

A Common Workload Interface for the Performance Prediction of High Performance Systems¹

E. Papaefstathiou, D.J. Kerbyson, G.R. Nudd,
D.V. Wilcox, J.S. Harper, S.C. Perry

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

Abstract

There is a wide range of performance models being developed for the analysis of present and future generation systems. A major concern in using these models however, is the provision of realistic application workload information. There is a need for a common interface that couples generic workload information to the underlying hardware characteristics. In this work a three level framework is described which enables the definition of workload information (both computational and mapping), and hardware performance characteristics to be included, in a modular and reusable format. The approach lends itself to performance studies that can be used in cross-platform performance comparisons as well as guiding the design and implementation of code on a single system. This is currently in use for the analysis of High Performance Computing systems but may equally be applicable in other domains where the analysis of resources is a concern.

1. Introduction

The design and implementation of a system is typically a lengthy procedure that requires many choices to be made, many of which will effect the achievable final performance considerably. This is further complicated in high performance computing where complex interactions between processing elements, memory systems and input/output channels take place. In addition, when considering the use of these systems, application code needs to be carefully implemented to take advantage of the underlying hardware in order to achieve performance. There is a clear need for the study of performance at each and every stage of the system development.

The choices available to both hardware and software engineers can be considered as being a sequence of what-if scenarios. That is to say, questions such as 'what is the impact on performance if the software is

implemented like ...' are often asked, and answered (by the engineers) in an ad-hoc manner. Such scenarios occur in many places in the design and implementation. For example:

In hardware design:

Architecture - choice in the system component architecture that effects performance, (e.g. in communication bandwidths, memory design).

In software design:

Algorithmic - choice of core algorithms in the software for key numerical routines, and
Mapping - choice of implementation in terms of parallelisation and use of system resources

In procurement:

System - choice of system for the execution of code, when several are available. This can be considered further, in the choice of a system to be used for the execution of an application when several systems are available. (e.g. for use in on-the fly scheduling [7]).

¹ in IEEE Int. Symp. On Computer Architecture, Workshop on Performance Analysis in Design (PAID'98), Barcelona, June 1998.

Fast and accurate performance prediction can be used to provide information on the resulting performance, which may be used to guide the design [2,12]. Such technology needs to consider hardware specifics, and the workload placed on the system by the application software both in terms of its constituent computation demands, and also its mapping onto the system. In the work described here, a three-tier performance framework is discussed that allows workload information to be defined (both computational and mapping), along with system specific performance characteristics. The framework has been carefully designed using object orientated principles and allows modular descriptions to be interchanged. The framework has been conceived to explore software design and implementation choices but is equally applicable to changes in the target system performance characteristics (in either existing or future systems). As is shown in the following sections, performance studies can be undertaken for different target architectures with no alteration to the workload descriptions. A toolset, PACE, is under development at Warwick which aims to address these issues.

2. A Performance Analysis & Characterisation Environment (PACE)

PACE is a performance prediction and analysis toolset that aims to support the definition of workloads throughout an application's lifecycle from initial design through to execution on a target system. The potential users of the tool include application programmers without a formal training in modelling and performance analysis. Currently, parallel applications written in C, Fortran 77 and 90, that utilise a message passing interface (MPI or PVM) are supported. In principal any hardware platform that utilises this programming model can be analysed within PACE, but the technique has been directed to clusters of workstations and the CRAY T3E to date. PACE allows the utilisation of more than one platform for an application thus providing support for heterogeneous systems.

The core component of PACE is a performance language, CHIP3S (Characterisation Instrumentation for Performance Prediction of Parallel Systems). CHIP3S provides a syntax that allows the description of the performance aspects of an application, and its parallelisation, to be expressed. This includes control flow information, resource usage information (e.g. number of operations), communication patterns, mapping, etc. The CHIP3S compiler translates the performance scripts into C language source code (Figure 1). This is linked to an evaluation engine library and to hardware specific models. The former includes the necessary steps required to combine workload descriptions with the hardware models in

order to produce performance predictions. The final result is an executable binary file that produces performance predictions parameterised in terms of problem sizes and system configurations. The user can change these parameters by use of appropriate command line options.

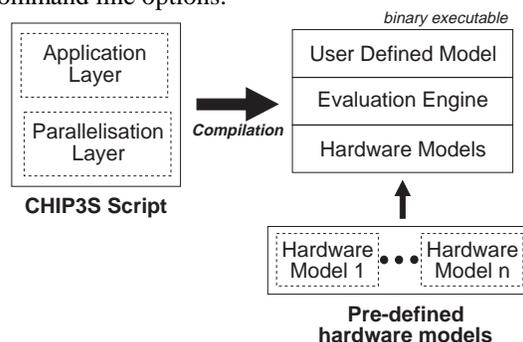


Figure 1 - PACE core system

CHIP3S is an object oriented language that adheres to a layered characterisation framework [9]. According to this framework the objects that comprise a performance study are organised into four categories namely: 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). CHIP3S provides the syntax to define the workload of the system through the application, subtask, and parallel template objects. Hardware objects are pre-defined with their structure, and interface, hidden from the user.

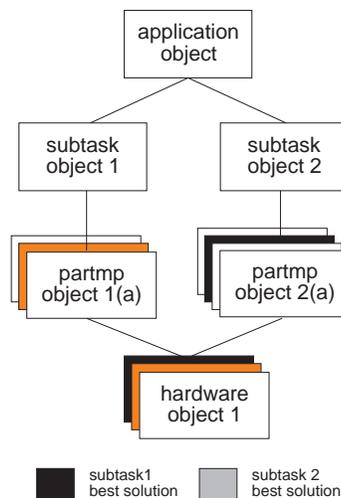


Figure 2 - Example HLF diagram illustrating possible parallelisation and hardware combinations

A key feature of the object organisation is the independent representations of computation, parallelisation, and hardware. This is possible due to

strict object interaction rules enforced by the CHIP3S syntax. The user can experiment to find the best parallelisation strategy and hardware platform pair by interchanging the appropriate objects. For example, consider an application that includes two computational intensive kernels that can be parallelised with a number of parallelisation strategies. Figure 2 shows a Hierarchical Layered Framework Diagram (HLFD) representing such a situation. The computational parts of the kernels are described separately in the two subtask objects, the various parallelisation strategies are described in several parallel template objects associated with each subtask, and a number of pre-defined hardware objects exist to support various hardware platforms. The shaded parallel templates, and hardware objects in the HLFD show the best solution that has been derived after experimentation with alternatives.

The performance aspects of the system are encapsulated into a hardware object. This is 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. Hardware models can be incorporated into PACE through the use of an Application Programming Interface (API). The API allows third party model developers to incorporate their system component models into the PACE hardware library. These models can represent performance characteristics from measurements (for available systems) but also may represent possible performance characteristics for future systems.

The features of PACE that promotes the development of extended workload descriptions for different level of representation abstractions are discussed below.

2.1 Model Abstraction

Different levels of abstraction in performance modelling is a fundamental requirement that is able to support the lifecycle of an application. PACE allows the definition of many levels of model abstraction, and is a combination of two factors:

1. the level of detail of the control flow information, and
2. the type of resource usage associated with each control statement.

For example, consider the two short CHIP3S code segments, 1 and 2, below. These represent the same computation stage of part of an application code. The exact function and the operations performed need not be known for this example.

The computation parts are expressed in CHIP3S as control flow procedures. The statements in the procedure (`compute`, `loop`) might represent an arbitrary number of operations. Each statement is associated with an operation count vector (`<...>`) that represents the resources required to execute that operation. Code 1, includes a fairly detailed representation of the actual code and the operation count done in terms of C language operations. This type of model can be obtained when the source code is available. Code 2 represents the same computation in a high level of abstraction. The control flow is not a one to one correspondence to the actual source code and the operation count considers only to predominant operations (in this case floating point). This type of model might be developed in the analysis and specification stages to determine bounds of performance.

```
subtask merge {
  ...
  proc cflow Txsort {
    compute <is clang, 3*IASG>;
    loop <is clang, WHILE, ICMP>, dtpp
      compute <is clang, ICMP, FCMP, IADD,
        4*VIDX>;
    loop <is clang, WHILE, ICMP>, dtpp
      compute <is clang, IMUL, 2*IADD,
        2*VIDX, FASG>;
    loop <is clang, WHILE, ICMP>, dtpp
      compute <is clang, IASG, 2*IADD,
        2*FASG>;
  }
  ...
}
```

Code 1 - C Operations

```
subtask merge {
  ...
  proc cflow Txsort {
    loop <0>, dtpp
      compute <is flops, CMP, 3*ASG>;
  }
  ...
}
```

Code 2 - Floating Point Operations

The level of abstraction for the control flow is selected by the user depending on the lifecycle stage. PACE provides many resource usage descriptions. Computation operations can be expressed as predominant operations (e.g. floating point [5]), high level language operations (e.g. FORTRAN language [10]), intermediate code operations (e.g. SUIF [3]), and component timing. Communication operations similarly can be high level where only the length of message and the distance amongst the processors is specified. In a detailed level, the communication descriptions can represent the message passing calls of the application in a one to one match and may include initialisation times, data packing, etc.

2.2 Performance Prototyping and Static Performance Prediction

PACE provides tools to automate model development. The *Application Characterisation Tool* (ACT) provides a semi-automated method to produce CHIP3S scripts from existing sequential or parallel code. The process of characterising the application with ACT is shown in Figure 3. Initially, the code is processed through the SUIF front end [3] and the C or FORTRAN application code is translated into the SUIF intermediate format. The program unknowns (e.g. loop iterations, conditional probabilities) that are not resolved by the static analysis are identified by either a profiler or manual user specification. The former provides an automated root in producing CHIP3S scripts but the model can only then be used to predict the performance of the application given a single problem size. Manual parameter specification removes this problem. ACT includes a mechanism to minimise the instances where the user intervention is needed. Given the intermediate format representation of the application and the problem unknowns, ACT produces the CHIP3S source code for both the computation and parallel parts of the application.

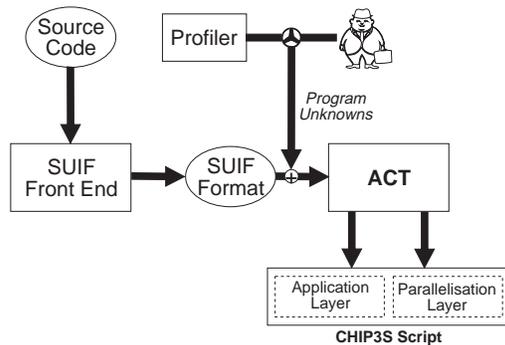


Figure 3 - Model creation process with ACT

During the implementation stage, when the parallel source code is available, ACT can be employed as a static performance prediction tool [1], Figure 4. The performance of the application can then be analysed for several parallel platforms, provided they are available as hardware objects.

PACE allows the development of models even when parts of the source code are not available. *Performance prototyping* is the terminology that is used within PACE for these types of performance studies. While functional prototyping aims to mimic the functionality of an application, performance prototyping aims to mimic the performance characteristics of the application. In the design stage a performance prototype of a parallel algorithm is required and in many cases the computational parts can be extracted by the sequential application with ACT, Figure 4. The parallelisation parts of the model (parallel template objects) can be developed by hand. Since, this is just a prototype only the predominant performance parts of the algorithm should be

represented in the model. For example, if PVM is the message passing employed only the communication statement could be represented in the model (ignoring packing/unpacking, initialisations, etc).

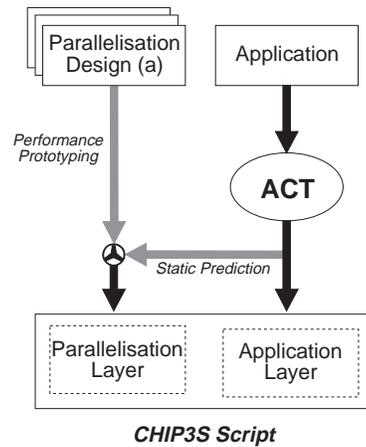


Figure 4 - Use of ACT for static performance estimations and performance prototyping

3. A Case Study

An example of performance analysis using PACE is illustrated for a small case study using an image processing application. IPKERNEL is an image processing application that identifies blob like objects in an image [8]. The application was parallelised for a cluster of workstation with the PVM message passing library. The Hierarchical Layered Framework Diagram of the application is shown in Figure 5. This diagram shows the main components in this application and includes twelve subtasks, four different parallel templates, and a PVM model for a SUN workstation cluster. Each object in this diagram is constructed using a separate CHIP3S performance script and the arcs indicate the passing of parameters between objects. A single application object determines the ordering of the subtasks.

The performance model for the application was produced by ACT with some user guidance to specify a number of program unknowns. PACE provides tools to perform scalability analysis and bottleneck analysis on when varying system and/or application parameters. However, the most commonly output facility is the creation of predictive trace files [6]. These are analogous to traditional traces obtained at run-time but contain predicted time information. The user can employ any monitoring tool to visualise the various performance aspects (e.g. [4,11]). An example predictive trace output is shown in Figure 6 which details a segment of the predictive behaviour of IPKERNEL on seven workstations.

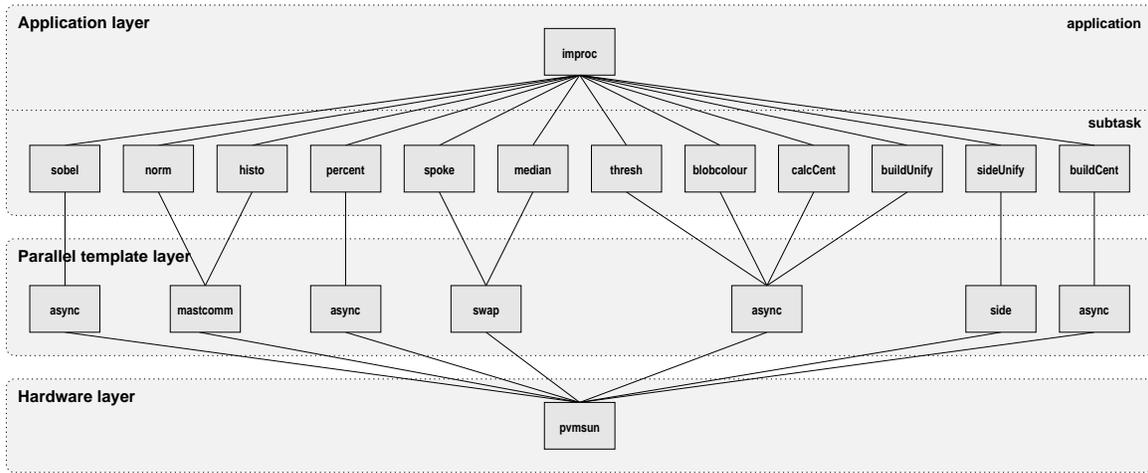


Figure 5 - HLF for the IPKERNEL application

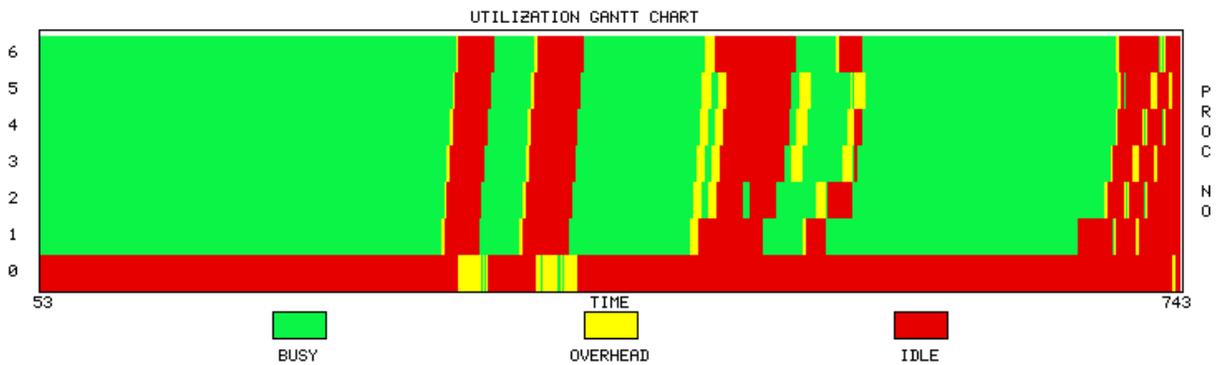


Figure 6 - IPKERNEL utilisation gantt chart produced from Paragraph with predictive traces from PACE

This information can be used to guide the choice of different architectural alternatives or different software strategies during design and implementation stages. These performance insights can be used as and when needed and do not rely on final system implementations. For instance in the output shown in Figure 6 there is a clear region of idle time on processor zero (a result of a master-slave type parallelism with processor zero acting as the master) which maybe avoided in a different implementation.

4. Summary

PACE is a performance analysis environment that promotes the definition of generic workload operations that can be reused by many system models. The three layer structure used, allows the separate definition of application, parallelisation, and hardware characteristics to be made. PACE supports both performance prototyping studies (when the application source code is under design and is not complete) and also static performance prediction from available source codes. In this latter case additional tools are available to automate the procedure of source code translation into CHIP3S

descriptions. Following the workload characterisation, PACE provides automated evaluation procedures that couples the application information with the hardware models.

The current PACE system enables performance predictions to be made for the analysis of resources used in High Performance Computing systems including dedicated machines (the CRAY T3E) and also Cluster computing environments using message passing interfaces (MPI and PVM). Scaling analysis are available along with predicted traces that can be used for application refinement before final implementation on a system.

The modular approach taken in PACE, in separate descriptions of workload information and the system hardware characteristics, may be equally applicable in other domains where the analysis of resources is a concern.

Acknowledgements

This work is funded in part by DARPA contract N66001-97-C-8530, awarded under the Performance Technology Initiative administered by NOSC, and by EPSRC grant GR/L13025.

References

- [1] T. Fahringer, Estimating and Optimizing Performance for Parallel Programs, *IEEE Computer*, Vol. 28(11), November 1995.
- [2] I. Gorton and I.E. Jelly, Software Engineering for Parallel and Distributed Systems: Challenges and Opportunities, *IEEE Concurrency*, Vol 5(3), July-September 1997.
- [3] M.W. Hall, J.M. Anderson, et.al, Maximizing Multiprocessor Performance with the SUIF Compiler, *IEEE Computer*, Vol.29(12), December 1996.
- [4] M.Heath, Recent Developments and Case Studies in Performance Visualization Using Paragraph, In: *Performance Measurement and Visualization of Parallel Systems*, Elsevier Science Publishers, 1993.
- [5] R.W. Hockney, *The Science of Computer Benchmarking*, SIAM, 1996.
- [6] D.J. Kerbyson, E.Papaefstathiou, and G.R. Nudd, Is Predictive Tracing Too Late for HPC Users?, In: R.J. Allan, A. Simpson, and D.A. Nicole. eds., *High Performance Computing*, Plenum Press, 1998.
- [7] D.J. Kerbyson, E. Papaefstathiou, and G.R. Nudd, Application Execution Steering Using On-the-fly Performance Prediction, In: P. Sloot, M. Bubak, and B. Hetzberger (eds.), *High Performance Computing and Networking*, Lecture Notes in Computer Science 1401, Springer-Verlag, 1998.
- [8] G.R. Nudd, T.J. Atherton, and D.J. Kerbyson, IPKERNEL: An Image Processing Benchmark, ESPRIT Project 5669 (MEASURE), Final Report, 1992.
- [9] E. Papaefstathiou, D.J. Kerbyson, and G.R. Nudd, A Layered Approach to Parallel Software Performance Prediction: A Case Study, In: L.Dekker, W.Smit, and J.C. Zuidervaart, eds., *Massively Parallel Processing Applications & Development*, pp. 617-624, North Holland, 1994.
- [10] B. Qin, H.A. Sholl, and R.A. Ammar, Micro Time Cost Analysis of Parallel Computations, *IEEE Trans on Computers*, Vol. 40(5), 1991.
- [11] D.A. Reed, R.A. Aydt, et. al., Scalable Performance Analysis: The Pablo Analysis Environment, In: *Proc. Scalable Parallel Libraries Conf.*, IEEE Computer Society, 1993.
- [12] C.U. Smith, *Performance Engineering of Software Systems*, Addison Wesley, 1990.