooperating tasks.

After investing substantial intellectual effort in developing a task or data parallel program, its execution may yield only a small fraction of peak system performance. And, even if the data parallel code is portable across multiple parallel architectures, it is highly unlikely that it will achieve high performance on all architectures. Even on a single parallel architecture, observed application performance may vary substantially as a function of input parameters. Thus, achieving high performance on a parallel system requires a cycle of experimentation and refinement in which one first identifies the key program components responsible for the bulk of the program's execution time and then modifies the program in the hope of improving its performance.

For this cycle of debugging and performance tuning to be effective and unobtrusive to practicing scientists, performance data must not only be accurate, it and program dynamics must be directly related to the source program. Failure to provide accurate data (or to relate it to the corresponding source code) makes the task of performance improvement and debugging both laborious and error-prone.

Data and task parallel compilers greatly heighten the distance between source language constructs and executable code, making it impossible to map dynamic program behavior to specific source code fragments without knowledge of the program transformations applied by the compiler and the run-time task management strategy. This problem is analogous to developing debuggers for use with code generated by optimizing compilers, though much more difficult.

To understand the causes for performance variability in data parallel codes, one needs high-level performance analysis tools and techniques. Unfortunately, most current performance tools are targeted at the collection and presentation of program performance data when the parallelism and interprocessor communication are explicit and the program execution model closely mimics that in the source code (i.e., as is the case for message-passing codes).

For data parallel languages like HPF, such tools can only capture and present dynamic performance data in terms of primitive operations (e.g., communication library calls) in the compiler-generated code; clearly, this falls far short of the ideal. At a minimum, to support source-level performance analysis of programs in data parallel languages, compilers and performance tools must cooperate to integrate information about the program's dynamic

behavior with compiler knowledge of the mapping from the low-level, explicitly parallel code to the high-level source.

More generally, a new compact between compiler writers and debugger and performance tool developers is needed. Under this compact, the compiler and the tool are co-equals, each providing functions of use to the other — performance data can guide compile-time program optimization, and compilers can provide information needed to estimate program performance scalabilty. In short, it is the basis for developing a set of tools that can break the modify/compile/execute cycle inherent in current optimization and debugging models.

Three presentations at the workshop addressed techniques for supporting performance tuning of task and data parallel languages. Pase and Williams [29] describe techniques for performance analysis of data parallel and message-passing codes on the CRAY T3D, and Nystrom et al. [26] describe their experiences with performance tools on the CRAY T3D. Malony and Mohr [22] describe tools for use with object-oriented languages that leverage the capabilities provided by the Sage compiler toolkit. Finally, Adve et al. [1] describe techniques for supporting performance analysis of Fortran D and HPF programs by exploiting knowledge of compile-time program transformations.

1.3.2 Real-time Adaptive Control

It is increasingly clear that a large and important class of national challenge applications are irregular, with complex, data-dependent execution behavior, and dynamic, with time-varying resource demands. Because the interactions between application and system software change across applications and during a single application's execution, ideally, runtime libraries and resource management policies should automatically and unobtrusively adapt to rapidly changing application behavior. For example, recent studies of application input/output behavior [17,6] have shown that tuning file system policies to exploit knowledge of application access patterns can increase performance by more than an order of magnitude.

Distressingly, the space of possible performance optimizations is large and non-convex, and the best match of application and resource management technique is seldom obvious *a priori*. Current performance instrumentation and analysis tools provide the data necessary to understand the causes for poor performance *a posteriori*, but alone they are insufficient to adapt to temporally varying application resource demands and system responses.

As noted in Sec. 1.3.1, software developers currently must engage in a time-consuming cycle of program development, performance measurement, debugging and tuning to create non-portable code that adapts to parallel system idiosyncrasies.

One potential solution to the performance optimization conundrum is integration of dynamic performance instrumentation and on-the-fly performance data reduction with configurable, malleable resource management algorithms and a real-time adaptive control mechanism that automatically chooses and configures resource management algorithms based on application request patterns and observed system performance. In principle, an adaptive resource management infrastructure, driven by real-time performance data, would increase portability by allowing application and runtime libraries to adapt to disparate hardware and software platforms and would increase achieved performance by choosing and configuring those resource management algorithms best matched to temporally varying application behavior.

Several systems have been built that support application behavior steering (i.e., guiding a computation toward interesting phenomena) [19,28]. Typically, application behavioral steering is interactive, with an application scientist studying near real-time scientific visualizations and guiding the application code by changing key application variables. In contrast to application behavioral steering, there have been fewer efforts to interactively steer or adaptively control application performance. One notable exception is the Falcon system, by Schwan et al. [8], that allows application developers to insert software sensors in their source code to monitor application activity. Performance data from these sensors can activate actuators inserted in the code by the developer. These actuators can change program behavior based on current conditions and measured performance.

As with support for data parallel languages, developing an infrastructure for adaptive control of application and system performance will require new interfaces for information acquisition and control of resource management policies. Task management and input/output libraries must embody mechanisms for resource management policy selection and configuration to match application needs. For example, small sequential file read requests are well served by a input/output library caching and prefetching policy, but large and irregular requests are better served by a policy that directly streams data from storage devices to application code.

Two workshop presentations described experiences with performance steering of parallel systems. Eisenhauer et al. [7] describe the Falcon system for online performance monitoring and adaptive control of application behavior and resource management policies. Kunchithapadam and Miller [21] describe techniques for exploiting breakpoint debuggers and performance tools to build a steering environment.

1.3.3 Heterogeneous Metacomputing and the WWW

The widespread use of workstation clusters for parallel computing is based on their ubiquity, low incremental cost, stable software development environments, and interconnection via well-understood communication protocols (e.g., PVM [10] and [13]). These same communication mechanisms make possible heterogeneous metacomputing where groups of similar or disparate parallel systems are coupled to collectively solve large problems. And as the wide-area research computing network infrastructure evolves to higher bandwidth (e.g., the very high-speed Backbone Network Service (vBNS)), the granularity of feasible distributed computations will continue to decrease until they are limited largely by speed of light latencies due to geographic separation.

However, the truly important change will be the presence of large data archives accessible via the WWW. The WWW has become the standard mechanism for distributed data access and information exchange among large groups of scientists, and an increasingly large fraction of scientific data repositories accept queries via the WWW. Among the key technical challenges will be not only developing faster network access to service these archives, for instance, but regulating these resources as the demand for them skyrockets. The information mining issues inherent in such information availability will require changes in our economic, social, and technical models of computing and cost recovery. In short, parallel computing must be seamlessly integrated with the growing information fabric of the Internet, making parallel, heterogeneous, geographically distributed computing (i.e., metacomputing) the norm.

The hard technical problem will be managing access to distributed data repositories. Typically, one will wish to pose queries that require access to multiple, geographically distributed data bases. The key question becomes when to move the data to the computation and when to move the computation to the data. In short, exploiting the network, for access to the data

archives and high-performance parallel systems for information mining will pose a plethora of difficult performance optimization problems. Because such distributed programs are likely to be based on autonomous agents [12] that cooperate to acquire and process information, distributed debugging and security verification are critical issues.

Two of the four workshop working groups considered the implications of metacomputing in considerable detail. One working group [38] considered tools needed to support development of NII-aware applications. A second group [39] considered tools needed to support workstation cluster computing. As a practical example of the problems debugging and performance tuning problems faced for workstation clusters, Geist et al. [11] and Hao et al. [15] describe their experiences building and supporting tools for PVM [10].

1.3.4 Tool Integration and Interfaces

The emerging debugging and performance tuning problems for task and data parallel languages, real-time adaptive control of irregular applications, and heterogeneous metacomputing are united by the need for new types of data and richer interfaces for acquiring that data. As noted in Sec. 1.3.1–Sec. 1.3.3, this includes access to compiler program analysis databases; runtime library scheduling and data management decisions; and network latencies, bandwidths, and remote system status for distributed applications.

Given the limited resources for tool construction, discussed in Sec. 1.4 below, collaboratively defining standard tool interfaces is key to developing portable tools that can operate on multiple hardware platforms. For example, building portable, compiler-independent HPF performance tools requires definition of access interfaces that export compiler information while hiding internal details. Yet with a few notable exceptions, it is impossible to obtain access to program transformation data without deep knowledge of a particular compiler's internal data structures and access to compiler source code.

Interface standardization must strike a delicate balance between technical needs and vendor competitive constraints. Unrestricted access to compiler transformations will reveal the structure and features of a particular compiler, something that may be viewed as a competitive advantage by a particular vendor. Similarly, access to SNMP network data [34] for distributed task management can reveal much about the structure, function, and capabilities of organizations transporting data on a wide area network. Simply put,

access must be tempered by implementation complexity; portability mandates standards, and a widely implemented, though restricted standard, is preferable to a broad, though rarely adopted standard.

However, standard interfaces for access to compiler data structures and dynamic distributed data are necessary, but not sufficient, to construct a new generation of performance and debugging tools for emerging application domains. Standard, consistent, friendly user interfaces are equally important. Many performance analysis and debugging tools fail, not because they cannot acquire, correlate and present the appropriate data, but because they are difficult or unintuitive to use. There are countless examples of users eschewing more powerful, sophisticated tools in favor of simpler, inaccurate tools. The widespread use of print/write for debugging reflects this fact.

Several workshop presentations and one working group considered external and internal tool interfaces. Breazeal and Ries [4] describe a possible infrastructure for tool development at Intel, Itzkowitz et al. [18] discuss experiences with tool design at Silicon Graphics, and Heller [16] describes design principles for access to data on large production systems. Helmbold and McDowell present a taxonomy of race detection techniques for debuggers. Finally, one of the four workshop working groups [37] and a workshop panel explored the desirability of integrated toolkits.

From a user perspective, Beazley and Lomdahl [2] summarize their experience moving a large code from one parallel system to another and the difficulties they encountered in optimizing the code's performance; Winstead et al. [36] describe their experiences using the Pablo environment to optimize input/output performance on the Intel Paragon XP/S, and Gupta [14] summarizes application experiences with a variety of parallel systems. Finally, Pancake [27] summarizes the experiences of the Parallel Tools Consortium (Ptools) in bringing together users, tools developers, and vendors.

1.4 Technology Transition Issues

Despite major research advances in system software, libraries, and tools, the commercial software infrastructure for massively parallel systems has not kept pace with the rapid evolution of new parallel architectures. The critical importance of robust, flexible, and efficient software tools has been recognized at national HPCC meetings, including those in Pasadena [24,25] and Pittsburgh [35], though little has changed in the past five years. The

workshop, like its predecessors, concluded that there are several fundamental reasons for the current dilemma, and they all relate to the lack of incentives for tool development and testing.

Massively parallel systems are a niche market, and the real cost of commercial software development for these platforms is high. The academic and laboratory research community is another potential source of software tools for parallel systems, but this community lacks the reward structure, the financial resources, and (often) the skills necessary to develop and support robust software tools. Moreover, because there is limited interaction between academic tool developers and potential tool users, many of the commercial and academic tools lack important features, are too difficult to use, or solve problems of little interest to application developers.

1.4.1 Software Tool Shortages

Is the dearth of effective software tools a new problem? No, the lack of good software development tools can be traced to the very origins of high-performance computing (e.g., early CDC Fortran compilers were very inefficient, and the CRAY 1 was delivered with essentially no software). However, as high-performance systems have become more widely available and accessible, the longstanding lack of tools has become more evident. More perniciously, the ferment in the HPCC market has led to short product life cycles, with vendors and application developers repeatedly moving to new architectures and programming models. Often, by the time system software has sufficiently stabilized and there is sufficient knowledge of common programming idioms to permit construction of robust and useful tools, the underlying hardware is no longer performance-competitive.

In short, the software infrastructure for massively parallel systems has been unable to keep pace with the rapid evolution of new parallel architectures. The paucity of basic software tools for application program development, debugging, and performance tuning has limited the use of massively parallel systems to a small cadre of hardy pioneers willing to brave a wilderness of experimental hardware, immature software, and frequently changing tools.

The critical importance of robust, flexible, easy-to-use, and efficient software tools has been emphasized repeatedly at national HPCC meetings, including those in Pasadena [24,25] and Pittsburgh [35]. Despite this recognition, little has changed in the past five years. Substantial research money is

directed toward study and development of software tools, but as one speaker at the 1993 workshop on performance tools [32] asked, "Why don't users use the tools that tool developers develop?" The workshop attendees agreed that the fundamental reasons for this dilemma relate to the lack of incentives for tool development and testing.

First, the massively parallel systems market is small, and the real cost of commercial tool development is high. As vendors prioritize development of new systems, functioning hardware, operating systems and compilers receive the highest priority, for machines cannot be shipped without these. However, competitive pressures and development schedule slippage often lead vendors to ship their new systems prematurely, before other infrastructure such as software development tools can be created. Due to these pressures on vendors to ship, the temptation is very high for them to simply "repackage" internal development tools for external use, even when these tools are inappropriate for users.

From a financial perspective, developing robust software tools for a parallel system with projected sales of 500 units is nearly as costly as developing software for the burgeoning personal computer market, but without concomitant financial incentives. These same economics preclude creation of a viable third party software industry (i.e., the independent software vendors (ISVs)). One of the workshop working groups [23] addressed the implications of market size for tool development.

Second, another potential source of software tools, the academic and laboratory research community, lacks the reward structure, the financial resources, and (often) the skills to develop and support robust software tools. The academic computer science community is rewarded, both tangibly and intangibly, for development of software prototypes and publication of the ideas underlying these, but not for the additional development needed to make these prototypes really usable by the larger HPCC community. Moreover, because there is limited interaction between academic tool developers and potential tool users, prototype tools often lack important features or are simply not useful to application developers.

Although there are a plethora of reasons for this situation, most are subsumed under the rubric of limited collaboration and funding. Inappropriate tools often result because application developers, tool developers, and vendors are not intimate collaborators in the tool development process; current academic and commercial realities do not reward such collaboration. Moreover, vendors are reluctant to embrace, extend, and market academic software

unless it is already robust, as evidenced by wide use within the application community. Finally, an undue emphasis on peak hardware performance, at the frequent expense of sustained, readily achievable performance, leads to rapid changes in hardware and system software.

1.4.2 Potential Technology Transition Solutions

Collaboratively developing robust, user-friendly performance tools for high-performance parallel systems is expensive and as intellectually challenging as national challenge science, yet it is often assumed to be "easy," despite the fact that tool developers on new systems face the same software problems as application developers, and their tools must interoperate with multiple applications. Moreover, effective tool development requires many mundane activities not normally associated with academic computer science research, namely testing early prototypes with friendly users who are application scientists, developing and testing intuitive user interfaces, and writing manuals and documentation. Most of the activities are not rewarded by the computer science research community. Activities such as supporting a user community, teaching training classes, and adapting software to new hardware and software releases will not bring tenure, peer adulation, better graduate students or (usually) larger research support; therefore, most academic computer scientists have little incentive to develop a tool beyond the prototype stage.

The workshop attendees concluded that a solution to this dilemma is the creation of a national software tool evaluation and testing center that would work with external academic researchers, local application scientists, and vendors to evaluate and test prototype software tools. One solution that is currently being explored is the Ptools Consortium [27], a loosely organized group of vendors and users primarily engaged in the development of software tools. However, as described below, the tools community believes a more rigorous, broad-based approach is also required that co-locates a formal testing and evaluation facility with one or more national groups of experienced computational scientists.

These views are not simply those of the editors. They mirror those expressed at the Pasadena [24,25] and Pittsburgh [35,40,41] meetings and at the eight performance and debugging tool workshops. The workshop attendees have repeatedly recommended a major project to understand the limitations of current tools, the requirements for future tools, and to exploit this knowl-

edge to transfer useful software prototypes to the application community and to commercial vendors.

Clearly, however, one center cannot and should not supplant traditional mechanisms for technology transfer from academia to industry. We would envision a more synergistic relationship with these traditional mechanisms. Moreover, it is important to realize that there are many possible definitions of software tool "success" that do not include commercialization. Indeed, as argued earlier, commercialization is extraordinarily difficult, and many valuable tools have no commercial market. Pragmatically, success means that a software tool is useful, tested, documented, and widely used. Hence, the goal of a center would be to serve as a focal point for the software tool development, vendor, and computational science communities to increase the number and breadth of useful and necessary tools and to encourage commercialization. To realize this goal, the workshop attendees and organizers envision four foci.

1. *Early Prototype Evaluation*: Typically, academic tool researchers develop simple software prototypes to demonstrate a proof of concept prior to publication. Because most of these projects are small, there are few opportunities to test the ideas with appropriate external users and to learn what aspects of the approach have practical merit. By coupling academic tool researchers with a group of friendly users at supercomputing centers, where there is the infrastructure needed to support early prototypes, the tool researchers can gain early feedback on the practicality of their ideas. Those prototypes that show promise of potential usefulness to application developers could then be further developed with the center's assistance, and where appropriate, in cooperation with one or more vendors. An additional bonus from this stage would be also to gain new research ideas.

2. *Mature Prototype Testing and Extension*: A smaller number of prototypes are sufficiently mature to be used and useful to a user community not co-located with the tool developer. Major reasons for the importance of co-location include tool bugs, lack of documentation, missing features, and support for only a small number of parallel hardware platforms. Proximity to developers increases the likelihood of interaction to aid in overcoming these deficiencies. This class of tools, however, have passed an initial "usefulness test" — they are being used by at least a portion of the application community. Hence, the second focus

of a center would be to work with users to aggressively test these tools, identify bugs and inadequacies, work with the original tool developers to fix those bugs and add missing features, and (where appropriate) extend the tool to other hardware platforms.

3. *Vendor Cooperation and Involvement*: Both the promising early prototypes and the more useful, mature tool prototypes should have vendors involved in their evaluation and testing early in the cycle if commercialization is at all an option. Not every tool "vetted" by the center would be a candidate for adoption by one or more vendors; however, one would hope that many would be viewed as sufficiently useful to be of interest to vendors of high-performance parallel systems.

4. *Software Packaging and Support*: Finally, tools deemed useful and worthy of dissemination, but not adopted by one or more vendors, must be documented, packaged for installation at remote sites, and, when problems arise, supported, patched, and upgraded. Although this work is mundane, some entity must assume this responsibility, else even good tools will not be used for long periods.

The need for "better" software performance analysis and debugging tools (where better means easier to use, more efficient, better integrated, and more informative) for high-performance parallel systems is a well-documented and widely recognized need. The Strategic Implementation Plan [5] of the *Committee on Information and Communications of the National Science and Technology Council* has noted that "Raising the productivity of the software industry through simplifying toolkits can yield significant dividends in the international marketplace and enable more rapid introduction of hardware advances into affordable production systems." Raising the productivity of applications developers through appropriate, easy-to-use software performance and debugging tools creates a larger market for these affordable production systems. Providing a place where these tools can be effectively tested, evaluated and improved can ensure success in the entire high-performance parallel computing industry.

1.5 Conclusions

Building successful software tools requires a varied mix of technical expertise, marketing and evangelization, and software support. As systems become in-

creasingly complex with shorter and shorter product cycles, the performance optimization and debugging problems become more intellectually challenging, and deploying effective tools based on anticipated user needs becomes increasingly difficult.

The goal of this workshop (and of its predecessors) was to bring together vendor tool developers, academic and government laboratory tool researchers, and application scientists to discuss the current situation and outline research issues and technology transition remedies. The remaining chapters of this book capture the current state of the tool development art.

References

[1] ADVE, V. S., MELLOR-CRUMMEY, J., ANDERSON, M., KENNEDY, K., WANG, J.-C., AND REED, D. A., "Integrating Compilation and Performance Analysis for Data Parallel Programs," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[2] BEAZLEY, D. M., AND LOMDAHL, P. S., "A Practical Approach to Portability and Performance Problems on Massively Parallel Supercomputers," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds. IEEE Computer Society Press, Dec. 1995.

[3] BERNERS-LEE, T., CAILLIAU, R., GROFF, J., AND POLLERMANN, B., World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications, and Policy 1*, **2** (1992), 52–58.

[4] BREAZEAL, D., AND RIES, B., "A Building Block Approach to Parallel Tool Construction," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[5] COMMITTEE ON INFORMATION AND COMMUNICATIONS, NATIONAL SCIENCE AND TECHNOLOGY COUNCIL. Strategic Implementation Plan: America in the Age of Information, Mar. 1995.

[6] CRANDALL, P. E., AYDT, R. A., CHIEN, A. A., AND REED, D. A., "Characterization of a Suite of Input/Output Intensive Applications," in *Proceedings of Supercomputing '95* (Dec. 1995).

[7] EISENHAUER, G., GU, W., KINDLER, T., SCHWAN, K., SILVA, D., AND VETTER, J., "Opportunities and Tools for Highly Interactive Distributed and Parallel Computing," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[8] EISENHAUER, G., GU, W., SCHWAN, K., AND MALLAVARUPU, N., "Falcon — Toward Interactive Parallel Programs: the Online Steering of a Molecular Dynamic Program," in *Proceedings of the Third International Symposium on High-Performance Distributed Computing* (Aug. 1994).

[9] FOSTER, I., AVALANI, B., CHOUDHARY, A., AND XU, M., "A Compilation System that Integrates High Performance Fortran and Fortran M," in *Proceedings of the 1994 Scalable High Performance Computing Conference* (1994).

[10] GEIST, A., BEGUELIN, A., DONGARRA, J., JAING, W., MANCHEK, R., AND SUNDERAM, V., *PVM: Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, 1994.

[11] GEIST, G. A., KOHL, J., AND PAPADOPOULOS, P., "Visualization, Debugging, and Performance in PVM," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[12] GOSLING, J., AND MCGILTON, H., "The Java Language Environment: A White Paper," Sun Microsystems, 1995.

[13] GROPP, W., LUSK, E., AND SKJELLUM, A., *Using MPI*, MIT Press, 1994.

[14] GUPTA, R. "Prospects of Solving Grand Challenge Problems," in *Debugging and Performance Tuning for Parallel Computer Systems*,

M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[15] HAO, M. C., WAHEED, A., KARP, A., AND JAZAYERI, M., "Multiple Views of Parallel Application Execution," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[16] HELLER, D., "Issues of Running Codes on Very Large Parallel Processing Systems," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[17] HUBER, J. V., ELFORD, C. L., REED, D. A., CHIEN, A. A., AND BLUMENTHAL, D. S., "PPFS: A High-Performance Portable Parallel File System," in *Proceedings of the 9th ACM International Conference on Supercomputing* (July 1995).

[18] ITZKOWITZ, M., YU, J., MCNAUGHTON, A., ORELUP, P., AND HANNA, C., "Visualizing Performance on Parallel Supercomputers," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[19] JABLONOWSKI, D., BRUNER, J., BLISS, B., AND HABER, R., "VASE: The Visualization and Application Steering Environment," in *Proceedings of Supercomputing '93* (Nov. 1993), 560–569.

[20] KOELBEL, C., LOVEMAN, D., SCHREIBER, R., STEELE, JR., G., AND ZOSEL, M., *The High Performance Fortran Handbook*, The MIT Press, Cambridge, MA, 1994.

[21] KUNCHITHAPADAM, K., AND MILLER, B. P., "Integrating a Debugger and a Performance Tool for Steering," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[22] MALONY, A., AND MOHR, B., "Program Analysis and Tuning Tools for a Parallel Object Oriented Language: An Experiment with the TAU

System," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[23] McGraw, J. M., "Leveraging software resources," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[24] Messina, P. and Sterling, T., Eds., *Pasadena Workshop on System Software and Tools for High-Performance Computing Environments*, SIAM, Jan. 1992.

[25] Messina, P. and Sterling, T., Eds., *Second Pasadena Workshop on System Software and Tools for HPC Environments*, Jan. 1995.

[26] Nystrom, N. A., Young, W. S., and Wimberly, F. C., "Methodologies for Developing Scientific Applications on the CRAY T3D," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[27] Pancake, C. M., "Collaborative Efforts to Develop User-Oriented Parallel Tools," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[28] Parris, M., Mueller, C., and Prins, J., "A Distributed Implementation of an N-body Virtual World Simulation," in *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems* (Apr. 1993), 66–70.

[29] Pase, D. M. and Williams, W., "A Performance Tool for the Cray T3D," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[30] Simmons, M., Koskela, R., and Bucher, I., Eds., *Instrumentation for Future Parallel Computing Systems.* Addison-Wesley Publishing Company, 1989.

[31] SIMMONS, M. L., HAYES, A. H., AND REED, D. A., Santa Fe Workshop on Parallel Computer Systems, Oct. 1991, Santa Fe, NM.

[32] SIMMONS, M. L., HAYES, A. H., AND REED, D. A., Keystone Workshop on Software Tools for Parallel Computing Systems: A Dialogue Between Users and Developers, Apr. 1993, Keystone, CO.

[33] SIMMONS, M. L., AND KOSKELA, R., Eds., *Parallel Computing Systems: Performance Instrumentation and Visualization*, Addison-Wesley Publishing Company, 1990.

[34] STALLINGS, W., *SNMP, SNMPv2 and CMIP: The Practical Guide to Network Management Standards*, Addison-Wesley Publishing Company, 1993.

[35] STEVENS, R., Workshop and Conference on Grand Challenge Applications and Software Technology, May 1993, Pittsburgh, PA.

[36] WINSTEAD, C., PRITCHARD, H. P., AND MCKOY, V., Tuning I/O Performance on the Paragon: Fun with Pablo and Norma. In *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[37] ROVER, D., MALONY, A., AND NUTT, G., "Integrated Environments Versus Toolkits," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[38] DILLY, R., "Tools and the NII," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[39] MCGRAW, J., "Tools for Workstation Clusters," in *Debugging and Performance Tuning for Parallel Computer Systems*, M. L. Simmons, A. H. Hayes, D. A. Reed, and J. Brown, Eds., IEEE Computer Society Press, Dec. 1995.

[40] BROWN, JEFFREY S., Supercomputer Debugging Workshop '91, November 1991, Albuquerque, NM.

[41] BROWN, JEFFREY S., Supercomputing Debugging Workshop '92, October 1992, Dallas, TX.