I/O, Performance Analysis, and Performance Data Immersion*

Daniel A. Reed
Tara Madhyastha
Ruth A. Aydt

Christopher L. Elford
Will H. Scullin
Evgenia Smirni

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Abstract

A large and important class of national challenge applications are irregular, with complex, data dependent execution behavior, and dynamic, with time varying resource demands. We believe the 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. Within the context of parallel input/output optimization, we describe the components of such a closed-loop control system based on the Pablo performance analysis environment, a portable parallel file system (PPFS), and virtual environments for study of dynamic performance data and interactive control of file system policies.

1 Introduction

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. For example, recent studies of application input/output behavior [2] showed that tuning file system policies to exploit knowledge of application access patterns can increase performance by more than an order of magnitude [5].

Unfortunately, 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.* Moreover, because the interactions between application and system software change across applications and during a single application's execution, runtime libraries and resource management policies should automatically and unobtrusively adapt to rapidly changing application behavior.

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. We believe the 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.

Based on this thesis, we describe the ongoing integration of the Pablo performance analysis environment [15, 14, 16] with input/output analysis software [2], flexible parallel file system policies [5], immersive virtual environments for interactive file system policy selection and optimization [18, 20], and an infrastructure for automatic classification of input/output behavior [8]. We also briefly describe Pablo's role in the Scalable I/O Initiative [13], a wide-ranging project to characterize application input/output performance, develop new language interfaces and file systems, and prototype input/output software.

The remainder of this paper is organized as follows. In §2, we describe performance instrumentation and analysis techniques, and summarize the lessons learned during the development of the Pablo performance analysis environment. In §3, we describe our experiences using software performance instrumentation techniques to capture and analyze the input/output dynamics of large, input/output intensive parallel applications and the relation of this effort to the new Scalable I/O Initiative. This is followed in §4 by a brief description of a flexible infrastructure for studying parallel file system policies, and in §5 by a description of virtual environments for study of dynamic performance data and interactive control of file system policies. In §6 we sketch our vision of a closed-loop adaptive control system for resource management policy selection and tuning, and in §7 we describe re-

*This work was 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, and NSF CDA 94-01124, and by the National Aeronautics and Space Administration under NASA Contracts NGT-51023, NAG-1-613, and USRA 5555-22.

lated work. Finally, §8 summarizes the current state of our work and outlines plans for continued research.

2 Performance Analysis Techniques

In [15], we argued that an ideal performance analysis environment should support interactive insertion of instrumentation points, as well as subsequent data analysis, reduction, and display. For monitoring and adaptive control of resource management policies, performance environments must also support real-time data analysis and extraction. These constraints dictate performance environments that are *portable* across a range of parallel architectures, allowing resource management policy tuning on multiple parallel systems, *scalable* with the size of the system being studied, allowing optimization on large systems, and *extensible*, allowing addition of environment functionality as needed.

These are not mere suppositions, but are based on our experiences building three generations of performance analysis tools for multiple generations of parallel systems [9, 17, 15]. In an earlier collaboration with Intel, we developed a distributed memory performance analysis environment that included operating system and application software instrumentation [17], hardware support for performance data capture [10], and a data analysis and visualization system [11, 12]. Although this environment provided detailed data on system and application software behavior, it could not be ported to other parallel systems without major modifications, it could not be extended without intimate knowledge of the environment's internal software structure, and could it not accommodate hundreds of processors.

To redress these problems, for the past five years, we have worked to develop a software performance analysis infrastructure, called Pablo [15, 14, 16], for portable, extensible performance data capture and analysis. Below, we describe the Pablo environment, with emphasis on those components that support real-time performance monitoring and adaptive control.

2.1 Pablo Software Infrastructure

The complete Pablo[1] environment consists of

- an extensible performance data metaformat (SDDF) and associated library that separates the structure of performance data records from their semantics, allowing easy addition of new performance data types,
- an instrumenting parser capable of generating instrumented SPMD source code,
- extensible instrumentation libraries that can capture event traces, counts, or interval times and reduce the captured performance data on the fly, and
- graphical performance data display and sonification tools, based on coarse-grain graphical programming, that support rapid prototyping of performance analyses.

[1] Pablo is a registered trademark of the Board of Trustees of the University of Illinois.

As Figure 1 suggests, the Pablo instrumentation software captures dynamic performance data via instrumented source code that is linked with a data capture library. During program execution, the instrumentation code generates performance data that can either be directly recorded by the data capture library or processed by one or more data analysis extensions prior to recording. After program execution completes, the data can be analyzed by a toolkit of data transformation modules capable of processing the self-describing data format and a graphical programming model that allows users to interactively connect and configure a data analysis graph.

This Pablo software infrastructure has been used as a basis for a portion of Intel's performance analysis tools on the Intel Paragon XP/S [19] and has been integrated with data parallel compilers to study the performance of Fortran D and HPF codes [1]. For additional details on the Pablo software design philosophy and our experiences, see [15, 14, 16].

2.2 Pablo Software Extensibility

Given the rapid changes in hardware platforms and programming models and the performance optimization problems inherent in an evolving market, software performance tools must be capable of evolving to accommodate unanticipated needs. Since its development, we have retargeted the Pablo software to study application input/output patterns [2], World Wide Web access patterns [6], data parallel languages [1], and parallel file system policies [5].

One of the keys to the Pablo environment's adaptability has been the inclusion of a self-defining data format (SDDF). SDDF data streams consists of a group of record descriptors and record instances. Much as structure declarations in the C programming language specify templates for storage allocation, SDDF descriptors define the structure for record instances. The data following the descriptors consists of a stream of descriptor tag and data record pairs. Each tag identifies the descriptor that defines the juxtaposed data. By separating the structure of data from its semantics, the Pablo SDDF library permits construction of tools that can extract and process SDDF records and record fields with minimal knowledge of the data's deeper semantics.

The second key to Pablo's adaptability has been the design of a set of software extension interfaces for the Pablo data capture library. Via these interfaces, one can register a set of performance data analysis modules that create and process performance data records prior to extraction. The function of such extensions can range from simple matching of procedure entry/exit records for computing procedure invocation lifetimes to sophisticated calculation of input/output activity histograms based on input/output activity traces or correlation of dynamic program behavior with data parallel source code [1].

Finally, the ability to extract performance data via several mechanisms (e.g., files or network sockets) has enabled a variety of data analysis modes. Real-time transmission of captured (and optionally processed) performance data from each node of a parallel sys-

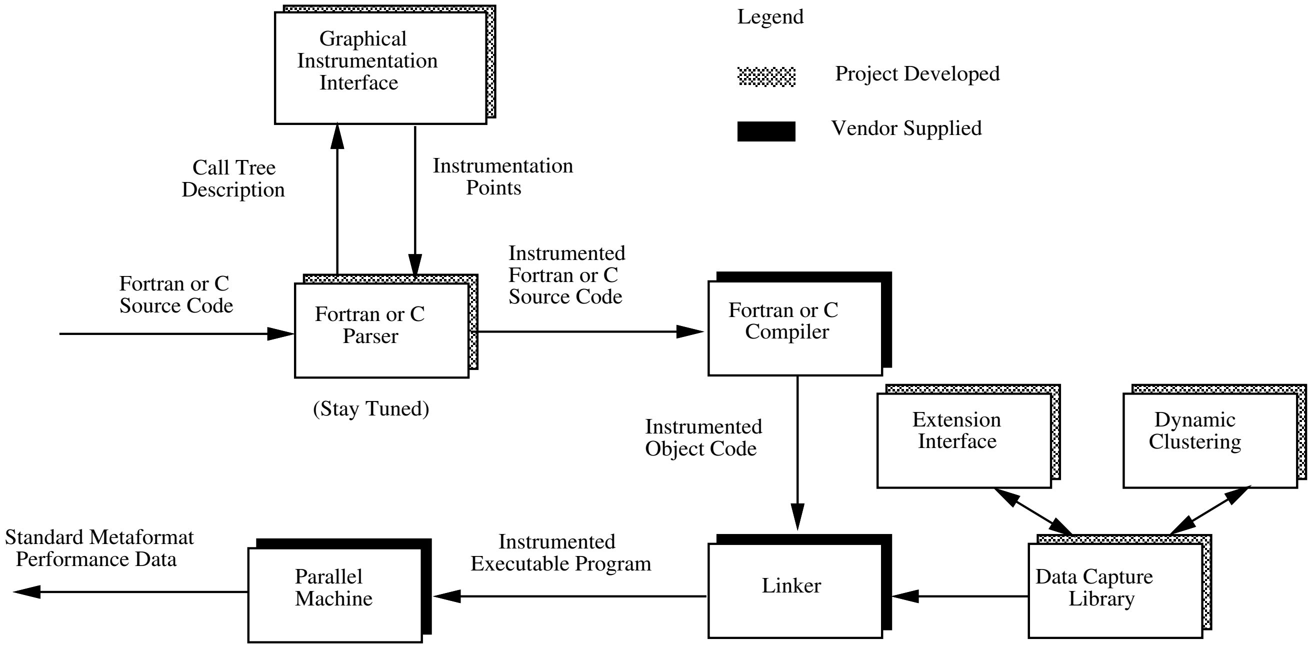


Figure 1: Pablo Instrumentation Infrastructure

tem makes possible real-time display and, with the requisite interfaces, interactive control of application or system behavior. As described in §5, we have used this capability of the Pablo library to develop a virtual environment for real-time data display and control.

3 Input/Output Characterization

The acceptance of massively parallel systems as the vehicle of choice for high-performance computing has produced a wide variety of machines and an even wider variety of potential input/output configurations, most with inadequate input/output capacity and performance. Understanding application input/output patterns is the first step to optimizing application performance with extant file systems and to developing more effective parallel input/output file system policies. By extending the Pablo performance instrumentation and analysis software, we have created an infrastructure for quantifying current parallel input/output patterns in parallel applications and measuring file system responses to those patterns.

3.1 Pablo Input/Output Extensions

Any instrumentation system must strike a balance between instrumentation detail and perturbation of application behavior. When characterizing input/output behavior, use of the input/output system to extract performance data can lead to particularly pernicious perturbations. To capture and analyze application input/output data while minimizing input/output perturbations, we exploited the Pablo data capture library's extension interfaces to develop a suite of input/output analysis routines that support both real-time calculation of statistical summaries and capture of detailed event traces. The former trades computation perturbation for input/output perturbation.

Detailed input/output event traces include the time, duration, size, and other parameters of each input/output operation. Statistical summaries can take any one of three forms: file lifetime, time window, or file region. File lifetime summaries include the number and total duration of file reads, writes, seeks, opens, and closes, as well as the number of bytes accessed for each file, and the total time each file was open. Time window summaries contain similar data, but allow one to specify a window of time for summarization. Finally, file region summaries are the spatial analog of time window summaries; they define a summary over the accesses to a file region.

3.2 A Characterization Example

During the past two years, we and others have used the Pablo input/output analysis software to study the behavior of a wide variety of parallel applications on the Intel Paragon XP/S and IBM SP/2 [2]. These applications represent a snapshot of current input/output practice on scalable parallel machines and reflect the developers' input/output design choices based on perceived and actual limitations of available input/output systems. These initial codes are but a small part of the nascent Scalable Input/Output initiative's (SIO) code suite, described in §3.4.

As one example, we have tracked the performance of multiple versions of an electron scattering code [21] across three releases of the Intel OSF/1 parallel file system (PFS) on the Caltech Paragon XP/S. Electron scattering results are of interest in many contexts, including semiconductor fabrication.

This particular electron scattering code computes

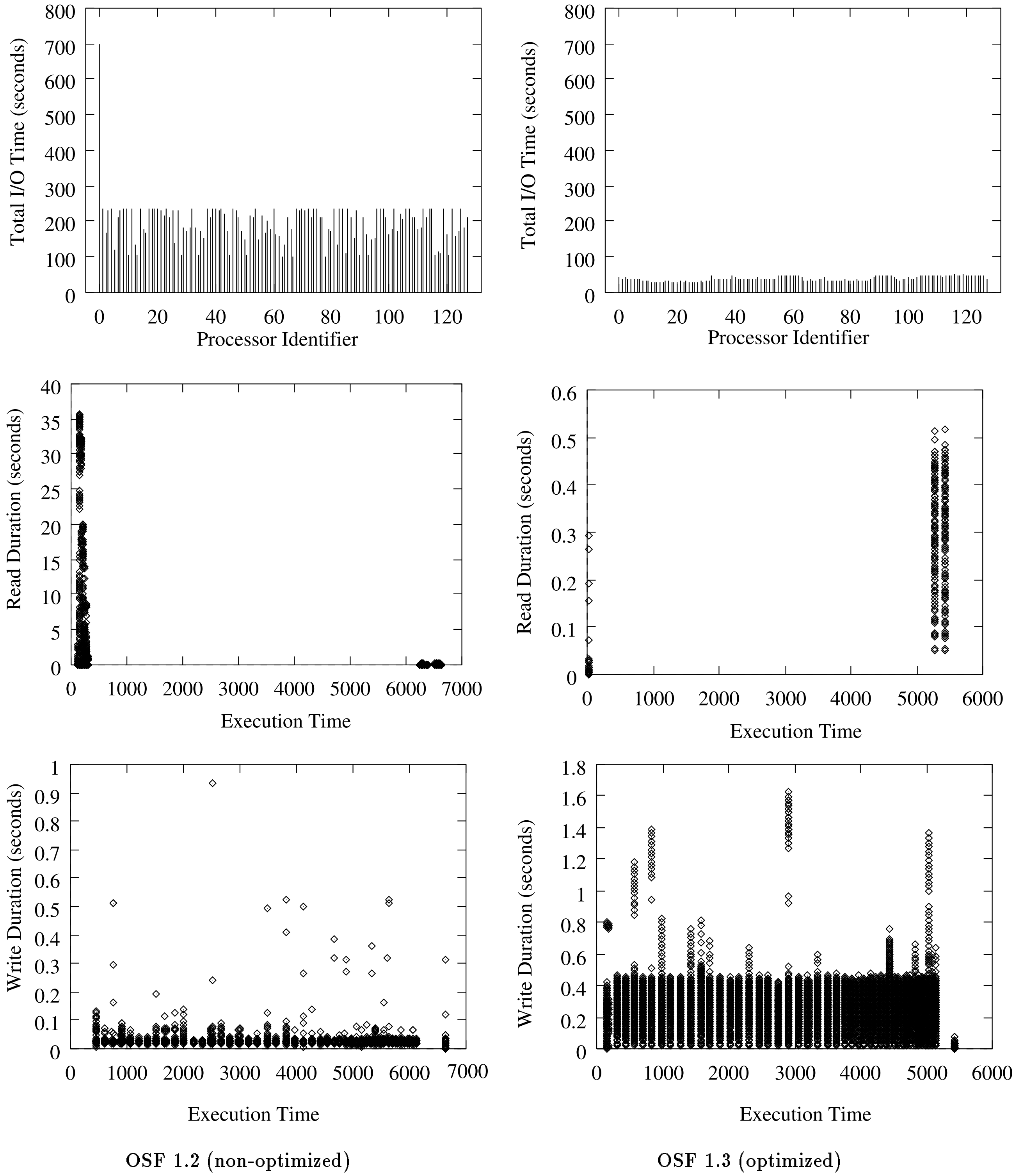


Figure 2: Intel Paragon XP/S Input/Output Performance (Electron Scattering Code)

the scattering probabilities for low-energy electron-molecule collisions by solving linear systems whose terms are obtained via numerical quadrature. Because the quadrature data is too voluminous to fit in the memory of the processors, it is written to disk and retrieved when needed. For a modest, though representative data set, the electron scattering code executes for roughly one and three quarter hours on 128 processors of the Intel Paragon XP/S. Production data sets have similar behavior, but with ten to fifty hour executions on 256 or 512 processors and much larger input/output data volumes.

Figure 2 shows the behavior of two snapshots of this code. In the first version, measured on Intel OSF 1.2, all processors concurrently open and read a set of initialization files with a variety of request sizes. Following this, all the processors compute and a single processor writes the quadrature data. A small interval of data reuse occurs near the end of the computation. In the second version, measured sixteen months later on Intel OSF 1.3, a single processor reads the initialization data in large chunks and broadcasts the data to the other nodes. The processors write quadrature data, but in larger units matched to the stripe size of the Intel parallel file system.

As Figure 2 shows, the effects of these application optimizations and improvements in the Intel parallel file system are striking. By avoiding concurrent file opens and reads, file system contention is reduced, and total input/output time decreases nearly an order of magnitude. Likewise, because the Intel file system does no caching and is optimized for large requests, increasing the read and write request sizes better exploits the file system capabilities and reduces total read/write time even though the individual write times increase.

3.3 Input/Output Implications

As analysis of the electron scattering code's behavior suggests, there are many opportunities for improving the performance of current parallel file systems (e.g., by better support for caching and prefetching) and to reduce the burden faced by application developers when tuning their codes to exploit file system idiosyncrasies. In particular, our characterization study [2] has shown that parallel applications exhibit a wide variety of input/output request patterns, with both very small and very large request sizes, sequential and non-sequential access, and a variety of temporal variations.

Small input/output requests are best managed by aggregation, prefetching, caching, and write-behind, though large requests are better served by streaming data directly to or from storage devices and application buffers. In consequence, we believe that design of parallel file systems that rely on a single, system-imposed file system policy is unlikely to be successful. In short, exploitation of input/output access pattern knowledge in caching and prefetching systems is crucial to obtaining a substantial fraction of peak input/output performance. Inherent in such an adaptive approach is the need to identify access patterns and choose policies based on access pattern characteristics. We will return to these topics in §4 and §6.

3.4 Scalable I/O Initiative

The Scalable I/O (SIO) Initiative [13] is a newly launched multi-agency (ARPA, DOE, NSF, and NASA) research program that includes application and system input/output characterization, networking, file systems and file system application programming interfaces (APIs), compiler and language support, and basic operating system services. The primary hardware testbeds for this research are the Intel Paragon XP/S at Caltech, the IBM SP-2 at Argonne National Laboratory, and the Convex Exemplar and Silicon Graphics Power Challenge Array at NCSA.

Organized as five major working groups (performance characterization, operating systems, languages and runtime systems, applications, and software integration) that include application developers, academic researchers and vendors, the goal of the initiative is a broad-based attack on the input/output problem that will produce not only new input/output optimization ideas but also the embodiment of these ideas in system software prototypes. In addition to the five working groups, two multidisciplinary groups are designing APIs for parallel file systems and developing a suite of parallel input/output benchmarks.

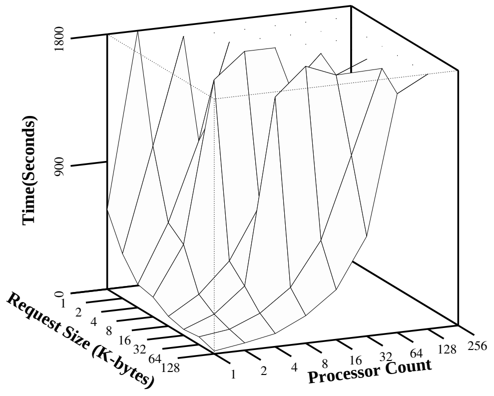
The SIO initiative builds on our earlier input/output characterization effort — the Pablo performance environment's input/output analysis software is the basis for the SIO initiative's study of application input/output behavior. In addition, we are working with Intel and other vendors to augment disk and network device drivers with instrumentation to record physical input/output behavior and to integrate this support with an extended version of the Pablo performance data capture software. By correlating application input/output request patterns with system input/output responses, one can quantify the efficacy of particular file system policies and, as described in §6, exploit this data to adaptively tune file systems.

4 Flexible Parallel File Systems

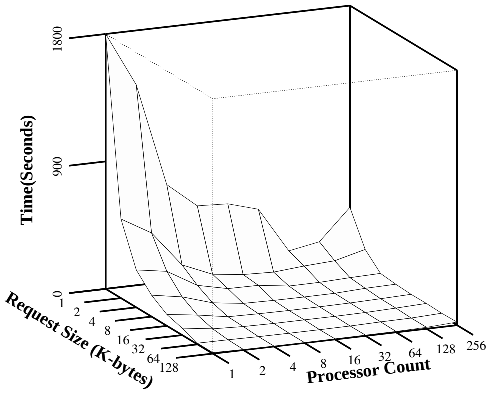
The data from our application input/output studies show that there is wide variation in temporal and spatial access patterns, and it seems unlikely that a single file system policy can minimize physical input/output operations and overlap all unavoidable physical input/output with application computation. To explore the thesis that performance is best maximized by tuning file system policies to application behavior, we continue to develop a portable parallel file system (PPFS) [5] to study the interaction of application access patterns, file caching and prefetching algorithms, and application file data distributions.

4.1 PPFS Architecture

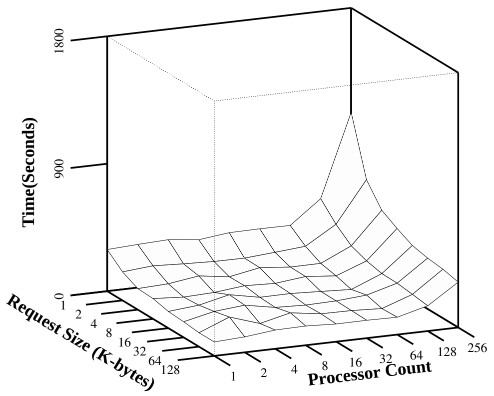
PPFS consists of a user-level input/output library that is portable across both parallel systems (the Intel Paragon XP/S, IBM SP/2, and Convex Exemplar) and workstation clusters. Although a user-level implementation sacrifices control of low-level details (e.g., file block placement), it allows us to quickly extend the software with minimal effort and to conduct a



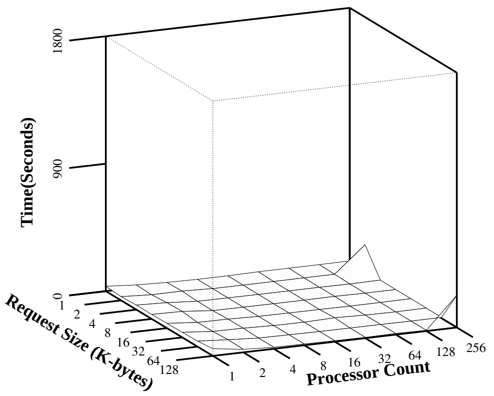
(a) Reads (PFS OSF 1.2)



(b) Writes (PFS OSF 1.2)



(a) Reads (PPFS)



(b) Writes (PPFS)

Figure 3: Intel Paragon XP/S PPFS Benchmark Performance (Sixteen I/O Nodes)

wide range of experiments more quickly than would be possible via system software modifications.

As Figure 6 shows, the key software components of PPFS are clients, servers, metadata server(s), and caching agents. A client consists of an instance of the user application code and the local caching, prefetching, and bookkeeping software that permits an application to use PPFS. Clients communicate with the metadata server to determine which data servers and caching agents will satisfy input/output requests. Caching agents are shared caches via which multiple clients share files. Finally, input/output servers are the ultimate resolvers of requests. Via this architecture, PPFS provides application control of a variety of file caching, prefetching, data layout, and coherence policies.

4.2 PPFS Instrumentation

To analyze the internal behavior of PPFS, we have used the Pablo instrumentation environment to capture event traces of internal PPFS state transitions, procedure calls, and input/output events. In addition, each PPFS client and server maintains data on file system performance (e.g., queue lengths and delays, service times, and request throughputs). Together, this provides the data needed to understand the efficacy of particular policy choices and configurations and, as we shall see in §6, to interactively adjust policy parameters.

4.3 PPFS Experiences

By carefully choosing PPFS parameters and data management policies to match application access patterns, it is possible to increase application input/output performance by an order of magnitude over that achievable with vendor parallel file systems [5]. Intuitively, request aggregation, write behind, prefetching, and caching policies match the application request stream to the capabilities of the underlying file system.

As an example of the efficacy of matching file system policies to application access patterns, Figure 3 shows the time to read and write a 64 MB file using a variety of request sizes and numbers of processors on an Intel Paragon XP/S with sixteen I/O nodes; see [5] for details. For the read benchmark, each processing node reads the entire file. For the write benchmark, each processing node writes an equal portion of the file, with record level interleaving.

Because PFS does not cache or prefetch data, large numbers of small reads are inefficient — each read sees the full input/output latency and the input/output nodes each see large numbers of small requests. Small writes suffer from many of the same problems as small reads — file system call overhead dominates total execution time.

By matching policies to access patterns, PPFS combines multiple pending read requests for the same record. Subsequent requests for the same record can be serviced from the server caches. For writes, PPFS aggregates multiple, small requests, creating large, contiguous data blocks that can be written efficiently.

In general, our experiments with large research codes have shown that investment in a malleable infrastructure is repaid with increased input/output performance. Tuning the file system policies to application needs, rather than forcing the application to use inappropriate and inefficient file access modes, is the key to performance. However, such tuning requires either the application developer to provide file system hints based on knowledge of access patterns or automated adaptation based on observed application behavior. Because our experience has shown that developers often do not understand nor wish to understand detailed input/output behavior, we believe automated adaptation is most appropriate.

5 Performance Data Immersion

Given the sensitivity of performance to the interaction of application request patterns with parallel system resource management policies, it is rarely possible to predict which of the potentially hundreds of performance metrics will best illuminate this interaction. In principle, it is possible to capture all metrics of potential interest and analyze the data off-line; in practice, the large volume of performance data makes this approach impractical. Instead, interactive selection and real-time display of performance metrics are necessary.

Although animated, two-dimensional graphics displays can provide some insights, understanding the evolving relations among abstract, multivariate data remains a difficult task. Understanding the shape of complex, three-dimensional objects is greatly simplified by reliance on binocular stereo images and our ability to change the viewing perspective — head and body movements allow one to compare a series of images.

By exploiting the Pablo environment's support for real-time data extraction, we have developed an immersive virtual environment, called Avatar [20, 18], for real-time, three-dimensional display of dynamic performance data. Avatar operates with (a) a head-mounted display and tracker, (b) the CAVE [3] virtual reality theater, or (c) a workstation display with stereo glasses. At present, Avatar supports three domain-independent display metaphors that accept real-time data from the Pablo instrumentation library (a three-

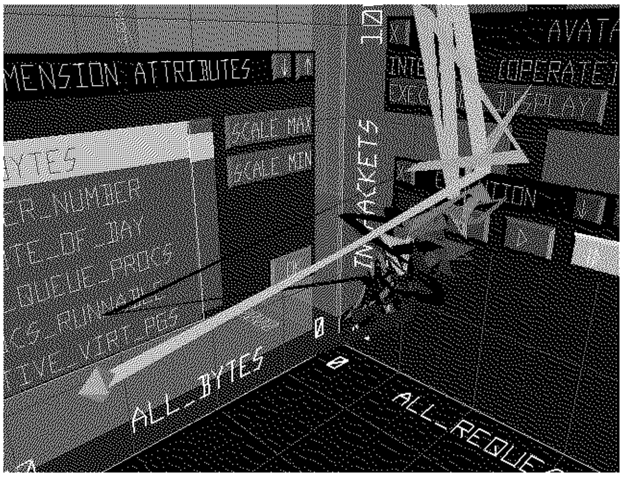


Figure 5: System Metrics (Scattercube View)

dimensional generalization of scatterplot matrices [18], a "time tunnel," and a geographic display [6]) with data sonification and sound spatialization.

5.1 Time Tunnels

The Avatar time tunnel display metaphor captures the time evolutionary behavior of a parallel code via a display consisting of a cylinder whose major axis is time. Along the cylinder periphery, each line is composed of segments, where the color and length of each segment indicate the type and duration of each behavior type in a parallel program. Cross-processor interactions (e.g., via message passing) are represented by chords that cut through the interior of the cylinder.

Figure 4 shows end and side snapshots of the time tunnel for the input/output behavior of the electron scattering code described in §3; in the figure, time increases as one moves forward from the viewer. File open operations are shown near the periphery, with subsequent file read operations shown near the center. The high cost of file opens is manifest as line segments of long duration. Likewise, contention for file access appears as segments that begin at the same time on each processor but complete at differing times.

5.2 Scattercubes

Although the time tunnel metaphor allows one to study processor interactions and application behavioral durations, it cannot show correlations among system performance metrics (e.g., paging or network packet rates). Scattercubes, a generalization of simple scatterplot matrices, allow one to study such metric correlations.

In a two-dimensional scatterplot matrix, all possible pairs of dimensions for a set of N-dimensional data are plotted against each other in separate scatterplots arranged in an N by N matrix. This shows all possible two-dimensional data projections and can be used to determine data dispersion and bivariate correlations. Our three-dimensional generalization of scatterplot matrices, which we call a scattercube, contains

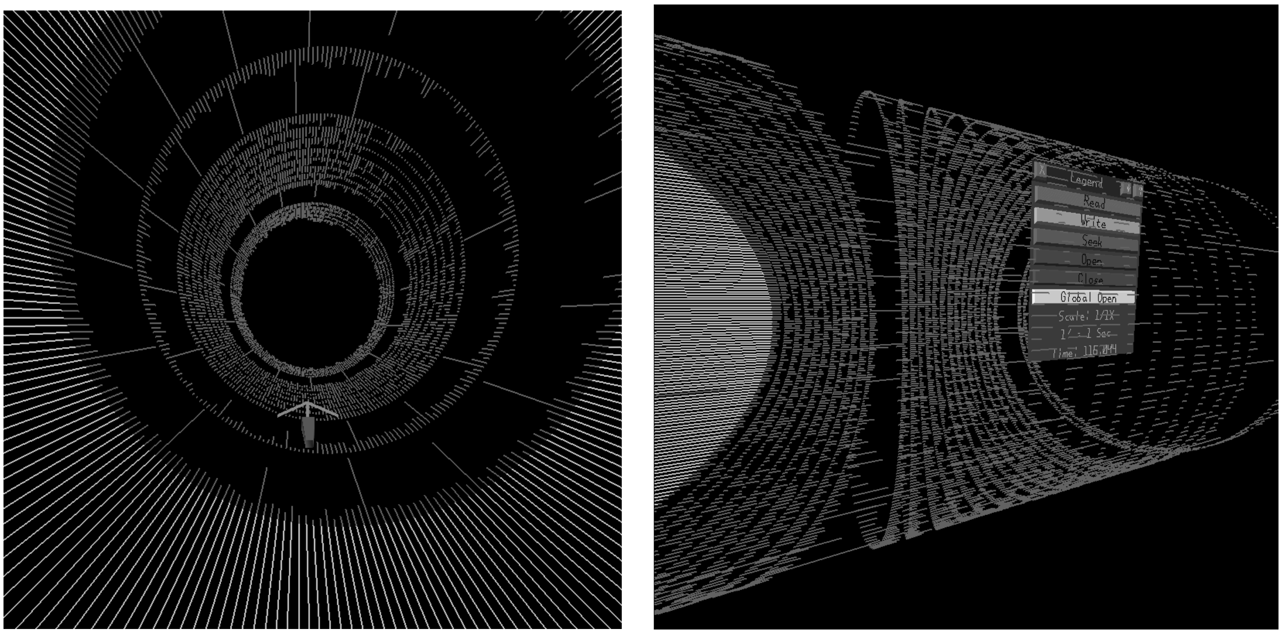


Figure 4: Electron Scattering I/O Dynamics (Time Tunnel Views)

N^3, three-dimensional scatterplots, allowing users to walk around and inside the data.

In each scatterplot cube, the coordinate axes correspond to three of the N performance metrics, and the time-varying position of each processor in the scatterplot cube is determined by the current values of the associated performance metrics. Geometrically, the behaviors of the p processors define a set of p curves in an N-dimensional performance metric space, with each scatterplot cube showing a different three-dimensional projection of this trajectory. To help analyze data point trajectories, it is possible to display these trajectories using history ribbons (i.e., markers of data paths) — Interactively enabling history ribbons for a subset of the data points allows one to see if the selected points cluster in one or more scattercubes.

Figures 5 and 7 show the interiors of scattercubes for two different sets of performance metrics. In both figures, the origin of the metric space is located at the rear of the cube, with the location of each processor denoted by an octahedron. History lines show the movement of the processors in the metric space, with evident clustering.

5.3 Immersion Experiences

We have successfully used Avatar to study the correlations of large numbers of dynamic performance metrics from parallel applications, with emphasis on input/output dynamics, and to display real-time data on the behavior of World Wide Web (WWW) servers [20, 6]. In both cases, the ability to walk and fly through the data, to examine it from multiple perspectives, and to interactively change real-time display attributes has proven invaluable — we have gained insights into behavioral and performance metric interactions that were not possible with two-dimensional graphics displays. Finally, real-time modification of application behavior and resource management policies, described in §6, has proven especially useful.

6 Adaptive Control: The Future

As our analysis of input/output dynamics showed, optimizing application performance requires a judicious match of application request patterns and resource management policies. Based on our instrumentation of large research codes and subsequent discussions with the code developers, we have found that they are often either unaware of their application dynamics or are surprised by the interaction of request patterns with system resource management policies.[2] Moreover, because the interactions between application and system software change across applications and during a single application's execution, we believe runtime libraries and resource management policies are needed that can automatically adapt to rapidly changing application behavior.

We believe the 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. Though completely automated policy config-

[2] This is not surprising. Their interests are in the science embodied in the application. Performance is an issue only when its lack precludes new scientific research.

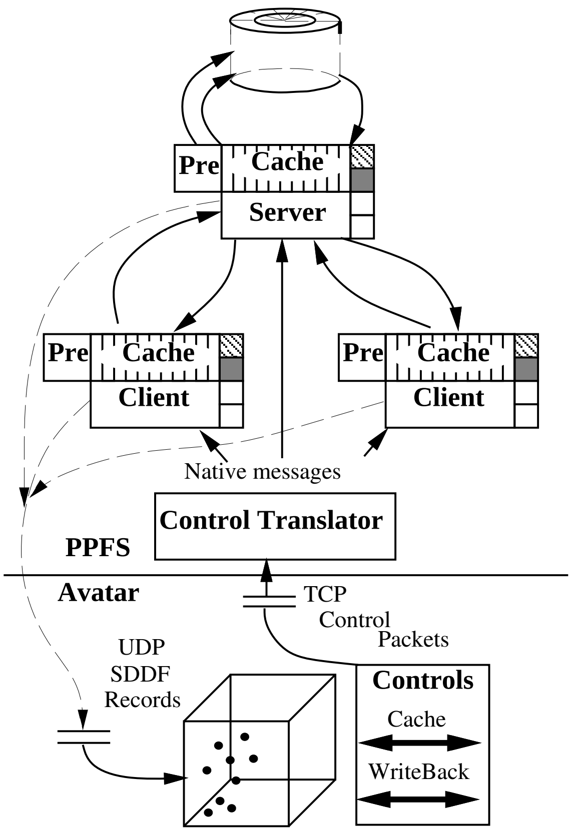


Figure 6: PPFS Interactive Controls

uration is the ultimate goal, intermediate goals must necessarily include interactive optimization.

6.1 Interactive Optimization

To develop better resource management policies, the policy developer must first understand the dynamics of policy and application interaction. This intuition is best gained by varying policies and policy parameters and studying the effects of these changes. By interactively modifying system parameters, one can test large numbers of policy alternatives during a single experiment and study just those performance metrics needed to understand the causes of variations in policy performance. In addition, interactive control and real-time performance data display make it possible for application developers to choose resource management policies and steer their computation's behavior based on their own domain-specific knowledge.

As a test of this approach to interactive performance optimization, we developed a prototype based on an extended version of the PPFS parallel file system, the Pablo performance instrumentation library, and the Avatar virtual environment; see [18] for details. As described in §4, we augmented PPFS with Pablo performance instrumentation to capture data on file system policies (e.g., client and server cache hit rates and request service durations). During application use of PPFS services, this performance data

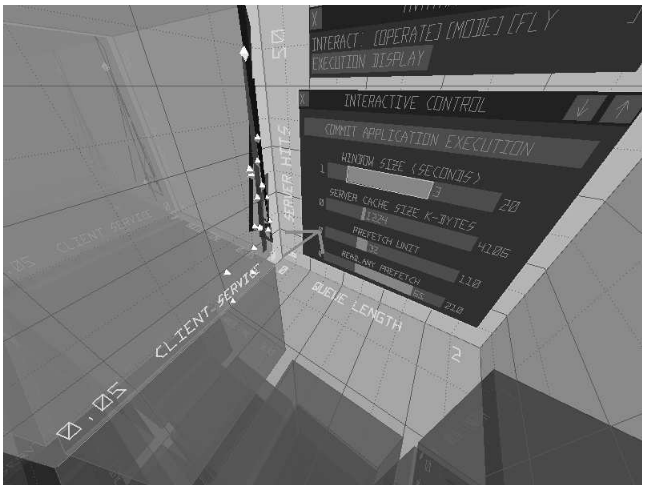


Figure 7: Interactive I/O Policy Selection

is extracted in real-time and displayed in the Avatar virtual environment.

As the schematic of Figure 6 shows, Avatar and PPFS are coupled by an adaptive control mechanism that allows users to interactively change file system policies based on observed performance and to study the effects of these changes. Each PPFS client and server can accept requests to change either file system policies (e.g., cache management) or policy parameters (e.g., cache size).

Figure 7 shows the Avatar interface for file system policy control. By interactively adjusting cache sizes and prefetch amounts, the user can match file system behavior to application resource demands. Because the effects of these changes are immediately visible in the virtual environment, the user can quickly explore a wide variety of file system alternatives.

6.2 Closed Loop Optimization

Automatic, dynamic adaptation of resource management policies to changing application resource demands requires many of the same software components as are needed for interactive optimization, though such a closed loop system replaces the human with a set of algorithmic decision procedures. In particular, closed loop optimization requires

- distributed software *sensors* for real-time performance capture and synthesis of quantitative performance metrics,
- distributed *actuators* that can enable and configure resource management policies,
- automatic *behavioral classification techniques* to identify and group application resource request patterns,
- a set of local (per parallel task) and global (per parallel program) *decision procedures* for selecting resource management policies and enabling actuators based on observed application resource

requests and the system responses captured by performance sensors, and

- *performance analysis tools* to assess the performance of current decision procedures, to conduct parametric performance studies using captured application behavioral data, and to identify improved decision procedures.

Realizing such an adaptive control infrastructure requires solving a plethora of open research problems in control theory and automatic classification. For instance, automatic, qualitative classification of application request patterns is needed to choose policy families. With qualitative knowledge that file read requests are roughly sequential, strided, and of modest size, decision procedures should choose a cache-based prefetch policy. Given a policy selection, performance sensors, placed in the file system read routines, would provide the data needed to choose cache sizes and prefetch distances.

As one step toward automatic policy selection, we have begun exploration of general techniques for qualitative classification of file access patterns [8]. We extended the PPFS software infrastructure to include a trained feed forward artificial neural network that accepts file request attributes (e.g., read or write, file offset, and request size). The result of the classification is a qualitative classification along three axes: sequentiality, read/write mix, and size. The qualitative classification is then used to select PPFS file system policies and policy parameters.

The preliminary results from our automatic classification and policy selection study [8] are extremely encouraging. For a suite of input/output intensive applications with complex, time-varying access patterns, we have achieved substantial speedups over that possible with standard Unix file systems. Because selection of file system policies is automatic, based on application access characteristics and known file system performance, the user need not understand either the detailed behavior of his or her application or the response of the system to the offered load. This lessens the cognitive burden of performance optimization and increases the portability of the application code.

7 Related Work

There is a rich history of control theory and many adaptive real-time systems have been built. The control problems posed by massively parallel systems are both similar and different from those posed by adaptive control of industrial processes. However, the problem of hysteresis is common to both — the feedback loop for performance monitoring and control parameter adjustment must be stable.

Several systems have been built that support application behavior steering (i.e., guiding a computation toward interesting phenomena), though there have been fewer efforts to interactively steer or adaptively control application performance. Leblanc [7] has developed an adaptive real-time system for robotic control that consists of a multiprocessor executing a group of adaptive cognitive tasks. More recently, Schwan [4] has developed an adaptive control mechanism for a molecular dynamics code and an atmospheric modeling code on the KSR-1 that allows users to steer application load distribution and to automatically adjust thread locking policies based on expected synchronization delay. Performance data from sensors can activate actuators inserted in the code by the developer. These actuators can change program behavior based on current conditions and measured performance.

8 Conclusions

Experience has shown 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 parallel system performance is extremely sensitive to the match of application resource demands and resource management policies, resource management policies must adapt to application-specific resource demands. For parallel file systems, we have shown that such adaptation can dramatically increase the achieved application input/output performance.

With flexibility comes the need for mechanisms to specify resource policy choices and configuration parameters. Based on our instrumentation of large research codes and subsequent discussions with the code developers, we have found that they are often either unaware of their application dynamics or are surprised by the interaction of request patterns with system resource management policies. Thus, one needs either an interactive scheme for exploration of resource management policy alternatives and their performance or an automatic scheme that can identify application resource demands and select appropriate resource management policies.

We believe the ultimate 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. Such a system would be complemented by an interactive virtual environment for real-time monitoring and performance tuning.

Acknowledgments

Phyl Crandall contributed to the early evaluation of input/output behavior, and Andrew Chien has been an insightful collaborator. Ilkyr Hamzaoglu, Jay Huber, and David Blumenthal helped develop and extend the portable parallel file system, showing that major input/output performance improvements were possible with file system tuning. Finally, Steve Lamm, Luis Tavera, and Keith Shields contributed greatly to the development of Avatar, our virtual environment for performance data immersion and control.

References

[1] Adve, V. S., Mellor-Crummey, J., Anderson, M., Kennedy, K., Wang, J., and Reed, D. A. An Integrated Compilation and Performance Analysis Environment for Data Parallel

Programs. In *Proceedings of Supercomputing '95* (Dec. 1995).

[2] 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).

[3] Cruz-Neira, C., D.J.Sandin, and DeFanti, T. Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. In *SIGGRAPH '93 Proceedings* (Aug. 1993), Association for Computing Machinery.

[4] 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).

[5] 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), pp. 385–394.

[6] Lamm, S. E., Scullin, W. H., and Reed, D. A. Real-time Geographic Visualization of World Wide Web Traffic. In *submitted for publication* (1996).

[7] LeBlanc, T. J., and Markatos, E. P. Operating System Support for Adaptive Real-time Systems. In *Proceedings of the Seventh IEEE Workshop on Real-Time Operating Systems and Software* (May 1990), pp. 1–10.

[8] Madhyastha, T., and Reed, D. A. Intelligent, Adaptive File System Policy Selection. In *submitted for publication* (1995).

[9] Malony, A. D., and Reed, D. A. Visualizing Parallel Computer System Performance. In *Instrumentation for Future Parallel Computing System s*, M. Simmons, R. Koskela, and I. Bucher, Eds. Addison-Wesley Publishing Company, 1989, pp. 59–90.

[10] Malony, A. D., and Reed, D. A. A Hardware-Based Performance Monitor for the Intel iPSC/2 Hypercube. In *1990 ACM International Conference on Supercomputing* (June 1990), Association for Computing Machinery, pp. 213–216.

[11] Malony, A. D., Reed, D. A., Arendt, J. W., Aydt, R. A., Grabas, D., and Totty, B. K. An Integrated Performance Data Collection Analysis, and Visualization System. In *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications* (Monterey, CA, Mar. 1989), Association for Computing Machinery, pp. 229–236.

[12] Malony, A. D., Reed, D. A., and Rudolph, D. C. Integrating Performance Data Collection, Analysis, and Visualization. In *Parallel Computer Systems: Performance Instrumentation and Visualization*, M. Simmons, R. Koskela, and I. Bucher, Eds. Addison-Wesley Publishing Company, 1990, pp. 73–97.

[13] Poole, J. T. Scalable I/O Initiative. California Institute of Technology, Available at `http://www.ccsf.caltech.edu/SIO/`, 1995.

[14] Reed, D. A. Performance Instrumentation Techniques for Parallel Systems. In *Models and Techniques for Performance Evaluation of Computer and Communications Systems*, L. Donatiello and R. Nelson, Eds. Springer-Verlag Lecture Notes in Computer Science, 1993, pp. 463–490.

[15] Reed, D. A. Experimental Performance Analysis of Parallel Systems: Techniques and Open Problems. In *Proceedings of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation* (May 1994), pp. 25–51.

[16] Reed, D. A., Aydt, R. A., Noe, R. J., Roth, P. C., Shields, K. A., Schwartz, B. W., and Tavera, L. F. Scalable Performance Analysis: The Pablo Performance Analysis Environment. In *Proceedings of the Scalable Parallel Libraries Conference*, A. Skjellum, Ed. IEEE Computer Society, 1993, pp. 104–113.

[17] Reed, D. A., and Rudolph, D. C. Experiences with Hypercube Operating System Instrumentation. *International Journal of High-Speed Computing* (Dec. 1989), 517–542.

[18] Reed, D. A., Shields, K. A., Tavera, L. F., Scullin, W. H., and Elford, C. L. Virtual Reality and Parallel Systems Performance Analysis. *IEEE Computer* (Nov. 1995), 57–67.

[19] Ries, B., Anderson, R., Auld, W., Breazeal, D., Callaghan, K., Richards, E., and Smith, W. The Paragon Performance Monitoring Environment. In *Proceedings of Supercomputing '93* (Nov. 1993), pp. 850–859.

[20] Scullin, W. H., Kwan, T. T., and Reed, D. A. Real-time Visualization of NCSA's World Wide Web Data. In *Symposium on Visualizing Time-Varying Data* (Sept. 1995).

[21] Winstead, C., and McKoy, V. Studies of Electron-Molecule Collisions on Massively Parallel Computers. In *Modern Electronic Structure Theory*, D. R. Yarkony, Ed., vol. 2. World Scientific, 1994.