

IS PREDICTIVE TRACING TOO LATE FOR HPC USERS?¹

Darren J. Kerbyson, Efstathios Papaefstathiou, John S. Harper,
Stuart C. Perry, and Graham R. Nudd

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

INTRODUCTION

An underlying goal in the use of high performance systems is to apply the complex resources to achieve rapid application execution times. It is often the case that performance issues are considered late in the application development when major design choices and system choices have already been finalised. Performance tuning tools, including parallel monitoring environments, are useful in these late stages providing a means in which to investigate and visualise the performance effects. However, during the development of an application, certain issues are typically decided upon without reference to their impact on performance (e.g. in the choice of a numerical implementation, or in the choice of a possible mapping to the system). There is a clear need for the study of performance at each and every stage of the development of high performance applications.

The aim of a novel prediction toolset PACE (Performance Analysis and Characterisation Environment), presented here, is to extend the traditional use of performance prediction to cover the full software development cycle. It incorporates facilities for both pre-, and post-, implementation analysis thus allowing alternatives to be explored prior to the commitment of an application (and its mapping) to a system, and also assists in the performance tuning of existing implementations. The approach is carefully structured consisting of modular performance models that reflect individual parts of the whole system (e.g. software components, parallelisation components, and system components).

¹ in High Performance Computing, Kluwer Academic, March 1999, pp. 57-67.

In this work, it is shown how PACE can be used to produce predictive traces representing the expected execution behaviour of an application given appropriate workload information. Predictive traces are analogous to traces collected at run-time except that the two key issues of: timing information, and event ordering information; are both determined by the prediction toolset. The trace data can be output from PACE in a commonly accepted format (such as PICL and SDDF) for use in existing performance monitoring environments (including Paragraph⁵ and Pablo¹²). Thus, predicted application execution can be viewed and examined within monitoring tools, already familiar to users, in order to identify performance hot-spots before system use.

A discussion of the various forms of performance analysis typically undertaken is given in the next section. The formation of prediction traces from the modular performance prediction toolset, PACE, is then described which illustrates the workload information required in a performance study, along with details on the performance model evaluation procedure to produce predictive traces. Example use of the resultant predictive traces is illustrated utilising two existing parallel performance monitoring environments on an example Financial Option code. The use of the predictive traces can significantly increase efficiency in final implementations when used during the development of application code.

PERFORMANCE PREDICTIONS

There are many performance issues that should be addressed during the development of application code which are not just confined to post-implementation ‘tuning’ analysis. Performance models can be constructed and used for predictive type analysis through-out. The models can range from ‘back of the envelope’ type calculations (or complexity analysis), in early software analysis stages, to detailed models in design and implementation stages. The level of abstraction incorporated into these models generally increases towards implementation, and formalistic approaches such as petri-nets, or queuing networks, are often employed to represent the structure of the system^{1,2}.

It is generally acknowledged that there have been few attempts at tools that provide the use and development of performance models throughout the software life-cycle mimicking the software development itself^{3,4,13}. Instead, individual tools are often utilised depending upon the stage of application development. These tools do however, have one thing in common - that is to provide the user with an estimation of execution time for a particular software formulation on a given target platform. Issues that are important in such performance studies include:

Execution time: a prediction of the time to execute the application given a set of application (e.g. problem size) and system parameters (e.g. number of nodes).

Scalability: how an application’s performance changes by the increasing of either the application and/or system parameters.

Sizing: determining the size of application that can be processed given constraints on either execution time and/or system resources

Such performance information is important in determining the time of application execution on a system but does not provide sufficient insight into the achieved performance

to enable refinement (tuning) of the application in a predictive sense. Further information on the predicted application execution is required in order to promote this process. The understanding of the system further depends upon how information is presented to the user. The visual representation of complex phenomenon aids in the performance understanding and behaviour of the target system. A key performance visualisation from a user's perspective is in the provision of space-time information (see Figure 9 for an example). This, a two dimensional chart, can be used to visualise processor activity (space) as a function of time.

The use of a space-time diagram follows directly from the generation of a trace file at the run-time of an application (after application porting and implementation). The trace file collected in this way is simply a list of events, that occurred during the execution of the application, on and between resources in the system.

Predictive traces, on the other hand, is a list of the same type of events but generated from the use of a suitable performance prediction system. They are analogous to execution traces and can be used and manipulated within the same monitoring performance analysis environments, Figure 1. Thus, performance refinement or tuning, can take place in advance of the porting (or even implementation) of an application to a target system.

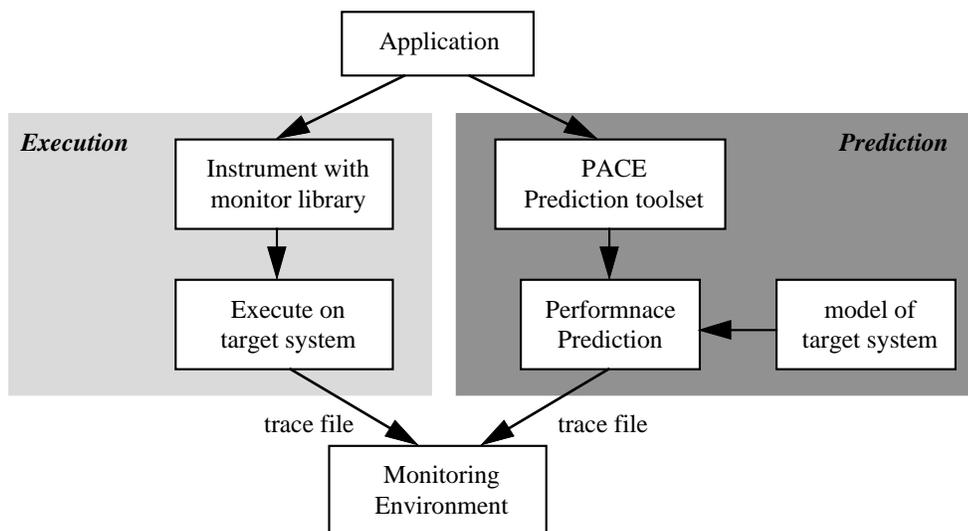


Figure 1. Analogous behaviour between execution and predictive traces.

There are three main features that need to be considered in the generation of predictive traces from a performance prediction system, namely:

Events: The information recorded during the predicted execution (e.g. inter-processor communication, processor idle periods, I/O etc.). If the predictive trace is to accurately portray its run-time counterpart then there should be a one-one mapping between the types of events possible.

Time stamping: Each event has an associated time-stamp to indicate when it occurred. This is simply the processor clock in a run-time system but in a prediction system each time-stamp needs to be individually calculated.

Ordering of events: This follows directly from accurate time-stamping to produce a chronologically ordered list of events.

Most performance prediction tools that have been developed to date have limited analysis capabilities, concentrating on producing overall estimates⁶. In the PACE toolset, described below, it is shown how suitable workload information, concerning both the computational aspects of an application, and their mapping to system resources, can be used in order to provide both realistic performance estimates, and sufficient activity information of the system to provide predictive tracing outputs.

THE PACE TOOLSET

The PACE toolset contains a number of individual components that are used to provide realistic performance predictions of expected application execution which takes into account the operation of the system as a whole. It considers detailed information from the application in terms of its computational cores, its mapping onto a high performance system (including necessary communication costs), and also time costs in terms of the underlying system performance characteristics. This three level structure is a modular approach whereby experimentation on factors such as the choice of the target platform, and also the mapping (parallelisation) that should be used, can be made using a criteria such as that to obtain best application performance.

The PACE toolset is comprehensive in its approach resulting from a period of extensive development, and its use in many different application areas^{8,9,10}. In this work, we illustrate the formation of a predictive trace file from PACE when using information from an application code (using a code segment taken from a Financial Option code), combined with predicted time costs on the target platform.

Workload Descriptions

Underlying the PACE toolset is a performance language, CHIP³S (Characterisation Instrumentation for Performance Prediction of Parallel Systems)¹¹, which provides the necessary syntax and semantics to support workload descriptions for parallel software. This includes: computational control flow information, resource usage information, mapping and communication information amongst others. The core components of PACE is shown in Figure 2.

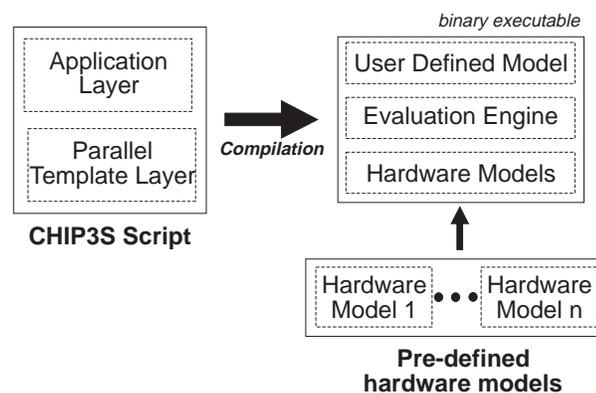


Figure 2. The core components of the PACE toolset.

The compilation of the CHIP³S performance scripts results in output binaries which can be linked to an evaluation engine along with hardware models for the target system. The evaluation engine combines the workload descriptions with the appropriate platform performance characteristics (encapsulated in the hardware models) to produce performance predictions (see below).

The hardware models for a specific system is a combination of: measurements (e.g. micro-benchmark results), models (e.g. statistical, and analytical), and hardware specifications. The hardware components of a system can be modelled at different levels of abstractions depending on the accuracy of predictions required. Models that represent a low-level of abstraction typically provide highly accurate predictions but require detailed workload information and result with long evaluation times.

The workload definitions in CHIP³S are organised into two layers: an application layer, and a parallel template layer. The former includes the workload descriptions for the computational parts of the application while the latter includes descriptions on how the resources of the systems are to be utilised and will interact. CHIP³S is dedicated to provide the necessary syntax to support these descriptions. In order to illustrate these descriptions, a code segment taken from the Alternating Direction Implicit (ADI) solution to a Partial Differential Equation, as shown in Figure 3 will be used. This is written in an SPMD (Single Program multiple data) style parallelism using the PVM message passing interface. Note that the code in Figure 3 is performed on all nodes in the system, and contains communication dependencies between nodes.

```

pvm_recv (tids[MY_ID-1], 30);
pvm_upkfloat (&d[i][MY_MIN_J-1], 1, 1);

for (j = 0; j <= MY_SIZE; j++) {
    if ( (j >= 2) && (j != SIZE_J) ) {
        fact = a/b[j-1];
        d[i][j] = d[i][j] - fact*d[i][j-1];
    }
}

pvm_initsend (PvmDataRaw);
pvm_pkfloat (&d[i][MY_MAX_J], 1, 1);
pvm_send (tids[MY_ID+1], 30);

```

Figure 3. Example C code taken from a Partial Differential Equation solution for a Financial Option.

The CHIP³S description for the computational part of the ADI code is shown in Figure 4. The computational control flow, and associated operations, are described in the application layer with the use of *cflow* procedures. These are similar to the procedures in the source code but encapsulate the control flow of the application. Each statement in this script is associated with a *Processor Resource Usage Vector* (PRUV) indicated by the brackets < ... > that represent the operations in the original source code. In this example the operation count is done in terms of input C language operations (clc) with each operation indicated by a four character code. Control flow statements include: *compute*, *loop*, and *case* (conditional execution statements). Note that the case statement has a probability of executing the branch.

```

proc cflow TxEliminate {
    loop ( < is clc, LFOR, CMLL, INLL>, MY_SIZE) {
        compute < is clc, 2*CMLL, ANDL>;
        case ( < is clc, IFBR>) {
            0.9: compute <is clc, ARF1, 3*ARF2, DFSL, 2*TFSL, MFSL,AFSL>;
        }
    }
}

```

Figure 4. An example computational description in CHIP³S contained in the application layer.

The parallel template layer makes reference to the resources used in the system to execute the application. The term *device* is used to refer to the use of any hardware, or software, resources such as hardware devices (CPU, interconnection network, I/O) and message passing libraries (e.g. packing, initialisation). A CHIP³S parallel template is shown in Figure 5 which represents the structure of the original ADI source code in Figure 3. In each `step` statement, a device is specified, e.g. `cpu`, `pvmrecv` (asynchronous receive using PVM) etc. The parameters to each step are specified using the `confdev` statement. For example, in a communication receive and send, the parameters indicate the source and destination processors respectively. It should be noted that the parameter for the `cpu` step, is the value predicted from the `TxEliminate` function as defined in the application layer (Figure 4).

```

step pvmrecv
step pvmunpack on my_id
step cpu on my_id
step pvminitsend on my_id
step pvmpack on my_id
step pvmsend
    { confdev my_id - 1, my_id; }
    { confdev 1, PVM_FLOAT; }
    { confdev TxEliminate; }
    { confdev PVM_DataRaw; }
    { confdev 1, PVM_FLOAT; }
    { confdev my_id, my_id + 1; }

```

Figure 5. An example parallel template description in CHIP³S.

Model Evaluation

The process of evaluating a PACE model, using the CHIP³S performance scripts, and producing performance predictions is undertaken by the evaluation engine detailed in Figure 6. Initially the application layer is evaluated to produce predicted computational workload information. These are then used in the parallel template step `cpu` devices

The individual call to specific devices, and their associated parameters, are passed into the evaluation engine. A dispatcher distributes input, from the workload descriptions, to an event handler and then to the necessary individual hardware models. The event handler is responsible for the construction of an event list for each processor in the system. Although the events can be resolved by the device involved in the step statement, the time spent using the device is still unknown. However, each individual hardware model can produce a time prediction of 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.

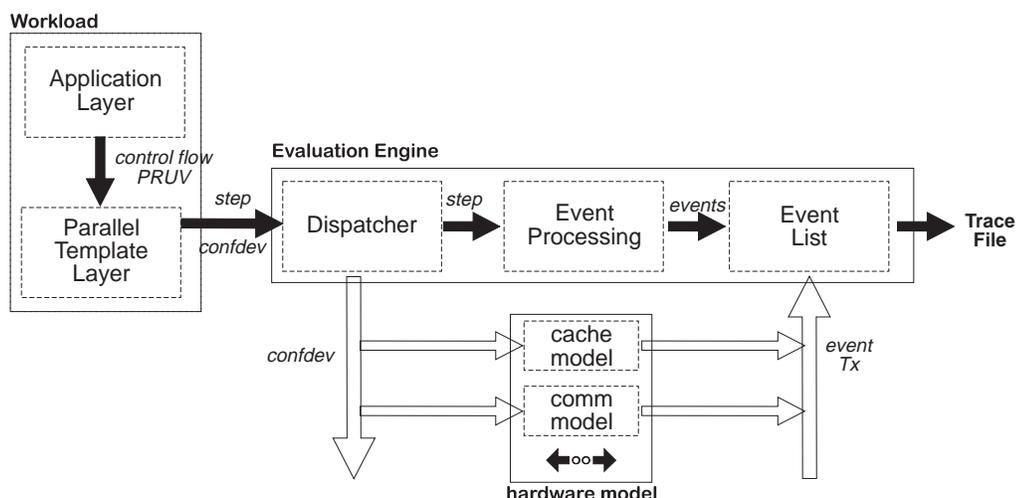


Figure 6. The evaluation process to produce a predictive trace within PACE

The processing of the event list is a two stage operation. The first is in the construction of the events, and the second is to resolve any ordering dependencies taking into account any contention factors. For example, in predicting the time for a communication, the traffic on the inter-connection network should be known to calculate channel contention. In addition, messages obviously 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 since it does not require interaction with any other events.

Predictive Traces

The ability of PACE to produce predictive traces derives directly from the event list formed during the model evaluation. An example event list produced by the evaluation engine is illustrated in Figure 7 using a space-time representation. The communication dependencies between three processors, using the ADI code segment, are shown. The events, illustrated graphically, correspond to the underlying records in the trace file.

The predictive events produced by the evaluation engine are PACE specific in comparison to general traced events. However, a mapping of these to events to a standard trace file format, recognised by an external monitoring environment, is a straight forward process. Example trace events supported by PACE are shown in Table 1. This is an extendible list in which events can be used to represent any information contained within the PACE model. The traces are produced by scanning each processor event list sequentially. During this process the local events are mapped to output trace events and formatted in a standard trace format. The output trace file is sorted according to the event time-stamps.

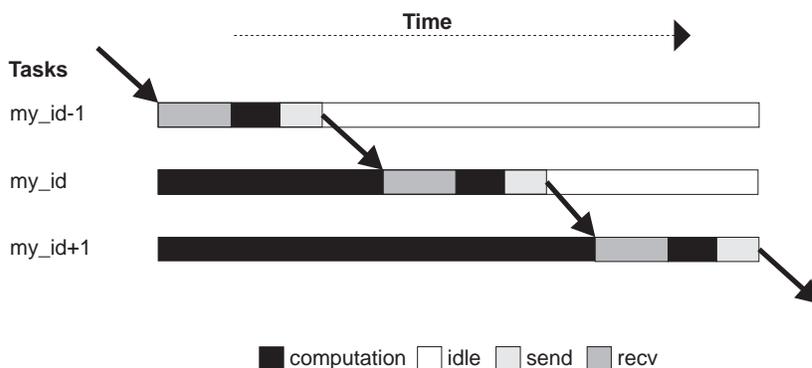


Figure 7. Example event list produced by the evaluation engine

Table 1. Example predictive events used within the PACE Evaluation Engine.

Event (start)	Event (end)	Description
SendBegin	SendEnd	Surrounds an asynchronous message send.
RecvBegin	RecvEnd	Surrounds a message being received.
RecvBlockBegin	RecvBlockEnd	A blocked receive (i.e. processor waiting).
CompBegin	CompEnd	A computation event
Overheadbegin	OverheadEnd	Computation events associated with parallel overheads
TaskBegin	TaskEnd	Surrounds a computational subtask on each processor.

VISUALISING PREDICTIVE TRACES

PACE can produce overall predictions of execution time, and scaling behaviour, of application code. An example of which is shown in Figure 8 for a monte-carlo simulation code. However, the significant feature of PACE in the generation of a predictive trace allows insight into the time estimation. The predictive traces produced by PACE can be output in one of two formats suitable for use with either the Paragraph⁵, or the Pablo environment¹². They are also quite different in their approach in forming a performance analysis environment. For example, one of Paragraph's main use is to effectively 'replay' a trace file using a number of built-in performance displays which extract appropriate events directly. This is in comparison to Pablo which requires an analysis session to be constructed in a graphical environment first, followed by the use of a number of built-in displays. In this graphical environment, the user effectively relates events from the input trace file to the different displays available.

The two monitoring environments quite different trace file formats. Paragraph uses the PICL¹⁴ (Portable Instrumented Communication Library) format in which each line in the trace file represents an individual event using a pre-specific data coding, thus allowing only certain events to be recorded. Pablo uses the SDDF (Self Defining Data Format) which enables the format of each event to be specified in the trace file along with the actual list of events. This is a more flexible approach, allowing system/application specific events to be defined and recorded, but requires trace-file specific construction of a performance analysis session.

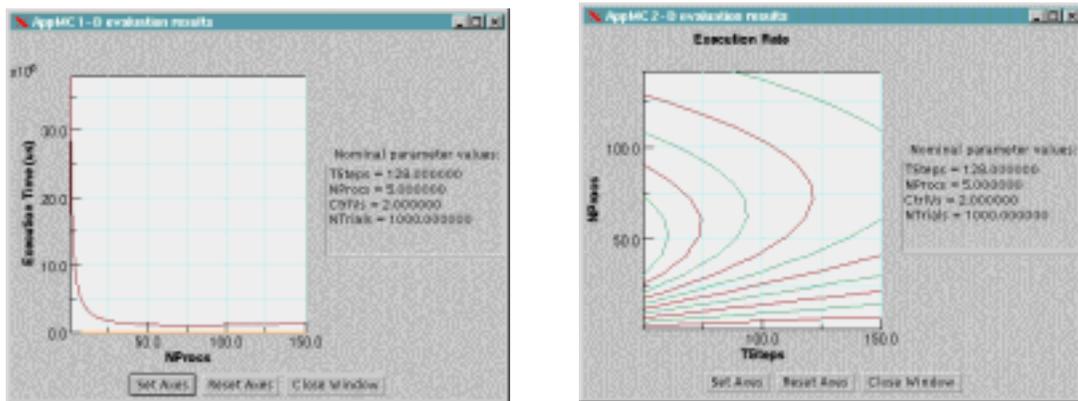


Figure 8. Predicted scaling behaviour of a monte-carlo simulation code as output by PACE

In order to illustrate the use of the predictive trace output from PACE, the core of a financial option code is used, a segment of which was shown in Figure 4. This code requires the solution of a partial differential equation, using the Alternating Direction Implicit (ADI) solution on a dense data grid. The resulting parallel code is a data decomposition of the data grid, but results in much communication between processing nodes.

This code was modelled within PACE, resulting in a binary which, when executed produced performance predictions for the application given a set of input parameters (data grid sizes, and also processor nodes in the system). The example here considered a hardware platform of a cluster of 6 SUN ULTRA workstations. A predictive trace (in either the PICL or SDDF format) can be output from this executable on the setting of an input flag. The overall performance predictions were validated with application run-times, and were found to

be within an error bound of 15%. The performance predictions were obtained in a matter of seconds from the binary executable. The speed of evaluation is a significant feature of PACE, not described in detail here, but has been used for on-the-fly performance prediction⁸.

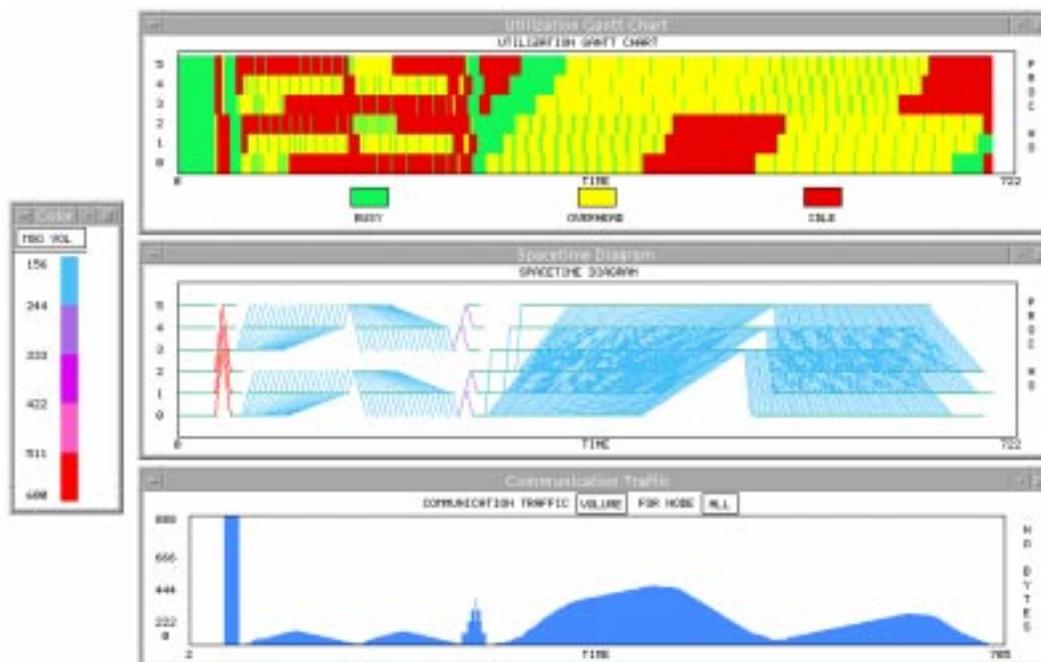


Figure 9. Example of a prediction trace analysed using space-time diagrams (from Paragraph).

Figure 9 shows three example views, derived from Paragraph, illustrating system activity as a function of time, for part of the predicted behaviour of the Financial code. The top chart indicates processor status, either busy, idle, or involved in parallel overheads. The second chart shows the computation/communication activities, colour coded according to message volumes. The final chart indicates the traffic in the system - an effective summation across processors from the middle chart. It can be seen, that there are many regions of both idle and overhead time which may be significant in under-achieving performance. These factors, once identified, could potentially be refined prior to application implementation.

Figure 10 shows an example predictive trace analysis session within Pablo. In the background, an analysis tree refers to 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 behaviour of the Financial code. 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’ the size and colour of which indicates traffic.

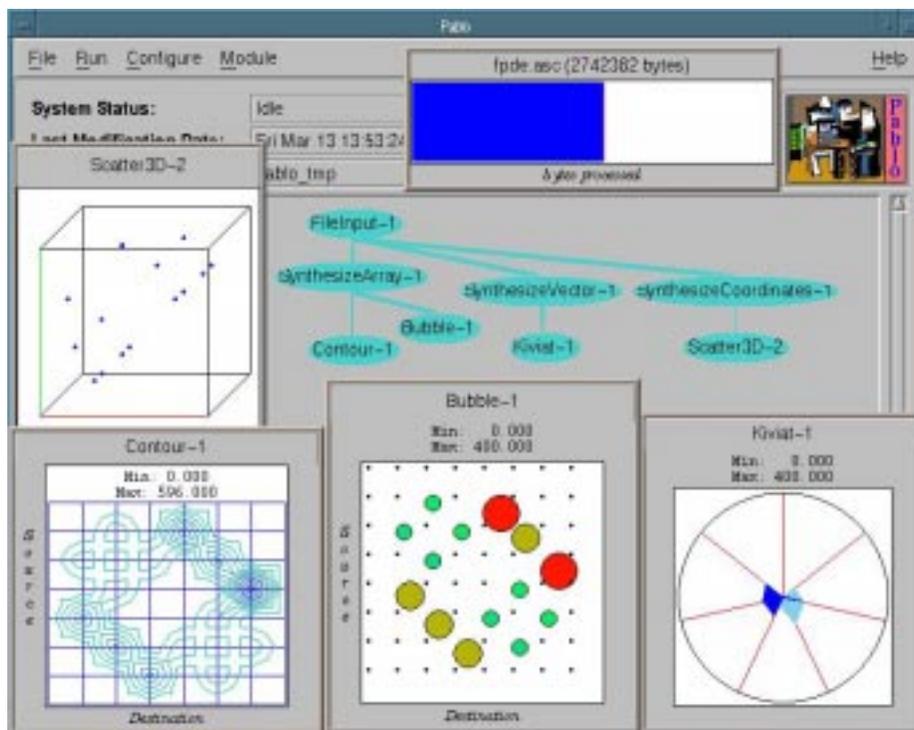


Figure 10. Example of a prediction trace analysed using the Pablo performance analysis environment.

The use of a monitoring environment is not just limited to the analysis of a single trace. Indeed, current work is underway on the identification, and quantification, of changes in performance between one trace and another⁸. Such perturbations in performance may be due to variability in system use, or possibly due to changes in underlying system software (e.g. message passing libraries). The use of PACE can foresee changes in achieved performance, when such system parameters change, but in a predictive sense.

SUMMARY

The development of high performance application code is driven by the need of achieving rapid execution times. At a certain point in the development of the application, the performance issues become the predominant concern. However, typically these concerns are left until after the porting and implementation of the code, with performance analysis relying solely on analysis of executions. Performance analysis is not restricted to this post-implementation analysis, but can (and should) be used during the development of the application using prediction studies. The PACE toolset, as presented here, provides facilities for performance prediction studies to take place. Such studies can aid the implementation and porting of application code by choosing underlying numerical techniques, or guiding the choice of parallelisation, based on expected performance that will be achieved when finally executing the application on the target platform.

A significant feature of PACE is in the provision of a predictive trace output. This, a trace file that contains a list of events that represents the expected behaviour of the system at run-time. The predictive trace mimics the generation of an execution trace (one which is collected at run-time), and can be manipulated in exactly the same way. Thus, available monitoring environments, such as Paragraph and Pablo (familiar to many HPC users) can be

utilised to analyse and explore the expected performance of the application code. The use of PACE in outputting predictive traces has been illustrated on a financial option code in this work. The use of the predictive traces can significantly increase efficiency in final implementations when used during the development of application code.

ACKNOWLEDGEMENTS

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

REFERENCES

1. T. Fahringer, Estimating and Optimizing Performance for Parallel Programs, *IEEE Computer*, Vol. 28(11), pp. 47-56 (1995).
2. A. Fercha, A Petri Net Approach for Performance Oriented Parallel Program Design, *Jnl. of Parallel and Distributed Computing*, Vol. 15(3), pp. (1992)
3. D.G. Green, C.J. Scott, A. Colbrook, and M. Surridge, HPCN tools: a European perspective, *IEEE Concurrency*, Vol. 5(3), pp. 38-43 (1997).
4. I. Gorton and I.E. Jelly, Software engineering for parallel and distributed systems, challenges and opportunities, *IEEE Concurrency*, Vol. 5(3), pp. 12-15 (1997).
5. M.T. Heath, A.D. Malony and D.T. Rover, The visual display of parallel performance data, *IEEE Computer*, Vol. 28(11), pp. 21-28 (1995).
6. T. Hey, A. Dunlop, and E. Hernandez, Realistic parallel performance estimation, *Parallel Computing*, Vol. 23, pp. 5-21 (1997).
7. K.L. Karavanic and B.P. Miller, Experiment management support for performance tuning, in: *Proc. SuperComputing* (1997).
8. D.J. Kerbyson, E. Papaefstathiou and G.R. Nudd, Application execution steering using on-the-fly performance prediction, in: *High Performance Computing and Networking*, Springer-Verlag, (1998).
9. E. Papaefstathiou, D.J. Kerbyson, G.R. Nudd and T.J. Atherton, An overview of the CHIP³S performance prediction toolset for parallel systems, in *Proc. of 8th ISCA Int. Conf. on Parallel and Distributed Computing Systems*, pp. 527-533 (1995).
10. E. Papaefstathiou, D. J. Kerbyson, G.R. Nudd, A layered approach to parallel software performance prediction: a case study, in: *Massively Parallel Processing Applications & Development*, L. Dekker, W. Smit, and J.C. Zuidervaart, eds., North-Holland, pp. 617-624 (1994).
11. E. Papaefstathiou, D.J. Kerbyson, G.R. Nudd and T.J. Atherton, An introduction to the CHIP³S language for characterising parallel systems in performance studies, Research Report RR280, Dep. of Computer Science, University of Warwick (1995).
12. C.U. Smith, *Performance Engineering of Software Systems*, Addison Wesley (1990).
13. D. A. Reed et. al., Scalable performance analysis: the Pablo performance analysis environment, in *Proc. Scalable parallel libraries conf.*, IEEE Press, pp. 104-113, (1993).
14. P.H. Worley, A new PICL trace file format, ORNL/TM-12125, Oak Ridge National Laboratory (1992)