# A Layered Approach to the Characterisation of Parallel Systems for Performance Prediction

G.R. Nudd, E. Papaefstathiou, Y. Papay
T.J. Atherton, C.T. Clarke, D.J. Kerbyson, A. F. Stratton, R. Ziani, M.J. Zemerly

Department of Computer Science, University of Warwick,
Coventry CV4 7AL, U.K.
email: warwick_peps@dcs.warwick.ac.uk

## Abstract

An approach to the characterisation of parallel systems using a structured layered methodology is introduced here. This approach is based upon two similar techniques which have been suggested for use in the modelling of computer systems. Conventional modelling techniques rely on workload parameters obtained from an existing system and the development of a system model. For software under development this is a hindering factor for performance evaluation and performance prediction. Software Performance Engineering (SPE) offers a solution to this problem by using a software execution model, in addition to the system model. The use of SPE for parallel software has several disadvantages:□complex nature of the SPE system model, non re-usable□model components, and lack of analytical methods to evaluate the system model. A novel layered approach for characterisation is presented here, using separate hardware, parallel paradigm, and application layers to overcome these disadvantages. Although this method can be used in modelling, it is primarily targeted towards characterisation studies due to it's hardware independent nature. The layered approach is illustrated, and results obtained, using an image processing benchmark.

## 1. Introduction

The capabilities of parallel machines has exceeded the power of traditional mainframes and vector supercomputers over the last decade. Although the popularity of parallel machines has increased dramatically in the scientific and engineering community, it has not yet been able to make an impact upon the mainstream computing community. A number of factors has contributed to this including the huge investment already made in sequential software.

To make parallel computing more readily available and cost effective software technology must mature that requires libraries allowing the reusability of parallel programs, and tools for predicting, measuring, and enhancing performance [Chandy91].

Software performance prediction is being extensively researched [Miller90, Gabber90, Pease91]. Software Performance Engineering (SPE) is a recent methodology that incorporates software/system modelling and characterisation while focusing on software efficiency [Smith90]. SPE has been successfully used in many sequential and a few parallel software development projects [Smith82a, Smith82b, Weishar87]. It can be considered as one of the strongest candidates for software performance prediction studies.

This paper focuses on software modelling and characterisation. A method is proposed for the layered characterisation of parallel software which, in contrast to SPE, does not require time-consuming procedures for the development of models but focuses on model reusability. It uses techniques from SPE adjusted to the needs of parallel software characteristics. The methodology can be incorporated into a software characterisation tool that will allow the development of parallel software characteristics either based on a sequential version or on the design of a new parallel application. Experimentation can be done with various re-usable parallelisation techniques on different hardware without the need of further model development.

One of the main features of characterisation models is their independence to the underlying hardware, opposed to models derived from modelling studies. For example a model for a master-slave paradigm that derives from a characterisation study will be independent of the way that the paradigm maps onto a computer system. However, the generality of characterisation studies leads to lower accuracy of performance prediction than in modelling. Characterisation and modelling are considered complementary methodologies for predicting the performance of parallel systems. Although the main ideas behind the layered approach originate from modelling techniques, they are more appropriate for characterisation studies since they focuse on re-usability and model generality.

The layered characterisation techniques originated from work in model structural decomposition [Jain89, Patel92]. The main goal of structural decomposition is to sub-divide the model into simpler models and to integrate various modelling methodologies, e.g. queuing networks, petri nets etc. These sub-models are loosely coupled and can be re-used in further characterisation and modelling studies. Structural decomposition suggests the sub-division of the model into four basic sub-models: workload scheduling, workload definition, resource scheduling, and resource definition. The proposed structure of such models has been modified here in order to include parallel software characteristics and SPE components.

This paper is structured as follows: An introduction is given for the methodology followed in SPE in section 2. The disadvantages of the methodology are identified especially for the characterisation of parallel software. The layered approach is introduced in section 3. Each layer of this characterisation approach is analysed in detail in section 4 using a case study of an image processing benchmark. Finally, in section 5, an implementation is described and results obtained are compared with real measurements.

## 2. Software Performance Engineering

A conventional modelling process starts by an examination of the computer system. The next step is the construction of a model; either a queuing network, petri-net, or simulation. Then the current execution pattern is measured, the workload is characterised, and the input parameters for the model are developed. The model results are compared with real measurements of the system and can be refined. Finally the model is re-evaluated after modifying workload and computer system resource parameters.

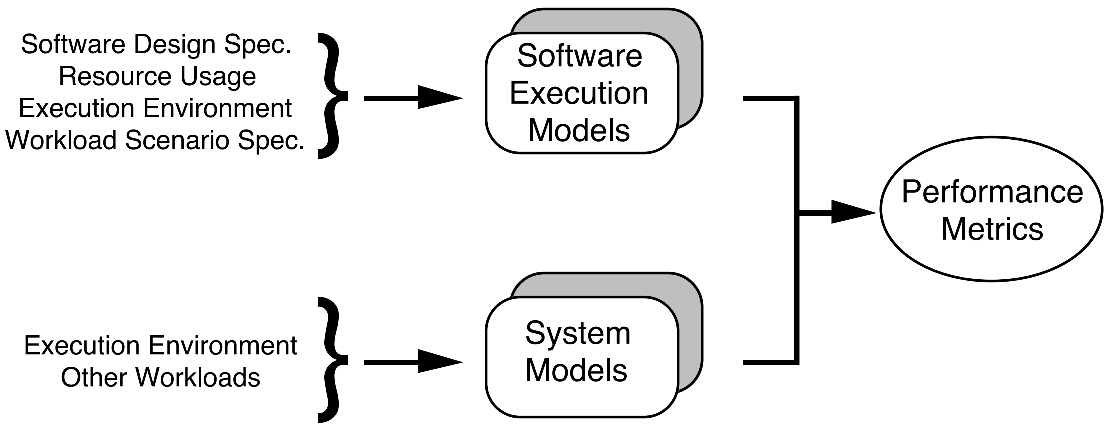


*Figure 1 - Software Performance Engineering Process*

A problem arises with the use of conventional modelling in the case of performance evaluation of software under development. The workload cannot be measured since the software has not yet been constructed or in our case, for the development of parallel software. Software Performance Engineering (SPE) enables these workload parameters to be determined without first constructing the parallel software.

Software Performance Engineering is a relatively new methodology for constructing software to meet performance objectives [Smith90]. The use of SPE begins early in the software life cycle and continues throughout design, coding, and testing stages. It uses quantitative methods to identify appropriate coding alternatives to give satisfactory performance.

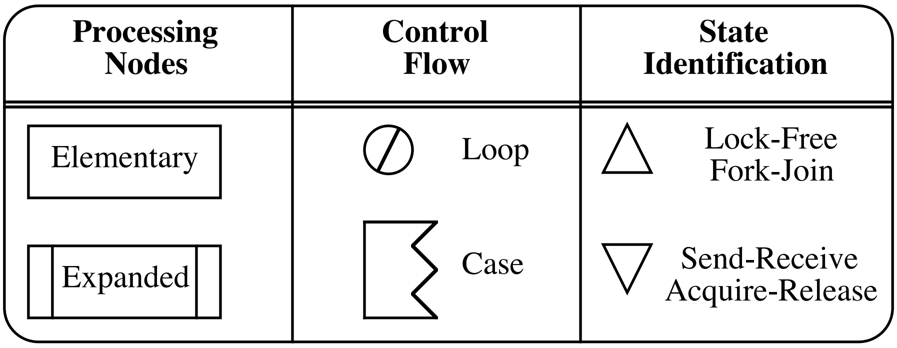


*Figure 2 - Software Execution Graph Notation*

The problem caused by the absence of workload information, in our case, is solved by complementing the conventional computer system model with a software execution model, Figure 1. The software execution model represents the key characteristics of the execution behaviour. The results of the evaluation of the software execution model are similar to the workload data used in conventional system models.

The software execution model uses workload scenarios, software design, execution environment, and resource usage to construct an execution graph model. Each component of the software is represented by a node in the graph. The graph notation includes control statements, hierarchical components, state identification nodes (lock-free, send-receive), and split nodes (concurrent processes). The basic elements of the graph notation are shown in Figure 2. The software execution model is evaluated by graph analysis algorithms to determine the workload parameters used in the system execution model.

The use of SPE provides a software execution notation, it integrates software and hardware models, and is a proven technology. However, it has a number of disadvantages that are particularly apparent in the case of performance prediction of parallel programs :

- The development of the system model, especially for parallel computers, is a time consuming and complicated procedure.

- Queuing networks and other analytical methods can not be used for the evaluation of the system model, due to factors including complex model topologies. In order to overcome this drawback extensive simulation, supported by sophisticated modelling tools, is required.

- The system model must incorporate software characteristics such as phase changes, resource allocation, process creation and destruction etc. The resulting system execution model is both hardware and software dependent. Consequently the model can be not reused in further modelling studies.

## 3. A Layered Approach to Characterising Parallel Software

The major problem in SPE is the development and evaluation of the system execution model. The purpose of this model is to determine the contention of system resources. In this paper an alternative approach for building a system execution model is presented. This method divides the system execution model into two separate sub-models: the parallel paradigm model and the hardware model.

The splitting of the system execution model is based on results from the Cm* project at Carnegie Mellon, including the performance degradation of a parallel application caused by algorithmic penalty, implementation penalty, and algorithm/implementation interaction [Gehringer88].

Therefore the overall system contention is attributable to two main factors:

- the resource contention caused by the characteristics of the parallel algorithm which is independent of the implementation. For example, in an application that uses a master-slave paradigm the dominant resource contention, as the number of slaves increases, is the master process. It receives messages from all slaves but can only process them sequentially [Greenberg91].

- the contention caused by the competition for hardware resources which is dependent on the parallel paradigm implementation. In the case of an application using a master-slave paradigm the processors exchange messages through the interconnection network. This can lead to contention in the communication network.

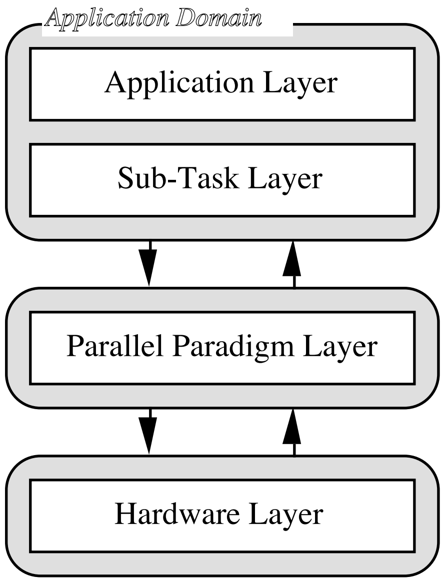


*Figure 3 - Layered Parallel Software Modelling Approach*

Our model representing the parallel paradigm is application and hardware independent. This leads to the suggested layered modelling approach, as shown in Figure 3. A model developed using this approach is organised into the following which are described below: application layer, sub-task layer, parallel paradigm layer, and hardware layer.

## 4. Using the Layered Approach

An image processing benchmark, *ipkernel,* is considered here as a case study [Nudd92, Measure92]. This benchmark detects objects within an image and reports their centroids. The benchmark consists of a collection of sub-tasks such as a sobel filter, dynamic thresholding, a spoke filter etc. The layered modelling approach is described below illustrated using the *ipkernel* benchmark.

### 4.1. Application Layer

The purpose of the application layer is to characterise the application in terms of a sequence of sub-tasks using a software execution graph. The execution graph is used to determine the overall performance of the application using graph analysis algorithms. The metrics for each sub-task is determined in the lower sub-task layer. Resource contention is not considered in this layer but it is taken into account in the lower layers.

The software execution graph for the *ipkernel* benchmark is shown in Figure 4. The structure of the benchmark is straightforward consisting of a sequence of parallel steps without any control statements. The overall execution time for this application is:

$$t_{total} = \sum_{i=1}^{n} t_i \qquad (1)$$

where $n$ is the number of sub-tasks (6 for *ipkernel*) and $t_i$ is the response time of each task.

The nodes of the software execution graph in the application layer consist of a number of elementary processing types, that are classified in the following categories:

- *Sequential Processing Nodes:* Tasks that are appropriate to be performed sequentially.
- *User Defined Parallel Sub-Tasks:* These are defined in the lower sub-task application layer using the same graph execution notation. This is illustrated, using the dynamic threshold operation in the *ipkernel*, in section 4.3.
- *Parallel Processing Generics:* These are frequently used pre-defined parallel sub-tasks which exist as a model library. The formation of such a library is currently under investigation [Peps93].

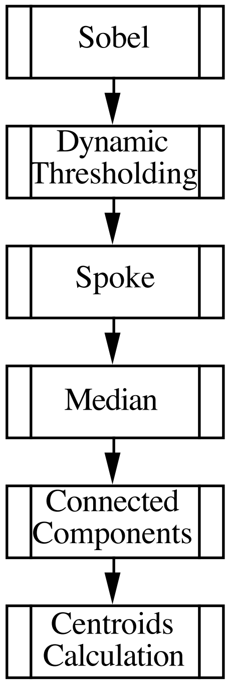


*Figure 4 - Software Execution Graph for the Ipkernel Benchmark*

## 4.2. Parallel Paradigm Layer

The purpose of the parallel paradigm layer is to identify the resource contention caused by the characteristics of the algorithm. It has been noted that a large number of computations fall into a surprisingly small number of prototypical structures [Gehringer88]. In many areas it is possible to identify a small set of algorithm structures that can be modelled using traditional modelling techniques [Allen86]. A number of models already exist for parallel paradigms [Greenberg91, Agrawal83, Rolia92].

In order to identify the most commonly used parallel paradigms a classification scheme is required. The aim of classification is to determine different classes, on the basis of similarities, amongst different parallel algorithms. The classification enables the extraction of the most frequently used constructs and also the definition of application independent parameters.

There have been several attempts to find a systematic approach to the description of the large variety of parallel algorithms including :

- The Cm* project which classified the algorithms according to their co-ordination mechanism [Gehringer88]. These classes are: asynchronous, synchronous, multiphase, pipeline, partitioning, and transaction processing some of which are shown in Figure 5.
- The BACS (Basel Algorithm Classification Scheme) scheme which examines three attributes of parallel algorithms namely, processes, data and interaction [Burkhart93].
- Jamieson [Jamieson87] identifies the characteristics of a parallel algorithm that have the greatest effect on its performance. She also highlights the connection between algorithmic characteristics and features of the hardware, in relation to the application's performance.

By modelling the most representative parallel paradigms it is possible to cover the majority of applications. A parallel paradigm model also includes the mapping of the algorithm onto network topologies. The purpose of determining the algorithm mapping is to identify the communication and synchronisation patterns that are inputs to the hardware layer. For example, different mappings of the master-slave paradigm on a mesh topology result in different communication radius which influences the communication delay. The automatic mapping of a parallel paradigm enables the hardware complexity to be hidden from the user.

The use of a parallel paradigm is demonstrated here using the dynamic thresholding of *ipkernel* benchmark. This operation is parallelised using the multiphase structure, Figure 5. The multiphase paradigm is used for the class of algorithms that are comprised of alternating serial and parallel phases. The response time for the multiphase algorithm, assuming the master is constantly busy, is :

$$t_r = N \cdot \left[ \underset{i=1}{\overset{k}{Min}}(t_i^{slave}) + \sum_{i=1}^{k}(t_i^{in} + t_i^{inseq}) + t^{seq} + k \cdot t^{out} \right] \qquad (2)$$

where: $N$ is the number of phases, $t_r$ is the system response time, $t_i^{slave}$ is the processing time required from slave i during the parallel phase, $t_i^{in}$ is the communication delay for slave i to send the results of the parallel phase to master, $t_i^{seq}$ is the processing time required for the master to incorporate the results of slave i, $t^{seq}$ is the sequential processing time for master processor, $t^{out}$ is the communication delay for master to send the results a slave, k is the number of processors.

In the above equation there are application dependent parameters (e.g. $t^{seq}$) that are determined by the application sub-task layer and hardware dependent parameters (e.g. $t^{out}$) that will be determined by the hardware layer.

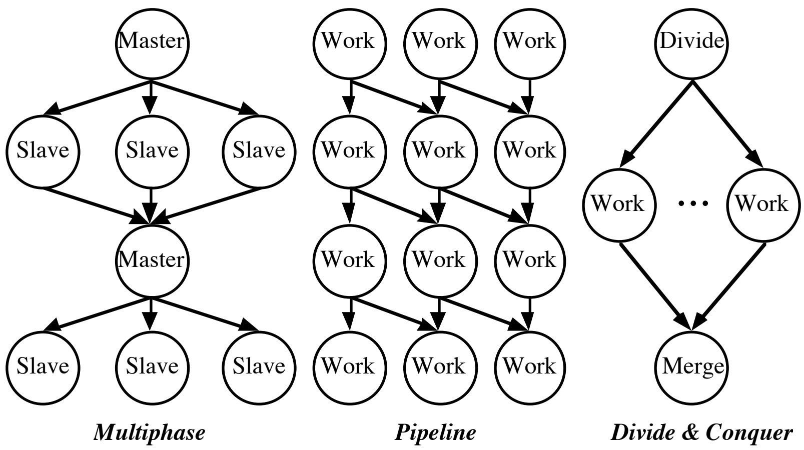


*Figure 5 - Task Graphs for Three of CM* Parallel Paradigm Classification Scheme*

### 4.3. The Application Sub-Task Layer

In the sub-task layer the application specific models are defined for each sub-task. The result of the evaluation of these models is the input to the parallel paradigm layer. The sequential parts of the parallel paradigm must be modelled and their response time determined.

Initially for each sub-task a software execution graph is constructed. The user must then identify the resource usage of each elementary node as defined in SPE. The methodology of this process might include software instrumentation of the sequential version of the program, measurement projection of the software running on a reference computer to the target hardware, and software characterisation [Smith90].

For the *ipkernel* dynamic threshold operation a software execution graph has been developed, part of which is shown in Figure 6. Each processor initially performs a local histogram for a part of the image. This is the parallel phase of the multiphase paradigm. Each processor then sends the local histogram to the master to generate a global histogram. Finally the master calculates and broadcasts the dynamic threshold value to the slave processors.

The parameters required for the multiphase paradigm and hardware layer can be determined using the software execution graph of the sub-task. For example:

$$t_i^{slave} = N_{pixels} \cdot t_{localhist} \; [\forall i \in (1,k)] \qquad (3)$$

### 4.4. Hardware Layer

The hardware layer is responsible for the characterisation of communication, synchronisation, and contention. The information required for this layer can be organised into a hierarchical structure similar to the one used in the architecture characterisation tool in the PAWS project [Pease91]. The requirements of the hardware model are less complex than the system model in SPE, making the development and evaluation procedure easier.

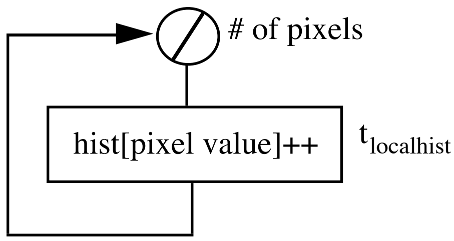


*Figure 6 - Software Execution Graph for Local Histogram*

The hardware model consists of static and dynamic performance parameters. The static parameters represent the delay of hardware operations that are not influenced by the run-time environment and can be determined either by benchmarking or configuration information. The number of processors is an example of a static configuration parameter. The dynamic parameters are influenced by the run-time environment and can be determined by analytical models or hardware characterisation e.g. [Zemerly93].

## 5. Results Using the Structured Layered Approach

The results obtained here are for the dynamic thresholding operation of the *ipkernel* benchmark modelled using the structured layered approach, as described in the previous sections, and compared with results obtained on a 128 node (Transputer T800) Parsytec Supercluster.

For the model development, it was assumed that the source code for the benchmark had not yet been ported on to the Parsytec machine. Consequently, the workload information could not be measured. The only measurement required for the model was the resource requirement of the sequential version of the *ipkernel*. If the hardware was not available for the measurement other techniques could be used to identify the hardware resources. The accuracy of the model was compared afterwards with measurements from the ported version of the benchmark.

The model was developed using Mathematica which has also been proposed as a characterisation and modelling tool by Patel [Patel92]. Mathematica is well suited for this purpose since it combines all the advantages of traditional third generation languages with additional features including advanced mathematical functions and presentation graphics.

The model has three types of input parameters: the hardware configuration, the resource usage of each function measured from the sequential version, and data dependency information such as the size of the image. The user can vary these parameters in order to determine their impact on the overall performance.

Although the case study has been selected for its simplicity to demonstrate the modelling approach, the results are interesting as shown by Figure 7. The response time of the operation decreases for a processor configuration up to 4x4 but increases for larger processor configurations. This phenomenon is caused by the parallelisation overhead. In the *ipkernel* operation the overhead required for the master to receive the local histogram, from each slave processor, and accumulating the global histogram causes the performance degradation.

The comparison between the results of the model and the measurements show a 7% error margin for the various processor configurations. It is expected that this error margin will be greater in a real application. However, this level of accuracy is acceptable and in some cases better than other modelling techniques [Lazowska84].

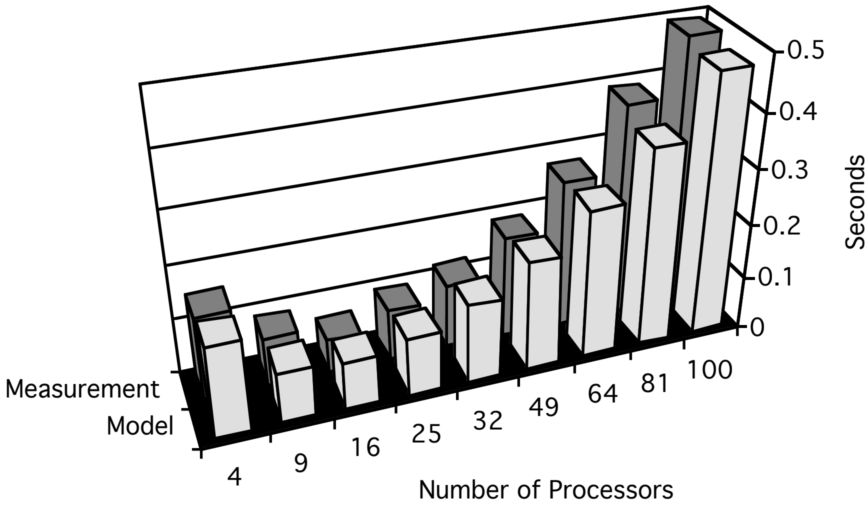


*Figure 7 - Ipkernel Model Results*

## 6. Conclusions

In this paper a novel approach to parallel software characterisation and modelling has been presented. The layered approach is based on the methodology provided in SPE and in model structural decomposition. The main goal of the layered modelling procedure is to develop independent models for the application, the parallel paradigm, and the hardware. Each of these models can be determined by characterisation studies. The independence of the layers, described here, has the following advantages:

- The time required for the development of models is reduced since there is no need for the development of different system models for each study.

- Usually the resulting model can be evaluated using analytical techniques as opposed to the system model of SPE studies which cannot.

- The parallel paradigm and hardware models are reusable. By defining the application layer and sub-task layer, experimentation can be performed to evaluate different paradigms on different hardware.

A disadvantage of this methodology is that the parallel software must conform to the set of parallel paradigms supported. This is not be a problem for the majority of applications, as it has been observed that most of the parallel application conform to a limited set of paradigms [Allen86, Gehringer88, Burkhart93].

Future work in the extension of the layered approach is in the selection of a set of frequently used algorithms to form a library of generics [Peps93]. Another task required for the refinement of the layered approach is the comparison of results with results from traditional modelling techniques, e.g. queuing networks.

## Acknowledgements


This work is funded in part by ESPRIT contracts 6942 - Performance Evaluation of Parallel Systems (PEPS) and 6173 - Design by Simulation and Rendering on Parallel Architectures (DESIRE).

# Bibliography


[Agrawal83] Agrawal, S.C. and Buzen, J.P., "The Aggregate Server Method for Analyzing Serialization Delays in Computer Systems," *ACM Transactions on Computer Systems*, vol. 1, no. 2, pp. 117-143, May 1983.

[Allen86] Allen, J., "Plenary Address," in *IEEE Workshop on VLSI Signal Processing,* 1986.

[Burkhart93] Burkhart, H., Korn, C.F., Gutzwiller, S., Ohnacker, P., and Wasel, S., "BACS: Basel Algorithm Classification Scheme," Tech. Rep. 93-3, University of Basel, Switzeland, 1993.

[Chandy91] Chandy, K.M. and Kesselman, C., "Parallel Programming in 2001," *IEEE Software*, pp. 11-20, November 1991.

[Gabber90] Gabber, E., "VMMP: A Practical Tool for the Development of Portable and Efficient Programs for Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 304-317, July 1990.

[Gehringer88] Gehringer, E.F., Siewiorek, D.P., and Segall, Z., *Parallel Processing: The Cm* Experience*. Digital Press, 1988.

[Greenberg91] Greenberg, G., A. and Wright, E., P., "Design and Analysis of Master/Slave Multiprocessors," *IEEE Transactions on Computers*, vol. 40, no. 8, 963-976, August 1991.

[Jain89] Jain, P., P. and Newton, P., "Putting Structure into Modeling," in *Proceedings of the 1989 Summer Computer Simulation Conference,* Austin, Texas, USA, 1989, pp. 49-54.

[Jamieson87] Jamieson, L.H., "Characterizing Parallel Algorithms," in *The Characteristics of Parallel Algorithms*, Jamieson, L.H., Gannon, D., and Douglass, R.J., Eds. MIT Press, 1987, pp. 65-100.

[Lazowska84] Lazowska, D., E., Zahorjan, J., Graham, G., S., and Sevcik, C., K., *Quantitative System Performance: Computer System Analysis Using Queueing Networks*. Englewood Cliffs, New Jersey, USA, Prentice Hall, 1984.

[Measure92] MEASURE, "Measurement of Array Architectures for Image Analysis Applications," Final Report, ESPRIT 5669, Warwick Strategic Technology Laboratories, Coventry, U.K., January 1992.

[Miller90] Miller, P., B., Clark, M., Hollingsworth, J., Kierstead, S., Lim, S.S., and Torzewski, T., "IPS-2: The Second Generation of a Parallel Program Measurement System," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 206-217, April 1990.

[Nudd92] Nudd, G.R., Atherton, T.J., and Kerbyson, D.J., "IPKERNEL: Image Processing Benchmark," Tech. Rep., Warwick Strategic Technology Laboratories, Coventry, U.K., 1992.

[Peps93] PEPS, "Characterising Processing Needs," Tech. Rep. D5.1, ESPRIT 6942, University of Warwick, Coventry, U.K., July 1993.

[Patel92] Patel, M., N., "Structuring Analytical Performance Models Using Mathematica," in *Proc. of the 6th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation,* Pooley, R. and Hillston, J. (Eds.), Ediburgh, U.K., 1992, pp. 273-286.

[Pease91] Pease, D., Ghafoor, A., Ahmad, I., Andrews, L., D., Foudil-Bey, K., and Karpinski, E., T., "PAWS: A Performance Evaluation Tool for Parallel Computing Systems," *IEEE Computer*, pp. 18-29, January 1991.

[Rolia92] Rolia, J.A., "Predicting the Performance of Software Systems," Ph.D. thesis, University of Toronto, Canada, 1992.

[Smith82a] Smith, C.U. and Browne, J.C., "Performance Engineering of Software Systems: A Case Study," in *Proc. National Computer Conference,* vol. 15, 1982, pp. 217-224.

[Smith82b] Smith, C.U. and Loendorf, D.D., "Performance Analysis of Software for a MIMD Computer," in *Performance Evaluation Review.* ACM Press, 1982.

[Smith90] Smith, U., C., *Performance Engineering of Software Systems*, The SEI Series in Software Engineering. Addison-Wesley Publishing Co., Inc., 1990.

[Weishar87] Weishar, D.J., "Incorporating Expert Systems Technology into Software Performance Engineering," in *Proc. Computer Measurement Group 1987,* 1987, pp. 720-722.

[Zemerly93] Zemerly, M.J., "Characterisation of Multi-Processor Systems," Tech. Rep., University of Warwick, Coventry, U.K., 1993.