Algorithms, Architectures and Models of Computation

R.N. Ibbett, T. Heywood, M.I. Cole, R.J. Pooley, P. Thanisch, N.P. Topham, G. Chochia, P.S. Coe, P.E. Heywood

Computer Systems Group
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ, UK.

Abstract

The ALgorithms, Architectures and MOdels of computation (ALAMO) project at the University of Edinburgh aims to investigate the scalability and efficiency with which the Hierarchical PRAM model of parallel computation may be implemented on realistic parallel architectures. This investigation is being performed using a programming language H-FORK together with HASE, a tool for the hierarchical design and simulation of computer architectures. This paper outlines the background to the project and the motivation for it.

1 Introduction

The Computer Systems Group at Edinburgh has research interests which include theoretical models of parallel computation, mapping and scheduling of processes to processors, the design of processor and multiprocessor architectures, the evaluation of systems through simulation, and object oriented databases. The ALAMO project ('ALgorithms, Architectures and MOdels of computation: simulation experiments in parallel systems design'), brings a number of these interests together. The project aims to address the first and fourth of the four 'Grand Challenge Problems in Computer Architecture' identified by the Purdue Workshop on Grand Challenges in Computer Architecture for the Support of High Performance Computing [23]. These are:

1. Idealised Parallel Computer Models

> "The model of parallel computation is fundamental to progress in high performance computing because the model provides the interface between parallel hardware and parallel software. It is the idealisation of computation that computer architects strive to support with the greatest possible performance. The model is the specification of the computational engine that language and operating systems designers can assume as they seek to enhance the power and convenience of parallel machines. [It is essential to identify] a small number of 'fundamental' models of parallel computation that serve as a natural basis for programming languages and that facilitate high performance hardware implementations."

4. Infrastructure for Prototyping Architectures

> "Given that computer generations change every 2 to 3 years, new ideas on architecture must be evaluated and prototyped quickly. Prototype development involves not only hardware, but also software in the form of compilers and operating systems. An infrastructure is needed to facilitate the study of the effects of new hardware technologies and machine organisations against different application requirements. This computer architecture challenge is to develop sufficient infrastructure to allow rapid prototyping of hardware ideas and the associated software in a way that permits realistic evaluation."

By combining work on these Grand Challenges we benefit from interaction between these two complementary areas of research. Specifically, we aim to identify the characteristics of hardware architectures that are capable of providing *cost-effective scalable* support for the Hierarchical PRAM (H-PRAM) [16, 17] model of parallel computation, which is a strong 'Idealised parallel computer model' candidate. In other words, the goal is to determine the properties of cost-effective (cheapest possible) systems based on scalable architectures and efficient support for the H-PRAM model. Design, simulation, and evaluation of hardware architectures involves the development and use of our Hierarchical computer Architecture design and Simulation Environment (HASE) [19].

This paper overviews the background, reasoning and motivation behind this project and summarises its current status. Section 2 describes the H-PRAM model. The H-PRAM's technical definitions are primarily concerned with parallel algorithm design and analysis, and thus the model needs an overlying programming model in order that application programs can be written and experiments run. Section 3 discusses programming models overlying the H-PRAM. Section 4 provides a brief description of the mapping strategy which underlies implementation of the H-PRAM on mesh-connected architectures. Section 5 describes the framework within which experiments are performed and the Hierarchical computer Architecture Design and Simulation Environment (HASE) used to simulate the architectures on to which H-PRAM programs are compiled. Finally, Section 5 covers the experimental method used to obtain and interpret results.

2 The H-PRAM Model of Computation

In parallel computing, a model of computation has a difficult task. It must mediate between the conflicting requirements of abstraction (ease of use) for algorithm design/analysis, and cost/resource details of realistic architectures.

The PRAM model is accepted as a good tool for parallel algorithm research, but fails to represent, even abstractly, realistic architectures since it ignores communication and synchronisation costs. In recognition of this fact, many other models have been proposed (see [9] and the references therein), none of which has yet gained general acceptance. In various ways they try to balance simplicity of use and reflectivity of architectural costs, and may be roughly classified into three types:

- PRAMs with locality (simplicity = shared memory, reflectivity = locality), e.g. H-PRAM [16, 17], YPRAM [24], LPRAM [2], BPRAM [1].
- Global message passing models (simplicity = global address space, reflectivity = message passing), e.g. BSP [25], LogP [7], and many more.
- 'Adjusted' PRAMs with 'architectural' features, such as asynchrony (many variants of Asynchronous PRAMs) or memory queues to represent contention (QRQW PRAM [10]).

Comparisons and critical analyses of various models motivating the study of the H-PRAM may be found in [14, 16].

The H-PRAM is a model which balances simplicity of use, and reflectivity of the costs and resources of architectural models employing direct networks (*i.e.* a 'good' H-PRAM algorithm will translate to a 'good' algorithm on a direct network model). It is the most general of the 'PRAM with locality' type models, *i.e.* has the least restrictions on partitioning the logical memory space into sub-memories. The H-PRAM uses the PRAM as a sub-model, allows the utilisation of general locality, and enforces determinate (testable) computation in the face of asynchrony, thus providing a basis for bridging theory and practice of parallel computing. It satisfies almost all of the goals of the 'Idealised parallel computer models' Grand Challenge. The remaining goal of cost-effective implementation of the model is that for which the ALAMO project is striving.

Structurally, the H-PRAM is a PRAM which can recursively partition itself into sub-PRAMs, giving rise to a hierarchy of synchronous PRAMs which operate asynchronously from each other. Sub-PRAMs may be any of the standard types, i.e. EREW, CREW, CRCW. A P-processor H-PRAM is thus a P-processor PRAM whose instruction set is *extended* by a partition instruction:

partition { p_1: Algorithm-1 (parameter-list);
p_2: Algorithm-2 (parameter-list);
...
p_q: Algorithm-q (parameter-list) }

where the p_i are positive integers with $\sum_{i=1}^{q} p_i = P$.

The operation partitions the P processors of the original PRAM into disjoint subsets of p_i processors running the specified algorithms. These sub-PRAMs operate asynchronously from each other. They synchronise after having finished their algorithms, *i.e.* at the termination of the partition instruction. The sub-PRAM sub-algorithms may themselves have partition instructions, giving rise to a recursive partitioning of the H-PRAM. The hierarchical structure of the overall computation can be represented by a series–parallel graph, where 'forks' correspond to the start of a partition instruction and 'joins' correspond to the termination of them.

The H-PRAM, algorithmic issues, and relationships to architectures are covered in detail in [16, 17], so these issues will not be discussed here. The following section considers programming models for overlying the H-PRAM computational model.

3 Programming Models for the H-PRAM

The existing body of work on the H-PRAM is predominantly concerned with the asymptotic analysis of algorithms, defined in conventional parallel pseudo-code. This pseudo-code suits its purpose well, but its informality is inappropriate for this project. If we are to simulate the behaviour of implementations of realistic programs then we will require a more precisely defined language in which to express them. Furthermore, the full power of the H-PRAM model lies in its ability to exploit locality which emerges dynamically in data dependent patterns. Existing H-PRAM analyses deal mainly with *oblivious* algorithms (for example the FFT [17]), in which the partition structure is a feature of the problem rather than of specific instances thereof. The only way to capture the dynamic evolution of non-oblivious computations is to run (or at least emulate) them[1]. The implied automation necessitates a more conventional programming notation.

3.1 The FORK language

A similar situation existed in the early days of PRAM algorithm design. The languages FORK [11] and its successor FORK95 [21] were designed to act as standards for PRAM algorithm description and as real languages intended for implementation. FORK95 is a block-structured C-like language in which statements are executed by a dynamically evolving hierarchy of groups of processes. Its semantics define the ways

[1] An attempt was made to analyse a non-oblivious N-body algorithm on the H-PRAM, but the bounds were extremely rough due to simplifying assumptions that were made in reaction to the difficulty of parallel, recursive probabilistic analysis.

in which the group hierarchy evolves. Informal PRAM notions of memory sharing and synchronisation are defined precisely with respect to this hierarchy. The hierarchy is affected by the execution of conditional statements and, more interestingly, by the execution of `fork` statements. `fork` allows the rearrangement of existing processes into new groups, which may then be assigned distinct tasks, still sharing access to common memory locations. Conventional rules of block-scoping, together with the ability to declare variables as `shared` or `private` (relative to the group hierarchy) combine to define the environment seen by a process and as a framework within which the traditional refinements of PRAM memory policy (CRCW, CREW and so on) can be applied.

3.2 An H-PRAM variant of FORK

We use a FORK-like language to describe H-PRAM computations. Our prototype language amends the original FORK model in several ways. Most importantly, we introduce a means of expressing the partitioning of memory which lies at the core of the H-PRAM model. Note that the existing `fork` is essentially a convenient syntactic sugaring which facilitates the expression of algorithms in a particular style. The shared memory space available to processes in the newly created groups is not immediately affected. In contrast, the spirit of the H-PRAM is that greater efficiency (and scalability) may be obtained by explicitly and exclusively dividing the shared memory amongst groups of processors. Thus we introduce a variant of `fork`, called `hfork` in which each new group must explicitly lay claim to those shared variables to which it requires exclusive access. It is a compile-time checking requirement to ensure that these claims are indeed mutually exclusive. Such a requirement has implications for the use of aliasing which must be considered carefully. A simple notation has been developed which allows the expression of the division of bulk data structures, such as arrays, similar in concept to occam's 'abbreviations' or Fortran 90's 'sections'. We are also considering some form of 'permute-and-partition' notation (as might be useful in the standard parallel FFT computation).

These extensions define a simple 'vanilla' H-PRAM language, which we develop in two directions. Firstly, for pragmatic reasons, we pare the language down to the minimum required to express realistic computations for emulation, allowing us to isolate and concentrate upon specifically H-PRAM related issues, as opposed to those which complicate both H-PRAM and PRAM implementations. For similarly pragmatic reasons we have initially chosen to restrict the power of the procedural abstraction mechanisms allowed.

Having developed clear insights into the concepts embodied by this simple "H-FORK" language, we will expand back towards a fuller language, with the re-introduction of conventional PRAM techniques (such as multi-threading to exploit parallel slackness).

3.3 Skeletal Extensions

Our second aim is to consider the extension of H-FORK to embody other possible notions of partition. In this work we will be guided by the principles of the 'skeletal' approach to parallel algorithm design. This model [6, 8] proposes that parallel programming systems be based around a collection of implicitly parallel program 'skeletons', each embodying the computational and communications structure of a class of algorithms, with parallel implementation of the structure being provided by the system. The parallel programming task is then reduced to the problem of selecting, specialising and composing appropriate skeletons.

As an algorithmic model, the H-PRAM is clearly a flexible 'divide & conquer' machine, and so formulation of a range of more or less prescriptive divide & conquer skeletons should be straightforward. More interestingly, a wide range of other skeletons has been proposed in the literature, ranging from the simple and general to the complex and specialised. We propose to investigate the efficacy with which these and perhaps other new structures can be executed on the H-PRAM and expressed in extensions of H-FORK. Most obviously, it seems that many such structures (for example, consider simple pipelines) will require the ability to exchange information between sub-PRAMS in restricted patterns. These patterns will no doubt lead to the definition of new forms of `fork`.

4 Mapping the H-PRAM to Architectures

It is intended to investigate the implementation of H-FORK programs and their mapping on to architectures whose interconnection topology is a mesh. Experimentation with architectural features will address processor design, router and synchroniser design, I/O port design, etc., but not topology. We are concentrating on the mesh since this is a cost-effectively scalable topology, both with current technology and with future technology presumably operating at the physical limits [3] [14].

Work so far has addressed the structural mapping of the H-PRAM to a mesh together with preliminary simulations of the resulting routing patterns [15, 4, 5]. In order to give a small taste of the strategy, we outline here a mesh indexing scheme which allows a very simple dynamic mapping. Indeed, it is being used because of the simplicity.

The H-PRAM model does not impose any constraints on the number and sizes of the sub-PRAMs created in a partition step other than those imposed by the available number of processors. Thus, we need to be able to partition into any number of arbitrarily-sized sub-meshes, and to do this dynamically at run-time. Though the sub-meshes will not generally be square, each p-processor sub-mesh must have diameter $O(\sqrt{p})$, and permit PRAM simulation on it in $O(\sqrt{p})$ mesh routing steps per PRAM step. The Peano indexing scheme [20] allows us to meet these criteria.

The Peano scheme indexes a mesh such that the processors in block A of Figure 1 get smaller indices than those in block B, further than those in C, and D (of course,

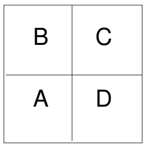

Figure 1: Peano scheme: processor indexing

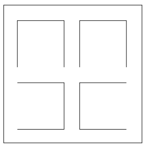

Figure 2: Peano scheme: recursive pattern

symmetric variants are equivalent). The scheme is nested recursively according to the pattern in Figure 2 such that an 8×8 mesh is indexed as follows:

22	23	26	27	38	39	42	43
21	24	25	28	37	40	41	44
20	19	30	29	36	35	46	45
17	18	31	32	33	34	47	48
16	13	12	11	54	53	52	49
15	14	9	10	55	56	51	50
2	3	8	7	58	57	62	63
1	4	5	6	59	60	61	64

The trick with the scheme is that any two nodes whose indices differ by d are at Manhattan distance $O(\sqrt{d})$ in the array. For further details see [4, 5, 15], in which we tie down the constant factor to at most $\frac{3}{2}$ in theory, and demonstrate very encouraging results in practice. We are also working to tackle the problem of implementing the dynamically evolving patters of synchronization within sub-groups of the processors implied by data dependent conditionals.

5 Experimental Framework

The primary goal is to demonstrate that it is possible to support an H-PRAM inspired programming model on a physical mesh architecture in a way that is *cost-effectively scalable*, in contrast with other models of a similar level of abstraction (eg PRAM). Thus we are interested in the comparing the results of changes in the characteristics of our system (*e.g.* architectural details, hardware mechanisms *vs.* software mechanisms), as opposed to comparison with others. Limited absolute comparison with either straight PRAM (using the same implementation) or some benchmark straightforward program on the same hardware may be desirable, but is really a side-issue.

It is our intention to produce results by simulation rather than analysis of the type performed in [17]. This has the key advantage of enabling the introduction of non-oblivious algorithms, which exploit the full power of the H-PRAM's dynamic flexibility. Clearly computational resources will limit the extent to which full, detailed simulation at a low level can be performed, and so careful abstraction will be required.

We will investigate scalability of each architectural structure by plotting performance *versus* scale curves, and will repeat this across a range of architectures (all mesh connected, but with varying degrees of hardware/software support for system mechanisms, *e.g.* routing, synchronising, etc.) to quantify 'cost-effectiveness'. We can then produce 'cost *versus* scalability' plots.

5.1 The Simulation Environment - HASE

HASE [19] addresses the fourth Purdue Grand Challenge. It allows for the rapid development and exploration of computer architectures at multiple levels of abstraction, encompassing both hardware and software.

Using HASE, designers can create and explore architectural designs at different levels of abstraction through a graphical interface based on X-Windows/Motif (figure 3 shows an example of a design window). Designs can be simulated in such a way that different parts of the architecture can be simulated at different levels of abstraction. This involves the creation of simulation code appropriate to each level. Thus in a multiprocessor, some processors could run code held in their (simulated) memory, while in others the code could be abstracted to *computation* and *communication* (send and receive) sections; simple timings can then be used to represent each computation section, with communication events following the appropriate protocol.

The components of a computer system lend themselves naturally to being modelled as objects, so HASE has been implemented in an object oriented language. Furthermore, many complex systems of interacting components can be more easily understood as a picture rather than as words, and in a computer architecture the dynamic behaviour of systems is frequently of interest. Thus the graphical interface allows users to view the results of simulation runs though animation of the design window.

HASE includes a design editor and object libraries appropriate to each level of abstraction in the hierarchy, plus instrumentation facilities to assist in the validation of the model. Thus the output from a simulation run can be used to animate the design drawings, or can be used to investigate the performance metrics of hardware architectures.

The HASE project has pioneered the use of object-oriented database technology in discrete-event simulation [12]; HASE uses an object oriented database management system (ObjectStore) to make the design objects and the entity library persistent. For each architecture model HASE allows many experiments with varying parameters to be performed. The database facilities provided through HASE manage not only the

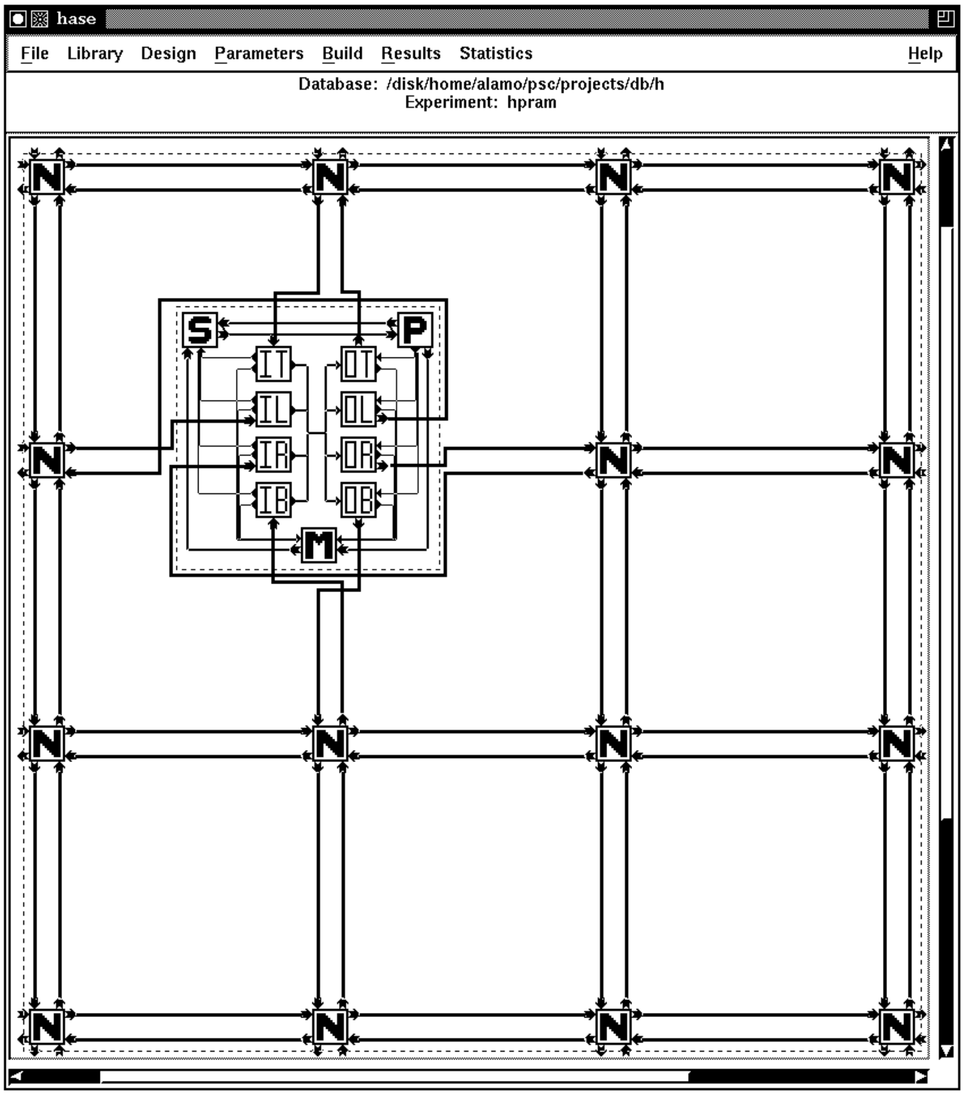

Figure 3: A HASE Design Window

results of each experiment, but also their relationship to the state of the architecture model that produced these results, including all input and output parameters and their values during the experiment. Work is in progress to provide *model exploitation facilities* based on [18].

HASE is being used not only as part of the ALAMO project, but also in a number of other architecture projects, including an evaluation of multiprocessor interconnection networks, parallel performance prediction, a simulation of the Stanford DASH architecture, and an on-line teaching system for computer architecture. Use of the client/server architecture of the database system to create a distributed simulation environment for HASE is also being investigated [13].

In the ALAMO project HASE will be used to design and simulate architectures to support the H-PRAM model, with the goal of determining the properties of cost-effective (cheapest possible) systems based on scalable architectures and efficient support of the H-PRAM model.

5.2 Experimental Method

The target architectural components are being created, tested and installed in the HASE library. We expect to investigate a variety of parallel system configurations, so we will generate a variety of interconnection components as well as processors and memory modules. The intention in HASE is to allow different components of a system to be simulated at different levels of abstraction and to be able to replace detailed sub-models with aggregated equivalents, in order to be able to trade off simulation time against the level of measurement information required.

This last simplification will also require great care and pre-analysis of behaviour. How far details of the behaviour of sub-models can be ignored when aggregating is, in general, difficult to decide.

From the algorithmic viewpoint, the key characteristics of an H-PRAM computation are its evolving group structure, and the number and extent of the implied synchronisations and data re-organisations. One reduction in the amount of detailed simulation required within HASE has been achieved by building a separate H-FORK emulator which, given a program and data set, will return a trace of the abstract evolution of the corresponding H-PRAM computation, annotated with information about synchronisations, regroupings and so on. These traces will then drive HASE computations, whose concern will be to quantify the effect of emulating the given H-PRAM structure on a range of target architectures, but without explicitly having to work through the detailed execution of instructions within the H-FORK processes.

The benefits of this approach can be increased further by deriving abstract workloads for the architectures, where stochastic models of workload replace explicit traces. Such an abstraction requires careful analysis of more detailed behaviour to identify pathological sequences of instructions leading to exceptional behaviour not preserved in averaging.

Acknowledgements

The ALAMO project is supported by the UK Engineering and Physical Sciences Research Council.

References

[1] A. Aggarwal, A. K. Chandra, and M. Snir, "On Communication Latency in PRAM Computations", *ACM Symposium on Parallel Algorithms and Architectures*, pp. 11-21, 1989

[2] A. Aggarwal, A. K. Chandra, and M. Snir, "Communication Complexity of PRAMs", *Theoretical Computer Science*, Vol. 71, pp. 3-28, 1990.

[3] G. Bilardi, F. P. Preparata,"Horizons of Parallel Computation". In "Future Tendencies in Computer Science, Control and Applied Mathematics. Int. Conf. on the Occasion of the 25th Anniversary of INRIA", A. Bensoussan, J-P. Verjus (eds.), LNCS 653, pp. 155-174, 1992.

[4] Chochia, G., Cole, M., and Heywood, T., "Implementing the Hierarchical PRAM on the 2D Mesh: Analyses and Experiments", *Proc. of the Seventh IEEE Symposium on Parallel and Distributed Processing*, San Antonio, 1995.

[5] Chochia, G., Cole, M., and Heywood, T., *"Lower Bounds on Average Time for Random Destination Mesh Routing and Their Utility as Performance Predictors for PRAM Simulation"*, University of Edinburgh, Computer Systems Group, ECS-CSG-18-95, 1995.

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

[7] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramanian, T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation", *Proc. 4th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, May 1993

[8] J. Darlington et al., "Parallel Skeletons for Structured Composition", *Proceedings of Principles and Practice of Parallel Programming '95.*

[9] P. Gibbons, "Models of Parallel Computation: An Overview", DIMACS Tech. Report 93-87, pp. 8-10 and 59-65, 1993.

[10] P. Gibbons, Y. Matias, and V. Ramachandran, "The QRQW PRAM: Accounting for Contention in Parallel Algorithms", *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 1994.

[11] T. Hagerup, A. Schmitt & H. Seidl, "FORK: A High-level Language for PRAMs", *Future Generation Computer Systems*, vol. 8, pp. 379-393, 1992.

[12] P.E. Heywood, P. Thanisch and R. Pooley "Object-Oriented Database Technology for Simulation Environments", *Proceedings UKSS '95*, Vol 2, North Berwick, April 1995.

[13] P.E. Heywood, G. MacKechnie, R.J. Pooley and P. Thanisch, "Object Oriented Database Technology Applied to Distributed Simulation", *Proc. EUROSIM Congress '95*, Vienna, Elsevier, 1995.

[14] T. Heywood and C. Leopold, "Models of Parallelism". In *Abstract Machine Models for Highly Parallel Computers*, J.R. Davy and P.M. Dew (eds), Oxford Univ. Press, 1995.

[15] T.Heywood and C. Leopold, "Dynamic Randomized Simulation of Hierarchical PRAMs on Meshes", *Aizu Int. Symp. on Parallel Algorithm/Architecture Synthesis*, March 1995.

[16] T. Heywood & S. Ranka "A Practical Hierarchical Model of Parallel Computation I: The Model", *Journal Parallel and Distributed Computing* 16, pp. 212-232, 1992.

[17] T. Heywood & S. Ranka "A Practical Hierarchical Model of Parallel Computation II: Binary Tree and FFT Algorithms, *Journal Parallel and Distributed Computing* 16, pp. 233-249, 1992.

[18] J. E. Hillston, "A Tool to Enhance Model Exploration", *Proc. Sixth International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Edinburgh, 1992.

[19] R.N. Ibbett, P.E. Heywood and F.W. Howell "HASE: A Flexible Toolset for Computer Architects, *The Computer Journal*, to appear.

[20] C. Kaklamanis & G. Persiano, "Branch-and-Bound and Backtrack Search on Mesh-Connected Architectures", *Proc. 4th ACM SPAA*, 1990.

[21] C.W. Kessler & H. Seidl, "Fork95 Language and Compiler for the SB-PRAM", *5th International Workshop on Compilers for Parallel Computers*, 1995.

[22] R.J. Pooley, "The Integrated Modelling Support Environment, a new generation of performance modelling tools", *Computer Performance Evaluation Modelling Techniques and Tools*, Elsevier Science Publishers, Amsterdam, 1991.

[23] H.J. Siegel, S. Abraham, *et al* "Report of the Purdue Workshop on Grand Challenges in Computer Architecture for the Support of High Performance Computing", *Journal Parallel and Distributed Computing*, 16, pp. 199-211, 1992.

[24] P. de la Torre, C. P. Kruskal,"Towards a Single Model of Efficient Computation in Real Parallel Machines", *Proc. Parallel Architectures and Languages Europe PARLE*, LNCS, pp.6-24, 1991.

[25] L. G. Valiant,"A Bridging Model for Parallel Computation", *Communications of the ACM*, Vol. 33, No. 8, pp. 104-111, 1990