Use of Performance Technology for the Management of Distributed Systems

Darren J. Kerbyson[1], John S. Harper[1], Efstathios Papaefstathiou[2], Daniel V. Wilcox[1], Graham R. Nudd[1]

[1]High Performance Systems Laboratory, Department of Computer Science, University of Warwick, UK
{djke,john}@dcs.warwick.ac.uk
[2]Microsoft Research, Cambridge, UK

Abstract. This paper describes a toolset, PACE, that provides detailed predictive performance information throughout the implementation and execution stages of an application. It is structured around a hierarchy of performance models that describes distributed computing systems in terms of its software, parallelisation and hardware components, providing performance information concerning expected execution time, scalability and resource use of applications. A principal aim of the work is to provide a capability for rapid calculation of relevant performance numbers without sacrificing accuracy. The predictive nature of the approach provides both pre- and post- implementation analyses, and allows implementation alternatives to be explored prior to the commitment of an application to a system. Because of the relatively fast analysis times, these techniques can be used at run-time to assist in application steering and efficient management of the available system resources.

1 Introduction

The increasing variety and complexity of high-performance computing systems requires a large number of systems issues to be assessed prior to the execution of applications on the available resources. The optimum computing configuration, the preferred software formulation, and the estimated computation time are only a few of the factors that need to be evaluated prior to making expensive commitments in hardware and software development. Furthermore, for effective evaluation the hardware system and the application software must be addressed simultaneously, resulting in an analysis problem of considerable complexity. This is particularly true for networked and distributed systems where system resource and software partitioning present additional difficulties.

The current research into GRID based computing [1] have the potential of providing access to a multitude of processing systems in a seamless fashion. That is, from a user's perspective, applications may be able to be executed on such a GRID without the need of knowing which systems are being used, or where they are physically located.

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 149-159, 2000.

Such goals within the high performance community will rely on accurate performance analysis capabilities. There is a clear need to determine the best application to system resource mapping, given a number of possible choices in available systems, the current dynamic behaviour of the systems and networks, and application configurations. Such evaluations will need to be undertaken quickly so as not to impact the performance of the systems. This is analogous to current simple scheduling systems which often do not take into account the expected run-time of the applications being dealt with.

The performance technology described in this work is aimed at provided dynamic performance information on the expected run-time of applications across heterogeneous processing systems. It is based on the use of a multi-level framework encompassing all aspects of system and software. By reducing the performance calculation to a number of simple models, arbitrarily complex systems can be represented to any level of detail.

The work is part of a comprehensive effort to develop a Performance Analysis and Characterisation Environment (PACE), which will provide quantitative data concerning the performance of sophisticated applications running on high-performance systems. Because the approach does not rely on obtaining data of specific applications operating on specific machine configurations, this type of analysis provides predictive information, including:

- Execution Time
- Scalability
- On-the-fly Steering
- System Sizing
- Mapping Strategies
- Dynamic Scheduling

PACE can supply accurate performance information for both the detailed analysis of an application (possibly during its development or porting to a new system), and also as input to resources allocation (scheduling) systems on-the-fly (at run-time).

An overview of PACE is given in the following sections. Section 2 describes the main components of the PACE system. Section 3 details an underlying language used within PACE detailing the performance aspects of the applications / systems. An application may be automatically translated to the internal PACE language representation. Section 4 describes how performance predictions are obtained in PACE. Examples of using PACE performance models for off-line and on-the-fly analysis for scheduling applications on distributed resources is included in Section 5.

2 The PACE System

PACE (Performance Analysis and Characterisation Environment) [2] is a performance prediction and analysis toolset whose potential users include application programmers without a formal training in modeling and performance analysis. Currently, high performance applications based on message passing (using MPI or PVM) are supported. In principal any hardware platform that utilises this programming model can be analysed within PACE, and the technique has been applied to various workstation clusters, the SGI origin systems, and the CRAY T3E to

date. PACE allows the simultaneous utilisation of more than one platform, thus supporting heterogeneous systems in meta-computing environments.

There are several properties of PACE that enable it to be used throughout the development, and execution (run-time scheduling), of applications. These include:

Lifecycle Coverage – Performance analysis can be performed at any stage in the software lifecycle [3,4]. As code is refined, performance information is updated. In a distributed execution environment, timing information is available on-the-fly for determining which resources should be used.

Abstraction – Different forms of workload information need to be handled in conjunction with lifecycle coverage, where many levels of abstraction occur. These range from complexity type analysis, source code analysis, intermediate code (compile time) analysis, and timing information (at run- time).

Hierarchical – PACE encapsulate necessary performance information in a hierarchy. For instance an application performance description can be partitioned into constituent performance models. Similarly, a performance model for a system can consist of many component models.

Modularity – All performance models incorporated into the analysis should adhere to a strict modular structure so that a model can easily be replaced and re-used. This can be used to give comparative performance information, e.g. for a comparison of different system configurations in a meta-computing environment.

The main components of the PACE tool-set are shown in Fig. 1. A core component of PACE is the performance language, CHIP^{3}S (detailed in Section 3) that describes the performance aspects of an application and its parallelisation. Other parts of the PACE system include:

Object Editor – to assist in the creation and editing of individual performance objects. Pre-defined objects can be re-used through an object library system.

Source Code Analysis – enables source code to be analysed and translated into CHIP^{3}S. The translation performs a static analysis of the code, and dynamic constructs are resolved either by profiling or user specification.

Compiler – translates the performance scripts into C language code, linked to an evaluation library and specific hardware objects, resulting in a self-contained executable. The performance model remains parameterised in terms of system configurations (e.g. processor mapping) and application parameters (data sizes).

Hardware Configuration – allows the definition of a computing environment in terms of its constituent performance model components and configuration information. An underlying Hardware Modeling and Configuration Language (HMCL) is used.

Evaluation Engine –combines the workload information with component hardware models to produce time predictions. The output can be either overall execution time estimates, or trace information of the expected application behavior.

Performance Analysis – both ‘off-line’ and ‘on-the-fly’ analysis are possible. Off-line analysis allows user interaction and can provide insights into expected performance. On-the-fly analysis facilitates dynamic decision making at run-time, for example to determine which code to be executed on which available system.

There is very little restriction on how the component hardware models can be implemented within this environment, which allows flexibility in their design and

implementation. Support for their construction is currently under development in the form of an Application Programming Interface (API) that will allow access to the CHIP^{3}S performance workload information and the evaluation engine.

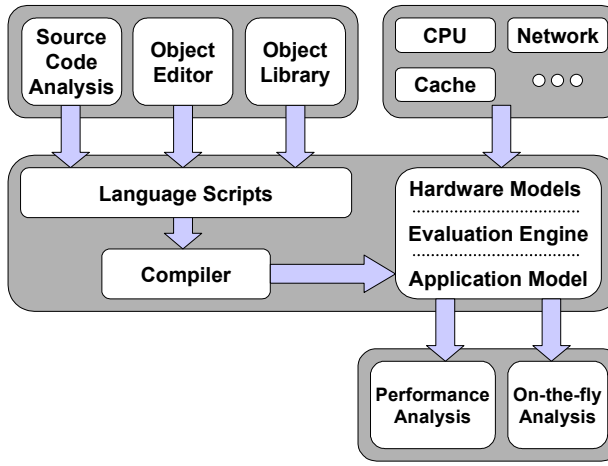

Fig. 1. Schematic of the PACE System.

3 Performance Language

A core component of PACE is the specialised performance language, CHIP^{3}S (Characterisation Instrumentation for Performance Prediction of Parallel Systems) [5] based on Software Performance Engineering principles [6]. This language has a strict object structure encapsulating the necessary performance information concerning each of the software and hardware components. A performance analysis using CHIP^{3}S comprises many objects linked together through the underlying language. It represents a novel contribution to performance prediction and evaluation studies.

3.1 Performance Object Hierarchy

Performance objects are organised into four categories: application, subtask, parallel template, and hardware. The aim of this organisation is the creation of independent objects that describe the computational parts of the application (within the application and subtask objects), the parallelisation strategy and mapping (parallel template object), and the system models (hardware object). The objects as follows:

Application Object - acts as the entry point to the performance model, and interfaces to the parameters in the model (e.g. to change the problem size). It also specifies the system being used, and the ordering of subtask objects.

Subtask Objects - represents one key stage in the application and contains: a description of sequential parts of the parallel program. These are modeled using CHIP^{3}S procedures which may be automatically formed from the source code.

Parallel Template Objects - describes the computation–communication pattern of a subtask object. Each contains steps representing a single stage of the parallel algorithm. A step defines the hardware resource.

Hardware Objects - The performance aspects of each system are encapsulated into separate hardware objects - a collection of system specification parameters (e.g. cache size, number of processors), micro-benchmark results (e.g. atomic language operations), statistical models (e.g. regression communication models), analytical models (e.g. cache, communication contention), and heuristics.

A hierarchical set of objects form a complete performance model. An example of a complete performance model, represented by a Hierarchical Layered Framework Diagram (HLFD) is shown in Fig. 2. The boxes represent the individual objects, and the arcs show the dependencies between objects in different layers.

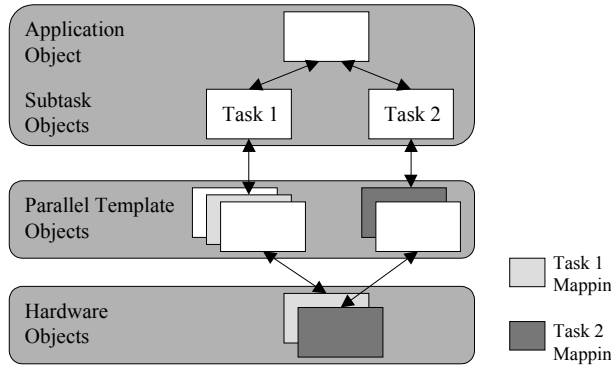

Fig. 2. Example HLFD illustrating possible parallelisation and hardware combinations.

In this example, the model contains two subtask objects, each with associated parallel templates and hardware objects. When several systems are available, there are choices to be made in how the application will be mapped. Such a situation is shown in Fig. 2 where there are three alternatives of mapping Task 1 (and two for Task 2) on two available systems. The shading also indicates the best mapping to these systems. Note that the two tasks are found to use different optimal hardware platforms.

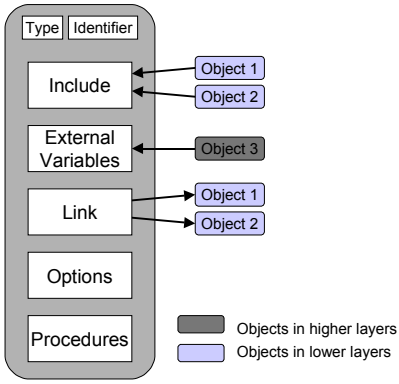

Fig. 3. Performance object structure

Include – references other objects used lower in the hierarchy.
External Variables – variables visible to objects above in the hierarchy.
Linking – modifies external variables of objects lower in the hierarchy
Option – sets default options for the object
Procedures –structural information for either: sub-task ordering (application object), computational components (sub-task objects), or computation / communication structure (parallel template objects).

3.2 Performance Object Definition

Each object describes the performance aspects of the corresponding system component but all have a similar structure. Each is comprised of internal structure (hidden from other objects), internal options (governing its default behavior), and an interface used by other objects to modify their behavior. A schematic representation of an object, in terms of its constituent parts, is shown in Fig. 3: A full definition of the CHIP^{3}S performance language is out of the scope of this paper [7].

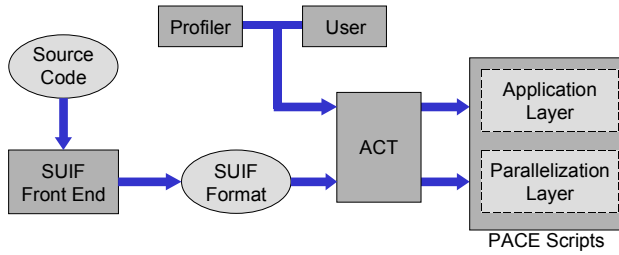

Fig. 4. Model creation process with ACT.

3.3 Software Objects

Objects representing software within the performance model are formed using ACT (*Application Characterisation Tool*). ACT provides semi-automated methods to produce performance models from existing sequential or parallel code for both the computational and parallel parts of the application, Fig. 4. Initially, the application code is processed by the SUIF front end [8] and translated into SUIF. The unknown parameters within the program (e.g. loop iterations, conditional probabilities) that cannot be resolved by static analysis are found either by profiling or user specification. ACT can describe resources at four levels:

Language Characterisation (HLLC) – using source code (C and Fortran supported).
Intermediate Format Code Characterisation (IFCC) - characterisation of compiler representation (SUIF, Stanford University Intermediate Format, is supported) .
Instruction Code Characterisation (ICC) – using host assembly.
Component Timings (CT) - application component benchmarked. This produces accurate results but is non-portable across target platforms.

3.4 Hardware Objects

For each hardware system modeled an object describes the time taken by each resource available. For example, this might be a model of the time taken by an inter-processor communication, or the time taken by a floating-point multiply instruction. These models can take many different forms, ranging, from micro-benchmark timings of individual operations (obtained from an available system) to complex analytical models of the devices involved. One of the goals of PACE is to allow hardware objects to be easily extended. To this end an API is being developed that will enable third party models to be developed and incorporated into the prediction system.

Hardware objects are flexible and can be expressed in many ways. Each model is described by an evaluation method (for the hardware resource), input configuration parameters, and access to relevant workload information. Three component models are included in Fig 5. The workload information is passed from objects in upper layers, and is used by the evaluation to give time predictions. The main benefit of this structure is the flexibility; analytical models may be expressed by using complex modeling algorithms and comparatively simple inputs, whereas models based on benchmark timings are easily expressed but have many input parameters.

To simplify the task of modeling many different hardware systems, a hierarchical database is used to store the configuration parameters associated with each hardware system. The Hardware Model Configuration Language (HMCL) allows users to define new hardware objects by specifying the system-dependent parameters. On evaluation, the relevant sets of parameters are retrieved, and supplied to the evaluation methods for each of the component models. In addition, there is no restriction that the hardware parameters need be static - they can be altered at run-time either to refine accuracy, or to reflect dynamically changing systems.

Component models currently in PACE include: computational models supporting HLLC, IFCC, ICC and CT workloads, communication models (MPI & PVM), and multi-level cache memory models [9]. These are all generic models (the same for all supported systems), but are parameterised in terms of specific system performances.

4 Model Evaluation

The evaluation engine uses the CHIP^{3}S performance objects to produce predictions for the system. The evaluation process is outlined in Fig. 5. Initially, the application and sub-task objects are evaluated, producing predictions for the workload. These predictions are then used when evaluating computation steps in the parallel templates.

Calls to each component hardware device are passed to the evaluation engine. A dispatcher distributes input from the workload descriptions to an event handler and then to the individual hardware models. The event handler constructs an event list for each processor being modeled. Although the events can be identified through the target of each step in the parallel template, the time spent using the device is still unknown at this point. However, each individual hardware model can produce a time prediction for an event based on its parameters. The resultant prediction is recorded in the event list. When all device requests have been handled, the evaluation engine processes the event list to produce an overall performance estimate for the execution time of the application (by examining all event lists to identify the step that ends last).

Processing the event list is a two-stage operation. The first stage constructs the events, and the second resolves ordering dependencies, taking into account contention factors. For example, in predicting the time for a communication, the traffic on the inter-connection network must be known to calculate channel contention. In addition, messages cannot be received until after they are sent! The exception to this type of evaluation is a computational event that involves a single CPU device - this can be predicted in the first stage of evaluation (interaction is not required with other events).

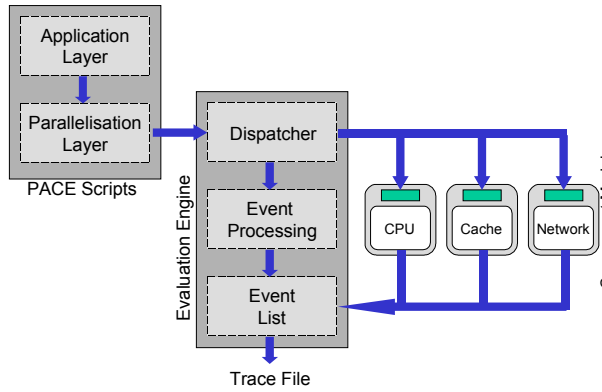

Fig 5. The evaluation process to produce a predictive trace within PACE.

The ability of PACE to produce predictive traces derives directly from the event list formed during model evaluation. Predictive traces are produced in standard trace formats. They are based on predictions and not run-time observations. Two formats are supported by PACE: PICL (Paragraph), and SDDF (PABLO) [10].

5 Performance Models in Use

The PACE system has been used to investigate many codes from several application domains including image processing, computational chemistry, radar, particle physics, and financial applications. PACE performance models are in the form of self-contained executable binaries parameterised in terms of application and system configuration parameters. The evaluation time of a PACE performance model is rapid

(typically seconds of CPU use) as a consequence of utilising many small analytical component hardware models. The rapid execution of the model lends itself to dynamic situations as well as traditional off-line analysis as described below.

5.1 Off-Line Analysis

A common area of interest in investigating performance is examining the execution time as system and/or problem size is varied. Application execution for a particular system configuration and set of problem parameters may be examined using trace analysis. Fig. 6 shows a predictive trace analysis session within Pablo. In the background, an analysis tree contains an input trace file (at its root node) and, using data manipulation nodes, results in a number of separate displays (leaves in the tree). Four types of displays are shown producing summary information on various aspects of the expected communication behavior. For example, the display in the lower left indicates the communication between source and destination nodes in the system (using contours to represent traffic), and the middle display shows the same information displayed using 'bubbles', with traffic represented by size and colour.

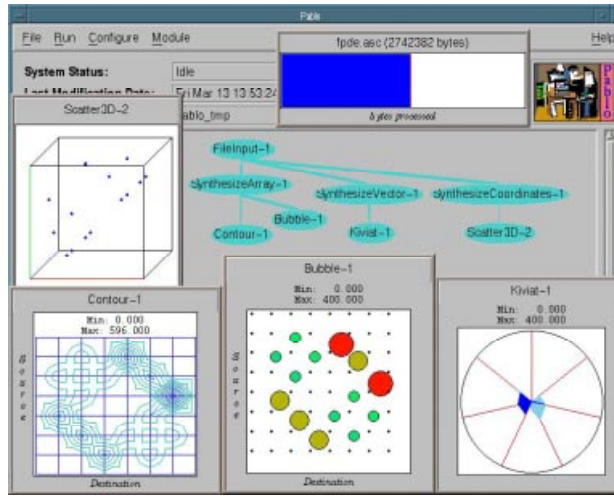

Fig. 6. Analysis of trace data in Pablo

5.2 On-the-Fly Analysis

An important application of prediction data is that of dynamic performance-steered optimisation [11,12] which can be applied for efficient system management. The PACE model is able to provide performance information for a given application on a given system within a couple of seconds. This enables the models to be applied on-the-fly for run-time optimisation. Thus, dynamic just-in-time decisions can be made about the execution of an application, or set of applications, on the available system

(or systems). This represents a radical departure from existing practice, where optimisation usually takes place only during the program's development stage

Two forms of on-the-fly analysis have been put into use by PACE. The first has involved a single image processing application, in which several choices were available during its execution [13]. The second is a scheduling system applied to a network of heterogeneous workstations. This is explained in more detail below.

The console window of the scheduling system, using performance information and a Genetic Algorithm (GA) is shown in Fig. 7. The coloured bars represent the mapping of applications to processors; the lengths of the bars indicate the predicted time for the number of processors allocated. The system works as follows:

1. (An application (and performance model) is submitted to the scheduling system.
2. The GA contains all the performance data for the currently submitted applications, and constantly minimises the execution time for the application set.
3. Applications currently executing are 'fixed' and cannot change the schedule.
4. Feedback updates the GA on premature completion, or late-running applications.

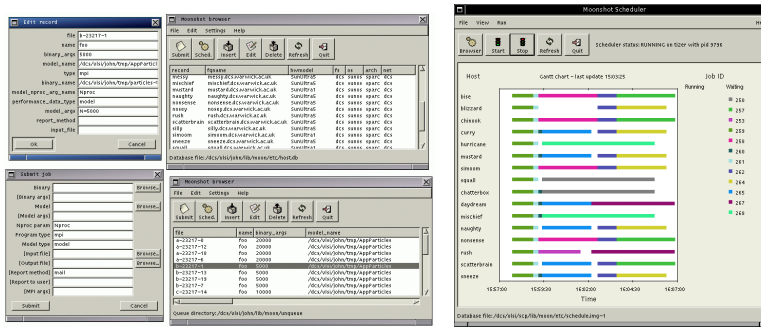

Fig. 7. Console screen of the PACE scheduling system showing the system and task interfaces (left panels) and a view of the Gantt chart of queued tasks (right panel).

One particular advantage of the GA method over the other heuristics tried is that it is an evolutionary process, and is therefore able to absorb slight changes, such as the addition or deletion of programs from its optimisation set, or changes in the resources available in the computing system.

6 Conclusion

This work has described a methodology for providing predictive performance information on parallel applications using a hierarchical model of the application, its parallelisation, and the distributed system. An implementation of this approach, the PACE toolset, has been developed over last three years, and has been used to explore performance issues in a number of application domains. The system is now in a

position to provide detailed information for use in static analysis, such as trace data, and on-the-fly analysis for use in scheduling and resource allocation schemes.

The speed with which the prediction information is calculated has led to investigations into its use in dynamic optimisation of individual programs and of the computing system as a whole. Examples have been presented of dynamic algorithm selection and system optimisation, which are both performed at run-time. These techniques have clear use for the management of dynamically changing systems and GRID based computing environments.

Acknowledgement

This work is funded in part by DARPA contract N66001-97-C-8530, awarded under the Performance Technology Initiative administered by NOSC.

References

1. I Foster, C Kesselman, „The GRID“, Morgan Kaufman (1998)
2. G.R. Nudd, D.J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper, D.V. Wilcox, „PACE – A Toolset for the Performance Prediction of Parallel and Distributed Systems“, in the *Journal of High Performance Applications*, Vol. 14, No. 3 (2000) 228-251
3. D.G. Green et al. „HPCN tools: a European perspective“, *IEEE Concurrency*, Vol. 5(3) (1997) 38-43
4. I. Gorton and I.E. Jelly, „Software engineering for parallel and distributed systems, challenges and opportunities“, *IEEE Concurrency*, Vol. 5(3) (1997) 12-15
5. E. Papaefstathiou et al., „An overview of the CHIP^{3}S performance prediction toolset for parallel systems“, in *Proc. of 8th ISCA Int. Conf. on Parallel and Distributed Computing Systems* (1995) 527-533
6. C.U. Smith, „Performance Engineering of Software Systems“, Addison Wesley (1990).
7. E. Papaefstathiou et al., „An introduction to the CHIP^{3}S language for characterising parallel systems in performance studies“, Research Report RR335, Dep. of Computer Science, University of Warwick (1997)
8. Stanford Compiler Group, *„*The SUIF Library*“*, *The SUIF compiler documentation set*, Stanford University (1994)
9. Harper, J.S., Kerbyson, D.J., Nudd, G.R.: Analytical Modeling of Set-Associative Cache Behavior, IEEE Transactions on Computers, Vol. 48(10) (1999) 1009-1024
10. D.A. Reed, et al., „Scalable Performance Analysis: The Pablo Analysis Environment“, in: *Proc. Scalable Parallel Libraries Conf.*, IEEE Computer Society (1993)
11. R. Wolski, „Dynamically Forecasting Network Performance Using the Network Weather Service“, UCSD Technical Report, TR-CS96-494 (1996)
12. J. Gehring, A. Reinefeld, „MARS - A framework for minizing the job execution time in a metacomputing environment“, *Future Generation Computer Systems*, Vol. 12 (1996) 87-99
13. D.J. Kerbyson, E. Papaefstathiou and G.R. Nudd, „Application execution steering using on-the-fly performance prediction“, in: *High Performance Computing and Networking*, Vol 1401, LNCS Springer-Verlag (1998) 718-727