

Application Execution Steering Using On-the-fly Performance Prediction¹

D.J. Kerbyson, E. Papaefstathiou, G.R. Nudd

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

Abstract

The execution of an application on a high performance system requires parameters concerning the problem in hand, and those that determine the system mapping, to be specified by a user. The system parameters are typically used to minimise the execution time. However, by the coupling of a performance model with an application, system parameters can be determined without user intervention. In the work presented here, a novel performance prediction system has been used to provide suitable performance models which can determine application mapping parameters, code execution decisions, and system choices *on-the-fly*. An example compact application of a convolution is used to illustrate the approach for automatically choosing the actual code to be executed, and the number of workstations in a cluster to be utilised. The performance prediction system is shown to have a reasonable accuracy (approximately 10% error), with a rapid evaluation time (typically < 2s).

Keywords: High Performance Computing, Performance Prediction, On-the-fly decision making, application steering.

1 Introduction

An underlying goal in the use of high performance systems is to apply the often quite complex systems to achieve rapid application execution times. The analysis of performance is currently an active research area. Traditionally performance prediction has been used prior to application implementation to explore alternative implementations e.g. different parallelisations, and also in analysing existing codes in order to assess the impact of increasing resources, and also to identify bottlenecks.

In the novel system utilised here, the traditional use of performance prediction is extended to encapsulate the full software cycle. It incorporates both pre- and post implementation analysis but also importantly it can be used to aid the actual execution of an application. Thus, utilising performance prediction to effectively steer the application execution onto an available system at run-time in an efficient manner. The prediction system provides detailed information on the expected application execution which is dependent on application parameters as well as current system status - which can only be resolved at run-time.

¹ in High Performance Computing and Networking, Lecture Notes in Computer Science, Vol. 1401, Springer-Verlag, April 1998, pp. 718-727.

Several other systems have been proposed which incorporate performance information to aid application execution e.g. [2,3,10] which have been applied for scheduling in Meta-computing systems. However, the performance information is usually extracted from previous timing information, and is not based directly on prediction. Thus, they have limited use for the application steering on changes in system configuration, and also in changing parallelisation methods. Performance prediction has not been used in such situations to date due mainly to in-accurate modelling of real-systems, and also to a slow speed of evaluation. The system used here attempts to tackle both of these issues which results in accurate performance models which can be evaluated quickly.

This paper concentrates on the use of on-the-fly use of performance prediction, the concepts of which are described in Section 2. An overview of the prediction system (PACE), which is under development at Warwick, is described in Section 3. This system enables a performance model to be incorporated into an application which can be used to steer its execution. The system enables rapid performance predictions to be calculated. A compact application containing a convolution kernel is used to illustrate the approach for determining system mapping, and calculation method, in Section 4.

2 On-the-fly 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, which can be broadly classed into two categories:

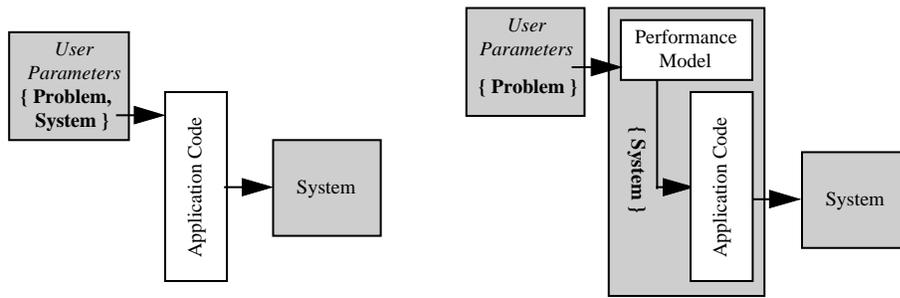
Problem parameters include those that specify the format of the data to be processed e.g. data size, and also the type of result required such as accuracy in a numerical calculation, or the number of iterations required in an iterative solution.

System parameters 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 how the application will be mapped onto the available system, and are normally used to reduce execution time. By coupling a suitable performance model into the application, predictions can be made to determine these system parameters using criteria such as requiring a minimum execution time. 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. Figure 1 illustrates both a user directed execution of an application and a performance directed approach.

For example, when determining the number of processors to be used in the application execution, a performance model can be used to evaluate a best apriori estimate based on the application and system performance characteristics. However, in a user-directed approach, the number of processors specified by the user may

naively be the maximum available or at best use pre-acquired knowledge from previous application runs but not necessarily the optimum. Thus, the performance model replaces what is quite often an ad-hoc decision procedure.



(a) User directed approach (b) Performance directed approach
Figure 1 - Execution of an application using problem and system parameters

The use of the performance model is not limited to just specifying the number of processors to be used. Consider the situation depicted in Figure 2 where an application has two methods of implementation (perhaps two ways in which a numerical solution can be calculated). In addition, two systems are available for its execution. In this scenario, the user specifies the problem parameters, but also decides on the code to be used and the target system along with relevant system parameters.

The use of a suitable performance model in this example can determine the code to be executed, on which target system, along with 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. Figure 3 illustrates this situation, and shows the chosen code and system (solid arrows), and other available choices (dotted arrows).

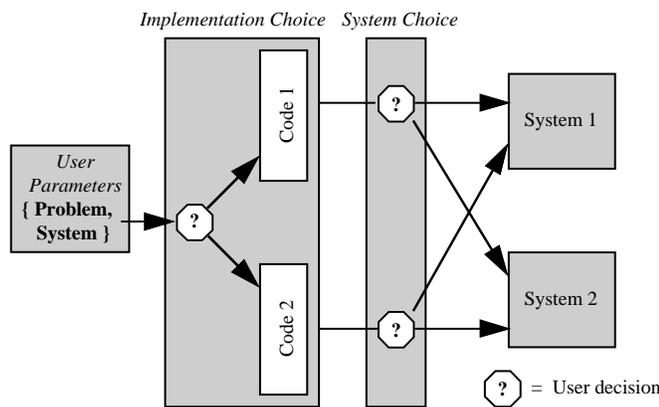


Figure 2 - User directed execution of an application having multiple implementations and available systems.

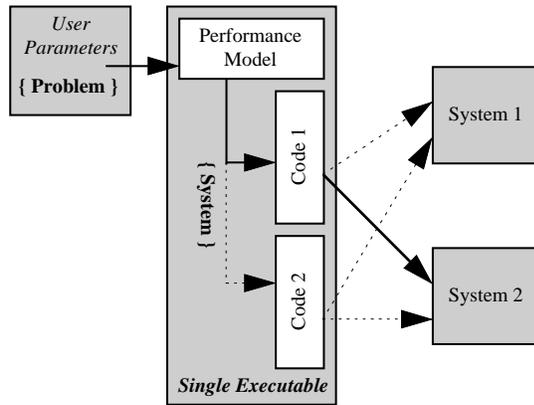


Figure 3 - Performance directed execution of an application with multiple implementations and available systems.

The performance model can be used to automatically determine a number of parameters based on a performance criteria, including:

Mapping: including the number of processors to be used

Implementation: method of solution when several are available, or choice of implementation if several are available (e.g. different parallelisations)

System choice: when multiple systems are available

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. Such a system is currently being explored using a Genetic Algorithm for the Scheduler implementation.

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 systems. The best scenario is taken to be that which results in a minimum predicted execution time. The total number of scenarios required to be evaluated is a product of the number of mappings, number of implementations, and number of systems. Thus, rapid performance model evaluation time is required.

3 The PACE Performance Prediction System

PACE (Performance Analysis and Characterisation Environment) is a performance prediction toolset suitable for a non-performance expert [6,7]. PACE supports the development of performance prediction models for sequential and parallel applications running on high performance systems. It is based on a layered

characterisation methodology [8], and is an analytical based approach that organises a performance model into three separate layers: an application layer (includes the computation parts), a hardware layer (includes hardware models), and a parallelisation layer (includes communication patterns).

PACE supports the entire software lifecycle including development, execution, and post-mortem performance analysis. A full description of PACE is out of the scope of this paper [6,7]. However, the PACE components utilised for the on-the-fly prediction are shown in Figure 4.

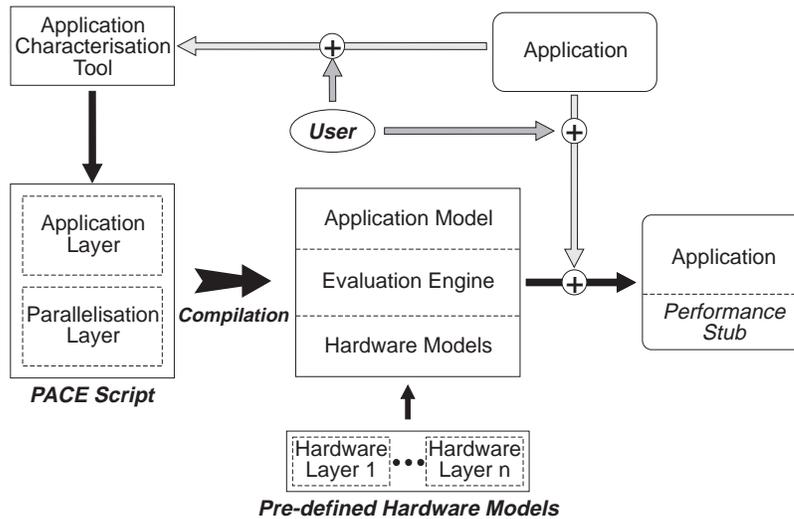


Figure 4 - Constructing a Performance Stub using PACE

The main part of PACE is a special purpose performance language which describe the control flow of the application, the computational resource requirements, and the parallelisation communication patterns. The language scripts can be produced either manually by the user, or by using tool support within PACE. The scripts are then converted into a performance model by the PACE compiler. The output of the compiler is an executable performance stub that includes: the application models, pre-defined hardware models, and an evaluation engine to produce performance predictions.

In the process of producing a performance model for on-the-fly steering, it is assumed that the parallel application source code is available. PACE includes an *Application Characterisation Tool* (ACT) that aids the conversion of source code into PACE scripts via the Stanford Intermediate Format (SUIF) [5]. ACT performs a static analysis of the code to produce the control flow of the application, operation counts in terms of high level language operations (C or FORTRAN) [9], and also the communication patterns. This is a semi-automated process as ACT requires user

intervention to resolve dynamic constructs (e.g. data dependent branching probabilities and loop iteration counts).

The PACE scripts when compiled, allow linkages to the application code. The user can modify the application code to utilise the performance model including the setting of problem sizes, system configuration parameters and also the invocation of the model for on-the-fly steering.

4 Application Case Study

Incorporating a performance stub (from PACE) to enable on-the-fly performance prediction is illustrated below for an example compact image processing convolution application which is common in many computationally demanding applications (e.g. Image microscopy [1]). It also has multiple methods which can be used to calculate the resultant data. Thus, when a high performance system is available, both the number of resources, and the calculation method have to be decided upon solely by the means of the performance stub. The choice of system and calculation is determined using both the user specifiable problem parameters, and the knowledge of the currently available resources.

4.1 Application description

A two-dimensional convolution operation can be expressed as: $F(i,j) = W * I(i,j)$ where $I()$ is the input image, $F()$ the output image, $*$ is the convolution operator, W is the convolution weighting function (kernel), and (i, j) indicates the data element. Each output element is a function of a windowed region of the input data weighted by the kernel. The two-dimensional convolution can be written as:

$$F(i, j) = \sum_{p=-R}^R \sum_{q=-S}^S W(p, q) * I(i - p, j - q)$$

where R and S define the size of the rectangular kernel.

The convolution can be calculated either in the spatial domain or in the frequency domain. A spatial convolution is simply a set of multiply-accumulate operations with sequential complexity proportional to M^2N^2 (where N^2 is the size of the input image, and M^2 is an assumed square kernel). A convolution in the frequency domain requires the transformation of the input image, and kernel, using a Fast Fourier Transform (FFT) followed by a point-wise multiplication. The sequential complexity of this calculation is proportional to $N^3 \log_2 N$ (due to the two dimensional FFT) and is independent of kernel size. The choice of processing is shown in Figure 5.

The size of the image ranges upwards from 256^2 pixels, and the convolution kernel considered ranges upwards from 13^2 elements. A typical processing scenario may be on a 1024^2 image using a 31^2 convolution kernel.

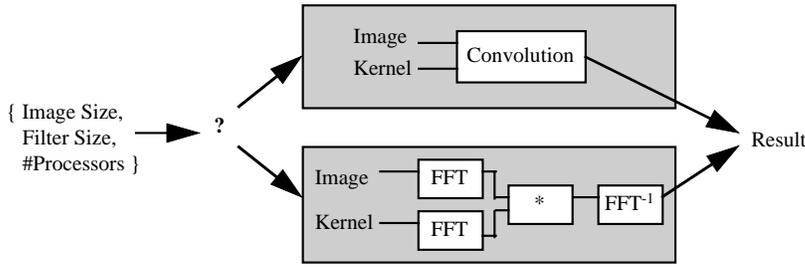


Figure 5 - The algorithmic choice available in the convolution operation.

Associated with the choice of processing is the mapping onto the available resources. In this example, it is assumed that a cluster of 32 Sun SparcStations are available for use as a parallel resource, using the PVM message passing interface [4].

4.2 Model Validation

A performance stub was constructed using PACE for both forms of convolution calculation. A validation procedure was carried out that samples the correctness of the model for a number of problem sizes and processor configurations. A comparison of the measurements from the machines in the Sun Workstation Cluster, and the predictions from the performance model is shown in Tables 1 and 2. It should be noted that the execution time for the frequency domain calculation is independent of the convolution kernel size.

		Number of Processors							
		2		4		8		16	
Image	Filter	Meas.	Pred.	Meas.	Pred.	Meas.	Pred.	Meas.	Pred.
256	15	30.31	29.73	17.51	15.29	9.68	8.21	6.55	4.96
	31	120.23	125.54	62.56	63.57	32.44	32.72	18.76	17.59
512	15	121.46	117.70	64.11	59.53	35.55	30.58	22.78	16.4
	31	561.05	540.18	254.05	251.51	136.31	127.32	68.94	65.51
1024	15	484.58	469.38	249.41	236.15	137.24	119.67	79.84	61.73
	31	2010.2	1997.7	1023.9	1001.8	518.54	504.00	269.10	255.38

Table 1 - Comparison of Measurements and Predictions for the spatial convolution (sec).

		Number of Processors							
		2		4		8		16	
Image		Meas.	Pred.	Meas.	Pred.	Meas.	Pred.	Meas.	Pred.
256		35.78	45.30	44.97	50.67	58.24	60.18	70.43	64.59
512		122.35	123.20	134.44	144.70	147.43	159.33	173.36	177.87
1024		841.54	909.85	823.10	780.67	845.76	822.77	870.76	847.89

Table 2 - Comparison of Measurements and Predictions for the frequency convolution (sec)

The comparisons indicate that the PACE performance stub conforms to the measurements within an acceptable error bound (typically < 10%). The system is assumed to be quiet during the measurement period and hence the performance model is valid for use during a quiet period on the target system. Incorporation of dynamic system loading into the PACE performance models, and its effects on the execution time, is currently in progress.

4.3 Use of the Performance stub

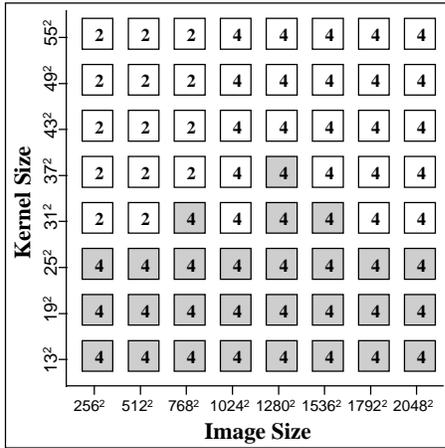
A single application executable was formed by the linking of both forms of the convolution, and the performance stub. The performance stub was thus used to direct the execution of the application both in terms of the choice of the calculation method, and also in the choice of the mapping onto the available system.

A summary of the use of the performance stub is shown in the four graphs in Figure 6. The user specifiable problem parameters are shown on the axis of the graphs, and the parameters determined by the model are shown as discrete numeric values (number of processors), and by shading (grey indicates a spatial convolution, and white indicates a frequency convolution). The four graphs differ only in the assumed size of the available system (either 4, 8, 16, or 32 SUN SparcStation 2's in this analysis).

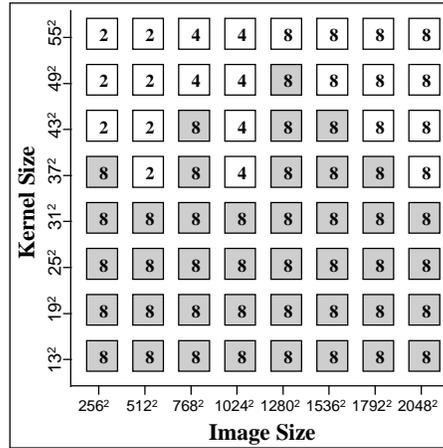
It can be seen, from Figure 6, that the spatial convolution method is chosen over the frequency method for small filter sizes across the range in image sizes and available systems. Indeed, the spatial method scales readily on a parallel system, and becomes the preferred method as both the system size increases. The frequency method is preferable for larger kernel sizes on smaller system sizes (due to communication cost scaling with the number of processors).

It should be noted that when the performance stub is used on-the-fly, only a limited number of different scenarios need be evaluated to determine the mapping and algorithm choice. The number of different scenarios is simply a multiple of the number of algorithmic choices, and the number of system configurations possible.

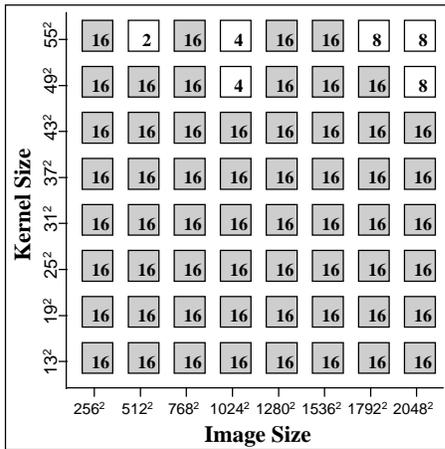
The advantage in using performance stubs on-the-fly is clear in that the user no longer needs to be concerned with the system use, but instead these decisions are made using information available within the model. However, there is a cost of using the performance model which can be quantified in terms of the additional time required to provide the decision making procedure, and also its effect in the increase of executable size. The additional time is small - measured to be less than 2 seconds for all scenarios depicted in the results above thus adding very little perturbation to the total application execution time. The executable size is constituted from the two different calculation forms, and the performance model. This is larger than an individual method's executable but does not effect overall processing time (other than in initial loading).



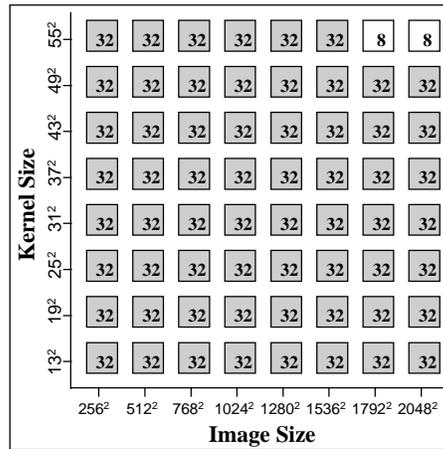
(a) maximum of four workstations



(b) maximum of eight workstations



(c) maximum of sixteen workstations



(d) maximum of thirty-two workstations

Figure 6 - Use of the performance stub for a range of problem and system sizes.

5 Summary

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 stub to be incorporated into an application executable and can be used in a decision making procedure to determine application system parameters. Accurate predictions can be produced, and rapid evaluation of the performance models enables on-the-fly use.

An example compact application of convolution was used to illustrate the approach. This showed the decision making procedures for determining the code to be used and

also its mapping on to the target platform of a workstation cluster. The PACE system is currently being extended to include dynamic loading effects, and also allow the mapping of applications onto heterogeneous processing systems.

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. D.A. Agard, Y. Hiraoka, P. Shaw, J.W. Sedat, Fluorescence Microscopy in Three Dimensions, in *Fluorescence Microscopy of Living Cells in Culture*, Elsevier, pp. 353-377, 1989
2. J.N.C. Arabe, A.B.B. Lowekamp, E. Saligman, M.Starkey, and P. Stephan, Dome Parallel programming environment in a heterogeneous multi-user environment, *Supercomputing*, 1995.
3. J. Gehring, A. Reinefeld, MARS - A framework for minizing the job execution time in a metacomputing environment, *Future Generation Computer Systems*, vol. 12, pp. 87-99, 1996
4. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, PVM - Parallel Virtual Machine, MIT Press., 1994
5. M.W. Hall, J.M. Anderson, et.al, Maximizing Multiprocessor Performance with the SUIF Compiler, *IEEE Computer*, Vol. 29 (12), December 1996
6. D.J. Kerbyson, E. Papaefstathiou, J.S. Harper, S.C. Perry, G.R. Nudd, Is Predictive Tracing Too Late for HPC Users?, in *High Performance Computing*, R.J. Allan, A. Simpson, D. A. Nicole (Eds), Plenum Press, 1998.
7. E. Papaefstathiou, D.J. Kerbyson, G.R. Nudd, 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, Orlando, 1995.
8. E. Papaefstathiou, D. J. Kerbyson, 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.
9. B. Quin, H.A. Scholl, R.A. Ammar, Micro Time Cosrt Analysis of Parallel Computation, *IEEE Trans. on Computers*, Vol. 40 (5), pp. 613-628, 1991.
10. R. Wolski, Dynamically Forecasting Network Performance Using the Network Weather Service, UCSD Technical Report, TR-CS96-494, 1996.