Dynamic Instrumentation and Performance Prediction of Application Execution

A.M. Alkindi , D.J. Kerbyson, and G.R. Nudd

High Performance Systems Laboratory, Department of Computer Science
University of Warwick, Coventry CV4 7AL, UK
{ahmed,djke,grn}@dcs.warwick.ac.uk

Abstract. This paper presents a new technique that enhances the process and the methodology used in a performance prediction analysis. An automatic dynamic instrumentation methodology is added to Warwick's Performance Analysis and Characterization Environment PACE [1]. The automation process has eliminated the need to manually obtain application information and data. The Dynamic instrumentation has given PACE the ability to extract and utilize data that were hidden and unobtainable prior to execution. We give two examples to illustrate our methodology. While it was impossible to perform the analysis using the original method due to lack of essential information, the new technique successfully enabled PACE to conduct the prediction analysis in a dynamic environment. The results show that with the automated dynamic instrumentation, the performance prediction analysis of dynamic application execution is possible and the results obtained are reliable. We believe that the technique implemented here could eventually be used in other performance prediction tool-sets, and therefore enhance the ways in which the performance of systems and applications is analysed and predicted.

Keywords: Dynamic instrumentation, performance optimization, performance analysis, modelling, PACE.

1 Introduction

In this age of Information Power GRID [2], a significant amount of work is being undertaken to ease the process of performance prediction. A number of valuable toolsets for performance optimisation through prediction exist. Falcon [3], Paradyn/Dynamist [4], Pablo [5] are being used to assist in performance prediction, tuning and identification of bottlenecks in applications. These tool-sets typically include various features to allow predicting with great accuracy the performance of applications. Some also allow instrumentation at various levels, data collection, and interrogation through visualisation modules. Warwick's Performance Analysis and Characterization Environment PACE is concerned with the prediction analysis and is central to our work in this paper.

In this work, we introduce a new methodology that enhances the performance prediction process in PACE. The new technique utilizes dynamic instrumentation by inserting special sensors in an application source code prior to its first execution. This automated insertion benefits the prediction process in two ways. First, the automation

B. Hertzberger et al. (Eds.) : HPCN Europe 2001, LNCS 2110, pp. 513-523, 2001.

would make it easier for the user to conduct as many studies as he wishes and without the need to manually obtain application information and data. Second, it would allow for the gathering of valuable run-time information from an application execution. This run-time (on-the-fly) information is then used to give PACE the accurate parameters it needs for application performance analysis and prediction. The results of this work show that the process of application performance prediction is greatly enhanced, and the cost of this process is reduced.

An important aspect of PACE is its ability to conduct performance prediction analysis of a given application, and to produce valuable information for the user [6]. However, one analysis aspect this tool-set fails to address is the complete run-time analysis. That is, no information is gathered or analysed while the system is executing. This prevents the user from getting real-time information (such as data, loop sizes, or conditional statements/computations) regarding the system. Up to now, this limitation has been slightly overcome through manual procedures in which specific data are manually inserted into the analysis process. However, these manual procedures have been limited to loops and conditional statements and not data. Furthermore, when the application execution is dynamic and is data-dependent, then the user of PACE cannot pass the correct information to it for a reliable analysis.

Beside our work, several studies have been conducted in order to address this particular problem. Some are introducing the use of historical performance data [7] also known as profiling, while others are implementing an instrumentation procedure through a language for dynamic program instrumentation [8]. However, current tools have yet to produce an accurate dynamic, on-the-fly performance prediction of a given application. For example, while Dynamist is a powerful performance analysis tool, that can give a variety of useful information about the program being executed, it has a number of problems. It produces huge overhead for the user because the analysis is done on the executable, therefore large system resources are required in order to conduct a stable analysis. Furthermore, the analysis must be conducted on the same platform that is being analysed. Another tool, Falcon, supports instrumentation, tuning and steering. It also applies the sensors technique on applications. However, it does not provide modelling and it relies on the availability of shared memory between application and local monitoring threads. None of these systems, however, provide or try to acquire information with regards to loop/conditional statements or data dependencies in the given application in order to enhance the accuracy of the prediction analysis. Our work, as indicated earlier, introduces a dynamic instrumentation technique, which we believe will tackle the problem stated above and improve on solutions provided by available performance prediction packages. We believe that the methodology presented here may pave the way for the development of future generations of performance prediction tools, which currently lack the ability to analyse or predict an application performance on a given platform dynamically and on-the-fly.

Many publications of previous work on the PACE tool exists, which we do not have space to reference properly. However, we will give a brief introduction to this performance prediction tool-set in the next section. A good tutorial on PACE can be found in our web site http://www.dcs.warwick.ac.uk/~hpsg/pace_top.htm.

2 PACE: Performance Analysis and Characterisation Environment

PACE is a modelling toolset for high performance and distributed applications. It includes tools for model definition, model creation, evaluation, and performance analysis. It uses associative objects organised in a layered framework as a basis for representing each of a system's components. An overview of the model organisation and creation is presented in the following subsections.

2.1 Model Components

Many existing techniques, particularly for the analysis of serial machines, use Software Performance Engineering (SPE) methodologies [9], to provide a representation of the whole system in terms of two modular components, namely a software execution model and a system model. However, for high performance computing systems, the organisation of models must be extended to take concurrency into account. The layered framework is an extension of SPE for the characterisation of parallel and distributed systems. It supports the development of several types of models: software, parallelisation (mapping), and system (hardware).
The functions of the layers are:

Application Layer – describes an application in terms of a sequence of subtasks. It acts as the entry point to the performance study, and includes an interface that can be used to modify parameters of a performance scenario (a user might perform an analysis many times with different parameters).

Application Subtask Layer – describes the codes within an application that can be executed in parallel.

Parallel Template Layer – describes the parallel characteristics of subtasks in terms of expected computation-communication interactions between processors.

Hardware Layer – collects system specification parameters, micro-benchmark results, statistical models, analytical models, and heuristics that characterise the communication and computation abilities of a particular system.

In the layered framework, a performance model is built up from a number of separate objects. Each object is of one of the following types: application, subtask, parallel template, and hardware. A key feature of the object organization is the independent representation of computation, parallelisation, and hardware. Each software object (application, subtask, or parallel template) is composed of an internal structure, options, and an interface that can be used by other objects to modify its behaviour. The main aim of these objects is to describe the system resources required by the application, which are modelled in the hardware object. Each hardware object is subdivided into many smaller component hardware models, each describing the behaviour of individual parts of the hardware system. For example, the memory, the CPU, and the communication system are considered as separate component models [10].

2.2 Model Creation

PACE users can employ a workload definition language CHIP³S to describe the characteristics of the application. CHIP³S is an application modelling language that supports multiple levels of workload abstractions [11]. When application source code is available, the Application Characterization Tool (ACT) can semi-automatically create CHIP³S workload descriptions. ACT performs a static analysis of the code to produce the control flow of the application, operation counts in terms of SUIF language operations, and the communication structure. This process is illustrated in Fig.1. SUIF, Stanford University Intermediate Format [12], is an intermediate presentation of the compiled code that combines the advantages of both high level and assembly language. ACT cannot determine dynamic related performance aspects of the application such as data dependent parameters. These parameters can be obtained either by profiling or with user support.

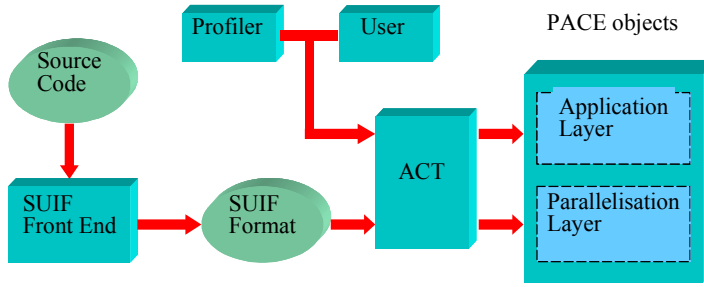


Fig. 1. Model Creation Process with ACT.

The CHIP³S objects adhere to the layered framework. A compiler translates CHIP³S scripts to C code that is linked with an evaluation engine and the hardware models. The final output is a binary file, which can be executed rapidly. The user determines the system/application configuration and the type of output that is required as command line arguments. The model binary performs all the necessary model evaluations and produces the requested results. PACE includes an option to generate predicted traces that can then further be analysed by visualization tools (e.g. PABLO) [13].

3 Dynamic Instrumentation of Application Execution

3.1 Overview

In this work, we introduce a new approach to gather maximum run-time information for the performance prediction analysis process. Our goal is to acquire application data dynamically, while at the same time not to lose the ability to analyse these data due to their incorrectness, or resources overhead. The run-time (on-the-fly) procedure would give the user an acceptable view of what is really happening inside the

application. This makes analysing and predicting the performance of this application more accurate, and therefore gives us a greater chance for better optimisation. Our aim has also been to enhance performance prediction methodology without losing the tool's original functionality. We believe that any on-the-fly performance prediction tool-set should be easy to use, which would make it a system for any user, and not just for performance experts.

3.2 An Automated Profiling Technique for PACE

The long and sometimes inefficient procedures mentioned above and described in [6] and [13] are summarized in the following steps:

- Execute an ACT command on the required application.
- Manually, collect the code produced from the output of the previous run and insert it into subtask object and then execute the modified object.
- Execute and link the remaining subtask objects to produce the desired model.

These long and painstaking steps must be executed separately and manually. Fig.2 shows the original manual process of creating an executable model in PACE. For an application to be analysed, and its performance to be precisely predicted, the user must interfere and insert the output from ACT to produce a reliable subtask, which is then linked with other subtasks to produce the executable model.

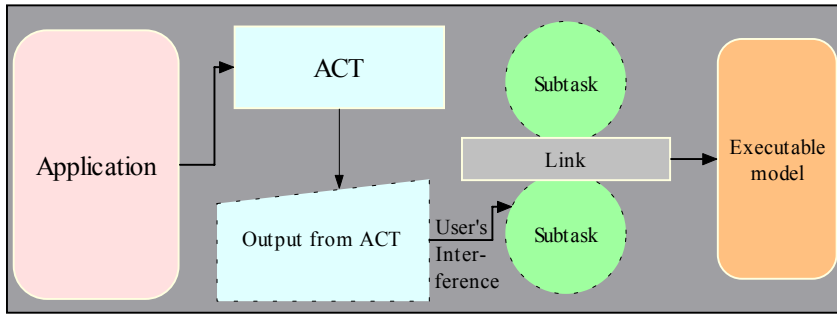
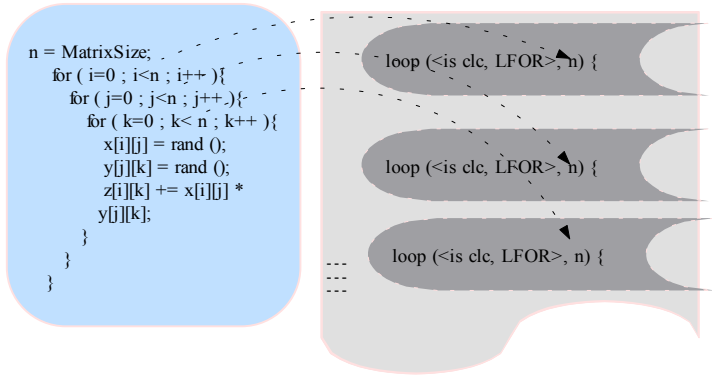


Fig. 2. Manual Implementation of PACE.

Table 1 shows loops representation in an ACT format from a matrix multiplication program. When the analysis requires hundreds or even thousands of runs, this interference is going to be an inefficient way to create an executable model.

Our automation solution creates a filtering routine that takes as input the output of ACT and filters out the necessary code (see fig.3). The routine inserts this code inside the subtask object without user interference. This automation enhances the process described earlier. A simple batch program could now run ACT, execute and link all the required subtask objects, and finally produce the executable model. This automation process reduces the cost for conducting multiple analyses on multiple or distributed systems.

Table 1. Loops Representation in an ACT Format from a Matrix Multiplication Program.



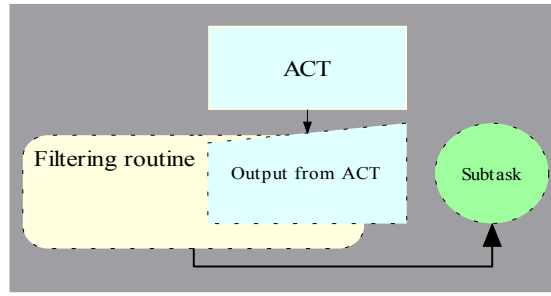


Fig. 3. Automatic Implementation of PACE.

3.3 Source Code Instrumentation and Dynamic Analysis

For PACE to be able to function in a dynamic environment with a reliable outcome, we introduce the source code instrumentation and dynamic analysis methodology. The original method requires a user to input application information (which might consist of parameters like loop sizes and conditional statements probabilities) to PACE when an analysis is performed. While the user might correctly guess the information in small applications with few loops or conditional statements using small sized data, it is going to be harder when big applications with huge loops or confusing conditional statements are the target of analysis. Trying to guess, or manually figuring out, the sizes of application loops and the probabilities of its conditional statements are not helpful in getting an accurate application performance prediction.

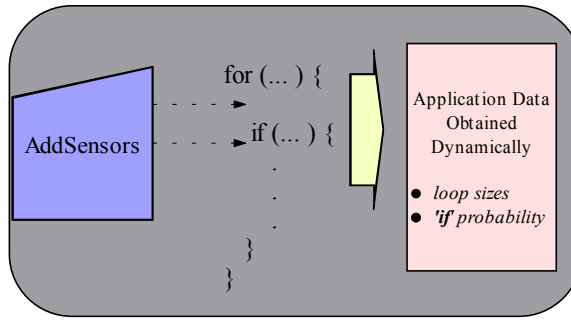


Fig. 4. Dynamic Instrumentation of an Application.

Our solution to this problem is to use an analyser, AddSensors, that inserts special sensors into the required application. The sensors, as shown in fig.4, reside inside fundamental statements (loops and conditional statements in the case of our examples) and are triggered when the application is first executed. These sensors gather run-time information and data for the user, and pass them to the subtask using the new automated procedure.

We illustrate this methodology on two examples, a matrix multiplication program, and a program to find the smallest value in an array. Tables 2 and 3 below show the result of executing the AddSensors routine on both programs. The advantage of this method is that it enables the user to have PACE gather information automatically and dynamically. With this information conducting a performance prediction analysis using PACE is now possible and the results are reliable.

Table 2. Adding Sensors to a Matrix Multiplication Program.

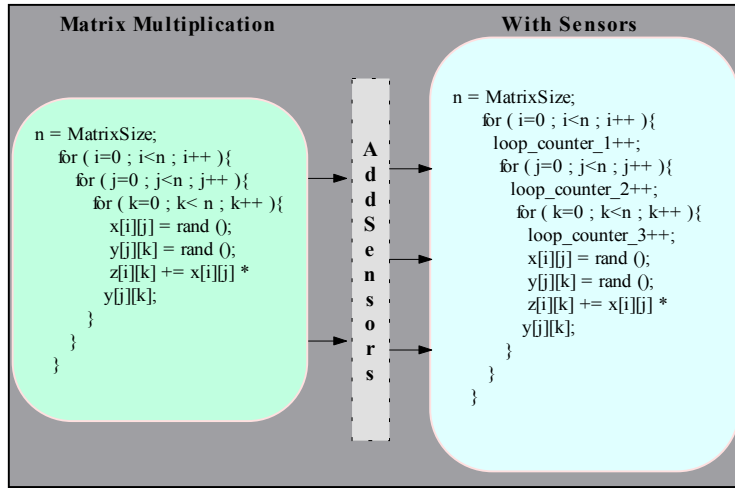
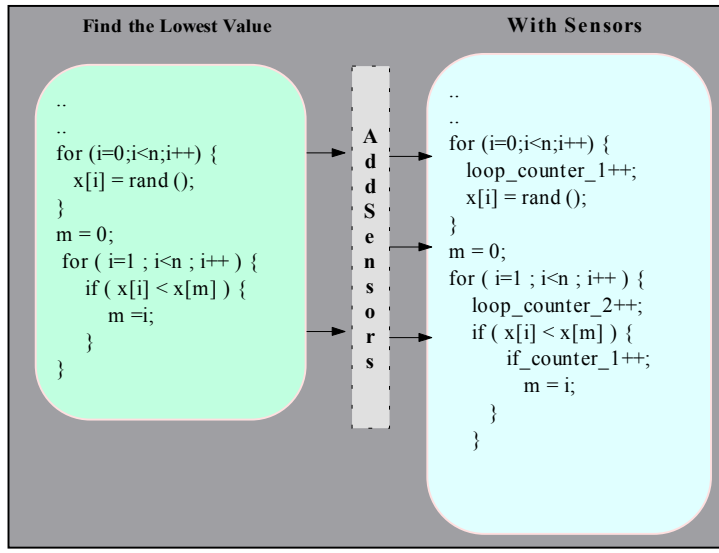


Table 3. Adding Sensors to a Program that Finds the Lowest Value in an Array.



Without dynamically obtained data, a user is not able to conduct performing prediction analysis by a tool-set like PACE. This is especially true when the analysis is performed in a dynamic environment where application execution is data-dependent. To illustrate this difficulty, we ran the prediction analysis for both example programs on a Sun Ultra10 machine using the original method of PACE. In both examples, there are data and information that PACE would require before utilizing its prediction capabilities. For example, the user should pass the loop size as well as the conditional statement probability in the matrix multiplication routine. However, if the size of the matrix is unknown prior to execution, then the user can no longer proceed with the study. For the purpose of comparison, we implemented the original methodology of PACE with manually obtained application information. In the best optimistic scenario, we assume that the user can guess the correct input information for PACE with ±10% accuracy. Fig.5 and 6 show the results of using this manual method of PACE prediction analysis in comparison to real-time execution. The grey area highlights the uncertain prediction results obtained as a result of uncertain input data from the user. That is, for a matrix of size 130×130 in fig.5, the prediction is anywhere between 0.5 seconds to one second. If the information that was fed to PACE had errors greater than the ±10%, then the grey area would have been larger and the outcome of the prediction would have been even more inaccurate. To overcome this problem, we implement the new automated instrumentation technique introduced in this paper. Fig.7 and 8 show the results of using the dynamically instrumented data for PACE prediction analysis in comparison to real-time execution. The matrix size in the first example varies between 10x10 to 200x200. We implement a random approach of selecting which matrix size to be used first when the program is executed. The reason behind this is so that the user gets unpredictable information when manually analyzing the source code, which makes him/her rely solely on the

dynamic instrumentation technique. The same rule is applied for the array sizes in the second example.

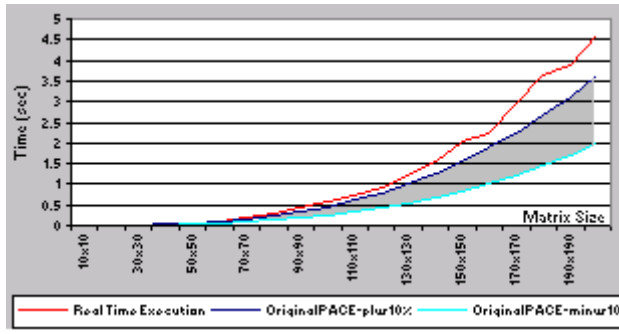


Fig. 5. Matrix Multiplication Performance Prediction with Uncertain Prediction Times.

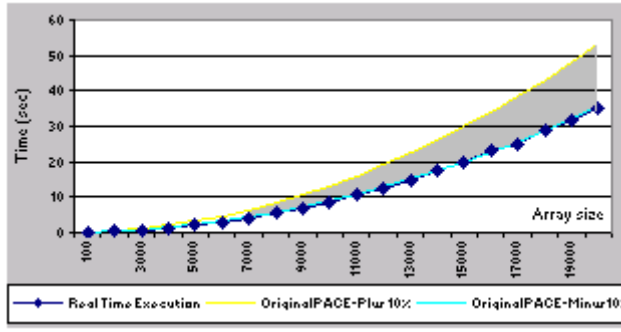


Fig. 6. Uncertain Prediction Times for a Program to find the Lowest Value in an Array.

4 Analysis and Conclusion

In this work, we show that the new automation and instrumentation technique for conducting performance prediction research using PACE is cheaper, faster, and more accurate than the original approach of PACE. This is true, because the user can now perform a large amount of analysis for many applications without the need for him/her to interfere. The automation process eliminates the need of manually obtaining ACT output and inserting it to an application object for a later use. The dynamic instrumentation gives PACE the ability to extract data that were hidden and unobtainable prior to execution. These data are essential for producing reliable prediction results from PACE. The user can now be satisfied that correct information was gathered for the application in question and that PACE is performing prediction analysis using reliable data.

We have used two examples, a matrix multiplication program, and a program to find the lowest value in an array, to illustrate our methodology. The results show that with the automated dynamic instrumentation, an application execution can now be analysed and predicted even when it is running in dynamic environment.

We believe that the methodology presented here may pave the way for the development of future generations of performance prediction tools, which currently lack the ability to accurately and dynamically analyse or predict an application performance on a given platform.

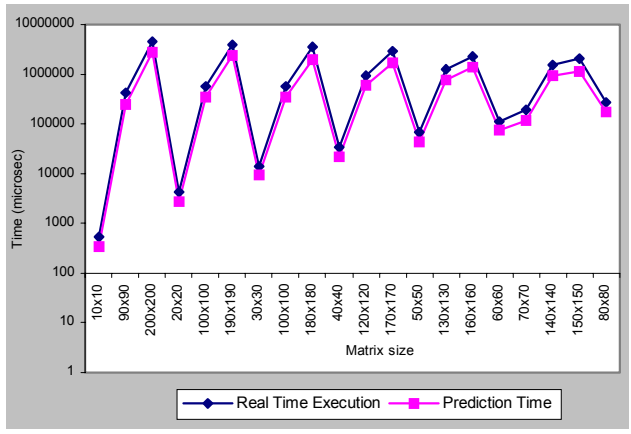


Fig. 7 Matrix Multiplication Performance Prediction Using Dynamic Instrumentation vs. Real Time Execution.

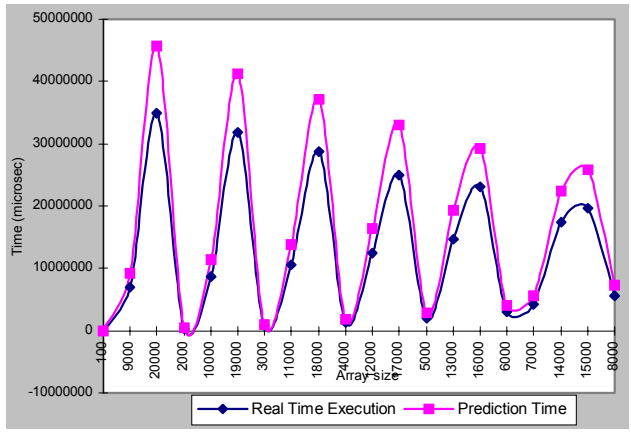


Fig. 8. Dynamically Instrumented Prediction for a Program to Find the Lowest Value in an Array.

Acknowledgments

This work is funded in part by DARPA contract N66001-97-C-8530, awarded under the Performance Technology Initiative administered by NOSC. Mr. Alkindi is supported on a scholarship by the Sultan Qaboos University, Oman.

References

1. Nudd, G.R., Papaefstathiou, E., et.al., A layered Approach to the Characterization of Parallel Systems for Performance Prediction, in Proc. of Performance Evaluation of Parallel Systems, Warwick (1993) 26-34.
2. Foster, I., Kesselman, C.: The Grid, Morgan Kaufmann, (1998).
3. Gu, W., Eisenhauer, G., Schwan, K.: On-line Monitoring and Steering of Parallel Programs, Concurrency: Practice and Experience 10 9 (1998) 699-736.
4. Miller, B.P., Callaghan, M.D., Cargille, J.M., Hollingsworth, J.K., Irvin, R.B., Karavanic, K.L., Kunchithapadam, K., Newhall, T.: The Paradyn Parallel Performance Measurement Tools, IEEE Computer 28 11 (1995) 37-46.
5. DeRose, L., Zhang, Y., Reed, D.A.: SvPablo: A Multi-Language Performance Analysis System, in Proc. 10th Int. Conf. on Computer Performance, Spain (1998) 352-355.
6. Papaefstathiou, E., Kerbyson, D.J., Nudd, G.R., Atherton, T.J.: An overview of the CHIP3S performance prediction toolset for parallel systems, in: 8th ISCA Int. Conf. on Parallel and Distributed Computing Systems, Florida (1995) 527-533.
7. Karen L. Karavanic and Barton P. Miller, Improving Online Performance Diagnosis by the Use of Historical Performance Data, SC'99, Portland, Oregon (USA) November 1999.
8. Jeffrey K. Hollingsworth, Barton P. Miller, Marcelo J. R. Gonçalves, Oscar Naim, Zhichen Xu and Ling Zheng, MDL: A Language and Compiler for Dynamic Program Instrumentation, International Conference on Parallel Architectures and Compilation Techniques San Francisco, California, November 1997.
9. Smith, C.U.: Performance Engineering of Software Systems, Addison Wesley (1990).
10. Harper, J.S., Kerbyson, D.J., Nudd, G.R.: Analytical Modeling of Set-Associative Cache Behavior, IEEE Transactions on Computers 48 10 (1999) 1009-1024.
11. Papaefstathiou, E., Kerbyson, D.J., Nudd, G.R., Atherton, T.J.: An overview of the CHIP3S performance prediction toolset for parallel systems, in: 8th ISCA Int. Conf. on Parallel and Distributed Computing Systems, Florida (1995) 527-533.
12. Wilson, R., French, R., Wilson, C., et.al.: An Overview of the SUIF Compiler System, Technical Report, Computer Systems Lab Stanford University (1993).
13. Kerbyson, D.J., Papaefstathiou, E., Harper, J.S., Perry, S.C., Nudd, G.R.: Is Predictive Tracing Too Late for HPC Users?, in High Performance Computing, Kluwer Academic, March 1999, 57-67.