PDRS: A Performance Data Representation System*

Xian-He Sun [1,2] Xingfu Wu [3,1]

[1] Dept. of Computer Science, Louisiana State University, Baton Rouge, LA 70803

[2] Dept. of Computer Science, Illinois Institute of Technology, Chicago, IL 60616

[3] Dept. of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208

sun@cs.iit.edu wuxf@ece.nwu.edu

Abstract. We present the design and development of a Performance Data Representation System (PDRS) for scalable parallel computing. PDRS provides decision support that helps users find the right data to understand their programs' performance and to select appropriate ways to display and analyze it. PDRS is an attempt to provide appropriate assistant to help programmers identifying performance bottlenecks and optimizing their programs.

1 Introduction

Many performance measurement systems have been developed in recent years. While these systems are important, their practical usefulness relies on an appropriate understanding of the measured data. When monitoring a complex parallel program, the amount of performance data collected may be very huge. This huge amount of performance data needs to be processed for further performance evaluation and analysis. A general performance measurement system always provides a facility that assists manipulation of this performance data. Data manipulation functions are often dependent on performance data organization and representation. The difficulty in providing an adequate performance environment for high performance computing is the lack of appropriate models, representations and associated evaluation methods to understand measured data and locate performance bottlenecks. Performance Data Representation System (PDRS) proposed in this paper is designed to attack this difficulty. PDRS is a general-purpose integrated system supported by performance database representation and the combination of performance visualization and auralization. It is based on our recent success in automatic performance evaluation and prediction.

Many performance measurement systems exist right now [3, 4, 5]. While these performance systems have made their contribution to the state-of-the-art of performance

* This work was supported in part by National Science Foundation under NSF grant ASC-9720215 and CCR-9972251.

evaluation, none of them has addressed the data presentation and understanding issue adequately. With the advance in performance measurement and visualization techniques, and increased use of large, scalable computing systems, data presentation and management becomes increasingly important. The PDRS is a post-execution performance data representation system designed for scalable computing, and is distinct from existing performance systems. First, while it supports conventional visualization views, it is designed based on the most recent analytical results in scalability and statistical analysis to reveal the scaling properties of a scalable computing system. Second, the system uses relational database, SQL and Java JDBC techniques such that performance information is easily retrieved, compared and displayed. Because of the complexity and volume of the data involved in a performance database, it is natural to exploit a database management system (DBMS) to archive and retrieve performance data. A DBMS will help not only in managing the performance data, but also in assuring that the various performance information can be presented in some reasonable format for users. Third, the system is implemented based on the combination of performance visualization and auralization techniques and object-oriented Java techniques such that it is easy for users to understand and use. Finally, the system supports the SDDF data format. It can be either used as a stand-alone application or easily integrated into other existing performance environments.

2 Design and Implementation of PDRS

Figure 2.1 depicts the design framework of PDRS. The technical approaches used to develop these components are discussed below section by section.

2.1 Trace Data Module

This module is in charge of collecting original performance data of parallel programs, and stores them with SDDF [1].

The large volume of data involved in parallel computations requires that instrumentation to collect the data selectively and intelligently. One way to collect data of a parallel program is to instrument the program executable so that when the program runs, it generates the desired information. PDRS is designed to use the Scala Instrumentation System (SIS) [11] to get the SDDF trace data file. PDRS also provides a general interface that can be used under any system, which provides the SDDF trace data interface.

2.2 Data Management Module

This module is in charge of performance data filtering and mapping.

Event histories of parallel programs are valuable information sources for performance analysis but the problem is how to extract the useful information from massive amounts of low-level event traces. Our system performs the data filtering as a preparation to store the event history into a relational database. The SDDF is a trace description language that specifies both data record structures and data record instances. We are building a performance database based on the SDDF specification. Our data management module is being implemented in Oracle DBMS.

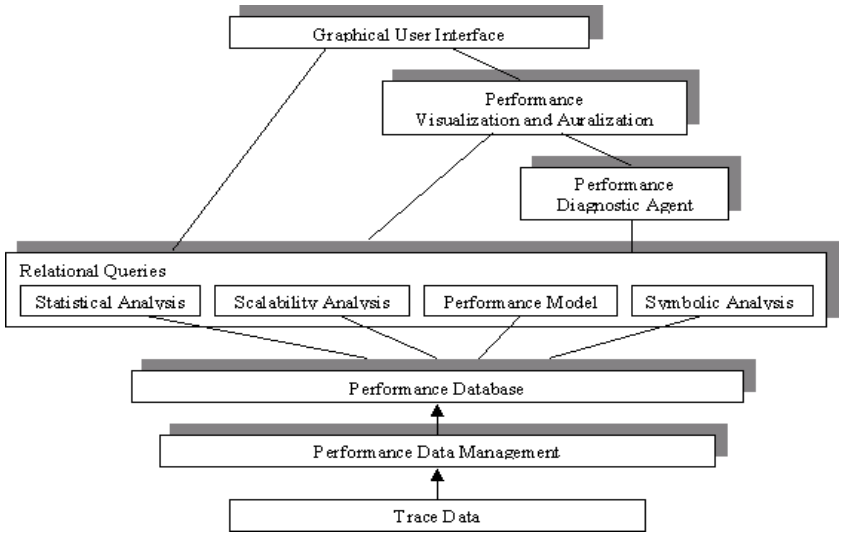

Figure 2.1 Design framework of PDRS

2.3 Performance Database

We classify the performance data saved in the SDDF tracefiles into five groups: processor information, memory information, program information, communication information and I/O information. Each group is represented as an entity relation in the performance database. An individual event in a relation is treated as a tuple with a given unique identifier.

The information retrieval is achieved by the relational database queries. The example below shows how objects can be retrieved using JDBC [13]. For instance, suppose that we want to get the communication events that occurred in processor 0, the query

select *sourcePE, destinationPE, messageLength, event_startTimestamp, event_endTimestamp* ***from*** *Communication Information* ***where*** *processor = 0.*

We may make the following SQL query by JDBC:

```
ResultSet rs = stmt.executeQuery( "select sourcePE, destinationPE,
messageLength, event_startTimestamp, event_endTimestamp
        from Communication Information where processor = 0");
while (rs.next()) {
        Object i1 = rs.getObject("sourcePE");
        Object i2 = rs.getObject("destinationPE");
        Object r1 = rs.getObject("messageLength");
        Object r2 = rs.getObject("event_startTimestamp");
        Object r3 = rs.getObject("event_endTimestamp");
}
```

Multiple versions of performance data are handled by specifying a version attribute in each tuple. By specifying a version number in each database query, we can get multiple versions of program performance for comparison. In addition to the default PDRS performance parameters, new performance parameters such as sound files can also be added by users and be supported by the database.

2.4 Relational Queries Module

This module includes four parts: Symbolic Analysis, Statistical Analysis, Scalability Analysis, and Performance Model Generator. The module is being implemented in JDBC. Its structure is shown in Figure 2.2. Java applications include the PDA, PVA, and GUI module implemented by Java. The JDBC provides a bridge between Java applications and performance database.

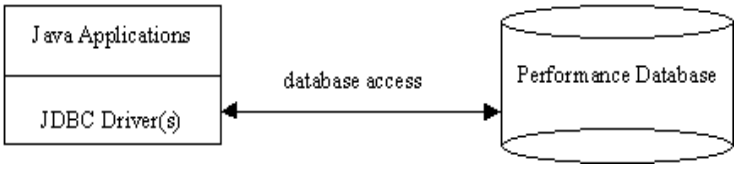

Figure 2.2 Relational Queries Module

We use symbolic evaluation [2, 6] that combines both data and control flow analysis to determine variable values, assumptions about and constraints between variable values, and conditions under which control flow reaches a program statement. Computations are represented as symbolic expressions defined over the program's problem and machine size. Each program variable is associated with a symbolic expression describing its value at a specific program point

Statistical Analysis determines code and/or machine effects, finds the correlation between program phases, identifies the scaling behavior of "difficult-segments", and provides statistical performance data [12] for the PDA (Performance Diagnostic Agent)

module and GUI. The development of the scalability analysis is based on newly developed algorithms for predicting performance in terms of execution time and scalability of a code-machine combination [8, 9, 11, 15]. Analytical and experimental results show that scalability combined with initial execution time can provide good performance prediction, in terms of execution times. In addition, crossing-point analysis [9] finds fast/slow performance crossing points of parallel programs and machines. In contrast with execution time, which is measured for a particular pair of problem and system size, range comparison compares performance over a wide range of ensemble and problem size via scalability and crossing-point analysis.

In addition to high-level performance prediction, PDRS also supports low-level performance analysis to identify performance bottlenecks and hardware constrains based on performance models chosen by the user. For example, we have proposed an empirical memory model based on a simplified mean value parameterization [14] to separate CPU execution time from stall time due to memory loads/stores. From traced information or information from the analysis modules, performance models can be generated to predict the performance at the component level, as well as over-all performance.

2.5 Performance Diagnostic Agent (PDA) Module

This module provides performance advice in order to help users find performance bottlenecks in their application programs. It also provides performance comparison and suggestions based on real performance results and predicted performance ranges. The PDA is based on our approaches to statistical analysis, scalability analysis and performance model generator. The function operation algorithm for this module is as follows.

```
Algorithm (Performance diagnosis):
        Performance analysis requests;
        switch (analysis type) {
                Statistical:
                        Retrieve the performance information required;
                        Get or compute the predicted performance range;
                        Compute the real result of requested performance parameter;
                        Compare the result with the performance range;
                        If (the result is not in the performance range)
                                Give an explanation (using graphics and sound);
                        break;
                Scalability:
                        Retrieve the performance information required;
                        Get or compute the predicted scalability results;
                        Compute the real scalability results;
                        Compare the real result with the predicted results;
                        Explain the compared results (using graphics and sound);
```

```
                break;
        Models:
                Retrieve the performance information required;
                Get the predicted performance range;
                Compute the real result of requested performance parameter;
                Compare the result with the performance range;
                If (the result is not in the performance range)
                        Give an explanation (using graphics and sound);
                break;
        Default: printf("No such analysis type");
                break;
}
```

In the algorithm, the PDA can provide suggestions and explanations when performance bottlenecks occur. Based on the statistical analysis, the PDA can retrieve the performance information from the performance database, then may provide the advice about program performance.

2.6 Performance Visualization and Auralization (PVA) Module and Graphical User Interface Module

This PVA module provides some graphical display of performance information about users' application programs and platforms. It is natural to use different visual objects to represent various performance data and use visualization techniques to gain insight into the execution of parallel programs so that their performance may be understood and improved. The basic goal of this module is to use graphics and sound (Java 2D, Java 3D and JavaSound) to display some advice and performance views about application programs. For example, based on performance comparison, some performance bottlenecks can be found in graphics. Some suggestions can be given in graphics, such as what applications are suitable for the platforms, what platforms are suitable for solving the applications, and how to modify the application program to be suitable for the platforms. When performance bottlenecks occur, sound is used to inform users about some performance problem in their application programs. The sound files are stored in a performance database.

The Graphical User Interface module is an integrated user-friendly graphical interface. It integrates the whole functions of the PVA module, and directly displays the performance data requested by users.

Figures 2.3 and 2.4 are two views of PDRS GUI. Figure 2.3 shows speed comparison of PDD and PT algorithms [7]. Figure 2.4 shows the Kiviat graph for performance comparison.

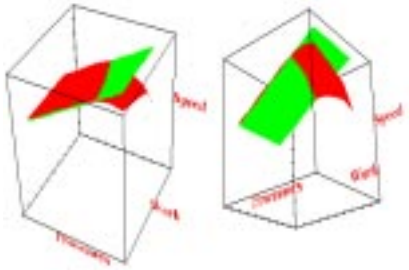

Figure 2.3 Speed comparison of PDD and PT algorithms

Figure 2.4 Kiviat Graph for Performance Comparison

3 Summary

We have presented the design of a Performance Data Representation System (PDRS) based on our current success of the development of the SCALA [10, 11] performance system for scalable parallel processing. While the PDRS has not been fully implemented at this time, some of its key components have been implemented and tested. Implementation results are very encouraging. PDRS highlights the performance data representation using relational database. Integrated into advanced restructuring compilation and performance analysis system, the proposed PDRS attempts to lift performance evaluation system to a new level. It is designed to provide developers a guideline on performance optimization, to assist the purchasers selecting systems best suited to their needs, and to give valuable feedback to vendors on bottlenecks that can be alleviated in future products. It has the potential to provide users with much more useful information than current existing performance systems. Many advanced technologies, such as database management, object-oriented programming, visualization and auralization are used in the PDRS. The integration of these technologies into compilation and performance analysis system is new, and very challenging. It can motivate many new

research and development issues. PDRS is only a first step toward the automatic performance analysis and optimization.

References

1. R. Aydt, *The Pablo Self-Defining Data Format*, Department of Computer Science, University of Illinois, April 1995, ftp://bugle.cs.uiuc.edu/pub/Release/Documentation/SDDF.ps.
2. T. Fahringer and B. Scholz, Symbolic evaluation for parallelizing compilers, in *Proc. of the 11th ACM International Conference on Supercomputing*, Vienna, Austria, ACM Press, July 1997, 261-268.
3. J. Kohn and W. Williams, ATExpert, *Journal of Parallel and Distributed Computing* 18, 1993, 205-222.
4. A.D. Malony and G.V. Wilson, Future directions in parallel performance environment, *Performance Measurement and Visualization of Parallel Systems*, Eds: G. Haring and G. Kotsis, Elsevier Science Publishers B.V., 1993, 331-351.
5. B. P. Miller, M.D. Callaghan, J.M. Cargille, J.K. Hollingsworth, R.B. Irvin, K.L. Karavanic, K. Kunchithapadam, and T. Newhall, The Paradyn parallel performance measurement tools, *IEEE Computer* 28, 11, 1995.
6. M. Scheibl, A. Celic, and T. Fahringer, *Interfacing Mathematica from the Vienna Fortran Compilation System*, Technical Report, Institute for Software Technology and Parallel Systems, Univ. of Vienna, December 1996.
7. X.-H. Sun, H. Zhang, and L. Ni, Efficient tridiagonal solvers on multicomputers, *IEEE Transactions on Computers* 41, 3 (1992), 286-296.
8. X.-H. Sun and D. Rover, Scalability of parallel algorithm-machine combinations, *IEEE Transactions on Parallel and Distributed Systems*, June 1994, 599-613.
9. X.-H. Sun, Performance range comparison via crossing point analysis, *Lecture Notes in Computer Science* 1388 (J. Rolim, ed.), Springer, March 1998.
10. X.-H. Sun, T. Fahringer, M. Pantano, and Z. Zhan, SCALA: A performance system for scalable computing, in *Proc. of the Workshop on High-Level Parallel Programming Models & Supportive Environments*, *Lecture Notes in Computer Science* 1586, Springer, April 1999.
11. X.-H. Sun, M. Pantano, and Thomas Fahringer, Integrated range comparison for data-parallel compilation systems, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, May, 1999, 448-458.
12. X.-H. Sun, D. He, K. Cameron, and Y. Luo, A Factorial Performance Evaluation for Hierarchical Memory Systems, in *Proc. of the IEEE Int'l Parallel Processing Symposium'99*, April 1999.
13. Sun Microsystems Inc., *JDBC: A Java SQL API*, Version 1.20, http://www.javasoft.com/products/jdbc/index.html, January 1997.
14. M. V. Vernon, E. D. Lazowska, and J. Zahorjan, An accurate and efficient performance analysis technique for multi-processor snooping cache-consistency protocols, in *Proc. 15th Annual Symp. Computer Architecture*, Honolulu, HI, June 1988, 308-315.
15. Xingfu Wu, *Performance Evaluation, Prediction, and Visualization of Parallel Systems*, Kluwer Academic Publishers, Boston, ISBN 0-7923-8462-8, 1999.