Chapter 1

Introduction

This thesis documents usable techniques for designing parallel programs with *a priori* knowledge of their run time. It asks whether the guesswork can be taken out of the design process and replaced with engineering decisions based on firm data.

There is currently a gulf between the sophisticated performance analysis techniques developed by the academic community and the techniques which are used in practice.

Because of this, the approach taken in this thesis deliberately focusses on the low level parallel programming tools actually used, in the hope that it will have an immediate practical benefit to developers. Higher level techniques can build on this foundation later.

The contribution of this work to the subject is both in the form of useful tools for characterising and predicting performance and in showing that post-mortem techniques may be supplemented by ante-natal design.

This introduction reviews performance analysis techniques described in the literature and outlines the approaches investigated in the thesis.

1.1 Parallel program design techniques

The two extremes in the art of performance analysis are PRAM style complexity theory and post mortem tracing.

In practice, programmers usually concentrate on writing programs with a clear structure and worry about the performance afterwards. This is not because they don't care about the performance, but because it takes too long to work it out.

Performance is not the only design aim of a parallel system, and may not even be the the most important. However it is the one which differentiaties parallel from sequential software development.

Various techniques for performance analysis are outlined below, including computation models, high level models, mathematical models, software and hardware engineering, real time systems development, simulation, micro-benchmarking and sequential techniques.

1.1.1 The null method

This is the common approach for developing parallel programs. If a performance analysis is done, it is an afterthought.

1.1.2 Post-hoc analysis

Many papers have been written about the performance of a program on an architecture. A good example is Singh and Hennessey's paper on an ocean modelling program [23]. These specific examples are interesting, but shed little light on how one is supposed to go about developing a different program on another architecture.

1.1.3 Higher level techniques

One approach is to restrict programs to using high level operations which have been implemented efficiently (e.g. algorithmic skeletons [9] and the parallel utilities library at the Edinburgh Parallel Computing Centre (EPCC) [8]). This has considerable appeal as it frees programmers from such low level concerns as performance. However the techniques are not sufficiently well advanced to be applied to all problems and are still an area of active research.

1.1.4 PRAM

Parallel algorithm researchers are concerned with predicting the asymptotic complexity of algorithms, where the quality of an algorithm may be expressed in big O notation (e.g. an O(log N) algorithm is "better" than an $O(N^2)$ one). This approach provides a clean, simple method of comparing algorithms, but has several major drawbacks.

The first is that the computational model is idealised and will therefore not (necessarily) have much relevance to actual implementations of algorithms. Work in progress to make the models more complex and realistic (such as the HPRAMs, other PRAMS) tends to make the model harder to use and hence less useful. The other approach, redesigning parallel computers to implement the models more effectively is fairly revolutionary and hence not likely to happen unless there

is overwhelming evidence in favour of programs being easier to design using a PRAM-based model. Interestingly, PRAM-advocates insist that a major deficiency of implicit parallelism through (say) dataflow or functional languages is that performance prediction becomes trickier.

The second drawback is that the basis for comparison (asymptotic complexity) is only valid for an infinite data size or number of processors. In practice, the constant factors may be more important for machine/program sizes of interest.

1.1.5 BSP

An interesting approach, proposed by Valiant [43] provides a computational model consisting of a sequence of parallel supersteps within which local computation is performed and communication requests are posted. Between each superstep is a global barrier operation after which all posted requests are guaranteed to complete.

Restricting synchronisation to global barrier operations (and the programming style to SPMD) simplifies the general performance prediction problem to one of estimating the maximum superstep computation time and the time for the global reorganisation of data at each superstep. The performance of a machine is characterised by three values determined experimentally: **s** is the speed of computation of a process in flops, **l** is the synchronisation latency cost in units of **s** and **g** is the number of flops per word required for all processors to communicate a message simultaneously. Hill, Crumpton and Burgess [17] present interesting results using an implementation of BSP (BSPlib) on an IBM SP/2 and ethernet, comparing simplistic pencil and paper modelling with results from a profiling version of the library.

1.1.6 LogP

An attempt to create a more realistic model based on actual machine parameters rather than an abstract ideal is LogP [10]. The parameters are **L**, an upper bound on the latency suffered by a word sent from one module to another, **o**, the overhead during which a processor is occupied sending or receiving a message, **g**, the minimum time interval between successive message transmissions of receptions, and **P** the number of processors [10]. The authors note that "Such a model must strike a balance between detail and simplicity in order to reveal important bottlenecks without making analysis of interesting problems intractable."

The parameters can be estimated using a simple benchmark routine. They provide a simple pipeline model for point to point communications. Costs for

collective communications may be expressed in terms of the point to point costs, but this is not incorporated directly in the model.

Another modelling notation is $R_\infty/n^{1/2}$ developed by Hockney and Jesshope, originally for vector performance. R_∞ is the maximum rate of transfer (for infinite message sizes) and $n^{1/2}$ is the message size which achieves half of this rate. This has some advantages over

$$startup\_time + message\_size * time\_per\_byte$$

in indicating the "break-even" point for message sizes in a usable way. Numrich [38] gives these values for point-point communication on the Cray T3D network.

1.1.7 Scalability analysis tools

The NASA AIMS/MK toolset [32] extracts a program execution graph from a run of a program and uses this to feed into a discrete event simulator. Sarukkai/Mehra offer *abstract interpretation* techniques for generating complexity estimates [40]. Dunlop et al [1] looked at estimating the workload on the floating point unit and the different parts of the memory hierarchy given Fortran source code, and used this estimate for predictions.

Another top level approach described by Driscoll [11] looks at the total time spent in communication and computation throughout the program, using a variant of Amdahl's law to predict speedups. Gustafson [14] looked at the case of problem size scaling with machine sizes.

1.1.8 Queueing model techniques

Queueing theory is a well developed mathematical technique for analysing steady state performance of queueing networks. King [25] describes the application of queueing theory to computer systems. Analytical solutions exist for simple networks but more realistic networks must be simulated. The basic parameters of a queueing model (arrival rate, queue sizes etc.) may correspond directly to design parameters.

Queueing models have been used for prediction. Liang and Tripathi [26] used a simple queueing model to analyse fork/join program graphs. Mak and Lungstrom [28] developed queueing models of architecture and program for their predictions.

1.1.9 Petri-net techniques

Petri nets [34] have been used for modelling behavioural aspects of concurrent systems, particularly detecting the presence of deadlocks. Standard Petri nets do not incorporate the notion of time, so timed and stochastic extensions are typically used for modelling performance of systems. The problem with these more complex varieties of Petri nets is that they make mathematical analysis more difficult, and the state space becomes too large to search. Even with standard Petri nets, the graphical models rapidly become incomprehensible. Hartleb [15] looked at stochastic graph techniques for parallel program performance. Graph nodes were deterministic, or used a random distribution. Reduction techniques were used for simplifying models. Wabnig [21] derived a Petri net model of the communications network from first principles and used this for performance studies.

1.1.10 Process algebras

Process algebras (CCS [29], CSP [19]) incorporate better support for modelling hierarchy than Petri nets, and are amenable to analysis using state space searching techniques.

Timed variants such as TCCS [30] may be used for proving properties such as "state X occurs *before* state Y", but are not so concerned with actual run times. PEPA [18] allows for stochastic state transitions, with standard techniques used to analyse the resultant Markov chains.

If delays are deterministic rather than probabilistic, then process algebras give no more insight than simulation.

1.1.11 Parallel software engineering

Traditional software engineering techniques (Yourdon, Mellor, etc) generate a large amount of concurrency in the initial "structured analysis" phase, which they subsequently remove to produce a sequential structured design. They have nothing to say about the problems of a parallel implementation.

Attempts to develop software engineering techniques for parallel systems such as PARSE [22] have focussed on extending dataflow techniques and defining their semantics more rigorously, but have no advice on how to build in efficiency. They are also geared towards distributed systems (a few distinct processes communicating) rather than parallel systems (many identical processes communicating).

1.1.12 Hardware engineering

Concurrency comes naturally to hardware engineers as electronic components all run in parallel. Timing issues are often central to the design, so predicted timing diagrams are drawn up early in the design.

Design tools are used to a far greater extent than in the software community, with graphical tools such as schematic editors, language based tools such as VHDL and Verilog simulators, state machine designers etc etc.

Unfortunately hardware is not software so hardware design techniques cannot be directly applied to engineering of parallel software.

The important differences are :

- software components are never specified as rigorously as hardware components.
- interactions between software components are less restricted.
- a software component may be orders of magnitude more complex than a hardware component.

Software gives the engineer infinite rope to play with, whereas hardware is always bounded by physical constraints like pin count and chip area, so by necessity hardware components are better defined than software ones. Attempting to restrict software to use a controlled interface is one of the aims of software engineering, but the temptation is always there to bypass the restrictions and use a "quick and dirty" technique. Doing this in a hardware design is of course also possible, but much less common. Having to cast ideas into the stone of a circuit board encourages cleaner designs than does the free form of software.

1.1.13 Real time systems

The area of real time systems covers similar timing issues to that of parallel programming, but a poor design may be fatal rather than just inefficient. The emphasis is on predictability rather than on absolute performance; an implementation must guarantee that a deadline is met.

The Flex language [24] inserts `#pragma` comments with the expected timing equation for each section of code. The MAXT approach [36] attempts to calculate the maximum execution time of programs using software annotations and an extra compilation step. The restrictions of this method are that compile time predictions of loop counts are not always possible and also recursion is not handled.

Park and Shaw [33] present a timing schema approach which benchmarks the performance of a generalised assembler code on an architecture (with instructions such as `mov, add, mul` etc.) and then parses high level source code in terms of these instructions.

1.1.14 Direct execution simulation

Direct execution simulators allow detailed modelling of hardware and tweaking of all kinds of parameters. Indeed Brewer [5] recommends using simulation as a development platform in preference to running on a machine, based on experience with the Proteus simulator [4] of shared memory software on the CM-5 machine. The network parameters are fed in using a network model such as that described in [2] or by doing a detailed hop by hop simulation.

One possible criticism of this approach for software development is that the models take too long to construct and verify, and it is no easier than running on the actual machine. Since the models are hidden from the programmer (behind the mystique of the simulator), it is another post-mortem like approach, the only difference being that the actual machine is not used.

The advantage of this detailed approach is that network contention and other such issues may be modelled as accurately as desired.

An example of simulation applied to message passing software is PS [3], a direct execution simulator for PVM based on the Ptolemy simulation system [6]. It uses a complex model of an ethernet interface for its communications subsystem, and has been used for PVM applications running on a small number of workstations across a network. Pouzet [35] described a simulation tool for Transputer applications.

Fahringer [12] developed a system for guiding compiler optimisation as part of the Vienna Fortran Compilation System. The system statically computes a small set of parameters which characterise the overall behaviour of a parallel Fortran application.

1.1.15 Benchmaps

At the boundary between simulation and analytical techniques less work has been done. *Benchmaps* were developed by Toledo [42, 41] for prediction of data parallel programs. His technique relies on benchmarking the operations of a data parallel language (NESL), fitting a linear equation to the data, and applying the model to a running program. The memory hierarchy is modelled in a simple way with different cost models applying depending on whether or not the data is likely to fit

inside the cache. He reported errors of about 33% in predictions of performance on Sun workstations and the CM-5. Cache conflicts in the SGI Indigo workstation meant that his method would only predict performance to within a factor of about 15. Saavedra developed a micro benchmark approach to characterise the low level performance of the KSR1 memory system [39].

1.1.16 Post mortem

If performance analysis is done at all by programmers at the moment, they are most likely to use one of the post mortem trace analysis tools. Examples include Paragraph [16], Pablo [37], Vispad [20], the Upshot tool included with MPICH [31] and XMTV included with the LAM implementation of MPI [7].

They work by instrumenting a program with tracing commands, and generating a trace file with time stamps.

Post mortem techniques measure one run of the performance on one machine (and say nothing about the performance on other machines, or with different data sizes).

1.1.17 Sequential code analysis techniques

Tools exist for optimisting code on sequential machines, such as `prof`, `gprof`, the `SPARCworks analyzer` etc. These measure figures such as the number of times each function is called and the percentage of time spent in each. MacDonald [27] describes methods for analytical predictions of sequential code execution times.

1.2 Thesis overview

The methods described in this thesis are based on the standard message passing model MPI [13]. Message passing was selected for two reasons; it is more standardised than shared memory and its explicit parallel nature cries out for a design technique.

The performance of the interface is characterised by a routine which generates datasheets for each MPI function. This characterisation of the primitives is performed in the same spirit as Toledo's benchmapping approach for data parallel programs [42]. The characterisation takes the form of an equation for each MPI function, along with graphs displaying the data from which the equation was derived. Chapter 3 describes the routines used to characterise the performance of parallel building blocks and generate the data sheets. (figure 1.1).

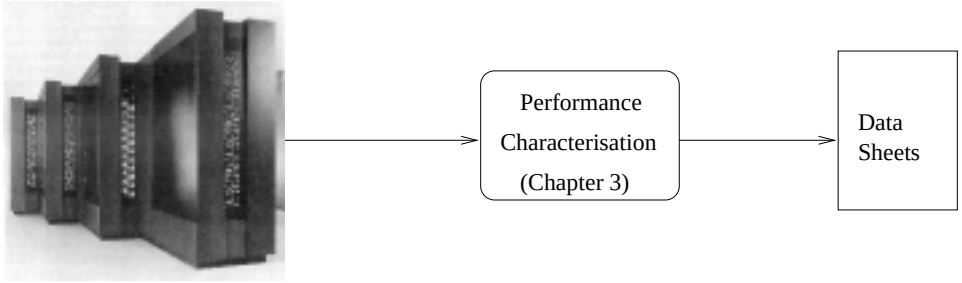

Figure 1.1: Performance predictions are based on automatically generated data-sheets for an architecture.

These data sheets may be used as they are for initial design. A simple calculating utility for evaluating the equations for given parameters was written to help with this.

Pencil and paper analysis becomes tedious and time consuming for all but the simplest program, so three computer aided techniques with increasing levels of sophistication were developed to help use the data sheets for practical development. Figure 1.2 gives an overview.

The first technique uses the data sheet results with a graphing package for rapid evaluations of the scalability of programs. This approach is presented in chapter 4. This allows experimentation at an early stage into the top level behaviour of algorithms.

A finer grain approach is presented in chapter 5. This uses the standard profiling mechanism of MPI to insert timings evaluated from the data sheets, a technique which is as easy to use as normal profiling. This allows automatic calculation of the expected timing diagrams, and copes with data dependent timings, something which compile time analysis techniques cannot handle. The results are compared with timing diagrams produced from standard profiles to assess the accuracy which can be expected from the approach. A similar approach applied to the simpler BSP model was described by Hill et al [17].

Chapter 6 investigates a simulation tool which extends the approach above to handle non deterministic programs as well as deterministic ones. It also permits inclusion of detailed hardware models in addition to the data sheet models.

Chapter 2 describes the experimental techniques used to evaluate the various approaches. A suite of problems requiring a variety of parallel implementation strategies was chosen to test the ease of use and accuracy of the design techniques.

These strands are drawn together in the conclusion (chapter 7) which evaluates the techniques and presents plans for future development.

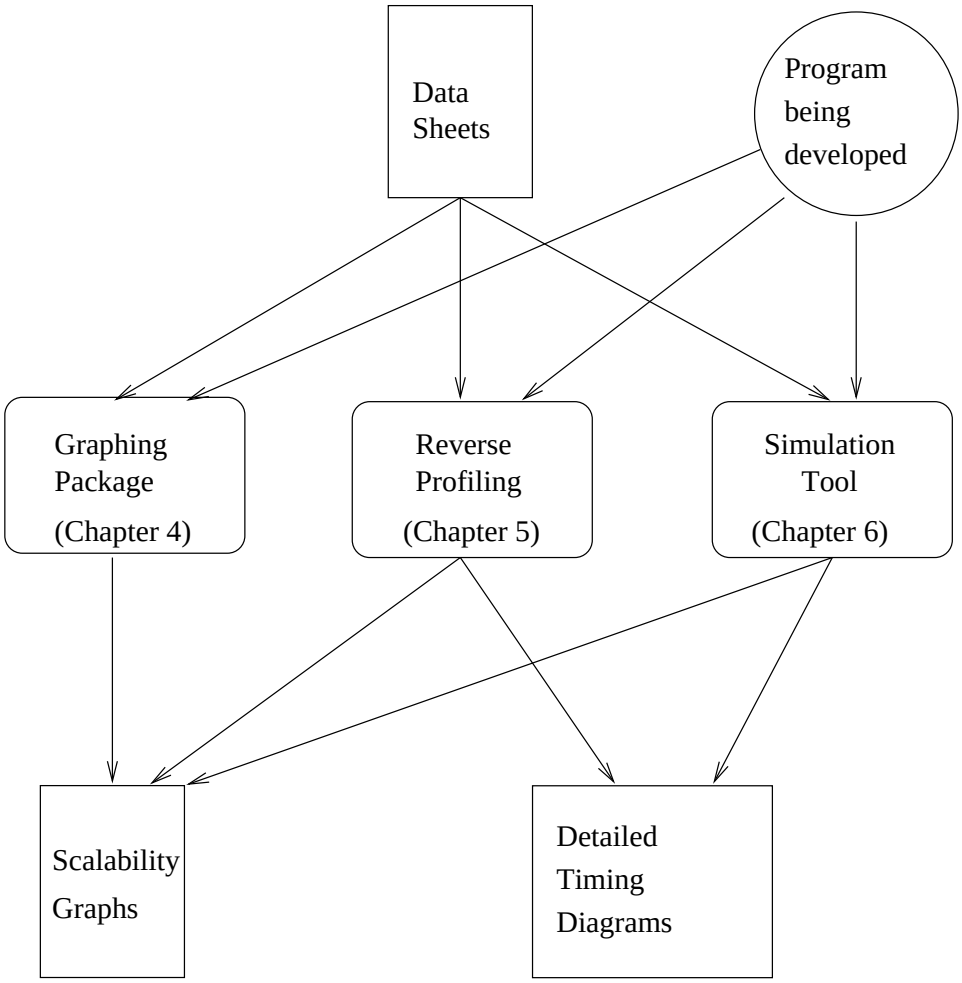

Figure 1.2: Three techniques for using the datasheets to help design programs.

Bibliography

[1] A.Dunlop, E.Hernandez, O.Naim, T.Hey, and D.Nicole. A Toolkit for Optimising Parallel Performance. In *HPCN International Conference Milan*, number 919 in LNCS, pages 548–553. Springer-Verlag, May 1995.

[2] A. Agarwal. Limits on interconnection network performance. *IEEE Trans. on Par. & Dist. Sys.*, 2(4), October 1991.

[3] R. Aversa, A. Mazzeo, N. Mazzocca, and U. Villano. The PS Project: development of a simulator of PVM applications for Heterogeneous and Network Computing. In Innes Jelly and Ian Gorton, editors, *Software Engineering for Parallel and Distributed Systems : Proceedings of the First IFIP TC10 International Workshop on Parallel and Distributed Software Engineering*. IFIP, Chapman and Hall, March 1996.

[4] E.A. Brewer, C.N. Dellarocas, A. Colbrook, and W.E. Weihl. PROTEUS: A high performance parallel-architecture simulator. Technical Report MIT/LCS/TR-516, MIT Laboratory for Computer Science, September 1991.

[5] E.A. Brewer and W.E. Weihl. Developing parallel applications using high-performance simulation. In *Proceedings of 1993 Workshop on Parallel and Distributed Debugging*. San Diego, CA, 1993.

[6] J. Buck, S. Ha, E. Lee, and D. Messerschmitt. Ptolemy: a framework for simulating and prototyping heterogenous systems. *Int. Journal of Comp. Sim.*, August 1992.

[7] G.D. Burns, R.B. Daoud, and J.R. Vaigl. LAM: An Open Cluster Environment for MPI. In *Supercomputing Symposium '94*, Toronto, Canada, June 1994.

[8] L.J. Clarke. PUL concepts I. Technical Report EPCC-KTP-PUL-CONC-I, Edinburgh Parallel Computing Centre, University of Edinburgh, 1991.

[9] M.I. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. Pitman & MIT Press, 1989.

[10] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. San Diego, CA, May 1993.

[11] M.A. Driscoll and W.R. Daasch. Accurate predictions of parallel program execution time. *Journal of Parallel and Distributed Computing*, 25(1), February 1995.

[12] T. Fahringer and H.P. Zima. A static parameter based performance prediction tool for parallel programs. In *Proceedings of the 7th ACM International Conference on Supercomputing*, July 1993.

[13] Message Passing Interface Forum. MPI: A Message Passing Interface. Technical report, University of Tennessee, June 1995.

[14] J.L. Gustafson. Reevaluating Amdahl's Law. *Communications of the ACM*, 31(5):532–533, May 1988.

[15] F. Hartleb. Graph models for Performance Evaluation of Parallel Programs. In A.Bode and M.Dal Cin, editor, *Parallel Computer Architectures : Theory, Hardware, Software, Applications*, number 732 in LNCS. Springer-Verlag, 1993.

[16] M.T. Heath and J.A. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, pages 29–39, Sept 1991.

[17] J.M.D. Hill, P.I. Crumpton, and D.A. Burgess. Theory, practice and a tool for bsp performance prediction. Technical Report TR-4-96, Oxford University Programming Research Group, Feb 1996.

[18] J. Hillston. PEPA: Performance Enhanced Process Algebra. Technical Report CSR-24-93, Department of Computer Science, University of Edinburgh, March 1993.

[19] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[20] A. Hondroudakis and R. Procter. The design of a tool for parallel program performance analysis and tuning. In *Proceedings of IFIP WG10.3 Working*

Conference on Programming Environments for Massively Parallel Distributed Systems. IFIP, April 1994.

[21] H.Wabnig, G.Haring, D.Kranzmuller, and J.Volkert. Communication Pattern Based Performance Prediction on the nCUBE-2 multiprocessor System. In *CONPAR*, pages 41–52, Linz, Austria, Sept 1994. Springer-Verlag.

[22] I.E. Jelly and I. Gorton. The PARSE project. In Innes Jelly and Ian Gorton, editors, *Software Engineering for Parallel and Distributed Systems : Proceedings of the First IFIP TC10 International Workshop on Parallel and Distributed Software Engineering*, pages 271–276. IFIP, Chapman and Hall, March 1996.

[23] J.P.Singh and J.L.Hennessy. Finding and exploiting parallelism in an ocean simulation program: experience, results and implications. *Journal of parallel and distributed computing*, 15:27–48, 1992.

[24] K.B. Kenny and K. Lin. Building flexible real-time systems using the flex language. *IEEE Computer*, 24(5):70–78, May 1991.

[25] P.J.B King. *Computer and Communication Systems Performance Modelling*. Prentice-Hall, 1990.

[26] De-Ron Liang and S.K. Tripathi. Performance prediction of parallel computation. In *Proc 8th IPPS*, pages 625–629. CS Press, 1994.

[27] N. MacDonald. Predicting execution times of sequential scientific kernels. In Christoph W. Kessler, editor, *Automatic Parallelization*, pages 32–44. Vieweg, 1994.

[28] V. Mak and S. Lundstrom. Predicting performance of parallel computations. *IEEE trans. on par. & distr. sys.*, 1(3), July 1990.

[29] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[30] F.M. Moller. A temporal calculus of communicating systems. Technical Report ECS-LFCS-89-104, Univ. of Edinburgh Dept. of Computer Science, 1989.

[31] MPICH - A Portable Implementation of MPI. `http://www.mcs.anl.gov/Projects/mpi/mpich/`, 1996.

[32] NASA AMES Research Center. AIMS: An Automated Instrumentation and Monitoring System.
`http://www.nas.nasa.gov/NAS/Tools/Projects/AIMS/`, 1995.

[33] C.Y. Park and A.C. Shaw. Experiments with a program timing tool based on source-level timing schema. *IEEE Computer*, 24(5):48–57, May 1990.

[34] J.L. Peterson. *Petri-net theory and the modelling of systems*. Prentice Hall, 1981.

[35] P.Pouzet, J.Paris, and V.Jorrand. Parallel Application Design: The Simulation Approach with HASTE. In W.Gentzsch and V.Harms, editors, *High Performance Computing and Networking II : Networking and Tools*, number 797 in LNCS, pages 379–393. Springer-Verlag, April 1994.

[36] P. Puschner and C.H. Koza. Calculating the maximum execution time of real-time programs. *J. Real-Time Systems*, 1(2):159–176, 1989.

[37] D.A. Reed, R.A. Aydt, R.J. Noe, P.C. Roth, K.A. Shields, B. Schwartz, and Luis F. Tavera. Scalable Performance Analysis: The Pablo Performance Analysis Environment. In Anthony Skjellum, editor, *Proceedings of the Scalable Parallel Libraries Conference*. IEEE Computer Society, 1993.

[38] R.W.Numrich, P.L.Springer, and J.C.Peterson. Measurement of Communication Rates on the Cray T3D Interprocessor Network. In W.Gentzsch and V.Harms, editors, *High Performance Computing and Networking II : Networking and Tools*, number 797 in LNCS, pages 150–157. Springer-Verlag, April 1994.

[39] R.H. Saavedra, R.S. Gaines, and M.J. Carlton. Micro benchmark analysis of the KSR1. In *Supercomputing '93*, Portland, Oregon, 1993.

[40] S.R. Sarukkai. Scalability analysis tools for SPMD message-passing parallel programs. In *MASCOTS '94: Proceedings of the 2nd International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, January 1994.

[41] Sivan Toledo. *Quantitative Performance Modelling of Scientific Computations and Creating Locality in Numerical Algorithms*. PhD thesis, Massachusetts Institute of Technology, 1995. Also available as Technical Report MIT-LCS-TR-656.

[42] Sivan Toledo. Performance prediction with benchmaps. In *Proceedings of the 10th International Parallel Processing Symposium, Honolulu, Hawaii*, pages 479–484, Los Alamitos, California, April 1996. IEEE, IEEE Computer Society Press.

[43] L.G. Valliant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.