

# Run-time Optimisation Using Dynamic Performance Prediction<sup>1</sup>

A.M. Alkindi<sup>\*</sup>, D.J. Kerbyson<sup>\*</sup>, E. Papaefstathiou<sup>†</sup>, G.R. Nudd<sup>\*</sup>

High Performance Systems Laboratory, Department of Computer Science,  
University of Warwick, UK

Email: {alkindi,djke,grn}@dcs.warwick.ac.uk,

<sup>†</sup> Microsoft Research, Cambridge, UK  
efp@microsoft.com

**Abstract.** With the rapid expansion in the use of distributed systems the need for optimisation and the steering of application execution has become more important. The unquestionable aim to overcome bottle-neck problems, allocation, and performance degradation due to shared CPU time has prompted many investigations into the best way in which the performance of an application can be enhanced. In this work, we demonstrate the impact of using a Performance Prediction Toolset, PACE, which can be used in Dynamic (*On-The-Fly*) decision making for optimising application execution. An example application, the FFTW (The Fastest Fourier Transform in the West), is used to illustrate the approach which itself is a novel method that optimises the execution of an FFT. It is shown that performance prediction can provide the same quality of information as a measurement process for application optimisation but in a fraction of the time and thus improving the overall application performance.

Keywords: Performance Optimisation, Dynamic Performance Prediction, Performance Modeling, Application Steering, FFTW.

## 1 Introduction

Advances in technology, increasing user interaction with complicated systems, and the existence of powerful communication networks, have all made it easier to create high performance solutions. However, the ease in which these solutions can be formulated is highly dependent upon the complexity of the available resources, and the nature of the applications involved. A significant amount of work is being undertaken to ease this process – much of which is based on the use of performance information to guide the application execution on to the target systems. The promise of Information Power GRIDS [1] relies on the availability of systems and the utilisation of performance information to guide application execution.

Several performance tool-sets have been proposed and implemented dealing mainly with performance instrumentation and measurement. Tools such as Falcon [2],

---

<sup>1</sup> in High Performance Computing and Networking, LNCS, Vol. 1823, Springer-Verlag, May 2000, pp. 280-289.

Paradyn [3], and Pablo [4] are being used to assist in the performance tuning and identification of bottlenecks in applications. These tool-sets typically include various features to allow application instrumentation at various levels, data collection, and interrogation through visualisation modules. Most of the performance monitoring tools are used for post-mortem analysis – i.e. to investigate the performance after the event has occurred. This may be in the analysis of the performance data after the end of application execution, or dynamically as the application is being executed. Other performance tools are being used to identify bottlenecks that may be inherent in the application design on particular systems. For example TASS is concerned with the relationship between parallel algorithms and architectures [5].

There has been very little work on dynamic optimisation using performance prediction while an application is being executed. Modelling approaches have the potential to provide the same performance information but without measurement on the systems. Accuracy of the performance information is important, and should be qualitatively the same as the measurements.

Several performance prediction tools are being developed including PAMELA [6,7] which enables a performance model to be constructed at compile time and combined with applications. This allows access to performance information whilst the program is executing.

In this work, we demonstrate the impact of using Dynamic (*On-The-Fly*) performance prediction to guide the execution of sequential applications. An example high performance application, the FFTW (Fastest Fourier Transform in the West) from MIT [8,9], is used to illustrate the possible use of this approach.

The toolset being developed at Warwick, PACE (Performance Analysis and Characterisation Environment) [10,11] encompasses the performance aspects of application software, its resource use and mapping, and the performance characteristics of hardware systems. It enables performance models to be constructed using an underlying performance specification language, CHIP<sup>3</sup>S (Characterisation Instrumentation for Performance Predication of Parallel Systems) [11], and to be evaluated dynamically requiring only a few seconds of CPU time. The model can be compiled into a self-contained binary which can be executed and linked with an application.

In this paper we show how a PACE performance prediction model can be used to dynamically determine the execution behavior of an application. The FFTW is used to demonstrate this capability but the approach is general and could be used in systems which are dynamic in nature such as the Information Power GRIDs. The overall performance of the FFTW can be improved due to a reduction in time required to determine the best FFT calculation method for a given target processor. The outcome of this work is to provide a performance modelling approach which can be used for dynamic decision making, and also provides improvements to the FFTW performance.

The use of dynamic performance prediction is introduced in Section 2. An overview of the PACE toolset is given in Section 3. In Section 4, the FFTW package is described and the use of PACE models are illustrated in Section 5. Results comparing the default behavior of the FFTW and the FFTW with PACE performance models are given in Section 6. These show that dynamic performance prediction can be used to accurately undertake the decision making processes to optimise the FFTW execution.

## 2 Application Steering using Performance Prediction

There are a number of decisions that a user has to make in order to execute an application. These are traditionally specified through the use of two types of application parameters, broadly classed into two categories:

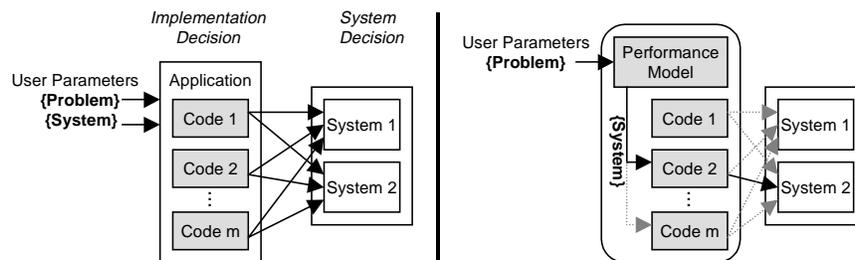
*Problem parameters* - those that specify the format of the data to be processed e.g. data size, and the type of results required such as accuracy in a numerical calculation, the number of iterations required in an iterative solution etc.

*System parameters* – those that specify how a target system will be utilised, e.g. in specifying the number of processors to be used in a high performance system, and possibly also to specify the platform to be utilised when several are available.

The problem parameters need to be specified by the user (always) and depend on the calculation required. However, the system parameters are used to determine the mapping of the application onto the available system, and are normally used to reduce execution time. By coupling a performance model into the application, predictions can be automatically made to determine these system parameters. These *on-the-fly* decisions can be made with negligible overhead in comparison to the total application execution time if the performance model is rapid in its evaluation.

In addition, there are many applications in which numerical routines are used which may have many methods available for solution. Consider the situation depicted in Fig. 1a where an application has several methods of solution and two target systems are available. In this scenario, the user traditionally specifies the problem parameters, decides on the actual code to be used, and the target system along with relevant system parameters.

The use of a performance model in this example can be used to determine which code should be executed, on which target system, along with the relevant system parameters, based on achieving the minimum execution time. Thus, the user need only specify the problem parameters, with the performance model directing the application execution. Fig. 1b illustrates this situation, and shows the chosen code and system (solid arrows), and other available choices (dotted arrows).



**Fig. 1.** Execution of an application having multiple code implementations and available systems. (a) User directed approach, (b) Performance directed approach.

The decisions made by the performance model require a number of separate scenarios to be evaluated. Each scenario is a function of the mapping, the implementation, and the available system(s). The best scenario is taken to be that

which results in a minimum predicted execution time over the total number of scenarios evaluated. Thus, rapid performance model evaluation time is required.

The FFTW is an example code having multiple implementations which minimise the execution of a Fast Fourier Transform using an extensive measurement procedure [8,9]. It is shown in Section 4 how this measurement procedure can be replaced using performance prediction.

In addition, individual performance models can be used collectively within a task scheduling environment in which a scheduler is responsible for maintaining useful activity on system resources. The scheduling process is enhanced by use of predicted execution times, and mapping information from each task, e.g. [12].

### **3 The Performance Analysis and Characterisation Toolset (PACE)**

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 sections.

#### **3.1 Model Components**

Many existing techniques, particularly for the analysis of serial machines, use Software Performance Engineering (SPE) methodologies [13], 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 into account concurrency. 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.

*Application Subtask Layer* – describes the sequential part of every subtask 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 comprised 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 in separate component models e.g. [14].

### 3.2 Model Creation

PACE users can employ a workload definition language (CHIP<sup>3</sup>S) to describe the characteristic of the application. CHIP<sup>3</sup>S 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 CHIP<sup>3</sup>S 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. 2. SUIF (Stanford Intermediate Format) [15] is an intermediate presentation of the compiled code that combines the advantages of both high level and assembly language. ACT cannot determine dynamic performance related aspects of the application such as data dependent parameters. These parameters can be obtained either by profiling or with user support.

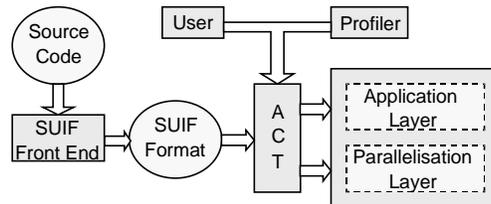


Fig. 2. Model Creation Process with ACT.

The CHIP<sup>3</sup>S objects adhere to the layered framework. A compiler translates CHIP<sup>3</sup>S scripts to C code which are 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 (PICL, SDDF) that can then further analyzed by visualization tools (e.g. PABLO) [16].

## 4 The Fastest Fourier Transform in the West (FFTW)

FFTW is a Portable C package that computes one or more dimensional complex discrete Fourier Transform (DFT) [8,9]. It is claimed that the FFTW can compute the transform in a faster time than other available software due to its self-configuring nature. It requires a sequence of measurements to be made for its software components (codelets) on the target system, and then uses a configuration of these to determine how best to compute the transform of any given size. There are three essential components that make up the FFTW, shown in Fig. 3.

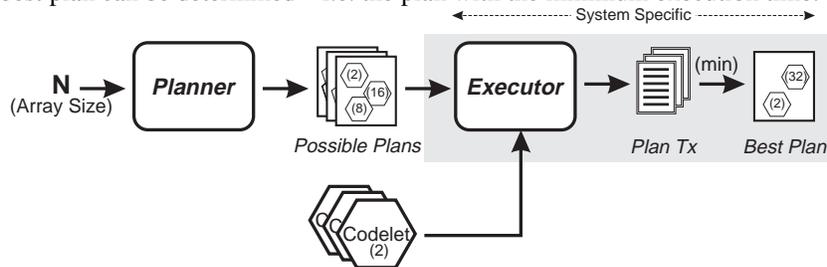
*Codelets:* These are a collection of highly optimized blocks of C routines, which are used to compute the transform. The codelets are generated automatically. There are two types of Codelets:

*Non-Twiddle:* codelets that are used to solve small sized transforms ( $N \leq 64$ ), and

*Twiddle:* codelets that solve larger transforms by the combination of smaller ones.

*Planner:* The planner determines the most efficient way in which the codelets can be combined together to calculate the required transform. Each individual combination of codelets is called a plan. Typically 10's of plans are available for a specific FFT size.

*Executor:* The execution of each plan is performed and measured by the Executor. This results in the run-time cost of each plan on a specific system. From this, the best plan can be determined – i.e. the plan with the minimum execution time.



**Fig. 3.** Main components of the FFTW

For example, for the FFTW to compute the transform of a complex array with a size  $N$ , the *Executor* factors it first into  $N = N_1 N_2$ . Then,  $N_2$  transforms of  $N_1$ , and  $N_1$  transforms of  $N_2$  are recursively computed hence calculating the exact transform of the complex array recursively.

Assuming that a complex array of size  $N=64$  is to be transformed, the number of possible FFTW plans is 20 as shown in Table 1. The plans are forwarded one by one to the Executor for measurement on the target system. The resulting execution times from each plan are included in Table 1 for a SUN Ultra 1. In this plans 1 to 6 contain only a twiddle codelet, and the remaining plans contain both a Twiddle and Non-Twiddle codelets. Plans that compute the full transform are numbers 1, and 16-20 (the remaining plans represent transforms of smaller sizes and may be a part of other plans). In this case, the best (fastest) plan to solve the  $N=64$  transform is Plan 16, which is computed by the Non-Twiddle Codelet 2 and Twiddle Codelet 32.

**Table 1.** Example FFTW plans for N=64 (Timings for SUN Ultra 1)

Plan #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
1	⊗																				○
2		⊗				⊗	⊗	⊗	○												
3			○																		
4				⊗				○	⊗		⊗	○		⊗	○						○
5					⊗																
6																					
7																					
8				⊗						○				○							⊗
9																					
10																					
16			⊗																		
32		⊗																			
64	⊗																				⊗
Time $\mu$ S	65	27	11	5	2	1	7	11	14	20	24	29	41	45	51	63	88	92	96	112	

The performance of FFTW relies heavily on the successful selection of the best plan. The procedure for selecting the optimum configuration comes with a high initial cost that involves the measurement of every plan, a process that needs to be repeated several times in order to remove measurement noise induced by system load. The initial tuning stage cost is exponentially increases when the size of the problem grows due to the resulting larger number of possible transformations.

## 5 Dynamically predicting the FFTW performance

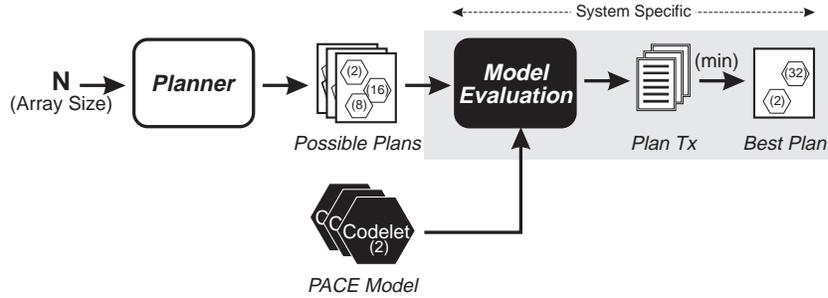
The measurement procedure required by the FFTW to select the best plan for a specific system can be replaced by using a set of PACE performance models. Separate models can be formed of individual codelets, and the operation of the planner implemented by selecting the minimum predicted execution time of an appropriate set of codelet models.

The model that represents the initial tuning stage of the FFTW includes a single application task that mimics the operation of the Planner. The models for the individual codelets are automatically generated by analyzing the existing codelet codes by ACT. A subtask represents an individual codelet called by the planner to produce a possible solution.

The PACE performance model generates and evaluates all possible plan models, instead of actually executing the codelets on the target platform. The planner uses the individual codelet model predictions to combine them to overall plan performance predictions. The planner keeps a record of all plan predicted times in order to identify the fastest combination. The predicted optimum plan is then used by the FFTW executor to perform the transformation. The evaluation of each plan can be performed for a range of systems and problems sizes. The evaluation time for this process is orders of magnitude smaller than the corresponding measurements. An overview of the prediction process is shown in Fig. 4.

The validity of the PACE optimization depends on the selection of the optimum plan. To achieve this the performance models require accurate prediction of each FFTW plan's execution time. The advantage of using PACE models is to minimize the initial measurement time that FFTW requires for each data size and each new

target system. Another advantage of using PACE to predict the best plan is reduce the influence of background loading which may effect the measurement procedure.



**Fig. 4.** Components of the FFTW using PACE performance prediction.

## 6 Results

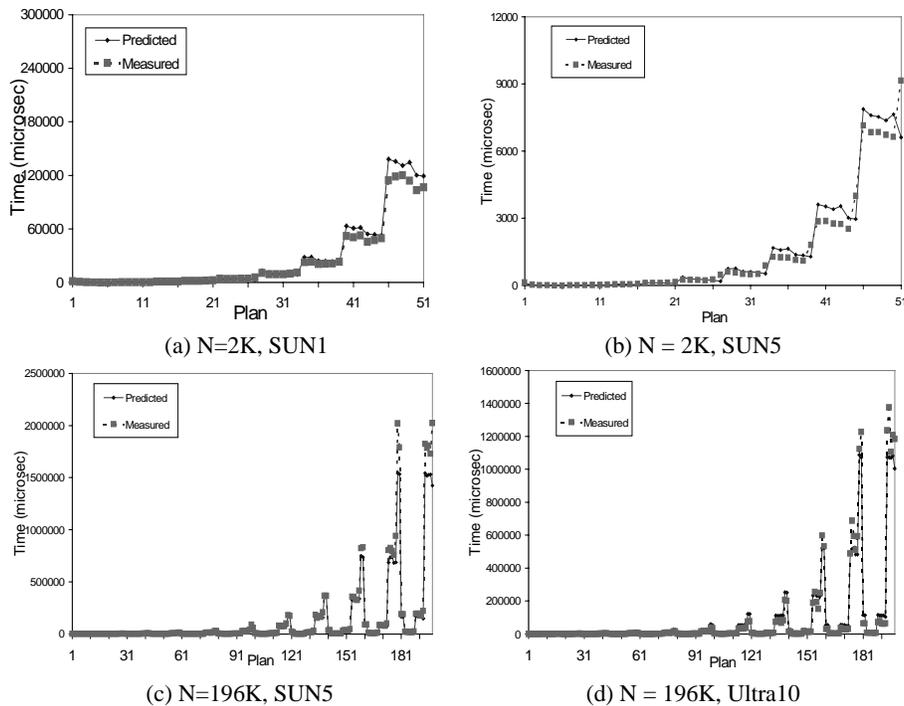
In this section we present a comparison of plan execution times (measured from FFTW) versus plan predicted times evaluated from PACE models for a range of array data sizes and workstations. In all cases PACE and FFTW best plans are calculated to be the same.

Four sets of comparisons between measurements and predictions for the execution of the FFTW for different configurations are shown in Figs. 5a to 5d. In all cases, the X-axis includes all the plans generated by the FFTW planner. Note that the plans generated include combinations of codelets that might not successfully compute the entire FFT. The FFTW planner generates redundant plans assuming that they might be needed by future twiddle codelets.

**Table 2.** Comparison of plan execution time (s) versus model evaluation times for a range of array sizes running on SUN Ultra 10 and SUN Ultra 1 workstations.

Size	SUN Ultra 1		SUN Ultra 10	
	FTTW (s)	PACE (s)	FTTW (s)	PACE (s)
256	0.006	0.007	0.003	0.005
2 K	0.059	0.010	0.032	0.006
128 K	7.069	0.130	3.842	0.009
196 K	16.537	0.028	8.761	0.018

Table 2 shows the time required for the FFTW executor to measure all plan scenarios vs. the time required to evaluate the plan models for several problem and system configurations. The array size varies from small arrays (256 elements) to more realistic cases with 196K elements. Timings are provided for the Sun Ultra 10 and Sun Ultra 1 workstations. With the exception of the 256 elements case the PACE model evaluation times are orders of magnitude faster than the execution times. Thus using PACE models results in a performance improvement to the FFTW.



**Fig. 5.** Comparison of FFTW execution times and PACE Performance Predictions.

## 7 Conclusion

In this work we have shown how a novel performance prediction system may be applied for on-the-fly performance prediction to steer the execution of applications. The PACE system enables a performance model to be incorporated into an application executable and can be used in a decision making procedure to determine how the application can be executed on the target system. Accurate predictions can be produced using this toolset, and rapid evaluation of the performance models enables on-the-fly use.

The FFTW high performance application was used to illustrate the approach. The existing application contained a costly measurement process to determine how best to calculate the transform on the target system. By incorporating a PACE performance model, it was shown that the same process could be undertaken using performance prediction in a fraction of the time, but resulting in the same transform calculation.

The PACE system is currently being extended to provide support for performance prediction in computational environments which may be dynamically changing, and to aid the scheduling of multiple applications on available resources. This corresponds in part to the challenges currently posed by the development of Computational GRIDs

## Acknowledgement

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. Foster, I., Kesselman, C.: *The Grid*, Morgan Kaufmann, (1998).
2. Gu, W., Eisenhauer, G., Schwan, K.: On-line Monitoring and Steering of Parallel Programs, *Concurrency: Practice and Experience* 10 9 (1998) 699-736.
3. 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.
4. 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.
5. Ramachandran, U., Venkateswaran, H., Sivasubramaniam, A., Singla, A.: Issues in Understanding the Scalability of Parallel Systems, in *Proc. of the 1st Int. Workshop on Parallel Processing*, Bangalore, India (December, 1994) 399-404.
6. Van Gemund, A.J.C., Reijns, G.L.: Predicting Parallel System Performance with Pamela, in *Proc. 1st Annual Conf. of the Advanced School for Computing and Imaging*, Heijen, The Netherlands (1995) 422-431.
7. Balakrishnan, S., Nandy, S.K., van Gemund, A.J.C.: Modeling Multi-threaded Architectures in PAMELA for Real-time High-Performance Applications, in *Proc. 4th Int. Conf. on High-Performance Computing*, Los Alamitos, California, IEEE Computer Society 407-414 (December, 1997).
8. Frigo, M., Johnson, S.G.: FFTW: An adaptive software architecture for FFT, In *Proc. of the IEEE Int. Conf. on Acoustics Speech, and Signal Processing*, 3, Seattle (1998) 1381-1384.
9. Frigo, M.: A fast Fourier Transform Compiler. in *Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI'99)*, Atlanta (1999).
10. 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.
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. Perry, S.C., Kerbyson, D.J., Papaefstathiou, E., Grimwood, R., Nudd, G.R.: Performance Optimisation of Financial Option Calculations, To Appear in *Parallel Computing*, Elsevier North Holland (2000).
13. Smith, C.U.: *Performance Engineering of Software Systems*, Addison Wesley (1990).
14. 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.
15. Wilson, R., French, R., Wilson, C., et.al.: An Overview of the SUIF Compiler System, Technical Report, Computer Systems Lab Stanford University (1993).
16. 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 (1999) 57-67.