Characterisation Survey*

M. J. Zemerly and E. Papaefstathiou
Department of Computer Science
University of Warwick
Coventry CV4 7AL
email: jamal/stathis@dcs.warwick.ac.uk

April 29, 1993

Abstract

The performance of a computer system is affected by the characteristics of its various hardware and software subsytems. Hardware and software characterisation give the ability to change machine and programs parameters and observe their effects on the performance of the system without changing the existing hardware which can be time consuming and cost prohibitive. This report provides a survey of the techniques and tools used in the hardware and software characterisation of parallel systems.

1. Introduction

Performance is one of the key factors that must be taken into account in the design, development and tuning of a computer system. The overall system performance is affected by the performance characteristics of the various subsystems and hardware and software components that constitute the system [Lav83]. Hence the design of new hardware or software components has system performance implications and should not be ignored. The performance of a system can be evaluated by measurements using hardware and/or software monitors either in user environment or under controlled benchmark conditions. Normally these monitors utilise some sort of hardware and/or software characterisation to achieve their goal. Hardware and software characterisation give the ability to change programs and machine parameters and observe the results of the modifications rather than changing the existing hardware which can be time consuming and cost prohibitive. This work complements two other ESPRIT projects: IPS1-1532 [E1588] and IPS2-IMSE-2143 [E2192]. The first studied a vector-oriented parallel architecture for a supercomputer and evaluated it by simulations at the hardware and software levels. The second is more closely related to this work and deals

*This work is supported by ESPRIT project no. 6942 “Performance Evaluation of Parallel Systems (PEPS)”.

with improving systems engineering methods throughout the design cycle of a system by developing a support environment for performance modelling.

The following sections provide a survey of the techniques and tools used in the hardware and software characterisation of parallel systems.

2. Characterisation Techniques

2.1. Hardware Characterisation Survey

Hardware characterisation gives an indication of what performance to expect from a parallel system running a certain application or benchmark. Many researchers have characterised parallel systems on the components level (e.g. cache, CPU, etc.) with or without combining the effects of all the components [HP90]. Some researchers included the characterisation technique within a monitoring or modelling system (e.g. PAWS). Others dealt only with one component and these will not be surveyed in this chapter. Typical performance measures for parallel systems are execution time, speedup, system utilisation, throughput, MIPS, MFLOPS, etc. Execution time is influenced by a wide variety of factors such as hardware technology, processor architecture, system architecture, operating system, language, compiler and even programming style. The execution time can then be taken as the direct indicator of performance for parallel programs since, unlike other measures, it combines influences of all the system levels [HP90].

Characterisation of multi-processor systems requires the characterisation of a single processor as well as some other issues such as diversity of architectures, number and configuration of processors, traffic per processor (inter-process communication), problem size and type of memory (shared or distributed), synchronisation or asynchronisation, granularity, to mention but a few.

Many researchers have attempted to characterise a single micro-processor since computer evolution but little work has emerged until the VAX RISC I architecture which was characterised by Emer and Clark [EC84]. Emer and Clark used a measurement monitor to keep the count of the number of microcode cycles executed at each microcode location. Live time-sharing as well as synthetic workloads were used in the experiments. The measurement technique yielded the amount of average processing time spent in various activities in the workload.

An early comprehensive survey of the quantitative methods used in computer performance evaluation was provided by Heidelberger and Lavenberg [HL84]. They reviewed work in three main areas: performance measurements, analytical performance modelling and simulation performance modelling during the period 1970-1984. Performance measurement is possible once the system is built, instrumented and running. Modelling is required in order to otherwise predict performance. Performance modelling is widely used not only during design and development but also for configuration and capacity planning. Performance models span the range from simple analytical queuing models to very detailed trace driven simulation models. The importance of the performance modelling is that it gives quantitative predictions and an insight into the structure and behaviour of a system. This is particularly valuable during the system design to discover any design flaws. They have classified performance measurement into 3 main areas: instrumentation, workload characterisation and statistical methods. Description of these and work achieved in the period above are given. Analytical performance models which mainly concentrated around queuing theory models are also described and reviewed so as the simulation performance models which mainly constitute of trace driven and stochastic discrete event simulation models.

Hockney [Hoc88] developed a model for SIMD and MIMD machines based on properties of a user program. Hockney characterised vector (SIMD) machines by the performance of their vector pipelines. The average processing rate was chosen as his performance metric. He derived an equation for the performance based on the average time for processing an element, startup time and the number of elements. He then derived the performance for a vector pipeline machines from the average processing time. For MIMD machine an equation for the performance (average processing rate) was derived in a similar way but here he introduced factors for synchronisation and communication times. These models were tested on a number of problems (benchmarks) and results were presented. From measurements he found that the startup time and synchronisation time vary approximately linearly with the number of processors.

Saavedra-Barrera *et al.* [SBSM89] described a way to predict performance by combining benchmarking and hardware characterisation. The idea is to create and use a machine characteriser which measures the performance of a given system in terms of a Fortran abstract machine. This analysis yields a set of parameters which characterises the system and spotlights its strong and weak points. Each parameter provides the execution time for some primitive operation in Fortran. Measurements for a large number of machine ranging from small workstations to supercomputers were presented. They distinguished, however, between system characterisation and performance evaluation. The system characterisation is defined as an n-value vector where each component represents the performance of a particular operation. The performance evaluation of a system is defined as the measurement of some number of properties or performance measures (e.g. execution time) during the execution of a workload.

Hennessy and Patterson [HP90] provided a comprehensive characterisation of one micro-processor through characterising its components. Namely, the CPU, the memory hierarchy and the I/O. Hennessy and Patterson though stopped short of providing an analytical expression for a processor as a whole. They argued that the execution time is the most meaningful performance measure for computer systems and they centred their processor characterisation on this measure. For each component they tried to devise a performance equation based on the characteristic of that component. For the CPU for example they measured the execution time as the product of the instruction count in a program, the average clock cycle per instruction, and the clock cycle time. Similar expression for the memory and I/O were also given. The memory hierarchy (cache and main memory) was characterised by the average access time as a function of the cache hit ratio and time to access the cache and the main memory. The I/O was characterised by the data throughput (i.e. number of bytes per second that can be transferred between the CPU and the main memory and disks during large transfers).

Murray [Mur90] and Wilkinson [Wil91] have also characterised a single processor in a similar fashion to Hennessy and Patterson with some modifications to the performance equations of the processor components especially in the memory hierarchy equations. Wilkinson introduced a measure termed *space-time product* which is the product of the memory used by a program and the amount of time that is used. This measure should normally be minimised to reduce cost. Wilkinson also derived some performance measure equations for models of parallel computation on multi-processor systems where he introduced some load imbalance and communication overheads. Speedup, efficiency, and processing (execution) time are used as performance metrics. He also derived equations for the bandwidth shared memory inter-connection networks. Murray also derived an analytical model for performance of vector pipeline systems.

Basu *et al.* [BSKP90] derived simple analytical models for evaluating performance of parallel message passing systems. In this system the execution of an algorithm is modelled as a repetitive cycles where a cycle consists of computation and communication. Both overlapped and non-overlapped computation-communication execution models are discussed. In the overlapped model, computation and communication can be partially or completely overlapped. In the non-overlapped model all the processors do some computation and then synchronise and exchange data. Basu *et al.* assumed that all inter-process communication times can be estimated *a priori* and that there are no queuing delays in the system. The data domain is assumed partitioned into sub-domains of equal sizes and all the processors execute the same program on a different data domain (SPMD). In these models the analytical expressions (for execution time, speedup and efficiency) account for the communication overheads and the number of the processors. Utilization of the processors can also be modelled. In an overlapped cycle computation and communication the fraction of overlap between the computation and communication is taken into account as well as the processor context switching (computation to communication and vice versa) overheads. All the parameters can be derived relatively easily by either direct measurements or by experiments.

2.2. Software Characterisation Survey

During the development of hardware and system software a great number of decisions must be made that will influence considerably the performance and the features of the system. In order to predict performance a system designer uses modelling. The modelling process includes the development of an abstract system that is used for the analytical or simulation prediction of the performance of the real system. Before the system design process however the engineer must have a feeling about basic design decisions and keep this insight during the design and development procedure. A method used to achieve this goal is software characterisation. With this method basic characteristics of software running on existing machines are extracted and studied in order to understand the impact of fundamental hardware and system software that are independent from the underlying architecture.

An example of the importance of software characterisation is the work of Hennessy and Patterson [HP90]. They presented measurements of a set of programs (gcc, Spice, TeX) running on a VAX computer. The measurements included frequency of instructions, use of addressing modes, size of operands etc. Based on these observations they came to the conclusion that the simple instructions are used with higher frequency than complex instructions. This conclusion was the basis for the development of RISC processors.

2.2.1. Benchmark Characterisation

This work addresses the prediction of the benchmark performance on a system that is under development [CH91]. The traditional approach is to simulate machine performance before the system is built and use the results to improve the design. The approach followed in this project is the definition of an abstract system that is general enough to include many system designs as special cases and then measure benchmark performance in terms of this abstract system. The method is based on high-quality architecture-independent compilers that use a portable intermediate language. From the benchmark characterisation procedure conclusion can be drawn about the system components of the system (processor, memory, operating system). Although the aim of the project was to expand the methodology for parallel systems no further results have been presented after the January of 1991.

2.2.2. Data Dependency Analysis For Parallel Software Characterisation

Data dependency analysis has been used to determine the order of program statement execution and to identify operations that may be executed in parallel. An approach to data dependency analysis is the use of intermediate form languages to represent the program. The study of the intermediate form represent the program's dependencies and parallelism and provides insight into the program's structure and organization. The intermediate form can be viewed as the characterisation of the parallel software because it can be mapped onto any parallel machine. This approach has many advantages such as that it provides a common target for any high level language, it allows a single application to be analysed on each machine that the intermediate form language supports and it allows users to perform a machine independent application language.

This characterisation method has been introduced in PAWS [PGA+91] application characterisation tool. The intermediate form representation language that has been used in PAWS is an acyclic graphical language. PAWS translates programs written in ADA to IF1 [Law85] and then it uses the hardware characterisation tool to estimate the behaviour of the application running onto a specific parallel computer

2.2.3. Parallel Algorithmic Characterisation

Another approach of parallel software characterisation is the classification of parallel algorithms. This is a higher level technique that organises the most common parallelisation techniques independently from the algorithm implementation on a specific parallel computer.

A first discussion about parallel algorithm classification took place in the 1st Workshop on the Taxonomy of Parallel Algorithms held in Santa Fe in 1987 [JGD87]. In this workshop many issues of the classification were addressed, the most important of which were:

1. The attributes of algorithms that are the most important in dictating the structure of a parallel algorithm and how they can be characterised and formalised.
2. The salient characteristics of parallel algorithms in various application domains such as speech recognition, image processing, matrix operations etc.
3. The communality in algorithm structure of the algorithms across the problem domains.

Another attempt has been made by the scientists of the Informatics Laboratory in University of Basel. The result of this effort is the Basel Algorithm Classification Scheme (BACS) that is presented in [BKG+93]. Based on this classification every parallel algorithm can be described in terms of:

1. Process Properties
   - Structure: Static or dynamic structure depending on whether or not the algorithmic topology changes.
   - Topology: mesh, ring, hypercube etc.
   - Execution Structure: A description of the processing-communication- synchronisation structure.
2. Interaction
   - Interaction of processes: broadcast, barrier, semaphores etc.
3. Data Attributes
   - Placement: List of all the atomic elements that are distributed
   - Distribution: Global, static, dynamic.

2.2.4. Other Projects

Software characterisation has been used often in software engineering [Yam90]. Software performance engineering uses software characterisation for software modelling studies as part of the workload and resource identification and quantification [Smi90]. It is included in the instrumentation and monitoring stage of the software modelling.

Other projects related to software characterisation can be grouped into two categories:

1. characterisation methodology ([Ros86, Sme86])
2. characterisation studies for specific areas of software behaviour ([MWB91, BS85, MB88]).

An example for the second category is a study for the memory referencing of executing programs independent of the environment in which they might run presented in [MB88]. The program behaviour is presented as phases and transitions. Various parameters of a selected set of UNIX programs (grep, ls, more, pwd, who) are quantified, including :

1. the distribution of holding size of each page set,
2. the process by which the program chooses new locality set at transitions,
3. the process by which the program generates references from within locality set etc.

3. Tools for Performance Evaluation

Some tools which have been developed to measure the performance of parallel computer system have used hardware and software characterisation techniques. A brief description of characterisation tools is given below.

3.1. Hardware Characterisation Tools

A number of tools for hardware characterisation of parallel systems have emerged.

SIMPLE (Source related and Integrated Multiprocessor and computer Performance evaluation modeLing and visualisation Environment) [Moh91] is a modular performance and visualisation tool environment for parallel and distributed systems based on monitoring of concurrent inter-dependent activities. SIMPLE is independent of the monitor device used and the system monitored. This is achieved by using a data access interface which measures data of arbitrary structure, format and representation. SIMPLE can support eight categories of activities which contribute to the performance evaluation. These are preparing the measurement, supporting the measurement, accessing traces, generating and integrated view, validating traces, evaluating traces, visualising traces, and modelling the performance. graph models or petri-nets models can be used to describe the functional behaviour of a program on a chosen level of abstraction. Adding timing, frequency and probability values to the functional model leads to a performance model which can study the performance behaviour of other configurations of the program.

ELAN (Efficiency Loss ANalysis) [Mos90] is an experimental environment that offers program development, configuration and debugging and measurement tool for program observation as well as performance evaluation and loss analysis in a shared-memory multi-processor system. ELAN uses the speedup and efficiency as the performance metrics. The efficiency of the system is derived from the losses in efficiency caused by 7 types of losses. These are: idle processor loss, shared memory access loss, conflict loss, garbage computation loss, braking loss (caused by inter-process communication mechanism), organisational application overhead and finally organisational system overhead. Tools to measure these losses are discussed.

3.2. Software Characterisation Tools

The characterisation of software procedure is a relative static procedure. The software under investigation is only characterised once. Due to the non repetitive procedure and the unique characteristics of each study there is a limited number of tools that can be used for characterisation studies.

Usually programs supplied from the operating system might be used as tools to carry out studies. After the completion of the study a statistical package or a spreadsheet can be used to query, present, and statistically formulate the acquired data. Two tools that can be used for software characterisation are: GNU C compiler and PAWS. Since PAWS consist of a hardware and software characterisation tools it will be discussed in separate section.

The GNU C compiler is a public domain compiler that includes an abstract machine intermediate language named Register-Transfer Language. This intermediate language can map to a wide variety of computers like Digital VAX, MIPS R2000, Motorola 68000, and Sparc [Sta89]. Consequently, observations made using the Register-Transfer Language is applied to all the target architectures. GNU C compiler is not a characterisation tool but can be used in combination with other tools to conduct characterisation studies. The disadvantage of GNU C is that it is a compiler for serial computers and consequently only sequential programs can be characterised.

Product: GNU C Compiler

Availability: Public domain software

Contact: The compiler can be downloaded from 'prep.ai.mit.edu' anonymous ftp. Information about the compiler can be found in the internet gnu.gcc news groups.

3.3. Hardware and Software Characterisation Tools

PAWS (Parallel Assessment Window System) [PGA+91] is an experimental system developed in Syracuse University for performing machine evaluation and comparison. PAWS is the best attempt made to characterise hardware and software of parallel systems. PAWS consists of four tools: the hardware characterisation tool, the application (in high level languages, Ada at the moment) characterisation tool, performance assessment tool and interactive graphical display tool.

The hardware characterisation tool can model different architectural configuration of parallel machines based on the number and flexibility of different functional units, the number of processors, memory bandwidths and memory hierarchy and the type of inter-process communication mechanism. The characterisation method partitions the architecture into computation, data movement and communication, I/O and control. Each of these partitions has a timing module which contribute to the overall timing structure.

The application characterisation tool translates application written in high-level source language into a single dependency graph. This allows users to view their application attributes. The application characterisation tool provides the facility to evaluate the level and degree of an application's parallelism. Application characterisation consists of a data dependency analysis to determine the order of program execution. It also identifies operations that may be executed in parallel. This approach has many advantages such as that it provides a common target for any high level language, it allows a single application to be analysed on different hardware supported by PAWS. It also allows users to perform a machine independent application analysis.

The performance assessment tool generates profile plots through the interactive display tool. These two tools allows users to evaluate the performance of any application (using the application characterisation tool) by generating a set of performance metrics which include speedup curve, parallelism profile curve, and execution profiles. These are generated for both the ideal case (i.e. theoretical upper bound performance) and for the application after it has been mapped onto a machine. A comparison of the two performance metrics shows the effects of mapping the application onto the machine.

PAWS is the best known characterisation tool. An ADA front end has been developed and C++ and FORTRAN front ends have been scheduled. Currently PAWS is funded by Rome Labs, a research branch of the US Air Force. A version of the tool has been distributed among the parallel user group of Rome Labs but the product is not commercially available.

Product: PAWS

Availability: Restricted to parallel user group of Rome Labs.

Contact: Dan Pease (peased@npac.syr.edu), Syracuse University.

4. Conclusion

Characterisation models and tools are needed to study the effects of various parameters on the performance of mapping applications (or benchmarks) onto multi-processor systems. This is an important design feature that allows modifications to existing hardware parameters to study the performance without physically changing the hardware which can be time consuming and cost prohibitive. Due to the non repetitive procedure and the unique characteristics of each study there is a limited number of developed models or tools that can be used to study the characterisation of parallel systems.

References

[BKG+93] H. Burkhart, F. C. Korn, S. Gutzwiller, P. Ohnacker, and S. Waser. BACS: Basel Algorithm Classification Scheme. Technical Report 93-3 Version 1.1, Universit tsrechenzentrum und Institut f r Informatik, Mittlere Strasse 142, 4056 Basel, March 1993.

[BS85] V. Bassili and R. Selby. Calculation and Use of An Environment Characteristics Software Metric Set. In *8th International Conference on Software Engineering*, pages 386–391, 1985.

[BSKP90] A. Basu, S. Srinivas, K. G. Kumar, and A. Paulraj. A Model for Performance Prediction of Message Passing Multiprocessors Achieving Concurrency by Domain Decomposition. In H. Burkhart, editor, *Proc. of the Joint Int. Conf. on Vector and Parallel Processing*, pages 75–85, Zurich, Switzerland, 10-13 September 1990. Springer-Verlag, Berlin. Lecture Notes in Computer Science, 457.

[CH91] M. T. Conte and W. W. Hwu. Benchmark Characterization. *IEEE Computer*, pages 48–56, January 1991.

[E1588] A Peliminary Study of a Vector Processing-Oriented Parallel Architecture. Esprit project number: 1532, 1987-1988.

[E2192] An Integrated Modelling Support Environment (IMSE). Esprit project number: 2143, 1989-1992.

[EC84] J. S. Emer and D. W. Clark. A Characterisation of Processor Performance in the VAX-11/780. In *Proc. 11th Int. Symp. on Computer Architecture*, pages 301–310, Ann Arbor, Michigan, 5-7 June 1984.

[HL84] P. Heidelberger and S. S. Lavenberg. Computer Performance Evaluation Methodology. *IEEE Trans. on Computers*, c-33(12):1195–1220, 1984.

[Hoc88] R. W. Hockney. Problem Related Performance Parameters for Supercomputers. In J. L. Martin, editor, *Performance Evaluation of Supercomputers*, pages 215–235. Elsevier Science Publishers (North-Holland), 1988.

[HP90] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 1990.

[JGD87] L. Jamieson, H., D. Gannon, and R. Douglas, J., editors. *The Characteristics of Parallel Algorithms*. MIT Press, 1987.

[Lav83] S. S. Lavenberg, editor. *Computer Performance Modelling Handbook*. Academic Press, New York, 1983.

[Law85] Lawrence Livermore National Laboratory. *An Intermediate Form Language IF1*, 1985.

[MB88] M. J. Murphy and B. R. Bunt. Characterising Program Behaviour With Phases and Transitions. In *Proc. of the 1988 ACM SigMetrics Conference on Measurement and Modeling of Computer Systems*, volume 16, pages 226–233. ACM Press, May 1988.

[Moh91] B. Mohr. SIMPLE: A Performance Evaluation Tool Environment for Parallel and Distrubuted Systems. In A. Bode, editor, *Proc. of the 2nd European Conf. on Distributed Memory Computing (EDMCC2)*, pages 80–89, Munich, 1991. Springer-Verlag. Lecture Notes in Computer Science, 487.

[Mos90] M. Moser. The ELAN Performance Analysis Environment. In H. Burkhart, editor, *Proc. of the Joint Int. Conf. on Vector and Parallel Processing (CONPAR)*, pages 189–199, Zurich, Switzerland, 10-13 September 1990. Springer-Verlag. Lecture Notes in Computer Science, 457.

[Mur90] W. D. Murray. *Computer and Digital Systems Architecture.* Prentice Hall, USA, 1990.

[MWB91] S. Majumdar, C. Woodside, and D. Bailey. Characterisation and Measurement of Parallelism in Communication Protocol Software. In *Proc. of International Conference on Parallel Processing*, volume 2, pages 270–271, 1991.

[PGA+91] D. Pease, A. Ghafoor, I. Ahmad, D. Andrews, K. Foudil-Bey, T. Karpinski, M. Mikki, and M. Zerrouki. PAWS: A Performance Evaluation Tool for Parallel Computing Systems. *Computer*, pages 18–29, January 1991.

[Ros86] L. L. Rose. Software Characterisation Indepent of Hardware. In *1986 Winter Simulation Conference Proc.*, pages 727–731, 1986.

[SBSM89] R. H. Saavedra-Barrera, A. J. Smith, and E. Miya. Machine Characterization Based on an Abstract High-Level Language Machine. *IEEE Trans. on Computers*, c-38(12):1659–1679, 1989.

[Sme86] R. Smelyanski. A Mathematical Model for Computing Dynamic Program Characteristics. *Programming and Computer Software*, 12(1):44–50, 1986.

[Smi90] C. Smith, U. *Performance Engineering of Software Systems.* The SEI Series in Software Engineering. Addison-Wesley Publishing Co., Inc., 1990.

[Sta89] R. Stallman, M. *Using and Porting Gnu CC.* Free Software Foundation, 1989.

[Wil91] B. Wilkinson. *Computer Architecture: Design and Performance.* Prentice Hall, UK, 1991.