Optimizing Hardware Granularity in Parallel Systems

Thomas Kelly

PhD

The University of Edinburgh

1995

Declaration

This thesis has been composed by me and is, except where indicated, my own work.

Thomas Kelly. November, 1995

Abstract

In order for parallel architectures to be of any use at all in providing superior performance to uniprocessors, the benefits of splitting the workload among several processing elements must outweigh the overheads associated with this "divide and conquer" strategy.

Whether or not this is the case depends on the nature of the algorithm and on the cost:performance functions associated with the real computer hardware available at a given time.

This thesis is an investigation into the tradeoff of *grain* of hardware versus *speed* of hardware, in an attempt to show how the optimal hardware parallelism can be assessed.

A model is developed of the execution time T of an algorithm on a machine as a function of the number of nodes, N. The model is used to examine the degree to which it is possible to obtain an optimal value of N, corresponding to minimum execution time.

Specifically, the optimization is done assuming a particular base architecture, an algorithm or class thereof and an overall hardware cost.

Two base architectures and algorithm types are considered, corresponding to two common classes of parallel architectures: a shared memory multiprocessor and a message-passing multicomputer. The former is represented by a simple shared-bus multiprocessor in which each processing element performs operations on data stored in a global shared store. The second type is represented by a two-dimensional mesh-connected multicomputer. In this type of system all memory is considered private and data sharing is carried out using "messages" explicitly passed among the PEs.

Acknowledgements

The pursuit of this research has introduced me to many people to whom I am indebted. First, I am grateful to my employer, Motorola Ltd., which supported this work in its entirety. In particular, I want to thank John Barr, Jim Ritchie and Chip Shanley; without their support I would not have been able to complete my work. Their encouragement, and patience, will always be appreciated.

In the Department of Computer Science at Edinburgh I found a good deal of support, and a rich atmosphere of study. To my supervisors Professor Roland Ibbet, and David Rees I am especially grateful. I also learned much from fellow students of whom Tim Harris, Neil MacDonald and Mike Norman were outstanding.

I also had the opportunity to spend time at the Department of Computing Science in the University of Glasgow. There, especially, did I begin the change of mind required to move from engineering to science. Advice and encouragement were gratefully received from John O'Donnell, Professor Simon Peyton-Jones, Mary Sheeran, Rob Sutherland, Pauline Haddow and Hans Stadtler. A particular place in my thanks is reserved for Mohamed Ould-Khaoua whose own persistence as well as encouragement and friendship were and are appreciated.

So to my family, closest to my heart, and the ultimate reason for any of my endeavours. They, my three girls, have encouraged, supported and sustained me most. And my parents; the work presented here rests upon the base with which they provided me.

Finally, I reserve the greatest thanks till last. For me, the period of study covered by this thesis was an apprenticeship in thought. Much as I am glad for the increase in knowledge I have gained in various aspects of Computing Science, far dearer to me was the opening of my eyes to thought itself, and the receipt of courage to ask, and ask, and ask. And in that respect I had the honour to learn from Lewis MacKenzie at the University of Glasgow. Through his mind,

bested only by his humility, I finally learned really to begin to think. This thesis is dedicated to him, teacher and friend.

For Lewis M. MacKenzie

"The herring are flying high tonight"

Table of Contents

1. Introduction **1**

1.1 Problem Statement 2

1.1.1 Processors - An Illustration 2

1.1.2 Space Scaling 7

1.2 Motivation 9

1.2.1 Parallel System Design 9

1.2.2 Optimizing Device Portfolios 10

1.2.3 Tradeoffs in a Fixed Cost System 12

1.3 Related work 14

1.4 Approach 17

1.4.1 Thesis Outline 21

2. VLSI Cost:Performance **22**

2.1 Introduction 23

2.2 Processing Hardware 23

2.2.1 Cost and Price 25

2.2.2 Modelling IC Cost 25

2.2.3 Microprocessor Cost:Performance 38

2.3 Memory Hardware 44

2.3.1 Memory Device Cost:Performance – t_M 44

2.4 Interconnection Network Hardware 47

2.4.1 Router Cycle Time 48

2.4.2 Router Channel Width 49

2.5 Summary 51

3. A Shared-memory Multiprocessor **53**

3.1 Introduction 54

3.2 Machine Architecture 54

3.3 The Algorithm 56

3.3.1 Critical Path 56

3.4 Software Parameters 59

3.5 Hardware Parameters 60

3.5.1 Space Scaling 60

3.5.2 Shared Memory Performance 62

3.6 Summary 66

4. A Message-passing Multicomputer **67**

4.1 Introduction 68

4.2 Machine Architecture 68

4.3 The Algorithm 68

4.3.1 A Graph Model of Algorithms 69

4.3.2 Grid Algorithm 72

4.3.3 Critical Path 75

4.4 Software Parameters 77

4.5 Hardware Parameters 78

4.5.1 Memory Performance - t_{M_l} 78

4.5.2 Mesh Network Latency - t_ρ 87

4.6 Summary 94

5. Discussion **95**

5.1 Introduction 96

5.2 The Simulator 96

5.2.1 Verilog Simulator 97

5.2.2 Simulation Control 99

5.2.3 Performance 99

5.3 Simulation Results 101

5.3.1 Shared Memory Multiprocessor 101

5.3.2 Message-passing Multicomputer 102

5.4 Model Behaviour 103

5.4.1 Shared Memory Multiprocessor Model 103

5.4.2 Message Passing Multicomputer Model 119

5.5 Shared Bus Example 123

5.6 Summary 126

6. Concluding Remarks **127**

6.1 Summary and Conclusions 128

6.2 Suggestions for Future Research 130

6.2.1 Scalability and Modularity 132

A. Microprocessor Survey Data **134**

B. Simulation **.138**

B.1 Bus Simulation Results 139

B.2 Mesh Simulation Results 146

List of Figures

1–1 Effect of linear cost:performance - Basic DAG 3

1–2 Effect of linear cost:performance - Connected DAG 4

1–3 Effect of sub-linear cost:performance 6

1–4 Weighted DAG Section 8

2–1 Probed Wafer Costs 32

2–2 Package Costs (Source: Microprocessor Report, October 4, 1993) . . 35

2–3 Processor Cost:Performance (t_I) 41

2–4 Adjusted Processor Cost:Performance (t_I) 42

2–5 Processor Cost:Performance (millions of "Int92 instructions" per second) 43

2–6 Memory Cost:Performance (memory access time) 45

2–7 Memory Cost:Performance (memory speed) 46

3–1 Shared Memory Multiprocessor 55

4–1 Directed Acyclic Graph 71

4–2 Levelled DAG 73

4–3 Overlap Areas 79

4–4 Overlap in several dimensions 82

4–5 Effects of Overlap Areas on Total Capacity 85

4–6 Effects of Overlap Areas on Memory Device Cost 85

4–7 Effects of Overlap Areas on Access Time 86

4–8 Examples of k-ary n-cubes 87

4–9 Decreasing Probability Communication 90

5–1 Verilog Router Block Diagram 98

5–2 Simulation Control 100

5–3 Single Processor Instruction Execution Time (t_I) 105

5–4 Single Processor Performance 106

5–5 Memory Access Time versus Total Capacity 107

5–6 Memory Speed versus Total Capacity 107

5–7 Bus Model - Execution time sensitivity to i 109

5–8 Bus Model - Bus waiting sensitivity to i 110

5–9 Bus Model - Execution time sensitivity to μ 111

5–10 Bus Model - Bus waiting sensitivity to μ 112

5–11 Bus Model - Memory requirements for various space scaling exponents 114

5–12 Bus Model - Execution time sensitivity to space scaling 114

5–13 Bus Model - Bus waiting sensitivity to space scaling 115

5–14 Instruction execution times for various processor cost:performance coefficients 116

5–15 Processor performance for various coefficients 117

5–16 Bus Model - Execution time sensitivity to processor cost:performance 118

5–17 Mesh Model - Time for Computation Only 120

5–18 Mesh Model - Latency, t_ρ versus N 121

5–19 Mesh Model - Router Cycle Time, t_R, versus N 121

5–20 Mesh Model - Message Length versus N 122

5–21 Mesh Model - Router Width versus N 122

5–22 Mesh Model - Bytes per Message versus N 123

5–23 Bus Model - Optimal N versus i_c 124

5–24 Bus Model - Optimal N versus μ 125

B–1 Bus Experiment 1 - Low-cost Components 141

B–2 Bus Experiment 2 - Low-cost Components 141

B–3 Bus Experiment 3 - Low-cost Components 141

B–4 Bus Experiment 1 - Mid-range Components 143

B–5 Bus Experiment 2 - Mid-range Components 143

B–6 Bus Experiment 3 - Mid-range Components 143

B–7 Bus Experiment 1 - High-end Components 145

B–8 Bus Experiment 2 - High-end Components 145

B–9 Bus Experiment 3 - High-end Components 145

B–10 Mesh Experiment 1a - Low-cost Components 148

B–11 Mesh Experiment 1b - Low-cost Components 148

B–12 Mesh Experiment 2a - Low-cost Components 149

B–13 Mesh Experiment 2b - Low-cost Components 149

B–14 Mesh Experiment 3a - Low-cost Components 150
B–15 Mesh Experiment 3b - Low-cost Components 150
B–16 Mesh Experiment 3c - Low-cost Components 150
B–17 Mesh Experiment 4 - Low-cost Components 151
B–18 Mesh Experiment 1a - Mid-range Components 153
B–19 Mesh Experiment 1b - Mid-range Components 153
B–20 Mesh Experiment 2a - Mid-range Components 154
B–21 Mesh Experiment 2b - Mid-range Components 154
B–22 Mesh Experiment 3a - Mid-range Components 155
B–23 Mesh Experiment 3b - Mid-range Components 155
B–24 Mesh Experiment 3c - Mid-range Components 155
B–25 Mesh Experiment 4 - Mid-range Components 156
B–26 Mesh Experiment 1a - High-end Components 158
B–27 Mesh Experiment 1b - High-end Components 158
B–28 Mesh Experiment 2a - High-end Components 159
B–29 Mesh Experiment 2b - High-end Components 159
B–30 Mesh Experiment 3a - High-end Components 160
B–31 Mesh Experiment 3b - High-end Components 160
B–32 Mesh Experiment 3c - High-end Components 160
B–33 Mesh Experiment 4 - High-end Components 161

List of Tables

2–1 Processor Cost:Performance 40

5–1 Default Parameters for Bus Model 104

5–2 Default Parameters for Mesh Model 119

B–1 Bus Simulation Experiments – 1 of 3 140

B–2 Bus Simulation Experiments – 2 of 3 142

B–3 Bus Simulation Experiments – 3 of 3 144

B–4 Mesh Simulation Experiments – 1 of 3 147

B–5 Mesh Simulation Experiments – 2 of 3 152

B–6 Mesh Simulation Experiments – 3 of 3 157

List of Key Symbols

C_A = Assembly cost per die

C_F = Final test cost – the cost to test a packaged device

C_P = Cost of a single processor

C_S = Probed (sorted) wafer cost per die

G = Base memory requirements in mesh

I = Total computation instructions on critical path of mapped algorithm

M_l = Total number of local memory accesses on critical path

M_s = Total number of shared memory accesses on critical path

N = Number of processing elements

P = Number of points in shared-memory algorithm

R_L = Total cost (resource) of all links

R_M = Total cost (resource) of all memory devices

R_P = Total cost (resource) of all processors

R_R = Total cost (resource) of all router devices

T_{MP} = execution time of mapped algorithm

T_{SM} = Execution time on shared-memory machine

W_R = Router width

δ = Interaction depth

μ = Fraction of instructions requiring local memory access

ρ = number of message receives on critical path

σ = number of message sends on critical path

d = DAG depth

f = task fanout (mesh)

i_c = Instructions per point (computation only)

i_s = Number of instructions required to prepare and send a message

j = instructions per task in mesh algorithm

m = Local memory per processor

n = dimension of data in mesh alg.

n_M = Total amount of memory in system

n_c = Number of physical channels per router

n_{M_l} = Total amount of local (private) memory

n_{M_s} = Total amount of shared memory

s = number of iterations in mesh algorithm

t_I = Single instruction execution time

t_M = Simple memory access time (i.e. ignoring contention)

t_R = Router cycle time

t_{M_l} = Time for local memory access

t_{M_s} = Time for shared memory access

t_ρ = time to receive a message

t_σ = time to send a message

y_A = Assembly yield

y_F = Final test yield – the fraction of packaged devices which pass final test

y_S = Probe yield – the fraction of probed die which pass probe test.

Chapter 1

Introduction

1.1 Problem Statement

It is not obvious that a parallel computer is always better than a uniprocessor. In general, it is not obvious that more processors are always better than fewer. If the cost of hardware is fixed, then the number of processing elements becomes one of several inter-related design parameters, and must undergo optimization with the others.

For a fixed cost, and for a given architecture, a number of actual designs are possible, ranging from one expensive processing element (PE) to several of a lower cost. The criteria governing which number of PEs is optimal are the subjects of study of this thesis. The main goal is to show how a model for optimizing N, the number of PEs, may be developed.

1.1.1 Processors - An Illustration

If the aggregate performance of all processing elements in a fixed cost system was independent of the number of elements, then parallelism would have little to offer computer architects as a means of increasing performance. N processing elements each of speed $1/N$ cannot outperform a single processing element of unit speed [28]. In fact, it is likely that the single PE would fare better than the many because of overheads, such as communication, in the multi-node system.

Consider the algorithm represented by the graph in figure 1–1. Each of the nodes represent a task, and the arrows are dependencies. Tasks C and D cannot begin execution until tasks A and B respectively have completed.

In this illustration it is assumed (unrealistically, as will be shown later) that the cost:performance of PEs is linear. In other words $\$D$ buys a total performance of M whether one purchases a single PE of that performance, or N PEs

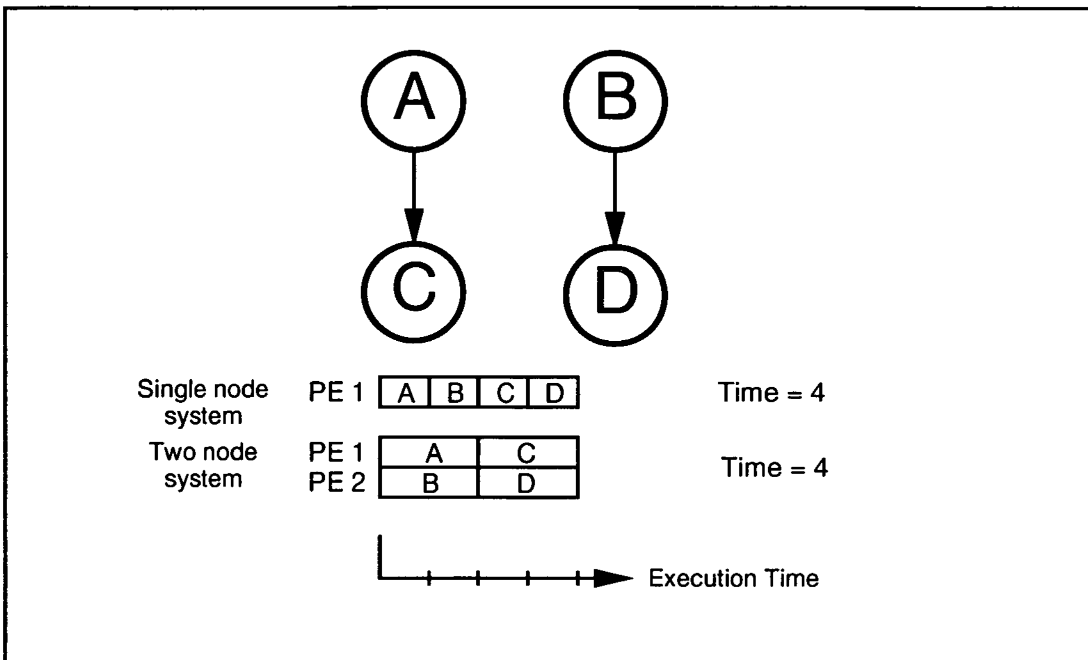


Figure 1–1: Effect of linear cost:performance - Basic DAG

each of performance M/N and cost $\$D/N$ [36] [37]. It is also assumed (again, unrealistically) that interprocessor communication carries no processing overhead and that the tasks consist of CPU-bound instructions only, and do not require external memory accesses. The figure shows the execution of the algorithm on a uniprocessor and on a two-PE system. It is apparent that while adding a second processor has allowed the use of concurrency (simultaneous execution of **A** and **B**, and of **C** and **D**), the fact that each of the PEs in the two-PE system is half as fast as the single PE means that no performance increase is actually achieved. The advantage of concurrent operation on the two node system is exactly cancelled by the necessary use of slower nodes.

However, the above is hardly a parallel algorithm. The task graph is not connected and effectively represents two unrelated sequential computations. In figure 1–2 the results of several tasks are combined by a single final task. The execution times on various systems, with the same assumptions as before, are also shown.

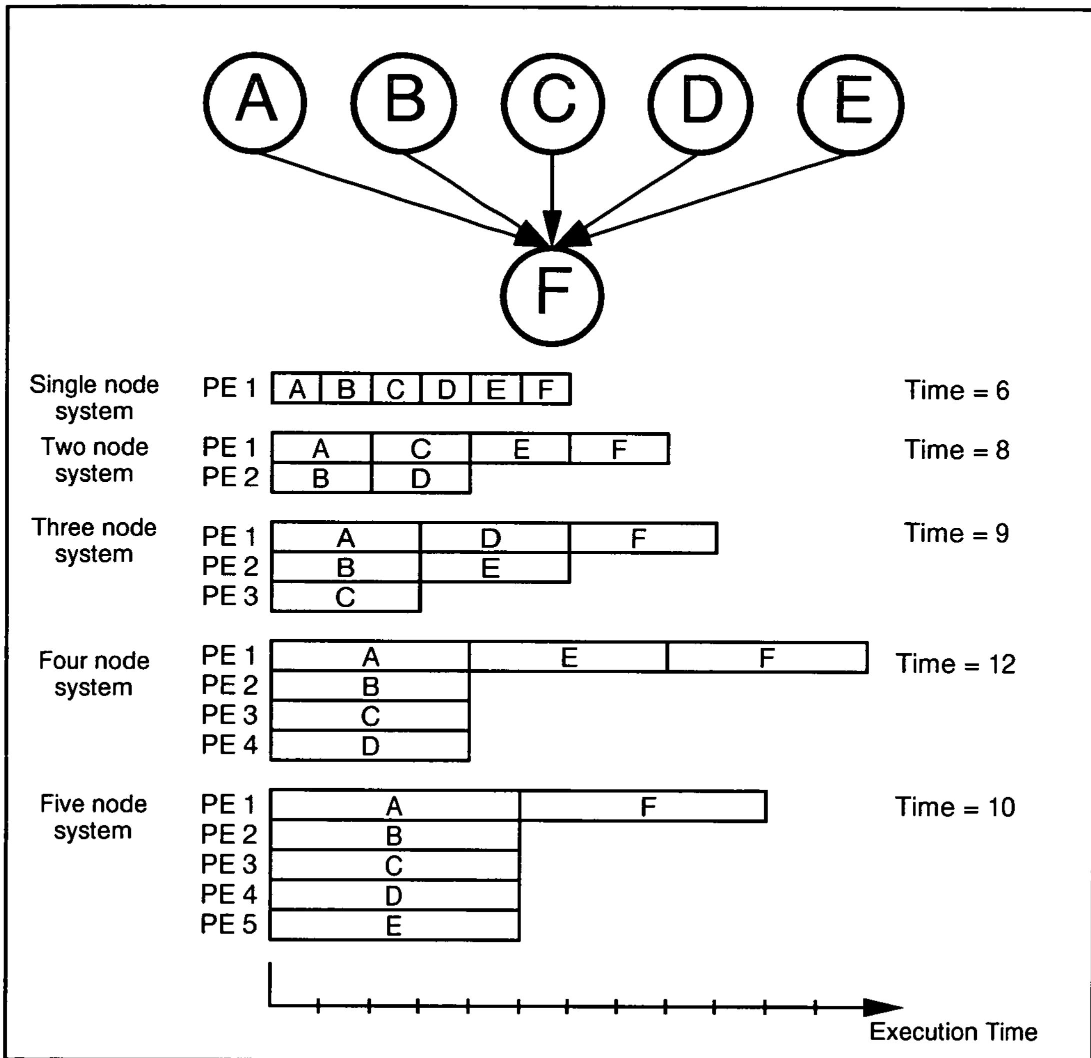


Figure 1–2: Effect of linear cost:performance - Connected DAG

Now, even though communication is still ignored, the multi-PE systems are decidedly slower than the single-PE system. Parallelism, in this case, is not merely a neutral option, but is in fact a performance-degrading factor.

Finally, consider the implictions of a sub-linear PE cost:performance. For example, if the performance P is defined to be the reciprocal of the time a PE takes to execute a task, then let P for a PE of cost C be given by:

$$P = \sqrt{C} \tag{1.1}$$

Now, with such a relationship between cost and performance, increasing the parallelism has obvious performance benefits (figure 1–3). The rate of decrease with N of the number of tasks the most heavily loaded PE must perform is greater than the rate of increase of the task execution time.

A key component then, in determining optimal N, is the precise nature of the PE cost:performance. This is discussed in detail in chapter 2.

However, in this example, as before, both communications and memory accesses are ignored.

If a time overhead is incurred whenever a PE must wait for a communication from another, then the benefits of using multiple PE's diminish. If this overhead itself is related to the number of PE's, (e.g. if it increases as a network increases in size), then the positive effects of sub-linear cost:performance have to be weighed against the negative effects of increasing network latency.

Also, if the amount of memory required in a system increases with N, then for a fixed total memory cost, the parallel system will be forced to use cheaper (and therefore slower) memory devices. An example of such space scaling is given below.

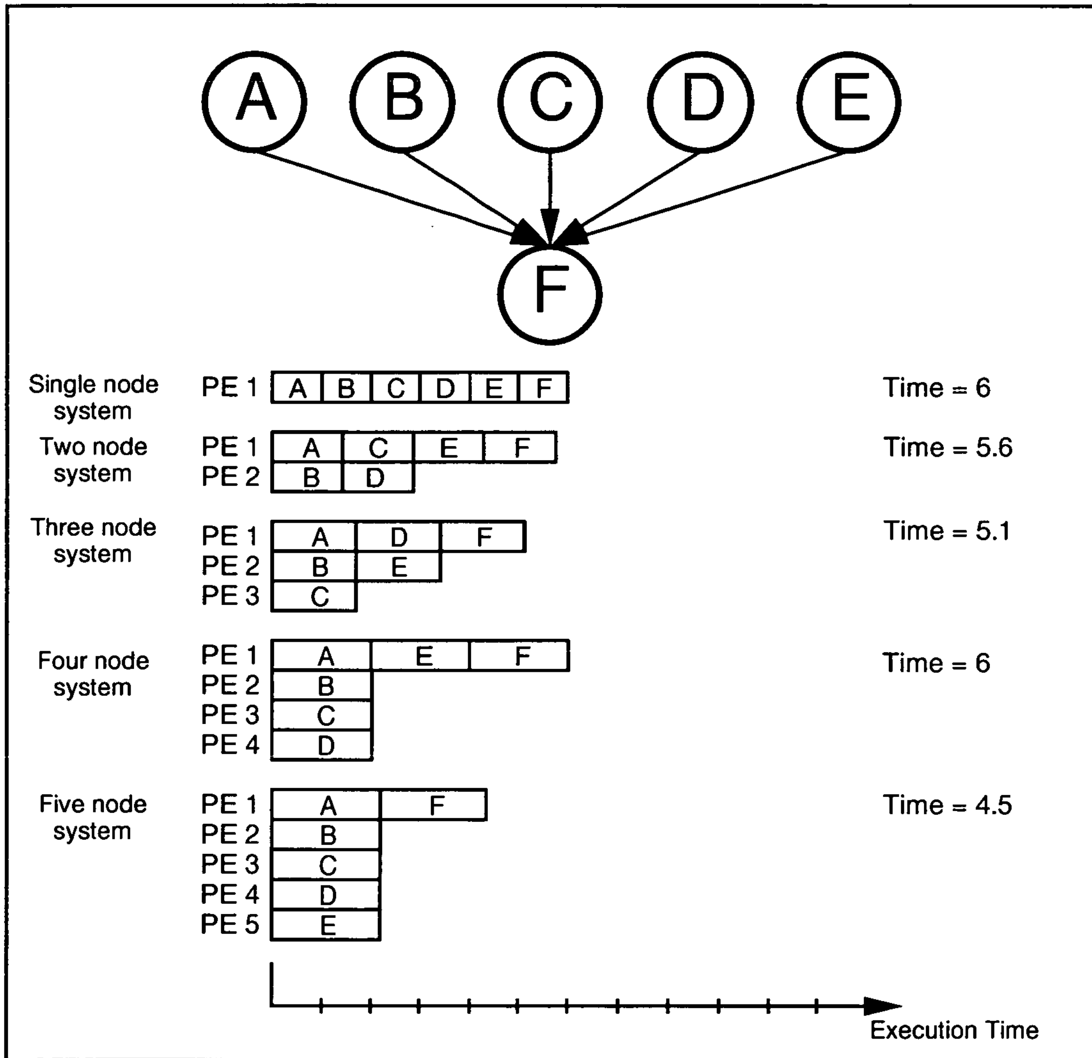


Figure 1–3: Effect of sub-linear cost:performance

1.1.2 Space Scaling

Let $\Lambda = (\Gamma, \Delta)$ be a graph of tasks, where Γ is the set of tasks, and Δ is a set of precedence relations among the members of Γ (see [70], and also page 69) section 4.3.1). For a set of processors, P, a *schedule* of Λ on P is a set of triplets, S such that:

$$S = \{\gamma, p, t : \gamma \in \Gamma, p \in P, t \in \mathcal{N}\}$$

where t is the starting time, on processor p, of the corresponding task γ. A general goal of scheduling is to minimize the maximum t in S.

Large amounts of literature exist on various aspects of scheduling in multicomputers – see [53] for a useful survey. Of particular interest here is the scenario in which obtaining an optimal schedule requires that some tasks in the DAG be recomputed on several processors [35] [57]. Recomputation may be required when the time to transmit the result of a computation from the processor on which it was obtained to a processor requiring the result is so great that the destination node can obtain the result faster by recomputing the result on its own behalf.

In the section of DAG shown in Figure 1–4, each of the arcs has a *weight* corresponding to the length of time required to send a message if the arc connects two tasks on separate processors. The figure shows the execution of the DAG on a single processor, on two processors without recomputation and on two processors with recomputation of task T_n. The existence of a communication delay for any data transfer sent from one processor to another means that the minimum execution time is obtained by running task T_0 simultaneously on both processors. In this way the communication delay is avoided completely.

In the context of this study, the degree to which recomputation reduces execution time cannot be understood without recognising the potential degradation in memory system performance of increasing the amount of memory in a fixed

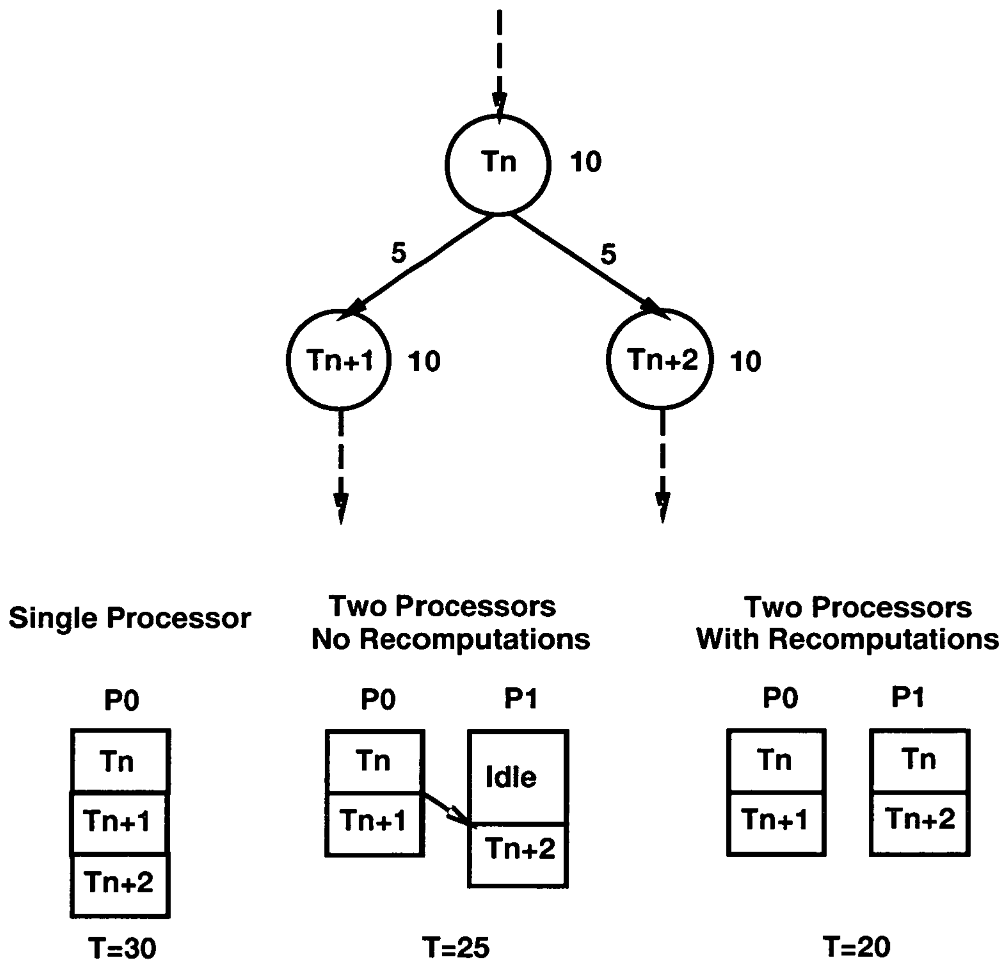


Figure 1–4: Weighted DAG Section

cost system. Recomputation requires that both data and code space be multiplied across as many processors as can usefully recompute a result, and this may require a reduction in the performance of that memory.

A specific example of space scaling will be discussed in detail in chapter 4.

1.2 Motivation

This work has two major motivations; the optimization of parallel system designs, and the optimization of semiconductor device portfolios for such designs.

1.2.1 Parallel System Design

During a panel discussion at a recent supercomputing conference [59] the speakers were asked for their opinions on which system philosophy – large numbers of cheap processors versus small numbers of expensive processors – was best. The speakers included representatives from companies advocating the use of large numbers of off-the-shelf processors and those offering systems consisting of a relatively modest number of custom-built, very high performance processing elements [51]. That this problem is still open can be seen from the continuing presence of machines (e.g. those from CRAY Research) which consist of a small number (less than 10) of extremely powerful processing elements while, at the same time, massively parallel machines using microprocessor-based PEs are also on offer [40]. A useful survey of this field can be found in [62] [73]. The panel's unanimous answer was, "it depends". The first motivation of this thesis is to begin to answer the following question, raised by that panel's answer:

Upon what and to what degree does it "depend"?

It may be that both types of system, and others besides, offer particular advantages in particular situation. (If so then perhaps the notion of a truly general

purpose parallel system is unrealisable [33]). But a model which quantifies those advantages and specifies the situations in which they exist will be valuable. Among other things, it may explain why it is that, so far, massively parallel systems have not been embraced by industry and business users. It is curious that, while research into parallel computers has developed tremendous momentum over the past decade (a cursory glance at the list of journals publishing research in this area indicates a continuing growth in the amount of work in progress in all aspects of the field), large-scale parallel machines have failed so far to find widescale application in the commercial arena. This failure is often blamed on deficiencies in various areas of parallel software. These vary from the practical lack of easy-to-use parallel compilers, efficient parallel operating systems, etc., to the more fundamental lack of a general purpose model which can span the gap between hardware and software [74] [26] [12].

But these deficiencies are only half the story.

The level of tolerance that a computer user has to a decrease in ease of use of a new computing environment compared with an existing system is directly related to the level to which performance can be boosted within that environment, again compared with the existing system. In other words, users will put up with a lot if the rewards are big enough.

A key reason that massive parallelism has been relatively unsuccessful in industry is not *per se* that computers in that architectural paradigm are difficult and costly to program, but that they are not outperforming equal-cost sequential systems enough to outweigh that difficulty and cost. Sequential systems are often still too good to be discarded in favour of their parallel cost-equivalent.

1.2.2 Optimizing Device Portfolios

Regardless of the current state of affairs, it seems reasonable to believe that eventually sequential systems will lose their attraction, not least because insurmount-

able physical limits to individual component speed will be met [48]. If this is true, then it is worthwhile considering the implications for providers of components for computers.

For most of the history of commercial computing machinery, the vast majority of computers have been single-node systems. Today, with mass produced personal computers and workstations, this is particularly obvious. Therefore, it is understandable that the manufacturers of microelectronic components for such machines will have based their device portfolios on the demands of single-node architectures. A typical system consists of a single microprocessor connected by relatively simple means to a memory system. The latter may be hierarchical with one or more levels of cache, and a high capacity storage medium such as disk. The range of memory, processor and support chips available, with their associated costs and performances reflect the strengths and weaknesses of the underlying von Neumann architecture as a model of computing machinery.

It is not obvious that the current situation is optimal for parallel machines.

The existing marketplace, in terms of volume of machines of a particular performance, influences the profile of the product portfolios of the various component manufacturers. Fast computers are currently built from fast parts, while low- to mid-range components are used for the larger volume markets. However, if it was found that the optimal number of nodes for a common application was greater than one, then this could cause a shift in the demands placed on the silicon producers. Similarly, it is not unreasonable to ask if the current ratio of memory:processor hardware resource will be the same in a massively parallel SIMD computer (for example) as it is in the typical high-performance workstation available today. If not, then again the range of devices, in terms of speed, capacity and cost may have to change to suit the new market created by parallel architectures.

Thus the second motivation of this thesis is to provide input for the develop-

ment of component portfolios. It will provide a means by which questions such as the following may be considered:

1. *Are fast expensive processors the best devices with which to implement very high performance computers?*

2. *Do current memory device portfolios offer the best range of speeds and capacities for high performance computers?*

3. *What performance and relative pricing (currently relatively unknown) is appropriate for interconnection devices?*

1.2.3 Tradeoffs in a Fixed Cost System

The tradeoff between number and speed of components is just one example of a more general principle which, through an investigation of optimal grain, this thesis is intended to highlight. The principle is that in a fixed cost system, a proposed enhancement in one part can only be implemented by reducing the performance (in the widest sense) in another part. In the context of this study, the enhancement consists of adding processors, and the manifold drawbacks include a reduction in the performance of the existing processors. Another example is the option to add more memory to an existing system. This is usually worthwhile, but it is only one of a number of alternatives, including increasing the speed of existing memory, or adding a cache (without increasing actual primary store capacity), or even adding further processors instead of memory. Deciding which enhancement to perform requires an understanding of not only its benefits, but also the drawbacks elsewhere in the system. Also, it is crucial to bear in mind that a system will tend to be only as fast as its slowest component. A great deal of existing parallel architecture research, while focussing on one specific aspect of the system (e.g. the interconnection network) leaves the question of the overall *balance* of hardware

unasked and unanswered. For example, in many interconnection network studies a key goal is the development of an interconnect which provides, for minimum cost, the highest performance *possible.* Performance is typically a combination of latency and throughput. It is generally acknowledged that electronic complete connections, while being excellent topologically, are too costly ($O(N^2)$) to be viable and so other networks, such as various multistage topologies are often proposed in an attempt to approach complete connection performance without incurring the high cost [21] [42] [43]. However, this approach, driven as it is by the goal of maximum possible performance, fails to take into account the fact that in a fixed-cost system the processing hardware may never need an interconnection scheme as powerful as an efficient multistage let alone a complete connection. The goal of network design should be the development of an interconnect which provides, for minimum cost, the highest performance *necessary*, as dictated by the demands of the workload, and in balance with the performance of the rest of the hardware.

Another example is memory device and microprocessor speed. A typical read access to memory will involve the microprocessor driving address and control information onto the appropriate signal lines on the memory devices, and then waiting for the relevant handshake signal before reading the data. Usually the handshake will be checked on a particular clock edge, and a full clock cycle (or "wait-state") must be inserted if the handshake is not ready.

If the memory is fast enough, it can respond with data to allow the processor to operate with zero wait-states and at maximum speed. Slower memory, on the other hand, may force the processor to stall as it inserts wait-states while the memory devices access and drive the data. So, in terms of this access mechanism alone, it is useless to invest significant amounts of the system budget in either very fast memory or very fast processors, if each is not matched to the other. The memory should be fast enough to provide valid data as soon as the processor can accept it, *but no faster.* Conversely, the processor cannot make use of any extra

speed (considering memory accesses only) over and above that corresponding to best-case memory access time.

Tradeoffs of this type will not be considered further in this thesis. Here the tradeoff of interest is the number of components versus their speed.

1.3 Related work

A significant amount of research has been carried out over the last two and a half decades into the performance of all aspects of computers, particularly parallel computers. This has ranged from general models of system performance, relying on relatively high level machine descriptions and statistical workload representations [32] [45], to precise simulation models of a specific detail of a microprocessor, or network switch, etc. [49] [55]. However, in much of this research, while the cost of hardware has not been completely ignored, it has not been fixed in such a way as to reflect the real tradeoffs faced by a budget-bound designer in the real world. The merits of some enhancement may be considered and weighed up against the cost of implementing the enhancement, but little work has been done to examine those merits in a fixed cost system where the enhancement can be implemented only at some cost to the rest of the system.

In parallel architectures in particular, relatively few studies have attempted to quantify the potential *negative* effects in any part of the system, of trying to induce a *positive* effect in another part. Models of *speedup* and *scalability*, for example, are common [54] [69] [66]. These examine how the overall performance of a system will increase as N is increased, but typically where the processor type is the same throughout the analysis. That is, the cost of the system increases appropriately.

The significance of the cost increase may be recognised as a decrease in processor utilization and system efficiency [20]. However, a decrease in efficiency is really significant only if an alternative solution exists which results in a higher

utilization and therefore a higher performance. Of more immediate significance is the fact that if system cost is fixed then either:

1. Cost is fixed at that of the initial system and therefore the proposed (more parallel) system must use cheaper PEs; or

2. Cost is fixed at that of the new system (i.e. one containing more PEs of the type in the original system). In this case, an alternative "new" system can be contemplated; that is one containing the same *number* of PEs as in the original machine, but where the PEs are more expensive.

Of specific relevance to this study is work by Barton and Withers [4]. They investigate the optimization of N by considering real processor cost:performance functions in the form:

$$C = dV^b \tag{1.2}$$

where C is processor cost, V is processor speed, and b and d are positive constants.

They conclude that

> *"for a given cost, delivered performance is maximized by selecting the fastest processor available at a given technology level, and employing as many as the budget allows".*

This conclusion is drawn using $b = 0.25$ but is made more general by noting that b can be expected to exceed unity as the leading edge of technology is approached. For $b > 1$ the optimal N may itself be greater than one, but is still finite. This study is a useful starting point but focusses on only one specific aspect of the problem; that is the speed of the processors as a function of their cost. The

effects of interprocessor communications are not included in their analysis, nor are the effects (if any) of memory references. Intuitively, interprocessor communication should be further reason to restrict the number of processors used, since communication can be a performance liability.

Similar work was reported by Van-Catledge [75]. In this paper the author provides a comparison of three broad classes of parallel computers under various combinations of problem size scaling and serial fraction. The comparison consists of finding the combination of clock speed, scaling factor and serial fraction which provides performance greater than or equal to that of some reference machine. The general conclusions support those in [4] and imply that a small number of fast PEs is superior to a larger number of slower PEs. In fact, like Barton and Withers, Van-Catledge ignores the effects of communications overhead and so, in that respect, understates the result. On the other hand, as indicated in [44], the superiority of the coarse-grain machines depends on the overheads associated with parallel processing (primarily communication) growing faster than $O(N)$. Further, it is pointed out that the exact nature of the cost:performance function of the processors can have a critical impact on the optimal number of nodes. Even if the overhead grows faster than $O(N)$, still the cost of nodes may begin to increase so dramatically at technology leading edges that it is better to use a large number of slower nodes than to invest in a small number of state-of-the-art devices.

The notion that parallelism may not be the panacea for the demand for higher computing power is not new. As early as 1967 Gene Amdahl suggested that, because of inherent sequentiality in some algorithms, parallelism may not offer such significant performance improvements as were being promised [3]. However even Amdahl's paper did not account for the further drawbacks of increasing N which occur if the total system cost is fixed. Gustafson further developed "Amdahl's Law" to account for the fact that often, parallelism is used not to increase speed of a fixed-size computation, but to increase the size of problem which can

be solved in a fixed time [23]. This provided more momentum for parallelism, but again did not take into account the effects of cost fixing. In 1987, Lundstrom tackled the problem of optimal use of computing resource in a general manner [46] and this was examined in a more detailed way by Ho and Snyder in 1990 [27]. The latter suggested a principle of computer design which considered a proposed enhancement to a computer to be cost-effective if the fractional reduction in computation time resulting from the enhancement was greater than the fractional increase in cost of the enhancement. This notion has some merit, but at the leading edge of technology and performance the only enhancements possible may not be "cost-effective" (their definition). This is because at that leading edge, serious diminishing returns are experienced as more and more resources must be expended to obtain a rapidly decreasing performance improvement. Lundstrom takes a broader view and considers the solution to the problem as a black box which must achieve an optimal cost:performance. This leaves open the option of implementing an enhancement of significant cost and minimal performance benefit, but for which the cost:performance is already optimal. In other words, an enhancement becomes "cost effective" if the performance benefits provided by the enhancement cannot be provided in any other, less expensive, way.

1.4 Approach

Two principal parameters of computer performance measurements are execution time, T and problem size, P. The latter could be some measure of the size of the data set on which some algorithm must be performed. In most situations, a computer user requires maximum P/T while restricting the range of one of the two parameters. In the context of this study, the goal is, for a fixed cost system, to choose N, the number of nodes, so as to minimize one of T or P as a function of the number of nodes. For either, the other parameter may be held constant. In other words, the problem is to choose the number of processors with which can

be solved the largest problem in a given time, or a given size of problem in the shortest time.

In practice the maximum attainable speedup in most algorithms is limited, for any particular problem size, by the sequential portion of the computation [3]. As a result, the greatest benefit of parallelism is often to increase the amount of work which can be carried out in a reasonable (i.e. acceptable to the user) time. This is as opposed to decreasing the execution time of a fixed size problem [23] [68]. There appears to be a tolerance, on the part of a users, to various delays experienced when requiring different tasks of a computer. For example, the refresh of a drawing in a CAD application must be no more than fractions of a second whereas it seems acceptable for the compilation of a small to medium high-level language program to take some tens of seconds or more. Large simulations may take hours or even days; the simulations carried out to investigate further the models in this thesis took several hours per run. An increase in computing power in circumstances like these would more likely result in larger drawings, and more complex compilations and simulations rather than (or at least in addition to) a decrease in time for the original problems.

Nevertheless, the time to solution, even for the largest problem, must be kept to some acceptable maximum. And even if an increase in N does allow an increase in problem size, the question still remains as to whether or not an equivalent (or greater) increase could be obtained using another equivalent cost system with a different number of nodes. Therefore, without loss of generality, the focus of attention in this study will be execution time. In order to emphasize the effects being considered, the PEs in all cases will be considered to be equally loaded. In other words the algorithms will have no sequential fraction. In reality, such a fraction will enforce an upper limit on the number of PEs that can be usefully employed. Although ignoring the unparallelizable aspect of a computation will bias any overall analysis in favour of parallelism, and towards a higher optimal N, its inclusion would serve to mask some of the specific effects of interest. Since it is

such effects that are the focus of study here, the algorithms used will be assumed completely parallel. In general, including the serial fraction would not be difficult and could be accounted for as a constant component (independent of N) of the critical path of the algorithm.

The basic approach will be to develop a model of the execution time T of an algorithm on a machine as a function of the number of nodes, N. The model will be used to examine the degree to which it is possible to obtain an optimal value of N, corresponding to minimum execution time.

Specifically, the optimization will be done assuming:

1. A base architecture.
2. An algorithm or class thereof
3. An overall hardware cost.

Two base architectures and algorithm types will be considered, corresponding to two common classes of parallel architectures: a shared memory multiprocessor and a message-passing multicomputer [6]. The former will be represented by a simple shared-bus multiprocessor in which each PE performs operations on data stored in a global shared store. The second type will be represented by a two-dimensional mesh-connected multicomputer. In this type of system all memory will be considered private and data sharing will be carried out using "messages" explicitly passed among the PEs

In addition, two sets of information are required, and will be developed in this study:

1. The cost:performance functions of the various hardware functions and components. For example, how many MFlops can be bought per dollar? How many MBytes of storage of a given access speed? How many MBytes/second of data transferred across what distance?

2. In terms of the above functions, the demands which the software will place upon the hardware.

The single advantage (in terms of increasing performance) of a parallel computer over a sequential system is that it can divide work among its various processing elements. This means that, ignoring overheads associated with communication, the workload on the most heavily loaded PE in a parallel system typically will be less than the workload on a sequential machine running the same program. However, opposing this advantage are several disadvantages, including:

- The workload on the PEs in a parallel system may be more than a simple $1/N$ division of the original program. For example, in some search algorithms the ratio of the number of comparisons done by a single PE system compared with that done by a PE in an N-node system is only $O(logN)$ and not $O(N)$ [41] [60]. Further, the parallel algorithm will typically require processing associated with communication as well as delays incurred by that communication.

- The hardware components in the parallel system, being more numerous than in the sequential system, are cheaper and therefore potentially slower than those in the latter.

- Some hardware components may constitute a shared resource, having to respond to demands from more than one PE. Such sharing can result in a PE being delayed in making use of a shared resource while another PE has ownership. Such contention does not exist in a sequential system.

Whether or not a parallel system will provide a higher performance than an equivalent cost sequential system depends on the net effect of combining these negative factors with the positive advantage of dividing the workload among the PEs. This can be generalized to compare two or more parallel systems with different numbers of PEs.

1.4.1 Thesis Outline

This thesis is laid out in the following manner. Following this introduction, in which the basic problem and context are described, chapter 2 discusses the relationship between the cost and performance of current (mid 1990's) VLSI technology. Specifically, models are presented of cost versus instruction execution time, memory access time and switch latency. Chapter 3 then incorporates the results from chapter 2 into a model of the execution time of an algorithm on a shared-bus multiprocessor. In a similar way, chapter 4 develops a performance model for an algorithm on a two-dimensional mesh multicomputer using a message-passing communications scheme. Chapter 5 compares the model predictions with simulation results and discusses some of the implications of the models presented to that point. Finally, chapter 6 concludes and provides suggestions for further development of this work.

Chapter 2

VLSI Cost:Performance

2.1 Introduction

This chapter analyzes the relationships between cost and performance of the components used in current computers.

The hardware can be split into three basic functions: processing, storage and interconnect. Corresponding to these are three basic components: microprocessors, memories and routers. In each case, it is assumed that the component is constructed in technology current in the mid 1990's. At this point in time, VLSI devices are common and ULSI components are imminent. Systems are constructed from ensembles of such components, packaged and grouped on printed circuit boards, or perhaps as unpackaged dice on ceramic substrates. These assumptions allow detail to be developed in the parameters in question, but do not limit the overall applicability of the analysis.

2.2 Processing Hardware

Advances in VLSI microprocessor performance are currently being achieved in two key ways: by increasing the operating frequency of a device and by increasing its internal parallelism.

Clock frequency is determined by several factors, gate-oxide thickness and "feature size" being two of the most crucial. The latter in particular is often quoted by semiconductor manufacturers as an indication of the sophistication of their manufacturing process. Feature size, typically quoted in microns, is a measure of the length of the silicon channel in the MOS transistor – the basic component of VLSI devices. The figure given may also refer to the width of metal lines on the mask used to print photoresist patterns during various stages of wafer fabrication. An alternative to this "drawn length" is the measured electrical length

of the channel, $L_{Effective}$, which is typically 10% to 20% smaller than the drawn length. Whichever is quoted, as feature size is decreased, switching times and signal propagation delays decrease. Shorter interconnects also present smaller capacitances to drivers bringing further speed improvements. In contrast with recent geometries of 1 to 2 micron and above, 0.8 micron designs are now commonplace in mid-range devices [30] and 0.65 micron offerings are already available. Sub-0.5 micron devices are planned.

Of at least equal significance to clock speed, in determining device performance, is the degree of internal "parallelism" in the device. This is most commonly seen as an increase in the width of the data path of microprocessors. 8-bit devices, still used in low-end single-chip microcontrollers have mostly given way to 16 and 32 bit architectures, and 64-bit devices are available. However, while earlier devices used their internal parallel data path as a means of executing binary arithmetic at increasing speeds, today's processors provide a more explicit use of parallelism in the form of superpipelining and multiple execution units. However internal parallelism is implemented, increasing this aspect of a device requires an increase in the number of transistors in the design.

A third factor affecting device performance in certain circumstances is pin-out. Increasingly, designs are "pad-limited", i.e. the limit on device performance is imposed not by circuit complexity but by the I/O bandwidth of the packaged chip. This is currently an issue because of the large amount of silicon space required for bond pads onto which the device pins are connected and because the area of a die grows faster than its perimeter length. Some of the newest packaging technologies are helping to overcome this problem by allowing pins to be connected across the surface of the die, rather than simply along its edges [19], however this is still an expensive option and in use for only the leading edge of the market. For the purposes of this study, it will be assumed that the pins are bonded to pads around the die perimeter and that pin-out is proportional to the square-root of the die area.

These two aspects, the number of transistors as a measure of device parallelism and feature size as key to clock frequency will be taken as the main factors determining device performance.

With this in mind, a cost model for ICs is now developed.

2.2.1 Cost and Price

Cost is a measure of what a manufacturer has to do to produce a device. Price is a measure of what a purchaser of that device must do to buy it.

If the manufacturer of a computer system purchases all of the required components from other manufacturers, then the difference between device costs and prices are large, varied, and subject to all the vagaries of large scale economics. Device prices must cover not only manufacturing costs but also design and development. Sales and marketing overheads may also be significant factors. Also, of course, the price of a device is heavily dependent on demand. While very general trends can be observed, there are so many factors at work (including, for example, pricing strategies which sustain real short-term losses in order to increase longer-term market share, or raising the price of older end-of-life devices to encourage purchasers to move to newer products) that actual device prices are of little use in obtaining the required relationship to performance.

If, on the other hand, the manufacturer of a computer system decides to manufacture the devices required to build the system, then price and cost are almost the same and one of the lower cost margins is appropriate. For this reason, cost will be used as the primary factor governing performance.

2.2.2 Modelling IC Cost

The production of a VLSI device can be considered as a sequence of steps, in each of which an initial number of devices of some form (die sites on wafers, individual

dice, etc) may be put through a process phase and then subjected to a test phase. The devices passing the test will continue onto the next step and the remainder are discarded. Alternatively, those failing a test may be retested under a less stringent set of parameters (e.g. at a lower clock frequency) and passed or failed accordingly. In some steps no processing is carried out and the devices are subjected to several tests in sequence. The fraction of input devices which pass a test phase is called the step *yield.* The final overall cost of a device is therefore the total cost of all steps, (taking into account the continually decreasing number of devices remaining within the process) divided by the final number of good devices. If the number of devices passed into the first step is D, then the total cost, C_{IC}, for a device produced in an n-step sequence is:

$$C_{IC} = \frac{D(C_{p_1} + C_{t_1}) + Dy_1(C_{p_2} + C_{t_2}) + ... + Dy_1 y_2 ... y_{(n-1)}(C_{p_n} + C_{t_n})}{Dy_1 y_2 ... y_n}$$

$$= \frac{\sum_{i=1}^{n} \left((C_{p_i} + C_{t_i}) \prod_{j=0}^{i-1} y_j \right)}{\prod_{k=0}^{n} y_k}$$

where:

C_{p_i} = cost of processing phase in step i

C_{t_i} = cost of testing phase in step i

y_i = yield of step i ($y_0 = 1$)

The processing phase (if one exists) in each step may consist of a large number of sub-processes and the number of steps in the overall sequence depends on the confidence the manufacturer has in the quality of each sub-process. In general it is beneficial to remove bad devices from the sequence as early as possible. This saves the cost of processing failed product any further, and may prevent non-functional parts from reaching the end-customer. However, these benefits must be set against the costs of testing. If these costs are significant compared with the costs of processing, then it may be acceptable to allow the progress of bad

devices down some way the sequence. A sufficiently small probability that bad devices exist in any numbers is also reason to minimize testing. When sub-process quality exceeds a certain level a given test may be deemed no longer necessary. As a result, a device which, in early years of manufacturing, required three tests to be applied to all devices may finally be manufactured and sold having only one final test applied to a sample of finished product. In this case the test is being used merely to tune and affirm confidence in the manufacturing process rather than systematically to remove faulty devices.

Often the manufacturer will combine the processing and testing costs in each step into a single figure. These step costs, along with the associated yields, are the focus of the various groups within the semiconductor manufacturing organization in their attempts to reduce final device cost. The overall cost is thus:

$$C_{IC} = \frac{\sum_{i=1}^{n} \left(C_i \prod_{j=0}^{i-1} y_j \right)}{\prod_{k=0}^{n} y_k} \tag{2.1}$$

where C_i is the combined processing and testing cost of step i.

For current microprocessors a typical manufacturing process consists of at least three steps: wafer fabrication and "sort", assembly, and final test. A burn-in step may also be carried out (before final test) in which packaged devices are operated at a high temperature for a short time (up to a week) in order to "kill off" devices of marginal quality.

Wafer fabrication consists of the numerous layering, patterning and other steps to take the basic silicon through to complete circuits on the wafer. During fabrication, several tests are carried out on the developing wafer to provide feedback to process engineers and to eliminate defective wafers as early as possible. These include various electrical measurements and visual and automatic optical tests. After processing, the wafers are tested in wafer sort. Here individual die on the built wafers are tested while still attached to each other and faulty die are identified. This is done using an attachment to the tester which makes contact with

the pads of the device using tiny wire probes – hence the alternate name, "probe test".

This point in the process is often taken as the first costing point by manufacturers, the cost being referred to as probed wafer cost. This covers not just the manufacture of the wafer, but the cost of probe test itself.

The wafer is then scribed and the individual dice separated. Those marked during probe are discarded and the remainder are packaged using one of several materials and techniques. A second significant cost, that of assembly, is recognised here.

Finally, the packaged devices are passed through a comprehensive test suite before being shipped to the customer. The final cost of the device is obtained at this point.

Numerous variations on this process are possible. For example, if the manufacturer has sufficient confidence in the wafer fabrication process then probe test may be considered an unnecessary expense. In this case all devices will be packaged and only then subjected to the final test. Alternatively, rather than omit wafer sort entirely, a minimal sampling probe test regime may be followed as a check on the manufacturing process up to that point.

For the sequence just described, following equation 2.1 the cost of the final device is:

$$C_{IC} = \frac{C_S + y_S C_A + y_S y_A C_F}{y_S y_A y_F} \tag{2.2}$$

where:

C_S = Probed (sorted) wafer cost per die

y_S = Probe yield – the fraction of probed die which pass probe test.

C_A = Assembly cost per die

y_A = Assembly yield

C_F = Final test cost – the cost to test a packaged device

y_F = Final test yield – the fraction of packaged devices which pass final test.

Probed Wafer Cost

For VLSI, wafer fabrication is usually the major component in the overall cost of an IC. Various factors affect this cost component including labour and material costs and also the cost of depreciating equipment. The latter is particularly significant in new wafer fabrication facilities. In older more mature installations which are fully depreciated, capital costs decrease but labour costs are still significant.

Three of the most important process parameters affecting cost are wafer size, number of fabrication sub-processes, and feature size. Current wafer fabrication costs vary from less than $20 to $60 or more per square inch of wafer depending on these factors [24] [25]. For example, decreasing feature size will, in general, require more expensive and less-depreciated equipment. Increasing the number of sub-processes (reflected, for example, in an increase in the number of layers of metal) will typically increase the time and materials used to build the wafer with a resulting increase in labour and material costs.

These three parameters are not independent of each other. As designs grow in size and complexity (i.e. in transistor count), all three parameters are affected. Feature size is decreased to increase device speed but also in order to keep the die size acceptably low. This is necessary to keep wafer sort yield high. Another area-saving method is to increase the number of layers of metal interconnect on the die. Single and double-layer metal processes are not sufficient for advanced microprocessors, and so three, four or more layers are being used. Also, since these

extra steps are typically taking place with smaller feature sizes, the cost of each step will also be increased. Wafer size itself is also increasing as device complexity grows. This is to achieve a large number of die per wafer allowing the wafer cost to be amortized over a larger number of units and reducing the relative number of non-functional edge dice.

Feature size is also significant in the testing component of C_S. The cost of wafer sort depends on the hourly cost of the probe facility and on the time to test the die.

VLSI testers are increasingly sophisticated and expensive machines, the required performance of which depends upon (among other factors) the speed of operation of the device under test. The depreciation cost of tester equipment is therefore a significant proportion of the probe cost for high performance devices. In general the capital costs are more significant in test than in fabrication and this distinction looks likely to increase for some time.

The test time depends on at least four factors; load time, index time, actual test time, and number of die probed concurrently. Load time is the time to prepare the tester for a particular product and may be several tens of minutes. However this is amortized over all devices subsequently tested using that program and is only significant if the test program is changed frequently. Index time is the time to step the wafer across the probe head.

The actual test time depends on the number of vectors required to achieve a given fault coverage and the speed at which these can be used. The former depends upon the circuit complexity and upon its observability [1]. Scan methods allow relatively high fault coverages for even very complex designs, however the time to achieve this still depends on the depth to which test patterns must be first clocked into and then out of the circuit. Assuming a fixed required fault coverage (e.g. 95 - 100%) actual test time will therefore be considered a function of the transistor count. This is as a measure of both complexity of the circuit and the

pin out (i.e. specifically the number of primary inputs and outputs) as described earlier. The number of die which can be probed at one time depends primarily on the pin-out (for a fixed pin-count probe head).

These factors are not unrelated. For example, there is a tradeoff between the index time and the degree of concurrent probing. Testing more than one die at a time may reduce the actual test time, however it can serve to increase index time as the effective increase in area being aligned under the probe head makes alignment more difficult and time consuming. Also, the relative importance of these factors changes as one moves from simple to more complex devices. Actual test time is less significant for very simple devices and may be of the order of only a few milliseconds. This is due partly to the low complexity of the design, but also to the use of concurrent probing made possible because of lower pin counts.

Unfortunately, while these tradeoffs are understood in a qualitative way, the fabrication/sort process is simply too complex to succumb to an attempt to model it in quantitative detail. The basic physical effects just described are often swamped by factors such as the familiarity of wafer fab personnel with a particular process (costs typically being higher on the early stages of the learning curve). Varying labour costs across the globe also produce significant variation in costs. Real day-to-day costing in the semiconductor industry uses real cost data from working wafer fabrication facilities across the globe, and the level of modelling is fairly superficial.

However, of the three parameters discussed above, feature size seems to be the dominant parameter. Assuming that both the number of process steps and the wafer diameter are both increasing as feature size decreases then probed wafer cost per unit area appears to be exponentially related to feature size. Let the cost per square millimetre be denoted by u and be expressed as:

$$u(\lambda) = ae^{-b\lambda} + u_{base} \tag{2.3}$$

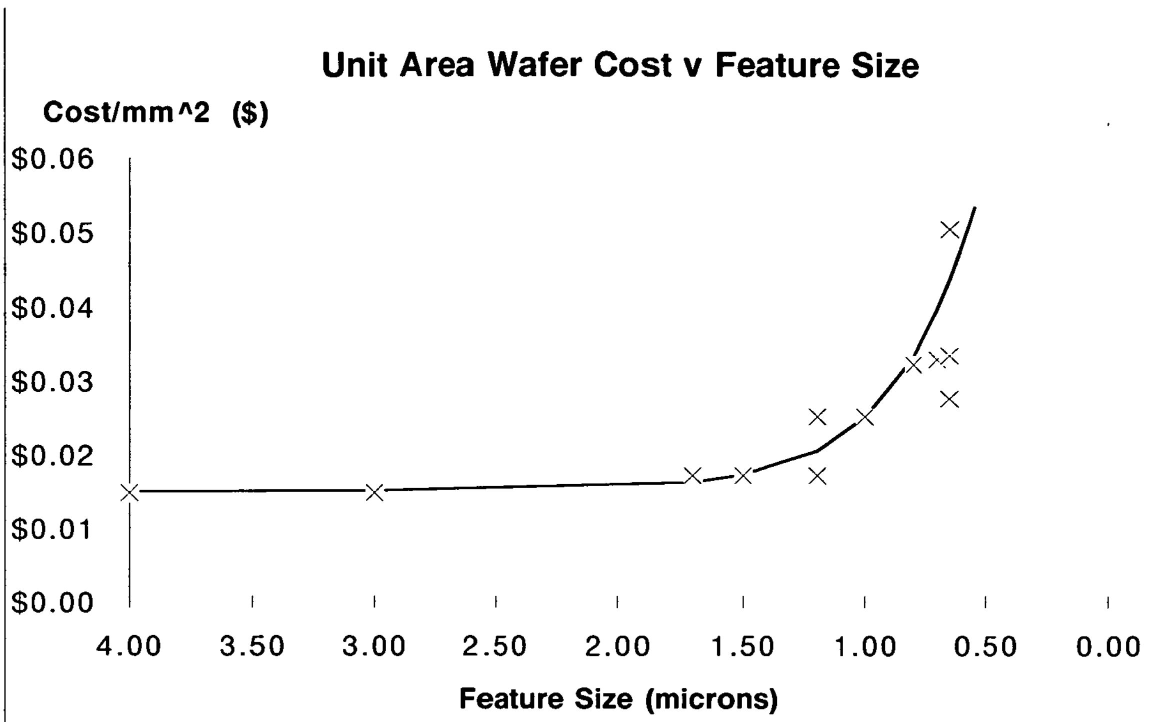


Figure 2–1: Probed Wafer Costs

where λ is the feature size in microns and u_{base} is the lower bound on cost dictated by basic wafer fab running costs and minimal profit requirements experienced by the manufacturer. The coefficients, a and b currently have values around 0.2 and 3 respectively and u_{base} is of the order of 0.015 [25] [77] [22]. See figure 2–1.

Again, it must be stressed that this relationship is more complex than simply feature size versus cost. The cost increases so rapidly below 1.0 micron because the wafer size is also being increased at that point. The number of layers of metal is also being increased as feature size diminishes. In addition to increasing the fabrication cost, these factors simultaneously increase the cost of probing the wafer.

Nonetheless, both wafer diameter and metallization are increasing in response to the same factor which is driving down feature size – i.e. device complexity. Thus feature size appears to be a good single indicator of the overall trend.

To obtain the final build cost for the die, equation 2.3 is multiplied by the die area. This assumes that the whole of the original wafer is available for die from which a device may be constructed. This ignores any test die placed on the wafer for monitoring the fabrication subprocesses and also any non-functional edge die (see p59 of [25]).

The probed wafer cost per die of area Amm^2 is thus:

$$C_S = A(0.2e^{-3\lambda} + 0.015) \tag{2.4}$$

Probe Yield

Several probe yield models exist, including Murphy, Seeds and Dingwall [25] [76]. These express the percentage of working die on a wafer as various functions of die area, defectivity, number of critical masks, etc. However, for simplicity and without serious loss of accuracy, a simpler exponential model is often used in the industry and will be applied here. Probe yield is given by:

$$y_S = \frac{1}{e^{AD}} \tag{2.5}$$

where:

A = Die area

D = Defect density (defectivity)

Defect density here refers to the number of killer defects which render a die non-functional. This is in fact related to feature size, or more specifically to circuit density. The closer together circuit elements are, the more likely is a defect to destroy a vital piece of circuit. As such, defect density varies between different types of circuit; memories, for example, typically have higher defectivities (all else being equal) than processors. The addition of a dense area such as a cache will, however, increase the density of killer defects for a processor. Like costs, actual

defectivity data is not generally available and for the purposes of this model, an average value of 1.0 per square centimere will be assumed [24]. For a die of area Amm^2 the probe yield is thus:

$$y_S = \frac{1}{e^{0.01A}} \tag{2.6}$$

This yield model (nor indeed the more sophisticated models mentioned earlier) does not take into account the effect on yield of discarding a whole wafer, even if it contains some good die, if the wafer's yield falls below some acceptable minimum. (e.g. 60% of average). This is done as a precaution, the manufacturer's confidence being low on a wafer with such abnormally low yield.

Assembly and Final Test

The cost of packaging depends on several factors, two of the most important being the type of material being used, and the number of pins in the package.

The packaging material is chosen to provide protection for the enclosed die, and a means of heat dissipation. Whereas early devices were built in dual-in-line plastic packages (PDIP), current high speed (and high power) devices may require ceramic pin-grid-array (CPGA) or quad-flat-pack (CQFP) or similar.

While there does seem to be a tendency to package larger, higher performance devices in packages built from more costly materials (and involving more costly processes), there is no easily accessible relationship. Therefore the package type will be assumed to be the same for all devices and the package cost will be governed by the number of pins (assumed earlier to be proportional to the square root of the die area).

While some models in use assume a constant cost/pin for a given package type, in actual fact the function is probably not exactly linear. As one would expect,

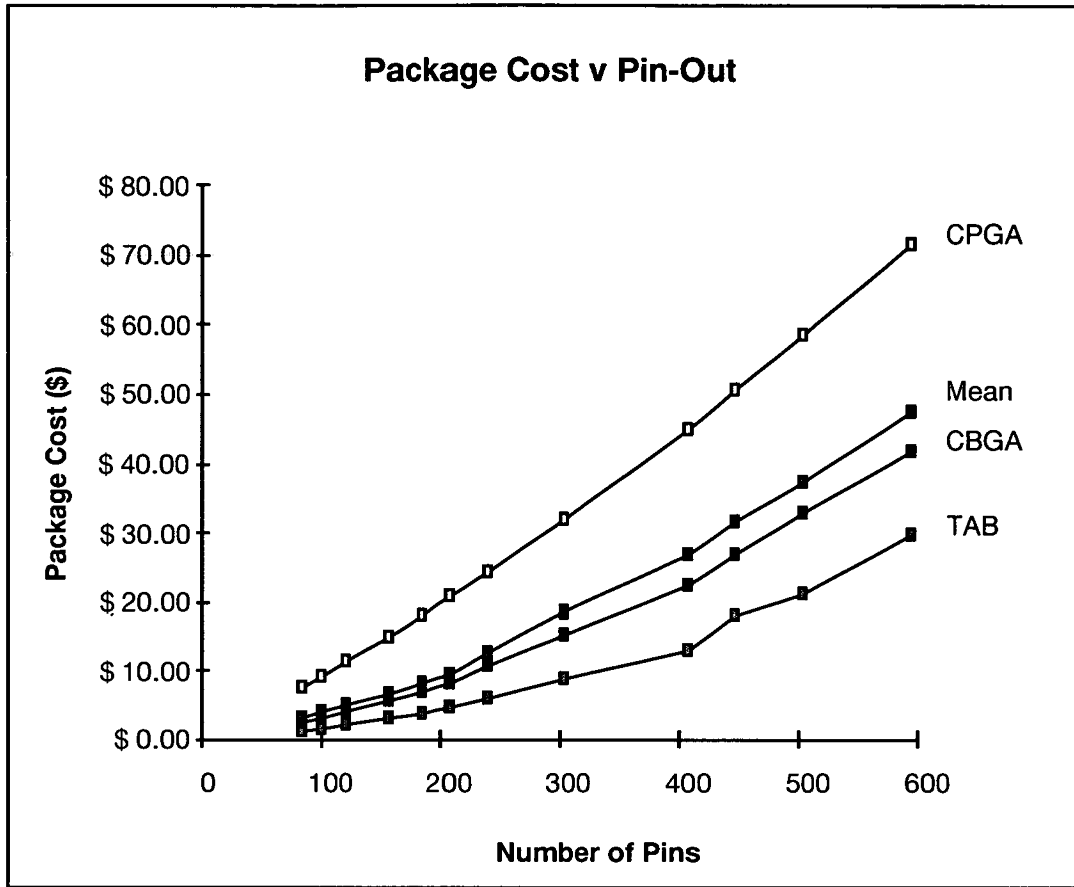


Figure 2–2: Package Costs (Source: Microprocessor Report, October 4, 1993)

the closer a packaging process is pushed to its leading edge, the faster does cost increase for a fixed proportional increase in "performance" (i.e. pin-out).

Figure 2–2 shows typical package costs for several package types. An average cost is also shown, and it is this which will be used in the rest of this analysis.

A simple model of cost versus pin-out, based on the data in figure 2–2 gives C_A of a P-pin device as approximately:

$$C_A = 9 * 10^{-4} P^{1.7} \tag{2.7}$$

As discussed earlier, pin-out will be regarded as being proportional to the length of the die perimeter length. Current package technologies allow between 2

and 6 pins per millimetre of die edge. Assuming an average of 4 pins per millimetre, pin-out is:

$$P = 4\sqrt{A}$$

and so assembly cost for a die of area A mm^2 is:

$$C_A = 9.5 * 10^{-3} A^{0.85} \tag{2.8}$$

Typically, assembled devices are given only a visual check before being passed onto final test. Irreparably bent leads are the commonest cause of failure at this stage, however yield at this stage, y_A, is almost 100% and will be assumed so for the rest of this analysis.

Final test cost is affected by similar factors to those affecting wafer sort although the relative importance of index time is increased since the test may be carried out at several temperatures, some requiring a "soak-time" to bring a device at ambient up to the relevant higher temperature. Hourly costs vary from less than $50 per hour to over $250 per hour for advanced state of the art testers. As with fabrication costs, feature size will be taken as the dominant factor affecting which type of tester must be used for a given device. The test cost per second, c_t, is roughly:

$$c_t = 0.5e^{-4\lambda} \tag{2.9}$$

The range of test times for devices tested on a particular machine is usually small enough to allow an average figure to be used for all devices. An average test time of 2 seconds is assumed, and the test cost per device is:

$$C_F = e^{-4\lambda} \tag{2.10}$$

Final test yield, y_F, unlike probe yield, is usually consistently high for production devices. Final test is principally a method of verifying the overall quality of the process from wafer sort onward and is not really intended as a means of removing inevitably faulty devices. Persistant failures in final test are regarded as a serious problem which must be corrected. Since no significant relationship is obvious between any of the performance characteristics and this particular yield, a constant 95% will be assumed for all devices.

Overall Cost

Combining equation 2.2 with equations 2.4, 2.6, 2.8 and 2.10, the overall cost for an IC is given by:

$$C_{IC} = \frac{Ae^{0.01A}(0.2e^{-3\lambda} + 0.015) + 9.5 * 10^{-3}A^{0.85} + e^{-4\lambda}}{0.95} \quad (2.11)$$

Die area depends on feature size, transistor count and circuit density, and also on the pin-out required on the device. Decreasing feature size produces smaller transistors which not only take up less space, but which can also be placed closer together on the die. High circuit densities are typically seen in regular circuitry such as memory devices. Optimising the layout of the basic memory cell gives significant area savings when those cells are laid out in hundreds of thousands, or even millions.

For a range of existing mid-range to high-end microprocessors, an approximate expression for die area as a function of transistor count T and feature size is:

$$A = 0.12\sqrt{\lambda}T^{0.94} \quad (2.12)$$

Fortunately, die area is generally available for most devices, and the published figures will be used where possible.

2.2.3 Microprocessor Cost:Performance

The purpose of this cost model is to establish a relationship between what it takes to produce a device and the performance of that device. To do this, a range of currently-available MOS microprocessor devices was surveyed. Various data were collected and are given in Appendix A. The device cost in each case was calculated using equation 2.11. Assessing device performance is more difficult.

A significant effort has been, and continues to be put into producing useful performance metrics for computers and their components. Several problems are faced in preparing such benchmarks, not the least of these being the desire, on the part of users of the metrics, for a single number to be attached to a device or system as a measure of its performance. Equally problematic is the ease with which devices or systems under test can be optimized with respect to the benchmark alone; i.e. without affecting the performance in general. Since benchmarks are often used by potential purchasers of systems to weigh one against another, it seems unavoidable that there will be a tendency on the part of manufacturers to try to cast their products in the best possible light.

In addition to these general problems microprocessor performance metrics are further complicated by the number of definitions of performance implied by the metrics themselves. The oft-quoted "millions of instructions per seconds" (MIPS) can be useful for comparing different processors with similar instruction sets. However, it is less appropriate when comparing CISC (complex instruction set computer) processors with devices following the RISC (reduced instruction set computer) paradigm. While the latter may exhibit a higher MIPS rating than the former, the "instructions" executed by a RISC machine often perform less actual work than those on the CISC machine. The difference in such "native" MIPS ratings may therefore be deceptive. In an attempt to solve this problem, normalized MIPS are often quoted, where the performance measurement is with respect to some standard machine (e.g. a VAX 11/780). Another measurement giving a

more useful performance comparison is millions of floating point operations per second (MFLOPS). Unfortunately, these metrics take little account of the nature of the software being run on the system containing the processor. Where some devices outperform others in floating-point intensive calculations, the positions may be reversed in code requiring intensive integer operations. To reflect this dependency, some benchmarks have been devised to measure the performance of devices running a particular type of computation (e.g. Dhrystone, Linpack, etc).

Finally, possibly the most significant problem with many device benchmarks is that they do not necessarily reflect the performance of a system containing the device. Since system performance depends on more than simply the raw speed of the processor, and since most computer users are concerned with the performance of the system as a whole, the current trend is to use a range of metrics which measure the total system performance over a wide range of types of code. One of the most common examples of this type of benchmark is the range of SPEC suites [11].

In deciding on a benchmark for use in this cost:performance model, it was necessary to choose one which provided a wide coverage, and which avoided the problems of comparing unequal instruction types. Although SPECint92 and SPECfp92 are system metrics, they are widely used as means of comparing the performance of the processors involved. Int92 was available for the widest range of devices, and so was the metric of choice. The value used was the maximum figure available on any machine using that device and so gives some idea of the potential performance of a processor. Because it is a system metric it cannot be used to compare devices of roughly the same performance, however its use here is to derive a more general relationship between performance and cost, and the performance differences resulting from differing system architectures are assumed insignificant when a wide enough range of devices is considered.

Table 2–1 shows cost performance data for several processors.

Table 2–1: Processor Cost:Performance

Device Name	Clock (MHz)	Area (mm^2)	Trans (k)	Geom. (μm)	Probe Cost	Probe Yield	Ass'y Cost	Test Cost	Int92	Total Cost
ARM610	25	26	360	0.6	1.34	77%	0.42	0.09	24	2.36
ARM710	33	34	570	0.6	1.75	71%	0.54	0.09	32	3.26
MPC601	100	74	2800	0.5	4.72	48%	1.18	0.14	110	11.81
MPC603	100	85	1600	0.5	5.43	43%	1.36	0.14	115	14.94
R4200	80	117	1300	0.6	6.03	31%	1.87	0.09	55	22.53
MC68040	40	164	1170	0.65	7.64	19%	2.62	0.07	35	44.3
R4400	200	134	2300	0.35	12.02	26%	2.14	0.25	117	50.86
21064AA	200	178.5	1750	0.68	7.83	17%	2.86	0.07	130	52.22
R4600	67	182.4	1900	0.64	8.67	16%	2.92	0.08	92.1	59.7
PA7150	125	202	906	0.75	7.76	13%	3.23	0.05	135	65
21064	200	234	1700	0.8	7.76	10%	3.74	0.04	106.5	88.73
MPC604	100	196	3600	0.5	12.51	14%	3.14	0.14	160	96.96
MC68060	50	198	2500	0.5	12.64	14%	3.17	0.14	49	99.86
21164	320	210	2800	0.5	12.52	12%	3.36	0.14	201.5	111.31
MPC620	133	289	6000	0.5	18.45	6%	4.62	0.14	300	354.5
R8000	75	297.6	3400	0.5	19	5%	4.76	0.14	108	397.36

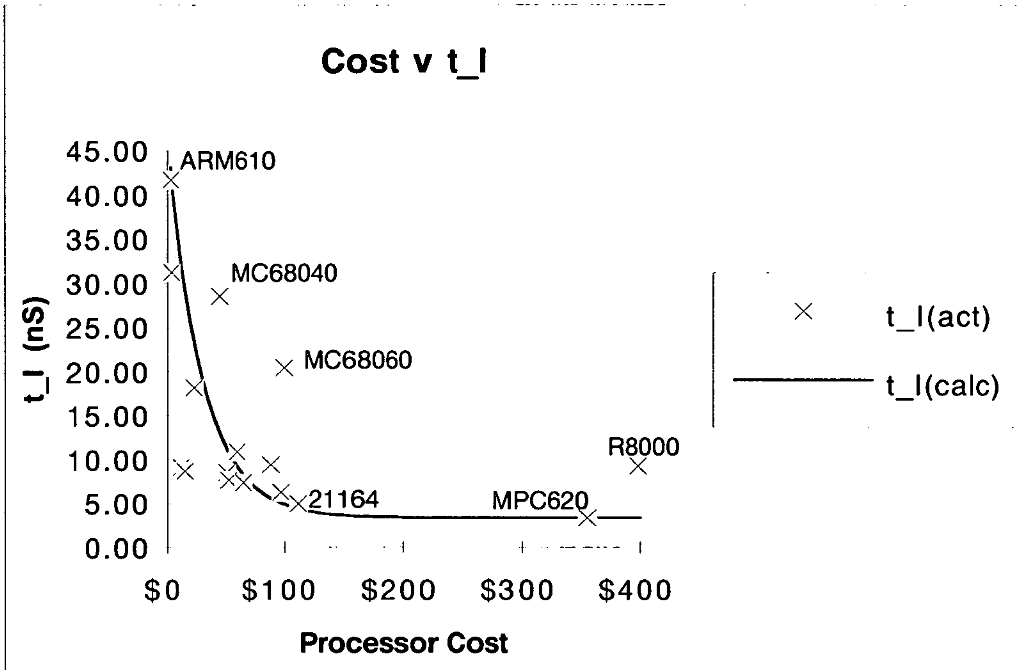


Figure 2–3: Processor Cost:Performance (t_I)

The Int92 figures in the table can be regarded as a rate of execution of work, similar to MIPS. Comparing the MIPS and Int92 values for a few devices shows that they are of the same order of magnitude (i.e. 10^6), and that MIPS is very approximately between 1 and 2 times greater than the Int92 figure. Now, this conversion can be precarious. A major limitation of MIPS is that it does not deal well with the wide range of instruction complexities available across all processors. For this reason, converting back from Int92 to a notional MIPS (i.e. by multiplying Int92 by, say, 1.5) is not generally useful. However, the relationship is mentioned to provide a sensible order of magnitude to the expression for t_I presented below. Int92 is therefore taken as a measure of millions of "Int92 instructions" per second. By expressing the computational workload in terms of such "instructions", a corresponding instruction time, t_I, can be expressed as simply the reciprocal of the performance. A graph of t_I against processor cost, C_P, is shown in figure 2–3.

Since the intention is to identify the highest available performance at or less than a given cost, it was assumed that performance is a monotonic function of cost

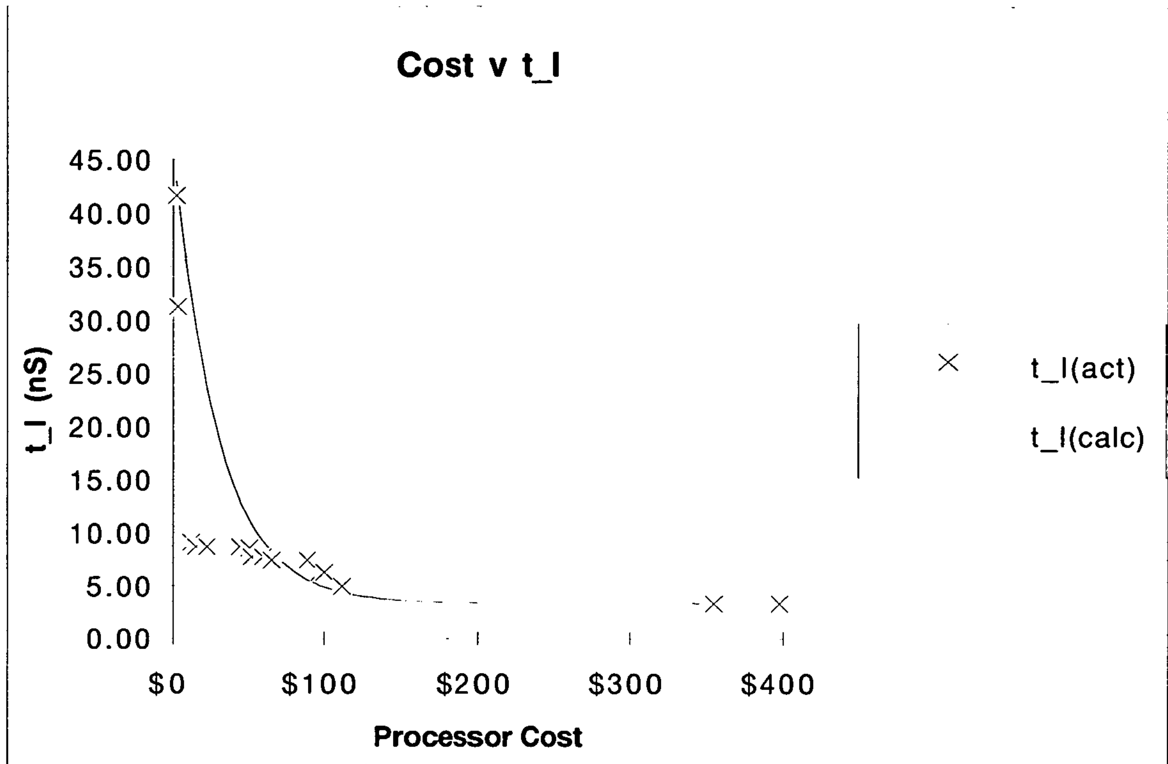


Figure 2–4: Adjusted Processor Cost:Performance (t_I)

and that the performance attainable at any given cost point was at least that at the previous point. This has the effect of removing anomolous dips in performance where a device of particularly low performance (e.g. a high-end CISC from a mature architecture) happens to possess an unusually high cost. Figure 2–4 shows t_I against C_P after removing such anomolies. Since the object here is simply to obtain an approximate relationship betwen cost and performance, strictly within the limits of the technology investigated, a visual curve fit is sufficient.

Figure 2–5 shows the reciprocal of the t_I curve, the units of speed being millions of "Int92 instructions" per second, as discussed. This performance curve has the general form:

$$t_I = l + me^{-nC_P} \tag{2.13}$$

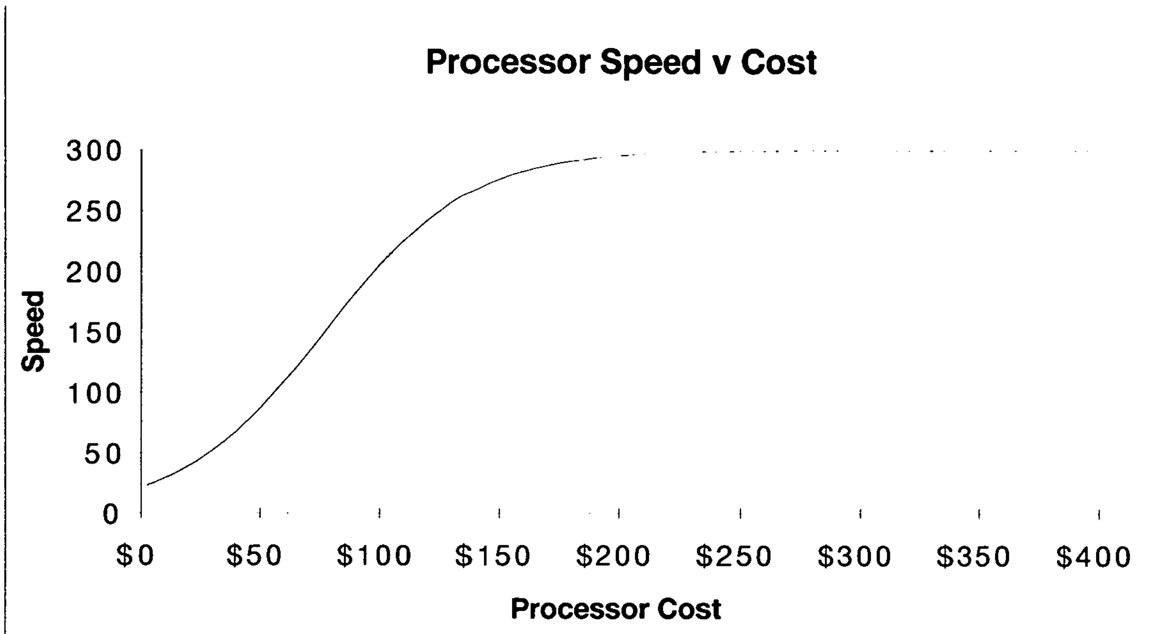


Figure 2–5: Processor Cost:Performance (millions of "Int92 instructions" per second)

where C_P is device cost, and l, m and n are positive reals. The curve shown has $l = 3.33 * 10^{-9}$, $m = 43 * 10^{-9}$ and $n = 0.1/3$ giving:

$$t_I = 3.33 * 10^{-9} + 43 * 10^{-9} e^{-\frac{0.1 C_P}{3}} \text{seconds} \tag{2.14}$$

While the relationships given are approximate, a general form of relationship can be identified. The exact nature can be made more or less accurate, as required and as available data allow.

Figure 2–5 shows the reciprocal of the above fit curve. The resulting "S" curve is typical of products implemented in a range of technologies, the oldest of which are mature and well-understood, but with significant development costs associated with the leading edge.

If the total processor cost (resource) is R_P, then C_P, the cost of a single device, is simply the total cost divided by the number of processors, t_I can now be expressed as the required function of N:

$$t_I = 3.33 * 10^{-9} + 43 * 10^{-9} e^{-\frac{0.1 R_P}{3N}} \text{seconds} \tag{2.15}$$

Equation 2.15 is the first of the three required hardware cost:performance functions.

2.3 Memory Hardware

If the space requirements of an algorithm grow as the number of processors is increased, then the speed of the memory system will decrease if either:

1. Memory subsystem performance decreases with array size, or
2. Memory component performance increases with cost

This section deals with the second of these two factors.

2.3.1 Memory Device Cost:Performance – t_M

The principal performance metric for memory devices is access time. This is usually a measure of the time delay between a valid address being presented to the device (usually qualified by a strobe signal of some sort) and a valid datum being returned. The other measure of importance is device capacity - the number of bits which can be stored. In practice the drive to increase capacity acts to restrict the device speed - the smallest cells, allowing the most dense layout, being among the slower options. To overcome this, memory designers use various architectures to enhance the access time of the slower DRAM devices. Since such devices are typically accessed using a row address component followed by a column address, it

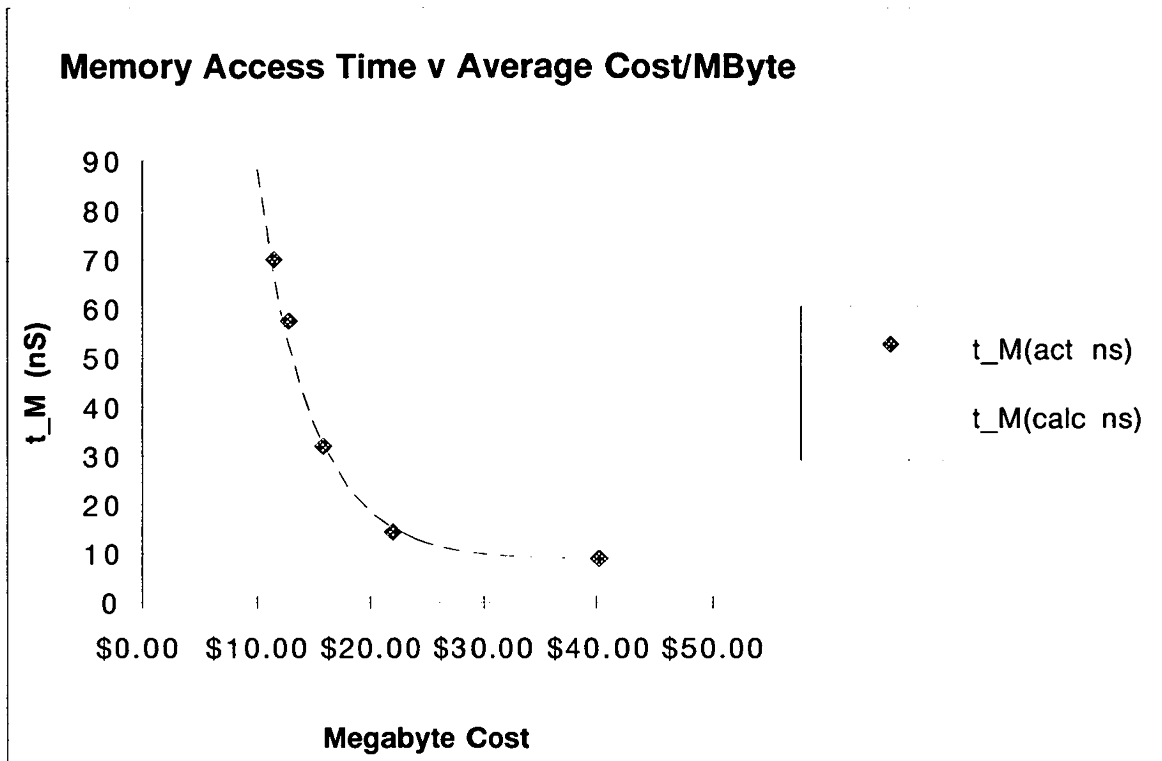


Figure 2–6: Memory Cost:Performance (memory access time)

is possible to allow fast access to a second, third or subsequent locations after an initial access has been made, provided the subsequent locations are in the same physical portion of the device as the first access. Page-mode and nibble-mode devices are examples of these techniques [52].

Despite the variation in architectures, memory devices are relatively simple compared with microprocessors, and the task of relating memory device parameters to cost and price is much easier than with the processors. Because of this, published memory device prices show a clear relationship to performance, and so will be used here.

Figure 2–6 shows memory access time as a function of the cost-per-megabyte. The latter was derived from the mean ASP (average selling price) for a range of SRAM devices in 1994 [18], and assumes a 50% mark-up of price over cost.

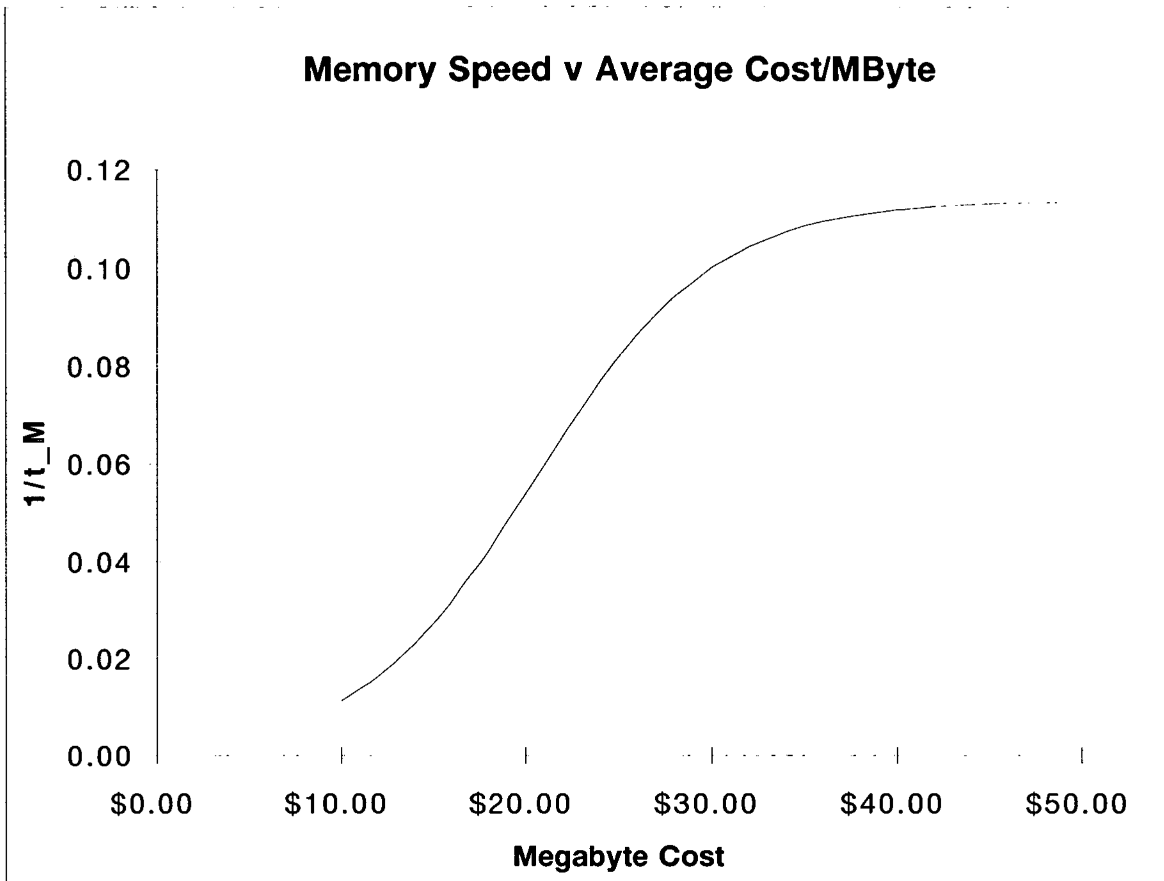


Figure 2–7: Memory Cost:Performance (memory speed)

As with processor cost:performance, a fit curve of the following form will be used:

$$t_M = l + me^{-nC_M}$$

where C_M is the cost of one megabyte. In this case, $l = 8.8*10^{-9}$, $m = 6.5*10^{-7}$ and $n = 0.21$.

Figure 2–7 shows the reciprocal of t_M.

Since the total cost, R_M, of memory must be split among the space requirements of all nodes in the system, the final expression for memory performance can now be written:

$$t_M = 8.8 * 10^{-9} + 6.5 * 10^{-7} e^{-\frac{0.21 R_M}{n_M}} \quad (2.16)$$

where R_M is total memory cost, and n_M is the total space requirement in megabytes.

Equation 2.16 shows how the performance of memory varies as the amouhnt of memory is varied and is the first of the two components needed to describe memory hardware performance. To obtain a relationship between memory performance and N, a relationship between amount of memory , n_M, and N is required. This depends on the algorithm and the base architecture and will be discussed in chapters 3 and 4.

The next section discusses the last of the three hardware functions: the interconnection network.

2.4 Interconnection Network Hardware

Interprocess or intertask communication is a distinct liability to a parallel algorithm. It is the undesirable side-effect of allocating related tasks to different processors. While dividing the work in this way produces the obvious benefit of reducing the total workload on any given processor, it has the disadvantage of forcing the processors to spend time communicating.

The basic building block of the network is the router component. The particular type used depends on the network chosen but in general these components provide the connections among processors and memory in a multiprocessor, and among PEs in a multicomputer. They provide a multiplexing/demultiplexing function, along with any associated arbitration. Routers may also provide buffering for data in transit.

Router performance is a measure of how quickly a datum arriving at an input port can be passed through to the output. Two aspects will be conisidered in this study:

- *Router Cycle Time* - t_R. This is the time between a single-bit datum arriving at an input until it is driven onto the appropriate output link.
- *Router Channel Width* - W_R. The size of datum which can be routed in a single cycle.

In this chapter, the concern is only with the speed of unloaded routers – that is, contention is not discussed. The effect of loading depends on the algorithm and the topology, and will be discussed, where appropriate, in later chapters

Typically, as N increases in a particular network, the number of routers also increases. For a fixed overall cost, this increase will result in a decrease in individual router performance, to the extent that the latter is related to cost.

Both aspects of performance, cycle time and width, are now discussed in turn.

2.4.1 Router Cycle Time

The router cycle time, t_R, will be considered a function of the technology as a whole, much in the same way as processing speed. Typical components of t_R are arbitration speed, which relates to clock frequency, and signal propagation delay. However, while there is a large range of processors in which one can investigate the relationship between cost and performance, the same is not true for network routers. Therefore, the same basic relationship between cost and performance which is used for processor speed will also be used for routers, however the coefficients and exponents must be treated with a degree of caution. Nonetheless, since the basic technologies are the same for both components, the general form of the relationship should be similar. Following equation 2.15, the router cycle time is:

$$t_R = \frac{1}{\alpha}\left(3.33 * 10^{-9} + 43 * 10^{-9} e^{-\frac{0.1 R_R}{3N}}\right) \text{seconds} \tag{2.17}$$

where R_R is the total cost of all router components and α is the number of router cycles which can be performed in one instruction time. The value of α depends on the precise way in which routing achieved. For simplicity, a value of 1 will be assumed. That is, the time to make the routing decision will be considered equivalent to the execution time of an instruction. For routing methods which can perform all of the decision-making in hardware, a larger value of α is likely.

2.4.2 Router Channel Width

The way in which the width, W_R, of the network routers varies with N depends on the particular cost metric used. The overall cost of the actual wiring between the routers may be used. This can be done, for example, by fixing the wiring density on a PCB. This is appropriate if the "costly" aspect of fabricating the interconnection is achieving the required resolution in PCB or VLSI lithography. Alternatively, if the cost of the wiring material itself is significant, then the following method could be used.

Assume that the PEs (in particular, their router components) are arranged on a planar interconnect medium with unit spacing between the eight non-local "ports" on the routers.

The total link resource (i.e. cost) is considered to be a single "wire" of length, R_L, which is divided into R_L unit-length single-bit wires. These are then grouped into as many links as are required by the whole system. The number of such unit-length wires, that is the width W_R of each link, depends on the total number of links in the system.

In a bidirectional two-dimensional square mesh, the total number of links (ignoring local links) is:

$$n_L = 4(N - \sqrt{N})$$

therefore, the channel width, $W_{R_{LCR}}$, for a link cost restricted scenario is:

$$W_{R_{LCR}} = \frac{R_L}{4(N - \sqrt{N})} \tag{2.18}$$

Note that here $W_{S_{LCR}}$ is effectively a measure of the *bandwidth* of a link. As was pointed out in [38], this is not always accurate. In systems where the messages are shorter, in length, than the link is wide then the extra width of the link cannot be used by the message. In other words no message is able to cross a channel in less than t_R. Nonetheless, for cut-through systems [39] using sufficiently large messages ($B >> W_R$), this factor is not critical. Even in short-message communication, the consequences of ignoring this effect are small if the average distance travelled by a message is sufficiently large. In that case latency is dominated by the message header establishing the connection between source and destination.

The preceding example is appropriate for a wire-limited implementation. However, an alternative method is to use the pin-out of the router device as the limiting factor in determining channel width. As VLSI technology advances, and an increasing number of transistors is implemented on a single device, the number of connections between the device and the rest of the system is becoming a bottleneck. Several approaches are being used to deal with this problem, including the development of new IC packages with smaller lead separation, and also the direct bonding of die to a substrate to produce a Multi-Chip Module. Regardless of the approach taken, the number of connections which may be made to a device is, after fixing other crucial variables such as the package material, strongly related to the manufacturing cost of the device. The effect of increasing pin-out has an even stronger (increasing) effect on device price.

The proportion of overall device cost due to packaging varies depending on the device size and complexity, and on the package types. In general, package cost is more significant for smaller parts, although newer package technologies (e.g. Ball Grid Array) can account for a significant portion of the cost of even larger devices. An average of a third will be assumed [10] and, following equation 2.7, the total pin-out of a router is given by:

$$P = \left(\frac{R_R}{2.7 * 10^{-3} N}\right)^{\frac{1}{1.7}}$$

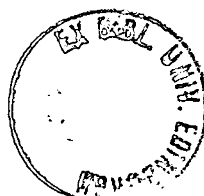
This assumes one processor per router. Assuming that the router pins are divided only among n_c communication channels (i.e. ignoring power, ground and other pins), the channel width for a router cost restricted scenario is:

$$W_{R_{SCR}} = \frac{1}{n_c}\left(\frac{R_R}{2.7 * 10^{-3} N}\right)^{\frac{1}{1.7}} \quad (2.19)$$

Since this expression provides a relationship between actual dollars and the channel width, it will be used for the remainder of the analysis. In the situations where link resource is the key factor, then equation 2.18 may be more appropriate.

2.5 Summary

This chapter has presented analyses of the factors affecting cost and performance for the components used to implement the three basic functions of a parallel computer: processing, memory and interconnect. In particular, expressions were derived for t_I (instruction execution time), t_R (router cycle time), W_R (router width) and t_M (memory access time). Each of these was given as a function of the cost of the component concerned.



These functions will now be used in the development of expressions for the execution times of algorithms on the two architecture types mentioned in section 1.4.

Chapter 3

A Shared-memory Multiprocessor

3.1 Introduction

In this chapter, a model of performance is developed for the shared memory paradigm running on a simple shared-bus multiprocessor. First the architecture of the machine is described, and then the algorithm is presented in terms of an equivalent sequential "critical path". These are combined to provide an expression for the execution time as a function of N, the number of processors.

3.2 Machine Architecture

The shared memory multiprocessor will consist of an ensemble of processors connected, by a single bus, to a shared store. (Figure 3–1).

Each processor also has a private memory used for storing programs and for local temporary variables. A single arbitor controls the bus and operates a round-robin scheme among requesting processors, using individual Bus_Request and Bus_Grant signalling. Once a processor has been given control of the bus, it retains tenure for as long as necessary. All memory accesses during this time are direct, the processor stalling while an access is satisfied. This is in contrast to posted accesses, where the processor presents a request and then continues with other work until interrupted by the memory with the result of the access.

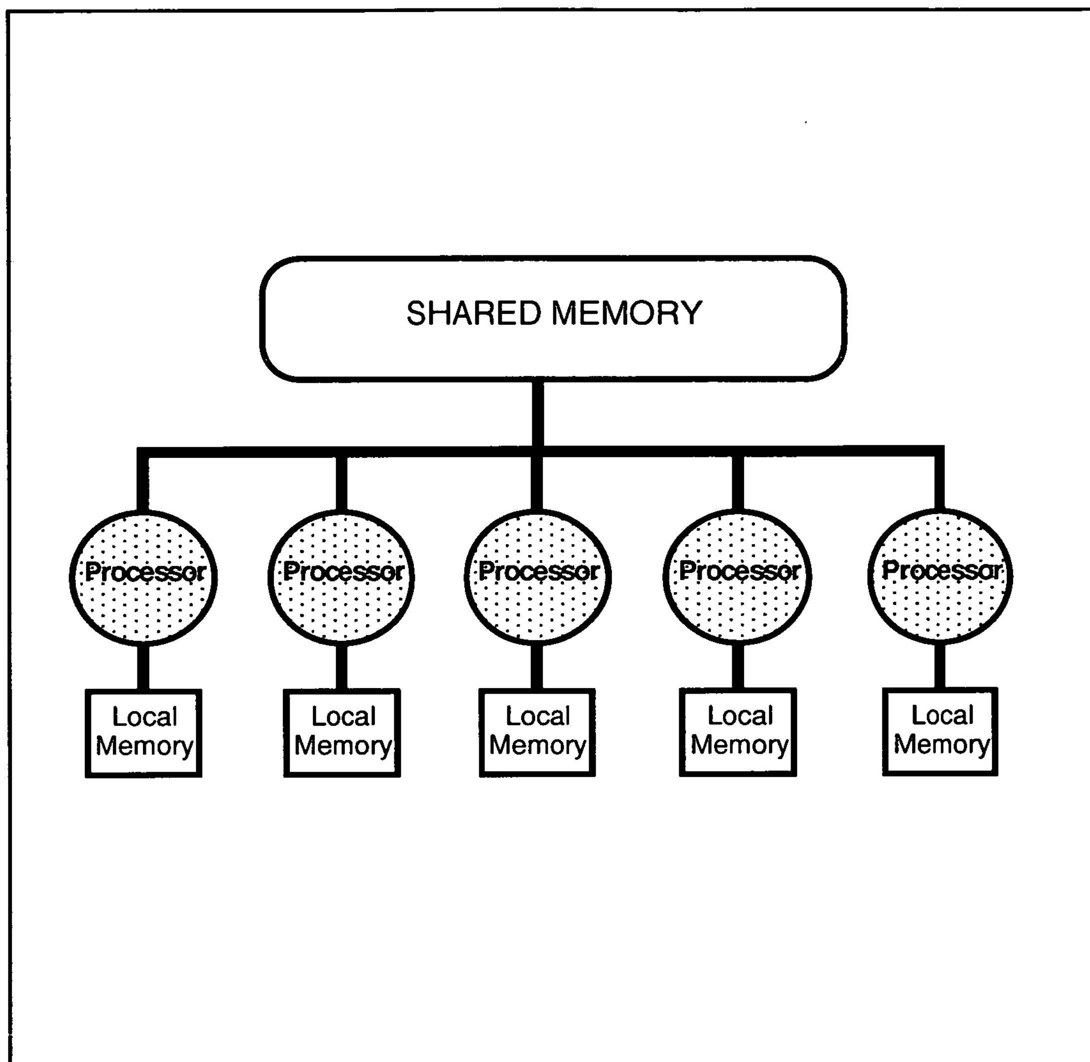


Figure 3–1: Shared Memory Multiprocessor

3.3 The Algorithm

The algorithm will consist of a series of arithmetic and logic operations performed on a data set of P points in the shared store. This corresponds to several common computations; for example, redex reduction in a graph representation of a functional program [58], or a matrix computation such as that described in [47]. Each set of operations will be preceded by a read, from the shared store, of one data point. Each point consists of a single word in shared memory. A series of i_c instructions is then performed on the datum, along with some associated private memory activity. The result is then written back to the shared store. Processors will continue in this way until all data have been processed.

3.3.1 Critical Path

Based on the above description of the algorithm, a processor in the shared memory system will, at any given time, be engaged in one of the following:

1. **Instruction Execution.** These are the instructions performed on the data point after retrieving it from the shared store.

2. **Local Memory Access.** This refers to an access to a PE's own private (and local) store. This could be an instruction fetch, or manipulation of a temporary, private variable.

3. **Shared Memory Access.** In contrast with 2., this is an access to the main shared store and as such may be subject to blocking by similar accesses from other processors. This includes time spent waiting for access to the shared bus.

4. **Idling.** This is time spent completely idle with no data points available requiring processing. This could occur towards the end of the computation when some processors have completed their last write-back and are waiting for the remaining processors to complete theirs.

If the situation arises, in the execution of the computation, where all processors have become idle and have no further work scheduled, then the algorithm can be considered to have terminated. Discounting memory stalls, where a processor is actually waiting for the response from another piece of hardware, a system-wide halt in work marks the end of the algorithm. Therefore, *during* the computation, there must exist, beginning at time zero, and ending at the termination, at least one *Critical Path* (CP) consisting of an unbroken sequence of operations (including memory stalls). The process of assessing the execution time of the parallel algorithm is therefore equivalent to obtaining the execution time of the sequential algorithm represented by the CP.

So, by restricting the analysis to the CP, item 4 can be ignored and the execution time of the algorithm is equivalent to the execution time of a sequential program consisting of some distribution of operations 1 to 3 in the above list. The execution time then becomes the sum of products of the time to perform of each of those operations with the number of occurrences of each:

$$T_{SM} = It_I + M_l t_{M_l} + M_s t_{M_s} \tag{3.1}$$

where:

T_{SM} = execution time

I = total number of instructions on critical path

t_I = time to execute one instruction

M_l = total number of local memory accesses on critical path

t_{M_l} = time for local memory access

M_s = total number of shared memory accesses on critical path

t_{M_s} = time for shared memory access

It is assumed that local memory references constitute a constant fraction of instruction executions, μ, which is independent of the number of processors. That is, M_l can be expressed as:

$$M_l = \mu I$$

and so the execution time is:

$$T_{SM} = I(t_I + \mu t_{M_l}) + M_s t_{M_s} \quad (3.2)$$

The next two sections examine the parameters in equation 3.2 for dependency on N. The parameters are divided into two types; I and M_s describe the operations on the critical path, and t_I, t_{M_l} and t_{M_s}, describe the times to perform those operations. The former will be referred to as *software* parameters, and the latter as *hardware* parameters.

3.4 Software Parameters

It will be assumed that each processor will deal with approximately the same number of data points as the others. Therefore, the critical path for N nodes operating on P points is simply the processing of $\lceil \frac{P}{N} \rceil$ points. Assuming $P \gg N$, the number of points per node will be considered as $\frac{P}{N}$ for all nodes. If each point requires i_c instructions, then:

$$I = \frac{i_c P}{N}$$

Although each point requires two accesses to shared memory the write from one point will be performed on the same bus tenure as the read of the next point. This can be considered as a single memory access of twice the duration of a single read or write (both assumed equal). This doubling of shared memory access time is introduced to the analysis in equation 3.5 and allows M_s to represent the number of such double accesses thus:

$$M_s = \frac{P}{N}$$

Equation 3.2 can now be rewritten as:

$$T_{SM} = \frac{i_c P}{N}(t_I + \mu t_{M_l}) + \frac{P t_{M_s}}{N} \tag{3.3}$$

The next section looks at the three hardware parameters, t_I, t_{M_l} and t_{M_s}.

3.5 Hardware Parameters

The instruction execution time, t_I is as given by equation 2.15:

$$t_I = 3.33 + 43e^{-\frac{0.1R_P}{3N}}\text{nanoseconds}$$

As discussed in section 2.3.1 memory performance is related to cost, and so to the amount of memory. The access time of a single memory device in the system is given by 2.16 and is therefore:

$$t_M = 8.8 + 650e^{-\frac{0.21R_M}{(n_{M_l}+n_{M_s})}}\text{nanoseconds}$$

Where n_{M_l} is the total amount of local (private) memory for all processors, and n_{M_s} is the total amount of shared memory in the system as a whole. It is assumed that the same type of memory device is being used to implement both local and shared store. If different device types were used (e.g. faster, more expensive SRAM for the local store, and cheap DRAM for the shared memory), then the two would be analyzed separately but using the same basic method.

Regardless of which devices are used, any relationship between memory performance and N depends on how n_{M_l} and n_{M_s} scale with the number of nodes.

3.5.1 Space Scaling

The total memory required (local or shared) to run a piece of software on a parallel machine can be split into three components as follows:

$$n_M = n_{M_{SYS}} + n_{M_{ROUTE}} + n_{M_{ALG}}$$

where:

$n_{M_{SYS}}$ = *System* memory. This is the total memory required for the operating system which supports the running of the software in question.

$n_{M_{ROUTE}}$ = *Routing* memory. This is memory for routing tables, and any other storage required for communications.

$n_{M_{ALG}}$ = *Algorithm* memory. This is the basic memory required by the computation alone. It includes the storage required for the program code, the problem data, the solution, and any intermediate storage.

$n_{M_{SYS}}$ is a result of the fact that computers are rarely used for a single computational purpose. The manufacturers of even the most application-specific systems will sell them to a reasonably varied market, and support software must be in place to act as a base upon which the equally varied applications can be built. That notwithstanding, system software will generally tend to be as small a portion as possible of the total requirements. Also, there is a growing use of high-performance hardware in embedded control applications in which there is little or no need for any operating system or other support software. For example, in computation intensive applications such as character recognition and raster image processing, multiple-processor embedded systems are beginning to emerge. Therefore, it will be assumed that, in general, $n_{M_{SYS}}$ will scale linearly with N (e.g. a microkernel will be placed on each PE) but that it will be small enough compared with the other two components to be ignored in most cases.

Routing memory, $n_{M_{ROUTE}}$, is more fundamentally linked to the running of software on a parallel machine. While a system requiring no support software can be envisaged, communication information is an essential and unavoidable requirement. In the MIMD paradigm, an $O(N)$ scaling is possible for fixed size routing tables, and $O(N^2)$ if the tables themselves scale linearly with N. However, it will be assumed that the relative size of routing space is small compared with $n_{M_{ALG}}$

in systems of N below a particular upper bound. Provided N does not scale above this limit, the space requirements will be dominated by the memory required for the algorithm itself.

$n_{M_{ALG}}$ is the space required for the algorithm code, and any data space required. As mentioned above, the data space includes both the input and output data sets, and any excess intermediate storage (if required).

For the multiprocessor system in question, the shared space will remain of constant size, and independent of N. Space scaling effects will therefore be seen only in an increase in the size of the processors' local memories. If each processor requires m bytes of local memory, then $n_{M_l} = mN$. For a constant n_{M_s} the local access time is therefore:

$$t_{M_l} = 8.8 + 650e^{-\frac{0.21R_M}{(mN+n_{M_s})}} \text{nanoseconds} \tag{3.4}$$

The shared memory access time, t_{M_s} depends on the same function, but is also affected by contention on the shared bus.

3.5.2 Shared Memory Performance

Access time to the shared store includes waiting time on the shared bus, and is derived as follows.

The combined bus and shared memory will be modelled as a single server queue with deterministic service times. The average effective access time, t_{M_s} is given by:

$$t_{M_s} = (N_Q + 1)2t_{M_l} \tag{3.5}$$

where N_Q is the average number of existing bus requests seen by a processor when asserting its own request line and t_{M_l} is as in equation 3.4. The factor of

two represents a double access, the write-back of a result, followed by the read of the next point.

Since a processor will block until its request has been serviced, the system is closed and the average arrival rate of jobs into the queue depends on how many processors have *not* yet made (ungranted) requests. Intuitively, a slow memory coupled with a large number of fast processors will result in the queue filling rapidly until $N_Q \approx N$. Conversely, a few slow processors connected to a very fast store would result in the fast servicing of requests and a relatively small N_Q. From the point of view of a processor making a request, the arrival rate, λ, of requests to the queue, is given by:

$$\lambda = \frac{N - N_Q - 1}{T_P}$$

where $N - N_Q - 1$ is the number of processors which do not yet have requests placed with the bus arbitor. T_P is the time such a processor spends between the completion of one shared access until it makes its next request.

The queue service rate, ν, is simply the reciprocal of twice the basic access time as given by equation 3.4:

$$\nu = \frac{1}{2t_{M_l}}$$

Assuming balanced flow, the flow of requests will stabilize at the bottleneck rate where $\lambda = \nu$, imposed by either the memory or the processors themselves. That is:

$$\frac{N - N_Q - 1}{T_P} = \frac{1}{2t_{M_l}}$$

The number of processor requests seen by any given requesting processor is thus given by:

$$N_Q = N - 1 - \frac{T_P}{2t_{M_l}}$$

for $\frac{T_P}{t_{M_l}} \leq (N-1)$. For $\frac{T_P}{t_{M_l}} > (N-1)$, $N_Q \approx 0$.

Processing time, T_P, is simply the time spent operating on a data point, in between bus requests:

$$T_P = i(t_I + \mu t_{M_l})$$

and so the expression for shared memory access time, including bus waiting is:

$$\begin{aligned} t_{M_s} &= \left(N - \frac{T_P}{2t_{M_l}}\right) 2t_{M_l} \\ &= \left(N - \frac{i(t_I + \mu t_{M_l})}{2t_{M_l}}\right) 2t_{M_l} \end{aligned}$$

Again, this is valid for for $\frac{T_P}{t_{M_l}} \leq (N-1)$. For $\frac{T_P}{t_{M_l}} > (N-1)$, the queue length tends to zero, and the shared memory access time is the basic access time given by equation 3.4, that is:

$$t_{M_s} = 2t_{M_l}$$

The shared memory access time is therefore given by:

$$t_{M_s} = \left(N - min\left[(N-1), \frac{i_c(t_I + \mu t_{M_l})}{2t_{M_l}}\right]\right) 2t_{M_l} \qquad (3.6)$$

where $min[x, y]$ is the lower of x and y. Note that these expressions for memory performance ignore the effect on memory array performance of an increase in the physical size of the array. Two examples are increased wire lengths and increased decode/buffer stages. On a planar memory array, the distance from an input port

to the most distant cell is $O(\sqrt{n_M})$. For very fast cells in large arrays, this increase in wire delay may become significant. Decoding and buffering for the purpose of satisfying device fan-out requirements may also give rise to increasing delays as the number of cells in the array grows. Currently, these delays are often absorbed in the early stages of memory access cycles by presenting the required address well in advance of the point in time at which the processor actually latches the data. As cycles become shorter, however, and processors increase in speed, the time required to decode the address (possibly 64-bits wide) down to the appropriate device chip select and address portion may become significant. Nonetheless, it is the speed of the memory devices themselves which is the more dominant factor, and the main point of concern here. The access time of the memory system will be assumed to be the access time of the devices from which the system is built, and the key effect of any space scaling will be, as with the processors themselves, to change the cost of the devices.

Equation 3.6 can now be combined with equation 3.3 to give:

$$T_{SM} = \begin{cases} \frac{P}{N}(T_P + 2t_{M_l}) & \text{when } N \geq \frac{T_P}{2t_{M_l}} + 1 \\ 2Pt_{M_l} & \text{when } N < \frac{T_P}{2t_{M_l}} + 1 \end{cases}$$

where:

$$T_P = i(t_I + \mu t_{M_l})$$

From the above it can be seen that once the bus begins to load, any dependence of T_{SM} on N stems solely from N-dependency in t_I or t_{M_l}. This is because the linear decrease in the number of points to be operated upon by any given processor is countered by a similar *increase* in the waiting time on the bus. This effect will be seen in more detail in chapter 5.

3.6 Summary

Expressions have been derived for the various components of equation 3.2, as functions of N, the number of processors in a shared memory multiprocessor. These functions depend on the technology cost:performance functions developed in chapter 2, and also upon the shared bus architecture described in this chapter.

The effects of the various components of the execution time will be investigated in chapter 5, and simulation results of a shared bus system under these cost:performance constraints will also be presented.

Chapter 4

A Message-passing Multicomputer

4.1 Introduction

This chapter presents a model of performance for a computation following the message-passing paradigm, running on a mesh-connected multicomputer. As before, the architecture of the machine is described, and then the algorithm is presented in terms of an equivalent sequential "critical path". These are combined to provide an expression of the execution time as a function of N, the number of processors.

4.2 Machine Architecture

In the multicomputer, all memory will be considered private to the individual processors. In addition to its memory, each processor also has a router through which it connects to the other nodes. The combined processor/memory/router entity is termed a processing element (PE) and these are arranged in a two-dimensional bi-directional rectangular mesh. The edges of the mesh are left unconnected, in contrast with the torus where the edges wrap round. The router is effectively a 5x5 (links in directions "north", "south", "east", "west" and "local") crossbar switch allowing non-blocking linking between any pairs of source and destination links (although higher-level routing strategies may restrict this).

4.3 The Algorithm

The algorithm to be modelled can be represented graphically by a set of layered, rectangular graphs as described in the next section.

4.3.1 A Graph Model of Algorithms

A processor *operation* will be considered to be an instruction or a memory access. An operation is said to *depend* on another if the former may not begin until the latter has completed. The *fan-in*, f_{IN}, of an operation, is the number of operations upon which it depends. The *fan-out*, f_{OUT}, of an operation is the number of operations which depend upon it. A *task*, $\gamma_i = \{I_0, I_1, ..., I_{n-1}\}$ is a set of n operations where γ has the following properties:

1. I_0 has either:
   (a) $f_{IN} = 0$ *or*
   (b) $f_{IN} = 1$ and depends on an operation with $f_{OUT} > 1$ *or*
   (c) $f_{IN} > 1$
2. I_j $(0 < j < n)$ depends on I_{j-1} only.
3. I_{n-1} has either:
   (a) $f_{OUT} = 0$ *or*
   (b) $f_{OUT} = 1$ and is depended upon by an operation with $f_{IN} > 1$ *or*
   (c) $f_{OUT} > 1$

Intuitively, a task is any set of operations each of which depends solely on predecessors within the set, *except* for the first and last operations in the set.

Let $\Gamma = \{\gamma_0, \gamma_1, ..., \gamma_{n-1}\}$ be a set of tasks, and let $\Delta = \{(\gamma_i, \gamma_j) : i, j = (0, 1, ..., n-1)$ and $i \neq j\}$ be a set of precedence relations among the members of Γ such that if $(\gamma_i, \gamma_j) \in \Delta$ then the first operation in γ_j depends on data from the last operation in γ_i. γ_i is said to be a *parent* of γ_j. γ_j is a *child* of γ_i.

Denote by $\Lambda = (\Gamma, \Delta)$ a *Directed Graph* where each task in Γ is a node of Λ and each relation in Δ an edge of Λ. A *cycle* of Λ is a closed walk such that the direction of the walk is as described by the precedence defined by Δ.

The algorithm will be represented by a connected Directed Acylic Graph - see Figure 4–1. This is a more exact model than the time-averaged process graph, [9], in which the nodes represent processes and the arcs the channels of communication between the processes. Unlike a task, a process may send and receive data throughout its lifetime, and may follow the sending of a message with further processing. That model provides a higher-level view of the behaviour of the software, and may be more intuitive for many purposes. However, the task-based DAG lends itself more easily to the extraction of a set of parameters describing the load on the underlying hardware, and so is the method used here.

In figure 4–1, γ_0 to γ_{13} are tasks and the directed arcs represent dependencies – e.g. task γ_6 cannot begin execution until tasks γ_3 and γ_4 have completed and sent their messages.

Each task, $\gamma_i \in \Gamma$ can be assigned an integer, l_i, called its *level* according to the following method.

Let f_{OUT_i} be the *fan-out* of task γ_i; i.e. the number of arcs directed out of the task. Similarly, let f_{IN_i} be the *fan-in* of task γ_i; i.e. the number of arcs directed into the node representing that task. For each task, let $outcount_i = f_{OUT_i}$ and $incount_i = f_{IN_i}$.

The procedure for assigning levels to the tasks is as follows:

1. Initially, let $l_i = 0$ for all tasks.

2. Find a task, γ_j, with $incount_j = 0$ and $outcount_j > 0$.

3. For each child of task γ_j:

    (a) Denote child task as task γ_c.

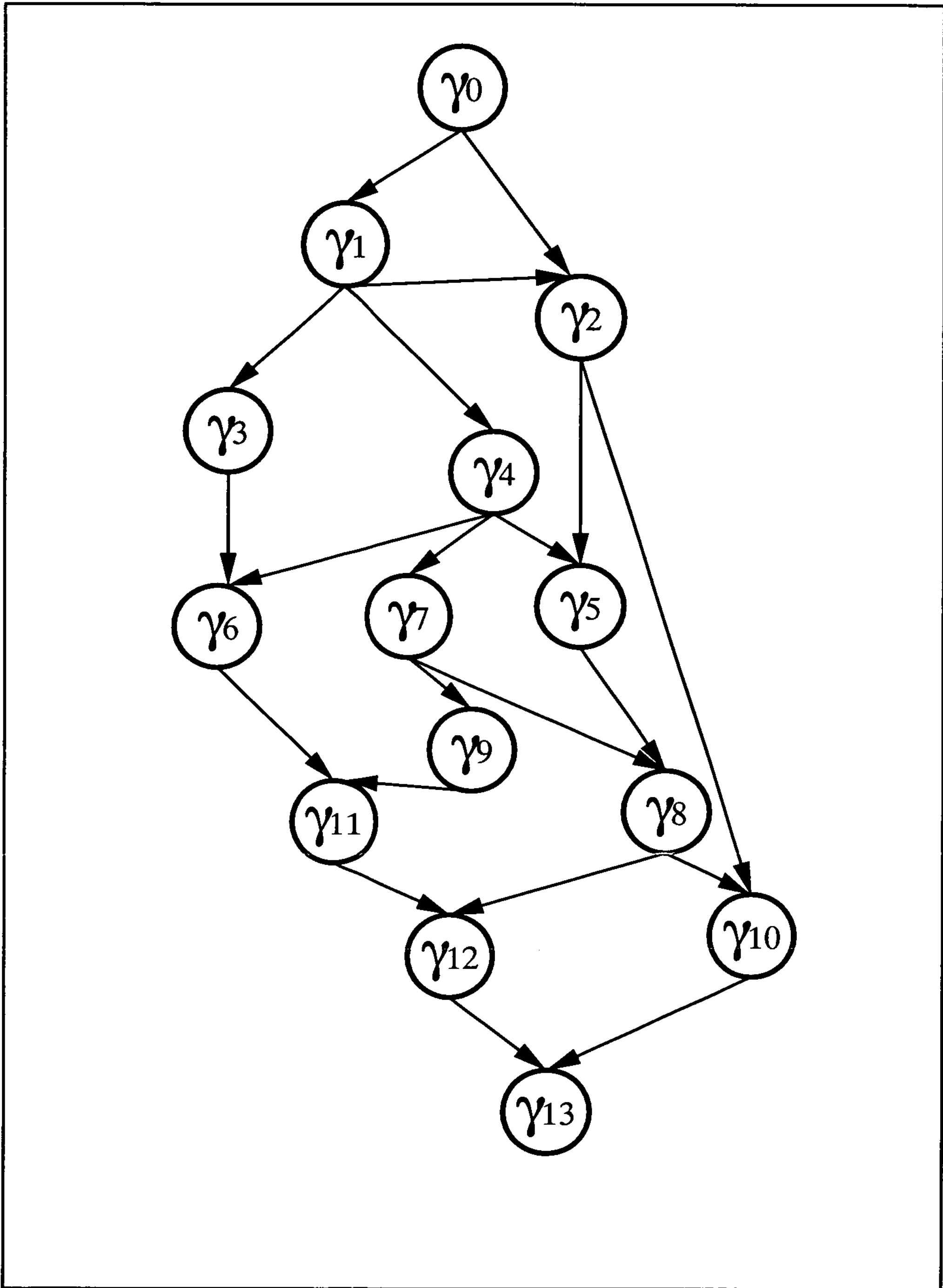


Figure 4–1: Directed Acyclic Graph

(b) Make $l_c = max\{l_j + 1, l_c\}$.

(c) $incount_c = incount_c - 1$.

(d) $outcount_j = outcount_j - 1$.

4. Repeat from 2 till $incount_i = 0$ for all tasks $\gamma_i \in \Gamma$.

Figure 4–2 shows the DAG in Figure 4–1 with tasks on relevant levels. Note that this method assigns the smallest possible l_i to task γ_i. Task γ_{10}, for example, could also have been given level $l_{10} = 6$ without violating the order imposed by Δ. If the task with the highest level number is denoted γ_x then let the dag *depth* be given by $d = x + 1$. Denote the number of tasks at a given level, l, by w_l (width).

Define a *layered* DAG to be one in which the following condition holds:

$$(\gamma_i, \gamma_j) \in \Lambda \Leftrightarrow j = i + 1$$

A *rectangular* DAG is one in which w_l is constant for all l. In this case, the subscript will be dropped and DAG width will be denoted simply by w.

The DAG can be represented by the following parameters:

$$\begin{aligned} w &= \text{DAG Width} \\ d &= \text{DAG Depth} \\ f &= \text{Fan-Out} \\ j &= \text{Instructions per task} \\ \mu &= \text{Memory accesses per instruction} \end{aligned}$$

4.3.2 Grid Algorithm

The specific algorithm used to investigate the message-passing paradigm is the transformation of an n-dimensional data space of size P, over s (steps) iterations. This type of algorithm is found in numerous simulation scenarios, a simple example

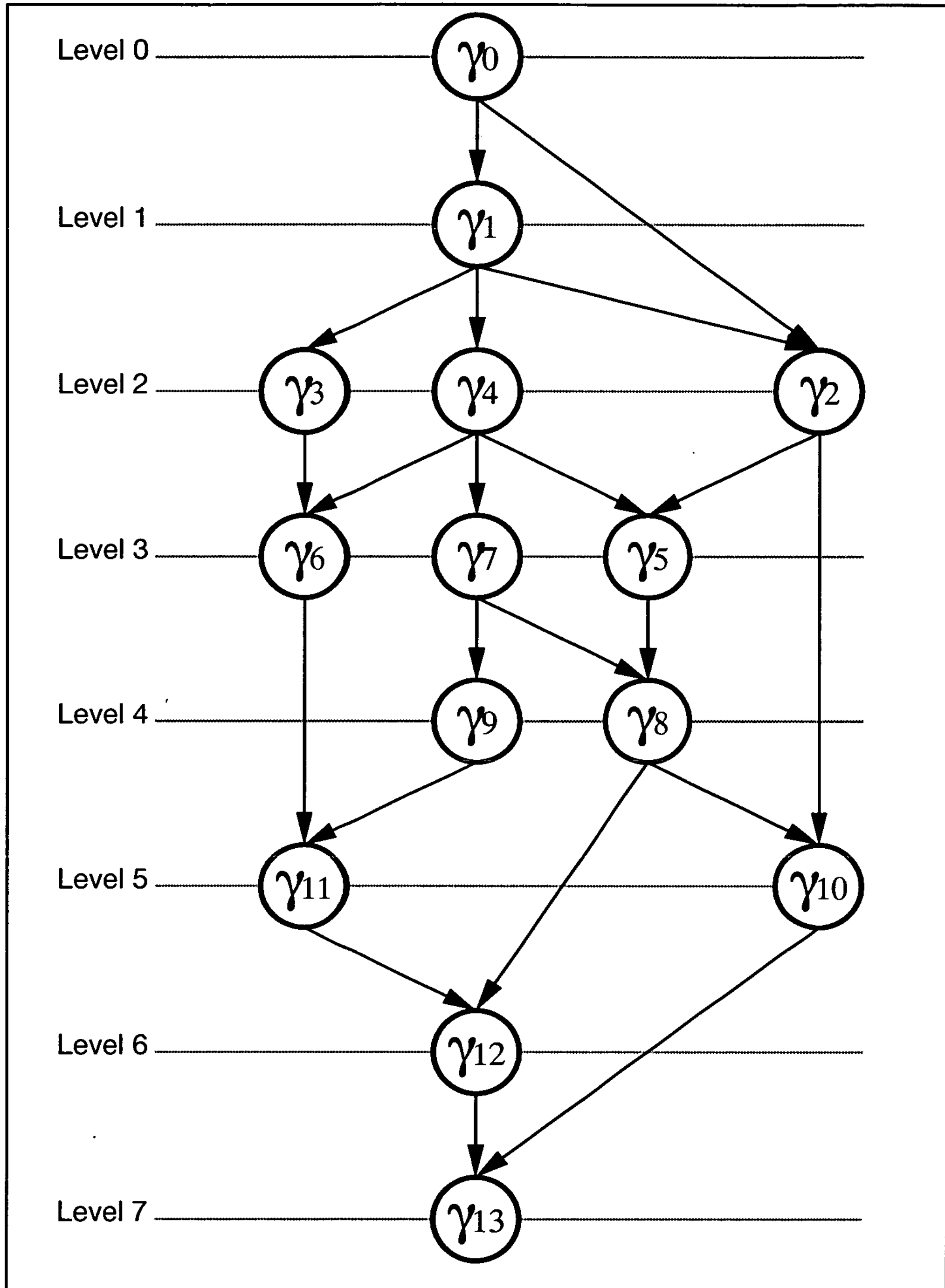


Figure 4–2: Levelled DAG

being the modelling of heat flow across a metal sheet. The data space is divided into an n-dimensional grid, and each processor is allocated a portion of the grid. At each time iteration, the processors perform i_c instructions on their own grid points. This will involve μi local memory accesses. At the end of an iteration, all points interacting with points on another PE will be combined into a message and sent on. Reciprocal message sends will be expected.

This can be represented by a set of DAGs with the following parameters:

$$\begin{aligned} w &= N \text{ (DAG width)} \\ d &= s \text{ (DAG depth)} \\ j &= \tfrac{Pi}{N} \text{ (Instructions per task)} \\ f &= 2n \text{ (Fan-out)} \end{aligned}$$

Ignoring edge effects, the dependencies among the points in the data space will be described by the *interaction depth*, $\delta \in \{0, 1, 2, \ldots\}$, as follows:

Point $A = (a_1, a_2, \ldots, a_n)$ interacts with point $B = (b_1, b_2, \ldots, b_n)$ if:

$$\exists j (1 \leq j \leq n) \text{ such that } \forall i (1 \leq i \leq n) \begin{cases} a_i = b_i & \text{if } i \neq j \\ 0 < |a_i - b_i| \leq \delta & \text{if } i = j \end{cases}$$

for positive integers i and j:

For example, in the case of a two-dimensional problem such as sheet heat flow, with $\delta = 1$ each grid point interacts with four neighbours. Mapped onto a 2-D mesh of processors, this would result in each physical PE also having four neighbours requiring messages.

Message sends will be completely non-blocking; that is, the only activity required of the processor to send a message is to prepare it for the router. Sufficient buffering is assumed to allow the processor to return to its next task after preparing the message from the previous one. Incoming messages from other nodes will be processed fully by the router before being presented to local memory. Stripping

of address information, and error detection/correction etc. are all the responsibility of the router and will not affect the processor. For simplicity, the rest of this analysis will be of a two-dimensional decomposition with $\delta = 1$.

4.3.3 Critical Path

At any point in time, a given processor in the mesh will be doing one of the following things:

1. **Computation Proper.** This is the computation associated with the algorithm itself, as would be expected of the same (parallel) code running on a single-node system. It does not include any overhead computation associated with the preparation of messages.

2. **Memory Reference.** Note that all memory references are local and not subject to sharing among processors. It will be assumed that there is no contention between the router and the processor.

3. **Sending** a message. This is the time from when a processor has to halt computation proper to begin sending a message, until it restarts computation after processing the message. The time taken to perform a send can be a significant portion of the overall time spent communicating [53]. It may involve the generation of an address header, and an error control datum. In a real system an operating system call may be required. Alternatively it may simply involve the processor providing a memory pointer packet to the router, the latter then taking complete control of the send. Regardless of the amount of work required in actually getting the message onto the network, only that work performed by the processor itself is considered here.

4. **Receiving** a message. This is the time from when a processor has to halt computation to wait for a communication, until it restarts computation after receiving the required message.

5. **Idling**. This is time spent idle with no task awaiting execution. This may occur in algorithms in which the parallelism varies with time. An example is at the start or end of a computation, where most nodes have either not begun work, or have completed their last allocated tasks and are waiting for work to begin, or for more heavily loaded nodes to complete.

As before, by restricting the analysis to the CP, idling can be ignored. The execution time is thus:

$$T_{MP} = It_I + M_l t_{M_l} + St_S + Rt_\rho \qquad (4.1)$$

where:

T_{MP}	=	execution time of mapped algorithm
I	=	number of computation instructions on critical path
t_I	=	time to execute one instruction
M_l	=	number of local memory accesses on critical path
t_{M_l}	=	time to access local memory
σ	=	number of message sends on critical path
t_σ	=	time to send a message
ρ	=	number of message receives on critical path
t_ρ	=	time to receive a message

All instructions will be considered equal. For example, no provision is made for mixes of floating-point versus integer arithmetic. However, adding such a detail would only be relevant to this study if the relative **fp** to **int** speed (for example) was variable depending on the cost of the processor. If this was the case then

the model could be extended to account for the effect by splitting I into several instruction types and including terms corresponding to the execution times of instructions of the various types.

Any computation associated with through-routing (i.e. the handling of messages in-transit passing through a node in this point-to-point topology) is not explicitly included here. In fact such computation is taken into consideration not in the processing components themselves but in the router components which make up part of the Interconnection Network.

Since each iteration ends with each node sending messages to its neighbours, the sends themselves can be overlapped by the time spent waiting for the reciprocal messages to arrive. The execution time is thus:

$$T_{MP} = I(t_I + \mu t_{M_l}) + Rt_\rho \tag{4.2}$$

where $\mu = \frac{M_l}{I}$ is the memory references per instruction.

The parameters in equation 4.2 are now examined for dependency on N. As before, those describing the nature of the critical path – I, and R – will be termed *software* parameters, and those describing the speed at which those three are performed – t_I, t_{M_l} and t_ρ – will be termed *hardware* parameters.

4.4 Software Parameters

The grid algorithm allows an even division of points among the PE's and so the number of instructions per node is given by:

$$I = \frac{Psi}{N}$$

It is assumed that each new iteration will not begin until all four (in the 2-D decomposition in question) messages have been received from the neighbouring nodes. Therefore, it is useful to consider the receipt of these as a single phase at the end of each iteration. The time to receive will reflect this, and R is simply the number of iterations, s.

4.5 Hardware Parameters

As before, t_I is given by equation 2.15:

$$t_I = 3.33 + 43e^{-\frac{0.1R_P}{3N}} \text{nanoseconds}$$

The remaining two parameters, t_{M_l} and t_p are more complex and will be dealt with in turn in the next two sections.

4.5.1 Memory Performance - t_{M_l}

The basic device access time is given by equation 2.16, but the actual performance depends on the cost of those basic devices; this in turn depends on space scaling on the mesh.

As with the shared memory system, it will be assumed that the memory cost is dominated by algorithm memory, $n_{M_{ALG}}$ (see page 61). However, contrastingly, in order to highlight the effects of space scaling in the grid decomposition, in this section only data space memory will be considered. Including program memory would serve only to add an extra scaling component which was linear with N and which may hide the more interesting effects caused by the overlap area method of grid decomposition. Code space could be included in the model simply by adding the appropriate linear function of N to equation 4.4

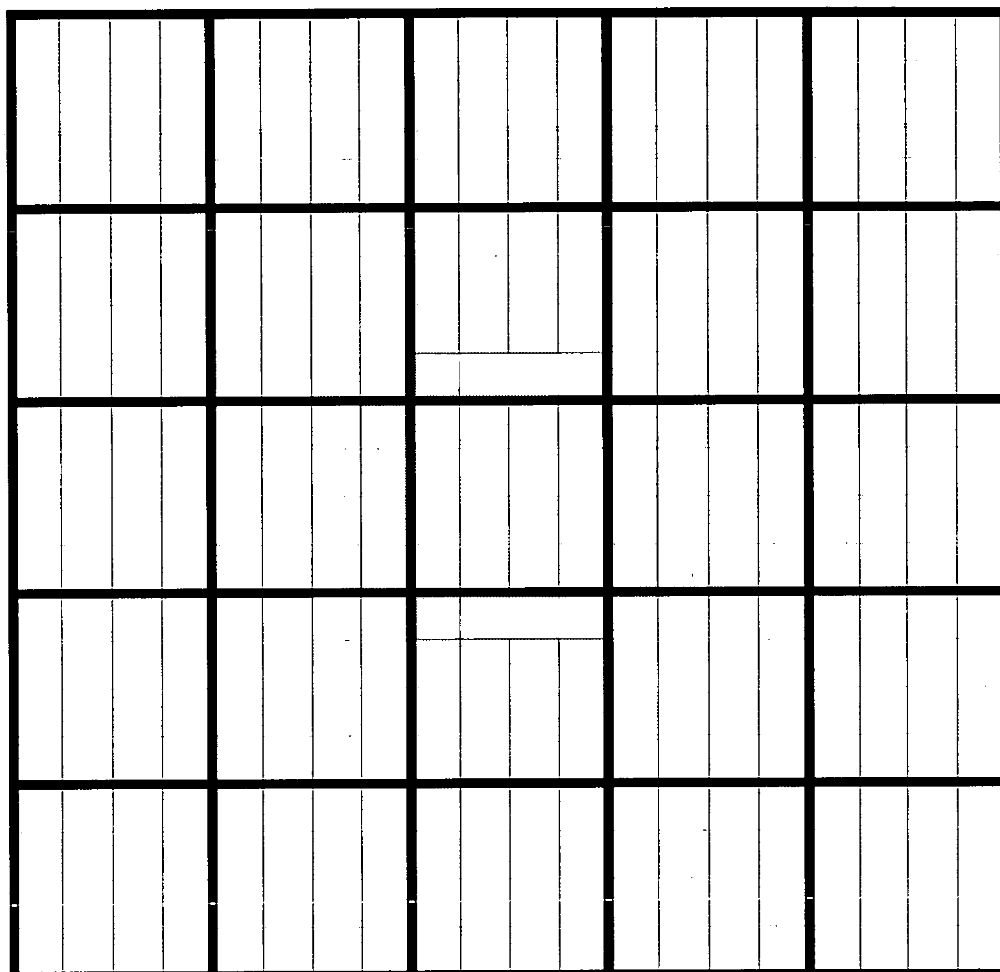


Figure 4–3: Overlap Areas

For the grid algorithm, each PE will require storage not only for its allocated grid points, but also for the "edge" data incoming from its neighbouring nodes. This is an example of "overlap areas" [67][78], the allocation of a "halo" of memory around the main data cell into which is placed the corresponding data from the neighbouring cells as they become available (Figure 4–3).

In the figure, each block of sixteen data elements is allocated to a particular processor's local memory. The shaded area around the central block is the data

required by that "central" processor from its neighbouring processors. In this case the space required by the central processor is $16 + 16$.

The shape and size of this halo will depend on the nature of the algorithm and upon the dimension of the data and of its decomposition. The overall memory requirement is the space required for the data plus the combined space for all the halos. In general, the halo of any particular cell is the combination of those parts of other cells which are required to update the cell in question, however a regular decomposition provides a typical example of such overlap.

Let $C = (c_1, c_2, \ldots, c_n)$ be the Cartesian coordinates of a data point in n-space and let $D = \{(c_1, c_2, \ldots, c_n) : \forall i = 1 \ldots n, 0 \leq c_i < k\}$ be a set of data representing some system to be modelled. The parameters n and k are positive integers and are called respectively the dimension and radix of D. Let $G = k^n$ be the number of elements in D. For two points $A, B \in D$, A is said to interact with B if the value of point A at time t is required as an input to the computation of point B at time $t + \Delta t$, where Δt is the size of the simulation time step. As described for the DAG, interactions of the following type are assumed: Point $A = (a_1, a_2, \ldots, a_n)$ interacts with point $B = (b_1, b_2, \ldots, b_n)$ if:

$$\exists j (1 \leq j \leq n) \text{ such that } \forall i (1 \leq i \leq n) \begin{cases} a_i = b_i & \text{if } i \neq j \\ 0 < |a_i - b_i| \leq \delta & \text{if } i = j \end{cases}$$

for positive integers δ, i and j:

δ is the interaction *depth.*

Let $P = (p_1, p_2, \ldots, p_n)$ represent a single processing element and let $E = \{(p_1, p_2, \ldots, p_n) : \forall i = 1 \ldots n, p_i < r\}$ represent an ensemble of PEs across which the n-dimensional D is to be distributed. $r = N^{1/n}$ is called the radix of the ensemble, and for simplicity assume that k/r is a positive integer. A partition of D across E is the set:

$$M = \{(P, \Delta) : P \in E, \Delta \subset D\}$$

such that

$$\forall i C \in \Delta \text{ if } \frac{p_i k}{r} \leq c_i < \frac{(p_i + 1)k}{r}$$

In effect, both the data and the processor ensemble can be considered as n-dimensional cubes with radices k and r respectively.

In addition to the $\frac{G}{N}$ points in Δ, each node also requires space in which to store the values of any points which interact with points in Δ but which are located on other nodes. A point $C \in \Delta$ will interact with non-local points (i.e. those on another node) if one or more of the coordinates of C satisfies the condition:

$$\frac{p_i k}{r} \leq c_i \leq \frac{p_i k}{r} + (\delta - 1)$$

or

$$\frac{(p_i + 1)k}{r} - \delta \leq c_i \leq \frac{(p_i + 1)k}{r} - 1$$

Under the interaction described earlier, the inner points – which incur overlap for $\delta > 1$ – require the same non-local data points as the outermost points and so can be ignored when calculating overlap space requirements. The points of interest then, are those which would interact with $\delta = 1$ in the above expression. For each coordinate c_i in point $C = (c_1, c_2, \ldots, c_n)$ define $O[c_i]$ as:

$$O[c_i] = \begin{cases} 1 & c_i = \frac{p_i k}{r} \text{ or } c_i = \frac{(p_i+1)k}{r} - 1 \\ 0 & \text{otherwise} \end{cases}$$

Each $O[c_i] = 1$ represents an overlap in the ith dimension and a requirement for δ extra points of storage. For example, in figure 4–4, face point $(1, 1, 0)$ overlaps in one dimension, edge point $(0, 2, 1)$ overlaps in two dimensions, and the corner point $(2, 2, 0)$ overlaps in all three dimensions.

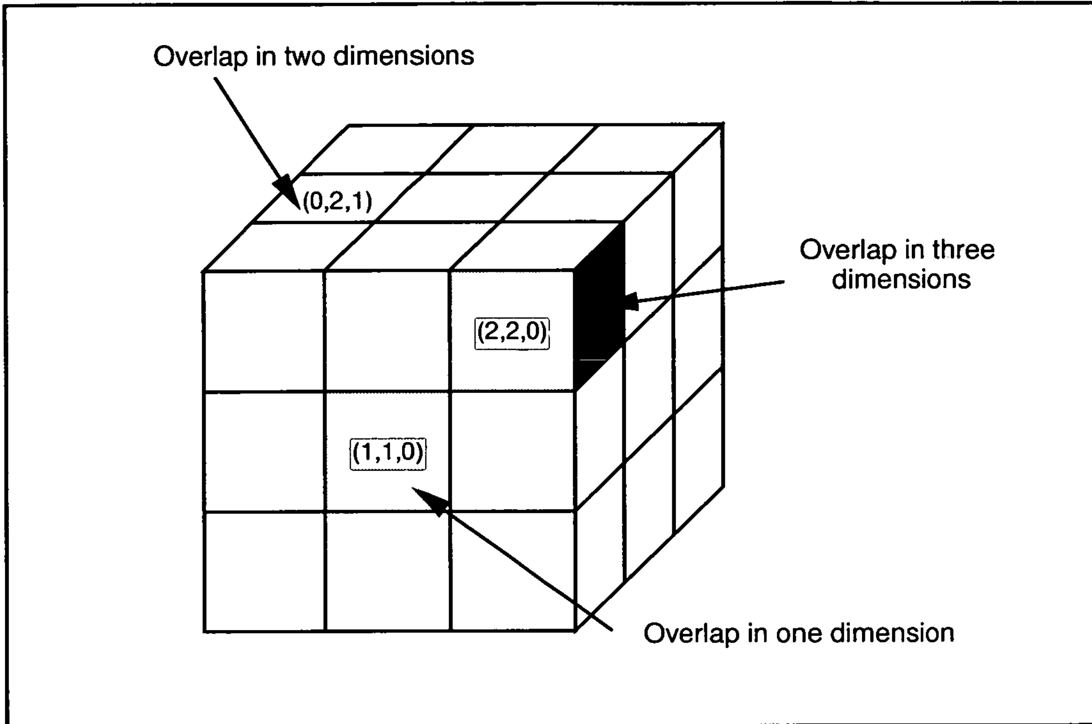


Figure 4–4: Overlap in several dimensions

The total overlap space incurred per point is thus:

$$h = \delta \sum_{i=1}^{n} O[c_i]$$

And the total overlap space for a processor is:

$$H = \sum_{C \in \Delta} \delta \sum_{i=1}^{n} O[c_i]$$

Or alternatively:

$$H = \delta \sum_{i=1}^{n} \sum_{C \in \Delta} O[c_i]$$

Since each possible value of each c_i occurs the same number of times, then for any given dimension i:

$$\sum_{C \in \Delta} O[c_i] = 2 \left(\frac{k}{r} \right)^{(n-1)}$$

Since Δ is homogeneous (being comprised of the complete set of n-tuples representing the data points on the processor in question), the overlap is the same for each of the n dimensions and so:

$$H = 2\delta n \left(\frac{k}{r}\right)^{(n-1)} \tag{4.3}$$

This is as expected if the interface between the processors is regarded as $2n$ $(n-1)$-dimensional "cubes" of side k/r.

For example, in two-dimensional data, with unit interaction depth, the interface between the data on one processor and those on its neighbours consists of four "lines" of data. In a three-dimensional data set, the interface consists of the eight "surfaces" of the processor's cube of local data. For a three-dimensional decomposition with $\delta = 1$, the overlap space requirement on a single processor is:

$$H = 6\left(\frac{k}{r}\right)^2$$

The overlap space as described is incurred on a by-processor basis, and so, ignoring edge-effects, the total space requirement for D is:

$$\begin{aligned} n_{M_{ALG}} &= G + NH \\ &= G + 2\delta nN\left(\frac{k}{r}\right)^{(n-1)} \\ &= G + 2\delta nG^{(\frac{n-1}{n})}N^{(\frac{1}{n})} \end{aligned}$$

From this total, an outside "surface" of the data space of $2n\delta k^{(n-1)}$ should be subtracted. The total space requirements, removing surface overlap, is:

$$n_{M_{ALG}} = G + 2n\delta G^{(\frac{n-1}{n})}(N^{\frac{1}{n}} - 1) \tag{4.4}$$

Equation 4.4 describes how the amount of memory will increase as N is increased.

For the two-dimensional decomposition in question, with $\delta = 1$, the space requirement on a base space of G is:

$$n_{M_{ALG}} = G + 4\sqrt{G}(\sqrt{N} - 1) \text{ data points} \quad (4.5)$$

Equation 4.5 can now be combined with the expression for basic component performance (equation 2.16) to give the access time, t_{M_l}, in seconds:

$$t_{M_l} = 8.8 + 650e^{-\frac{0.21R_M}{b(G+4\sqrt{G}(\sqrt{N}-1))}} \text{nanoseconds}$$

where b is the space per data point in megabytes.

Figures 4–5, 4–6 and 4–7 show how total capacity, device cost and access time must vary with N in a two-dimensional grid decomposition of a 1MByte data space with interaction depth of 2.

From figures 4–5 and 4–6 it can be seen that the increase in memory (and corresponding decrease in unit cost) is relatively modest. Increasing N from 1 to 1000 incurs an overhead of only 25% in total space requirements. However, this is only for the data space as discussed. Including program memory, which may scale linearly with N, and the other components of memory could produce more serious scaling.

Nonetheless, the effects of even this modest space scaling induced by the overlap areas are particularly obvious in figure 4–7. The access time increases by over 50% when increasing the number of nodes from 1 up to 1000. Notice, however, that the increase in access time is more pronounced in small numbers of nodes and lessens as one increases above a few hundred processing elements.

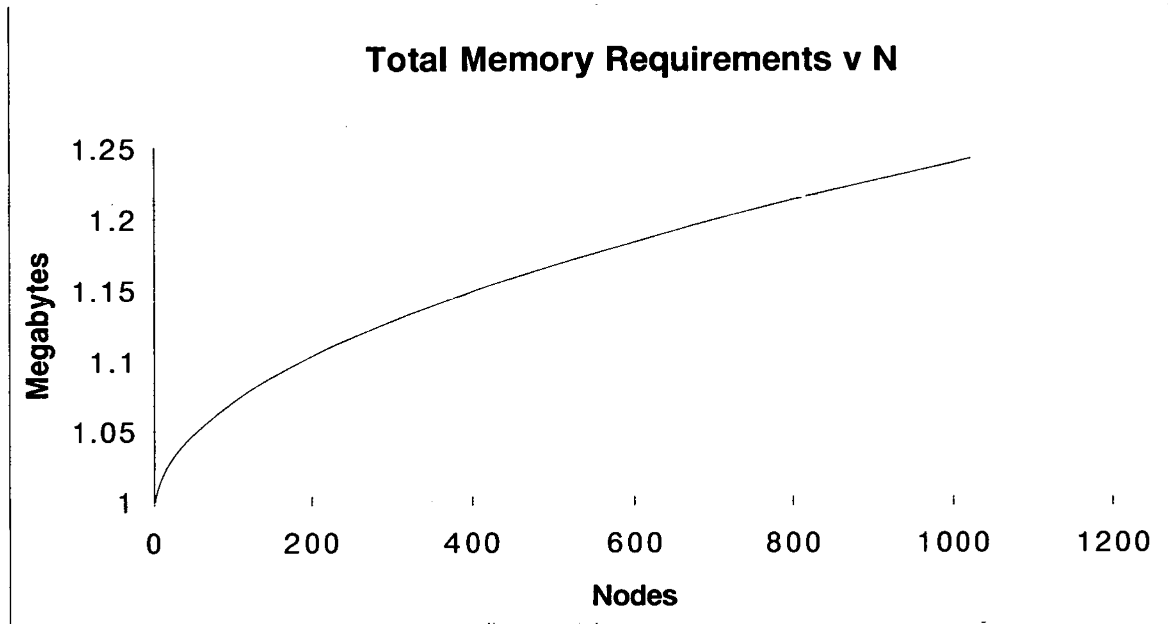


Figure 4–5: Effects of Overlap Areas on Total Capacity

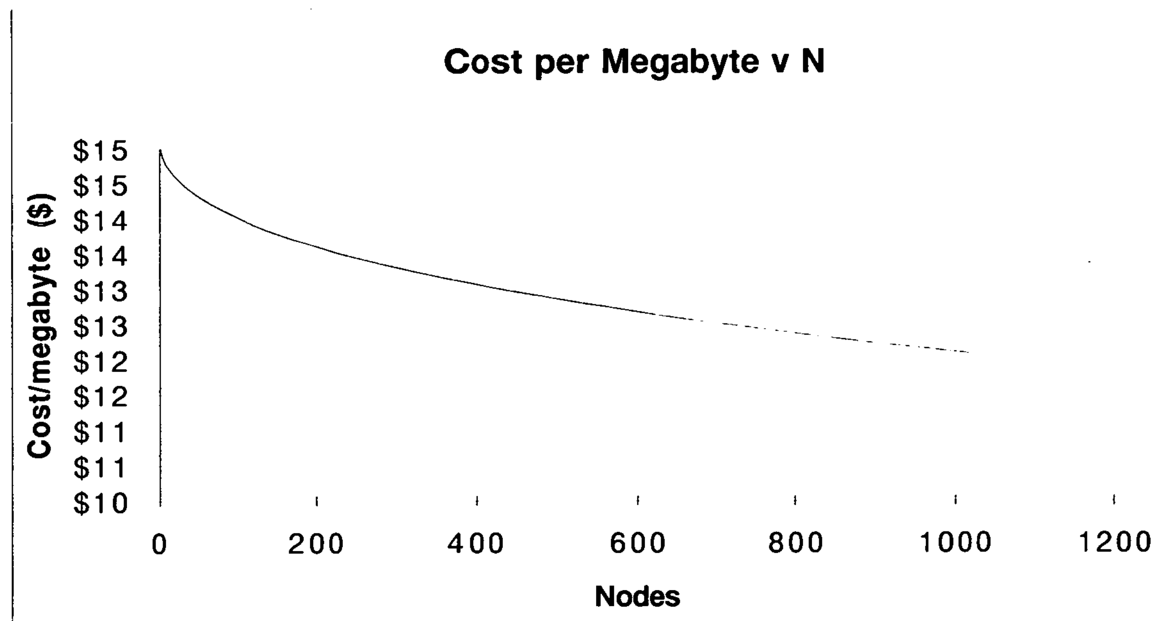


Figure 4–6: Effects of Overlap Areas on Memory Device Cost

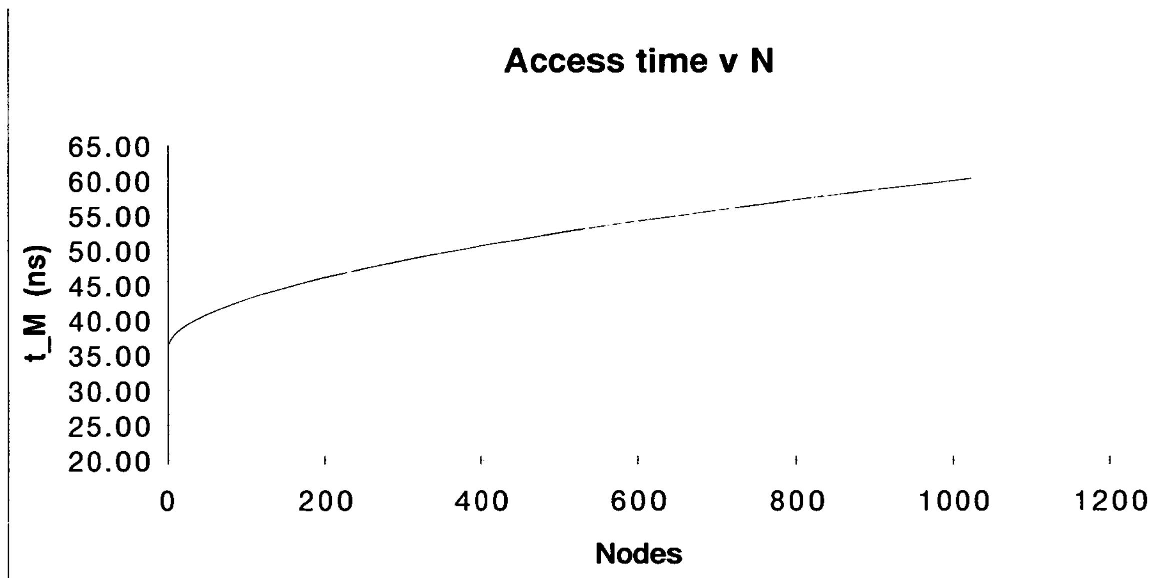


Figure 4–7: Effects of Overlap Areas on Access Time

4.5.2 Mesh Network Latency - t_ρ

The mesh is a member of the k-ary n-cube family of interconnection topologies (Figure 4–8) [13]. A k-ary n-cube is an n-dimensional cube, the edge of which is a ring (torus) or line (mesh) of k PEs giving a total number of $N = k^n$ PEs. The binary hypercube ($k = 2$) is a common example [65]. The meshes under consideration are those k-ary n-cubes with $n = 2$.

In k-ary n-cubes, associated with each PE is typically a single router component connecting the PE to the network in each of the n dimensions. Latency in these topologies is affected by several factors, and has been the subject of numerous studies [13] [38] [64]. In general, the latency depends on the number of routers

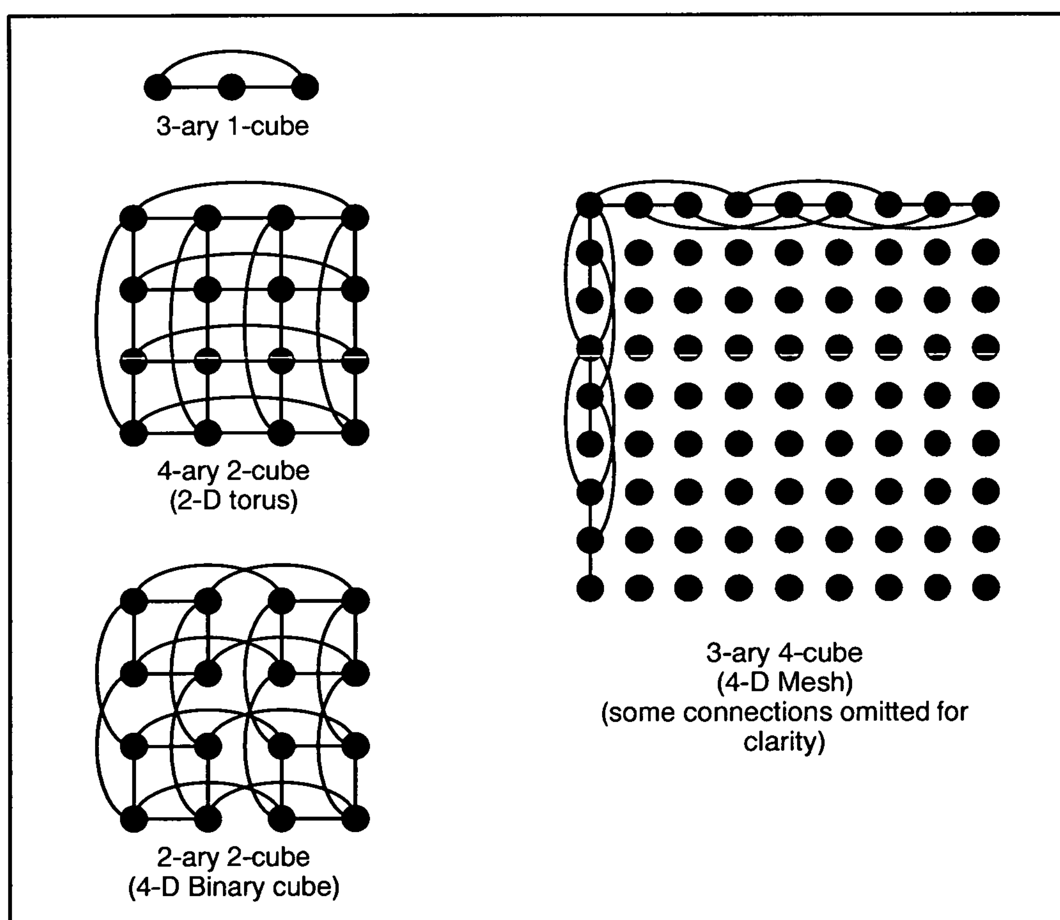


Figure 4–8: Examples of k-ary n-cubes

traversed, the performance of those components (under the relevant load), and the buffering strategy used for through-routing.

In a *store-and-forward* system, where the entire message is received by each intermediate router before being forwarded to the next, then, ignoring contention, the average latency, t_{SF}, is given by:

$$t_{SF} = t_R D \frac{B}{W_R}$$

where:

D	=	average number of hops between source and destination.
W_R	=	router channel width in bytes
t_R	=	router cycle time
B	=	length of message in bytes

Here it is assumed that the transit time across the links connecting neighbouring routers is negligible. Even if this transit time becomes significant compared with the router delay, it is not affected by N in a fixed-degree network such as the mesh. For variable-degree topologies, like the binary hypercube, the link length may vary with N, and could be important. For a network using virtual cut-through or wormhole routing, [39] [16], a message is routed directly from input to output on intermediate routers, provided the output is free. It is not necessary to wait for the entire message to arrive before forwarding. Each message is divided into a number of "flits" (FLow-control digITS) [14] and only the header-flit, containing the address information incurs the full penalty of traversing the intermediate links and routers [1]. After the path between source and destination has been set up, it remains open at any given point until the tail flit passes. The transfer of the message is therefore pipelined, with each flit of the body of the message appearing to

---

[1]Here, "flit" is being used synonomously with both "flit" and "phit" in [14]; thus flow-control is performed on portions of message equal in size to a single link width

experience only a single link and router delay. Assuming that the message header is a single W-byte flit, the latency, t_{VCT} for cut-through routing is:

$$\begin{aligned} t_{VCT} &= Dt_R + (\frac{B}{W_R} - 1)t_R \\ &= t_R(D + \frac{B}{W_R} - 1) \end{aligned} \tag{4.6}$$

This indicates the pipeline effect of these buffering strategies, in contrast with the less sophisticated store-and-forward method. For the remainder of this section the analysis will be restricted to virtual cut-through routing.

Average distance travelled - D

The average distance travelled by a message, D, depends on the communication locality. This is a function of the precedence relations of the graph, the network topology, and the routing strategy.

Locality can be described by $\xi(h)$, the probability that a child of a given parent task will be situated on a PE which is h links distant from the parent's PE. In other words, $\xi(h)$ is the probability that any given message will be sent to a destination PE situated h links ("hops") from the sending node. Examples of ξ can be found in [17]. In the following, for simplicity, the network is assumed to be symmetric.

Uniform Communication – In uniform communication the destination of a message will be chosen with equal probability from among all PEs. Assuming a symmetric network, ξ is given by:

$$\xi(h) = \frac{N_h}{N} \tag{4.7}$$

where N_h is the number of PEs which are h hops from the source processor.

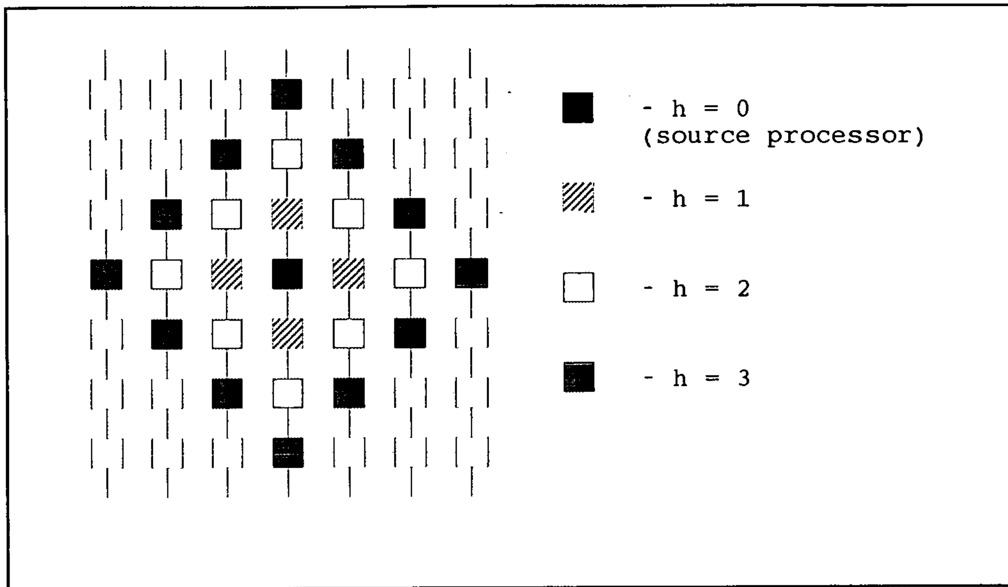


Figure 4–9: Decreasing Probability Communication

Decreasing Probability Communication – Here the probability that a message will be sent to a PE h hops away decreases as h increases. A simple example of this is in the nearest neighbour connected network shown in Figure 4–9:

$$\xi(h) = \frac{\gamma N_h}{(h+1)} \tag{4.8}$$

where $\gamma = \frac{1}{\sum_{h=0}^{h_{max}} \frac{1}{h+1}}$ is a normalizing constant used to ensure the total probability is unity and h_{max} is the distance in hops of the PE furthest from the source PE.

Sphere of Locality Communication – In this form of ξ each PE is considered to be at the centre of a group of PEs, each of which is no more than r hops away. In some architectures r can be considered the radius of a sphere, hence the name. A message from a PE will be sent to a randomly chosen PE within the sphere, with some probability, β, and to a randomly chosen

PE outside the sphere, with probability $1-\beta$. Typically $\beta > 0.5$. This form corresponds to Reed's Sphere of Locality in [61] [17].

For the grid algorithm on a mesh of dimension equal to that of the data, the locality depends on the interaction depth, and the number of PEs. For a data set of size $G = k^n$, a PE will transmit to its $2n$ nearest neighbours (1 hop away) if $\delta > 0$. It will *also* transmit to its next nearest neighbours if $\delta > \sqrt[n]{\frac{G}{N}}$. In the same way, messages will also be sent to the PEs lying 3 hops distant if $\delta > 2\sqrt[n]{\frac{G}{N}}$. In general, messages will be sent to all PEs lying h hops from the source if:

$$\delta > (h-1)\sqrt[n]{\frac{G}{N}}$$

The average distance travelled by a message, D, is therefore:

$$\begin{aligned} D &= \frac{\sum_{h=1}^{\left\lceil \delta \sqrt[n]{\frac{N}{G}} \right\rceil} h}{\left\lceil \delta \sqrt[n]{\frac{N}{G}} \right\rceil} \\ &= \frac{\left\lceil \delta \sqrt[n]{\frac{N}{G}} \right\rceil + 1}{2} \end{aligned}$$

For a 2-D mesh and data set, with $\delta = 1$, this reduces to:

$$D = \frac{1}{2}\left(\left\lceil \sqrt{\frac{N}{G}} \right\rceil + 1\right)$$

That is, $D = 1$ for $N > 1$ (N will not exceed G since that is the maximum parallelism of the algorithm).

This is a trivial case of Sphere of Locality with $r = \beta = 1$.

Equation 4.6 gives the latency for a single message traversing an unloaded network. To calculate t_ρ, the blocking due to the time the neighbouring nodes spend preparing messages must be considered. While the message sends are non-blocking, the PE will spend some time preparing the messages. Let i_s be the

number of instructions required to prepare and send a message, and so let $t_{PREP} = i_s(t_I + \mu t_{M_l})$ be the time to prepare a message for sending. The time for the head flit of the first of the four messages to cross the local router is:

$$t_{PREP} + t_R$$

The second message must wait until the whole of the first has gone, and so the total time until its head flit crosses the router is:

$$t_{PREP} + ht_R + t_{PREP} + t_R$$

where: $h = \frac{B}{W_R}$ is the number of flits in the message. The time for the third message is:

$$t_{PREP} + ht_R + t_{PREP} + ht_R + t_{PREP} + t_R$$

and the delay experienced by the header of the last message is:

$$t_{PREP} + ht_R + t_{PREP} + ht_R + t_{PREP} + ht_R + t_{PREP} + t_R \quad = \quad 4t_{PREP} + 3ht_R + t_R$$

Since a message will be in either of the four positions with equal likelihood, and since there are four incoming messages, it will be assumed that one of the four will indeed be the last from its source. The additional time for this last message to reach the destination node is simply ht_R, since the other three messages will aready have been received.

The total waiting time, t_ρ is therefore:

$$\begin{aligned} t_\rho &= 4t_{PREP} + 3ht_R + t_R + ht_R \\ &= 4\left(i_s(t_I + \mu t_{M_l}) + t_R\left(\frac{B}{W_R} + \frac{1}{4}\right)\right) \end{aligned} \quad (4.9)$$

Router Cycle Time - t_R

Router cycle time, t_R is as given by equation 2.17 in chapter 2:

$$t_R = 3.33 + 43e^{-\frac{0.1R_R}{3N}} \text{nanoseconds}$$

This treats the process of connecting an input and output pair, and driving the data across the router as of the same order as an instruction execution on a processor. The coefficient α in equation 2.17 can be varied to reflect a router cycle significantly shorter than the instruction cycle. R_R in the above expression is the total cost of all routers.

Router Channel Width - W_R

Router channel width, W_R, was also developed in chapter 2, and is given by equation 2.19. This will be expressed as 8-bit bytes and for a 10-channel router (bidirectional in each dimension, plus a bidirectional port to the local node) is given by:

$$W_R = \frac{1}{80}\left(\frac{R_R}{2.7 * 10^{-3}N}\right)^{\frac{1}{1.7}}$$

Message Length - B

The last parameter required is B, the number of bytes per message. This depends on the number of points on the sending PE which interact with points on the destination PE. Ignoring edge effects, the total number of points to be sent to any PE is given by the halo space in equation 4.3. This is split among the $2n$ "faces" of the n-dimensional cube of points allocated to the PE. The message length, assuming one byte per point, is thus:

$$B = \delta\left(\frac{k}{r}\right)^{(n-1)} + \text{header} + \text{tail}$$

The header and tail portions vary according to the implementation. The header will usually contain the destination address and perhaps a message length datum. The tail may be a check datum. For the purposes of this analysis, the header is assumed to be three bytes, and the tail is unused, and so B for a 2-D mesh with $\delta = 1$ is:

$$B = \left(\frac{k}{r} + 3\right) \tag{4.10}$$

4.6 Summary

Expressions have been derived for the various components of equation 4.2 as functions of N, the number of processors in a shared memory multiprocessor. These functions depend on the technology cost:performance functions developed in chapter 2, and also upon the algorithm and machine architecture as described in this chapter.

The effects of the various components of the execution time will be investigated in chapter 5, and simulation results of a 2-D mesh under these cost:performance constraints will also be presented.

Chapter 5

Discussion

5.1 Introduction

This chapter takes the models developed in chapters 3 and 4, and discusses the significance of the various parameters involved. First, in section 5.2, simulation results are used to validate the models for both architectures. The simulations were of real hardware systems, containing processors, memory, arbiters, routing elements etc. The cost and performance figures for the simulations were also taken from real devices. This contrasts with the performance models themselves which used the various derived functions of cost and performance as described in the previous chapters. Since the range of actual cost and performance data consists of discrete values, the process of validation had to depart from the overall principle of cost fixing. For each simulation experiment, the cost of *individual* components was fixed and their number varied. In other words overall system cost was allowed to increase as N was varied. Each experiment compared results calculated using the models with actual results for component costs at the low, mid and high-points of the available data. This method allowed an extensive test of the accuracy of the models' predictions, and produced confidence in these. Once this confidence was achieved, it was then possible to proceed to investigate the cost fixing effects with the models alone. This discussion is presented in section 5.4

5.2 The Simulator

This section discusses the simulations used to validate the models described in chapters 3 and 4. First, the simulator itself is described. Only the mesh simulator, being the more complex of the two used, is described here. Then the experiments and corresponding results used to validate both the bus and mesh models are presented.

5.2.1 Verilog Simulator

Various simulation tools were considered, many providing a front-end to a common high-level language [7] [8] [31]. The chosen tool was Verilog [71] and the Cadence Verilog-XL [29] simulator. Verilog is one of the two most popular languages (the other being VHDL [5]) widely used for hardware description. It provides a useful mapping between HDL and hardware and this can be exploited using synthesis tools.

The simulator consisted of two basic types of entity, processors (with integral memory) and routers. The processors executed instructions, initiated local memory references and then prepared a message which was passed to the router. Each processor and router were combined into a single node component and these were connected in a two-dimensional bi-directional mesh.

The router consisted of five pairs (north, south, east, west and local) of buffers; each pair consisting of one input and one output. Connections among these were achieved using a 5x5 crossbar switch. This allowed non-blocking connections between pairs of inputs and outputs. Figure 5–1 shows a block diagram of the router component.

Routing was bidirectional with wormhole buffering and so a simple X-Y restricted routing scheme sufficed to avoid deadlock. The router buffers had attached fifo queues, but these were not used in the simulations described. Messages were sent as a sequence of phits, the number of which was fixed for any given simulation run, but in general depended on the algorithm (e.g. the size of the halo in the overlap areas computation), and the router channel width. The first three phits were the coordinates of the destination, node, the coordinates of the source, and a unique ID for that message. Only the first of these three was required for routing purposes; the others were used for debugging.

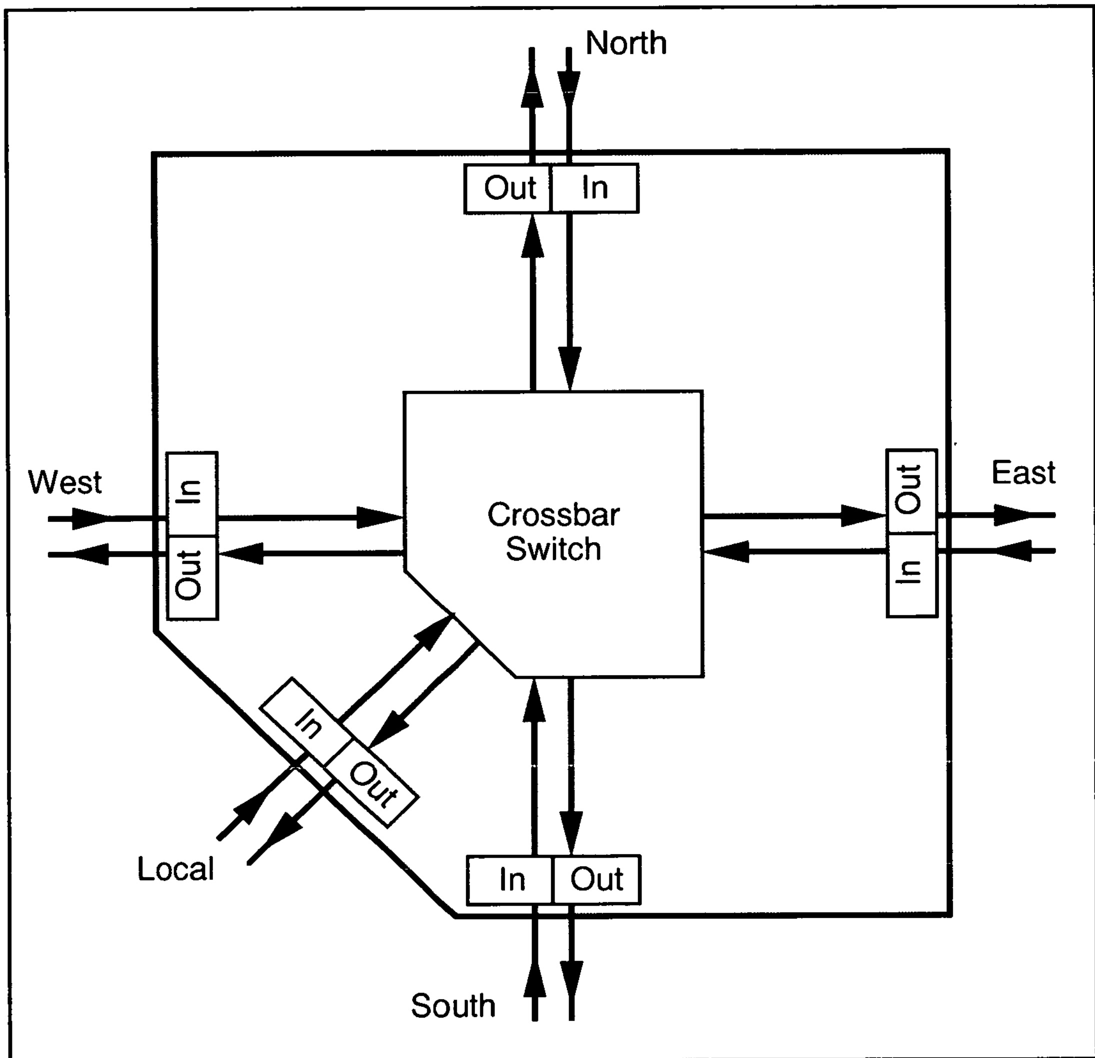


Figure 5–1: Verilog Router Block Diagram

5.2.2 Simulation Control

One drawback of the Verilog language was its lack of support for run-time instantiation of simulation entities. Lacking an equivalent of VHDL's "for ... generate" construct, in Verilog all router and processor modules had to be explicitly instantiated. Since varying the number of such components was a key aspect of the simulation, a C front-end was built to generate the Verilog fixtures for the various values of N being examined.

This C controller used look-up tables to specify timings for the various operations performed by the simulator, and controlled the collection of data after the run. The controller would oversee experiments described in a parameter file; the file could describe either a set of experiments explicitly, or a range of parameters from which a full-factorial experiment would be performed. Figure 5–2 provides an overview of the simulation control.

5.2.3 Performance

Simulator performance proved to be a serious limiting factor. While simulations of several thousand nodes on relatively modest hardware are regularly reported [56], the Verilog-XL simulator was extremely memory-hungry. Simulations of under 1000 nodes exceeded the several hundred megabyte swap limits on a large server and resulted in disk-thrashing. Towards the end of the project, progress was made using Chronologic's VCS [63]. This compiled simulator (Cadence's interprets the source) gave up to 10 times savings on memory and commensurate improvements in simulation time.

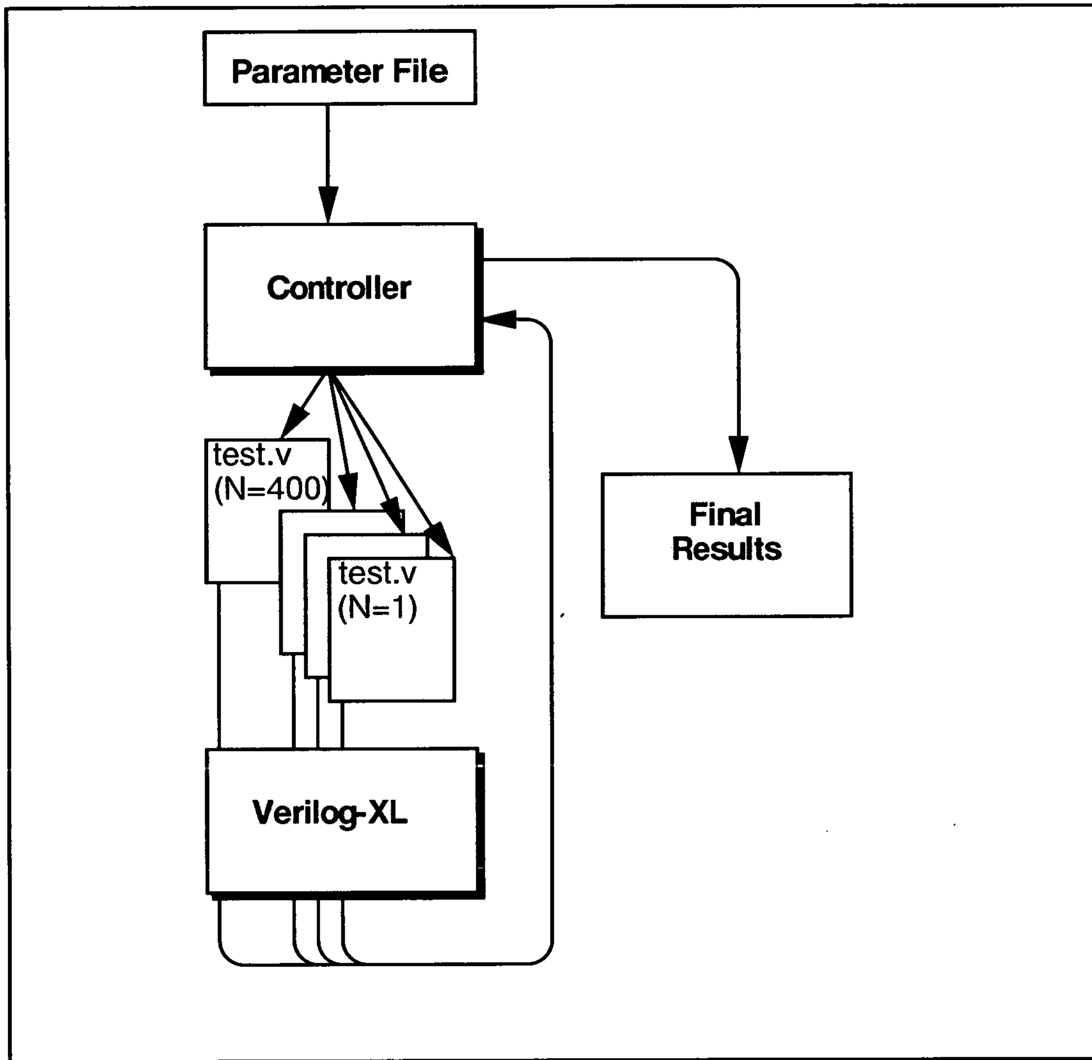


Figure 5–2: Simulation Control

5.3 Simulation Results

This section describes the various simulation experiments performed to check the accuracy of the model predictions. The shared memory system is described first, and then the message-passing multicomputer. Full results tables and graphs are given in appendix B.

5.3.1 Shared Memory Multiprocessor

As described in chapter 3, the shared memory system consists of N processors, each with a private store, connected to a common shared memory by a single bus. The shared memory contains P data points, each of which must be operated on using a set of i instructions, and μi private memory references. After processing a point, a processor writes the result back to the shared store and, in the same bus tenure, reads the next unprocessed point. Each point consists of a single word of memory.

To validate the model, a range of simulation experiments was performed. The simulator modelled real hardware constructs and used actual cost and performance figures for the processors and memory devices. The simulation experiments focused on the algorithm parameters i and μ. P was made sufficiently large to hide the initial bus loading transients which are not represented in equation 3.6 and so a single simulation cycle was sufficient for each value of N. In the first run, $\mu = 0.1$ and $i = 10$. Systems were simulated for $N = 1$ to 100. While 100 nodes is very modest in the wider context of parallel systems, as will be seen from the results, this is usually high enough to cause the bus to saturate. The base memory sizes were 5 Megabytes for the shared store, and 0.5 Megabytes per processor of local store. This was then repeated for $i = 100$ and $i = 500$. These three runs were then repeated again twice; first with $\mu = 0.5$ and then $\mu = 1$. In each case,

the cost of a single processor and single memory device were fixed. Since N was varied for each run, the total system cost would increase as N was increased. This was done because the range of available processor cost:performance figures was limited and consisted of discrete values.

This set of experiments was then repeated twice, once for components in the middle of the cost range examined, and once for the most expensive components.

Figures B–1 to B–9 in Appendix B plot execution time against N for each experiment. Simulation results matched the model to within an average of under 3.5%.

5.3.2 Message-passing Multicomputer

The multicomputer described in chapter 4 consists of N processing nodes arranged in a two-dimensional rectangular mesh. Each node contains a processor, a private memory store and a router through which the processor communicates with others.

The algorithm is the transformation, over s iterations, of an n-dimensional data space of P points. In each iteration each point requires the execution of i instructions, and μi local memory accesses. Each iteration ends with the transmission of the relevant points to neighbouring nodes, as described in chapter 4.

The simulations used the following fixed parameters throughout. The data space consisted of $P = 4096$ points arranged in two dimensions. The number of iterations was five. Router cost was fixed at $350 each for all simulations. As will become apparent, this was necessary to ensure that one of the significant aspects of router performance – I/O bandwidth – did not dominate all other effects. As with the bus simulations, three sets were run corresponding to low-cost, mid-range and high-end memory and processors.

Each experiment consisted of three sets of simulation runs, each for a square mesh from 1 to 400 nodes. Over the three sets one of the key algorithm parameters was varied, while the others were fixed.

Experiment 1a simulated instruction execution only, varying i_c. Experiment 1b again varied i_c but did this within the context of communication and memory operations. Experiments 2a and 2b simulated varying i_s, with first a low then high value for i_c. Experiments 3a to 3c varied μ, first with two values of i_c and no communications, and then with messages. Finally experiment 4 varied the number of bytes per grid point.

Figures B–10 to B–33 plot execution time against N for each experiment. Simulation results matched the model to within an average of 11%.

5.4 Model Behaviour

Having gained confidence in the accuracy of the various performance models, it is now possible to investigate,using the models alone, the effects of cost fixing. This section considers the effect on execution time, and on optimal N, of varying some of the key model parameters. First, the shared-bus model is discussed, and then the message-passing model.

5.4.1 Shared Memory Multiprocessor Model

A significant factor in this system is the loading on the shared-bus. As expected, in a fixed cost system this easily becomes a bottleneck as the number of processors is increased. The increase in loading due to the increase in N is exacerbated by the fact that the service time of the shared memory is also increasing. On the other hand, this is partly compensated for by the increased time the processors spend in

Parameter	Description	Value
P	Number of points in data set	10000
i	Number of instructions per point	100
μ	Fraction of instructions requiring local memory access	0.1
m	Local memory per processor	0.5 MBytes
n_{M_s}	Total shared memory	5 MBytes
R_P	Total cost of processors	$500
R_M	Total cost of all memory	$250

Table 5–1: Default Parameters for Bus Model

between bus requests, due to their increased instruction execution time and ever-slowing local memory. Note that for simplicity the model (nor the simulations) does not include the effects on performance of the increasing electrical length of the bus as N grows. Capacitive loading and transmission-line effects such as wire-OR glitching can have very serious performance implications and would tend to make the optimal N even lower than described below.

The sensitivity of the model to any given parameter may well depend on the precise domain over which it is being varied, and the domains of the other parameters. For example, the effect of varying the degree to which memory scales (linear in the model presented so far, but not necessarily so in general) will be more obvious in a system with low i and/or high μ. Any parameter values differing from those used in the simulations are mentioned where appropriate. The default values are given in table 5–1.

While the simulation experiments allowed the overall system cost to increase, and so kept the individual component costs constant throughout a run, in the following discussion the overall cost of each of the three hardware functions is kept constant. As a result, component cost varies inversely with N.

Using equation 2.15 (page 44), figures 5–3 and 5–4 show how processor perfor-

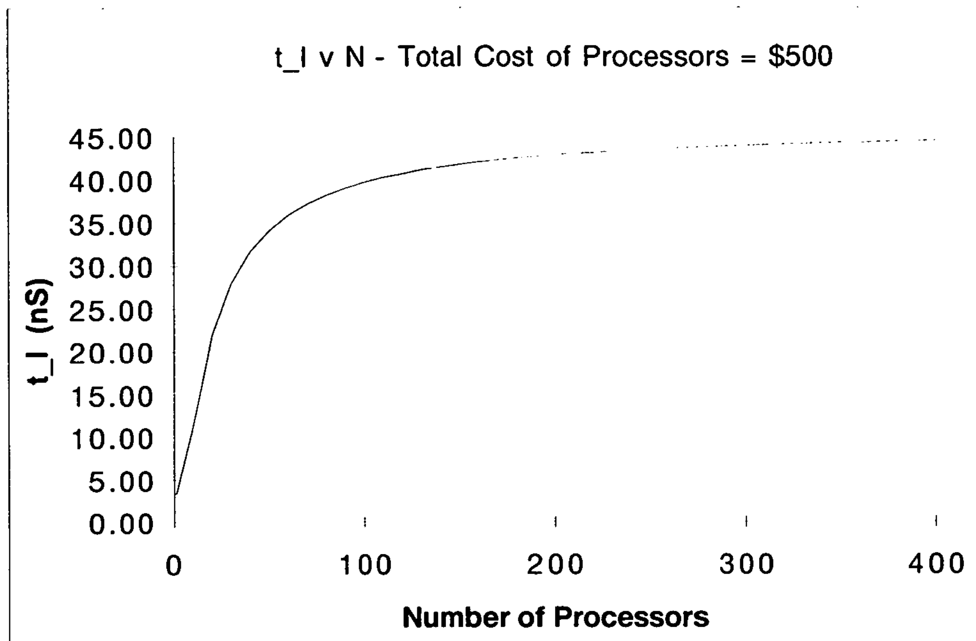


Figure 5–3: Single Processor Instruction Execution Time (t_I)

mance varies with N in a system with a total processor cost of $500. The latter graph measures processor speed as described in section 2.2.3. The graphs show that, in effect, above about 50 nodes (for this particular total cost), N can be increased with little impact on the processor performance. This is because, below a cost of around $10, processor speed is not significantly dependent on cost. In the mid 1990s, at this low-end of VLSI technology, devices typically are distinguished not by raw speed but by the extent to which they integrate, on a single chip, functions previously made available on separate devices. The effect of this is to render the aggregate performance of the multiprocessor an almost linear function of N above the "knee" in the graphs. For example, for a total system cost of $500, the aggregate performance is 300 million "Int92 instructions" per second for a single node, around 1250 for 50 nodes, and approximately 8000 for 400 nodes. In other words, if other factors suggest that the optimal processor cost is $10 at most (i.e. that optimal N is at least 50 in this case), then there is a strong case for using as many of the cheapest available nodes as possible. However, the effect of memory cost:performance must also be considered.

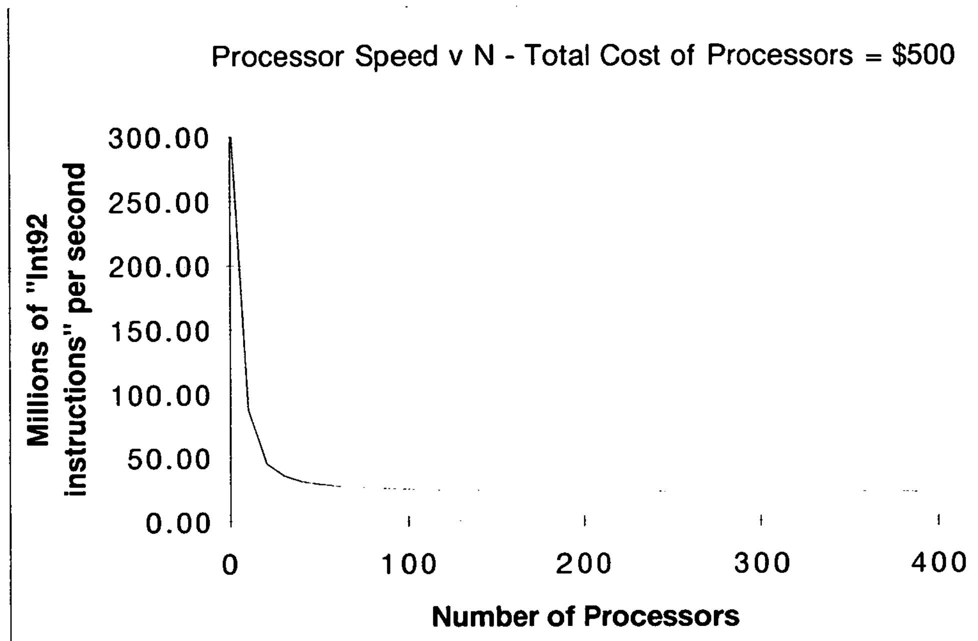


Figure 5–4: Single Processor Performance

For a total memory system cost of \$500, figures 5–5 and 5–6 (using equation 2.16 on page 47) show how the memory performance in the shared bus architecture drops as the capacity is increased. As with the processors, the nature of the memory cost:performance function means that the impact of increasing the amount of memory lessens as the total memory increases. However, this "flattening" is not as pronounced as with the processor devices, and in systems with significant space scaling the implications on memory performance of increasing N may continue to be serious for very high values of N. The effect, on the value of the optimal N, of varying the degree of space scaling is discussed later.

The next four sections consider the effects on optimal N, of varying the parameters describing the algorithm and the hardware.

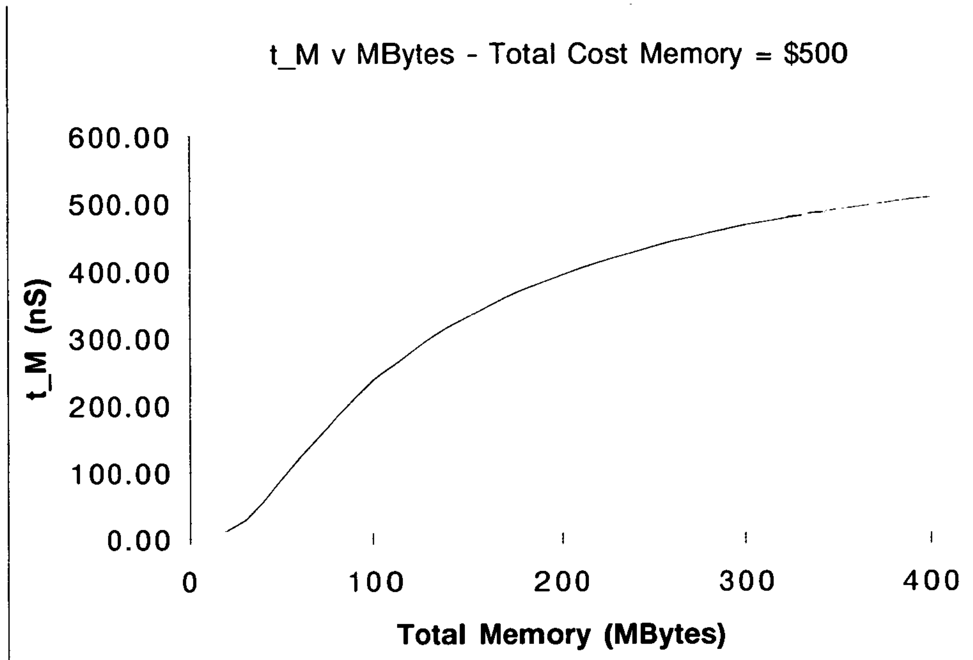


Figure 5–5: Memory Access Time versus Total Capacity

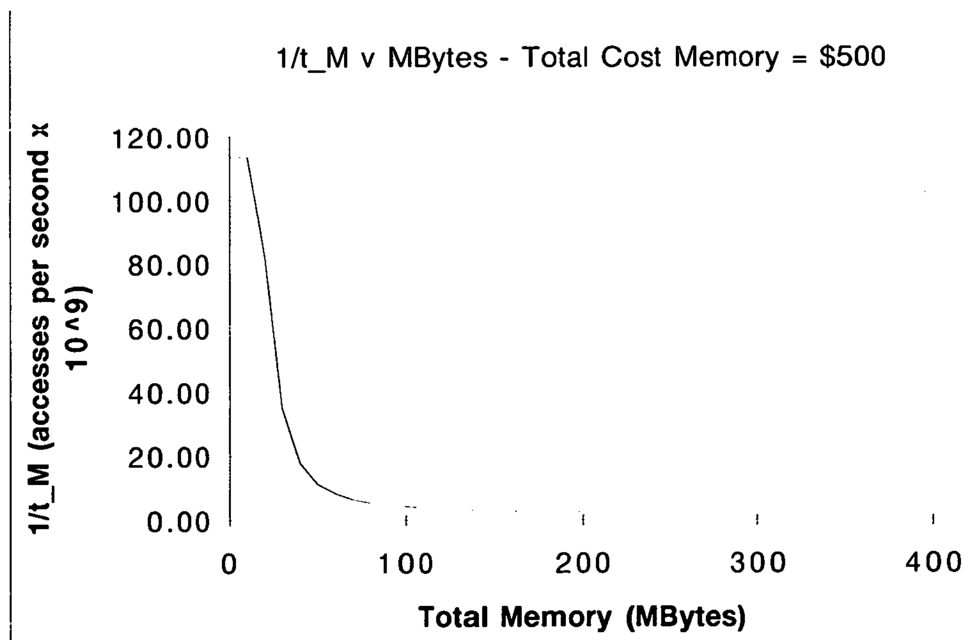


Figure 5–6: Memory Speed versus Total Capacity

Number of Instructions – i_c

The effect of increasing the number of instructions to be performed on a point depends principally on the nature of the hardware cost:performance functions. For sub-linear functions, increasing i tends to increase optimal N. This is due to two effects. First, the critical path length, being inversely linear with N, decreases faster than the time to execute instructions increases. Second, the computation to communication ratio increases, lessening the load on the shared bus.

Figure 5–7 shows how the optimal number of nodes increases as i_c is increased. For $i_c = 10$, consituting very fine grain parallelism, the optimal number of nodes is only 4. As i_c increases, the loading on the bus decreases for a given number of nodes, and the optimal N increases: optimal N is 30 for $i_c = 100$ and 70 for $i_c = 500$. Expressed in terms of processor costs, the optimal system for fine grain parallelism (i.e. small i_c) would be constructed from a few expensive nodes of cost:performance around that of the 21164 (see table 2–1 on page 40). Increasing i_c makes slower, less expensive nodes more attractive and $i_c = 500$ is best dealt with using a system of nodes with cost:performance between that of the ARM710 and the MPC601.

Notice that the convergence of the three curves shown is as predicted by the model given in chapter 3, and coincides with the points at which the buses in the three systems shown begin to load. Once a processor can expect to have to wait for another to be serviced before receiving tenure, any additional processors increase this loading. The slight rise in execution time at just under 10 nodes (i.e. for devices between around \$50 and \$40) is due to the superlinear nature of processor cost performance at the low end of the range investigated. This can be seen clearly on figure 2–5 on page 43.

Figure 5–8 shows how the bus "queue lengths" increase as N is increased. Recall that the queuing model used was of a closed system and that a processor would only issue a single request before stalling to wait for service. This dependence of

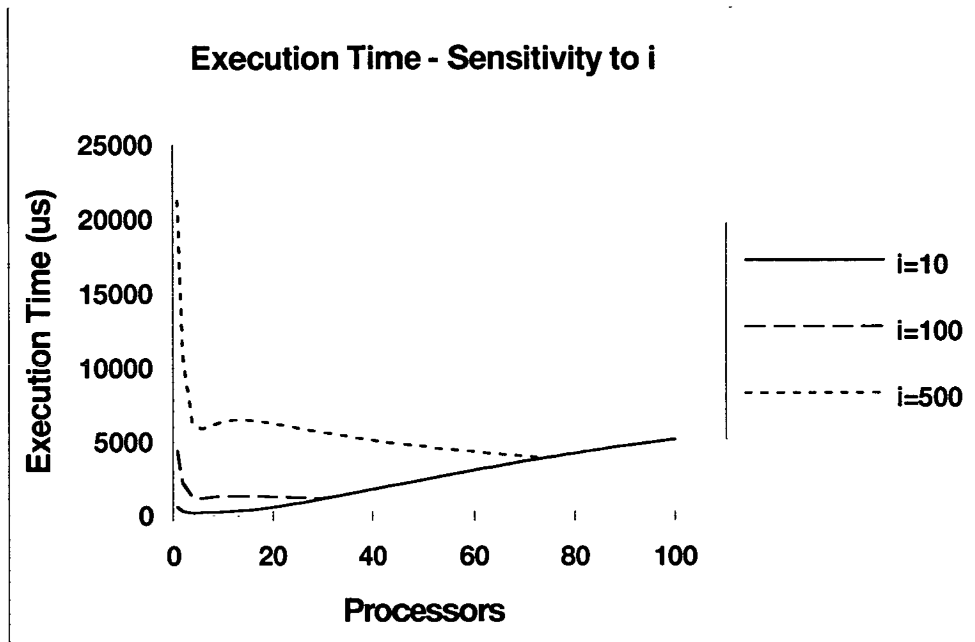


Figure 5–7: Bus Model - Execution time sensitivity to i

average arrival rate upon the queue length itself provides a "braking" effect on the loading of the queue, and the queue length grows only linearly wih N once loading begins.

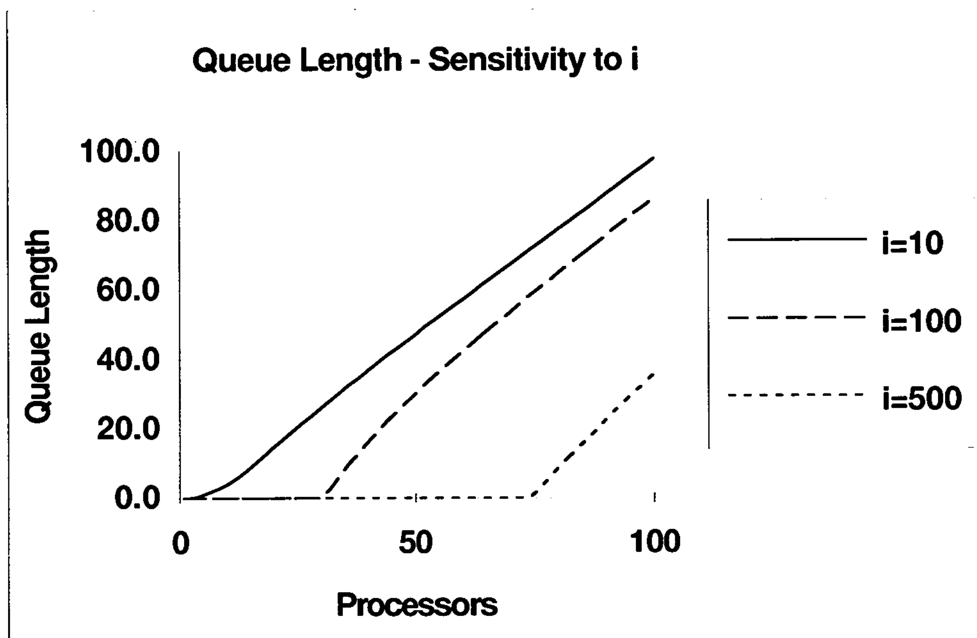


Figure 5–8: Bus Model - Bus waiting sensitivity to i

Local Memory Activity – μ

Increasing the amount of local memory references will, as with processors, tend to increase the optimal N, provided the memory cost:performance is sub-linear (i.e. if increasing memory device cost incurs increasingly diminishing returns in terms of improved access time). Figure 5–9 was produced using shared bus models, with a total memory cost, R_M of \$100. This shows how optimal N increases with μ. Again, as with the processors themselves, this is due to the reduced bus loading seen when processors spend more time in their local stores. Of course, this effect is lessened slightly by the fact that the increase in memory size will impact the performance of the shared store, effectively increasing the queue service time and lengthening the queue. Optimal N is 6 when one local access is performed for every 10 instructions, but this increases to 56 when every instruction requires a local access. In practice, μ may be greater than unity, if every instruction has to be fetched from the local store.

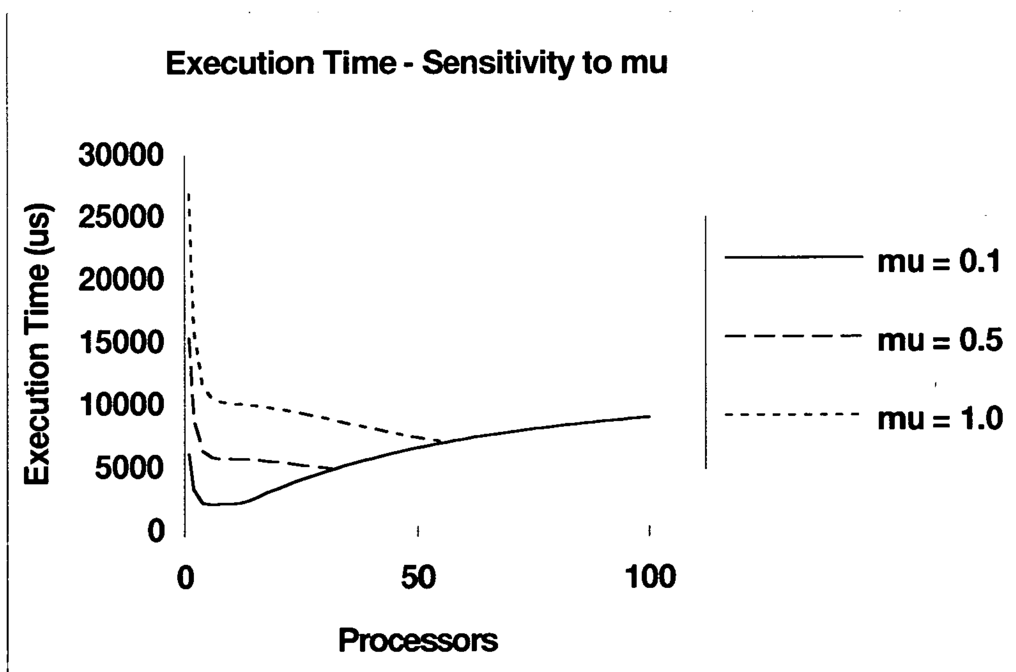


Figure 5–9: Bus Model - Execution time sensitivity to μ

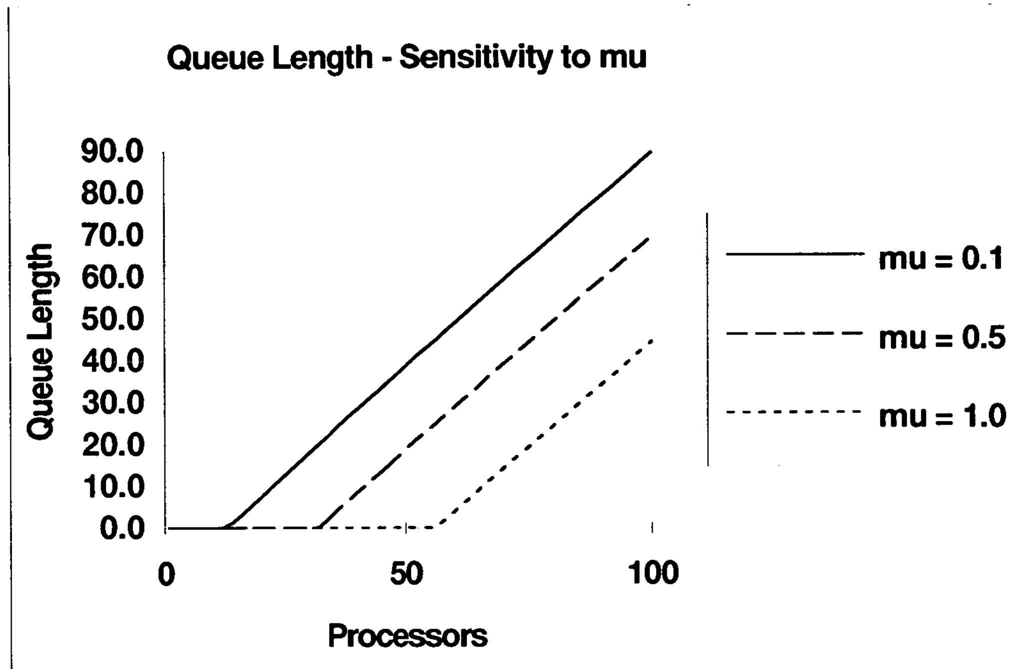


Figure 5–10: Bus Model - Bus waiting sensitivity to μ

Figure 5–10 shows how queue length dependency on N is affected by μ, the sharp increases in length corresponding to the second "knee" in the relevant curves in figure 5–9.

Space Scaling

In the system scenario discussed in chapter 3, and in the simulations, the total memory scaled almost linearly with N. In section 3.5.1, the total memory requirements were given as:

$$n_{M_{ALG}} = mN + n_{M_s}$$

where m is the local space per processor, and n_{M_s} is the shared space.

Figure 5–11 shows total memory requirements of the form:

$$n_{M_{ALG}} = mN^{exp} + n_{M_s} \tag{5.1}$$

for $exp = 0.1$, 1.0 and 1.2.

The presence of the exponent, especially when greater than unity, could represent the growth of local memory requirements due to cache management or virtual memory translation tables, etc.

Intuitively, one would expect systems with super-linear space-scaling to have lower optimal N than those with exponents of less than unity. This is seen in figure 5–12, produced from the model after using equation 5.1 as the denominator of the exponent in equation 3.4 (see page 62). The lack of convergence on the three curves, in contrast with the two parameters examined so far, is due to the fact that, as shown in chapter 3, after bus loading begins, T_{SM} depends mainly on device access time.

The slight "knees" in the curves in figure 5–12 correspond with the onset of bus loading, and coincide with the points shown in figure 5–13.

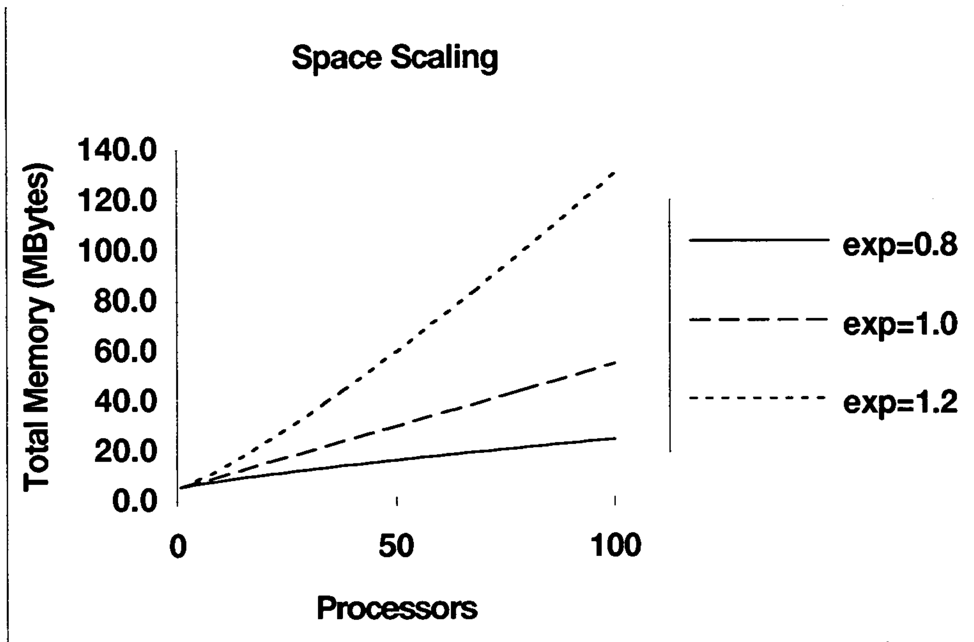


Figure 5–11: Bus Model - Memory requirements for various space scaling exponents

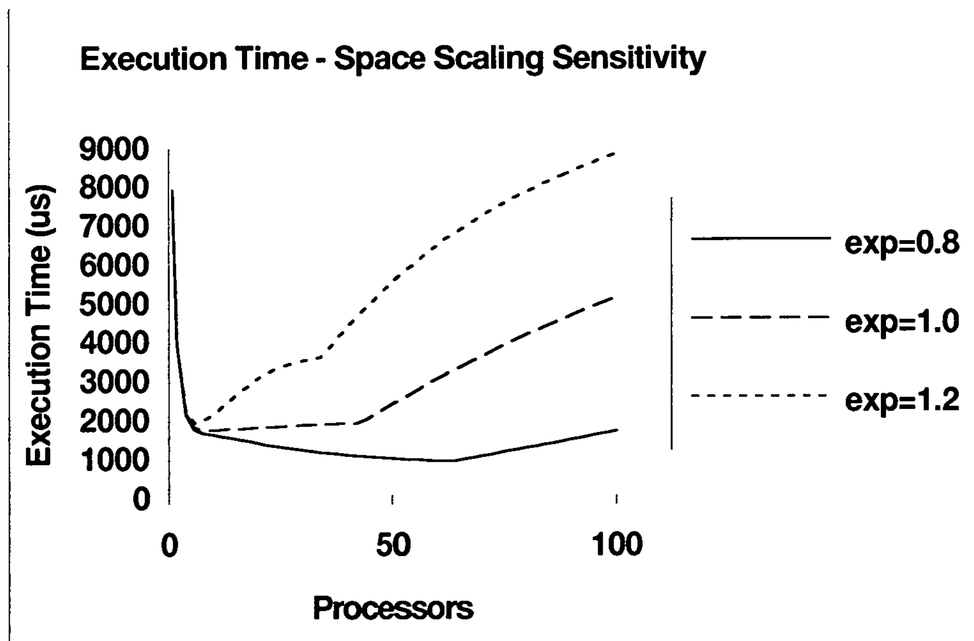


Figure 5–12: Bus Model - Execution time sensitivity to space scaling

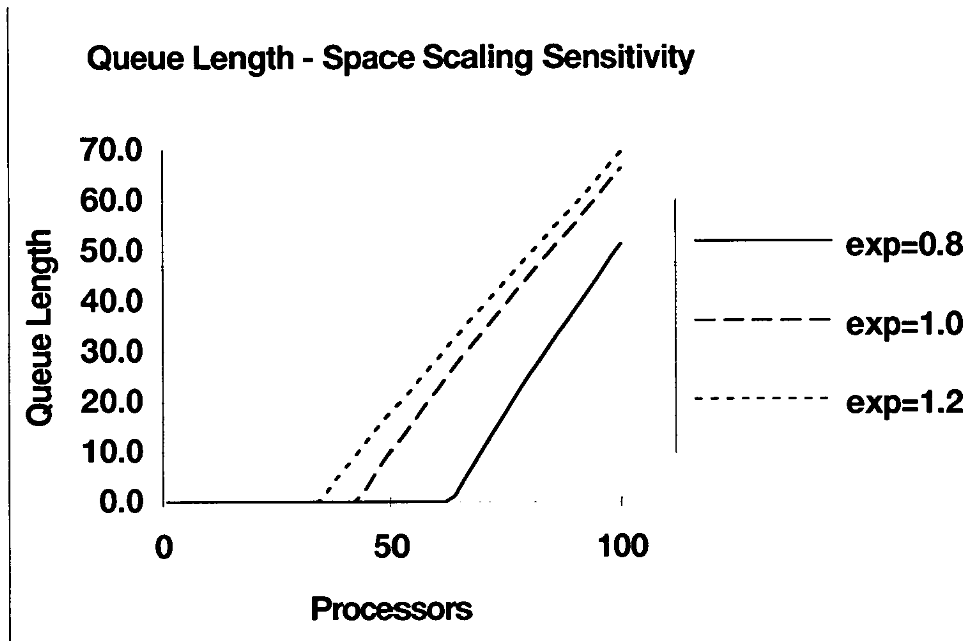


Figure 5–13: Bus Model - Bus waiting sensitivity to space scaling

Hardware Cost:Performance

The execution time of an instruction on a processor of cost $C_P = R_P/N$ has been shown (see equation 2.15) to be approximated by:

$$t_I = 3.33 + 43e^{\frac{-0.1}{3}C_P}\text{nanoseconds}$$

This final section concerning the shared memory architecture considers the effect on optimal N of varying the processor cost:performance function. By varying the coefficient of C_P in the above expression, alternative t_I may be investigated, along with the implications for T_{SM}. Varying the coefficient could correspond to an exploration of the processor cost:performance function at costs above that of the most expensive single chip MOS devices. This would allow an investigation of bipolar processors, and of board-level processors (see Chapter 6). It could also correspond to an examination of future processor cost:performance functions, where various physical limits on device construction become significant. Alternatively, various issues other than just cost and performance may restrict a designer's choice

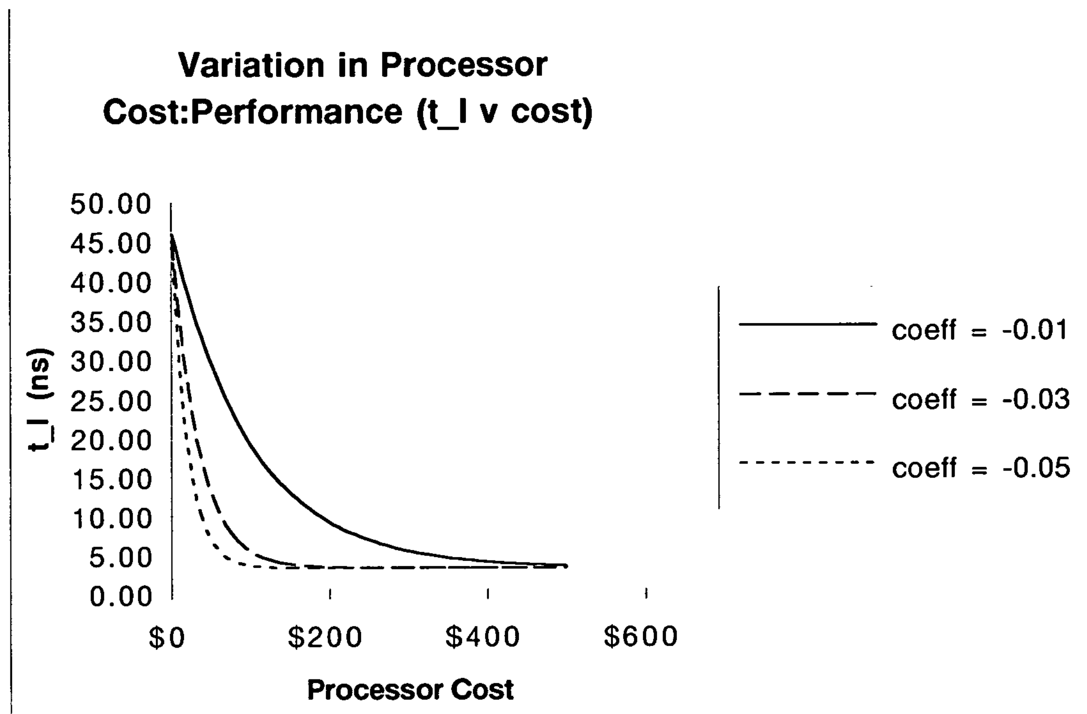


Figure 5–14: Instruction execution times for various processor cost:performance coefficients

of processor. For example, some computer manufacturers tend to focus on mature device families, letting competitors take the risk with newer devices. Therefore, the range of processors available over which to investigate the "number" versus "speed" tradeoff may be restricted to low-end devices. In such a case, a more accurate curve to fit the restricted cst and performance data is likely to show more super-linearity, and a higher coefficient may be more suitable.

Figure 5–14 shows t_I as a function of cost for coefficients of -0.01, -0.03 and -0.05 (corresponding to values near and around the current actual value). Figure 5–15 shows the reciprocal plots representing processor performance.

Figure 5–15 in particular highlights the degree to which the variation in processor performance within a particular cost range depends on the coefficient. With a coefficient of -0.05, the curve rises sharply and is almost constant above $150. The curve for a coefficient value of -0.01, on the other hand, is super-linear for much of

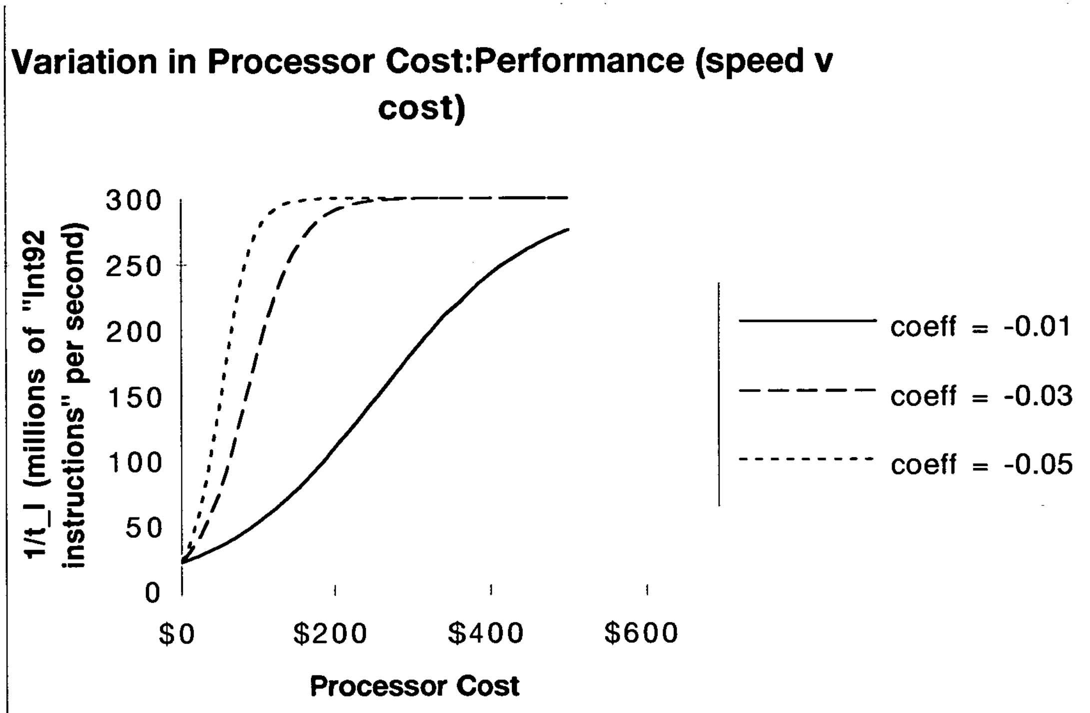


Figure 5–15: Processor performance for various coefficients

its extent, and only begins to show sub-linearity above $350. From this, one would expect machines using a processor technology following the -0.01 curve to have a higher optimal N, all else being equal, compared with a technology following the -0.05 curve. In the super-linear portion of the cost:performance curve, it is better to use a single node of cost X than N nodes each of cost X/N.

Figure 5–16 plots execution time for the three coefficients and confirms that the -0.01 technology has higher optimal N (34 processors) than either the -0.03 technology (30 processors) or the -0.05 technology (8 processors).

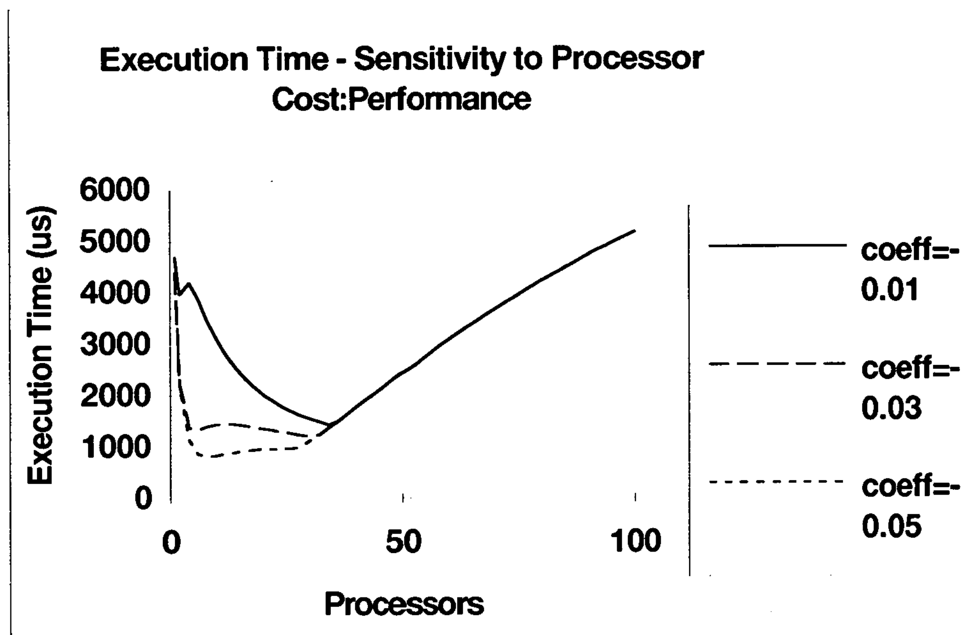


Figure 5–16: Bus Model - Execution time sensitivity to processor cost:performance

Parameter	Description	Value
P	Number of points in data space	4096
s	Iterations	5
i_c	Number of computation instructions per point	100
i_s	Number of instructions per message send	1
μ	Fraction of instructions requiring local memory access	0.1
b	Memory per grid point	1 byte
R_P	Total cost of processors	$500
R_M	Total cost of all memory	$0.36
R_R	Total cost of all routers	$5000

Table 5–2: Default Parameters for Mesh Model

5.4.2 Message Passing Multicomputer Model

The default parameters used in this section are given in table 5–2. A note of explanation is required for the total memory cost. The unusually low figure is due simply to the fact that the algorithm as described uses only 4 KBytes of memory. The cost given corresponds to just over $90 per megabyte.

In contrast with the bus architecture, in which the communications latency could quickly become a bottleneck, the localized communications pattern in the mesh renders that architecture more scalable for this algorithm. An algorithm with non-localized communications would suffer more seriously from message latency as N increased, and optimal N would be lower (see Chapter 6). An investigation of the sensitivity of N to the various parameters in the model shows that only in unrealistically extreme situations, (e.g. where the number of instructions required to send a message is significantly greater than the instructions proper) does optimal N fall below the upper end of the range.

This occurs for several reasons. First, as with the bus, the number of computation instructions on the critical path is decreasing linearly with N, while the growth in the time to perform any of those instructions, t_I, is generally increasing

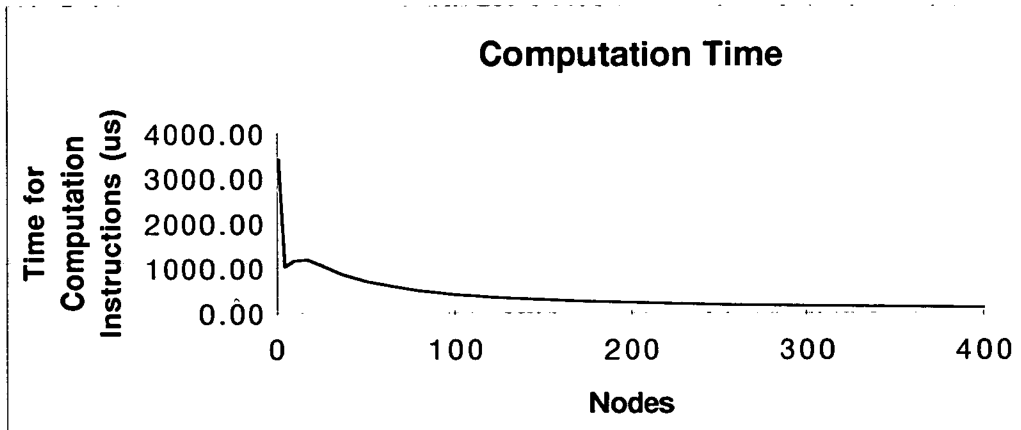


Figure 5–17: Mesh Model - Time for Computation Only

only sub-linearly (see figure 5–3). Figure 5–17 shows the time spent by a processor on computation only, as a function of the number of nodes. As with the bus, this argues for a large number of slow devices if other effects are ignored.

Also, communication latency is affected only insofar as the hardware parameters are also so affected. Unlike non-local communications, blocking is not significantly increased by increasing N, and so t_ρ is most sensitive to speed of the the three components types – processors, memory and routers - and also to the width of the router I/O port. From equation 4.9 it can be seen that latency can be dominated by either the router cycle time, t_R, or by the message preparation time which is strongly dependent on i_s.

A comparison of figure 5–18 with figure 5–19 shows how t_ρ follows the router cycle time t_R, when i_s is small (1/100th of i_c in this case). If i_s is increased (e.g. if communication required the services of high-level system services such as a OS kernel), then t_ρ depends principally on the performance of the processors themselves and on memory performance if μ is sufficiently high.

Figure 5–20 shows that the message length is largely unaffected by N, and so does not affect optimal grain.

Message length remains fairly constant because while router width decreases

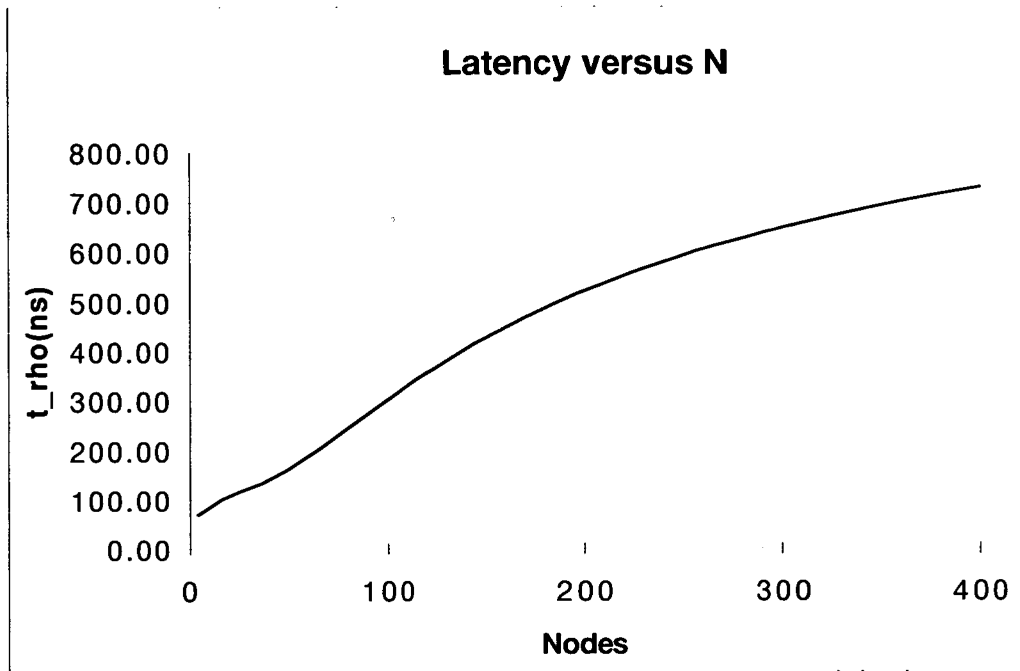


Figure 5–18: Mesh Model - Latency, t_ρ versus N

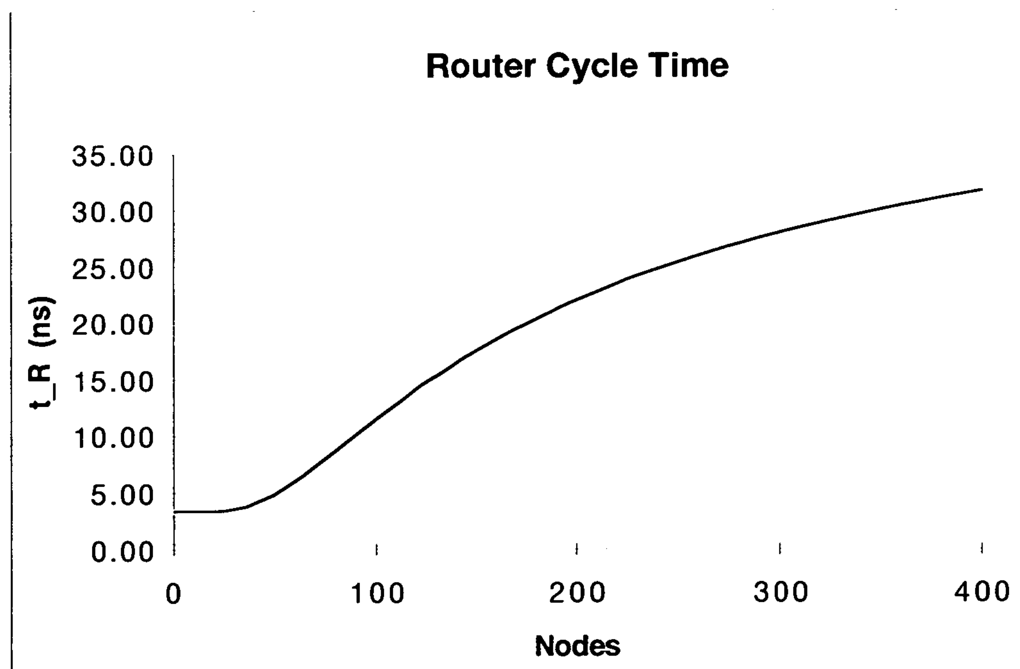


Figure 5–19: Mesh Model - Router Cycle Time, t_R, versus N

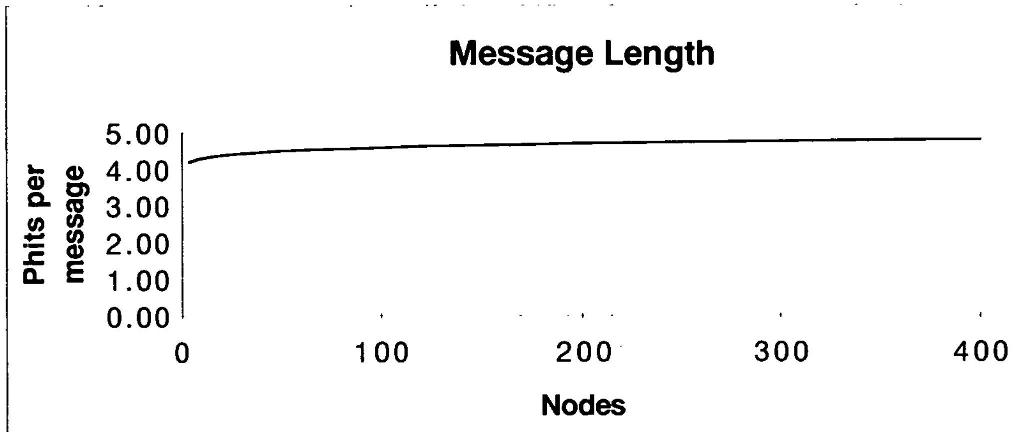


Figure 5–20: Mesh Model - Message Length versus N

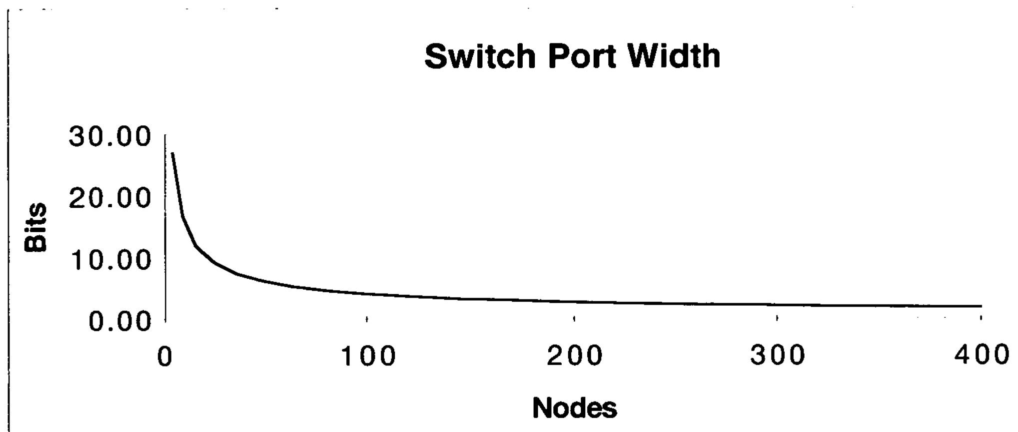


Figure 5–21: Mesh Model - Router Width versus N

with N (figure 5–21), the halo size, and therefore the number of bytes per message, is also decreasing (figure 5–22).

The conclusion to be drawn from figures 5–19 to 5–22 is that, for this mesh algorithm, interprocessor communication is not a serious limitation on the use of parallelism. The narrowing of the width of router channels is effectively countered by the steady decrease in the amount of data being transmitted at the end of each iteration. In addition, router cycle time, t_R is not increasing fast enough with N to overcome the benefits produced by the decreasing critical path.

Finally, the space scaling due to overlap areas is relatively slight and while the

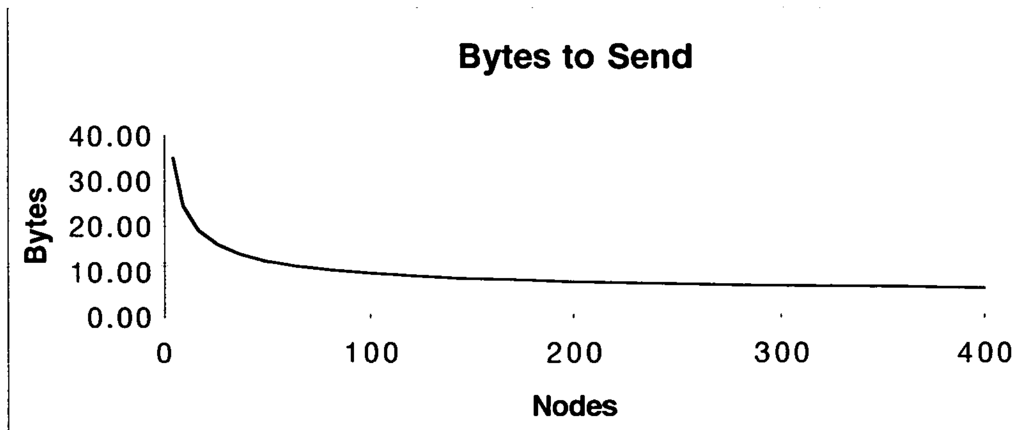


Figure 5–22: Mesh Model - Bytes per Message versus N

memory access time is increasing with N, the linear decrease in the number of memory references on the critical path more than compensates. For a different space scaling scenario (e.g. if local code space on each processor was included), the increased memory requirements may begin to cause sufficient degradation in memory speed as to reduce the optimal N.

5.5 Shared Bus Example

This section describes an example of the bus architecture showing how optimal N and processor type can be identified.

For the shared bus, the grain of computation is important. The more instructions that are required per point (i.e. the coarser the grain), the more processors can be used. Ironically this means that for a fixed *total number* of instructions, the more parallelism available (i.e. the finer the grain), the less able is the bus to support it. Figure 5–23 was generated from the shared bus models in chapter 3 by plotting execution time against N for a range of values of i_c and extracting the N corresponding to minimum T_{SM} in each case. The other parameters are: $P = 10000$ (number of points), $\mu = 0.1$ (local memory accesses per instruction),

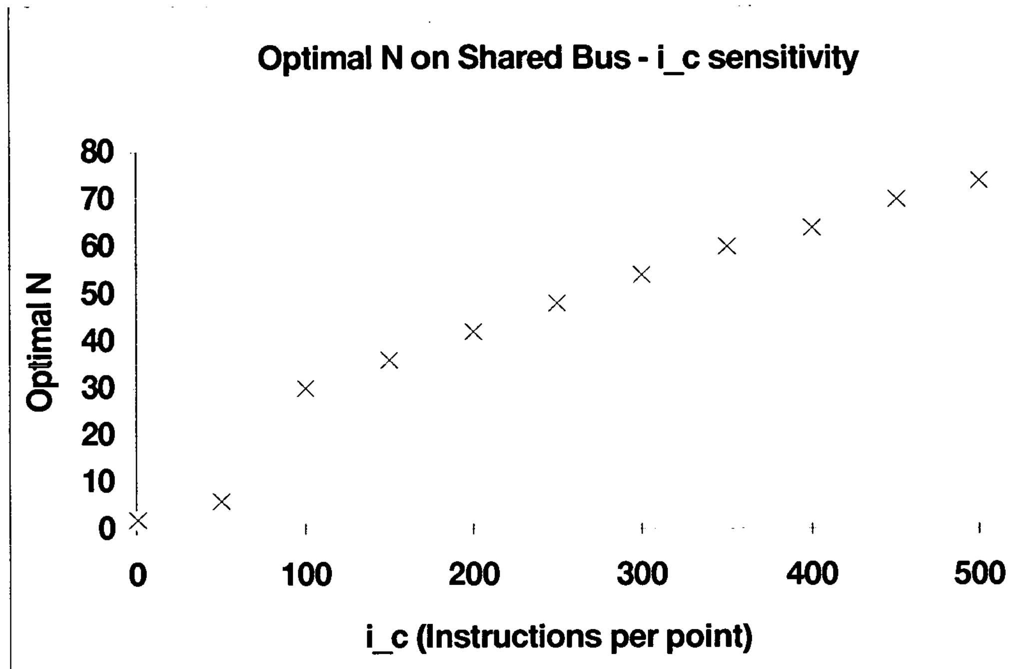


Figure 5–23: Bus Model - Optimal N versus i_c

$m = 0.5MBytes$ (local memory per processor), $n_{M_s} = 5MBytes$ (total shared memory), $R_P = \$500$ (total cost of processors), $R_M = \$250$ (total cost of all memory).

With a total processor cost, R_P, of \$500, the graph indicates that the optimal processor, for this particular algorithm and ignoring electrical effects on the bus, varies from extremely high performance devices such as the MPC620 for very fine grain computations ($i_c < 10$), to more modest processors (relatively speaking) such as the MPC601 when i_c is large. Of course, this analysis focuses on the mid to high range of all available processors.

The amount of local memory activity is also a key factor in determining which processor type should be used. Because performance of the memory devices studied degrades slower, as N increases, than the length of the critical path, local memory accesses are not an obstacle to the use of parallelism. On the contrary, despite a slowing of memory with increasing N, the fact that local accesses tend to keep processors off the bus suggests that increasing μ will correspond with an

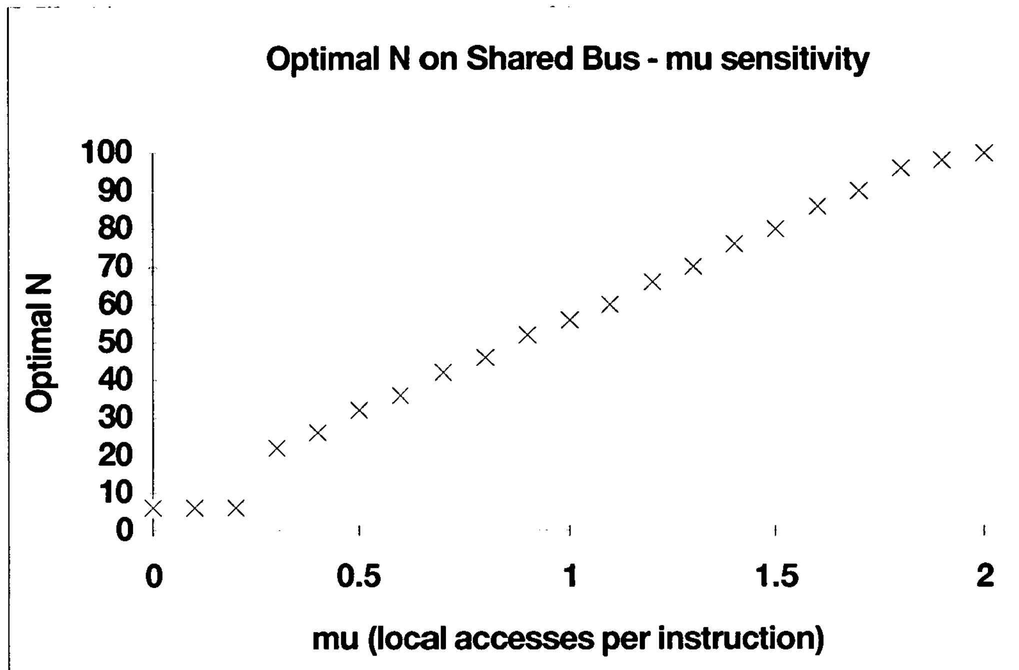


Figure 5–24: Bus Model - Optimal N versus μ

increase in the optimal number of nodes. Figure 5–24 was generated in the same way as figure as the above i_c sensitivity graph and shows optimal N against μ. The other parameters are: $P = 10000$ (number of points), $i_c = 100$ (instructions per point), $m = 0.5MBytes$ (local memory per processor), $n_{M_s} = 5MBytes$ (total shared memory), $R_P = \$500$ (total cost of processors), $R_M = \$100$ (total cost of all memory).

Note that the example here, as throughout this study, is based on SRAM devices, as presented in [18]. Systems using DRAM devices may require a different function in place of equation 2.16, however the method of analysis would be the same.

5.6 Summary

In this chapter, simulation results have been used to validate the models presented previously. The models' sensitivity to variation in some of their key parameters were then discussed.

In the bus model, since aggregate performance tended to increase almost linearly with N above a certain number of nodes (about 10 for a total node cost of $500), the bus loading was the key factor in determining optimal N. Similarly, since memory performance was relatively unaffected by cost at the low-end of the device range, increasing the amount of memory (a linear space scaling in the simulations) did not, in itself, act to degrade performance seriously. The main factor limiting the use of parallelism was the relationship between shared memory access time and N. Once the bus began to load the benefit of adding more processors was quickly outweighed by the increased bus waiting time. This was further emphasized when super-linear space-scaling was considered.

For the message-passing mesh architecture, communications locality and modest space scaling meant that optimal N was almost always at the highest N investigated. Locality makes a decrease in router speed and a narrowing of inter-node links the key communications drawbacks of increasing N. The problem of router speed decreasing with N was generally outweighed by the increased concurrency. The narrowing links were not a serious limitation because the amount of data being sent decreased with N and so message length remained relatively constant. For systems with more severe space scaling, or non-local communications patterns, optimal N could be expected to decrease.

Chapter 6

Concluding Remarks

6.1 Summary and Conclusions

Parallel architectures have, in terms of decreasing the execution time of a problem, one significant advantage over sequential machines:

- The amount of work allocated to the most heavily-loaded processor is less in the parallel machine.

If the total hardware cost is fixed, then opposing this advantage are several disadvantages:

- The processors in the parallel machine are slower.
- The memory devices in the parallel machine are slower, since memory requirements typically increase with the number of processors.
- The components used to connect the processors and memories in the parallel machine have performances and bandwidth which vary inversely with the number of processors.
- Additional work is required of the processors in the parallel machine, due to communications.
- Sharing of resources among the various processors may force delays when a given resource is found to be busy.

This thesis has presented an investigation of these advantages and disadvantages and their net effect on the execution time of a problem on a system of N nodes.

The first conclusion of this work is to reaffirm the scale of the problem at hand. Optimizing hardware granularity is a significant problem depending on a large set

of input parameters and any particular result depends heavily on the system under investigation.

The principal aim of this thesis was not to give precise results concerning optimal N, but rather to present an approach to the problem which could, in any specific circumstances, lead to a better understanding of the tradeoffs inherent in varying the number of processors. This has been done by examining two specific systems, and describing the parallel computation in a way that can be expressed as a workload on the hardware. The methods used, and the underlying principle – that a hardware enhancement in one part a fixed cost system will have an associated performance reduction elsewhere – can be applied more generally, and in a wider area than simply optimizing grain.

In the two systems examined, an important factor was the difference between the rate of decrease with N of the length of the critical path and the rate of decrease of component speed, again with the increase in the number of processors. Since hardware cost:performance was modelled as showing very rapidly diminishing returns beyond a certain cost, and since the critical path was decreasing roughly linearly with N, one would expect that high degrees of parallelism would be possible, unless other factors such as resource sharing, or communications overhead became significant. This was seen in the different behaviour of the two systems with different interconnection strategies. Whereas the lack of blocking in the mesh allowed N to be increased to a maximum, the poor scalability of the single shared bus did not. However, the mesh's scalability was seen under very localized communication so that the only serious impact on latency was through a decrease in the switch speed. Since switching time was, because of the nature of the hardware cost:performance, scaling up at a lower rate than that of the decrease of the critical path, an increase in N did not seriously degrade overall performance.

Some general points are noted:

1. A VLSI cost model was presented and the expected sub-linearity of VLSI devices was identified in memory and processor components. As suggested in [72] both the feature size and die area (number of transistors) are significant factors affecting device cost, with a key effect being wafer fabrication equipment depreciation.

2. A survey of microprocessor data was carried out, and this was combined with the cost model data to determine a cost:performance function for the basic components. Reasonable approximate curves were proposed.

3. Space scaling was described in detail for a grid decomposition, and this was used to show how memory system performance will degrade with N.

4. A scheme for representing a DAG algorithm as a workload on N processors was presented and used to develop the performance model in detail.

5. Verilog simulations were presented and provided validation of the models to within 3.5% for the shared-bus, and 11% for the mesh.

6.2 Suggestions for Future Research

Several investigations spring from the current one. In general, these would involve taking a specific aspect of the analysis and developing more precise models with more precise (but less general) results.

The effects, on cache management strategies, of increasing N is one possible area of study. As N increases, the amount of work required to maintain cache coherency in some schemes may become prohibitively large. A possible approach is to express the change in the amount of cache-maintenance work done as a function of N and to use this in conjunction with the cost:performance models presented here, to identify an overall effect on execution time.

An investigation into the effects of different (blocking) communications patterns, would also be informative. As was shown in this thesis, while the mesh's non-blocking communications gave optimal N at the highest value, the bus could quickly saturate making large N a poor choice. Work presented in [2] could provide a starting point for a more general latency model.

Further work could be done to extend the processor cost model out beyond single-chip VLSI devices, to include board-level processors. In the analysis presented here, obviously parallel architectures are needed when the total processor cost is greater than that of leading edge single-chip device. By identifying a cost:performance function for board-level CPUs (perhaps using bipolar technologies) the tradeoff between number and speed of processors may be investigated to higher cost ranges. It would also be useful to devise a more precise model of switch cost:performance than the one presented here. Switch components are still relatively uncommon but several experimental and a few commercial devices have been produced and could form the beginning of a study [15] [50] [34].

Also, the general principle of fixing system cost in order to investigate the net effects of a proposed optimization, could be applied to tradeoffs among the amount of resource used for each of the three main hardware functions: processors, memory and interconnect. Keeping the three hardware functions independent, as was done here, ignores the fact that if the performance of one component drops very rapidly as N increases, it may become a bottleneck which dominates all other effects. Bearing in mind that varying N is only one of several options available when designing an optimal architecture, and investigation into trading off processor, memory and interconnect cost would be difficult, but useful.

Alternative sources of space scaling effects are worthy of investigation. For example, as mentioned in chapter 1, the discussions of the merits of recomputation in scheduling (e.g. [35] and [57]) currently take no account of the impact on memory performance of executing the same task on several processors. For

high performance networks using memory with a poor cost:performance function, recomputation may become a liability.

Finally, the future implications of this modelling approach could be investigated, particularly with respect to examining the relative importance of scalability and modularity in parallel systems.

6.2.1 Scalability and Modularity

Ultra-high performance is not the only reason for considering parallism in computer architectures. Another, possibly more widely useful property, is the ability to increase the performance of a parallel system by adding more nodes. This is useful for two reasons. First, it allows an existing piece of hardware to be retained and improved, rather than discarded as is the case today. Second, it allows a range of products, providing a range of costs and performances, to be developed from a single design effort in both hardware and software. The latter is particularly useful since each individual VLSI component typically has a minimum support overhead (design, marketing, sales support, etc.) and so the fewer different components an OEM needs to support the target market, the better.

However, it is the notion that the ability to add nodes to an existing system is valuable, that warrants further study. For example, the ability to add more nodes to a fixed-degree topology such as the mesh has been used to argue its merits over a variable-degree topology such as the binary cube. The ability to add more nodes should be considered in light of the fact that when the time to upgrade arrives, VLSI technology can be expected to have moved on, making the upgrade with older nodes less attractive.

Specifically, the problem is as follows:

In year Y_1, a system is built using N_{Y_1} nodes. Some time later, in year Y_2, a

sum of money is made available with which to upgrade the available computing resource. The user has a choice:

1. Purchase more nodes of the original type, and add these to the existing system, or

2. Discard the existing system and build a new optimal system, with the available technology.

Apart from the various factors discussed in this thesis, a key factor is the growth of VLSI device performance with time. A useful starting point for this is [72]. Practical experience at Edinburgh, in the early 1990's, suggests that, at the moment, VLSI is still advancing fast enough to make scalablity for upgrading's sake of limited worth. This may change however, as the technology reaches its upper limits. The circumstances under which this would occur merit further investigation.

Appendix A

Microprocessor Survey Data

Key to Survey Table

Name	Device Name/Number
Mfr	Principal Manufacturer
Clk	External Bus Clock Frequency (MHz)
Mips	Dhrystone MIPS
Mflops	Millions of floating-point operations per second
Int92	Maximum quoted Int92 in system using this processor
Int92	Maximum quoted FP92 in system using this processor
KD/s	Kdrhystones per second
Area	Die area mm^2
Trans	Number of transistors (1000's)
Ftr	Feature size (drawn) (microns)

Name	Mfr	Clk	Mips	Mflops	Int92	FP92	KD/s	Area	Trans	Ftr
21064	DEC	133			62.6	107.8		234	1700	0.8
21064	DEC	150			74.3	126.0		234	1700	0.8
21064	DEC	166						234	1700	0.8
21064	DEC	200			106.5	200.4		234	1700	0.68
21064A	DEC	225			135	205			2800	0.68
21064A	DEC	225			170	290			2800	0.68
21064AA	DEC	100							1750	0.68
21064AA	DEC	133			65	112			1750	0.68
21064AA	DEC	150			74	126			1750	0.68
21064AA	DEC	166			90	140			1750	0.68
21064AA	DEC	175			114	162			1750	0.68
21064AA	DEC	182			103	176			1750	0.68
21064AA	DEC	190			122	185			1750	0.68
21064AA	DEC	200			130	184			1750	0.68
21066	DEC	166			70	105				
21068	DEC	66			30	50				
21164	DEC	225			135	205			2800	0.5
21164	DEC	275			170	290			2800	0.5
21164	DEC	320			201.5	366.5			2800	0.5
29000	AMD	16								
29000	AMD	20								
29000	AMD	25								
29000	AMD	30								
386/387	INTEL	33			6.2	3.3			1200	
486DX	INTEL	25			13.3	6.6			1200	
486DX	INTEL	33			18.3	9.5				
486DX	INTEL	50			30.1	14			1200	
486DX2	CYRIX	66			32.2	16				0.7
486DX2	CYRIX	80								0.7
486DX2	INTEL				25.4	15.9				
486DX2	INTEL	66			32.2	16				0.7
486DX2	INTEL	80								0.7
486DX3	INTEL	99			48	24			1200	

Name	Mfr	Clk	Mips	Mflops	Int92	FP92	KD/s	Area	Trans	Ftr
486DX4	INTEL	100								
486SLC	INTEL	33								
486SX	INTEL	25								
486SX	INTEL	33								
6502	ROCKWELL									
80186	INTEL	8								
80188	INTEL	8								
80286	INTEL	10								
80386	INTEL	16								
80386	INTEL	20								
80386	INTEL	25								
8080A	INTEL	2								
8088	INTEL	5								
80C186	INTEL	10								
80C188	INTEL	10								
80C286	INTEL	16								
80C286	INTEL	12.5								
80C85A	INTEL	3								
80C86	INTEL	8								
84C00A	TOSH	6								
ARM2	ARM	8					5.3	5.29	25	1.2
ARM3	ARM	25					14.8		250	0.8
ARM6	ARM	33						7.15	33.5	0.8
ARM60	ARM							16	43	0.6
ARM610	ARM	25	25		24			26	359	0.6
ARM7	ARM	33					53	4.96	35.6	0.8
ARM700	ARM	55					69	68	579	0.8
ARM710	ARM	33			32			34	570	0.6
MC6800	MOT	1								
MC68000	MOT	8						30	68	2.5
MC68000	MOT	10						30	68	2.5
MC68000	MOT	12						30	68	2.5
MC68010	MOT	8						41.4	84	2.6
MC68010	MOT	10						41.4	84	2.6
MC68010	MOT	12						41.4	84	2.6
MC68020	MOT	12						39.7	190	0.8
MC68020	MOT	16						39.7	190	0.8
MC68020	MOT	20	5.2	0.19			9	39.7	190	0.8
MC68020	MOT	25	6.5	0.24			11	39.7	190	0.8
MC68020	MOT	33	8.7	0.32			15	39.7	190	0.8
MC68030	MOT	16	4.5	0.26			7.8	55.4	273	1.0
MC68030	MOT	20	5.4	0.32			9.4	55.4	273	1.0
MC68030	MOT	25	6.7	0.4			11.7	55.4	273	1.0
MC68030	MOT	33	9.0	0.53			15.6	55.4	273	1.0
MC68030	MOT	40	10.8	0.6			18.8	55.4	273	1.0
MC68030	MOT	50	13.5	0.8			23.5	55.4	273	1.0
MC68040	MOT	25	26.1	3.5	19		45.5	163	1170	0.65
MC68040	MOT	33	34.8	4.7			61	163	1170	0.65
MC68040	MOT	40	43.8		35	23	72.7	163	1170	0.8
MC68040	MOT	45	41.3	5.6			72.7	163	1170	0.65
MC68060	MOT	50	90		49			198	2500	0.5
MC6809	MOT	1								
MC68EC000	MOT	8	1.2				2.1	28.6	68	0.8
MC68EC000	MOT	15	2.5				4.4	28.6	68	0.8
MC68EC040	MOT	20	20.9	0.2			36.4	128	777	0.65
MPC601	MOT/IBM	50			51	63		120	2800	0.6

Name	Mfr	Clk	Mips	Mflops	Int92	FP92	KD/s	Area	Trans	Ftr
MPC601	MOT/IBM	60						120	2800	0.6
MPC601	MOT/IBM	66			62	80		120	2800	0.6
MPC601	MOT/IBM	80			77	93		120	2800	0.6
MPC601	MOT/IBM	100			110	130		74	2800	0.5
P24C	INTEL	99								
P24T	INTEL									
P54C	INTEL									0.6
P54MC	INTEL									
P6	INTEL									
P86	INTEL/HP									
PA7100	HP	99			109.1	167.9		202	806	0.75
PA7100LC	HP	60			58.1	78.5		202	906	0.75
PA7100LC	HP	75			82.6	127.2		202	906	0.75
PA7100LC	HP	80			84.1	79		202	906	0.75
PA7100LC	HP	100			101	137		202	906	0.75
PA7150	HP	125			135	200		202	806	0.75
PA7200	HP									
PA9000	HP									
PENTIUM	INTEL	60			58.3	52.2		290	3100	0.8
PENTIUM	INTEL	66			64.5	56.9		290	3200	0.65
PENTIUM	INTEL	90			90	72.7			3200	0.65
PENTIUM	INTEL	99			96.8	85.4				
PENTIUM	INTEL	100			100	80.6			3200	0.65
POWER	IBM	63			73.3	134.6				
POWER2	IBM	72			126	260.4			19200	
POWER2532	IBM	25			20.9	39.4				
POWER3332	IBM	33			27.7	51.9				
POWER3364		33			28.5	64.6				
R4200	MTI	40			50	24			1300	0.6
R4200	MTI	40			55	30			1300	0.6
R4400	MTI	50			59.1	62.1		186	2300	0.6
R4400	MTI	75			94.2	105.2		186	2300	0.6
R4400	MTI	150							2300	0.6
R4400	MTI	200			117	131		134	2300	0.35
R4600	MTI	50			60	68			1900	0.64
R4600	MTI	67			92.1	82			1900	0.64
R8000/TFP	MTI	75	300	300	108	310		298	3400	0.5
RSC3308	IBM	33			20.4	29.1				
RSC4608	IBM	46			28.5	39.9				
Sparc(H)	SUN	66			67	93				0.65
Sparc(H)	ROSS	100			111	135		135		0.5
Sparc(M)	SUN	50			23	18				0.8
Sparc(S)	SUN				89	103				
Sparc(S)	SUN	50			65	80		256	3000	0.7
Sparc2(M)	SUN	100			63	56				
T5	MTI	100			250	300			5200	
Z80A	ZILOG	4								
Z80B	ZILOG	6								
Z84C	ZILOG	8								
i860XP	INTEL	50		75					2500	0.8

Appendix B

Simulation

B.1 Bus Simulation Results

This section provides results for the bus simulations. Three sets of experiments were performed; one for low-cost components, one for mid-range devices, and one for devices at the high-end of the cost and performance ranges. Tables B–1 to B–3 provide a summary of the parameters used in the bus simulations. Figures B–1 to B–9 plot execution time against N for each experiment. Simulation results matched the model to within an average of under 3.5%.

Low-cost components								
			Memory (MBytes)		CPU		MEM	
P	i	μ	Local	Shared	Cost	t_I	Cost	t_M
Experiment 1								
10000	10	0.1	0.5	5	2.36	41.67	11.5	70
10000	100	0.1	0.5	5	2.36	41.67	11.5	70
10000	500	0.1	0.5	5	2.36	41.67	11.5	70
Experiment 2								
10000	10	0.5	0.5	5	2.36	41.67	11.5	70
10000	100	0.5	0.5	5	2.36	41.67	11.5	70
10000	500	0.5	0.5	5	2.36	41.67	11.5	70
Experiment 3								
10000	10	1.0	0.5	5	2.36	41.67	11.5	70
10000	100	1.0	0.5	5	2.36	41.67	11.5	70
10000	500	1.0	0.5	5	2.36	41.67	11.5	70

Table B–1: Bus Simulation Experiments – 1 of 3

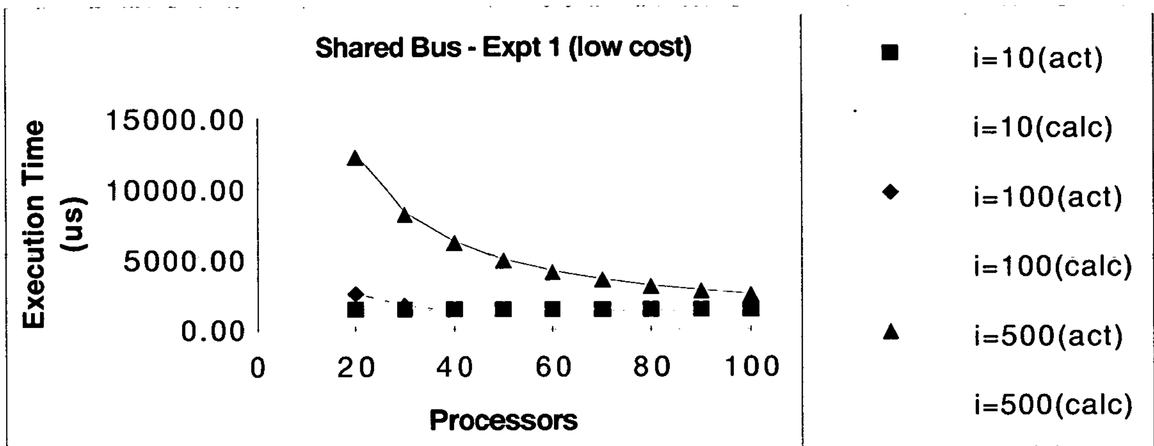


Figure B–1: Bus Experiment 1 - Low-cost Components

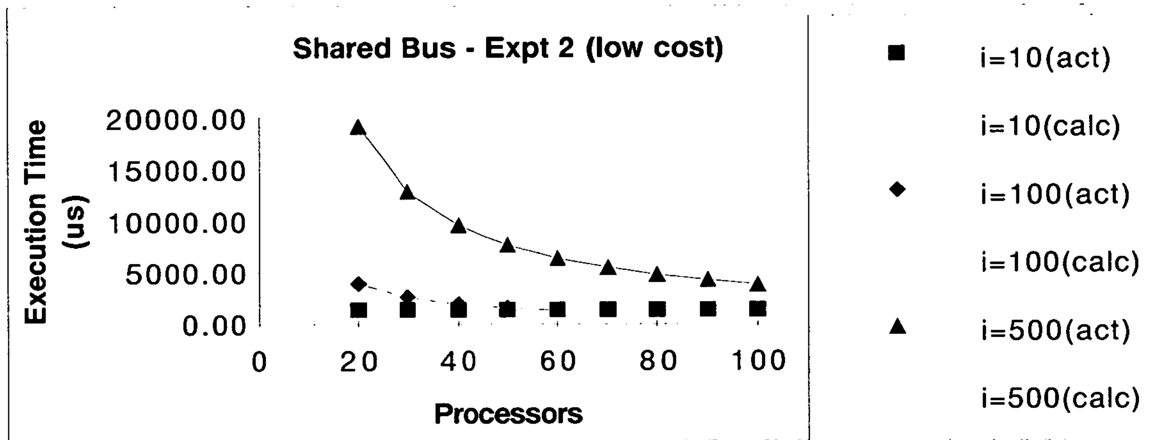


Figure B–2: Bus Experiment 2 - Low-cost Components

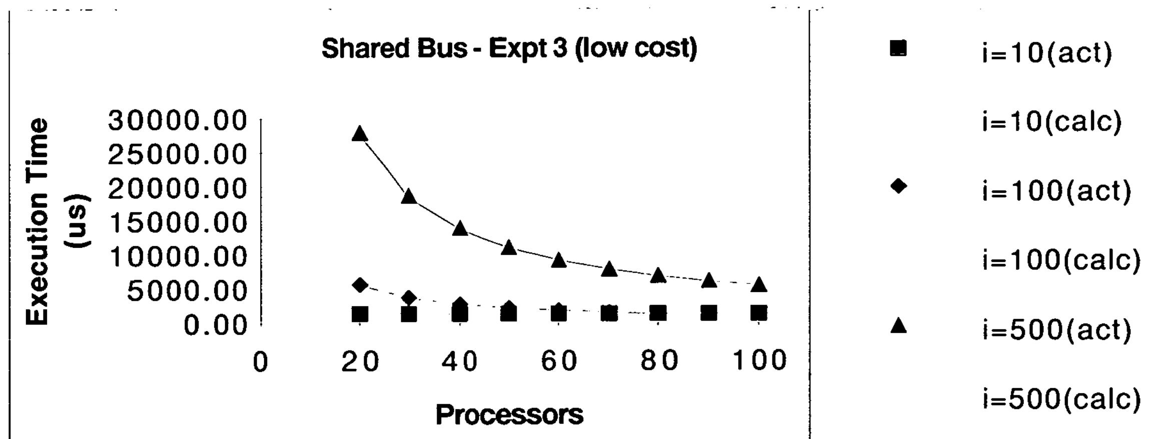


Figure B–3: Bus Experiment 3 - Low-cost Components

Mid-range components								
			Memory (MBytes)		CPU		MEM	
P	i	μ	Local	Shared	Cost	t_I	Cost	t_M
Experiment 1								
10000	10	0.1	0.5	5	50.86	8.55	15.82	32
10000	100	0.1	0.5	5	50.86	8.55	15.82	32
10000	500	0.1	0.5	5	50.86	8.55	15.82	32
Experiment 2								
10000	10	0.5	0.5	5	50.86	8.55	15.82	32
10000	100	0.5	0.5	5	50.86	8.55	15.82	32
10000	500	0.5	0.5	5	50.86	8.55	15.82	32
Experiment 3								
10000	10	1.0	0.5	5	50.86	8.55	15.82	32
10000	100	1.0	0.5	5	50.86	8.55	15.82	32
10000	500	1.0	0.5	5	50.86	8.55	15.82	32

Table B–2: Bus Simulation Experiments – 2 of 3

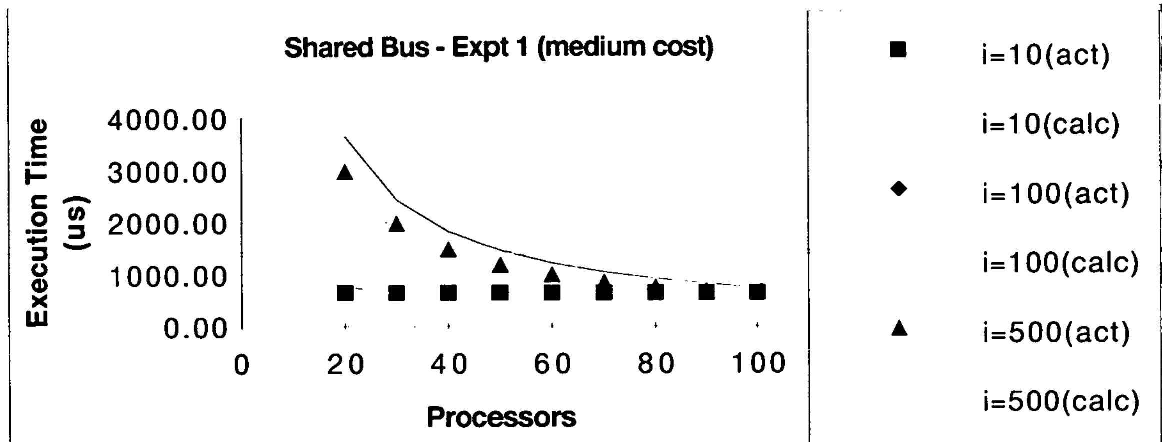


Figure B–4: Bus Experiment 1 - Mid-range Components

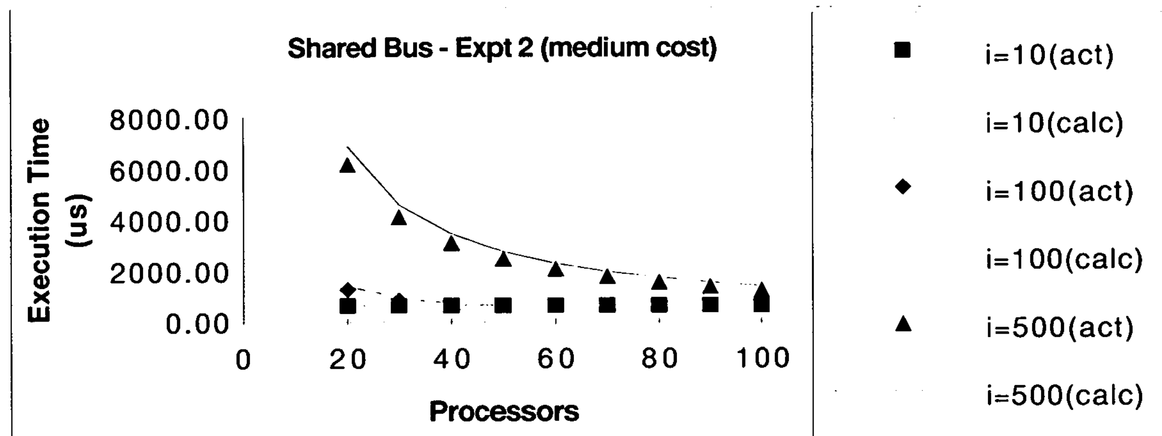


Figure B–5: Bus Experiment 2 - Mid-range Components

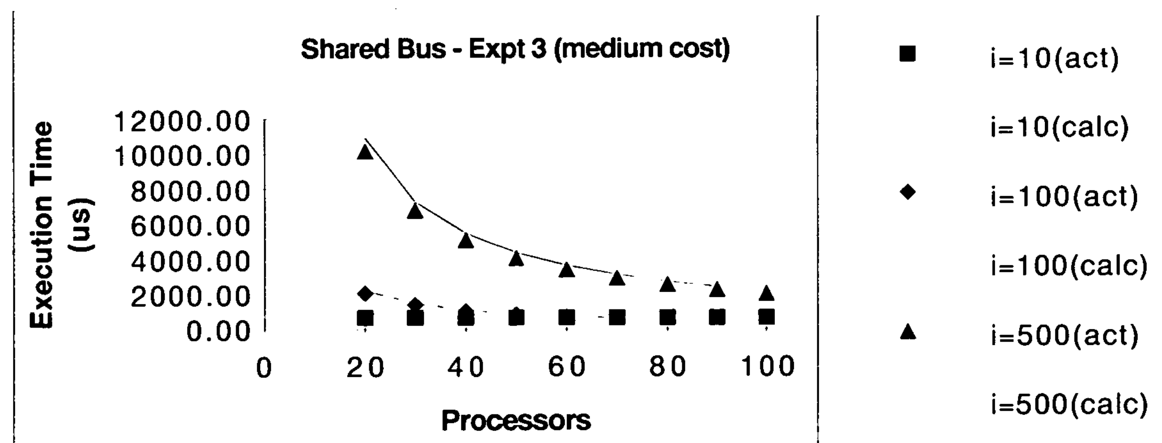


Figure B–6: Bus Experiment 3 - Mid-range Components

High-end components								
			Memory (MBytes)		CPU		MEM	
P	i	μ	Local	Shared	Cost	t_I	Cost	t_M
Experiment 1								
10000	10	0.1	0.5	5	354.5	3.33	40.23	9
10000	100	0.1	0.5	5	354.5	3.33	40.23	9
10000	500	0.1	0.5	5	354.5	3.33	40.23	9
Experiment 2								
10000	10	0.5	0.5	5	354.5	3.33	40.23	9
10000	100	0.5	0.5	5	354.5	3.33	40.23	9
10000	500	0.5	0.5	5	354.5	3.33	40.23	9
Experiment 3								
10000	10	1.0	0.5	5	354.5	3.33	40.23	9
10000	100	1.0	0.5	5	354.5	3.33	40.23	9
10000	500	1.0	0.5	5	354.5	3.33	40.23	9

Table B–3: Bus Simulation Experiments – 3 of 3

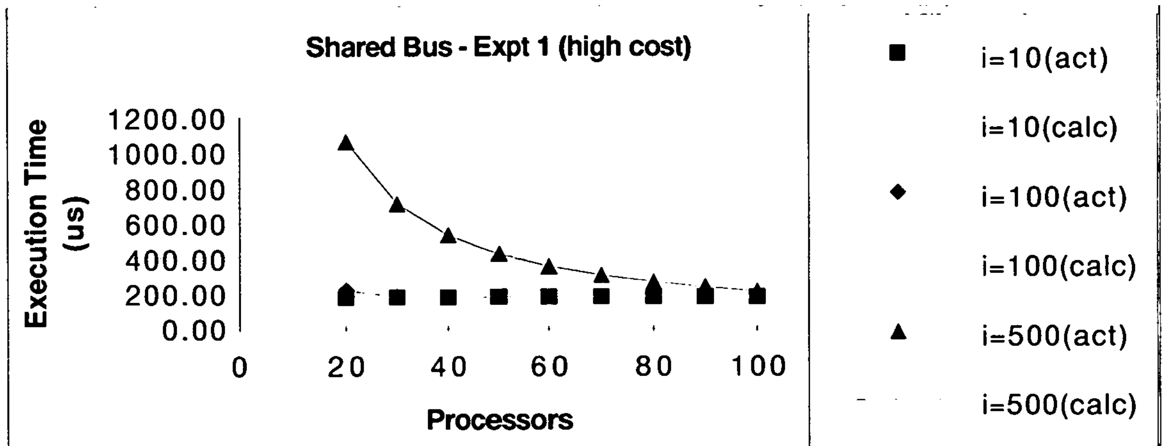


Figure B–7: Bus Experiment 1 - High-end Components

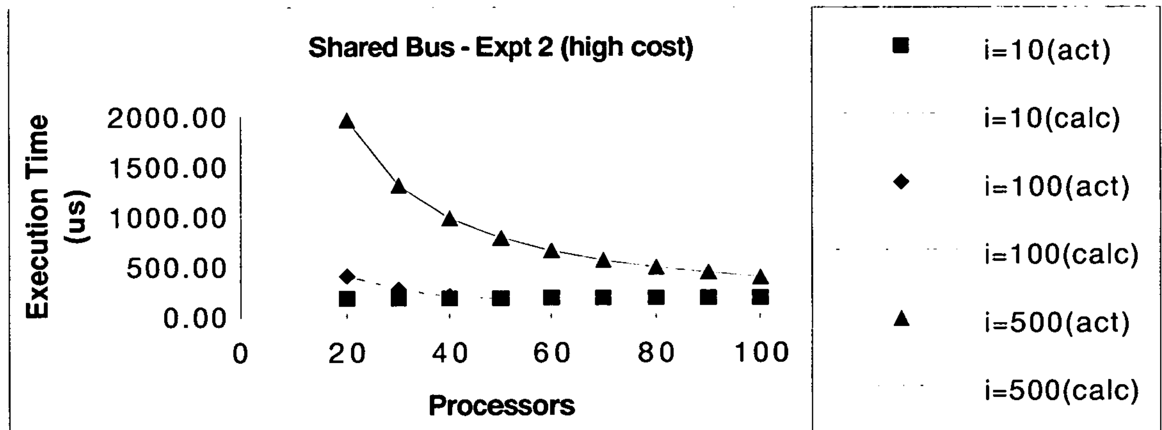


Figure B–8: Bus Experiment 2 - High-end Components

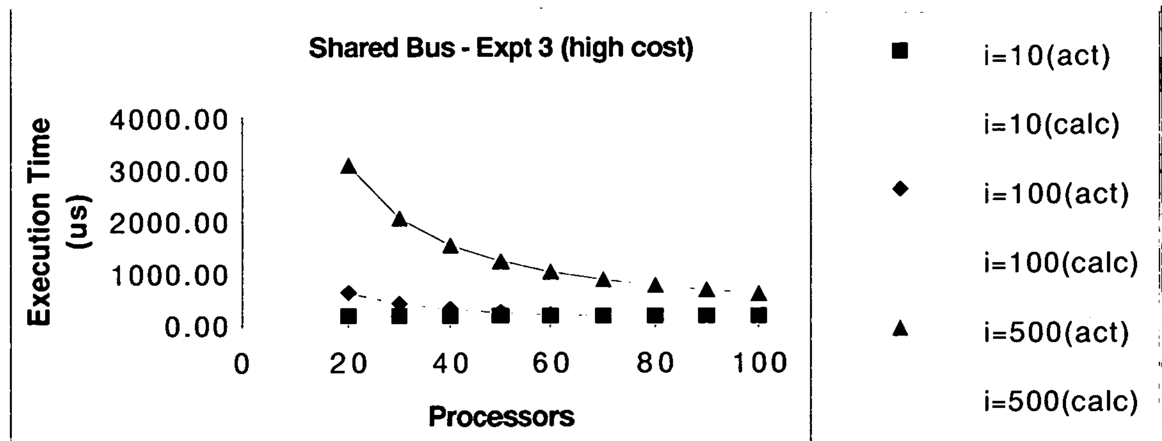


Figure B–9: Bus Experiment 3 - High-end Components

B.2 Mesh Simulation Results

This section provides results for the mesh simulations. As discussed in chapter 5, the simulations used the following fixed parameters throughout. The data space consisted of $P = 4096$ points arranged in two dimensions. The number of iterations was five. Switch cost was fixed at $350 for all simulations. As with the bus simulations, three sets were run corresponding to low-cost, mid-range and high-end memory and processors.

Each experiment consisted of three sets of simulation runs, each for a square mesh from 1 to 400 nodes. Over the three sets one of the key algorithm parameters was varied, while the others were fixed.

Experiment 1a simulated instruction execution only, varying i_c. Experiment 1b again varied i_c but did this within the context of communication and memory operations. Experiments 2a and 2b simulated varying i_s, with first a low then high value for i_c. Experiments 3a to 3c varied μ, first with two values of i_c and no communications, and then with messages. Finally experiment 4 varied the number

Tables B–4 to B–6 summarize the simulations. Figures B–10 to B–33 plot execution time against N for each experiment. Simulation results matched the model to within an average of 11%.

Low-cost Components											
	i_c	i_s	μ	bpg	δ	CPU Cost	t_I	MEM Cost	t_M	SWITCH Cost	t_S
Expt. 1a	1	0	0	1	0	2.36	41.67	11.5	70	354.5	3.33
	100	0	0	1	0	2.36	41.67	11.5	70	354.5	3.33
	500	0	0	1	0	2.36	41.67	11.5	70	354.5	3.33
Expt. 1b	1	50	50	1	1	2.36	41.67	11.5	70	354.5	3.33
	100	50	50	1	1	2.36	41.67	11.5	70	354.5	3.33
	500	50	50	1	1	2.36	41.67	11.5	70	354.5	3.33
Expt. 2a	1	1	50	1	1	2.36	41.67	11.5	70	354.5	3.33
	1	50	50	1	1	2.36	41.67	11.5	70	354.5	3.33
	1	100	50	1	1	2.36	41.67	11.5	70	354.5	3.33
Expt. 2b	100	1	50	1	1	2.36	41.67	11.5	70	354.5	3.33
	100	50	50	1	1	2.36	41.67	11.5	70	354.5	3.33
	100	100	50	1	1	2.36	41.67	11.5	70	354.5	3.33
Expt. 3a	1	0	1	1	0	2.36	41.67	11.5	70	354.5	3.33
	1	0	51	1	0	2.36	41.67	11.5	70	354.5	3.33
	1	0	101	1	0	2.36	41.67	11.5	70	354.5	3.33
Expt. 3b	100	0	1	1	0	2.36	41.67	11.5	70	354.5	3.33
	100	0	51	1	0	2.36	41.67	11.5	70	354.5	3.33
	100	0	101	1	0	2.36	41.67	11.5	70	354.5	3.33
Expt. 3c	50	50	1	1	1	2.36	41.67	11.5	70	354.5	3.33
	50	50	51	1	1	2.36	41.67	11.5	70	354.5	3.33
	50	50	101	1	1	2.36	41.67	11.5	70	354.5	3.33
Expt. 4	50	50	50	1	1	2.36	41.67	11.5	70	354.5	3.33
	50	50	50	10	1	2.36	41.67	11.5	70	354.5	3.33
	50	50	50	20	1	2.36	41.67	11.5	70	354.5	3.33

Table B–4: Mesh Simulation Experiments – 1 of 3

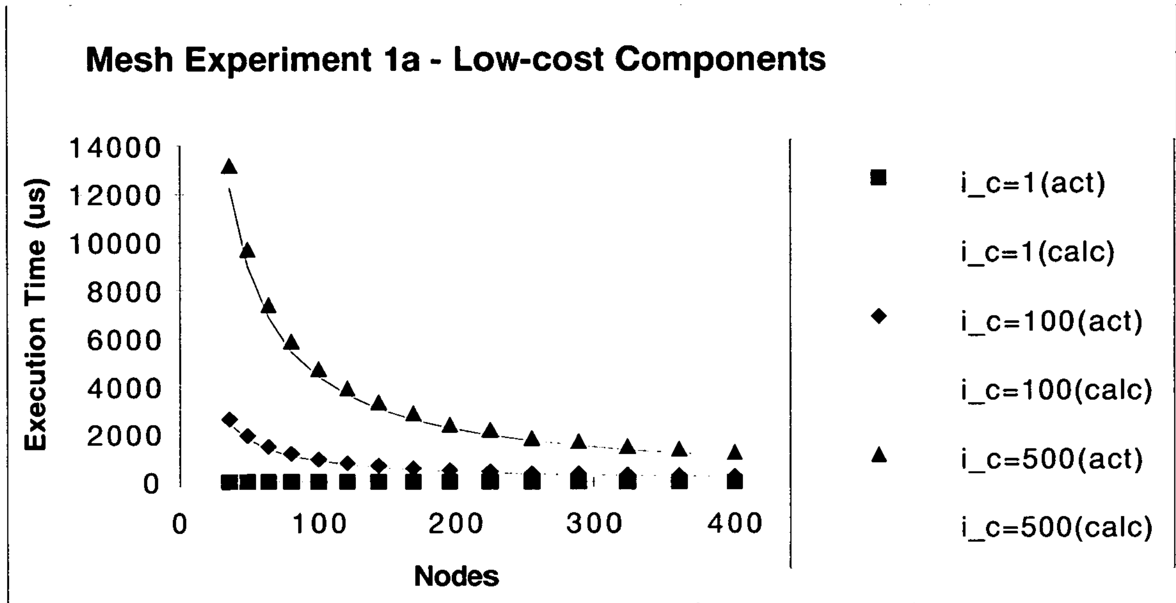


Figure B–10: Mesh Experiment 1a - Low-cost Components

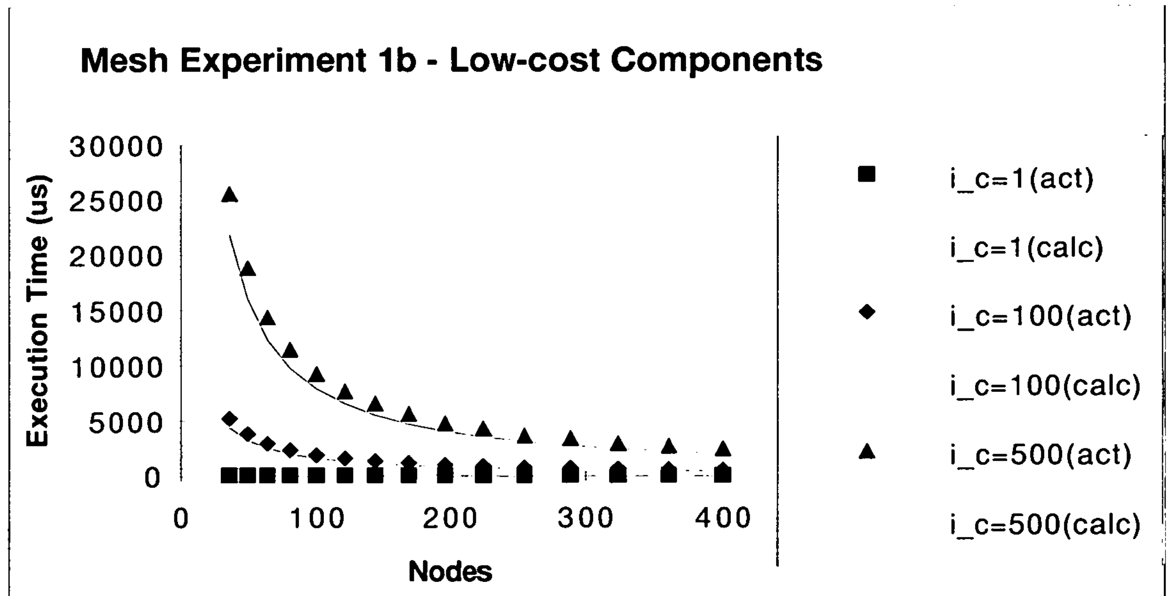


Figure B–11: Mesh Experiment 1b - Low-cost Components

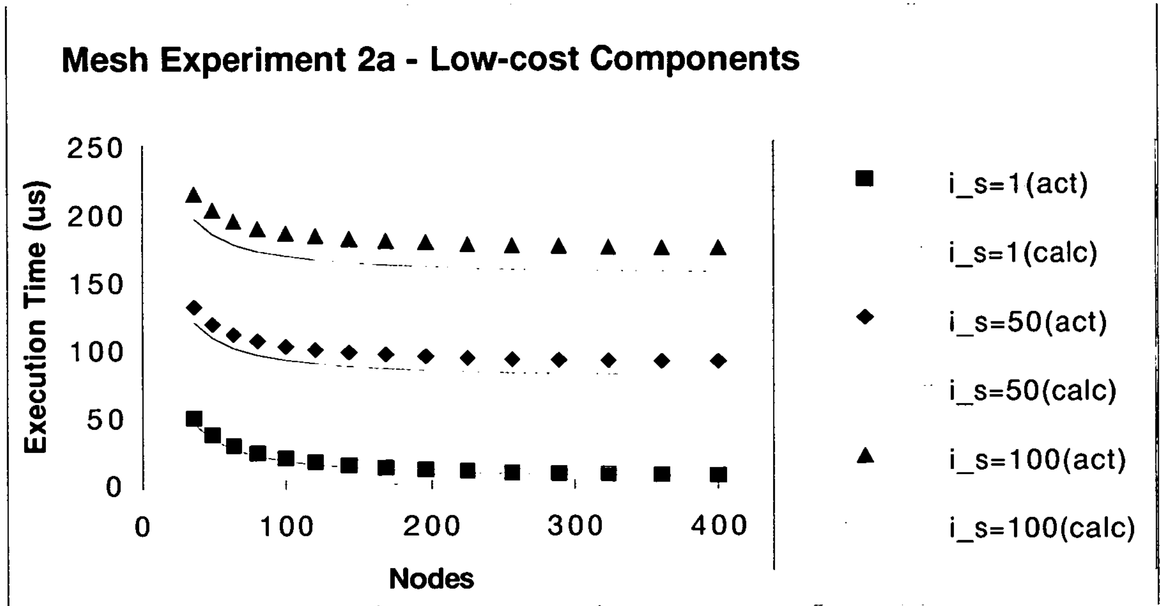


Figure B–12: Mesh Experiment 2a - Low-cost Components

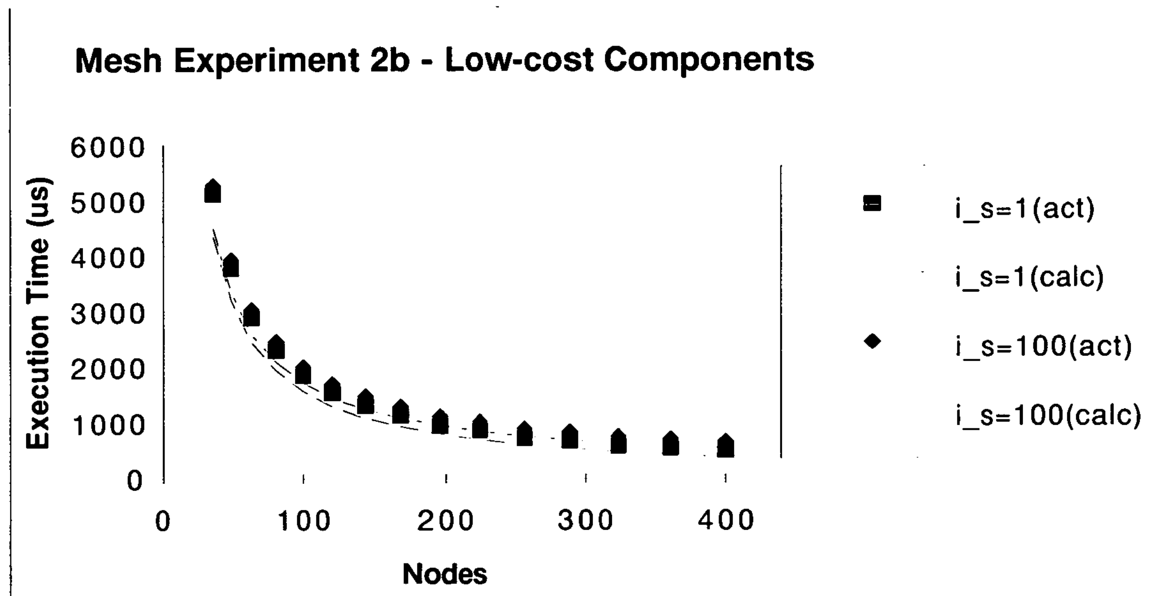


Figure B–13: Mesh Experiment 2b - Low-cost Components

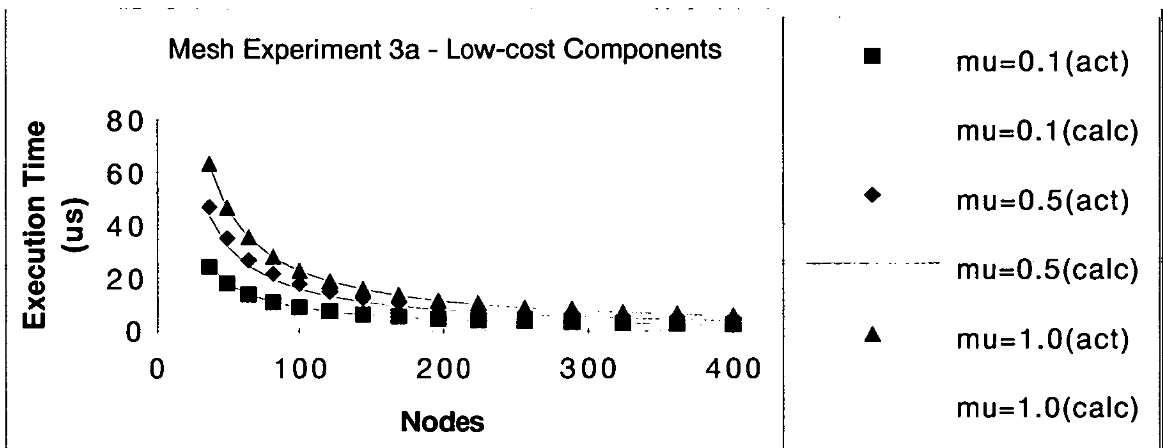


Figure B–14: Mesh Experiment 3a - Low-cost Components

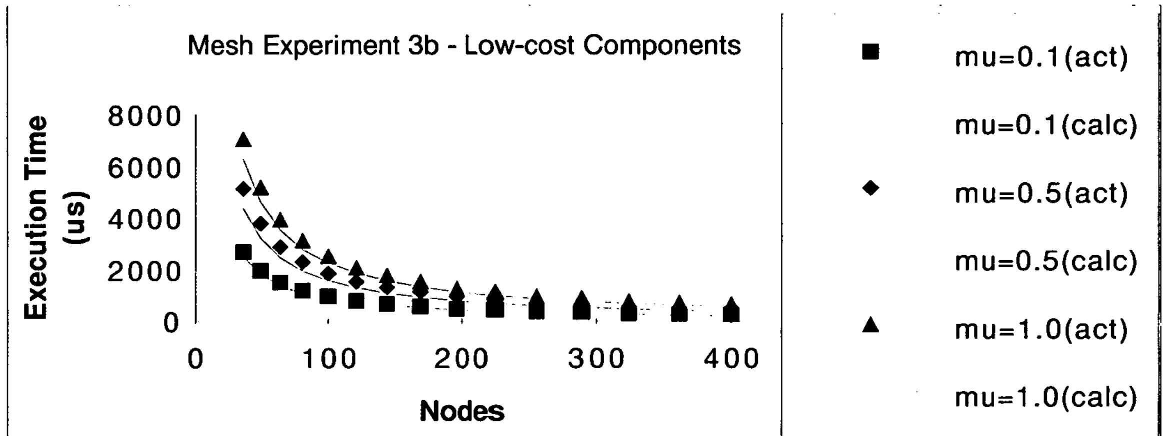


Figure B–15: Mesh Experiment 3b - Low-cost Components

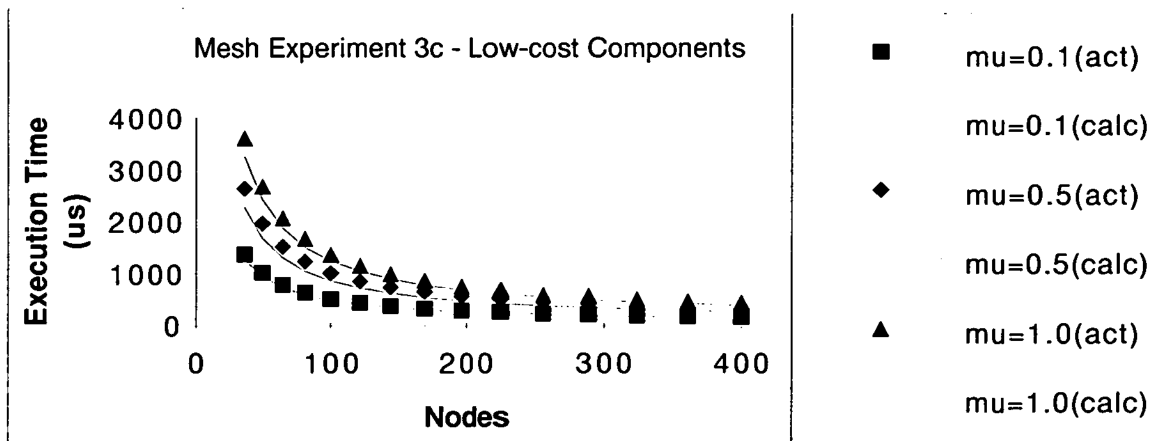


Figure B–16: Mesh Experiment 3c - Low-cost Components

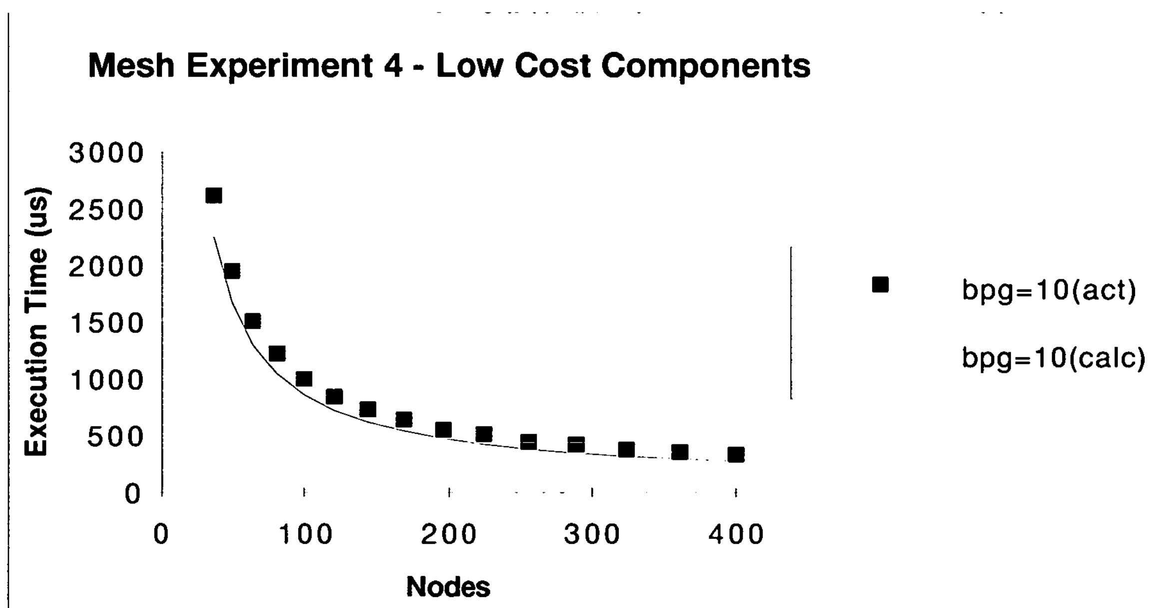


Figure B–17: Mesh Experiment 4 - Low-cost Components

Mid-range Components											
	i_c	i_s	μ	bpg	δ	CPU Cost	t_I	MEM Cost	t_M	SWITCH Cost	t_S
Expt. 1a	1	0	0	1	0	50.86	8.55	15.82	32	354.5	3.33
	100	0	0	1	0	50.86	8.55	15.82	32	354.5	3.33
	500	0	0	1	0	50.86	8.55	15.82	32	354.5	3.33
Expt. 1b	1	50	50	1	1	50.86	8.55	15.82	32	354.5	3.33
	100	50	50	1	1	50.86	8.55	15.82	32	354.5	3.33
	500	50	50	1	1	50.86	8.55	15.82	32	354.5	3.33
Expt. 2a	1	1	50	1	1	50.86	8.55	15.82	32	354.5	3.33
	1	50	50	1	1	50.86	8.55	15.82	32	354.5	3.33
	1	100	50	1	1	50.86	8.55	15.82	32	354.5	3.33
Expt. 2b	100	1	50	1	1	50.86	8.55	15.82	32	354.5	3.33
	100	50	50	1	1	50.86	8.55	15.82	32	354.5	3.33
	100	100	50	1	1	50.86	8.55	15.82	32	354.5	3.33
Expt. 3a	1	0	1	1	0	50.86	8.55	15.82	32	354.5	3.33
	1	0	51	1	0	50.86	8.55	15.82	32	354.5	3.33
	1	0	101	1	0	50.86	8.55	15.82	32	354.5	3.33
Expt. 3b	100	0	1	1	0	50.86	8.55	15.82	32	354.5	3.33
	100	0	51	1	0	50.86	8.55	15.82	32	354.5	3.33
	100	0	101	1	0	50.86	8.55	15.82	32	354.5	3.33
Expt. 3c	50	50	1	1	1	50.86	8.55	15.82	32	354.5	3.33
	50	50	51	1	1	50.86	8.55	15.82	32	354.5	3.33
	50	50	101	1	1	50.86	8.55	15.82	32	354.5	3.33
Expt. 4	50	50	50	1	1	50.86	8.55	15.82	32	354.5	3.33
	50	50	50	10	1	50.86	8.55	15.82	32	354.5	3.33
	50	50	50	20	1	50.86	8.55	15.82	32	354.5	3.33

Table B–5: Mesh Simulation Experiments – 2 of 3

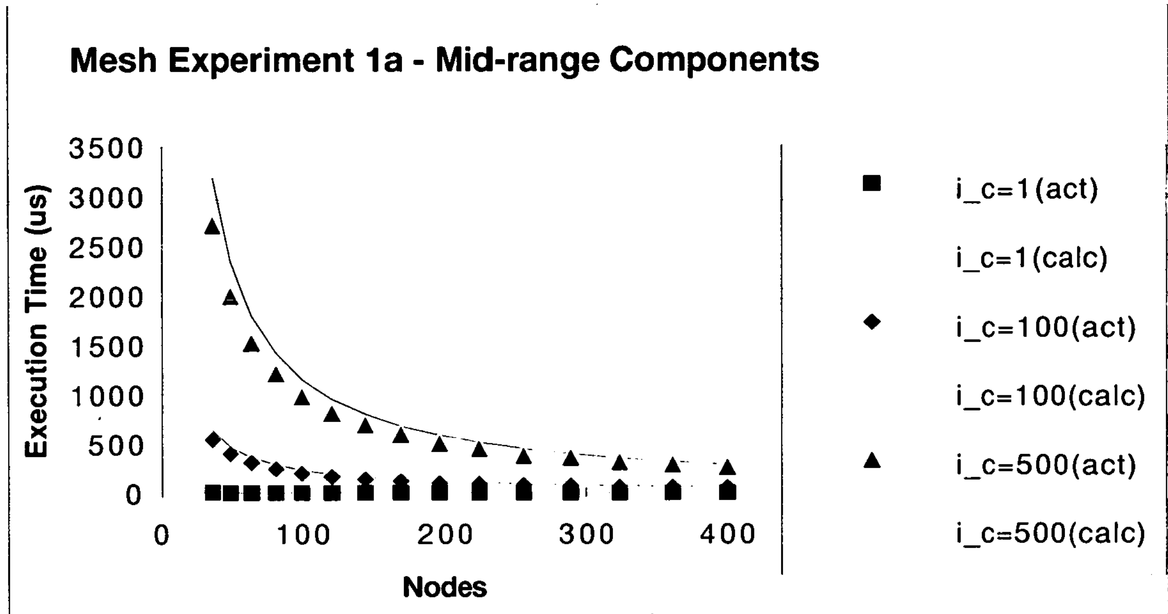


Figure B–18: Mesh Experiment 1a - Mid-range Components

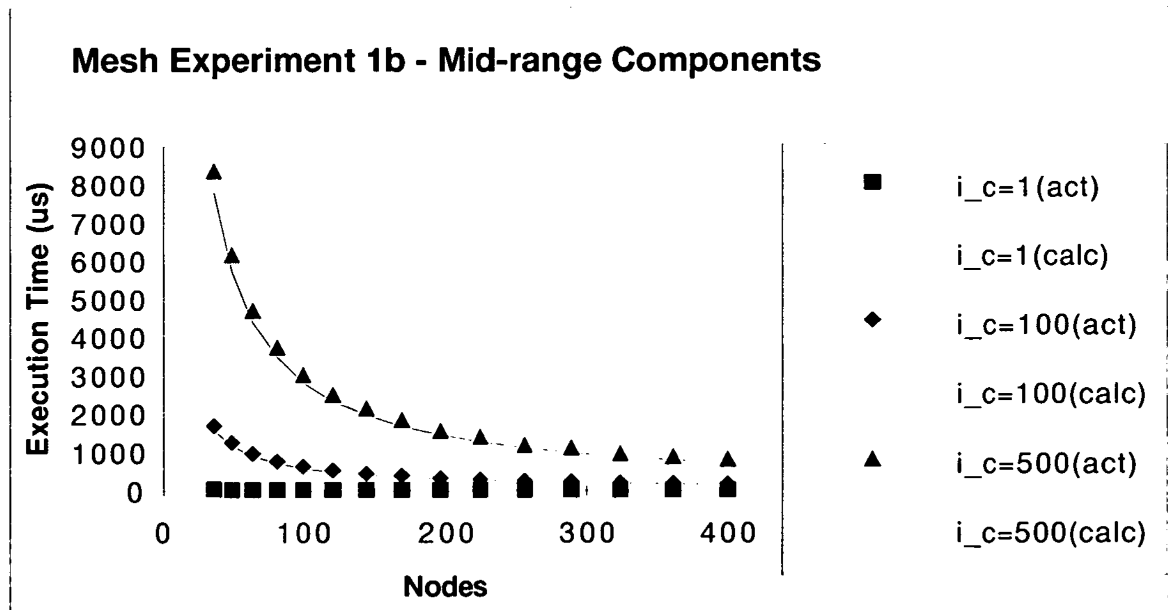


Figure B–19: Mesh Experiment 1b - Mid-range Components

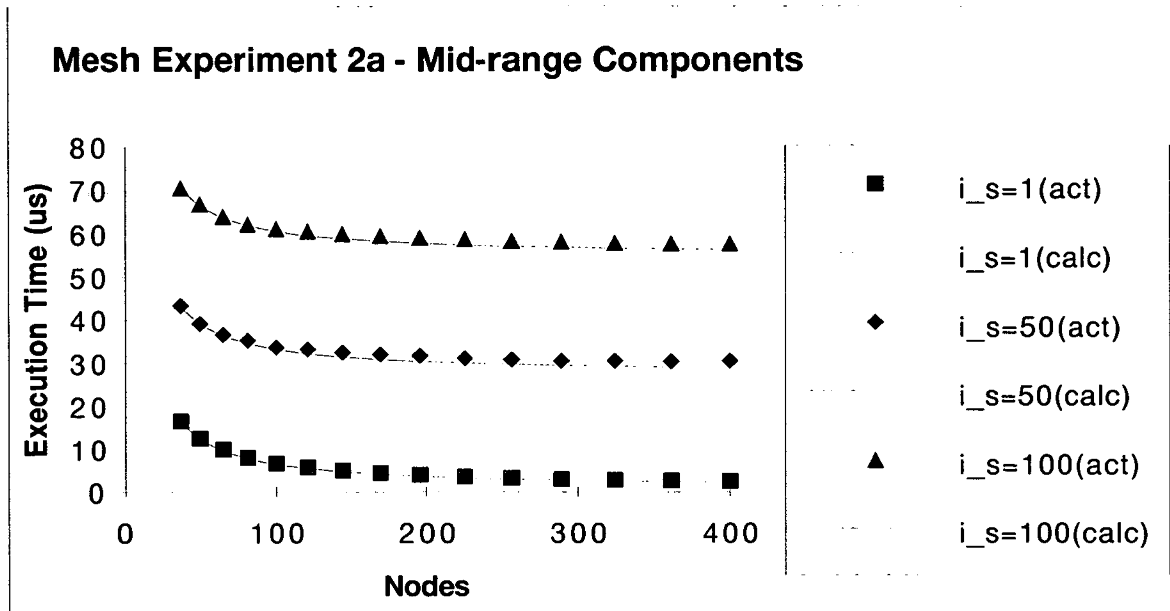


Figure B–20: Mesh Experiment 2a - Mid-range Components

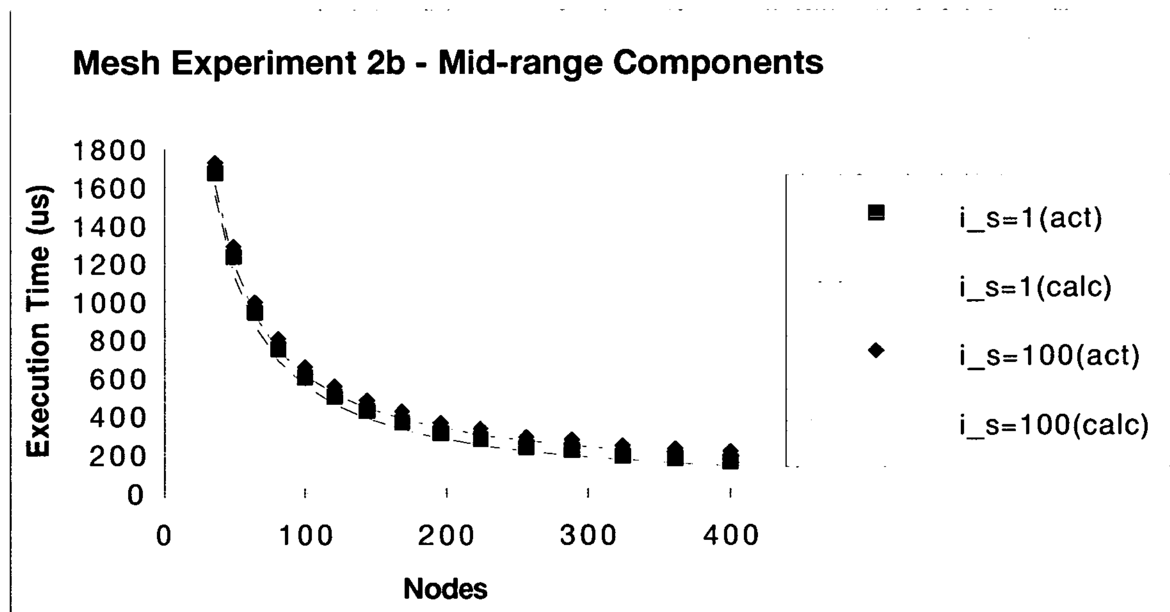


Figure B–21: Mesh Experiment 2b - Mid-range Components

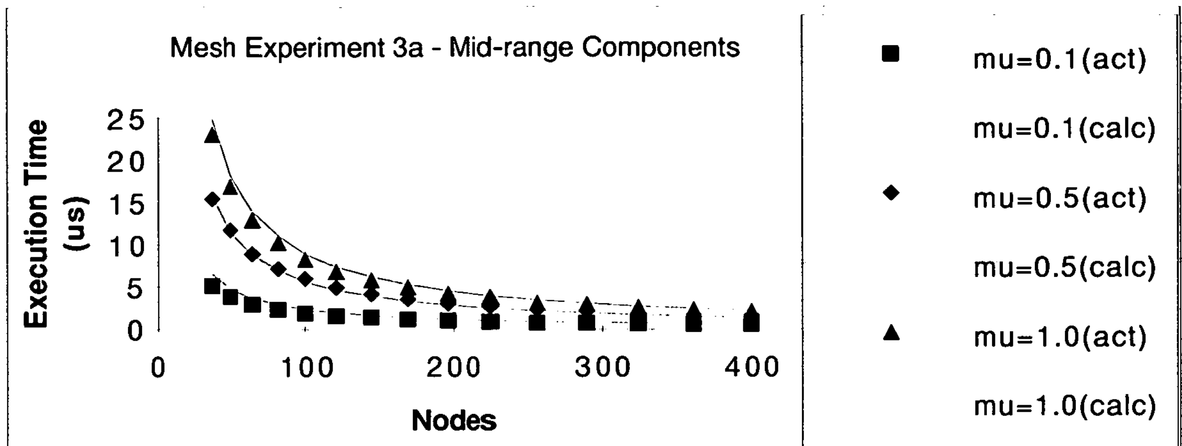


Figure B–22: Mesh Experiment 3a - Mid-range Components

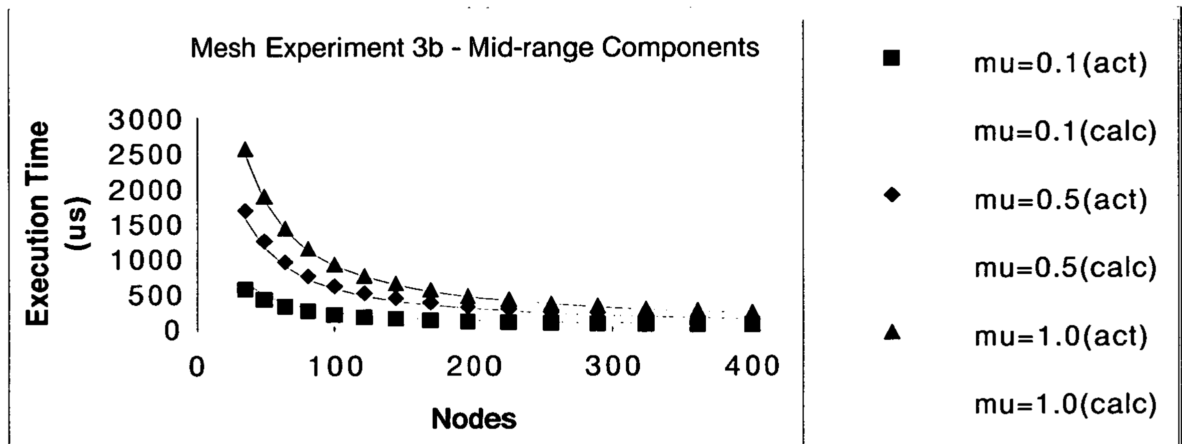


Figure B–23: Mesh Experiment 3b - Mid-range Components

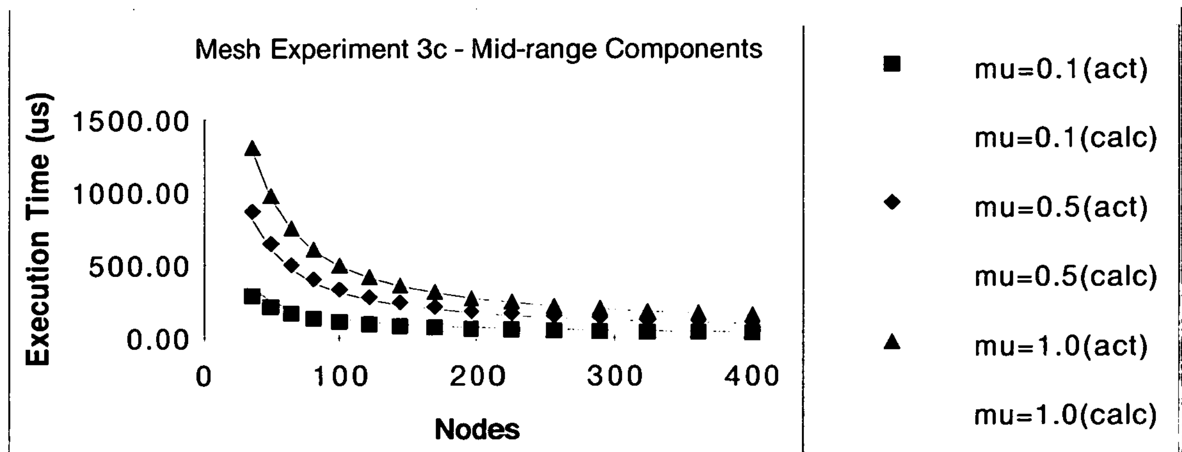


Figure B–24: Mesh Experiment 3c - Mid-range Components

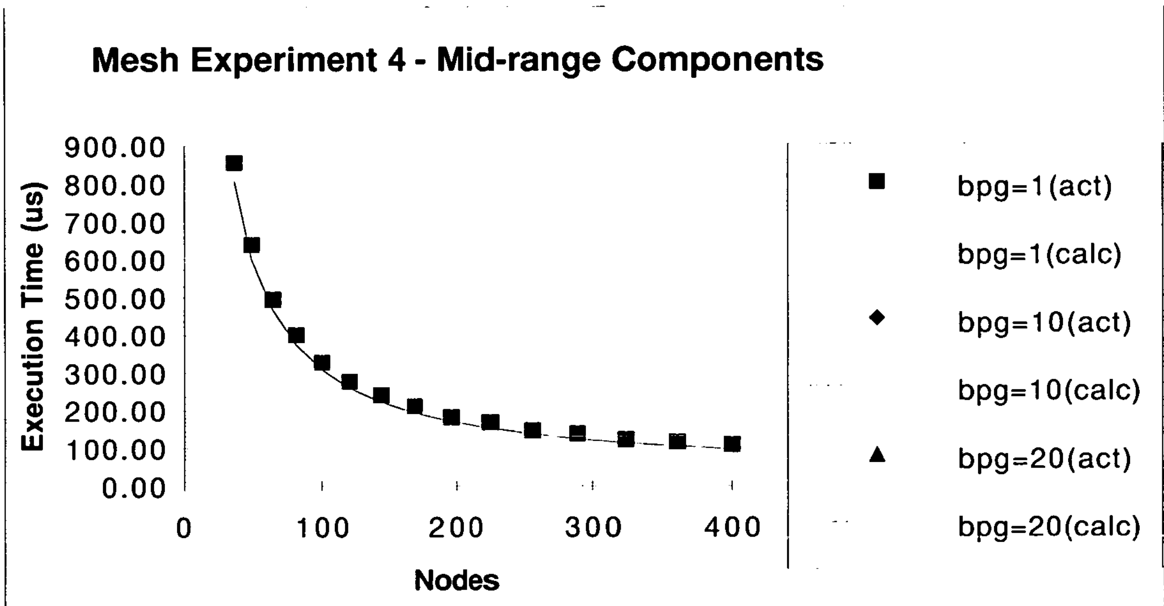


Figure B–25: Mesh Experiment 4 - Mid-range Components

High-end Components											
	i_c	i_s	μ	bpg	δ	CPU Cost	t_I	MEM Cost	t_M	SWITCH Cost	t_S
Expt. 1a	1	0	0	1	0	354.5	3.33	40.23	9	354.5	3.33
	100	0	0	1	0	354.5	3.33	40.23	9	354.5	3.33
	500	0	0	1	0	354.5	3.33	40.23	9	354.5	3.33
Expt. 1b	1	50	50	1	1	354.5	3.33	40.23	9	354.5	3.33
	100	50	50	1	1	354.5	3.33	40.23	9	354.5	3.33
	500	50	50	1	1	354.5	3.33	40.23	9	354.5	3.33
Expt. 2a	1	1	50	1	1	354.5	3.33	40.23	9	354.5	3.33
	1	50	50	1	1	354.5	3.33	40.23	9	354.5	3.33
	1	100	50	1	1	354.5	3.33	40.23	9	354.5	3.33
Expt. 2b	100	1	50	1	1	354.5	3.33	40.23	9	354.5	3.33
	100	50	50	1	1	354.5	3.33	40.23	9	354.5	3.33
	100	100	50	1	1	354.5	3.33	40.23	9	354.5	3.33
Expt. 3a	1	0	1	1	0	354.5	3.33	40.23	9	354.5	3.33
	1	0	51	1	0	354.5	3.33	40.23	9	354.5	3.33
	1	0	101	1	0	354.5	3.33	40.23	9	354.5	3.33
Expt. 3b	100	0	1	1	0	354.5	3.33	40.23	9	354.5	3.33
	100	0	51	1	0	354.5	3.33	40.23	9	354.5	3.33
	100	0	101	1	0	354.5	3.33	40.23	9	354.5	3.33
Expt. 3c	50	50	1	1	1	354.5	3.33	40.23	9	354.5	3.33
	50	50	51	1	1	354.5	3.33	40.23	9	354.5	3.33
	50	50	101	1	1	354.5	3.33	40.23	9	354.5	3.33
Expt. 4	50	50	50	1	1	354.5	3.33	40.23	9	354.5	3.33
	50	50	50	10	1	354.5	3.33	40.23	9	354.5	3.33
	50	50	50	20	1	354.5	3.33	40.23	9	354.5	3.33

Table B–6: Mesh Simulation Experiments – 3 of 3

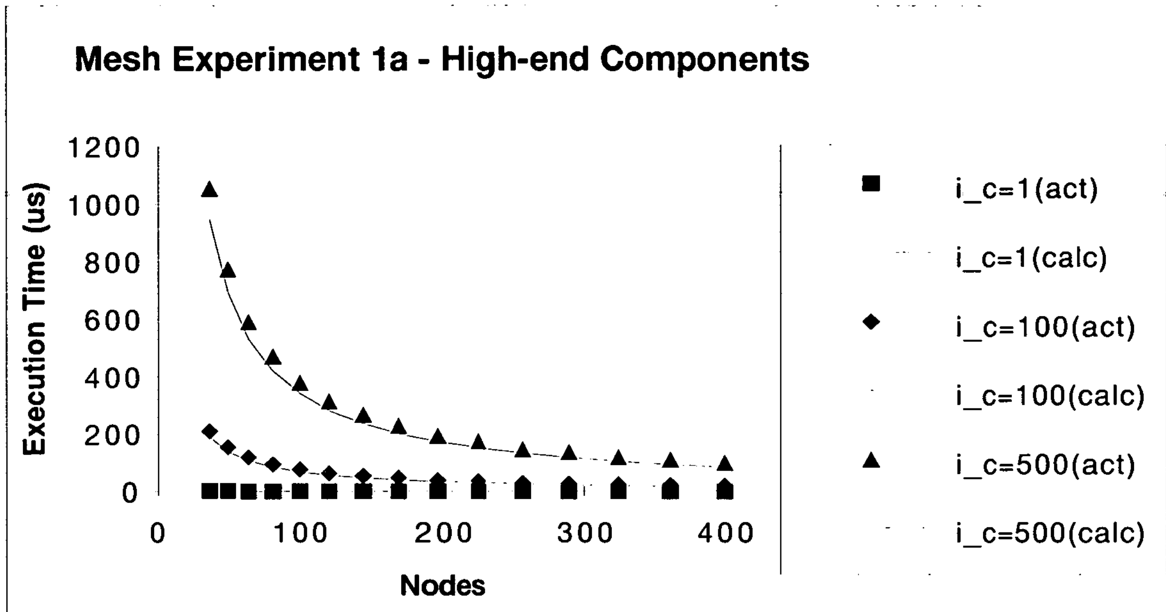


Figure B–26: Mesh Experiment 1a - High-end Components

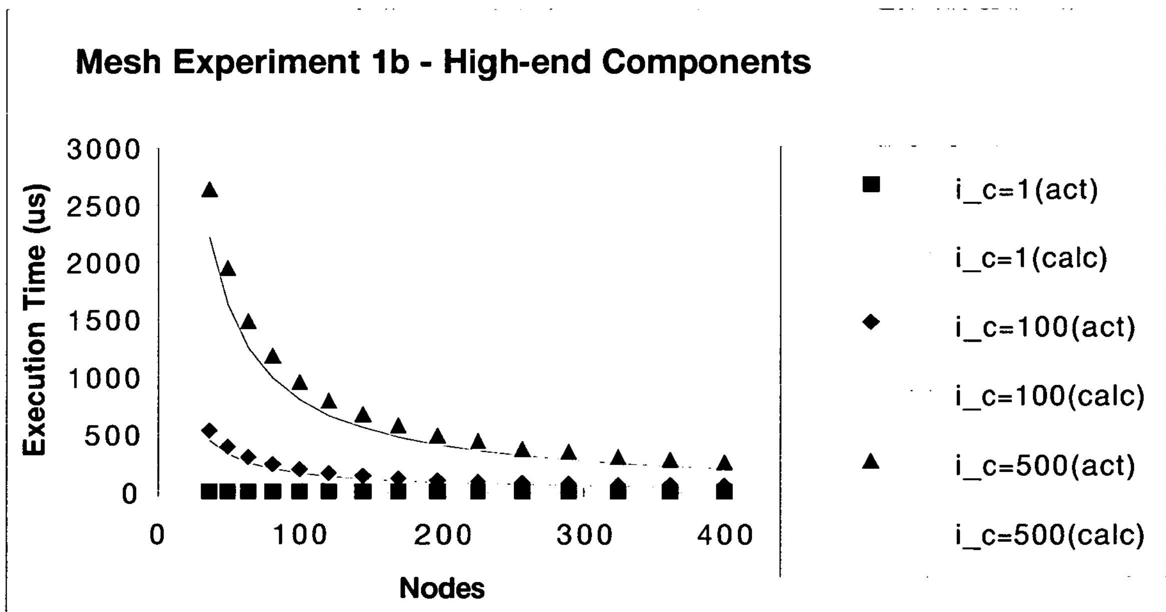


Figure B–27: Mesh Experiment 1b - High-end Components

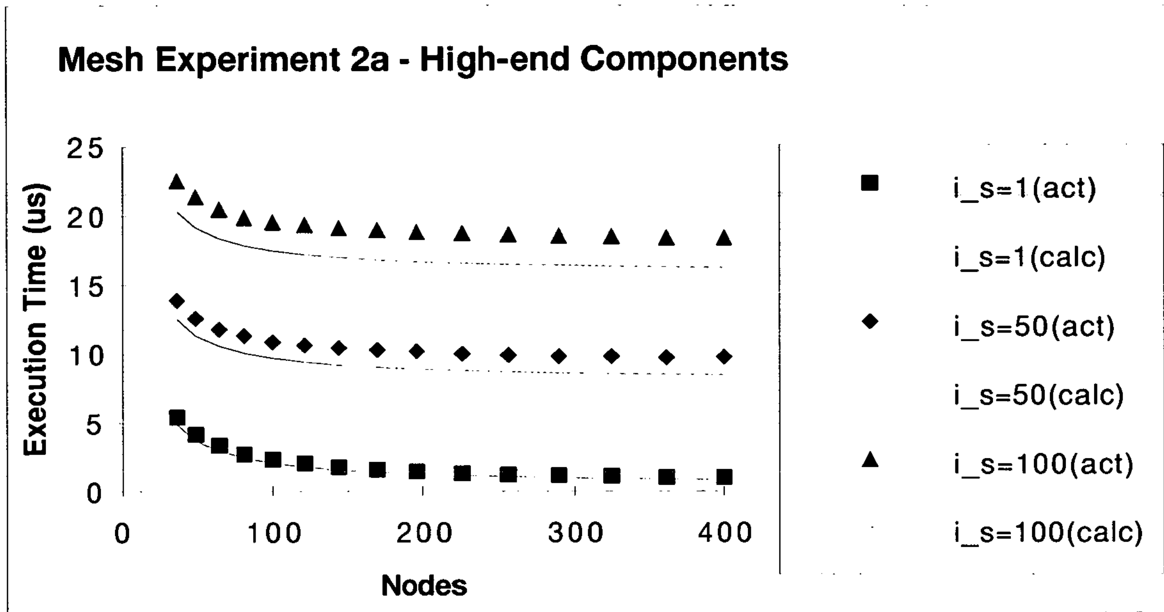


Figure B–28: Mesh Experiment 2a - High-end Components

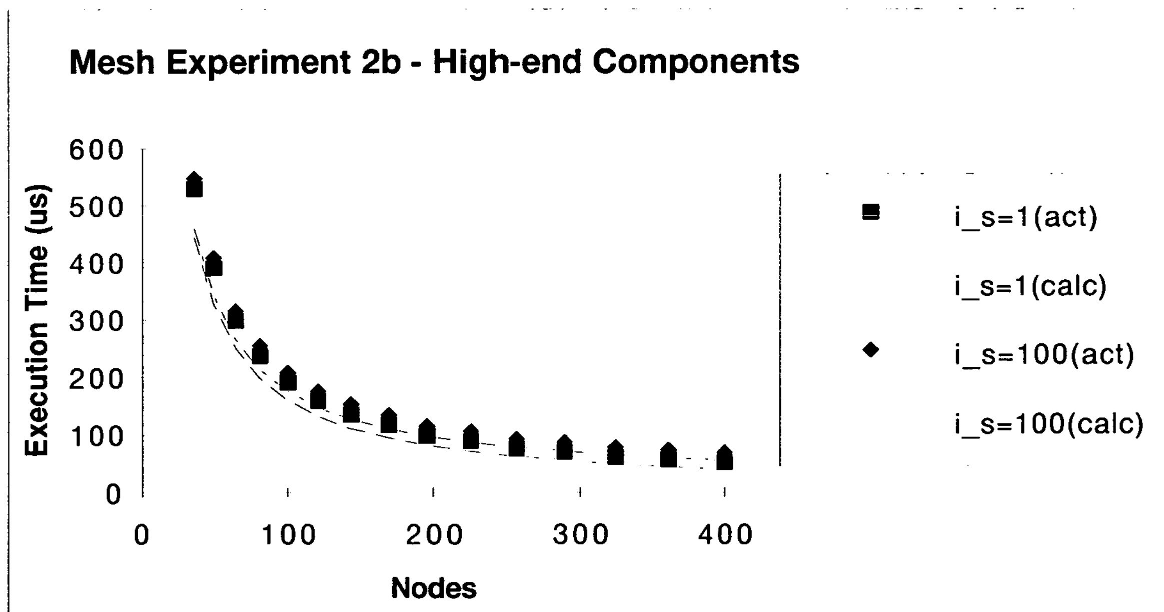


Figure B–29: Mesh Experiment 2b - High-end Components

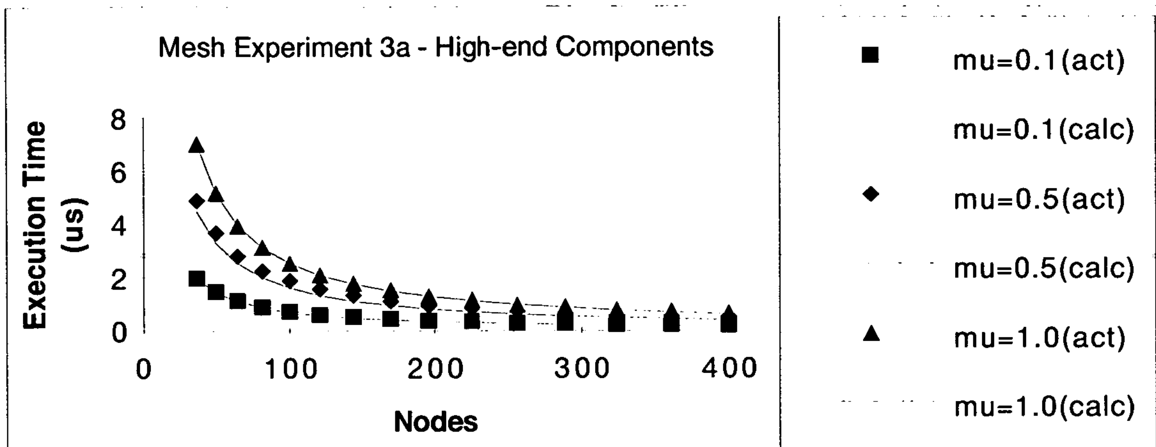


Figure B–30: Mesh Experiment 3a - High-end Components

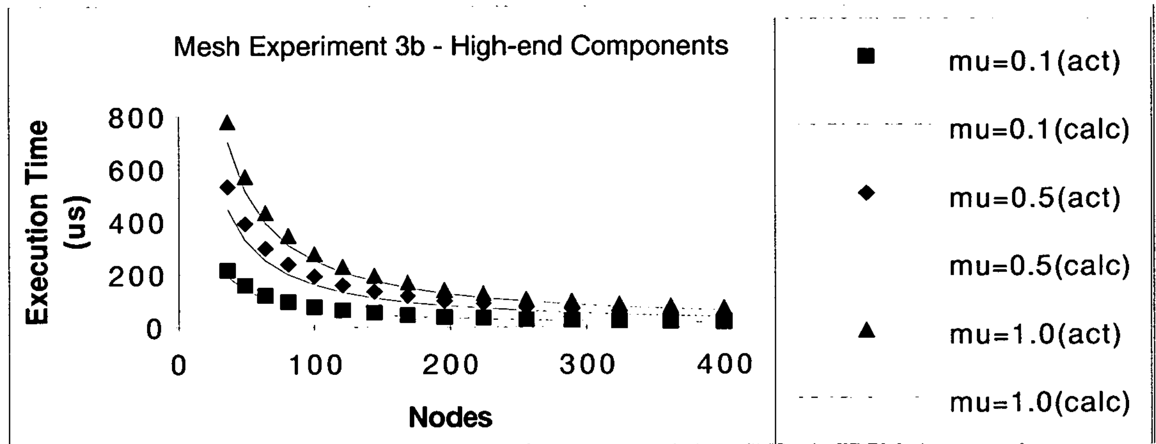


Figure B–31: Mesh Experiment 3b - High-end Components

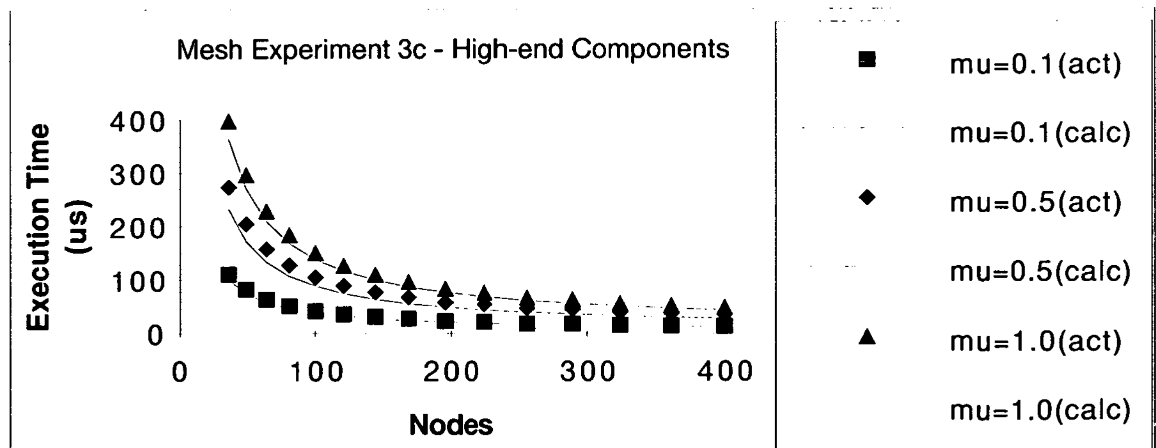


Figure B–32: Mesh Experiment 3c - High-end Components

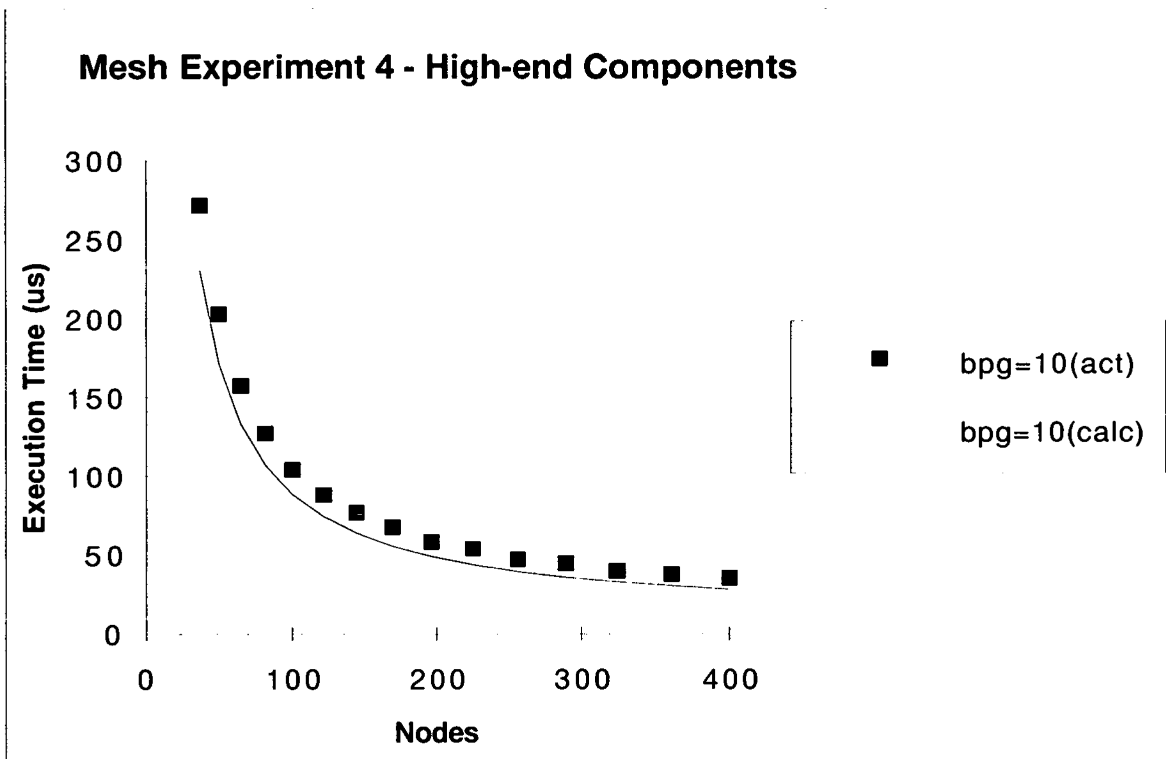


Figure B–33: Mesh Experiment 4 - High-end Components

Bibliography

[1] Miron Abramovici, Melvin Breuer, and Arthur Friedman. *Digital Systems Testing and Testable Design.* Computer Science Press, 1990.

[2] Anant Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, October 1991.

[3] Gene Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the IFIPS Spring Joint Computer Conference*, pages 483–485, 1967.

[4] M.L. Barton and G.R. Withers. Computing performance as a function of the speed, quantity and cost of the processors. *Proceedings of Supercomputing 1989*, pages 759–764, 1989.

[5] Jayaram Bhasker. *A VHDL Primer.* Prentice-Hall, 1992.

[6] Laxmi Bhuyan. Interconnection networks for parallel and distributed processing. *IEEE Computer*, pages 9–12, 1987.

[7] Graham Birtwhistle. *SIMULA BEGIN.* Lund New York, 1973.

[8] Graham Birtwhistle. *DEMOS : a system for discrete event modelling on SIMULA*. London, Macmillan, 1979.

[9] Rosemary Candlin, Peter Fisk, Joe Phillips, and Neil Skilling. A statistical approach to predicting the performance of concurrent programs. *Proceedings of EWPC'92, the European Workshops on Parallel Computing*, page 616, March 1992.

[10] Garry Christie. Private communication. Motorola Ltd., August 1994.

[11] Standard Performance Evaluation Corporation. c/o NCGA national computer graphics association. Fairfax, VA 22031, USA.

[12] David Culler, Richard Karp, and David Patterson. Logp: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, California, May 1993. ACM.

[13] William J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6):775–785, June 1990.

[14] William J. Dally. Virtual-Channel flow control. *IEEE Transactions on Parallel and Distributed Computing*, 3(2):194–205, March 1992.

[15] William J. Dally and Charles L. Seitz. The torus routing chip. *Distributed Computing*, 1(3):187–196, 1986.

[16] William J. Dally and Charles L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[17] Sivarama Dandamudi. *Hierarchical Interconnection Networks for Multicomputer Systems*. PhD thesis, University of Saskatchewan, Saskatoon, Saskatchewan, Canada, November 1988.

[18] Dataquest Europe Ltd. European MOS memory market consumption forecast 1991 - 1997. Technical Report SEMI-EU-MT-9301, Dataquest, June 1993.

[19] Daryl Doane and Paul Franzon, editors. *Multichip Module Technologies and Alternatives.* Van Nostrand Reinhold, 1993.

[20] Derek L. Eager, John Zahorjan, and Edward D. Lazowska. Speedup versus efficiency in parallel systems. *IEEE TC*, 38(3):408–423, March 1989.

[21] T. Feng. A survey of interconnection networks. *IEEE Computer*, pages 12–27, December 1981.

[22] Stuart Forbes. Private communication. Motorola Ltd., July 1994.

[23] John L. Gustafson. Reevaluating Amdahl's law. *CACM*, 31(P):532–533, May 1988.

[24] Linley Gwennap. Estimating IC manufacturing costs. *Microprocessor Report*, pages 12–16, August 1993.

[25] John Hennessy and David Patterson. *Computer Architecture. A Quantitative Approach.* Morgan Kaufman Publishers Inc, 1990.

[26] Anthony J.G. Hey. General-purpose parallel computing. In *Conference on Very Large Scale Computing in the 21st century*, Cape Cod, October 1990.

[27] Samuel Ho and Lawrence Snyder. Balance in architectural design, 1990.

[28] Kai Hwang and Faye Briggs. *Computer Architecture and Parallel Processing.* McGraw Hill, 1984.

[29] Cadence Design Systems Inc. *Verilog-XL Reference Manual.* Cadence, 1991.

[30] Motorola Inc. *MDA08 CMOS Standard Cell Data.* Motorola, 1991.

[31] Jade Simulations International. *SIM++ : a discrete event simulation language.* JSI Inc., 1988.

[32] Raj Jain. *The Art of Computer Systems Performance Analysis.* Wiley, 1991.

[33] J.E.Smith, W.C.Hsu, and C.Hsiung. Future general purpose supercomputer architectures. In *Proceedings*, pages 796–804, 1990.

[34] Christopher Frank Joerg. Design and implementation of a packet switched routing chip. Technical Report MIT/LCS/TR-482, Massachusetts Institute of Technology, MIT Lab. for Computer Science, Cambridge MA, December 1990.

[35] Hermann Jung, Lefteris Kirousis, and Paul Spirakis. Lower bounds and efficient algorithms for multiprocessor scheduling of directed acyclic graphs with communication delays. *Information and Computation*, 1(105):94–104, 1983.

[36] Thomas Kelly. How medium is "Medium"? In *IEE Colloquium on Medium Grain Distributed Computing*, The Institution of Electrical Engineers, Savoy Place, London, March 1992.

[37] Thomas Kelly and Roland Ibbett. Parallelism versus performance – matching parallel hardware to software. In G.R.Joubert, D.Trystram, and D.J.Evans, editors, *Parallel Computing: Trends and Applications*, pages 437 – 444. Elsevier Science B.V., 1994.

[38] Thomas Kelly, Lewis MacKenzie, and Mohamed Ould-Khaoua. Effects of message length, decision time and wiring density on k-ary n-cube latency. In *Permian Basin Supercomputing Conference*, University of Texas of the Permian Basin, March 1992.

[39] Parviz Kermani and Leonard Kleinrock. Virtual Cut-Through: a new computer communication switching technique. *Computer Networks 3*, pages 267–286, 1979.

[40] R.E. Kessler and J.L. Schwarzmeier. CRAY T3D: A new dimension for Cray Research. In *Digest of Papers, CompCon Spring '93*, pages 176–182, San Francisco, February 1993. IEEE, IEEE Computer Society Press.

[41] C. P. Kruskal. Searching, merging and sorting in parallel computation. *IEEE Transactions on Computers*, C-32(10):942–946, October 1983.

[42] Clyde P. Kruskal and Marc Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Transactions on Computers*, C-32(12):1091–1098, December 1983.

[43] Kumar and Reddy. Augmented shuffle-exchange multistage interconnection networks. *IEEE Computer*, pages 30–40, June 1987.

[44] Vipin Kumar and Anshul Gupta. Analyzing scalability of parallel algorithms and architectures. Computer science technical report, University of Minnesota, Minneapolis, MN – 55455, May 1991.

[45] Edward D. Lazowska. *Quantitative System Performance: Computer System Analysis Using Queuing Network Models.* Prentice-Hall, 1984.

[46] Stephen F. Lundstrom. Applications considerations in the system design of highly concurrenturrent multiprocessors. *IEEE TC*, C-36(11):1292–1309, November 1987.

[47] Lewis M. MacKenzie. *The Application of Microelectronic Technology to Physics Research.* PhD thesis, University of Glasgow, Glasgow, Scotland, November 1983.

[48] Carver Mead and Lynn Conway. *Introduction to VLSI Systems*, chapter 9. Addison-Wesley, October 1980.

[49] Sharad Mehrotra, Chien-Ming Cheng, Kai Hwang, Michel Dubois, and D.K. Panda. Algorithm-driven simulation and performance projection of a RISC-based orthogonal multiprocessor. *1990 International Conference on Parallel Processing*, 3:244–253, 1990.

[50] P.R. Miller, C.R.Jesshope, and J.T.Yantchev. The mad-postman network chip. In *Proceedings of Transputing - Volume 2*, pages 517–536, 1991.

[51] Kenichi Miura, Moriyuki Takamura, Yoshinori Sakamoto, and Shin Okada. Overview of the Fujitsu VPP500 Supercomputer. In *Digest of Papers, CompCon Spring '93*, pages 128–130, San Francisco, February 1993. IEEE, IEEE Computer Society Press.

[52] Motorola. *Memory Device Data - DL113 Rev 7.* Motorola Inc., 1991.

[53] M.G. Norman and P. Thanisch. Models of machines and computations for mapping in multicomputers. *ACM Computing Surveys*, 25(3), 1993.

[54] Daniel Nussbaum and Anant Agarwal. Scalability of parallel machines. *CACM*, 34(3):56–61, March 1991.

[55] Mohamed Ould-Khaou, Lewis MacKenzie, and Robert Sutherland. Performance of switching methods in cobra networks. Departmental Research Report AH-93-01, University of Glasgow, Dept. of Computing Science, Glasgow G12 8QQ, Scotland, United Kingdom., November 1993.

[56] Mohamed Ould-Khaoua. *Hypergraph-based Interconnection Networks for Large Multicomputers.* PhD thesis, University of Glasgow, Glasgow, Scotland, February 1994.

[57] Christos H. Papadimitriou and Mihalis Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM Journal of Computing*, 19(2):322–328, April 1990.

[58] Simon Peyton-Jones. *The implementation of functional programming languages*. Prentice-Hall, 1987.

[59] Giacomo Polosa. Technical forum. Supercomputing Europe '93, February 1993.

[60] Michael J. Quinn. *Designing Efficient Algorithms for Parallel Computers*, chapter 5, pages 108–109. McGraw-Hill, 1987.

[61] Daniel A. Reed. Queueing network models of multicomputer networks. In *Proceedings of 1983 ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, pages 190 — 197, Minneapolis, Minnesota, 1983. ACM.

[62] R.W.Hockey and C.R.Jesshope. *Parallel Computers 2*. Adam & Hilger. ISBN 0-85274-812-4, 1988.

[63] John Sanguinetti and Peter Eichenberger. *VCS2 User's Guide*. Chronologic Simulation, 1994.

[64] Steven L. Scott and James R. Goodman. *Performance of Pipelined-Channel k-ary n-cube Networks*, 1900.

[65] Charles L. Seitz. The cosmic cube. *CACM*, 28(1):22–33, 1985.

[66] Marc Snir. Scalable parallel computers and scalable parallel codes: From theory to practice. In *Heinz Nixdorf Symposium*, pages 19–27, 1992.

[67] Karl Solchenbach and Ulrich Trottenberg. Suprenum: System essentials and grid applications. *Parallel Computing*, 7:265–281, 1988.

[68] Xian-He Sun and John L. Gustafson. Sizeup: A new parallel performance metric. *1991 International Conference on Parallel Processing*, 2:298–299, 1991.

[69] Xian-He Sun and Lionel M. Ni. Another view on parallel speedup. *IEEE ?*, pages 324–333, 1990.

[70] M. Swamy and K. Thulasiram. *Graphs, Networks and Algorithms.* John Wiley and Sons, 1981.

[71] Donald Thomas and Philip Moorby. *The Verilog Hardware Description Language.* Kluwer Academic Publishers, 1991.

[72] Walid Touma. *The Dynamics of the Computer Industry.* Kluwer Academic Publishers, 1993.

[73] Arthur Trew and Greg Wilson. *Past, Present, Parallel – A Survey of Available Parallel Computing Systems.* Springer-Verlag, 1991.

[74] Leslie G. Valiant. A bridging model for parallel computation. *CACM*, 33(2):103–111, August 1990.

[75] Frederic A. Van-Catledge. Towards a general model for evaluating the relative performance of computer systems. *International Journal of Supercomputer Applications*, 3(2):100–108, Summer 1989.

[76] Peter vanZant. *A Practical Guide to Semiconductor Processing.* McGraw-Hill, 1990.

[77] Simon Young. Private communication. Motorola Ltd., July 1994.

[78] Hans Zima and Barbara Chapman. Compiling for distributed-memory systems. Technical Report ACPC/TR 92-17, University of Vienna, Austrian Centre for Parallel Computation, University of Vienna, November 1992.