School of Computing

Design, Development and Evaluation of an Efficient Hierarchical Interconnection Network

Stuart Malcolm Campbell

This thesis is presented as part of the requirements for the award of the Degree
of Doctor of Philosophy
of the
Curtin University of Technology

July, 1999

Acknowledgements

I would like to thank my supervisor, Dr Mohan Kumar, for his guidance and his unfailing patience, persistence and simple availability over the years.

I would also like to thank Dr Bill Smythe and Dr Arun Somani for their reviews and criticisms of drafts of this thesis; such style and clarity as it has owes much to them. To Professors Horst Bunke and Terry Caelli I owe the desire and opportunity to investigate parallel graph algorithms, a fascinating field.

I gratefully acknowledge the support of the School of Computing at Curtin University and of the University itself, the research described in this dissertation was funded by a Curtin University Postgraduate Scholarship. I wish to thank the many staff and students at Curtin whose presence made these years as enjoyable as they were educational. I particularly wish to thank my father, Malcolm Campbell, who helped me find a way to do this.

Finally I must express my deepest debt of gratitude to my wife Linda, without whose faith, hand patting and long suffering support this thesis would not have been possible.

Abstract

Parallel computing has long been an area of research interest because exploiting parallelism in difficult problems has promised to deliver orders of magnitude speedups. Processors are now both powerful and cheap, so that systems incorporating tens, hundreds or even thousands of powerful processors need not be prohibitively expensive. The weak link in exploiting parallelism is the means of communication between the processors. Shared memory systems are fundamentally limited in the number of processors they can utilise. To achieve high levels of parallelism it is still necessary to use distributed memory and some form of interconnection network. But interconnection networks can be costly, slow, difficult to build and expand, vulnerable to faults and limited in the range of problems they can be used to solve effectively. As a result there has been extensive research into developing interconnection networks which overcome some or all of these difficulties. In this thesis it is argued that a new interconnection network, Hierarchical Cliques (HiC), and a derivative, $FatHiC$, possesses many desirable properties and are worthy of consideration for use in building parallel computers. A fundamental element of an interconnection network is its topology. After defining the topology of HiC, expressions are derived for the various parameters which define its underlying limits of performance and fault tolerance. A second element of an interconnection network is an addressing and routing scheme. The addressing scheme and routing algorithms of HiC are described. The flexibility of HiC is demonstrated by developing embeddings of popular, regular interconnection networks. Some embeddings into HiC suffer from high congestion, however the $FatHiC$ network is shown to have low congestion for those embeddings. The performance of some important, regular, data parallel problems on HiC and $FatHiC$ are determined by analysis and simulation, using the $2D$-mesh as a means of comparison. But performance alone does not tell the whole story. Any parallel computer system must be cost effective. In order to analyse the cost effectiveness of HiCs an existing measure was expanded to provide a more realistic model and a more accurate means of comparison. One aim of this thesis is to demonstrate the suitability of HiC for parallel computing systems which execute irregular algorithms requiring dynamic load balancing. A new dynamic load balancing algorithm is proposed which takes advantage of the hierarchical structure of the HiC to reduce communication overheads incurred when distributing work. To demonstrate performance of an irregular problem, a novel parallel algorithm was developed to detect

subgraph isomorphism from many model graphs to a single input graph. The use of the new load balancing algorithm in conjunction with the subgraph isomorphism algorithm is discussed.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Parallel computing has existed as an active field of research since the earliest days of computing. In general, parallelism is used either to increase performance in a cost effective way, or to obtain performance levels which are not available from a single processor. Several early parallel computers used tens or hundreds of processors. These machines were generally intended to be used in solving large scale scientific or engineering problems [Stone(1987), Hockney and Jesshope(1988)]. More recently, massively parallel machines using thousands of processors have been built, and used for solving a variety of problems with high computational complexity or involving huge data sets [Hwang(1993)]. Currently, symmetric multiprocessors and clusters of workstations, each system generally involving less than a hundred processors, are allowing parallel processing to be applied to an incredible variety of problems from every avenue of science, engineering, art and commerce.

Nearly all parallel algorithms require interprocessor communication. Communication is costly in two ways. Firstly, in the cost of hardware and software required to construct and operate the interconnection network. This forms a major part of the cost of most parallel systems. Secondly, in reduced performance of the parallel computer, caused by the delays inherent in any communication system. It is clearly desirable to reduce both the monetary cost of the interconnection network and the performance losses caused by its operation, however these are often mutually contradictory goals. Attempting to

balance the competing requirements of high performance and low cost has meant that the design of interconnection networks has been an area of continuing research interest.

This thesis proposes two new interconnection networks, the Hierarchical Cliques (HiC) network, and a derivative, the Fat Hierarchical Cliques ($FatHiC$) network. The major goal of this thesis is to establish the suitability of these networks for use in parallel computing. This will be achieved by detailed analysis of fundamental network properties such as diameter and average inter-PE distance, by the determination of measures of fault-tolerance such as fault diameter and average two terminal reliability, and by the analysis of embedding quality for embeddings of important topologies. Network performance will be demonstrated by studies of the network throughput and latency, and by comparison of results of studies of algorithm performance, using analysis and simulation, with appropriate existing networks. The networks will be shown to be cost-effective using a modified version of an analysis technique which allows the total cost of an interconnection network to be combined with the efficiency of an algorithm, on that interconnection network, into a single expression for the cost effectiveness of the interconnection network when executing that algorithm.

A minor goal of this thesis is to establish the suitability of the HiC and $FatHiC$ interconnection networks for parallel systems solving irregular problems. A novel, receiver initiated, dynamic load balancing algorithm will be presented which relies on the hierarchical structure of the HiC. It is simple, scalable and takes advantage of locality of data to reduce communication overhead and improve performance. The scalability of the load balancing algorithm will be shown to be superior to that of several existing load balancing algorithms on the hypercube. Finally, an irregular algorithm will be described as an example of the use of the new load balancing algorithm. The irregular, parallel algorithm is a new method for detecting subgraph isomorphisms.

1.1 Properties of interconnection networks

An interconnection network can be defined in terms of its flow control, routing and topology [Dally(1990a)].

A number of techniques for flow control have been studied and used in parallel computer systems. In general though, a given flow control technique can be implemented with any combination of topology and routing algorithm. The simple, though inefficient, method of store and forward is assumed in this thesis. More sophisticated flow control techniques could be applied for better performance.

Routing methods, including an addressing scheme, are usually closely tied to a particular topology. To ensure high performance, routing must be efficient and fast. Routing algorithms must ensure that conflict in demand for network resources does not result in deadlock.

The topology of a network is the underlying pattern of the physical communication links between processors. Numerous topologies have been proposed, studied and used. The topology of a network will influence, among other things; the latency of messages, the network fault tolerance, the difficulty and cost of implementing the network, the ease with which new and existing algorithms are implemented, and the efficiency of those algorithms.

Once an interconnection network has been defined, its potential performance and cost can be analysed to determine its suitability for use in a parallel computer. Both performance and cost are influenced by the communication, switching, processing and programming technology of the times. For example, in [Dally(1990b)] it was shown that "low-dimensional networks (e.g., tori) have lower latency and higher hot-spot throughput than high-dimensional networks (e.g., binary n-cubes) with the same bisection width." While widely influential, the results were derived for a massively parallel system employing a VLSI communication network. This is no longer the dominant paradigm in parallel computing. Measures of performance and cost must be abstract in order to avoid becoming obsolete when technology changes.

Performance is also heavily influenced by the communication patterns which occur. These are determined by the algorithms which are to be implemented upon the parallel computer. A parallel computer may offer excellent performance for one algorithm or class of algorithms, and very poor performance for another. This variability in

performance must be taken into account when considering the cost of a machine, as high cost may be justified by high performance. Thus the concept of cost effectiveness occurs. Performance and cost effectiveness are truly meaningful only for a specific parallel system, comprising a parallel computer and an algorithm implemented upon it.

The majority of algorithms which have been successfully implemented on parallel computers have a very regular structure. Traditional interconnection network topologies, such as meshes and binary hypercubes, provide a communication structure to which many regular algorithms map naturally and efficiently. As the range of problems which parallel computers are being used to solve increases, more algorithms must be developed which can be implemented efficiently on parallel architectures. Many of these algorithms exhibit unstructured and unpredictable computation times and communication patterns. These are irregular algorithms. Recently, considerable effort has been devoted to developing algorithms which alleviate the poor match between many network topologies and the requirements of unstructured algorithms. A complementary approach is to use a network with a topology more suited to the communication requirements of unstructured algorithms. A minor goal of this thesis is to establish the suitability of the *HiC* interconnection network for parallel systems solving irregular problems.

1.2 Structure of the Thesis

Chapter 2 provides necessary background information on the terms and concepts used in the thesis. Interprocessor communication networks are commonly and conveniently described in graph theoretic terms. The aspects of graph theory which will be used in this thesis are explained and necessary terms are defined. Some fundamental concepts of network fault tolerance analysis are introduced. The process of evaluating the suitability of an interconnection network for use in a parallel computer must involve comparison with other interconnection networks. Several important interconnection networks are described. The class of irregular problems is defined.

Chapter 3 describes the topology and addressing scheme of HiCs in detail. Two topological parameters which are generally accepted indicators of performance are derived; network diameter and average interprocessor distance. The fault-tolerance of k-ary trees is poor, failure of a single node or link can disconnect the system. The extra links in HiCs result in improved fault tolerance. Fault tolerance of HiCs is analysed by determining the connectivity, probability of disconnection, fault-diameter, two-terminal reliability and average two-terminal reliability of HiCs. The fault-diameter of HiCs is shown to be superior to that of binary hypercubes. Due to the fixed degree of the HiC, however, the average two-terminal reliability is inferior to that of the binary hypercube. Routing algorithms for HiCs are described which are efficient and easy to implement.

The ability of HiCs to embed other major interconnection schemes is an important indicator of its ability to provide efficient communication for a wide variety of algorithms. Embeddings for binary trees, binary hypercubes and two dimensional meshes are described. Owing to the tree based nature of the HiC, edge congestion is quite high for these embeddings. The $FatHiC$ is a derivative of the HiC, based upon the fat-trees of [Leiserson(1985)]. Embeddings into HiC are shown to be effective in $FatHiC$, but with improved edge congestion. Because of topological similarities, the HiC and $FatHiC$ are compared with the generalised fat tree (GFT) when discussing the quality of embeddings. The embeddings are found to be comparable in dilation and expansion. The $FatHiC$ has edge congestion somewhat superior to that of the GFT, at significantly less cost.

Chapter 4 broadens the analysis of the performance capabilities of the HiC. Two fundamental measures of network performance are throughput and latency. Simulation studies are used to examine the throughput and latency of HiCs under various traffic conditions. The network is found to be optimal until saturation, and stable at all times. Performance of a particular algorithm on a parallel computer can be expressed in terms of speedup, efficiency and iso-efficiency. Floyd's algorithm and *ascend/descend* algorithms were chosen as representative regular algorithms. Analysis of these two algorithms reveals that performance on the HiC is equivalent to that of a $2D$-mesh for Floyd's algorithm, but suffers from congestion at the higher levels of the hierarchy

for *ascend/descend* algorithms. Simulation studies are used to verify the results. In Chapter 3 the $FatHiC$s were shown to overcome the problem of congestion in mapping mesh and n-cube based data sets. Analysis and simulation are used to confirm the expected increase in performance, with the $FatHiC$ showing performance superior to both the HiC and $2D$-mesh.

Chapter 5 presents an analysis of the cost and cost effectiveness of the HiC and $FatHiC$ interconnection networks. Cost is determined using a version of *Cost Effectiveness Factor*, as introduced in [Sarkar(1993)], modified to include the cost of switches. The cost effectiveness factors of HiC and $FatHiC$ are compared with those for the d-dimensional mesh and GFT. Both HiCs and $FatHiC$s are shown to have a superior cost effectiveness factor to meshes of dimension 3 or higher. An HiC with cliques of size four is found to have a cost effectiveness factor very similar to that of the best GFT considered. Having determined cost effectiveness factor, the cost effectiveness of HiCs and $FatHiC$s is derived for the algorithms whose performance was studied in Chapter 4. For Floyd's algorithm the HiC is more cost effective than either the $FatHiC$ or the GFT, though the $2D$-mesh is the most cost effective. In the case of *ascend/descend* algorithms, however, the $FatHiC$ is the most cost effective network, superior to the GFT, $2D$-mesh and HiC.

Chapter 6 discusses the suitability of the HiC interconnection network for parallel computers implementing irregular algorithms. Probably the most fundamental problem in implementing irregular algorithms is load balancing; spreading work evenly among the processors without introducing inordinate overheads. A new, neighbourhood based, receiver initiated, dynamic load distribution algorithm is described. The scalability of the algorithm is analysed and compared with existing algorithms. In order to demonstrate the application of the load balancing algorithm, a novel application algorithm will be introduced, the Parallel Network (PN) algorithm; a parallel, deterministic algorithm for finding subgraph isomorphisms from a database of attributed, directed model graphs to an attributed, directed input graph. The algorithm distributes data amongst processors so that, as work is generated, it is queued at the processor which generates it. The load balancing algorithm then efficiently distributes work from queues to idle

processors.

1.3 Contributions of this Thesis

This thesis introduces two new interconnection networks, the HiC and the $FatHiC$. The suitability of both interconnection networks for use in parallel computing will be demonstrated through detailed analysis of network properties such as diameter, average inter-PE distance, fault-tolerance, performance and cost-effectiveness; and by comparison of analytic and simulation results with appropriate existing networks. The $FatHiC$ network will be shown to provide cost effective communication for regular algorithms which require extensive global communication. The HiC will be shown to be less costly, and suitable for problems which have dense local communication. A novel subgraph isomorphism detection algorithm is used, in conjunction with a new load balancing algorithm, to demonstrate the suitability of HiCs in implementing irregular algorithms.

Chapter 2

Background

The purpose of this Chapter is to introduce underlying concepts and define basic terms and symbols which will be used in the course of the thesis. The main body of the thesis is devoted to the description and development of a multiprocessor interconnection network based upon a new topology. Topology, from the Greek *topos* meaning "place", refers to the pattern of physical connections between processors. Multiprocessor interconnection network topologies are commonly and conveniently described in graph theoretic terms. Additionally, several properties of graphs are commonly used as indicators of suitability of topologies for use in interconnection networks. Section 2.1 contains definitions of graphs and digraphs, as well as various basic properties of graphs which will be used to describe and compare interconnection network topologies.

Expressions for fault-tolerance of multiprocessor interconnection networks are generally based upon graph theory. However, graph theoretic ideas are often used in concert with other concepts, such as probability, to provide measures for fault-tolerance which allow for the unpredictable nature of component failures. Section 2.2 provides an introduction to the graph theoretic and probabilistic fundamentals of fault-tolerance analysis in interconnection networks for parallel computers.

There is no single, universal measure of 'goodness' which can be used to judge parallel interconnection networks. All networks have strengths and weaknesses which must

be considered when deciding which network to implement. As a result new networks cannot be considered in isolation, rather they must be compared with existing networks before their worth can be ascertained. Section 2.3 describes a small number of widely studied and used networks which will form the basis for comparison with *HiC*s.

Finally, the purpose of all computer systems is to solve problems, and being able to provide efficient communication when solving a large number of problems is the ultimate aim of any parallel interconnection network. Some parallel interconnection networks have been designed to be, or have proven, particularly suitable for solving particular classes of problems. Section 2.4 describes irregular problems; a large class of problems. The *HiC*s will be shown to be well suited for execution of irregular problems.

2.1 Graph Theory

Many interconnection networks, algorithms and data structures are conveniently described in graph theoretic terms. This section introduces terminology that will later be used to describe *HiC*s and other interconnection networks, as well as algorithms used to demonstrate the advantages of *HiC*s. Though there is some degree of uniformity in the definitions and notations found in the graph literature, by no means is there a universally accepted standard. The definitions and notations used here generally follow those in [Bondy and Murty(1976)] or [Buckley and Harary(1990)].

A graph G is an ordered triple $(V(G), E(G), \Phi_G)$ consisting of a nonempty set $V(G)$ of *vertices*, a set $E(G)$, disjoint from $V(G)$, of *edges*, and an *incidence function* Φ_G that associates with each edge of G an unordered pair of (not necessarily distinct) vertices of G. The number of vertices in G is denoted by $|V(G)|$, and the number of edges in G by $|E(G)|$. An example of a graph is:

$$H = (V(H), E(H), \Phi_H)$$

where

$$V(H) = v_1, v_2, v_3, v_4$$
$$E(H) = e_1, e_2, e_3, e_4, e_5, e_6$$

and Φ_H is defined by

$$\Phi_H(e_1) = v_1v_2, \Phi_H(e_2) = v_2v_3, \Phi_H(e_3) = v_3v_4$$

$$\Phi_H(e_4) = v_4v_1, \Phi_H(e_5) = v_3v_1, \Phi_H(e_6) = v_4v_2$$

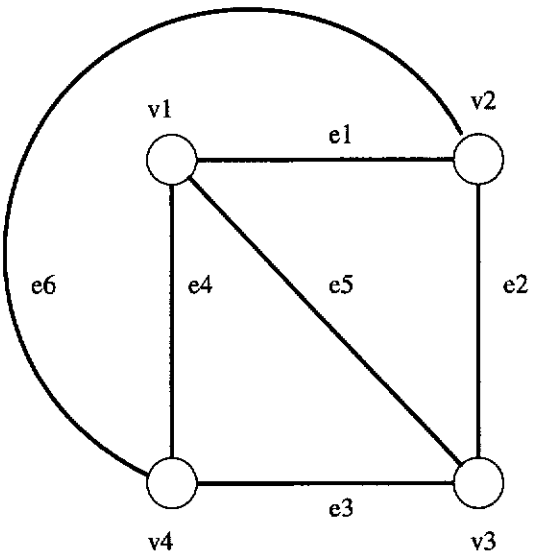


Figure 2.1: Diagram of graph H

A directed graph or digraph G is an ordered triple $(V(G), E(G), \Phi_G)$ consisting of a nonempty set $V(G)$ of *vertices*, a set $E(G)$, disjoint from $V(G)$, of *edges*, and an *incidence function* Φ_G that associates with each edge of G an ordered pair of (not necessarily distinct) vertices of G. Thus a digraph differs from a graph in that edges incorporate an additional property, that of direction. In general, all of the concepts and properties of graphs apply equally to digraphs, however, concepts which involve the notion of orientation apply only to digraphs. An example of a digraph is:

$$H = (V(H), E(H), \Phi_H)$$

where

$$V(H) = v_1, v_2, v_3, v_4$$
$$E(H) = e_1, e_2, e_3, e_4, e_5, e_6$$

and Φ_H is defined by

$$\Phi_H(e_1) = v_1v_2, \Phi_H(e_2) = v_3v_2, \Phi_H(e_3) = v_4v_3$$

$$\Phi_H(e_4) = v_4v_1, \Phi_H(e_5) = v_1v_3, \Phi_H(e_6) = v_2v_4$$

For a digraph D, if e is an edge and μ and ν are vertices such that $\Phi_D(e) = (\mu, \nu)$, then e is said to *join* μ to ν; μ is the *tail* of e and ν is the *head*. We may say that $\mathbf{tail}(e) = \mu$ and $\mathbf{head}(e) = \nu$.

An attributed graph G is an ordered quadruple $(V(G), E(G), \Phi_G, \alpha_G)$ where α_G is a mapping function that associates with each vertex of $V(G)$ an attribute from a set of *attributes* A. The attribute of a vertex μ is represented as $\alpha_G(\mu)$.

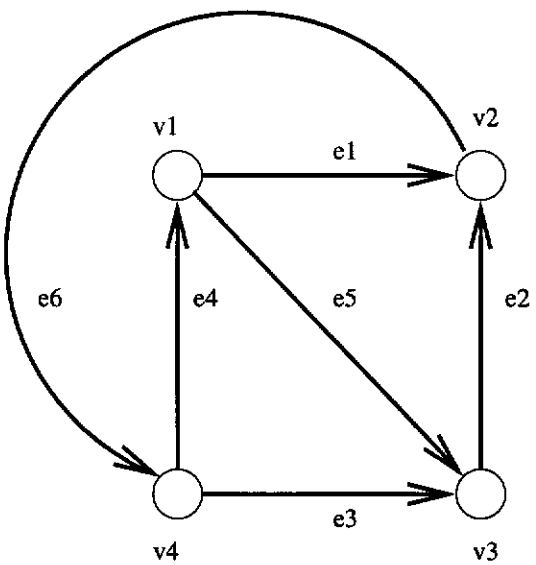


Figure 2.2: Diagram of digraph H

For a graph G, if e is an edge and μ and ν are vertices such that $\Phi(e) = \mu\nu$, then e is said to *join* μ and ν; the vertices μ and ν are called the *ends* of e. The ends of an

edge are said to be *incident* with the edge, and an edge is *incident* with its ends. Two vertices which are incident with a common edge are *adjacent*. An edge with identical ends is called a *loop*. A graph is *simple* if it contains no loops and no two of its edges join the same pair of vertices.

When just one graph is under discussion it will usually be denoted by G. In this case symbols such as $V(G)$ may be written simply as V, omitting the G which served only to identify which graph the symbol was referring to. Also, when dealing with simple graphs, it is often convenient to refer to the edge with ends μ and ν as 'the edge $\mu\nu$'. This results in no ambiguity since, in a simple graph, at most one edge joins any pair of vertices. When describing interconnection networks the convention in the literature is to call vertices *nodes* and edges *links*.

A graph H is a *subgraph* of G (written $H \subseteq G$) if $V(H) \subseteq V(G), E(H) \subseteq E(G)$ and Φ_H is the restriction of Φ_G to $E(H)$. Figure 2.3 shows a graph H which is a subgraph of graph G.

Suppose that V' is a nonempty subset of V. The subgraph of G whose vertex set is V' and whose edge set is the set of those edges of G that have both ends in V' is called the subgraph of G *induced* by V' and is denoted by $G[V']$; the subgraph $G[V']$ is an *induced subgraph* of G. The induced subgraph obtained from G by deleting the vertices in V' together with their incident edges is denoted by $G - V'$. In Figure 2.3, H is not an induced subgraph of G since $E(H)$ does not contain e_1, e_3, e_5, e_6.

If G is a simple graph, then a *clique* of G is a subset V' of V such that $G[V']$ is fully connected. That is, a clique of G is a fully connected induced subgraph of G. A clique with $|V'| = k$ is known as a k-clique. If G is fully connected, then G is a clique.

A *path* in G is a finite non-null sequence $P = v_0 e_1 v_1 e_2 \ldots e_k v_k$, whose terms are alternately vertices and edges, such that, for $1 \leq i \leq k$, the ends of e_i are v_{i-1} and v_i and the vertices $v_0, v_1, \ldots, v_k$ are distinct. P is said to be a path *from* v_0 *to* v_k, or a (v_0, v_k)-path. The integer k is the *length* of P. Figure 2.3 uses a dashed line to show a path of length 4 from v_4 to v_5.

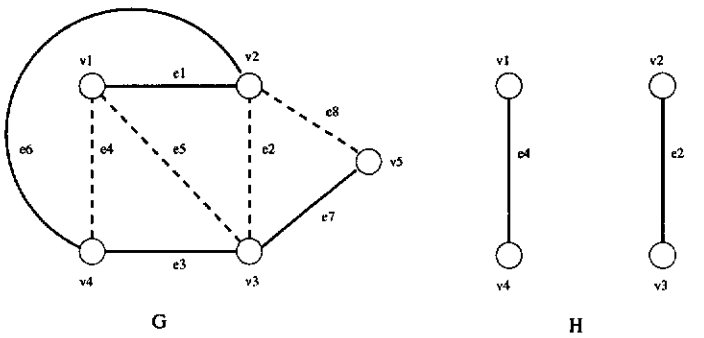


Figure 2.3: Properties of G and H

Two vertices μ and ν of G are said to be *connected* if there is a (μ, ν) path in G. Connection is an equivalence relation on the vertex set V. Thus there is a partition of V into nonempty subsets $V_1, V_2, \ldots, V_\omega$ such that two vertices μ and ν are connected if and only if both μ and ν belong to the same set V_i. The subgraphs $G[V_1], G[V_2], \ldots, G[V_\omega]$ are called the *components* of G. If G has exactly one component, G is *connected*; otherwise G is *disconnected*. A simple graph in which each pair of distinct vertices is joined by an edge is called a *fully connected* graph. In Figure 2.3 graph G is connected while graph H is disconnected.

A *vertex cut* of G is a subset V' of V such that $G - V'$ is disconnected. A *k-vertex cut* is a vertex cut of k elements. A fully connected graph has no vertex cut. If G has at least one pair of distinct nonadjacent vertices, the *connectivity* $\kappa(G)$ of G is the minimum k for which G has a k-vertex cut; otherwise $\kappa(G)$ is defined to be $|V(G)| - 1$. Thus $\kappa(G) = 0$ if G is disconnected. G is said to be *k-connected* if $\kappa(G) \geq k$. In Figure 2.3 graph G is 2-connected.

An *edge cut* of G is a subset E' of E such that $G - E'$ is disconnected. A *k-edge cut* is an edge cut of k elements. The *edge-connectivity* $\lambda(G)$ of G is the minimum k for which G has a k-edge cut. Thus $\lambda(G) = 0$ if G is disconnected. G is said to be *k-edge-connected* if $\lambda(G) \geq k$. In Figure 2.3 graph G is 2-edge-connected.

If vertices μ and ν are connected in G, the *distance* $d(\mu, \nu)$ between μ and ν is the length of a shortest (μ, ν)-path in G. In graph G of Figure 2.3 the distance between v_4 and v_5 is 2. If used in a general sense, with reference to one or more pairs of nodes with identical distance, the symbol d may be used rather than $d(\mu, \nu)$. The *average*

inter-vertex distance $\overline{d}(G)$ is the average distance between all pairs of nodes in G. The *diameter* $D(G)$ of G is the maximum distance between two vertices of G. In Figure 2.3 $D(G) = 2$.

The *degree* $\delta(\nu)$ of a vertex ν in G is the number of edges incident with ν. The minimum degree of vertices of G is denoted by $\delta(G)$, or just δ, and is known as the degree of graph G. In Figure 2.3 $\delta(G) = 2$, since $\delta(v_5) = 2$.

Two graphs G and H are said to be *isomorphic* (written $G \cong H$) if there are bijections $\theta : V(G) \to V(H)$ and $\phi : E(G) \to E(H)$ such that $\Phi_G(e) = \mu\nu$ if and only if $\Phi_H(\phi(e)) = \theta(\mu)\theta(\nu)$; such a pair (θ, ϕ) of mappings is called an *isomorphism* between G and H. Figure 2.4 shows two graphs G and H which are isomorphic.

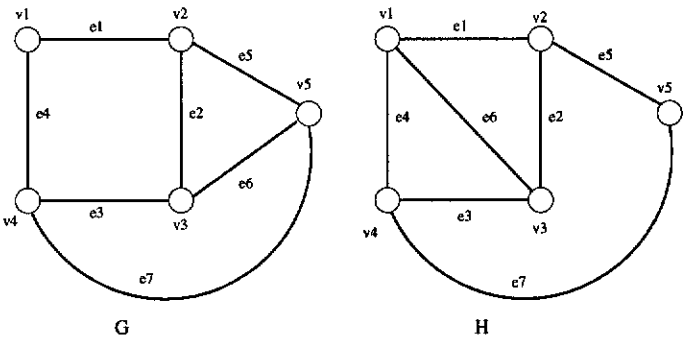


Figure 2.4: Isomorphic graphs G and H

For two graphs G and H, G is said to be *subgraph isomorphic* to H if there are injections $\theta : V(G) \to V(H)$ and $\phi : E(G) \to E(H)$ such that $\Phi_G(e) = \mu\nu$ if and only if $\Phi_H(\phi(e)) = \theta(\mu)\theta(\nu)$; such a pair (θ, ϕ) of mappings is called a *subgraph isomorphism* between G and H. Figure 2.5 shows two graphs G and H where G is subgraph isomorphic to H.

For two attributed graphs G and H, G is said to be *subgraph isomorphic* to H if there are injections $\theta : V(G) \to V(H)$ and $\phi : E(G) \to E(H)$ such that $\alpha_G(\mu) = \alpha_H(\theta(\mu))$ and $\Phi_G(e) = \mu\nu$ if and only if $\Phi_H(\phi(e)) = \theta(\mu)\theta(\nu)$.

In graph theoretic terms a *tree* is a simple graph T such that T is connected and if the edge $e_i \in E(T)$ then $T - e_i$ is disconnected; in other words T is a minimal connected graph. Figure 2.6 shows all the trees on five vertices. It has been shown that, in a tree,

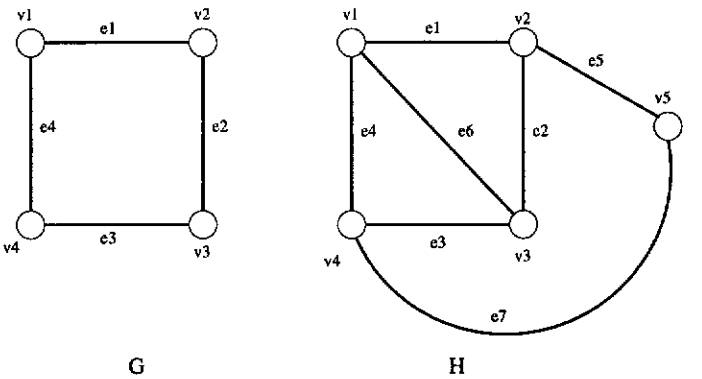


Figure 2.5: Graph G Subgraph Isomorphic to graph H

any two vertices are connected by a unique path. A *rooted tree* has a particular vertex

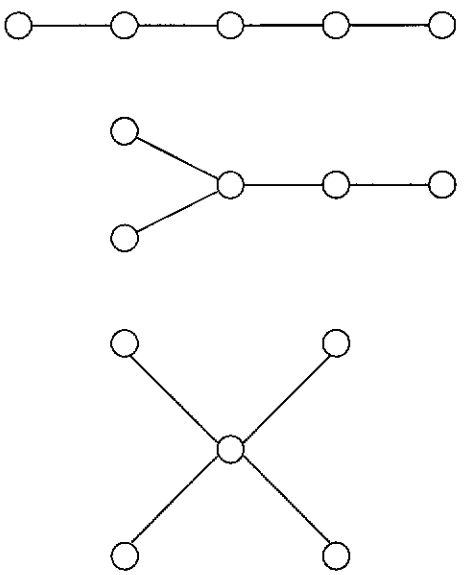


Figure 2.6: All trees on five vertices

designated as the *root*. All trees in this thesis are assumed to be rooted trees. Consider a rooted tree T, with $(\mu, \nu) \in V(T)$ where ν is not the root. If P_1 is the path from ν to the root and $\mu \in P_1$ then μ is an *ancestor* of ν and ν is a *descendant* of μ. Further, if μ and ν are adjacent and $\mu \in P_1$ then μ is the *parent* of ν and ν is a *child* of μ. If a vertex has no child it is known as a *leaf*. If every vertex which is not a leaf has the same number of children k, then T is known as a k-ary tree. The *height* $h(T)$ of a tree T is the maximum distance between the root and any leaf. If, in a k-ary tree T, all the leaves are at the same distance from the root, then T is known as a *complete* tree.

Consider a rooted tree T, with $(\mu, \nu) \in V(T)$ where neither μ nor ν is the root. We

define the *least common ancestor* of μ and ν, denoted $lca(\mu, \nu)$ as follows. $\exists lca(\mu, \nu) \in V(T)$ such that $lca(\mu, \nu)$ is an ancestor of μ, and $lca(\mu, \nu)$ is an ancestor of ν, and the path from μ to $lca(\mu, \nu)$ is edge disjoint from the path from ν to $lca(\mu, \nu)$.

2.2 Fault-Tolerance

Fault-tolerance in computers was defined in [Avizienis(1971)] as

> the ability to execute specified algorithms correctly regardless of hardware failures and program errors.

This implies that analysing fault-tolerance of a particular computer system requires knowledge of the algorithms that must be executed correctly and of the types of faults which can occur. Such knowledge is very implementation dependent. When studying multiprocessor interconnection networks it is desirable to be able to define measures of fault-tolerance which indicate the likelihood that a multiprocessor will remain operational in the presence of faults, without relying on specific knowledge of the algorithms to be run or the faults which may occur. In this thesis, this is made possible by only considering faults which can cause the network to fail. In order to do this it is necessary to define what constitutes a *failed* network. Three definitions arise naturally [Slyke and Frank(1972), Brecht and Colbourn(1986)].

Definition 2.1 *The network has failed if it is disconnected; that is, if there exists any pair of operational nodes which cannot communicate.*

Definition 2.2 *The network has failed if a certain fraction of the operational nodes cannot communicate.*

Definition 2.3 *The network has failed if none of the operational nodes can communicate.*

If a fault is considered to be the removal of a node or link then, using any of the definitions of failure, a system is said to be able to *tolerate* any set of faults, whether node or link removals, which do not cause the network to fail.

A number of measures of fault-tolerance have been proposed; in general they can be considered to be either probabilistic or deterministic. Many of these measures use graph theoretic concepts and were first developed for use in general networks. This section will review the background to the measures which are used later in the thesis.

2.2.1 Deterministic Fault-Tolerance

Let the interconnection network be represented by a graph G. Two simple measures of the fault-tolerance of G are connectivity $\kappa(G)$ and edge-connectivity $\lambda(G)$ [Boesch(1972)]. If failure for G accords to Definition 2.1, then G can tolerate *at least* $(\kappa(G) - 1)$ node faults or *at least* $(\lambda(G) - 1)$ link faults. Connectivity and edge-connectivity can be said to represent minimal failure conditions, since they define the absolute minimum of, respectively, node and link failures which can cause the network to fail. Connectivity and edge-connectivity suffer two deficiencies as measures of fault-tolerance [Esfahanian(1989)]. Firstly, they do not indicate the fraction of G which may be disconnected by a minimal failure condition, and therefore are not useful indicators of the extent of damage which a system may suffer as the result of a minimal failure condition occurring. Secondly, in common with other deterministic measures of fault-tolerance, they make no allowance for variations in the likelihood of failure of different components within the network.

The second concern is addressed by the use of probabilistic measures of fault-tolerance. The first concern was addressed with the introduction of measures of connectedness such as the *atomic number* of a graph, defined informally in [Boesch and Felzer(1972)] as "the 'size' of the smallest component created by a node disconnecting set". Such measures, however, have been little used in parallel computing research, which restricts their utility for the comparison of interconnection network topologies. Connectivity and edge-connectivity, though widely accepted as inadequate, are still the most generally

used deterministic measure of fault-tolerance for interconnection networks.

While much effort has been devoted to determining the conditions leading to failure of a network, the way in which network performance degrades as faults occur is also important. In [Krishnamoorthy and Krishnamurthy(1987)] the concept of *fault diameter* of a network was introduced. For any graph G with connectivity $\kappa(G)$, the fault diameter $f(G)$ is the largest diameter obtained by deleting a set of $(\kappa(G) - 1)$ nodes. Fault diameter is an indicator of the possible degradation in performance of a network in the presence of faults. In [Krishnamoorthy and Krishnamurthy(1987)] two classes of graphs based on fault diameter were distinguished.

Definition 2.4 *A class of graphs* $\mathbf{G_i}$ *is* ***strongly resilient*** *if there exists a constant* $\mathbf{t}$ *such that* $\mathbf{f(G) \leq D(G) + t}$ *for all* $\mathbf{i}$.

Definition 2.5 *A class of graphs* $\mathbf{G_i}$ *is* ***weakly resilient*** *if there exists a constant* $\mathbf{t}$ *such that* $\mathbf{f(G) \leq D(G) \times t}$ *for all* $\mathbf{i}$.

Clearly, the performance of a strongly resilient network can be expected to be less degraded under fault conditions than the performance of a weakly resilient network.

2.2.2 Probabilistic Fault-Tolerance

When dealing with probabilistic fault-tolerance each node and link of a network is considered to have a specific probability of being operational. Many studies of network reliability assume either only nodes or only links may be faulty. It is also common practice to assume that all nodes or links have the same probability p of being operational, and that any failures are statistically independent. Work has, however, been done to address the problem of fault-tolerance in networks with nodes and links with different probabilities of failure, or having correlated failures [Slyke and Frank(1972), Esfahanian(1989)].

There are three measures of probabilistic fault-tolerance which are commonly considered. The most general of these is the *k-terminal reliability* $R_k(G)$ of a graph G. For a

graph G with $|V(G)| \geq k$, consider a set of nodes V' where $V' \subset V(G)$ and $|V'| = k$. $R_k(V')$ is the probability of finding in G a path, entirely composed of operational links and nodes, between every pair of nodes in V'. If $K(G)$ is the set of all sets of k nodes in $V(G)$ then $R_k(G)$ is defined as $(\min R_k(V')) : \forall \quad V' \in K(G)$.

An important special case of the k-terminal reliability is the *all-terminal reliability* $R(G)$ of a graph G, the probability of finding a path entirely composed of operational links and nodes between every pair of operational nodes in G. Thus $R(G)$ is the probability that G is connected. The computation of $R(G)$ for general networks has been shown to be a #P-complete problem [Provan and Ball(1983)].

A second important special case of k-terminal reliability is the *two-terminal reliability* or *path reliability* $R_2(\mu, \nu)$ between a pair of nodes μ and ν in $V(G)$. $R_2(\mu, \nu)$ is defined as the probability of finding a path entirely composed of operational links and nodes between μ and ν [Colbourn(1987)]. If $S(G)$ is defined as the set of all sets of two distinct nodes in $V(G)$ then the two-terminal reliability $R_2(G)$ of graph G is defined as $(\min R_2(\mu, \nu)) : \forall(\mu, \nu) \in S(G)$. It has been shown in [Ball(1980)] that determining $R_2(G)$ is an NP-hard problem.

While two-terminal reliability can give an indication of the likelihood of failure of communications between specific pairs of nodes, or even classes of pairs of nodes, it gives no indication of the average path reliability of a pair of nodes in the graph. A graph with a few short, reliable links and many long, unreliable ones is indistinguishable from a graph with many short, reliable links and few long, unreliable ones. Average two-terminal reliability $\overline{R_2(G)}$ overcomes this shortcoming. $\overline{R_2(G)}$ is defined as $\frac{\sum R_2(\mu,\nu) \forall (\mu,\nu) \in S(G)}{|S(G)|}$, where $|S(G)| = \binom{|V(G)|}{2}$.

Since computing the reliability of networks has been shown to be a hard problem, most work has focused on determining upper and lower bounds for $R(G)$ and $R_2(G)$, which are expressed as $\lceil R(G) \rceil$, $\lceil R_2(G) \rceil$, $\lfloor R(G) \rfloor$ and $\lfloor R_2(G) \rfloor$. Several approaches have been taken to determining bounds, including work by [Lomonosov and Polesskii(1971)] and [Brown *et al.*(1990)].

An approach used in [El-Amawy and Latifi(1991)] involves a number of steps to reduce

the complexity of the problem. For multiprocessor networks it is often possible to reduce the number of node pairs which need to be considered using symmetry. A network is *symmetric* if it is isomorphic to itself with any node labelled as the origin. A symmetric network appears the same if viewed from the perspective of any node. Therefore all values of $R_2(\mu, \nu)$ can be determined by considering only those sets of nodes in $S(G)$ which contain a given node μ. Therefore in symmetric networks $R_2(G) = (\min R_2(\mu, \nu)) : \forall(\mu, \nu) \in S(G)$ containing node μ.

The main simplifying technique is to consider only a subset of available paths, specifically the link or node disjoint paths, between any pair of nodes (μ, ν) in $S(G)$. A lower bound $\lfloor R_2(\mu, \nu) \rfloor$ can be established by determining the probability of at least one link or node disjoint path between μ and ν consisting entirely of operational links and nodes. The following description assumes that only links may be faulty, though the approach works equally well for faulty nodes. Let p denote the probability of a link being operational. Then the probability of any path of length l being operational is p^l and the probability of failure of such a path is $1 - p^l$. The probability of failure of i link disjoint paths, all of length l, is given by $(1 - p^l)^i$. Therefore the probability that at least one of i link disjoint paths of length l is operational is $1 - (1 - p^l)^i$.

Combining the techniques, it is possible to determine a lower bound $\lfloor R_2(G) \rfloor$ by determining the minimum value of $R_2(\mu, \nu)$ over a restricted set of paths between a restricted set of nodes.

Another probabilistic measure proposed by [El-Amawy and Latifi(1991)], is based upon the observation that two networks with the same connectivity or edge-connectivity do not necessarily have the same likelihood of a minimum failure set occurring. The probability of a minimum failure set occurring can be used in conjunction with connectivity or edge-connectivity as an indicator of the likelihood of failure.

2.3 Interconnection Networks

A parallel computer consists fundamentally of a set of processors and an interconnection network to allow communication between the processors. "The topology of an interconnection network can be either static or dynamic. *Static networks* comprise point-to-point direct connections which do not change during program execution. *Dynamic networks* are implemented with switched channels, which are dynamically configured to match the communication demand in user programs" [Hwang(1993)]. The performance of an interconnection network is affected by *network latency*, which is the worst case time delay for a unit message to be transferred through the network. Ideally, every processor could communicate directly with every other processor, keeping latency low. A simple, dynamic, bus based system allows this, but only a small number of processors can be supported before conflicts on the bus start to cause *congestion*, which forces some messages to wait in queues and reduces performance.

An alternative is a static, point to point connection scheme with communication links between particular pairs of processors. Such direct interconnection schemes are described in graph theoretic terms but, as noted in Section 2.1, processors are designated as *nodes* rather than vertices and communication channels are called *links* rather than edges. An obvious network to consider is the fully connected network, with a link between every pair of processors. The fully connected network is ideal in some regards. It has simple addressing and routing; each node has some unique address and routing consists simply of selecting the correct link from the total address space. There is never any conflict on any links, so congestion never occurs and the routing is deadlock-free. The diameter of the network is one, so the network latency is the minimum possible. Fully connected networks are extremely fault-tolerant. An N node system has degree $\delta = N - 1$ and link-connectivity $\lambda = N - 1$. The fully connected network is completely tolerant of node failures, as no number of node failures can ever cause disconnection of the network. Any single link fault causes the diameter to rise by only one. All communication patterns map directly onto the network, reducing programming effort and increasing algorithmic efficiency.

Against these advantages are practical and economic factors. Since the number of links in a fully connected network with N nodes is $\binom{N}{2}$ and the degree of each node is $N-1$ it becomes physically difficult to connect the required number of links to each node and to fit all the links into the space between the nodes. Even if the physical difficulty could be overcome, it would probably remain economically unattractive to build fully connected networks with more than a small number of nodes, as the total cost of the links would increase in proportion to $\binom{N}{2}$.

A simple, low cost network with a node degree of only two is the ring network. The ring topology has been used in commercial parallel computers, such as the CDC Cyberplus, and the addressing and routing are simple. However conflict and congestion are very likely to occur and the network latency is $\mathcal{O}(N)$ steps for an N node ring. Fault-tolerance is poor, a single node or link fault will double the network diameter and any two node or link faults will disconnect the network. Few application problems map naturally onto a ring topology, many programs tend to be inefficient despite having required extra programming effort.

In between the extremes of the fully connected network and the ring network there are a range of possible interconnection network topologies. Each topology can be characterised in terms of its diameter and node degree, the simplicity of addressing and routing schemes, its fault-tolerance, the ease with which data structures and communication patterns can be mapped to it, its cost and ease of fabrication. Each topology offers a different combination of advantages and drawbacks, and topologies which are suitable for some applications may be inappropriate for others. The following sections describe some of the most important interconnection network topologies and discuss why they have been successful and what their shortcomings are.

2.3.1 Hypercube

A binary n-cube Q, commonly called a hypercube, consists of $N = 2^n$ nodes and $L = n2^{n-1}$ links arranged in n dimensions, with each node connected to one other node per dimension. Each node has an n-bit binary address. If two nodes are adjacent

within the hypercube their addresses will differ in only one bit. Figure 2.7 shows a hypercube with four dimensions.

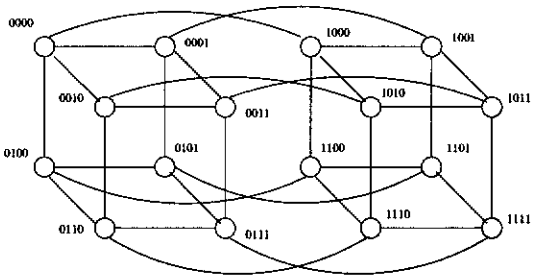


Figure 2.7: A four dimensional hypercube

The hypercube has many features which are desirable in a multiprocessor interconnection network. Addressing is simple and elegant. Simple, minimal, deterministic, routing and broadcast algorithms using dimension ordering were described by [Sullivan and Bashkow(1977)] and later shown to be deadlock-free [DeBenedictis(1982)]. Deadlock free adaptive routing algorithms have also been described, often using the concept of virtual channels [Dally and Seitz(1987)] to prevent deadlock. The diameter of a binary hypercube is $D(Q) = n$, which is small, as is the average internode distance $\overline{d}(Q) = n2^{n-1}/2^n - 1 \approx n/2$ for large N.

The hypercube has excellent fault-tolerance characteristics [Saad and Schultz(1988), Esfahanian(1989), Tien and Raghavendra(1993)]. The network has link-connectivity $\lambda(Q) = n$, connectivity $\kappa(Q) = n$ and degree $\delta(Q) = n$. The hypercube has fault diameter $f(Q) = n+1$. A lower bound on the two terminal reliability $\lfloor R_2(Q) \rfloor$ was determined in [El-Amawy and Latifi(1991)]. If the probability of a link being operational is p, then $\lfloor R_2(Q) \rfloor = \min(1 - ((1-p^d)^d(1-p^{(d+2)}))^{(n-d)})$ over all distances $0 < d \leq n$.

The hypercube has been shown to efficiently embed many important topologies and data structures, including rings and linear arrays [Saad and Schultz(1988)], meshes [Saad and Schultz(1988), Chan(1991)], pyramids [Stout(1986), Lai and White(1990)] and trees [Bhatt *et al.*(1992)]. The structure of hypercubes allows efficient, natural, easily programmable communication patterns for important problems such as FFTs. The extensive amount of research conducted into this topology has resulted in many algorithms being developed and optimised for hypercubes.

A message passing multiprocessor using an interconnection network with a hypercube topology was proposed by [Squire and Palais(1963)] and a microprocessor array based on the indirect binary hypercube was proposed by [Pease(1977)], but practical systems were not constructed until the eighties. Several commercial and many research machines have been built, including the Cosmic Cube, Intel iPSC/1 and iPSC/2, NCUBE/10, Floating Point Systems T-series and Thinking Machines Corporations CM-2 [Seitz(1990), Seitz(1985), Hockney and Jesshope(1988), Hwang(1993)]

The major drawback of the hypercube is the node degree, which increases with the dimension. This makes the hypercube difficult to scale, since adding nodes requires extra connections to be made to every existing node and large systems require each node to have a large number of connections. This led to cost concerns and caused the hypercube to fall from favour as parallel systems moved towards higher levels of system integration.

2.3.2 Mesh

In general an n-dimensional mesh M (no wrap-around links) with radix r (the number of nodes on an edge, in each dimension) has $N = r^n$ nodes and $L = n(r-1)r^{n-1}$ links. There are several methods of addressing nodes of a mesh; row major, column major, snakelike etc. Fundamentally though, a meshs' address consists of an ordered sequence of numbers, one number for each dimension. One corner of the mesh is designated the origin and allocated the address consisting of all zeroes. For any given node, its position relative to the origin in the i^{th} dimension is indicated by the value in the i^{th} position of its address. Figure 2.8 shows a two-dimensional mesh with $r = 4$.

As with the hypercube, simple, minimal, deterministic routing and broadcast algorithms are possible using dimension ordering techniques, while adaptive routing algorithms often use virtual channels. The mesh has diameter $D(M) = n(r-1)$, which is quite large. Modern mesh connected multiprocessors rely on cut-through routing techniques to reduce the effect of distance on latency. Fault-tolerance is generally acceptable. The degree of an internal node x is $\delta_x = 2n$, the degree of a face or edge node

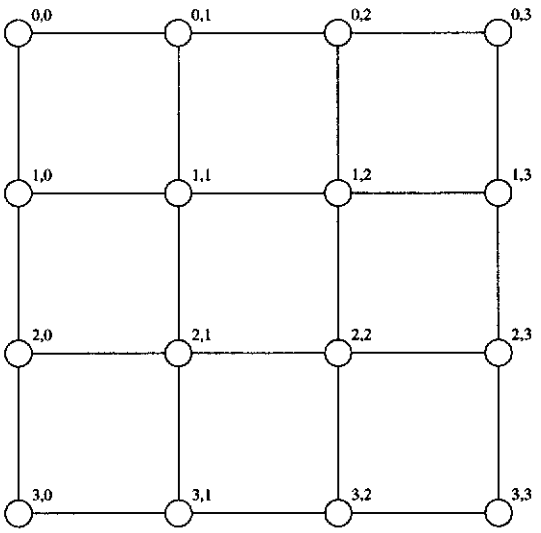


Figure 2.8: A two-dimensional mesh with radix four

y is $n < \delta_y < 2n$ and the degree of a corner node z is $\delta_z = n$, therefore the degree of the network is $\delta(M) = n$. The mesh has connectivity $\kappa(M) = n$ and link-connectivity $\lambda(M) = n$, which is rather low. The fault diameter is $f(M) = n(r-1)$, the same as the diameter. The low fault diameter value is not quite as good as it appears, however, since it is influenced by the low connectivity. For example, a two dimensional mesh has $\kappa(M) = 2$, so the fault diameter is determined by removing only one node.

Some important groups of problems map onto meshes easily and naturally. Images and regular finite element meshes map readily onto mesh networks of the same dimension, while preserving nearest neighbour connections which are essential for many algorithms commonly used on such data sets. Dense matrices also map easily, but since nearest neighbour connections are seldom important for matrix operations, more complex mappings are often required for efficient communication. The communication patterns required to solve problems such as the FFT do not map naturally to meshes, neither do many other interconnection network topologies, such as hypercubes. Some clever algorithms have been developed to allow meshes to provide reasonable performance when solving problems which do not map naturally to meshes, but low dimensional meshes are fundamentally limited in the range of problems for which they can provide optimal or near optimal communication. The major benefits of mesh topologies are the ease of construction and high link bandwidth made possible by their regularity and low

degree [Dally(1990b)].

The mesh topology has a long history in parallel computing. A two-dimensional mesh of (primitive) processors was proposed by [Unger(1958)] in order to deal with spatial data, but technology of the time was insufficient to allow implementation. Many of the early parallel computers, such as the ILIAC IV and the ICL DAP [Reddaway(1973)] used variations of two-dimensional meshes as the topology for their interconnection networks. As richer connection topologies, such as the binary hypercube, became popular the simple two-dimensional mesh lost popularity. More recently, however,commercial systems such as the Intel Delta, Ametek Series 2010 and SuperNode 1000 employed two-dimensional mesh interconnection networks, while the J-Machine and Cray T3D use three-dimensional mesh interconnection networks [Hockney and Jesshope(1988), Seitz(1990), Hwang(1993)]. The resurgence in interest in the mesh is mainly attributable to three factors. Firstly, VLSI systems were found to be wire limited, which is to say that the dominant design considerations were the amount of space and power consumed by interconnections, and the delays those interconnections introduced. Analysis showed that low-dimensional meshes made more effective use of chip area than higher-dimensional networks [Dally and Seitz(1986), Dally(1990b)]. Secondly, the introduction of various cut-through routing techniques reduced the effect of network diameter on message latency [Kermani and Kleinrock(1979), Dally(1990b)]. Thirdly, the development of integrated, commodity hardware suitable for the construction of mesh based multiprocessor interconnection networks made the construction of such systems more cost effective [Walker(1985), Dally and Seitz(1986)]. While variations of the basic mesh, such as the ILIAC mesh and the torus, offer some advantages (and some disadvantages), they are sufficiently similar that the important parameters of all the topologies can be approximated by examining only the simple mesh.

2.3.3 Trees, Fat Trees and Generalised Fat Trees

A complete k-ary tree of height h, as defined in Section 2.1, has a number of nodes $N = \sum_{x=0}^{h} k^x$ which can be converted to the closed form $N = (k^{h+1} - 1)/(k - 1)$ for $k \neq 1$. It has $L = N - 1$ links. A simple addressing scheme assigns two numbers, x and

y, to each node. The value of x, where $0 \leq x \leq h$, indicates the level at which the node occurs within the tree. It is assumed that the leaves occur at level 0, the root at level h. [1] The value of y, where $0 \leq y < k^{(h-x)}$, identifies the node amongst the nodes of level x. The parent of any non-root node (x, y) will have address $(x+1, y/k)$. Figure 2.9 shows a complete 3-ary tree of height two.

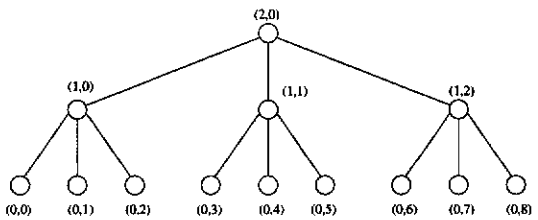


Figure 2.9: A complete 3-ary tree of height 2.

When considering tree-based multiprocessor topologies it is very common to consider the network as dynamic. Only the nodes at level 0, the leaves, represent processing elements (PEs). All other nodes represent switching elements (SEs). The mesh and hypercube, discussed earlier, are normally static networks, and the number of PEs equals N, the number of nodes. A complete k-ary tree of height h has a number of PEs $N_P = k^h$ and a number of SEs $N_S = k(k^h - 1)/(k-1)$ for $k \neq 1$.

Routing is simple, the source processor sends the message up the tree until it reaches the node at the root of the smallest subtree containing both the source and the destination nodes. The message is then routed down the tree to the destination. Since no cycles are possible routing is deadlock-free. Both the diameter and the average internode distance are calculated using the distances between leaf nodes only. A complete k-ary tree of height h has diameter $D = 2h$, which is large for values of k which are practicably small. The average internode distance of $\overline{d} = (2(k-1)\sum_{x=1}^{h} xk^{x-1})/(k^h - 1)$, is also large.

Fault-tolerance of a tree is rather poor. For a k-ary tree, the degree of any leaf node is one, the degree of the root node is k and the degree of any internal node is $k+1$, so the network has degree $\delta = k+1$. Link and node connectivity are both one. Fault diameter is not applicable, since the removal of any link or non-leaf node will disconnect

[1] Note that it is common practice in some areas of study to designate the root as occupying level 0. In studies of parallel topologies, however, assigning the root to level h is standard practice.

the network. Tree based systems which require fault-tolerance typically incorporate redundant links and nodes [Dutt and Hayes(1990)].

The tree network is scalable, cheap and relatively easy to implement. Although not as popular as the hypercube or mesh, tree topologies have been widely studied for use in interconnection networks, and the DADO multiprocessor built at Columbia University was a static ten level binary tree [Hockney and Jesshope(1988)]. The major problem with tree networks is the communication bottleneck caused by congestion in links at higher levels.

A solution to the bottleneck problem is the fat tree [Leiserson(1985)]. The fat tree has links of increased bandwidth between nodes which are close to the root. This alleviates the problem of congestion without altering any of the fundamentals of the topology. Further development of the fat tree concept resulted in a more flexible family of topologies. Generalised fat trees were defined and comprehensively described in [Öhring *et al.*(1995)], where they are represented as $GFT_{(h,m,w)}$. Here h is the height of the tree, m is the number of children of each non leaf node, and w is the number of parents of each non-root node. A GFT is a tree, as defined in Section 2.1, only when $w = 1$. However, terminology applicable to trees is used when describing GFTs with no confusion.

GFTs were informally defined as follows: $GFT_{(h+1,m,w)}$ is recursively generated from m distinct copies of $GFT_{(h,m,w)}$, each denoted as $GFT^j_{(h,m,w)}$ for $0 \leq j \leq m-1$, and w^{h+1} additional nodes such that each top-level node $(h, k + j \cdot w^h)$ of each $GFT^j_{(h,m,w)}$, for $0 \leq k \leq w^h - 1$ is adjacent to w consecutive new top-level-nodes (*i.e.* level $h+1$ nodes), given by $(h+1, k \cdot w), \ldots, (h+1, (k+1) \cdot w - 1)$. Figures 2.10 and 2.11 show $GFT_{(2,2,3)}$ and $GFT_{(2,4,2)}$ respectively.

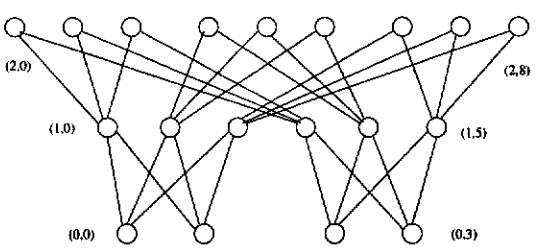


Figure 2.10: A $GFT_{(2,2,3)}$.

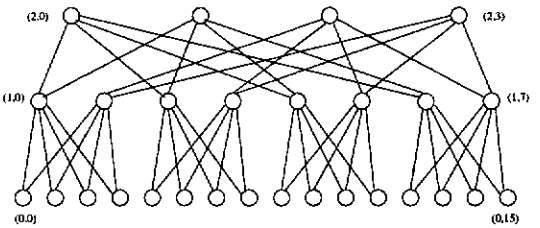


Figure 2.11: A $GFT_{(2,4,2)}$.

GFTs were defined as dynamic interconnection networks, which is to say only leaf nodes are PEs, with all other nodes acting as SEs. For any $GFT_{(h,m,w)}$ the number of PEs $N_P = m^h$, and the number of SEs $N_S = \sum_{x=1}^{h}(m^{h-x} \times w^x)$ which converts to the closed form $\frac{w(w^h-m^h)}{(w-m)}$ when $w \neq m$. The number of links $L = \sum_{x=0}^{h-1}(m^{h-x} \times w^{x+1})$ which converts to the closed form $\frac{wm(w^h-m^h)}{(w-m)}$ when $w \neq m$. Addressing and routing remain essentially the same as for standard trees. The routing for GFTs differs from that for trees in that messages being passed up the tree may have a variety of routes to choose from. Simple, deterministic, deadlock-free routing algorithms are available, as are deadlock-free dynamic routing algorithms. A $GFT_{(h,m,w)}$ has diameter $D = 2h$, which is the same as a tree of the same height, and average inter-PE [2] distance $\overline{d} = 2h - \frac{m^h-hm+h-1}{m^{h+1}-m^h-m+1}$. The value of $\overline{d}$ can obviously be manipulated by varying m and h as part of a trade-off between cost and performance.

Fault-tolerance of GFTs can also be altered by varying the underlying parameters m and w. Degree of the network is $\delta = m + w$ and connectivity is $\kappa = w$. Generalised fat trees can be more fault-tolerant than fat trees, since a single parent node no longer acts as an articulation point for an entire subtree. Congestion in the upper levels is also less likely to occur. The increased degree and number of links in the system, however, makes the network more difficult and expensive to implement.

[2]The average inter-node distance of static networks such as the mesh or hypercube are most meaningfully compared with the average inter-PE distance of dynamic networks such as the GFT, rather than the average inter-node distance which includes all the switching nodes. For this reason we have used the symbol $\overline{d}$ to represent the average inter-PE distance when describing dynamic networks.

2.3.4 Other Interconnection Networks

Many other interconnection networks have been proposed and studied, though most are variations, combinations or generalisations of the major types previously described. This section provides a brief overview of some of the more important networks which have been described in the literature.

Reported variations of the direct binary hypercube include the cube-connected cycles [Preparata and Vuillemin(1981)], the folded hypercubes [El-Amawy and Latifi(1991)], enhanced hypercubes [Tzeng and Wei(1991)], twisted n-cubes [Esfahanian *et al.*(1991)], and the crossed cube [Efe(1992)]. Most of these networks offer advantages over the ordinary hypercube in one or more important areas, such as degree, diameter or fault-tolerance. The k-ary n-cube is an important generalisation of the hypercube, mesh and torus [Dally and Seitz(1987)]. Hierarchical networks have been proposed which allow large systems to be created by interconnecting modules based upon densely connected networks. Some examples are the hierarchical hypercube [Malluhi and Bayoumi(1994)], the hierarchical interconnection networks [Dandamudi and Eager(1990)], the extended hypercube [Kumar and Patnaik(1992)], and the hypernet [Hwang and Ghosh(1987)]. Hierarchical systems generally aim to provide the benefits of their densely connected sub-network in a more scalable package.

2.4 Irregular Algorithms

Much of the effort invested in parallel computer design and development to date has been directed towards providing rapid solutions for parallel algorithms with a strongly regular structure. These include such scientific and engineering mainstays as matrix manipulation, fast Fourier transforms and the solution of partial differential equations.

Irregular algorithms have an irregular structure, or deal with sparse or irregular data, or both. As a result, parallel irregular algorithms have patterns of communication and computation which are unstructured or changing dynamically. Examples include sparse matrix operations [Moreira *et al.*(1998), Fu and Yang(1997)], unstructured multigrid

computation fluid dynamic solvers [Brezany *et al.*(1994)], molecular dynamics codes [Brezany *et al.*(1994)] and algorithms in machine vision [Chung *et al.*(1998)]. Some of these areas have long been studied, but recently parallel irregular algorithms have been attracting wider interest as a class, as parallel computing has become better supported and more widely available.

2.4.1 Dynamic Load Balancing

Algorithmic irregularity may be due to data structures which are irregular, such as sparse matrices or unbalanced trees, or to the form of the algorithms function, which may behave in a reactive fashion to data. Whatever the cause, parallel irregular algorithms present difficulties when seeking efficient implementations on many interconnection networks. While regular parallel algorithms can be structured to take maximum advantage of the natural communication pattern of an underlying topology, parallel irregular algorithms rely on dynamic load balancing to achieve efficiency. With dynamic load balancing, processors may often need to request or distribute work. Determining where to look for work, or for processors to perform work, can be a significant problem. A dynamic load balancing strategy determines how each processor will proceed when seeking to receive or distribute work. When applied to a specific parallel architecture, a strategy becomes an algorithm. A number of dynamic load balancing strategies are surveyed in [Kumar *et al.*(1994)], and the scalability of their corresponding algorithms on a number of architectures is analysed. The strategies can be broadly classified as receiver initiated or sender initiated. In receiver initiated schemes, work is split only when an idle processor requests work from a target processor with more than some threshold quantity of work. The target processor then divides its work and passes some to the requesting processor. If a processor receives a request for work when it is idle, or has too little work, then it rejects the request. If an idle processor has a request for work rejected, it selects another target and sends it a work request. In sender initiated schemes, work is divided into sub-tasks then allocated to processors, either on request or according to some pre-determined allocation scheme. Although sender initiated strategies have been proposed, receiver initiated strategies are in general more

efficient. The following sections describe some of the receiver initiated strategies which were analysed in [Kumar *et al.*(1994)].

2.4.1.1 Nearest Neighbour (NN)

In the nearest neighbour scheme, an idle processor will request work from its nearest neighbours in a round robin fashion. This scheme is simple to implement and keeps all communication local, reducing overheads. A disadvantage is that extra time may be required to propagate localised work across the entire network.

2.4.1.2 Random Polling (RP)

This is a very simple scheme, whereby an idle processor requests work from a processor selected at random. This scheme does not take advantage of locality, and may result in processors receiving multiple simultaneous requests for work.

2.4.1.3 Asynchronous Round Robin (ARR)

Using this strategy, each processor maintains a variable *target*. An idle processor reads the value of its *target* and sends a work request to the processor with that address. The value of *target* is incremented modulo P each time a request is sent. This scheme is simple, but does not take advantage of locality, nor does it ensure that a single processor will not receive multiple work requests at the same time.

2.4.1.4 Global Round Robin (GRR)

Under this strategy a global variable *target* is stored at a single processor. Any idle processor requests the value of *target*, then requests work from the processor with that address. The value of *target* is incremented modulo P each time its value is requested. This scheme is simple and ensures that processors will not receive multiple requests for

work at the same time. However,it does not take advantage of locality, and contention for the value of *target* can cause a bottleneck.

2.4.1.5 Scheduler Based Load Balancing (SB)

This scheme employs a single processor as a scheduler, which maintains a queue of all possible processors which may have work to distribute. An idle processor requests work from the scheduler, which polls processors on the queue until one is found which has work, then directs the work to the requesting processor and adds the address of the receiver to the queue. This scheme is rather more complex, though it does ensure that processors will not receive multiple requests for work at the same time, and it reduces the number of requests for work which are sent to idle processors. However,it does not take advantage of locality, and contention for the scheduler can cause a bottleneck.

2.5 Conclusion

This chapter introduced the fundamental terms and concepts which will be used in the remainder of this thesis. A short introduction to basic graph theory was provided, as graph theory is conveniently and commonly used in describing and analysing interconnection networks. Some fundamental material relating to fault-tolerance in interconnection networks was also introduced. The general requirements for an interconnection network were discussed, then some of the more important and successful interconnection networks were described and their strengths and weaknesses reviewed. Finally the difficulty inherent in mapping irregular algorithms onto existing interconnection networks was discussed, and some dynamic load balancing strategies were described.

The next chapter will provide a detailed description of the HiC and $FatHiC$ interconnection networks. Fundamental parameters of the two new networks, which affect their performance, cost, utility and fault-tolerance will be derived and compared with those for existing interconnection networks.

Chapter 3

Structure, Parameters and Properties of *HiCs*

3.1 Introduction

In the previous chapter the requirements and constraints of interconnection networks for multiprocessors were discussed. The interconnection network makes parallel processing possible and yet, simultaneously, is a fundamental restriction on the effectiveness and efficiency of parallel processing. Current interconnection networks offer high performance, but the demands placed upon parallel computers continue to grow and change. More types of algorithms are being implemented on parallel computers than previously, in particular irregular algorithms. New interconnection networks must be developed which allow effective, efficient communication between processors while executing a wide variety of parallel algorithms.

Underlying the ability of any interconnection network to be effective and efficient is its topology. The basic characteristics of any interconnection network, including minimum latency, throughput, reliability and cost, are affected by the parameters of the underlying topology. The addressing scheme used by an interconnection network is also important, as it can affect the efficiency of routing algorithms and the ease with

which message passing processes communicate. With this in mind, a number of existing interconnection networks were described and their advantages and disadvantages analysed. This chapter describes the topology, addressing scheme and message passing algorithms of HiCs in detail, establishing the basic system descriptions required in later sections. This chapter also discusses the fault-tolerance of HiCs, as well as their ability to embed other major interconnection schemes.

The purpose of Section 3.2 is to provide a comprehensive description of the HiCs interconnection scheme. This includes a description of the topology and addressing scheme. Section 3.3 then derives a number of important parameters of the topology. These are used directly within the chapter as indicators of various properties of HiCs, as well as in later chapters when discussing other measures such as performance and cost. It should be noted that HiCs could be implemented as either static or dynamic networks. In this thesis, however, they are only considered as dynamic networks, with PEs at leaf nodes while all other nodes act as switches. Section 3.4 describes the network's susceptibility to and tolerance of faults. The routing algorithms of HiCs are detailed in Section 3.5. Finally, Section 3.6 discusses embeddings into HiCs by other multiprocessor interconnection network topologies.

3.2 Structure of HiCs

$HiC_{(k,h)}$ is a k-ary tree of height h modified so that groups of nodes on the same level form cliques. Members of a clique are referred to as *neighbours*. The root node is at level h, and has address 0. The k children of the root node are at level $h-1$ and form a clique. They have addresses consisting of a single unique digit in the range 1 to k. In general let μ be a node at level l of $HiC_{(k,h)}$, where $(0 \leq l < h-1)$. Then μ has address M consisting of a sequence of digits $\langle M_l, \ldots, M_{h-1} \rangle$, where each digit is in the range 1 to k. Consider a second node ν in the same HiC as μ. If ν's address N is a proper suffix of M then ν is an ancestor of μ and μ is a descendant of ν. If $M = \langle M_l, N \rangle$ then ν is the parent of μ and μ is a child of ν. If a sequence P exists such that $N = \langle P, N_{h-1} \rangle$ and $M = \langle P, M_{h-1} \rangle$ then μ and ν are neighbours.

As an example of the structure and addressing scheme of an HiC, consider Figure 3.1. Figure 3.1 shows part of an HiC with $k = 4$ and $h = 3$. (Only one quarter of the level 0 nodes are shown). The root node has address 0 and is at level 3. The root node has four children at level 2, with addresses 1, 2, 3 and 4. The nodes at level 2 form a clique; in Figure 3.1 nodes of a clique are shown enclosed in a dashed line. Each level 2 node has four children at level 1. The address of a node at level 1 consists of the address of its parent node appended to a digit between 1 and 4. The digit distinguishes the level 1 node from its siblings. Thus nodes 12, 22, 32 and 42 are all children of node 2. Nodes at level 1 are neighbours if their parents are neighbours and the first digit of their address is the same; for example nodes 21, 22, 23 and 24 form a clique. The address of a node at level 0 consists of the address of its parent node appended to a digit between 1 and 4. The digit distinguishes the level 0 node from its siblings. Thus nodes 141, 241, 341 and 441 are all children of node 41. Nodes at level 0 are neighbours if their parents are neighbours and the first digit of their address is the same, for example nodes 241, 242, 243 and 244 form a clique.

3.3 Parameters of HiCs

For any $HiC_{(k,h)}$ there are $h+1$ levels, with the leaf nodes at level 0 and the root node at level h. The total number of nodes $N = \frac{(k^{(h+1)}-1)}{(k-1)}$ for $k \neq 1$. Recall that we use only leaf nodes as processors (PEs), with all other nodes acting as switches (SEs). We wish to determine the number of SEs N_S, the number of PEs N_P and the number of communication links L. At any level l, where $0 \leq l \leq h$, there are $N_l = k^{(h-l)}$ nodes. Therefore the total number of leaf nodes $N_P = k^h$. The number of switching nodes $N_S = \sum_{l=1}^{h} k^{(h-l)}$ which can be converted to the closed form $N_S = \frac{k^h-1}{k-1}$ for $k \neq 1$. The number of links $L = L_T + L_C$ where L_T represents the number of links in a k-ary tree of height h and L_C represents the number of links connecting nodes into k-cliques. The total number of links within a k-ary tree of height h is given by $L_T = \sum_{l=1}^{h} k^l$. The number of links within a k-clique is given by $k(k-1)/2$. The number of k-cliques within $HiC_{(k,h)}$ is given by $\sum_{l=0}^{h-1} k^{(h-1-l)}$. Therefore, $L_C = (k(k-1)/2) \times \sum_{l=0}^{h-1} k^{(h-1-l)}$ which simplifies to $L_C = (k-1)/2 \times \sum_{l=1}^{h} k^l$. Therefore $L = (k+1)/2 \times \sum_{l=1}^{h} k^l$ which can

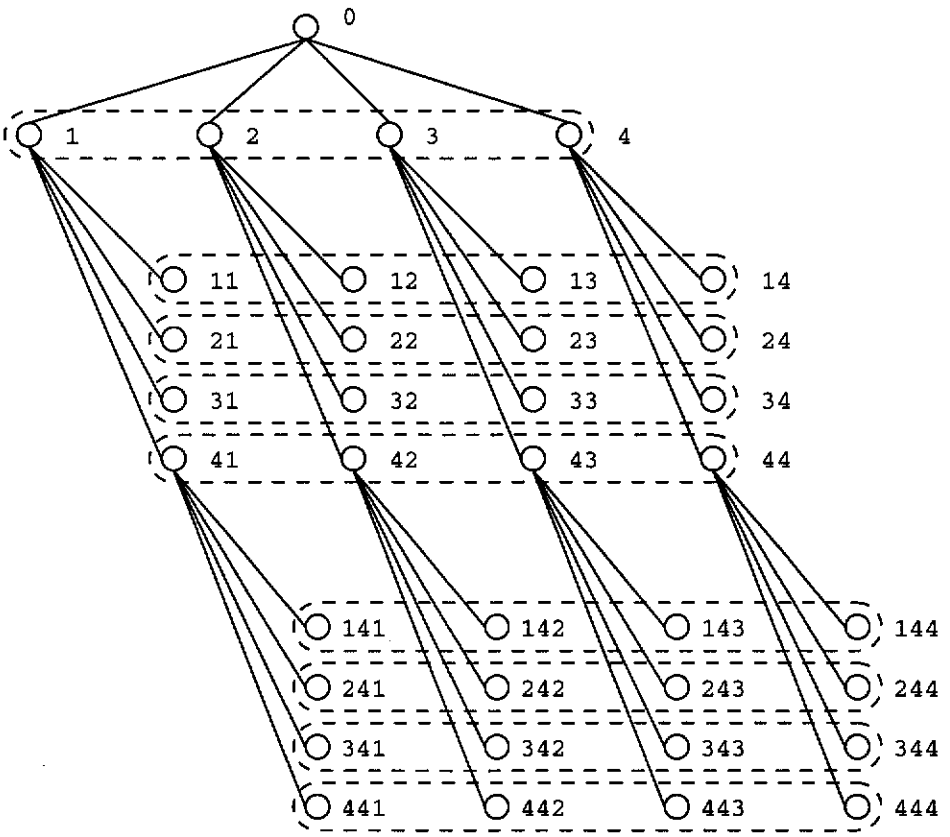


Figure 3.1: Part of a HiC with k = 4 and h = 3.

be converted to the closed form $L = \frac{(k+1)}{2}\frac{k(k^h-1)}{(k-1)}$ for $k \neq 1$.

One of the most important properties of a multiprocessor interconnection network is the diameter. Before deriving the diameter of an $HiC_{(k,h)}$ we require two lemmas.

Lemma 1 *For any $HiC_{(k,h)}$ with $k > 1$ and $h > 0$, if two nodes μ and ν exist at level 0, with addresses $M = \langle M_0, \ldots, M_{h-1} \rangle$ and $N = \langle N_0, \ldots, N_{h-1} \rangle$ respectively, such that $M_x \neq N_x$ for $0 \leq x \leq h-1$, then $d(\mu, \nu) = 2h-1$.*

Proof: Since $M_x \neq N_x$ for $0 \leq x \leq h-1$ no prefix P can exist such that $M = \langle P, M_{h-1} \rangle$ and $N = \langle P, N_{h-1} \rangle$, therefore μ and ν cannot be neighbours. Now consider the ances-

tors of μ and ν at level l. Their addresses will be $\langle M_l, \ldots, M_{h-1} \rangle$ and $\langle N_l, \ldots, N_{h-1} \rangle$ respectively. Then, since $M_x \neq N_x$ for $0 \leq x \leq h-1$, μ and ν can have no common ancestor nearer than the root node. Further, the same consideration which says that μ and ν cannot be neighbours rules that none of their ancestors at levels $l < h-1$ can be neighbours. Therefore the shortest path between μ and ν passes through the link between their respective ancestors at level $h-1$ and has a length of $2h-1$. □

Lemma 2 *For any $HiC_{(k,h)}$ with $k > 1$ and $h > 0$ there exist at least two nodes μ and ν at level 0, with addresses $M = \langle M_0, \ldots, M_{h-1} \rangle$ and $N = \langle N_0, \ldots, N_{h-1} \rangle$ respectively, such that $M_x \neq N_x$ for $0 \leq x \leq h-1$.*

Proof: Nodes at level 0 have an address made up of h digits in the range 1 to k. The number of unique addresses which can be generated with a sequence of h digits, where each digit is in the range 1 to k, is k^h. The number of nodes at level 0 is k^h, and each node has a unique address. □

Theorem 1 *A hierarchical clique $HiC_{(k,h)}$ with $k > 1$ and $h > 0$ has diameter $D = 2h-1$.*

Proof: An $HiC_{(k,h)}$ is a k-ary tree of height h, modified with extra links. A complete tree of height h has diameter $D = 2h$. This is a well known result. The extra links which form the cliques can not increase the diameter. Since the nodes at level $h-1$ form a clique the distance from any node at level $h-1$ to any other node at level $h-1$ is $d = 1$. Therefore the maximum possible distance between two nodes in an $HiC_{(k,h)}$ is $2h-1$. From Lemmas 1 and 2 we know that at least one pair of nodes μ, ν will exist with distance $d(\mu, \nu) = 2h-1$. □

Another important parameter of a multiprocessor interconnection networks is the average distance between processors. As discussed in section 2.3.3, the average inter-PE distance is used in dynamic network topologies rather than the average inter-node distance, as it is a more useful indicator of performance. However, the symbol ($\bar{d}$) is still

used to represent the average inter-PE distance as it is frequently compared with the average inter-node distance of static networks.

Theorem 2 *A hierarchical clique $HiC_{(k,h)}$ with $k > 1$ and $h > 0$ has average inter-PE distance*

$$\overline{d} = \frac{-\frac{2}{(k-1)}k^{h+1} + (2h+1)k^h - k^{h-1} + \frac{2}{(k-1)}k}{k^h - 1}.$$

Proof: Observe that the HiC topology is symmetric from the point of view of the leaves. Without loss of generality then, $\overline{d}$ can be determined by considering one leaf node, μ. The distance from μ to another leaf node ν is $d(\mu, \nu)$. We wish to determine $\sum d(\mu, \nu)$ for all ν at level 0.

- The sum of distances to μ's $(k-1)$ neighbours is $(k-1)$.
- At level l $(0 < l < h)$, μ has an ancestor u, which is the least common ancestor of μ and $(k-1)k^{(l-1)}$ leaf nodes. The distance from any of these leaf nodes to μ is $2l$, so the sum of the distances equals $2l(k-1)k^{(l-1)}$. Considering all leaf nodes which have a least common ancestor, other than the root,with μ, the sum of distances is $\sum_{l=1}^{h-1} 2l(k-1)k^{(l-1)}$.
- Each of u's $(k-1)$ neighbours at level l is the least common ancestor of one of μ's neighbours and $(k-1)k^{(l-1)}$ leaf nodes. The distance from any of these leaf nodes to μ is $2l+1$, so the sum of the distances equals $(k-1)^2k^{(l-1)}(2l+1)$. Considering all leaf nodes which have a least common ancestor, other than the root, with one of μ's neighbours, the sum of distances is $\sum_{l=1}^{h-1}(k-1)^2k^{(l-1)}(2l+1)$.

These three cases cover all possible leaf nodes. Their partial results can be combined

to give the expression for the total sum of distances.

$$\sum d(\mu,\nu) = (k-1) + (k-1)\sum_{l=1}^{h-1} 2lk^{(l-1)} + (k-1)^2 \sum_{l=1}^{h-1} k^{(l-1)}(2l+1) \quad (3.1)$$

$$= (k-1)(k^{(h-1)} + 2\sum_{l=1}^{h-1} lk^l) \quad (3.2)$$

$$= (k-1)k^{(h-1)} + \frac{2((h-1)k^{h+1} - hk^h + k)}{(k-1)} \quad (3.3)$$

$$= -\frac{2}{(k-1)}k^{h+1} + (2h+1)k^h - k^{h-1} + \frac{2}{(k-1)}k. \quad (3.4)$$

The total number of distances is $N_P - 1 = k^h - 1$,

$$\therefore \overline{d} = \frac{-\frac{2}{(k-1)}k^{h+1} + (2h+1)k^h - k^{h-1} + \frac{2}{(k-1)}k}{k^h - 1}$$

□

Values of average inter-PE distance for various configurations of $HiC_{(k,h)}$ are plotted in Figure 3.2. The plots clearly show the reduced average distance which is obtained in a system with a given number of PEs by increasing the value of k. This improvement is gained at the cost of extra links and extra switch complexity, which naturally increases the cost of any system. To justify the extra cost, algorithms must perform better with the extra PE connectivity provided by the extra links. The same argument applies to the GFT and n-cube topologies. Figure 3.2 shows both of these topologies have lower average inter-PE distance than HiC for reasonable numbers of PEs, but both use more links to achieve this end. Greater efficiency must be obtainable to justify the greater cost. This is discussed further in Chapter 5.

3.4 Fault-Tolerance

In this section the fault-tolerance of the HiC interconnection network is studied. The connectivity, fault diameter, two-terminal reliability and average two-terminal reliability of the HiC topology are determined. The results are compared with those of the binary hypercube. The binary hypercube is a fault-tolerant topology and its characteris-

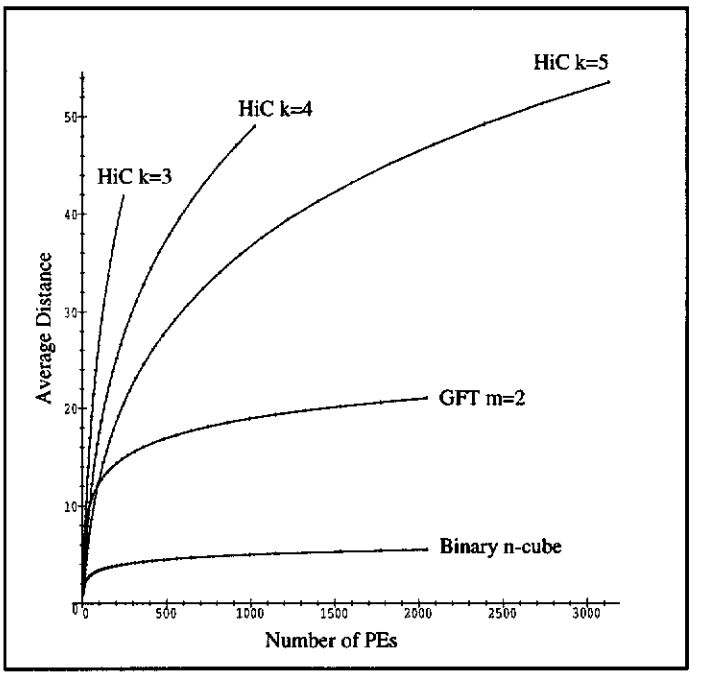


Figure 3.2: Average Distance between PEs

tics have been widely studied and reported [Saad and Schultz(1988), Esfahanian(1989), Tien and Raghavendra(1993)]. This makes it an ideal topology to use as a benchmark for fault-tolerance. In general, any system with fault-tolerance greater than or near to that of the binary hypercube can be considered to have acceptable fault-tolerance.

3.4.1 Connectivity

We first prove some theorems regarding basic graph theoretic properties of HiCs.

Theorem 3 *Graph degree* $\delta(HiC_{(k,h)}) = k$.

Proof: $\delta(G) = \min(\delta_\mu) : \forall \mu \in V(G)$. In $HiC_{(k,h)}$, degree $\delta_\mu = k$ when μ is the root node or any of the leaf nodes and $\delta_\mu = 2k$ when μ is any other node. □

Theorem 4 *Graph connectivity* $\kappa(HiC_{(k,h)}) = k$.

Proof: In $HiC_{(k,h)}$, k node-disjoint paths exist from any leaf μ to any other node via μ's parent and $k-1$ neighbours. The root node has k node-disjoint paths to any other node via its k children. Any node ν at level l, where $0 < l < h$, has k node-disjoint paths to descendants, or descendants of ν's neighbours, via ν's $k-1$ neighbours and the appropriate one of ν's children. Node ν has k node-disjoint paths to all other nodes via ν's parent and $k-1$ neighbours. In $HiC_{(k,h)}$, therefore, k node-disjoint paths exist between any two nodes. □

The $HiC_{(k,h)}$ is therefore $(k-1)$ node fault-tolerant.

Theorem 5 *Graph link-connectivity* $\lambda(HiC_{(k,h)}) = k$.

Proof: In $HiC_{(k,h)}$, k link-disjoint paths exist from any leaf μ to any other node via μ's parent and $k-1$ neighbours. The root node has k link-disjoint paths to any other node via its k children. Any node ν at level l, where $0 < l < h$, has k link-disjoint paths to descendants, or descendants of ν's neighbours, via ν's $k-1$ neighbours and the appropriate one of ν's children. Node ν has k link-disjoint paths to all other nodes via ν's parent and $k-1$ neighbours. In $HiC_{(k,h)}$, therefore, k link-disjoint paths exist between any two nodes. □

The $HiC_{(k,h)}$ is therefore $(k-1)$ link fault-tolerant.

A more accurate reflection of the practical fault-tolerance of the network is given by determining the probability of any fault set of κ nodes or λ links disconnecting the network [El-Amawy and Latifi(1991)].

Theorem 6 *The probability p of a given k-node fault set disconnecting an $HiC_{(k,h)}$ is*

$$p = \frac{k^h(2^k-2) + k^{(h-1)} - k(2^k-2) - 1}{k-1} \times \frac{(N-k)!k!}{N!}.$$

Proof: The total number F of k-node fault sets possible is $F = \binom{N}{k}$ which can be represented as $F = \frac{N!}{(N-k)!k!}$. Let T_l be the number of k-node fault sets which can occur

at level l which cause disconnection of the network. Let $T = \sum_{l=0}^{h} T_l$ be the total number of k-node fault sets which cause disconnection of the network. At level $l = h$ there is only the root node, therefore $T_h = 0$. At level $l = h - 1$ disconnection will occur only if all k nodes in a clique are faulty. There is only one clique at this level, therefore $T_{(h-1)} = 1$. At level l, where $0 < l < (h-1)$, disconnection of the network will occur if n $(0 < n \leq k)$ nodes in a clique are faulty, and the parents of the $(k-n)$ non-faulty nodes in the clique are faulty. The number of ways such a failure can occur at a particular clique is $2^k - 1$. The total number of cliques for $0 < l < (h-1)$ is $\sum_{l=1}^{h-2} k^{(h-l-1)} = \frac{(k^{(h-1)}-k)}{(k-1)}$ for $k \neq 1$. Therefore $\sum_{l=1}^{h-2} T_l = \frac{(2^k-1)(k^{(h-1)}-k)}{(k-1)}$. At level $l = 0$, disconnection of the network will occur if n $(0 < n < k)$ nodes in a clique are faulty, and the parents of the $(k-n)$ non-faulty nodes in the clique are faulty. The number of ways such a failure can occur at a particular clique is $2^k - 2$. The total number of cliques for $l = 0$ is $k^{(h-1)}$. Therefore $T_0 = k^{(h-1)}(2^k - 2)$. Summing terms we find:

$$T = T_h + T_{(h-1)} + \sum_{l=1}^{h-2} T_l + T_0 \tag{3.5}$$

$$= 1 + \frac{(2^k - 1)(k^{(h-1)} - k)}{(k-1)} + k^{(h-1)}(2^k - 2) \tag{3.6}$$

$$= \frac{k^h(2^k - 2) + k^{(h-1)} - k(2^k - 2) - 1}{k-1}. \tag{3.7}$$

The probability of any given k node fault set causing disconnection is given by T/F. □

3.4.2 Fault-Diameter

From Theorem 4 we have $\kappa(HiC_{(k,h)}) = k$. The fault diameter $f(HiC_{(k,h)})$ is the largest diameter of the network in the presence of a fault set of $k-1$ nodes. Determining $f(HiC_{(k,h)})$ requires the determination of $\max d(\mu, \nu)$ for each pair of PEs in $HiC_{(k,h)}$ given any $k-1$ node fault set.

In Section 2.2.2, $S(G)$ is defined as the set of all sets of two distinct nodes in $V(G)$. As we are concerned with the ability of a network to provide communication between PEs, we redefine $S(HiC_{(k,h)})$ as the set of all sets of two distinct leaf nodes in $V(HiC_{(k,h)})$.

The pairs of leaf nodes within $S(HiC_{(k,h)})$ can be classified according to the distance $d(\mu, \nu)$. We divide $S(HiC_{(k,h)})$ into three classes and prove a lemma on each.

Lemma 3 *If $d(\mu, \nu) = 1$ there are k node and link disjoint paths between μ and ν of length at most $d(\mu, \nu) + 2$.*

Proof: Nodes μ and ν have addresses $M = \langle M_0, \ldots, M_{h-1} \rangle$ and $N = \langle N_0, \ldots, N_{h-1} \rangle$ respectively. Let u be a leaf node which is a neighbour of μ and of ν.

i) There is one path of length 1:
   $\mu \rightarrow \nu$.

ii) There are $k - 2$ paths of length 2:
   $\mu \rightarrow$ one of $k - 2$ possible nodes $u \rightarrow \nu$.

iii) There is one path of length 3:
   $\mu \rightarrow \mu$'s parent $\rightarrow \nu$'s parent $\rightarrow \nu$.

□

Lemma 4 *If $d(\mu, \nu) > 1$ and $d(\mu, \nu)$ is ODD there are k node and link disjoint paths between μ and ν of length at most $d(\mu, \nu) + 1$.*

Proof: Nodes μ and ν have addresses $M = \langle M_0, \ldots, M_{h-1} \rangle$ and $N = \langle N_0, \ldots, N_{h-1} \rangle$ respectively. Let node u with address $U = \langle M_0, \ldots, M_{h-2}, N_{h-1} \rangle$ and node v with address $V = \langle N_0, \ldots, N_{h-2}, M_{h-1} \rangle$ be two other leaf nodes. Then u is a neighbour of μ and $lca(u, \nu)$ exists at level l, where $0 < l < h$. Also, v is a neighbour of ν and $lca(v, \mu)$ exists at level l, where $0 < l < h$. Let $\mathbf{u}$ be any leaf node other than u which is a neighbour of μ. Let $\mathbf{v}$ be a leaf node other than v which is a neighbour of ν such that $lca(\mathbf{v}, u)$ exists at level l.

i) There is one path of length $d(\mu, \nu)$:
   $\mu \rightarrow lca(\mu, v) \rightarrow v \rightarrow \nu$.

ii) There is one path of length $d(\mu, \nu)$:
$\mu \rightarrow u \rightarrow lca(u, \nu) \rightarrow \nu$.

iii) There are $k-2$ paths of length $d(\mu, \nu)+1$, each including:
$\mu \rightarrow$ one of $k-2$ possible nodes $\mathbf{u} \rightarrow \mathbf{v} \rightarrow \nu$.

□

Lemma 5 *If $d(\mu, \nu)$ is EVEN there are k node and link disjoint paths between μ and ν of length at most $d(\mu, \nu)+2$.*

Proof: Distance $d(\mu, \nu)$ is EVEN if and only if $lca(\mu, \nu)$ exists at level l, where $1 < l < h$. Let $\mathbf{u}$ be any leaf node which is a neighbour of μ. Let $\mathbf{v}$ be a leaf node which is a neighbour of ν such that $lca(\mathbf{v}, u)$ exists at level l.

i) There is one path of length $d(\mu, \nu)$:
$\mu \rightarrow lca(\mu, \nu) \rightarrow \nu$.

ii) There are $k-1$ paths of length $d(\mu, \nu)+2$ each including:
$\mu \rightarrow$ one of $k-1$ possible nodes $\mathbf{u} \rightarrow \mathbf{v} \rightarrow \nu$.

□

Lemma 6 *In an $HiC_{(k,h)}$ with $h \geq 2$, any two leaf nodes have at least k node disjoint paths between them of length $2h$ or less.*

Proof: From Lemma 3, if $d(\mu, \nu) = 1$ there are k node disjoint paths of maximum length 3 between μ and ν. For $h \geq 2$, $3 \leq 2h-1$.

From Theorem 1 $D(HiC_{(k,h)}) = 2h-1$, which implies that $D(HiC)$ is always ODD. Therefore $\max(d(\mu, \nu)) = D(HiC)$ if $d(\mu, \nu)$ is ODD. From Lemma 4, if $d(\mu, \nu)$ is ODD, there are k node disjoint paths of maximum length $d(\mu, \nu)+1$ between μ and

ν. Combining these two results, we see that nodes μ and ν with $d(\mu, \nu)$ ODD, have k node disjoint paths of maximum length $2h$ between them.

From Lemma 5 if $d(\mu, \nu)$ is EVEN there are k node disjoint paths between μ and ν of length at most $d(\mu, \nu) + 2$. But distance $d(\mu, \nu)$ is EVEN if and only if $lca(\mu, \nu)$ exists at level l, where $1 < l < h$. For $l < h$ path length $2l + 2$ is never greater than $2h$. Therefore there are k node disjoint paths of maximum length $2h$.

These three cases include all possible pairs of leaf nodes, hence the proof. □

Theorem 7 *A hierarchical clique $HiC_{(k,h)}$ has a fault diameter $f = 2h$ for $k > 2$ and $h > 1$.*

Proof: Let μ and ν be two leaf nodes at distance $d(\mu, \nu)$. From Theorem 1 we know that at least one pair of nodes μ, ν exists with distance $d(\mu, \nu) = D(HiC_{(k,h)}) = 2h - 1$. If $k > 2$ then a fault set of $k - 1$ nodes can break all the paths between μ and ν of length $2h - 1$. From Lemma 6 we see that a fault set of $k - 1$ nodes cannot increase $d(\mu, \nu)$ to more than $2h$. The largest diameter in the presence of a $k - 1$ node fault set is therefore $2h$. □

The $HiC_{(k,h)}$ is strongly resilient, since $f = D+1$. This indicates that even under maximally faulty conditions the performance of the $HiC_{(k,h)}$ will not be severely degraded.

In Figure 3.3 the fault diameter of the binary n-cube is compared with that of $HiC_{(4,h)}$ and $HiC_{(3,h)}$ for a range of network sizes. The fault diameter of the $HiC_{(3,h)}$ increases most rapidly with increasing network size, followed by the n-cube. The $HiC_{(4,h)}$ clearly has the lowest fault diameter for all network sizes considered. Given $HiC_{(k1,h1)}$ and $HiC_{(k2,h2)}$ with identical numbers of PEs, if $k1 > k2$ then $f(HiC_{(k1,h1)}) < f(HiC_{(k2,h2)})$.

3.4.3 Two-Terminal Reliability

Two-terminal reliability $R_2(G)$ is a probabilistic measure of the reliability of graph G, defined in Section 2.2.2. As we are concerned with the ability of a network to provide communication between processors, the two-terminal reliability for the HiC deals only with leaf nodes. The two-terminal reliability $R_2(HiC_{(k,h)})$ is defined as $\min(R_2(\mu,\nu)) : \forall(\mu,\nu) \in S(HiC_{(k,h)})$. In this section a lower bound for $R_2(HiC_{(k,h)})$ will be determined.

From Theorem 5, there are at least k link-disjoint paths between any given pair of leaf nodes μ and ν in $HiC_{(k,h)}$. Assume that link failures are statistically independent and occur randomly in time. A lower bound $\lfloor R_2(\mu,\nu) \rfloor$ can be determined by establishing the probability of at least one of the k link-disjoint paths between leaf nodes μ and ν being completely operational. Using Lemmas 3, 4 and 5, $\min(\lfloor R_2(\mu,\nu) \rfloor) : \forall(\mu,\nu) \in S(HiC_{(k,h)})$ can be obtained, a lower bound on $R_2(HiC_{(k,h)})$. Assume that all links have a probability p of being operational. From Section 2.2.2 we see that the probability

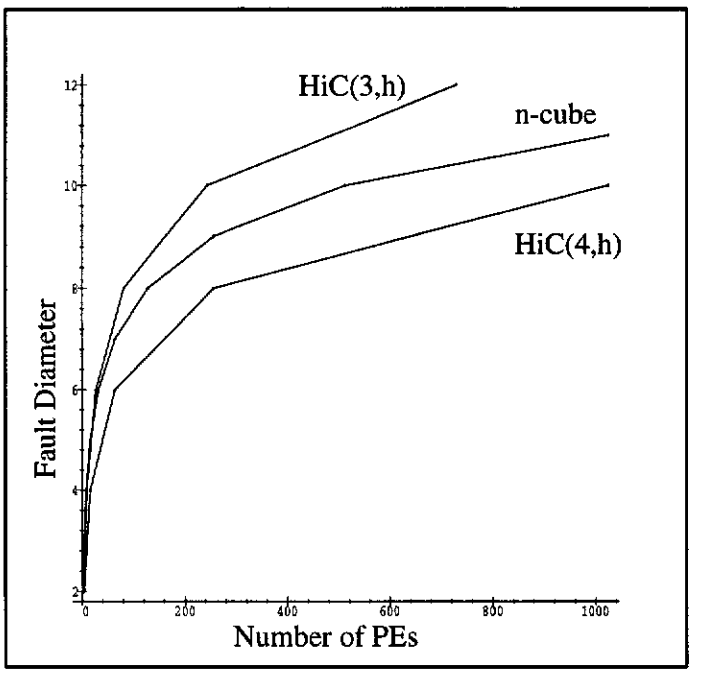


Figure 3.3: The Fault-Diameter of n-cube and HiC.

of at least one of i link disjoint paths of length l being operational is $1-(1-p^l)^i$.

Theorem 8 *If $h > 1$ then $\lfloor R_2(HiC_{(k,h)})\rfloor$ is given by $(1-(1-p^{D(HiC)+1})^k)$.*

Proof: From Theorem 1, $D(HiC_{(k,h)}) = 2h-1$, which implies that $D(HiC)$ is always ODD. Therefore $\max(d) = D(HiC)$ if d is ODD and $\max(d) = D(HiC)-1$ if d is EVEN.

From Lemma 5 we know that if $d(\mu,\nu)$ is EVEN there are k link-disjoint paths between μ and ν of length at most $d(\mu,\nu)+2$. Therefore a lower bound $\lfloor R_2(\mu,\nu)\rfloor$ is given by $(1-(1-p^{d(\mu,\nu)+2})^k)$. But $\max(d) = D(HiC)-1$ if d is EVEN. Therefore

$$\min(1-(1-p^{d+2})^k) = (1-(1-p^{D(HiC)+1})^k).$$

From Lemma 4 we know that if $d(\mu,\nu) > 1$ is ODD there are k link-disjoint paths between μ and ν of length at most $d(\mu,\nu)+1$. Therefore a lower bound $\lfloor R_2(\mu,\nu)\rfloor$ is given by $(1-(1-p^{d(\mu,\nu)+1})^k)$. But $\max(d) = D(HiC)$ if d is ODD. Therefore

$$\min(1-(1-p^{d+1})^k) = (1-(1-p^{D(HiC)+1})^k).$$

A lower bound $\lfloor R_2(HiC_{(k,h)})\rfloor$ is given by $\min(\lfloor R_2(\mu,\nu)\rfloor) : \forall(\mu,\nu) \in S(HiC_{(k,h)})$. Since $\min(\lfloor R_2(\mu,\nu)\rfloor)$ is given by $(1-(1-p^{D(HiC)+1})^k)$ if $d(\mu,\nu)$ is EVEN or if $d(\mu,\nu)$ is ODD and greater than one, and if $h > 1$, we conclude that $\lfloor R_2(HiC_{(k,h)})\rfloor$ is given by

$$(1-(1-p^{D(HiC)+1})^k).$$

□

3.4.4 Average Two-Terminal Reliability

Average two-terminal reliability $\overline{R_2(G)}$ was defined as $\frac{\sum R_2(\mu,\nu):\forall(\mu,\nu)\in S(G)}{|S(G)|}$ in Section 2.2.2, where $|S(G)| = \binom{|V(G)|}{2}$. In this section a lower bound on average two-terminal

reliability for the $HiC_{(k,h)}$ will be determined. A lower bound on the average two-terminal reliability of the binary hypercube will also be determined, and compared with that of the HiC for a range of system sizes.

A network is *symmetric* if it is isomorphic to itself with any node labelled as the origin. A symmetric network appears the same if viewed from the perspective of any node. Therefore all values of $R_2(\mu,\nu)$ can be determined by considering only those sets of nodes in $S(G)$ which contain a given node μ. Define $S(G,\mu) \subset S(G)$ as the set of all sets of two distinct nodes in $V(G)$ which contain node μ. For symmetric networks, then, we can redefine $\overline{R_2(G)}$ as $\frac{\sum R_2(\mu,\nu):\forall(\mu,\nu)\in S(G,\mu)}{|V(G)|-1}$. Symmetric networks have a further property however. Some constant k exists such that $R_2(\mu,\nu) = k : \forall(\mu,\nu) \in S(G)$ with a given distance d. Define $S(G,\mu,d) \subset S(G)$ as the set of all sets of two distinct nodes in $V(G)$ which contain node μ and have a given value of distance d. We can now redefine $\overline{R_2(G)}$ as

$$\frac{\sum((R_2(\mu,\nu):\forall(\mu,\nu)\in S(G,\mu,d))\times|S(G,\mu,d)|):\forall S(G,\mu,d)\in S(G,\mu)}{|V(G)|-1}. \quad (3.8)$$

Starting with the easy part, we note once again that for the HiC we are concerned only with the PEs, we therefore replace $|V(G)|-1$ with k^h-1.

While proving Theorem 8 we established lower bounds $\lfloor R_2(\mu,\nu)\rfloor$ for all pairs of distinct nodes $(\mu,\nu) \in S(HiC,\mu)$.

i) $\lfloor R_2(\mu,\nu)\rfloor$ is $(1-(1-p^3)^k) : \forall(\mu,\nu) \in S(HiC,\mu,d:1)$.

ii) $\lfloor R_2(\mu,\nu)\rfloor$ is $(1-(1-p^{d+2})^k) : \forall(\mu,\nu) \in S(HiC,\mu,d:EVEN)$.

iii) $\lfloor R_2(\mu,\nu)\rfloor$ is $(1-(1-p^{d+1})^k) : \forall(\mu,\nu) \in S(HiC,\mu,d:ODD>1)$.

It remains to determine $|S(HiC,\mu,d)|$ for each $S(HiC,\mu,d) \in S(HiC,\mu)$.

i) If $d(\mu,\nu)=1$ then μ and ν are neighbours, so

$$|S(HiC,\mu,d:1)| = k-1.$$

ii) If $d(\mu, \nu)$ is even then μ and ν share a common ancestor, so

$$|S(HiC, \mu, d : EVEN)| = k^{(\frac{d}{2})} - k^{(\frac{d-2}{2})}.$$

iii) If $d(\mu, \nu) > 1$ is odd then μ and ν are not in the same clique and do not share a common ancestor, so

$$|S(HiC, \mu, d : ODD > 1)| = (k^{\frac{d-1}{2}} - k^{\frac{d-3}{2}})(k-1).$$

Thus a lower bound $\lfloor\overline{R_2(HiC_{(k,h)})}\rfloor$ is made up of the sum of three terms. The first term comes from node pairs with a distance of 1.

$$Term1 = \frac{(k-1)}{(k^h-1)}(1-(1-p^3)^k)$$

The second term comes from node pairs with an even distance.

$$Term2 = \sum_{\frac{d}{2}=1}^{h-1}(1-(1-p^{d+2})^k)\frac{k^{(\frac{d}{2})} - k^{(\frac{d-2}{2})}}{k^h-1} \tag{3.9}$$

$$= \frac{(k-1)}{k(k^h-1)}\sum_{\frac{d}{2}=1}^{h-1} k^{\frac{d}{2}}(1-(1-p^{d+2})^k) \tag{3.10}$$

The third term comes from node pairs with an odd distance greater than 1.

$$Term3 = \sum_{\frac{d-1}{2}=1}^{h-1}(1-(1-p^{d+1})^k)\frac{(k^{\frac{d-1}{2}} - k^{\frac{d-3}{2}})(k-1)}{k^h-1} \tag{3.11}$$

$$= \frac{(k-1)^2}{k(k^h-1)}\sum_{\frac{d-1}{2}=1}^{h-1} k^{\frac{d-1}{2}}(1-(1-p^{d+1})^k) \tag{3.12}$$

$\lfloor\overline{R_2(HiC_{(k,h)})}\rfloor$ is $Term1 + Term2 + Term3$.

We now determine a lower bound $\lfloor\overline{R_2(Q)}\rfloor$ on average two-terminal reliability for a hypercube Q of dimension n. Since the hypercube is symmetric, the expression for $R_2(G)$ given in Formulae 3.8 is applicable. The total number of node pairs considered

is given by $|V(Q)| - 1 = 2^n - 1$. We can also determine that $|S(Q, \mu, d)| = \frac{n!}{(n-d)! \times d!}$. From Section 2.3.1 a lower bound $\lfloor R_2(\mu, \nu) \rfloor$ for a hypercube Q of dimension n is given by $1 - ((1 - p^d)^d (1 - p^{(d+2)}))^{(n-d)})$. Therefore a lower bound $\lfloor \overline{R_2(Q)} \rfloor$ is given by

$$\frac{n!}{(2^n - 1)} \sum_{d=1}^{n} \frac{(1 - ((1 - p^d)^d (1 - p^{(d+2)})^{(n-d)}))}{(n - d)! \times d!}.$$

3.4.5 Comparison

The lower bounds determined for the average two-terminal reliability of hypercube, folded hypercube, $HiC_{(4,h)}$ and $HiC_{(3,h)}$ are plotted against the number of PE's in Figure 3.4. The lower bounds on the two-terminal reliability of the hypercube and folded hypercube rise with network dimension. This is as expected, since the connectivity and node degree of a hypercube or folded hypercube also increase with dimension. The $HiC_{(k,h)}$ network, by contrast, has fixed connectivity and node degree, consequently the lower bound decreases with network size. The rate of decrease is much lower in

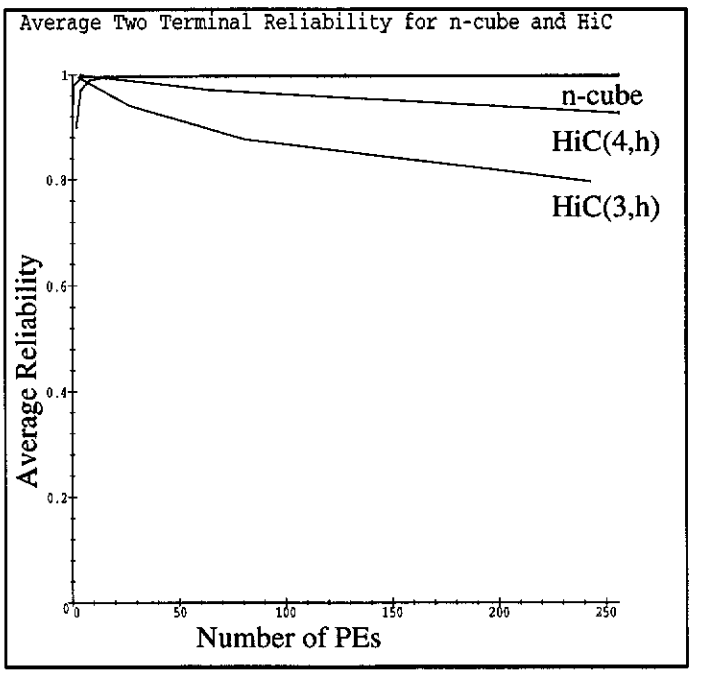


Figure 3.4: The average two-terminal reliability of hypercube and HiC.

the $HiC_{(4,h)}$ than in the $HiC_{(3,h)}$. Further increasing the value of k leads to higher bounds for the value of average two-terminal reliability, but at the cost of increased node degree. An $HiC_{(4,4)}$ has 256 PEs, node degree of 8 and an average two-terminal reliability of over 0.9 with an operational link probability of $p = 0.9$. With the same operational link probability, a hypercube of the same size has an average two-terminal reliability of nearly 1 but node degree of 16.

3.5 Message Routing Algorithms

3.5.1 One to One Communication

PEs and SEs handle message routing in different ways. PEs either source or receive messages, SEs serve only as intermediate nodes for PE to PE communication. Let the address of the current node be $\langle C_l, \ldots, C_{h-1} \rangle$ where $(0 \leq l < h)$ and the destination address be $\langle D_0, \ldots, D_{h-1} \rangle$.

A PE sending a message compares $\langle C_0, \ldots, C_{h-2} \rangle$ with $\langle D_0, \ldots, D_{h-2} \rangle$. If they are not the same the message is routed to the PE's parent. If they are the same then the destination PE is a neighbour and the message is passed directly. This is expressed formally in Algorithm 1.

```
if ⟨C_0, ..., C_{h-2}⟩ = ⟨D_0, ..., D_{h-2}⟩ then
  Route message to neighbouring PE ⟨D_0, ..., D_{h-1}⟩
else
  Route message to parent
```

Algorithm 1: HiC PE message routing algorithm

A SE at level l has address $\langle C_l, \ldots, C_{h-1} \rangle$ for $(0 < l < h-1)$. Upon receipt of a message a SE compares its own address with the corresponding digits from the destination node address. If they are the same the SE is an ancestor of the destination node, and the message can be routed to the child node which is either the destination or an ancestor of the destination. Otherwise, if the addresses differ only in the rightmost digit, then a

neighbour of the SE is an ancestor of the destination node, and the message is routed via that neighbour. In any other case the message is routed further up the hierarchy, to the SEs' parent. An SE at level $l = (h-1)$ has address (C_l). If $(C_l) = (D_l)$ then the SE is an ancestor of the destination node, and the message is routed to the child node which is the destination or an ancestor of the destination. Otherwise the message is routed to the neighbour of the SE which is an ancestor of the destination node. This is expressed formally in Algorithm 2.

```
if (0 < l < h − 1) then
  if ⟨C_l, ... , C_{h−1}⟩ = ⟨D_l, ... , D_{h−1}⟩ then
    Route message to child ⟨D_{l−1}, ... , D_{h−1}⟩
  else
    if ⟨C_l, ... , C_{h−2}⟩ = ⟨D_l, ... , D_{h−2}⟩ then
      Route message to neighbouring SE ⟨D_l, ... , D_{h−1}⟩
    else
      Route message to parent
else
  if (l = h − 1) then
    if ⟨C_l⟩ = ⟨D_l⟩ then
      Route message to child ⟨D_{l−1}, D_l⟩
    else
      Route message to neighbouring SE ⟨D_l⟩
```

Algorithm 2: *HiC* SE message routing algorithm

3.5.2 One to Many Communication

The *HiC* broadcast algorithms are very simple. The broadcast algorithm used by the source PE, with address $\langle S_0, \ldots, S_{h-1} \rangle$, is trivial. The source PE routes the message only to its parent. For a SE at level l, where $(0 < l \leq h-1)$, the broadcast algorithm is shown in Algorithm 3. Note that the message is transmitted to neighbours only at level $l = h-1$. At all other levels only parent or child links are used. This is to prevent PEs receiving duplicate messages.

```
if l = h − 1 then
  Send message to all neighbours
else
  if ⟨C_l, ..., C_{h−1}⟩ = ⟨S_l, ..., S_{h−1}⟩ then
    Send message to parent and all children such that
    ⟨C_{l−1}, ..., C_{h−1}⟩ ≠ ⟨S_{l−1}, ..., S_{h−1}⟩
  else
    Send message to all children
```

Algorithm 3: *HiC* SE broadcast algorithm

3.5.3 Fault-Tolerant Communication

Fault-tolerant routing is always a compromise; routing algorithms must always offer high performance and therefore demand simplicity, yet determining a route in the presence of many faults may entail complexity. The intention here is to demonstrate that the *HiC* topology allows alternative routes to be explored in a simple and systematic way. Two separate algorithms are described. A method for fault-tolerant routing of messages at the PE level is given in Algorithm 4. Messages which cannot be routed directly to their destination are always routed up to the parent if possible. The aim is to ensure that the bulk of routing is carried out in the SEs'. A fault-tolerant method of routing messages within SEs is given in Algorithm 5.

In order to allow message routing to take place in a network with faulty nodes and links, the routing node requires information about the status of other nodes and the links to them. Assume that a routing node is aware of the status of all nodes directly connected to it; parents, children or neighbours. A node is defined to be *accessible* if both the node and the direct link to it are known to be functioning. Let the address of the current node be $C = \langle C_l, \ldots, C_{h-1} \rangle$. Let the address of the current node's parent be $P = \langle C_{l+1}, \ldots, C_{h-1} \rangle$. Let the address of the destination node be $D = \langle D_0, \ldots, D_{h-1} \rangle$.

```
if D is not a neighbour then
  if P is accessible then
    Route message to P.
  else
    if a neighbour is accessible then
      Route message to that neighbour.
    else
      Destination unreachable
else
  if D is accessible then
    Route message to D.
  else
    if P is accessible then
      Route message to P.
    else
      if a neighbour is accessible then
        Route to that neighbour.
      else
        Destination unreachable
```

Algorithm 4: *HiC* PE fault-tolerant message routing algorithm

```
if D is a descendant then
  if the child in the line of descent is accessible then
    Route message to that child
  else
    if a neighbouring SE is accessible then
      Route message to that neighbour
    else
      Destination unreachable
else
  if D is a descendant of a neighbour then
    if that neighbour is accessible then
      Route message to that neighbour
    else
      if that child which has a neighbour in the line of descent is accessible then
        Route message to that child
      else
        if any other neighbouring SE is accessible then
          Route message to that neighbour
        else
          Destination unreachable
  else
    if parent is accessible then
      Route message to parent
    else
      if a neighbouring SE is accessible then
        Route message to that neighbour
      else
        Destination unreachable
```

Algorithm 5: *HiC* SE fault-tolerant message routing algorithm

3.6 Network Embeddings

The ability of a parallel computer's interconnection network to emulate other networks is important if it is to effectively employ algorithms and data structures developed for different parallel architectures.The quality of such an emulation can be determined by studying embeddings of one topology into another. Let G and H be undirected graphs. G represents a guest graph and H represents a host graph. Using the terminology of [Monien and Sudborough(1990)] an embedding of G into H is a mapping $\mathcal{F}$ from the nodes of G to the nodes of H. The mapping from any node g in G to a node h in H is represented by $\mathcal{F}(g) \equiv (h)$. The *dilation* of an embedding $\mathcal{F}$ is the maximum distance in the host between the images of adjacent guest nodes. The *expansion* of the embedding $\mathcal{F}$ is the ratio of the number of nodes in the host graph to the number of nodes in the guest graph, i.e. $|V(H)|/|V(G)|$. The number of edges of G routed through edge $e \in E(H)$ in embedding $\mathcal{F}$ is $c(e)$. The *edge congestion* of $\mathcal{F}$ is $\max(c(e)) : \forall e \in E(H)$.

Binary structures such as binary trees, binary hypercubes or meshes, map most naturally onto the $HiC_{(k,h)}$ when k is a power of two. Cost factors, discussed in Chapter 5, show $k = 4$ to be a desirable value. In the following work, all mappings are made to $HiC_{(4,h)}$.

3.6.1 Binary Trees

For binary trees, two different embedding strategies are possible. If the binary tree is considered as a dynamic network with only leaf nodes as PEs, then SEs of the binary tree can be mapped to SEs of the HiC. Alternatively if considered as a task graph or data structure, then every node must be mapped to a PE in the host HiC. In the following work both cases are considered.

First consider the case of a dynamic tree network. For any $t \geq 0$, the complete binary

tree $B(t)$ of height t can be embedded in $HiC_{(4,h)}$ where $h = \lfloor \frac{t+2}{2} \rfloor$. The embedding can be described in the following manner. For binary tree $B(t)$, the root node is at level zero, the leaf nodes at level t. The root node is represented by $(0,0)$. The children of the root node are represented by $(1,0)$ and $(1,1)$. In a binary tree, b_j is a node at level j, where $(t \geq j \geq 1)$. For $HiC_{(4,h)}$ the address of a node μ at level l is represented by $M = \langle M_l, \ldots, M_{h-1} \rangle$, the address of a node ν at level l is represented by $N = \langle N_l, \ldots, N_{h-1} \rangle$.

$\mathcal{F}(0,0) \equiv (1)$
$\mathcal{F}(1,0) \equiv (2)$
$\mathcal{F}(1,1) \equiv (3)$
for $j = 1 \rightarrow t$ **do**
  if j is ODD **then**
    if $\mathcal{F}(b_j) \equiv \mu$ such that M_{h-1} is ODD **then**
      $\mathcal{F}(b_j$'s first child$) \equiv \nu$ such that $N = \langle M_{h-1}, M \rangle$
      $\mathcal{F}(b_j$'s second child$) \equiv \nu$ such that $N = \langle (M_{h-1}+1), M \rangle$
    else
      $\mathcal{F}(b_j$'s first child$) \equiv \nu$ such that $N = \langle M_{h-1}, M \rangle$
      $\mathcal{F}(b_j$'s second child$) \equiv \nu$ such that $N = \langle (M_{h-1}-1), M \rangle$
  else
    if $\mathcal{F}(b_j) \equiv \mu$ such that M_{h-1} is EVEN **then**
      $\mathcal{F}(b_j$'s first child$) \equiv \nu$ such that $N = \langle M_l, \ldots, M_{h-2}, 1 \rangle$
      $\mathcal{F}(b_j$'s second child$) \equiv \nu$ such that $N = \langle M_l, \ldots, M_{h-2}, 3 \rangle$
    else
      $\mathcal{F}(b_j$'s first child$) \equiv \nu$ such that $N = \langle M_l, \ldots, M_{h-2}, 2 \rangle$
      $\mathcal{F}(b_j$'s second child$) \equiv \nu$ such that $N = \langle M_l, \ldots, M_{h-2}, 4 \rangle$

Algorithm 6: First Binary Tree Embedding

$B(t)$ is a subgraph of $HiC_{(4,h)}$, and can therefore be embedded with both dilation and edge congestion of one. The expansion is $\frac{1}{2^{t+1}-1} \sum_{l=0}^{h} 4^{h-l} = \frac{4^{h+1}-1}{3(2^{t+1}-1)}$.

Now consider the case of a binary tree structure where each node of the tree must be mapped to a PE in a HiC. For any $t \geq 0$, the complete binary tree $B(t)$ of height t can be embedded in $HiC_{(4,h)}$ where $h = \lfloor \frac{t+2}{2} \rfloor$. The embedding can be described in the following manner. For binary tree $B(t)$, the root node is at level zero, leaf nodes at level t. The root node is represented by $(0,0)$. The children of the root node are represented by $(1,0)$ and $(1,1)$. In a binary tree, b_j is a node at level j, where $(t \geq j \geq 1)$. For $HiC_{(4,h)}$ the address of a node μ at level l is represented by $M = \langle M_l, \ldots, M_{h-1} \rangle$,

the address of a node ν at level l is represented by $N = \langle N_l, \ldots, N_{h-1} \rangle$. A sample embedding is shown in Figure 3.5.

$\mathcal{F}(0,0) \equiv \mu$ with address $M = \langle M_0, \ldots, M_{h-1} \rangle$ such that $M_x = 1$ for $(0 \leq x \leq h-1)$.
$\mathcal{F}(1,0) \equiv \nu$ such that $N = \langle M_0, \ldots, M_{h-2}, 2 \rangle$
$\mathcal{F}(1,1) \equiv \nu$ such that $N = \langle M_0, \ldots, M_{h-2}, 4 \rangle$
$\mathcal{F}(b_1$'s first child$) \equiv \nu$ such that $N = \langle (M_0 + 2), \ldots, M_{h-1} \rangle$
$\mathcal{F}(b_1$'s second child$) \equiv \nu$ such that $N = \langle M_0 + 2), \ldots, (M_{h-1} - 1) \rangle$
for $j = 2 \rightarrow t$ **do**
  if j is EVEN **then**
    $\mathcal{F}(b_j$'s first child$) \equiv \nu$ such that $N = \langle M_0, \ldots, (M_{(j/2)-1} - 1), \ldots, M_{h-1} \rangle$
    $\mathcal{F}(b_j$'s second child$) \equiv \nu$ such that $N = \langle M_0, \ldots, (M_{(j/2)-1} + 1), \ldots, M_{h-1} \rangle$
  else
    $\mathcal{F}(b_j$'s first child$) \equiv \nu$ such that $N = \langle M_0, \ldots, (M_{(j/2)-1} - 1), (M_{(j/2)} + 2), \ldots, M_{h-1} \rangle$
    $\mathcal{F}(b_j$'s second child$) \equiv \nu$ such that $N = \langle M_0, \ldots, (M_{(j/2)} + 2), \ldots, M_{h-1} \rangle$

Algorithm 7: Second Binary Tree Embedding

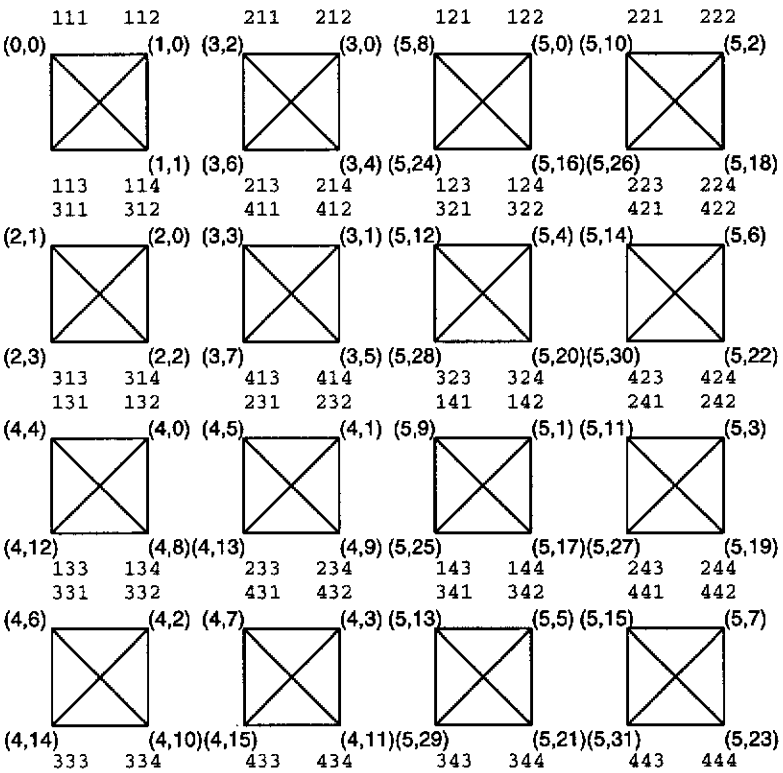


Figure 3.5: An embedding of a binary tree.

This embedding has:

dilation $2h-1$

expansion $\frac{4^{h+1}-1}{3(2^{t+1}-1)}$

edge congestion
$$\begin{cases} t & : \quad t < 4 \\ 4^{t-h} & : \quad t \quad EVEN \geq 4 \\ 3 \times 4^{t-h-1} & : \quad t \quad ODD \geq 4. \end{cases}$$

3.6.2 Binary Hypercubes

A binary hypercube $Q(n)$ can be embedded into $HiC_{(4,\lfloor\frac{n+1}{2}\rfloor)}$. For binary hypercube $Q(n)$ the address of a node q can be represented as a binary string $(q_{n-1},\ldots,q_0)$, [Monien and Sudborough(1990)]. For any binary string $(q_x,\ldots,q_{x-y})$, where $y \leq x$, we represent the decimal value of the binary number $(q_x,\ldots,q_{x-y})$ by $(q_x,\ldots,q_{x-y})_{10}$. For $HiC_{(4,h)}$ the address of a leaf node μ is represented by $M = \langle M_0,\ldots,M_{h-1}\rangle$. The embedding can be described as follows.

```
for all q ∈ Q(n) do
  if n is EVEN then
    F(q) ≡ μ
    for i = 0 to ⌊n/2 − 1⌋ do
      M_i = (q_{n−2i}, q_{n−2i−1})_10 + 1
  else
    F(q) ≡ μ
    M_{h−1} = (q_1, q_0)_10 + 1
    M_{h−2} = (q_{n−1})_10 + 1
    for i = 1 to ⌊n/2 − 1⌋ do
      M_{i−1} = (q_{n−2i}, q_{n−2i−1})_10 + 1
```

Algorithm 8: Binary Hypercube Embedding

This embedding has:

dilation $2(h-1)$

edge congestion 2^{n-3}

expansion $\frac{4^{h+1}-1}{3.2^n} = \begin{cases} 1 & : \quad n \quad EVEN \\ 2 & : \quad n \quad ODD. \end{cases}$

The dilation of this embedding is derived in the following way. A node q of $Q(n)$ has address $(q_{n-1}, \ldots, q_0)$. If a node p of $Q(n)$ is one of the n nodes directly connected to q then p's address differs from that of q in that one bit q_x is replaced by $\overline{q_x}$, where $0 \leq x \leq n-1$. From Algorithm 8 it is clear that, if $\mathcal{F}(q) \equiv \mu$ and $\mathcal{F}(p) \equiv \nu$, then the address of ν differs from that of μ in that one digit M_y is replaced by $M_y \pm 1$, where $0 \leq y \leq h-1$. If $y = h-1$ then μ and ν are neighbours. If $0 \leq y < h-1$ then $lca(\mu, \nu)$ occurs at level $y+1$. The maximum distance between adjacent nodes of $Q(n)$ is therefore $2(h-2+1) = 2(h-1)$.

3.6.3 Two Dimensional Meshes

Mesh M of dimension $2^h \times 2^h$ can be embedded into $HiC_{(4,h)}$. The embedding can be described in the following manner. A node of mesh M is identified as (x, y), where $0 \leq x, y < 2^h$. For $HiC_{(4,h)}$ the address of a leaf node μ at level l is represented by $M = \langle M_0, \ldots, M_{h-1} \rangle$. A sample embedding is shown in Figure 3.6.

for $x = 0 \rightarrow 2^h - 1$ **do**
  for $y = 0 \rightarrow 2^h - 1$ **do**
    $\mathcal{F}(x, y) \equiv \mu$ such that $M_{h-1} = (2\lceil y/2 \rceil + 3/2 + 1/2(-1)^{\lceil x/2 \rceil + 1}) \bmod 4$
    for $i = 1 \rightarrow h - 1$ **do**
      $M_{(i-1)} = (2\lfloor \frac{y}{2^i} \rfloor + 3/2 + 1/2(-1)^{\lfloor \frac{x}{2^i} \rfloor + 1}) \bmod 4$

Algorithm 9: 2D mesh Embedding

This embedding has *dilation* $2(h-1)$ and *edge congestion* 2^{h-1}.

3.6.4 Embeddings into the $FatHiC$

In all the embeddings considered above the edge congestion grows rapidly. This indicates congestion will occur in higher level communication channels, under uniform communication conditions, with these embeddings into the HiC. This problem has already been extensively studied, since this is possibly the most fundamental problem

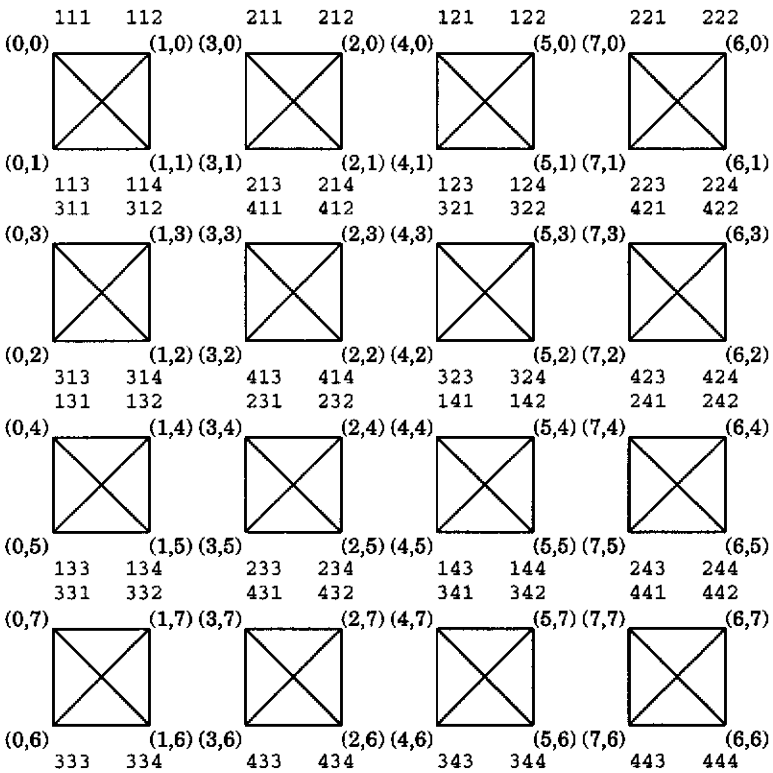


Figure 3.6: An embedding of a 2D mesh.

with tree based networks. A well accepted solution has been proposed, the Fat tree of [Leiserson(1985)]. In order to prevent congestion in the upper levels dominating communication time for regular algorithms involving a large proportion of PEs transmitting across the diameter of the network simultaneously, the bandwidth of links in upper layers is increased. Applying the same principle, define a $FatHiC_{(k,h)}$ as being identical with an $HiC_{(k,h)}$, except that the bandwidth of parent-child links increases with the level of the nodes incident to them. The rate of increase of bandwidth required is dictated by the cost and performance requirements of the system; the higher the bandwidth of the links the higher the expected performance, but at a higher cost. While considering embeddings into the $FatHiC_{(k,h)}$ the link bandwidth will be assumed to increase by a factor of k at each level of the network. For example, a $FatHiC_{(4,3)}$ would have links of bandwidth B from leaf nodes at level zero to their parents at level one, links of bandwidth $4B$ from nodes at level one to their parents at level two, and links of bandwidth $16B$ from nodes at level two to the root node. Assuming that the cost of

the extra bandwidth is approximated by representing each link of bandwidth kB as k separate links, the number of links in this version of a $FatHiC_{(k,h)}$ is $L = k^h(\frac{k}{2}+h)-\frac{k}{2}$.

Since $FatHiC_{(k,h)}$ is topologically identical to $HiC_{(k,h)}$ any network which can be embedded into $HiC_{(k,h)}$ can be embedded into $FatHiC_{(k,h)}$ in exactly the same manner, with the same dilation and expansion. The congestion may be reduced. All embeddings considered in the previous sections were into $HiC_{(4,h)}$, these same embeddings are used here into $FatHiC_{(4,h)}$, and the congestion is determined.

Considering the second binary tree embedding of Section 3.6.1, the congestion of the embedding is 3 for $t > 2$.

Considering the binary hypercube embedding of Section 3.6.2, the congestion of the embedding is 2.

Considering the two dimensional mesh embedding of Section 3.6.3, the congestion of the embedding is 1.

3.6.5 Embedding Quality

In order to provide an indication of the relative quality of these embeddings, Table 3.1 shows embedding parameters for $HiC_{(k,h)}$, $FatHiC_{(k,h)}$ and another hierarchical network, the generalised fat trees $GFT_{(h,m,w)}$ [Öhring *et al.*(1995)]. The GFT's were discussed in Section 2.3.3. All the results given here are for $GFT_{(h,2,2)}$. This configuration has a total number of nodes (PEs and SEs) $N(GFT) = (h+1) \times 2^h$ and a number of links $L(GFT) = h(2^{h+1})$.

For each of the guest graphs studied, the dilation and expansion of the embeddings into each of the host graphs are of the same order. In general, the dilation of the GFT is higher by some small factor, indicating possible higher latency for communications between mapped nodes. For each embedding the edge congestion of the HiC grows

3.7 Conclusion

This chapter described the topology and addressing scheme of the HiC and $FatHiC$ interconnection networks. Simple, efficient message passing algorithms were also described. These elements together allow a functional parallel computer to be based upon either interconnection network.

Some fundamental parameters of the networks, commonly used as indicators of performance, were derived. Diameter is slightly superior to other hierarchical networks, such as k-ary tree and GFT. The average inter-PE distance was shown to decrease with increasing values of k, at the cost of higher node degree and more links in the network. A number of standard network parameters commonly used as indicators of network fault tolerance were derived, including degreeδ, connectivity κ and link connectivity λ, as was the probability of network disconnection caused by a set of κ nodes or λ links. The fault diameter was derived; the $HiC_{(k,h)}$ was shown to be strongly resilient, with a fault diameter similar to that of the hypercube for a similar number of PEs. Lower bounds for the two-terminal reliability and average two-terminal reliability were calculated. The lower bound on the hypercube's average two-terminal reliability was found to be higher than that of the HiC for large numbers of PEs, as a result of its much higher, and increasing, degree. The HiC has fixed degree for greater scalability and lower cost. Simple, efficient routing algorithms were presented, and a general scheme for fault-tolerant routing was outlined. Finally, network embeddings for the binary tree, hypercube and $2D$-mesh into the $HiC_{(4,h)}$ were described. While expansion and dilation of the embeddings was low, the congestion was high. The $FatHiC_{(k,h)}$ was introduced, and shown to embed the three networks with low congestion. Results were compared with embeddings into a GFT, and the $FatHiC$ was shown to give superior embeddings.

Having described the topology, addressing scheme, routing algorithms, fault tolerance

and embedding capability of the *HiC* and *FatHiC* networks, the next chapter will examine the expected performance of the networks.

Chapter 4

Performance

4.1 Introduction

This chapter considers the performance of the *HiC*. The primary reason for using a parallel computing system is to increase performance. However, even in serial computers specification of performance is not a simple matter. The effects of caching strategy, bus speeds, memory hierarchy etcetera mean that different architectures perform differently with the same algorithms and problem sizes. Algorithmic complexity theory and benchmarking represent two approaches to dealing with this problem. In a parallel computing environment the issue is even more complicated. Problems may contain a number of potential sources of parallelism and parallel architectures may be suited to exploiting one or more of those sources. Performance of a parallel architecture can only be accurately determined for a particular algorithm and problem size due to differences in data mapping and communication requirements and speed.

A number of performance measures have been developed and used for parallel systems. Each has strengths and weaknesses. In order to give as broad an indication as possible of the performance capabilities of the *HiC*, several of the more commonly used

measures are applied in this chapter. All of them rely on characterising a combination of architecture and algorithm as a single parallel system. In this chapter simple, regular algorithms which are representative of a large class of algorithms are used for all analyses in order to make the results as meaningful as possible.

Network performance is often characterised in terms of *throughput* and *latency*. The network throughput τ is defined as the sustained data delivery rate given an applied load. The network latency is the average time spent by a message in the network. It does not include source queueing delay. By measuring or calculating the value of these parameters for appropriate loads and communication patterns it is possible to gain an understanding of the general behaviour of the network and to predict communication patterns for which the network will be particularly suited.In this chapter simulations of the network running with artificially determined loads are used to allow analysis of the throughput and latency characteristics of the HiC.

Parallel execution time, speedup and efficiency are three widely used system performance metrics. Parallel execution time T_p is the time elapsed from the moment a parallel computation starts to the moment the last processor finishes execution. Speedup S has been defined in a number of different ways, but the most widely accepted definition is the ratio of the serial execution time of the fastest known serial algorithm (T_s) to the parallel execution time of the chosen parallel algorithm (T_p), given that the processor used in the sequential computer is identical to those used for the parallel implementation. Efficiency E is the ratio of speedup (S) to the number of processors (p). Thus $E = \frac{T_s}{pT_p}$. All of the above metrics are dependent upon parallel architecture, parallel algorithm and problem size. The ultimate goal for a given algorithm and problem size is generally to reduce parallel execution time as much as possible. However a parallel algorithm is often more expensive to develop than a sequential one and a parallel computer more expensive than a uni-processor. The speedup indicates the benefit of parallelisation, while the efficiency indicates the degree of utilisation of

the processors available. Efficiency can therefore be thought of as providing a simple cost benefit measure for a parallel system. More sophisticated measures of cost are available and will be discussed in Chapter 5. Speedup relates performance to both the number of processors in the architecture and the problem size. The isoefficiency function determines the increase in problem size or workload W necessary to maintain constant efficiency as the number of processors p increases [Gupta and Kumar(1993)]. The workload W is defined as the complexity of the serial algorithm, therefore $W = T_s$. Parallel overhead time T_o is all time spent by processors not working directly on solving the problem, therefore $T_o = pT_p - T_s$. Substituting into the expression for efficiency, the result is:

$$E = \frac{W}{pT_p} \tag{4.1}$$

$$= \frac{W}{T_o + W} \tag{4.2}$$

$$= \frac{1}{1 + T_o/W}. \tag{4.3}$$

From Equation 4.3, efficiency will remain constant if $W = KT_o$, where K is a constant of proportionality. Thus isoefficiency occurs when the workload increases at a rate sufficient to overcome the increase in system overheads caused by an increase in the number of processors. Since the overheads are dependent upon the algorithm being executed, isoefficiency is a measure of the scalability of a particular algorithm on a particular architecture. All of these metrics provide useful information about a single system. They also allow comparison between systems. Thus proposed new systems can be compared in a meaningful way with existing, established systems. In this chapter the widely studied $2D$-mesh is analysed under identical conditions to the HiC and their performances compared.

In Section 4.2 the HiC interconnection network is studied in terms of throughput and latency, using a simulation of the network under different traffic patterns and loads. In Section 4.3 measures of performance of the systems formed by Floyd's all-pairs

shortest path algorithm and the HiC and $2D$-mesh networks are derived and compared. Section 4.4 compares the measured performance of simulations of Floyd's algorithm on both networks with the performance predicted by analysis. In Section 4.5 measures of performance of the systems formed by the *ascend/descend* class of algorithms and the HiC, $FatHiC$ and $2D$-mesh networks are derived and compared. A weakness of the HiC network under the communication pattern required by the *ascend/descend* class of algorithms is identified, and the $FatHiC$ is shown to overcome the problem. The $FatHiC$ is also shown to perform better than the $2D$-mesh. Section 4.6 validates the analytical results by measuring communication times from simulations of the systems. Section 4.7 concludes the chapter.

4.2 Throughput and Latency of the HiC

In this Section a simulation of a $HiC_{(4,4)}$ with 256 PEs is discussed and its performance assessed in terms of two parameters, throughput and latency.

While maximum potential throughput is obviously important, the saturation point and the behaviour after saturation are also significant. Saturation can occur at different load levels depending on traffic patterns and represents the maximum achievable throughput for a given traffic pattern. The behaviour of the network when the applied load exceeds the saturation point is important, as instability can result leading to degraded performance. It is desirable for the throughput to remain stable after saturation, whether due to bursty or sustained demand.

The network latency is the average time spent by a message in the network. It does not include source queueing delay. Factors which can affect latency include message size, network size and the number of messages being sent. Low latency is desirable, as is stability.

4.2.1 Traffic Patterns

In the simulations each PE generates messages independently according to a uniform distribution and the destinations are chosen according to the following traffic patterns.

- *Uniform Traffic.* Destinations are chosen randomly, with equal probability between PEs.
- *Localised Traffic.* Destinations are chosen according to a weighted probability: 25% of traffic is addressed to a neighbour, (three PEs) 25% of traffic is directed to PEs at a distance of two or three, (twelve PEs) 25% of traffic is directed to PEs at a distance of four or five (forty eight PEs) and 25% of traffic is directed to PEs at a distance of six or seven (one hundred and ninety two PEs).

The uniform traffic pattern is a commonly used benchmark in network routing studies [Petrini and Vanneschi(1997)]. It is representative of well balanced shared memory computations. The localised traffic pattern illustrates the capabilities of the network under the sort of traffic which occurs when algorithms involve relatively intense local communication and less frequent communications to distant PEs.

4.2.2 Simulation Results

We observe the behaviour of a simulated $HiC_{(4,4)}$ with 256 PEs. The unidirectional bandwidth of the links equals the message length. For simplicity store and forward flow control was used and infinite buffers were assumed. Each simulation runs for 3500 cycles in order to reach steady state, then performance data is collected for 1500 cycles. The simulation results are presented according to the Chaos Normal Form [1] (CNF). Chaos Normal Form comprises a pair of graphs. The first displays throughput versus applied

[1] See http://www.cs.washington.edu/research/projects/lis/chaos/www/presentation.html

load, the second displays latency versus applied load. In both graphs the applied load is expressed as normalised bandwidth. Normalised bandwidth expresses the load as a fraction of the bisection bandwidth limited maximum load before saturation for uniform random traffic. The bisection bandwidth is not the limiting factor in an *HiC*. Rather we calculate the injection rate which causes saturation of the links between neighbours in the top level clique under uniform random traffic. Under uniform random traffic conditions, 75% of the messages from a PE are passed up to the top level clique to be rerouted to their destination. Of the messages arriving at a top level SE, 25% are routed to each of the three neighbours, the remaining 25% are routed down to child nodes. For an $HiC_{(4,4)}$, if each PE injects one message per cycle there are twelve messages per cycle passing through each link in the top level clique. For bandwidth of one message per cycle the load which causes saturation of the top level links is 0.083 messages injected per PE per cycle. All load and throughput figures are normalised to this limit.

4.2.2.1 Uniform Traffic

Referring to Figure 4.1, it can be seen that the normalised throughput equals the normalised load until the top level links saturate, at a normalised applied load of 1. This is an optimal result. From Figure 4.2 it can be seen that the latency also remains very low (average latency of 25.1 at normalised load of 0.9) until saturation occurs. After saturation the rate of increase of throughput declines, but the spare capacity of lower levels in the network ensure that throughput continues to increase. The latency rises sharply after saturation, but the rate of increase levels off. This is due to the increased delivery times of messages delivered over congested links being partially counteracted by the increased number of messages delivered over uncongested links. A further drop in the rate of increase of throughput occurs when the normalised load exceeds 3. This is in accordance with expectation, as the links leading up to the second top level SEs also begin to saturate at this load level. Again latency begins

to increase, but the system remains stable. Note that the simulation assumed infinite buffers. Thus for loads which cause saturation, average latency will continue to rise as long as the load is maintained. The figures given are indicative of the relative levels of latency at the corresponding loads, but absolute values depend on the number of cycles during which measurements are made.

4.2.2.2 Localised Traffic

Figure 4.3 shows that the network performance with a localised traffic pattern is improved, but the behaviour is more complex. The normalised throughput equals the normalised load until a little before the top level links saturate, at a normalised load of 3, which is nearly optimal. Figure 4.4 shows that latency remains very low until about 70% of the load which causes saturation. Latency then rises, but the rate of increase starts to decline after saturation, due to the increasing number of local messages being delivered. Throughput continues to increase, but at a slightly reduced rate. However, a further decrease in the rate of increase of throughput occurs, and the rate of increase in latency rises again. The rate of increase in latency then decreases sharply, and even becomes negative, before the links at the second level saturate, at a normalised load of 9, and the latency starts to rise again.

As in the case of uniform traffic, the spare capacity in the lower levels of the network allows for continued increase in throughput after the saturation of upper level links, but here the majority of traffic is routed locally, so the saturation of the top level has relatively less impact. Also, the average latency does not increase as rapidly as in the case of the uniform pattern, since a greater proportion of messages are not being delayed. The effects of lower level links saturating can be observed as load continues to increase but, unlike the uniform traffic case, the latency curve is not monotonic, and not all changes in rate of change correlate with saturation of links. However, the system remains stable under all conditions.

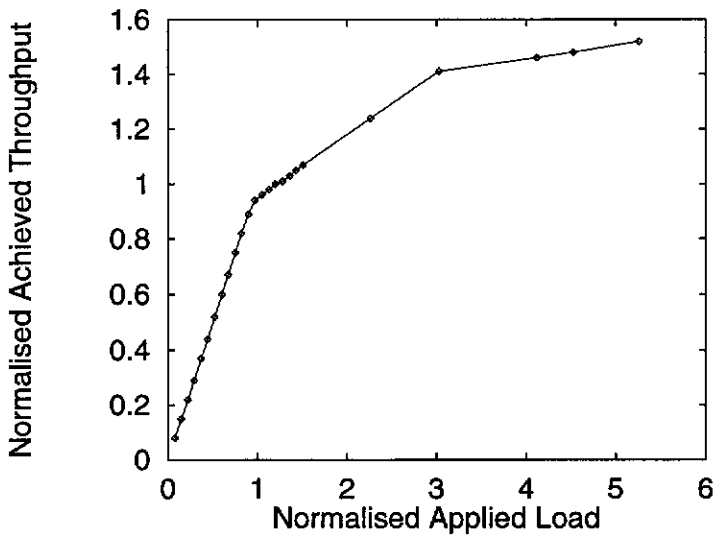


Figure 4.1: Normalised throughput vs normalised applied load (uniform traffic)

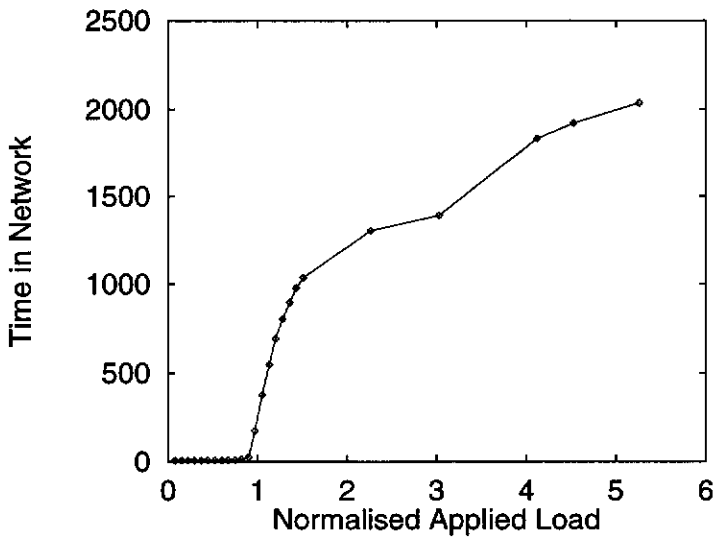


Figure 4.2: Network latency vs normalised applied load (uniform traffic)

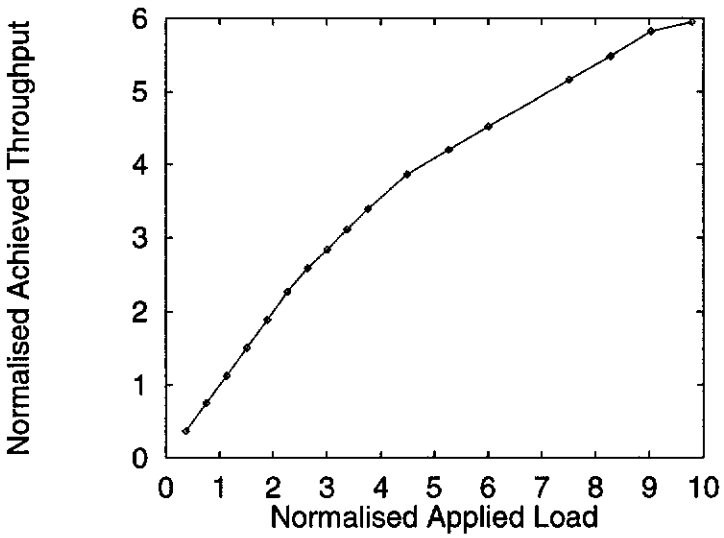


Figure 4.3: Normalised throughput vs normalised applied load (local traffic)

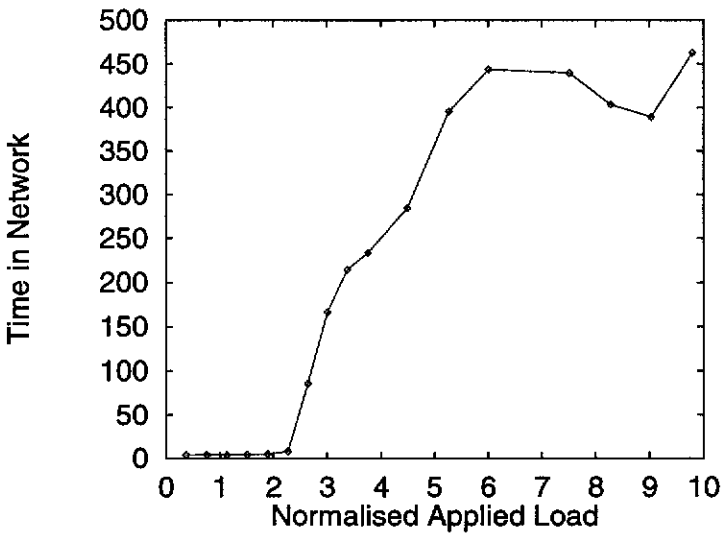


Figure 4.4: Network latency vs normalised applied load (local traffic)

4.3 Performance Analysis: Floyd's Algorithm

The measures of system performance relate to the performance of a particular algorithm on a specified architecture. Floyd's all-pairs shortest path algorithm is a widely useful algorithm with performance characteristics which have been analysed for a variety of parallel topologies and architectures [Kumar and Singh(1991)]. Floyd's algorithm uses a proximity matrix to calculate the shortest path between all pairs of nodes in a connected graph. For a graph G with n nodes a $n \times n$ matrix is required. The sequential algorithm uses n stages, with each stage requiring n^2 compare and replace steps. The sequential execution time T_s is therefore $\Theta(n^3)$. Expressions for speedup, efficiency and iso-efficiency of the Floyd's shortest path algorithm on the HiC will be derived and the results compared with those for the $2D$-mesh. A block mapped, synchronised parallel algorithm is used in each case, but the time for synchronisation is ignored for simplicity. Store and forward message passing is assumed, with startup time represented by t_s and the time to transmit a packet over the link represented by t_w.

4.3.1 HiC

If a $HiC_{(4,h)}$ with p processors is assumed, then $p = k^h = 2^{2h}$ and $2h = \log p$. The time spent in useful work is $T_{calc} = \Theta(\frac{n^3}{p})$. An efficient mapping of a two dimensional matrix onto the HiC is achieved by simply viewing the PEs as a mesh and mapping the data points by column and row. Each PE will contain $\frac{n^2}{p}$ data elements, in $\frac{n}{\sqrt{p}}$ rows and columns. Since elements from only one row or column are communicated at each step, the cost of a single data transmission will be $(t_s + \frac{nt_w}{\sqrt{p}})$. Broadcasting along a row or column can be done in $2(h-1)+2(h-2)+\ldots+2(1)+1$ steps. This can be viewed as $1 + 2\sum_i^{h-1} i = 1 + (h-1)(h-1+1)$ which simplifies to $(h^2 - h + 1)$. However, as the value of h becomes greater than two, congestion starts to occur in the upper layers. The congestion adds a further $2^{h-2}-1$ steps to the communication process. The

number of communication steps is therefore $(2^{h-2}+h^2-h) = \Theta(2^{h-2})$ for $h \geq 3$. Total communication time T_{comm} for all n stages is therefore $n \times (\frac{n}{\sqrt{p}}) \times \Theta(2^{h-2}) = \Theta(n^2)$. The total execution time for the algorithm on p processors is $T_p = T_{calc} + T_{comm}$. The parallel execution time T_p, speedup S, efficiency E and overhead T_o are shown below.

$$\begin{aligned} T_p &= T_{calc} + T_{comm} \\ &= \Theta(\frac{n^3}{p}) + \Theta(n^2) \end{aligned}$$

$$\begin{aligned} S &= \frac{T_s}{T_p} \\ &= \frac{\Theta(p)}{1+\Theta(\frac{p}{n})} \end{aligned}$$

$$\begin{aligned} E &= \frac{T_s}{pT_p} \\ &= \frac{1}{1+\Theta(\frac{p}{n})} \end{aligned}$$

$$\begin{aligned} T_o &= pT_p - T_s \\ &= \Theta(n^3) + \Theta(pn^2) - \Theta(n^3) \\ &= \Theta(pn^2) \end{aligned}$$

From Equation 4.3 $W = KT_o$. Since workload $W = n^3$ we have isoefficiency when $\Theta(n^3) = \Theta(pn^2)$, so the overall isoefficiency function is $\Theta(p^3)$.

4.3.2 $2D$-mesh

For a simple square mesh with no wrap-around and p processors, the time spent in useful work is $T_{calc} = \Theta(\frac{n^3}{p})$. Each PE will contain $\frac{n^2}{p}$ data elements, in $\frac{n}{\sqrt{p}}$ rows and columns. Since elements from only one row or column are communicated at each step, the cost of a single data transmission will be $(t_s + \frac{nt_w}{\sqrt{p}})$. Broadcasting along a row or

Table 4.1:

Floyd's Algorithm on $HiC_{(4,h)}$ and $2D$-mesh.			
Network	Speedup	Efficiency	Isoefficiency
$HiC_{(4,h)}$	$\frac{\Theta(p)}{(1+\Theta(\frac{p}{n}))}$	$\frac{1}{(1+\Theta(\frac{p}{n}))}$	$\Theta(p^3)$
$2D$-mesh	$\frac{\Theta(p)}{(1+\Theta(\frac{p}{n}))}$	$\frac{1}{(1+\Theta(\frac{p}{n}))}$	$\Theta(p^3)$

column can be done in $\sqrt{p}$ steps. Total communication time T_{comm} for all n stages is therefore $\Theta(n^2)$. The parallel execution time, speedup, efficiency and overhead are shown below.

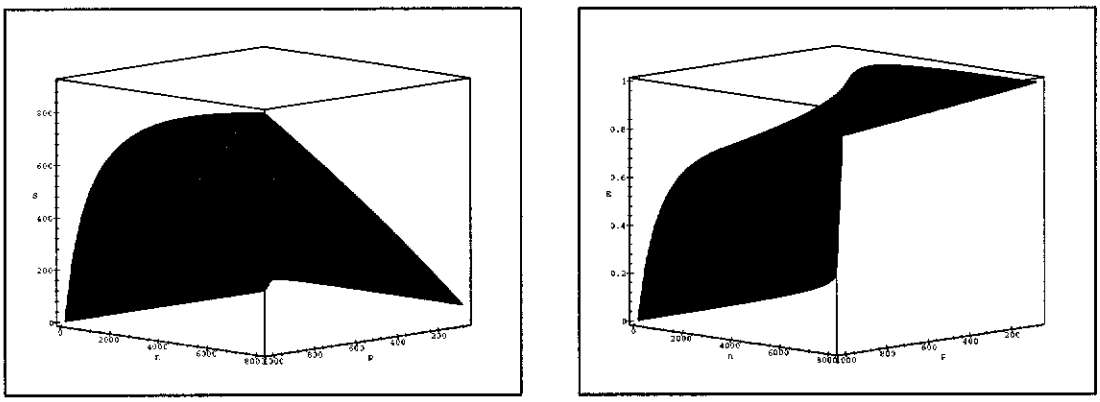
$$\begin{aligned} T_p &= \Theta(\frac{n^3}{p}) + \Theta(n^2) \\ S &= \frac{\Theta(p)}{1+\Theta(\frac{p}{n})} \\ E &= \frac{1}{1+\Theta(\frac{p}{n})} \\ T_o &= \Theta(pn^2) \end{aligned}$$

From Equation 4.3 $W = KT_o$. Since workload $W = n^3$ we have isoefficiency when $\Theta(n^3) = \Theta(pn^2)$, so the overall isoefficiency function is $\Theta(p^3)$.

4.3.3 Interpretation

Table 4.1 shows that the results obtained for the $HiC_{(4,h)}$ are identical with those for the $2D$-mesh. The results indicate that implementations of Floyd's algorithm on the two networks should be able to achieve similar levels of performance, and that efficiency should scale with load in a similar way.

Figure 4.5 shows plots of the speedup and efficiency, for both $HiC_{(4,h)}$ and $2D$-mesh, against n in the range 4 to 8192 and p in the range 64 to 1024. Speedup is seen to increase approximately linearly with p for large enough n. For fixed p, speedup increases rapidly with n until near maximum speedup (p) is reached. Efficiency remains above 0.8 for even large values of p if n is sufficiently large. Efficiency declines for all values



(a) (b)

Figure 4.5: Speedup (4.5(a)) and Efficiency (4.5(b))of Floyd's Algorithm on $2D$-mesh and $HiC_{(4,h)}$

of p with insufficient n, but the roll-off begins earlier with large values of p. In general then, Floyd's algorithm can be implemented on $HiC_{(4,h)}$ and $2D$-mesh in such a way that the computation costs will dominate the communication costs, with a reasonable workload.

A parallel algorithm is scalable if the efficiency can be kept constant, by increasing the problem size, as the number of processors increases. The isoefficiency function describes the rate of growth of the problem size necessary to maintain a fixed efficiency, as a function of the number of processors. The lower the rate of growth, the more scalable the algorithm on that architecture. A linear function would be ideal, but a low order polynomial is an acceptable growth rate. Both $HiC_{(4,h)}$ and $2D$-mesh have an isoefficiency function of $\Theta(p^3)$, which is acceptable.

4.4 Simulation: Floyd's Algorithm

To validate the performance results predicted by the earlier analysis, the problem was run on simulations of the $HiC_{(4,h)}$ and $2D$-mesh topologies for a range of values of p and n. All simulations used code which fully implemented Floyd's algorithm. The simulation determined running time based on *ticks* counted at each processor, the number and size of messages sent and the link speed. A single tick represented a small number of lines of code executed. A processor speed of $100MIPS$ was assumed and a single tick was assumed to use an average of 100 instructions, to give a final speed of $1Mticks/sec$. A link speed of $16Mbytes/sec$ was assumed, therefore a simulated speed of $16bytes/tick$ was used. All final execution times for simulations are given in ticks.

When deriving T_{comm} it is sufficient to say that transmitting $\frac{n}{\sqrt{p}}$ data elements takes $\Theta(\frac{n}{\sqrt{p}})$ units of time.[2] In order to meaningfully compare the derived value with the communication time observed during simulation it is necessary to determine the constants involved. Firstly, since both the pivot row and column are broadcast, the number of data elements actually sent is $\frac{2n}{\sqrt{p}}$. Secondly, the simulator has been set to a link speed of 16 data elements per tick. Therefore the total transmission time will be $\frac{2n}{16\sqrt{p}}$ or $\frac{n}{8\sqrt{p}}$. It is also worth reiterating that, except in the case of the $HiC_{(4,2)}$ with $p = 16$, the value of T_{comm} is determined entirely by the value of n, and does not vary with p.

Table 4.2 shows that the simulated communication time on the $HiC_{(4,h)}$ exactly conformed with the values predicted by analysis. This result validates the model. In contrast, the values for T_{comm} on the $2D$-mesh were consistently overestimated by the model. During analysis communication along a row or column was considered to take $\Theta(\sqrt{p})$ steps. However the actual maximum number of steps is $\sqrt{p} - 1$. The effect of this is negligible for large p, but is quite significant for some of the small values of p used in the simulations. A further inaccuracy arises because the analysis assumed that only

[2]Note that, as the values of p and n are both quite small, the complete expression for T_{comm} was used to determine values, not just the high order parts of the expression.

Table 4.2:

T_{comm} for Floyd's Algorithm on $HiC_{(4,h)}$ and $2D$-mesh.				
Topology		$n = 64$	$n = 256$	$n = 1024$
HiC Analytic	$p = 16$	384	6144	98304
	$p = 64$	512	8192	131072
	$p = 256$	-	8192	131072
HiC Simulation	$p = 16$	384	6144	98304
	$p = 64$	512	8192	131072
	$p = 256$	-	8192	131072
$2D$-mesh Analytic	$p = 16$	512	8192	131072
	$p = 64$	512	8192	131072
	$p = 256$	-	8192	131072
$2D$-mesh Simulation	$p = 16$	320	5120	81920
	$p = 64$	352	5632	90112
	$p = 256$	-	5888	94208

Table 4.3:

T_p for Floyd's Algorithm on $HiC_{(4,h)}$ and $2D$-mesh.				
Topology		$n = 64$	$n = 256$	$n = 1024$
Uniprocessor	$p = 1$	262144	16777216	1.073741e+09
HiC Analytic	$p = 16$	16768	1.054720E+06	6.7207168E+07
	$p = 64$	4608	270336	1.6908288E+07
	$p = 256$	-	73728	4.325376E+06
HiC Simulation	$p = 16$	22752	1.14624E+06	6.86546E+07
	$p = 64$	8048	319424	1.76699E+07
	$p = 256$	-	100320	4.72051E+06
$2D$-mesh Analytic	$p = 16$	16896	1.056768E+06	6.7239936E+07
	$p = 64$	4608	270336	1.6908288E+07
	$p = 256$	-	73728	4.325376E+06
$2D$-mesh Simulation	$p = 16$	21920	1.136E+06	6.8503E+07
	$p = 64$	6864	305088	1.74589E+07
	$p = 256$	-	89312	4.56896E+06

one message would be sent by the originating node. In practice though, any originating PE which is not on an edge of the mesh will send messages in both directions. This reduces communication time by a maximum of a factor of two.

Table 4.3 shows that the total simulated parallel execution time for both $2D$-mesh and $HiC_{(4,h)}$ was always larger than the predicted time. This reflects the extra complexity of the algorithm as implemented on a simulated message passing environment. In particular setting local variables and address calculation contribute extra complexity, but

without altering the fundamental order of the algorithm's complexity. The percentage difference between predicted and observed execution times increased with p but decreased with n, so that any value of n large enough to run efficiently on p runs at close to the expected values of speedup and efficiency. This is due to the extra elements of the implementation being flooded out by the n^3 complexity of the main section of the algorithm. Table 4.3 also shows that the $2D$-mesh runs slightly faster than the HiC. This is attributable to the better than expected communication time of the $2D$-mesh.

4.5 Performance Analysis: *ascend/descend* Algorithms

A widely useful class of algorithms, described in [Preparata and Vuillemin(1981)], is the *ascend/descend* class of algorithms. Assume that $n = 2^i$ input data elements are stored in a vector. An algorithm is in the *descend* class if it performs a series of basic operations that are successively $2^{i-1}, 2^{i-2}, \ldots, 2^0$ locations apart. An algorithm is in the *ascend* class if it performs a series of basic operations that are successively $2^0, 2^1, \ldots, 2^{i-1}$ locations apart. Thus $n/2$ basic operations are performed in each of $\log n$ steps. Many important regular algorithms can be described in terms of this basic class, including FFTs, merging, sorting, convolution etc. Assuming that the basic operation can be performed in constant time, the sequential execution time T_s is $\Theta(n \log n)$.

Expressions for speedup, efficiency and iso-efficiency of the *ascend/descend* class of algorithms on the $HiC_{(4,h)}$, $2D$-mesh and $FatHiC_{(4,h)}$ will be derived and compared. Store and forward message passing is assumed, with startup time represented by t_s and the time to transmit a packet over the link represented by t_w. The time for synchronisation is ignored for simplicity.

4.5.1 HiC

If a $HiC_{(4,h)}$ with p processors is assumed, then $p = k^h = 2^{2h}$ and $2h = \log p$. The time spent in useful work is $T_{calc} = \Theta(\frac{n \log n}{p})$. Each PE contains n/p data elements, so the cost of a single data transmission is $(t_s + t_w(n/p))$ which is $\Theta(n/p)$. For this communication pattern, an efficient mapping of a one dimensional array onto the HiC is achieved by simply viewing the PEs as a mesh, dividing the data into blocks of length n/p and mapping the blocks in row major fashion. Communicating with the required PEs can be done in $[2(h-1)+2(h-1)+2(h-2)+2(h-2)+\ldots+2(1)+2(1)+1+1]$ steps. This can be viewed as $2(1+2\sum_{x=1}^{h-1} x)$, which simplifies to (h^2-h+2). This expression is order $\Theta(h^2)$, which transforms to $\Theta(\log^2 p)$. There are therefore $\Theta(\log^2 p)$ communication steps.

As was the case with Floyd's algorithm, congestion starts to occur in the upper layers when $h \geq 3$. Here, however, the congestion is greater as the number of PEs sending messages is always equal to $p/2$. Delay due to congestion adds $2\sum_{x=1}^{h-2}(k^x - 1)$ steps to the total communication time. This expression simplifies to $2(\frac{(k^{h-1}-k)}{(k-1)} - h + 2)$. Congestion therefore causes delays of order $\Theta(k^{h-1})$, which transforms to $\Theta(p)$.

$$\begin{aligned} T_{comm} &= (n/p)(\Theta(\log^2 p) + \Theta(p)) \\ &= \Theta(n) \end{aligned}$$

The parallel execution time, speedup, efficiency and overhead are shown below.

$$\begin{aligned} T_p &= T_{calc} + T_{comm} \\ &= \Theta(\frac{n \log n}{p}) + \Theta(n) \end{aligned}$$

$$\begin{aligned} S &= \frac{T_s}{T_p} \\ &= \frac{\Theta(p \log n)}{\Theta(\log n) + \Theta(p)} \end{aligned}$$

$$\begin{aligned} E &= \frac{T_s}{pT_p} \\ &= \frac{1}{1 + \Theta(\frac{p}{\log n})} \end{aligned}$$

$$\begin{aligned} T_o &= pT_p - T_s \\ &= p(\Theta(\frac{n \log n}{p}) + \Theta(n)) - \Theta(n \log n) \\ &= \Theta(pn) \end{aligned}$$

From Equation 4.3 $W = KT_o$. Since workload $W = \Theta(n \log n)$ efficiency remains constant when $\Theta(\log n) = \Theta(p)$. Thus the isoefficiency function is $\Theta(2^p p)$.

4.5.2 FatHiC

The *FatHiC* network was introduced earlier as a means of reducing the problem of congestion while maintaining the benefits of the *HiC*. This section studies the requirements and performance of a *FatHiC* system capable of performing the *ascend/descend* class of algorithms without congestion.

Assuming that upper level links have their bandwidth increased sufficiently to prevent congestion, expressions for the performance of the *ascend/descend* class of algorithms

can be calculated as for the HiC, but neglecting the term introduced by congestion.

$$T_{comm} = \Theta(\frac{n \log^2 p}{p}) \tag{4.4}$$

The parallel execution time, speedup, efficiency and overhead are shown below.

$$\begin{aligned} T_p &= T_{calc} + T_{comm} && (4.5)\\ &= \Theta(\frac{n \log n}{p}) + \Theta(\frac{n \log^2 p}{p}) && (4.6) \end{aligned}$$

$$\begin{aligned} S &= \frac{T_s}{T_p} && (4.7)\\ &= \frac{\Theta(p \log n)}{\Theta(\log n) + \Theta(\log^2 p)} && (4.8) \end{aligned}$$

$$\begin{aligned} E &= \frac{T_s}{pT_p} && (4.9)\\ &= \frac{1}{1 + \Theta(\frac{\log^2 p}{\log n})} && (4.10) \end{aligned}$$

$$\begin{aligned} T_o &= pT_p - T_s && (4.11)\\ &= \Theta(n \log^2 p) && (4.12) \end{aligned}$$

From Equation 4.3 $W = KT_o$. Since workload $W = \Theta(n \log n)$ efficiency remains constant when $\Theta(\log n) = \Theta(\log^2 p)$. Thus the isoefficiency function is $\Theta(2^{\log^2 p} \log^2 p)$

4.5.3 $2D$-mesh

Assume a simple square mesh with no wrap-around and p processors. The time spent in useful work is $T_{calc} = \Theta(\frac{n \log n}{p})$. Each PE contains n/p data elements, so the cost of a single data transmission is $(t_s + t_w(n/p))$ which is $\Theta(n/p)$. For this communi-

cation pattern, an efficient mapping of a one dimensional array onto the $2D$-mesh is achieved by simply dividing the data into blocks of length n/p and mapping the blocks in row major fashion. Communication between rows takes place in $\log(\sqrt{p})$ steps and communication along columns takes a further $\log(\sqrt{p})$ steps, giving a total of $\log p$ steps.

$$\begin{aligned} T_{comm} &= \frac{n}{p} \times 2 \sum_{x=0}^{\log p^{1/2}} 2^x && (4.13)\\ &= \frac{n}{p} \times \Theta(\sqrt{p}) && (4.14)\\ &= \Theta(\frac{n}{\sqrt{p}}) && (4.15) \end{aligned}$$

$$\begin{aligned} T_p &= T_{calc} + T_{comm} && (4.16)\\ &= \Theta(\frac{n \log n}{p}) + \Theta(\frac{n}{\sqrt{p}}) && (4.17) \end{aligned}$$

$$\begin{aligned} S &= \frac{T_s}{T_p} && (4.18)\\ &= \frac{\Theta(p \log n)}{\Theta(\log n) + \Theta(\sqrt{p})} && (4.19) \end{aligned}$$

$$\begin{aligned} E &= \frac{T_s}{pT_p} && (4.20)\\ &= \frac{1}{1 + \Theta(\frac{\sqrt{p}}{\log n})} && (4.21) \end{aligned}$$

$$\begin{aligned} T_o &= pT_p - T_s && (4.22)\\ &= \Theta(n\sqrt{p}) && (4.23) \end{aligned}$$

Table 4.4:

ascend/descend Algorithm on HiC, $FatHiC$ and $2D$-mesh.			
Network	Speedup	Efficiency	Isoefficiency
$HiC_{(4,h)}$	$\frac{\Theta(p \log n)}{(\Theta(\log n)+\Theta(p))}$	$\frac{1}{(1+\Theta(\frac{p}{\log n}))}$	$\Theta(2^p p)$
$FatHiC_{(4,h)}$	$\frac{\Theta(p \log n)}{(\Theta(\log n)+\Theta(\log^2 p))}$	$\frac{1}{(1+\Theta(\frac{\log^2 p}{\log n}))}$	$\Theta(2^{\log^2 p} \log^2 p)$
$2D$-mesh	$\frac{\Theta(p \log n)}{\Theta(\log n)+\Theta(\sqrt{p})}$	$\frac{1}{(1+\Theta(\frac{\sqrt{p}}{\log n}))}$	$\Theta(2^{\sqrt{p}} \sqrt{p})$

From Equation 4.3 $W = KT_o$. Since workload $W = \Theta(n \log n)$ Efficiency remains constant when

$$\Theta(\log n) = \Theta(\sqrt{p}).$$

Thus the isoefficiency function is $\Theta(2^{\sqrt{p}}\sqrt{p})$.

4.5.4 Interpretation

Table 4.4 shows the speedup, efficiency and isoefficiency of *ascend/descend* class algorithms on the $HiC_{(4,h)}$, $FatHiC_{(4,h)}$ and $2D$-mesh. The HiC performs poorly, both speedup and efficiency are low unless $n >> p$, as they are dependent upon a p term in the denominator. The isoefficiency shows that the growth rate in workload required to maintain constant efficiency with increasing p is an exponential of p. The *ascend/descend* class of algorithms are not practically scalable on the HiC.

The $2D$-mesh performs better than the HiC, speedup and efficiency depend on a $\sqrt{p}$ term in the denominator, rather than a p term as in the HiC, allowing reasonable values to be obtained with larger values of p. The isoefficiency is also better, as it is an exponential of $\sqrt{p}$. This means the *ascend/descend* class of algorithms are more scalable on $2D$-mesh than HiC. This is because the basic network communication time of the HiC, dictated by its diameter, is overpowered by the effect of congestion in the higher layers. Thus the benefit of the HiC's lower diameter is outweighed by the effect of the meshs' large cross-section bandwidth. The larger the networks, the more

pronounced the effect.

Finally, the $FatHiC$ offers the best performance of the three networks. The speedup and efficiency are dependent upon a $\log^2 p$ term in the denominator, allowing higher values of speedup and efficiency with large p than either the HiC or $2D$-mesh. The isoefficiency is also improved, it is an exponential function of $\log^2 p$, so the $FatHiC$ also scales more effectively than either the HiC or $2D$-mesh. The effect of the lower diameter of the underlying tree topology has become apparent, now that the effects of congestion have been removed, allowing superior performance to be obtained.

Although $\log^2 p$ is asymptotically lower than $\sqrt{p}$, for practical values of p, ($p < 64$k), $\sqrt{p}$ is lower than $\log^2 p$. The relative performance of the $2D$-mesh and $FatHiC$ networks is determined in practice by the constants and lower order terms discarded in the preceding complexity analysis. The results of including these factors are shown in the next section.

4.6 Simulation: *ascend/descend* Algorithms

To validate the performance results predicted by the earlier analysis, the algorithm was run on simulations of the HiC, $FatHiC$ and $2D$-mesh networks for a range of values of p and n. Since the communication time T_{comm} is the area of interest, only the communication pattern was simulated. This allows the relative communication speeds of the different topologies to be compared without concern for the relative speeds of computation and communication. All three networks have identical values of T_s and T_{calc}, so T_{comm} can be compared directly as an indication of algorithm performance.

All links in the HiC and $2D$-mesh have a bandwidth of 1. In order to simulate the $FatHiC$ it is necessary to determine the bandwidth of the upper level links required to prevent congestion. Consider a packet which must be passed from level 0, up to level

1 and back down to level 0. If the throughput of the links at this level is designated τ_1, then trivially $\tau_1 + \tau_1 = 2 \times 1 \times \tau_1$, which says that the time for the full trip equals the time taken going up plus the time taken going down. If packets must be passed from level 0 up to level 2 and back down to level 0 the situation is more complex. All k packets from the level 0 nodes will arrive at the parent node in time $\tau_1 \times 1$. Some of the packets going up the second link will therefore be delayed, the final packet will take time $k\tau_2$ to get through the queue and traverse the link. However, there is no delay on the way back down, as the packet stream has been serialised, so the time for the up and down parts of the trip are no longer symmetric. With no congestion, the final packet to be delivered will take time $\tau_1 + k\tau_2 + \tau_2 + \tau_1 = 2 \times 2 \times \tau_1$. The situation becomes still more complex if packets are to be sent from level 0, up to level 3 and back down again. Now k^2 packets will be routed through the link up to level 3, so once again some packets will be delayed. The final packet through the link will be delayed by all those packets which had not already passed over the link before the final packet arrived. Thus the final packet will take time $\tau_1 + k\tau_2 + k^2\tau_3 - (k-1)\tau_2$ to reach level 3. Once again, the packet stream has been serialised and there are no delays on the trip down. Assume that links at the lowest level have bandwidth of $B_1 = 1$ packet per second. Then throughput $\tau_1 = 1$ seconds per packet. In this case

$$\tau_x(k^{x-1} + 1) = 2x - 2\sum_{i=1}^{x-1} \tau_i.$$

This is a recurrence relation, which can be expressed as

$$\tau_x(k^{x-1} + 1) = 2 + \tau_{x-1}(k^{x-2} - 1).$$

The throughput of a link between a node at level $x-1$ and a node at level x is therefore

$$\tau_x = \frac{2 + \tau_{x-1}(k^{x-2} - 1)}{k^{x-1} + 1}$$

and the link bandwidth is $B_x = \frac{1}{\tau_x}$. Note that the links to the root node are never

Table 4.5:
T_{comm} for *ascend/descend* Algorithms on $HiC_{(4,h)}$,$FatHiC_{(4,h)}$ and $2D$-mesh.

Topology		$n = 64$	$n = 256$	$n = 1024$
HiC Analytic	$p = 16$	24	96	384
	$p = 64$	20	80	320
	$p = 256$	-	62	248
HiC Simulation	$p = 16$	24	96	384
	$p = 64$	20	80	320
	$p = 256$	-	62	248
$FatHiC$ Analytic	$p = 16$	24	96	384
	$p = 64$	14	56	224
	$p = 256$	-	26	104
$FatHiC$ Simulation	$p = 16$	24	96	384
	$p = 64$	14	56	224
	$p = 256$	-	26	104
$2D$-mesh Analytic	$p = 16$	24	96	384
	$p = 64$	14	56	224
	$p = 256$	-	30	120
$2D$-mesh Simulation	$p = 16$	24	96	384
	$p = 64$	14	56	224
	$p = 256$	-	30	120

used in normal routing, and may optionally be discarded. So a $FatHiC_{(4,4)}$ capable of performing *ascend/descend* algorithms with no congestion has $B_1 = 1$, $B_2 = 5/2$ and $B_3 = 85/16$. The horizontal links have bandwidth of 1.

In order to compare the results of simulation and analysis, all low order terms and constants were included in the analytical expressions. For practical values of p they are often very significant.

The simulation results conform exactly to the expected values. For $p = 16$ all three networks perform identically. As p increases the relative performance of the HiC network declines. The $FatHiC$ and $2D$-mesh continue to perform identically when $p = 64$, but when p is increased to 256 the $FatHiC$ is seen to give the best performance.

4.7 Conclusion

This chapter considered the performance of the *HiC* interconnection network. Simulations were used to demonstrate that the proposed routing algorithms are stable up to and beyond saturation. Latency is very low while the network is unsaturated and throughput will continue to rise with increased uniform load even after the top layers saturate. Traffic patterns which exhibit locality of reference yield greatly improved throughput and latency. The performance of Floyd's all pairs shortest path algorithm on the *HiC* was studied and results were compared with results for the popular $2D$-mesh. It was shown that the *HiC* gives very similar performance to the mesh for this type of regular problem. Another regular problem, the *ascend/descend* class of algorithms, showed the susceptibility of the *HiC*s tree based structure to congestion near the root. The *FatHiC*, which eliminates the problem of congestion, offered superior performance to both the *HiC* and $2D$-mesh networks for the *ascend/descend* class of algorithms. Its increased performance, however, must be balanced against the extra cost and complexity incurred in realising the extra bandwidth. This will be further discussed in Chapter 5.

In general then, the *HiC* and *FatHiC* are stable platforms which can be used to solve some important regular problems as efficiently as currently accepted networks. In Chapter 5 it will be shown that the *HiC* and *FatHiC* are also cost competitive.

Chapter 5

Cost Analysis

5.1 Introduction

As stated in Chapter 4 the primary reason for using a parallel computing system is to increase performance. However, the increase in performance is accompanied by an increase in cost. The goal of a system designer, or purchaser, is to obtain maximum performance for minimum cost. In serial systems cost and performance data can be compared more or less directly. Parallel systems, however, include more levels of complexity. Firstly, performance of a parallel architecture can only be measured meaningfully with respect to a particular algorithm or class of algorithms. Secondly, most parallel architectures allow systems of different sizes to be constructed and their performance with respect to a particular algorithm may alter as a function of their size. Thirdly, the rate of increase in cost as system size is increased may not be linear and may vary across architectures. The first two points were discussed in Chapter 4 and are addressed by the use of metrics such as speedup, efficiency and isoefficiency. Efficiency can also be used to relate system performance to system cost, since it determines the effective use made of the available processors. But efficiency is a very simplistic and inadequate measure, inadequate for representing many of the costs of real multiprocessor

systems. Such systems are more than just a collection of processors. They also include an interconnection network, the cost of which may be a significant portion of the total system cost. In order to relate the cost of a parallel system to its performance, the cost and performance of the interconnection network must also be considered.

Relating the cost of a multiprocessor connection network to its performance is a difficult task which has been undertaken by a number of researchers. An analysis of the costs of VLSI parallel systems was performed in [Dally(1990b)]. It was found that the predominant cost was that of connections between devices and that the delay of the interconnections was the main limit on performance. Using bisection width as a measure of cost k-ary n-cube interconnection networks were investigated and low dimensional networks were shown to outperform high dimensional networks with the same interconnection cost. This result was ascribed to the large number of connections which are unused in a high dimensional network during any particular communication phase. A lower dimension network with fewer links makes better use of the bandwidth available. Being a modified tree topology, the *HiC* has a very low bisection width which is unsuitable for use as a measure of interconnection system cost. However the *HiC* has fixed, reasonably low node degree, which is necessary to keep channel bandwidth up and interconnection cost down. The work of [Andrews and Polychronopoulos(1991)] was motivated by interest in the relative cost and performance of homogeneous and heterogeneous shared memory multiprocessors. Their models therefore included information on processor performance, as well as the cost of memory modules, switches and different types of processors. These models are not appropriate for a homogeneous, distributed memory multiprocessor. It was pointed out in [Sarkar(1993)] that while efficiency of an algorithm relates the performance of the total parallel system to the number, and hence cost, of the processors, it fails to take account of the cost of the interconnection network. A new measure of performance was proposed, called *Cost Effectiveness*, which incorporates the total cost of a parallel system and the algorithmic efficiency into a single expression.

In Section 5.2 the *Cost Effectiveness Factor* as introduced in [Sarkar(1993)] is described. A problem with the original definition is described, and a new formulation developed. The cost effectiveness factor for each of HiC, $FatHiC$, d dimensional mesh and $GFT_{(h,m,w)}$ is derived, and the results compared. Section 5.3 then determines the cost effectiveness of Floyd's algorithm and *ascend/descend* class algorithms on each interconnection network and compares the results. Section 5.4 concludes the chapter.

5.2 Cost Effectiveness Factor

The cost of a multiprocessor system is defined as

$$C(p) = c_p p + c_l l \tag{5.1}$$

where c_l is the cost of a single communication link, c_p is the cost of a single PE, p is the number of PEs and l is the number of links. Cost effectiveness $CE(p)$ of a p PE system is defined as $CE(p) = \frac{c_p}{C(p)} \times S(p)$ which simplifies to

$$CE(p) = \frac{E(p)}{1 + c_l/c_p \times l/p} \tag{5.2}$$

where $S(p)$ is the speedup and $E(p)$ is the efficiency of the system. For a particular architecture then, $CE(p)$ is the product of two factors, one representing the efficiency of a particular algorithm on that architecture, the other representing the physical cost of the system. Equation 5.3 defines the factor representing the physical cost of the system, called the cost effectiveness factor by [Sarkar(1993)].

$$CEF(p) = CE(p)/E(p) \tag{5.3}$$

In [Sarkar(1993)] it is assumed that the entire cost of all communication hardware is proportional to the number of links. This is not necessarily a sound assumption, for two

reasons. Firstly, consider an interconnection network where all nodes are processors, such as a mesh. Increasing the dimension of the network while maintaining the number of PEs constant will increase the number of links, and hence the cost of the links, linearly. However the complexity and cost of the switching hardware at each node increases in the order of the square of the dimension of the network. Secondly, consider an interconnection network such as a tree or MIN, where some nodes are not processors but only switches. The number and complexity of switches may not be a simple function of the number of links. In both of these cases the total cost of the interconnection network is not captured by use of a cost function based on the number of links.

This problem is most readily overcome by expressing the cost of the switches explicitly [Campbell and Kumar(1996)]. A new factor $(c_s \times Q \times s)$ is introduced to Equation 5.1, where c_s is the base cost of a switch, Q is the complexity of the switches and s is the number of switches. Now the cost of a multiprocessor system is defined as $C(p) = c_pQp + c_ss + c_ll$ and

$$CEF(p) = \frac{1}{1 + (c_s/c_p \times Q \times s/p) + (c_l/c_p \times l/p)}$$

Let $\alpha = (c_s/c_p)$ and $\beta = (c_l/c_p)$, now

$$CEF(p) = \frac{1}{1 + \alpha Qs/p + \beta l/p} \tag{5.4}$$

It remains to make a reasonable estimate of values for α and β. Clearly these values are very dependent on architecture and current technology and pricing. Using current "commercial off the shelf" prices for a system based on a cluster of workstations architecture; values of 3/500 for α and 3/50 for β were decided upon. These values show a single switching element to be less costly than a single link, with a single processing node far more expensive than either. The actual switch cost will be modified by the switch complexity, which may be significant.

5.2.1 *HiC*

From Section 3.3 the $HiC_{(k,h)}$ has a number of PEs $p = N_P = k^h$ and a number of links $l = L = \frac{k(k+1)(k^h-1)}{2(k-1)}$. Although the PEs have so far been considered solely as processors, in practice some method must exist for each PE to transmit messages to, and receive messages from, its $k-1$ neighbouring PEs and its parent SE; in effect a switch. There are thus two types of switches. Type 1 occurs at non-leaf nodes, which are of degree $2k$. Thus $2k \times 2k$ switches are required and the complexity of Type 1 switches is $Q_1 = 4k^2$. Type 2 occurs at leaf-nodes, which are of degree k. All messages are being routed to or from the PE, Therefore $1 \times k$ switches are required. The complexity of Type 2 switches is $Q_2 = k$. From Section 3.3, the number of Type 1 switches $s_1 = N_S = \frac{(k^h-1)}{(k-1)}$ and the number of Type 2 switches $s_2 = N_P = k^h$. The term $\alpha Qs/p$ in Equation 5.4 becomes

$$\alpha(Q_1 s_1/p + Q_2 s_2/p) = \alpha k(\frac{4(k^h-1)}{k^{h-1}(k-1)} + 1) \tag{5.5}$$

The term $\beta l/p$ in Equation 5.4 becomes

$$\beta l/p = \frac{\beta(k+1)(k^h-1)}{2k^{h-1}(k-1)} \tag{5.6}$$

Gathering terms, the resulting expression for cost effectiveness factor is

$$CEF(p) = \frac{1}{1 + \alpha k + \frac{(k^h-1)}{k^{h-1}(k-1)}(4\alpha k + \frac{\beta(k+1)}{2})} \tag{5.7}$$

Figure 5.1 shows plots of CEF as a function of the number of PEs in the system, with various values of k. The plot clearly shows that a value of $k = 3$ yields the best cost effectiveness factor in any size system. While not planar, there are fewer crossing links in this configuration than for higher values of k. This is possibly the optimum configuration for some irregular problems, where mapping data into regular two or

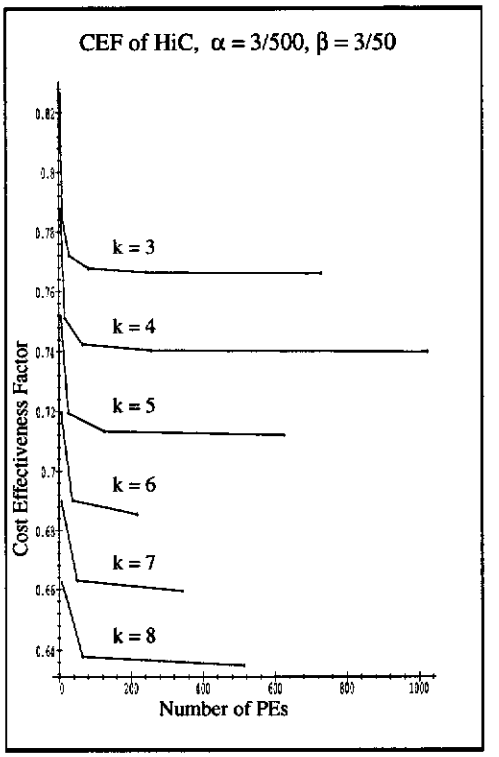


Figure 5.1: CEF of the HiC

three dimensional problems is not important. However a value of $k = 4$ also results in acceptable CEF, while providing a PE interconnection pattern which is easy to map many problems onto. Throughout this thesis a value of $k = 4$ is used. In general, the plot shows that increasing k decreases the cost effectiveness factor in a fairly uniform manner.

5.2.2 $FatHiC$

Section 4.5.2 analysed the performance of an *ascend/descend* algorithm executed on a $FatHiC_{(4,h)}$. The bandwidth of the "Fat" links required to prevent congestion while executing an *ascend/descend* algorithm was determined in Section 4.6. This section will determine the cost effectiveness factor of such a $FatHiC$.

The required throughput of a link between a node at level $x-1$ and a node at level x is

$$\tau_x = \frac{2 + \tau_{x-1}(k^{x-2} - 1)}{k^{x-1} + 1}$$

and the link bandwidth is $B_x = \frac{1}{\tau_x}$. Note that the links to the root node are never used in normal routing, and they will be discarded here as they are of high bandwidth and their cost would be significant. Assuming that link cost increases linearly with bandwidth, the cost of links is most easily represented by counting each single link, with bandwidth B, as B links of unit bandwidth. The total number of links in the $FatHiC_{(k,h)}$ network can now be calculated. From Section 3.3 a $HiC_{(k,h)}$ has a number of links $L = L_T + L_C$ where L_T represents the number of links in a k-ary tree of height h, and L_C represents the number of links connecting nodes into k-cliques. The expression for L_C remains the same, as no intra-clique links require increased bandwidth to prevent congestion while executing an *ascend/descend* algorithm.

$$L_C = \frac{(k-1)}{2} \sum_{x=1}^{h} k^x \tag{5.8}$$

$$= k(k^h - 1)/2 \tag{5.9}$$

The new expression for L_T incorporates extra links to model the higher bandwidth links, but does not include links to the root node.

$$L_T = \sum_{x=1}^{h-1} (k^{h+1-x} B_x) \tag{5.10}$$

$$\therefore L = k(k^h - 1)/2 + \sum_{x=1}^{h-1} k^{h+1-x} B_x \tag{5.11}$$

The term $\beta l/p$ in Equation 5.4 becomes $(\frac{\beta(k^h-1)}{2k^{h-1}} + \frac{\beta}{k^h} \sum_{x=1}^{h-1} k^{h+1-x} B_x)$. All other terms

remain the same as for HiC, so the resulting expression for cost effectiveness factor is

$$CEF(p) = \frac{1}{1 + \alpha k + \frac{k^h - 1}{k^h - 1}(\frac{4\alpha k}{k-1} + \frac{\beta}{2}) + \frac{\beta}{k^h}\sum_{x=1}^{h-1} k^{h+1-x} B_x} \tag{5.12}$$

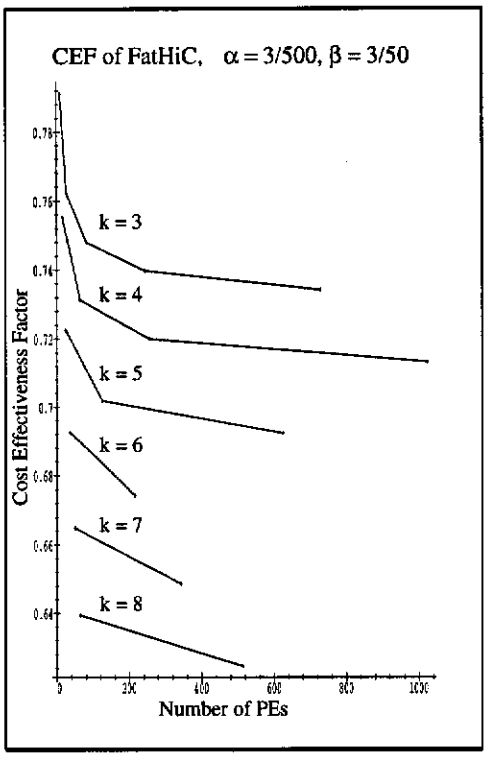


Figure 5.2: CEF of the $FatHiC$

Figure 5.2 shows a number of plots of CEF as a function of the number of PEs in the $FatHiC$ system, with various values of k. Comparing the plots with those in Figure 5.1, the cost effectiveness factor of the $FatHiC$ is lower than that of the HiC for any value of k and decreases more rapidly as the number of processors increases. For reasonable numbers of processors, however, the CEF of the $FatHiC$ is not greatly decreased from that of the equivalent HiC. It is noticeable that the cost effectiveness factor of the $FatHiC_{(3,h)}$ is closer to that of the $FatHiC_{(4,h)}$ for similar numbers of processors than is the case with the HiC. The increasing cost of links at high levels in the hierarchy of a $FatHiC$ mean that there is a greater cost penalty for increasing the height of the

network.

5.2.3 nD-mesh

From Section 2.3.2 an n-dimensional mesh M (no wrap-around links) with radix r (the number of nodes on an edge, in each dimension) has $N = r^n$ nodes and $L = n(r-1)r^{n-1}$ links. Therefore $p = N = r^n$ and $l = L = n(r-1)r^{n-1}$. Since the mesh is a static network, the number of switches $s = N = r^n$, and the complexity of each switch is $(2n+1)^2$.

The cost effectiveness factor is therefore

$$CEF(p) = \frac{1}{1 + \alpha(2n+1)^2 + \beta n(1 - \frac{1}{r})} \tag{5.13}$$

Figure 5.3 shows plots of CEF as a function of the number of PEs in the system, with various values of n. A one dimensional mesh has the best cost effectiveness factor. CEF decreases quite uniformly as the dimension is increased.

5.2.4 Generalised Fat Trees

Generalised fat trees were described in 2.3.3 as $GFT_{(h,m,w)}$, where h is the height of the tree, m is the number of children of each non leaf node, and w is the number of parents of each non-root node. Any $GFT_{(h,m,w)}$ has a number of PEs $p = N_P = m^h$ and a number of links $l = L = \frac{wm(w^h - m^h)}{(w-m)}$ when $w \neq m$. There are three types of switch. Type 1 occurs at root nodes. There are $s_1 = w^h$ root nodes. Each root node has m children and requires a switch of complexity $Q_1 = m^2$. Type 2 occurs at inner nodes. There are $s_2 = \sum_{x=1}^{h-1}(m^{h-x}w^x)$ inner nodes, which transforms to the closed form $\frac{wm(w^{h-1} - m^{h-1})}{(w-m)}$ when $w \neq m$. Each inner node has m children and w parents, requiring a switch of complexity $Q_2 = (m+w)^2$. Type 3 occurs at leaf nodes, which are

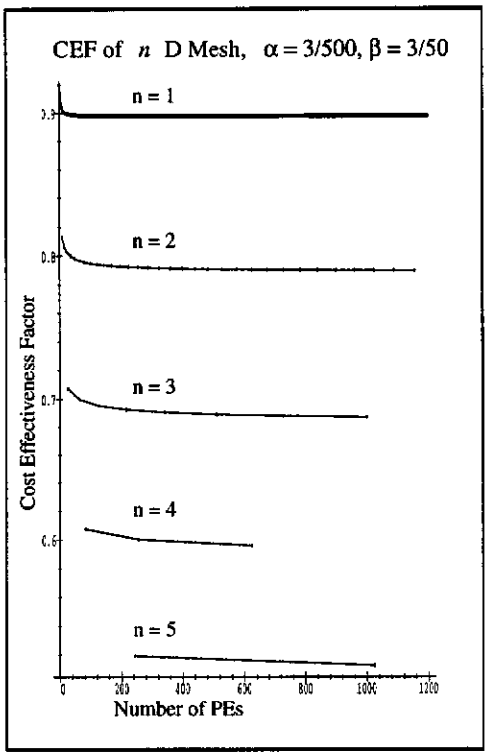


Figure 5.3: CEF of the n dimensional mesh

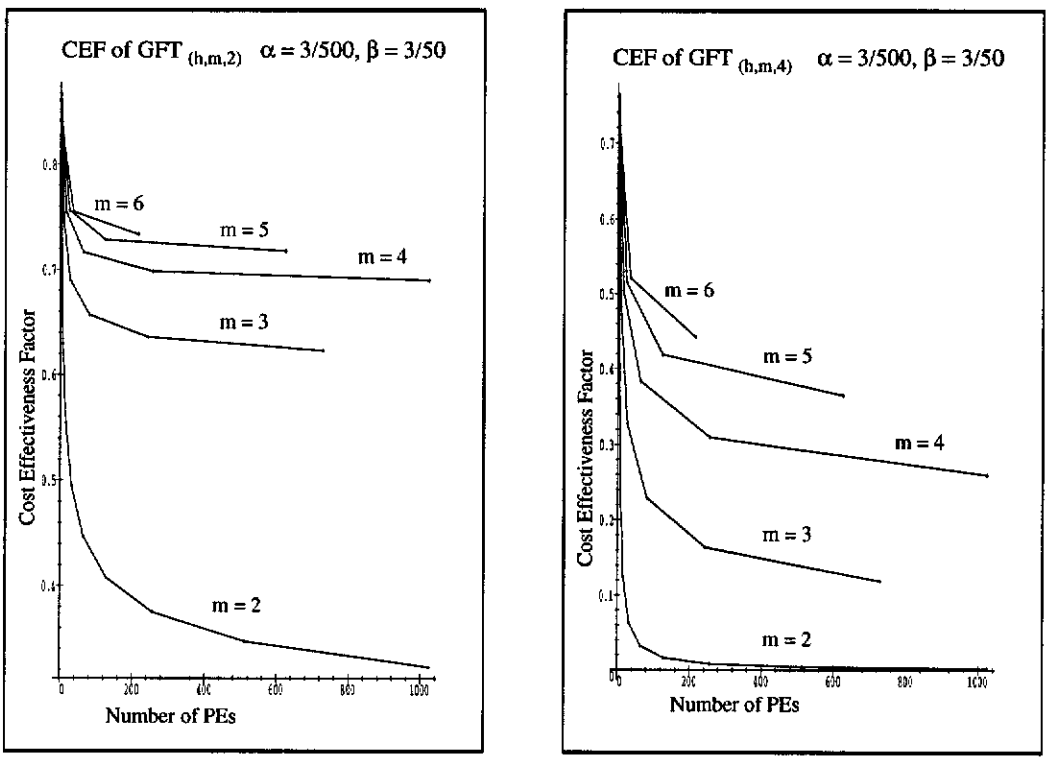
the PEs. There are $s_3 = p = m^h$ leaf nodes. Each leaf has w parents. The switch need only allow one parent to communicate with the PE, so the required switch complexity is $Q_3 = w$. The terms for switch cost and link cost in Equation 5.4 can now be calculated.

$$\alpha(s_1Q_1 + s_2Q_2 + s_3Q_3)/p = \alpha(\frac{w^h m^2}{m^h} + \frac{wm(w^{h-1} - m^{h-1})(m+w)^2}{m^h(w-m)} + w) \tag{5.14}$$

$$\beta l/p = \frac{\beta wm(w^h - m^h)}{m^h(w-m)} \tag{5.15}$$

Gathering terms, the final expression for cost effectiveness factor is determined.

$$CEF(p) = \frac{1}{1 + \alpha(\frac{w^h m^2}{m^h} + w) + \frac{wm}{m^h(w-m)}(\alpha(w^{h-1} - m^{h-1})(m+w)^2 + \beta(w^h - m^h))} \tag{5.16}$$



(a) (b)

Figure 5.4: Cost Effectiveness Factor: (a) $GFT_{(h,m,2)}$ (b) $GFT_{(h,m,4)}$

Figure 5.4 shows two plots of CEF as a function of the number of PEs in the system. CEF is observed to decrease with increased values of w or decreased values of m. For a given number of PEs, a low value of m means that h must be greater, increasing the number of links and switches. However, increasing m too far means that the complexity of the switches starts to dominate the cost and CEF starts to decline. Practically, no further increase in CEF is obtained by increasing m above 6. For any combination of

m and h, increasing w means extra links and more and more complex switches, with a corresponding decrease in CEF. The lowest value of w which maintains any of the advantages of a fat tree is 2. Therefore the GFT with the highest cost effectiveness factor is $GFT_{(h,6,2)}$.

5.2.5 Comparison

By comparing the graphs in Figures 5.1, 5.2, 5.3 and 5.4 it can be seen that the one and two dimensional meshes have the highest CEF. The CEF of both $HiC_{(3,h)}$ and $HiC_{(4,h)}$ is lower than that of the two dimensional mesh but higher than that of the three dimensional mesh. $HiC_{(4,h)}$ has very similar CEF to the $GFT_{(h,6,2)}$, the GFT with the highest cost effectiveness factor, and the $FatHiC_{(3,h)}$. The $FatHiC_{(4,h)}$ has a lower CEF again, though still superior to the three dimensional mesh.

The cost effectiveness factor depends upon the cost of the processors as a fraction of the total system cost. It therefore rewards sparsely connected networks, as the fraction of the total system cost spent on the interconnection network hardware is relatively small. More richly connected networks may, however, offer improved efficiency if algorithmic communication patterns map well to the topology. A HiC with $k = 4$ may perform a matrix operation much more efficiently than a one dimensional mesh, justifying the extra expense of the denser communication network. In general though, for any algorithm which can be performed with the same efficiency on two networks, the network with the higher CEF is more cost effective. If two networks have a similar value of CEF then they will have similar cost effectiveness for algorithms which have similar efficiency on both networks. The $HiC_{(4,h)}$ has a slightly lower CEF than the two dimensional mesh and a higher CEF than three dimensional meshes or most GFTs.

5.3 Cost Effectiveness

Having determined the expressions for CEF of the HiC, $FatHiC$, mesh and GFT, it is necessary to establish the efficiency of a specific algorithm on each network in order to compare the cost effectiveness of each network for that particular algorithm. Efficiency of an algorithm will depend on the specific configuration of each network. This section deals with the $HiC_{(4,h)}$, $FatHiC_{(4,h)}$, $2D$-mesh and $GFT_{(h,4,2)}$ networks.

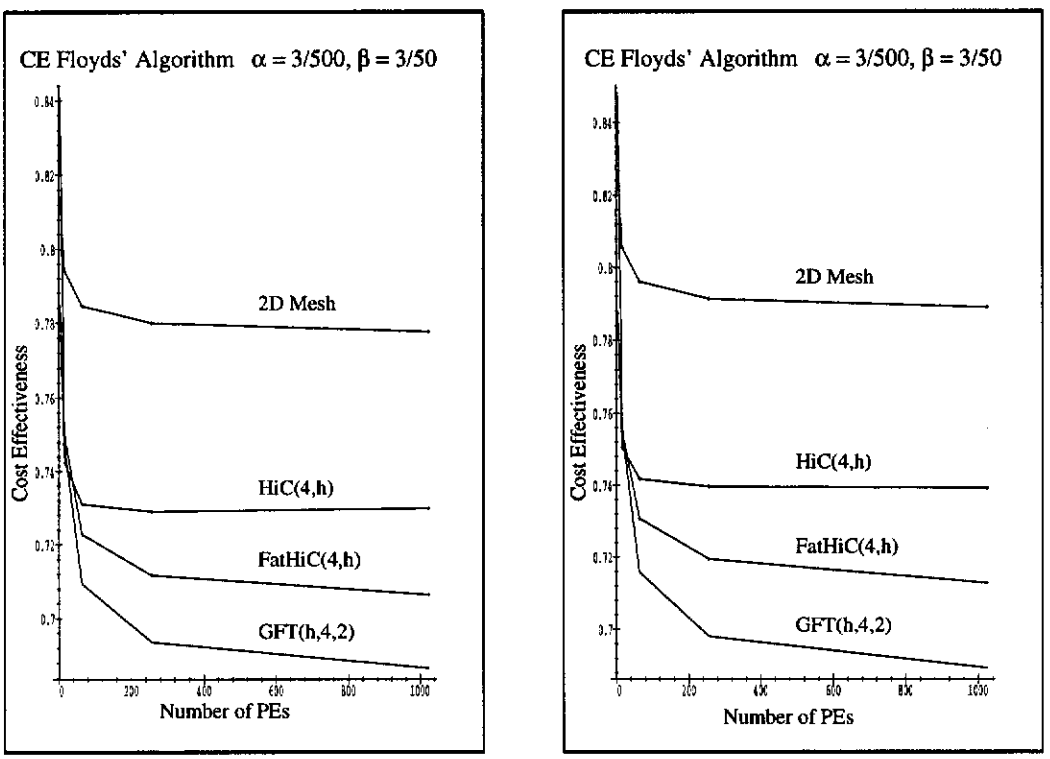
5.3.1 Cost Effectiveness of Floyd's Algorithm

Efficiency of Floyd's Algorithm on the $HiC_{(4,h)}$ and $2D$-mesh was determined in Section 4.3. For values of p below approximately 64000, however, the high order terms do not provide a very good approximation to the real value of efficiency. This section will consider values of p below that limit, therefore lower order terms are included in the following expressions for efficiency.

From Section 4.3, for a problem size n and number of PEs p, $E = \frac{1}{1+\sqrt{p}(2^{h-2}+h^2-h)/n}$ on the $HiC_{(4,h)}$ and $E = \frac{1}{1+(p/n)}$ on the $2D$-mesh. The efficiency of Floyd's algorithm on the $FatHiC_{(4,h)}$ can be derived in the same manner as that of the $HiC_{(4,h)}$, excluding the term 2^{h-2} in T_{comm} which accounts for congestion. $E = \frac{1}{1+\sqrt{p}(h^2-h)/n}$ on the $FatHiC_{(4,h)}$. The $GFT_{(h,4,2)}$ executes Floyd's algorithm with efficiency $E = \frac{1}{1+(\sqrt{p}h/n)}$.

Using these expressions for efficiency, and Equation 5.3, the cost effectiveness of executing Floyd's algorithm on each network can be determined. Figure 5.5 shows the resulting values for each topology under two different load conditions, $n = 64p$ and $n = 1024p$. For each load the $2D$ mesh is the most cost effective topology, followed by the $HiC_{(4,h)}$, $FatHiC_{(4,h)}$ and $GFT_{(h,4,2)}$. The difference in cost effectiveness in each case is small but noticeable. The cost effectiveness of the GFT is not only the lowest, it also declines most rapidly as p increases. The $FatHiC$ also shows a continuing decline

in cost effectiveness for increasing p with this algorithm, while the curves for both HiC and mesh remain essentially level for p greater than 64. Thus increasing the size of the HiC beyond 64 PEs does not impact negatively on its cost effectiveness for running Floyd's algorithm, unlike the GFT. Comparing the results obtained under different work loads, the cost effectiveness is seen to rise for all topologies as the work per PE increases. This reflects the increase in efficiency each topology achieves with increased workload. The increase in cost effectiveness for each topology is approximately the same, reflecting the similar impact of an increase in workload on the efficiency of each topology.



(a) (b)

Figure 5.5: CE of Floyd's algorithm on the $HiC_{(4,h)}$, $2D$-mesh, $FatHiC_{(4,h)}$ and $GFT_{(h,4,2)}$: (a) $n = 64p$ (b) $n = 1024p$

5.3.2 Cost Effectiveness of *ascend/descend* Algorithms

This section will consider values of p below the level at which high order terms provide a good approximation to the total result, therefore lower order terms are included in the following expressions for efficiency. From Section 4.5 *ascend/descend* algorithms with problem size n execute on the $HiC_{(4,h)}$ with efficiency

$$E = \frac{1}{1 + ((h^2 - h + 2) + 2(\frac{(4^{h-1}-4)}{(4-1)} - h + 2))/\log n}.$$

The $2D$-mesh achieves efficiency

$$E = \frac{1}{1 + 2(2\sqrt{p} - 1)/\log n)}$$

while the $FatHiC_{(4,h)}$ has efficiency equal to that of the $HiC_{(4,h)}$ without the influence of congestion

$$E = \frac{1}{1 + (h^2 - h + 2)/\log n}.$$

The *ascend/descend* class of algorithms on the $GFT_{(h,4,2)}$ have a basic communication time of $2h(h+1)$ for $h \neq 1$. However it suffers congestion in the upper layers if $h > 3$, which is to say with $p > 64$. Therefore a $GFT_{(h,4,2)}$ with $p > 64$ has communication time

$$T_{comm} = \frac{n}{p}(2h(h+1) + (2^{h-3} - 1))$$

and efficiency of

$$E = \frac{1}{1 + (2h(h+1) + (2^{h-3} - 1))/\log n}.$$

Figure 5.6 shows the resulting values for each topology with a load of $n = 1024p$. None of the topologies studied here are outstandingly cost effective for *ascend/descend* class algorithms, however the $FatHiC_{(4,h)}$ is clearly the most cost effective for the values of p considered. The cost effectiveness of all the topologies declines as p increases. The cost effectiveness of the $GFT_{(h,4,2)}$ declines least rapidly, however the difference in

slopes of the curves for the $FatHiC$ and GFT is so small that the GFT is unlikely to become more cost effective than the $FatHiC$ for any practical value of p. The cost effectiveness of the $HiC_{(4,h)}$ declines most rapidly. The superior cost effectiveness factor of the $HiC_{(4,h)}$ is quickly overpowered by the reduced efficiency caused by congestion. The $2D$-mesh also performs poorly, the communication time increases as a factor of the square root of the number of processors, so the good cost effectiveness factor is dominated by the poor communication performance. The cost effectiveness of the $FatHiC_{(4,h)}$ decreases slightly more rapidly than that of the $GFT_{(h,4,2)}$, which indicates that the decrease in efficiency which the GFT suffers, due to congestion, as h increases is not as rapid as the increase in cost of the $FatHiC$ due to increasing cost of links. Even so the $FatHiC$ is significantly more cost effective than the GFT for systems involving hundreds of PEs, and far more cost effective than either the HiC or $2D$-mesh.

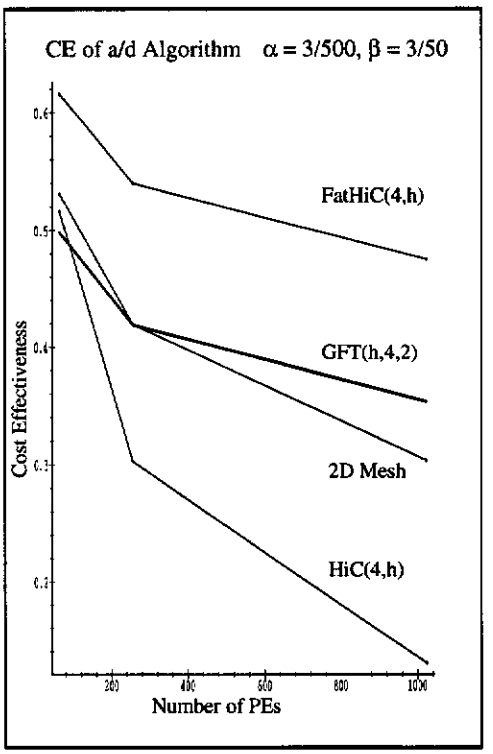


Figure 5.6: CE of *ascend/descend* class algorithms on the $HiC_{(4,h)}$, $2D$-mesh and $GFT_{(h,4,2)}$: $n = 1024p$

5.4 Conclusion

This chapter related the cost of the HiC network and its variant the $FatHiC$ to their performance. A novel modification of cost effectiveness factor, introduced by [Sarkar(1993)], allowed the cost effectiveness of the HiC and $FatHiC$ to be compared with that of the $2D$-mesh and GFT for two important regular problems. For Floyd's all pairs shortest path algorithm the HiC gave a cost effectiveness which was stable with increasing numbers of processors and not greatly inferior to that of the $2D$-mesh. The $FatHiC$ was less cost effective than the HiC for Floyd's algorithm, the increased efficiency offered by improved communication was not sufficient to outweigh the extra cost of the fat links for this application. Similarly the increased connectivity of the GFT was not of sufficient use in implementing Floyd's algorithm to justify its cost. For the *ascend/descend* class of algorithms the cost effectiveness of the HiC was decreased by congestion in the interconnection network. The $FatHiC$'s gain in efficiency by eliminating the delays caused by congestion exceeded the increase in cost caused by the extra bandwidth of high level links. The $FatHiC$ was the most cost effective network considered for the *ascend/descend* class of algorithms.

In general the HiC interconnection network allows cost effective solutions of some important regular problems. The HiC is competitive with the popular $2D$-mesh for such problems. It is also competitive with the GFT. Congestion in the network may reduce efficiency for some classes of problem. In such cases the $FatHiC$ may prove a cost effective alternative. In the next chapter, the HiC and $FatHiC$ networks will be shown to have advantages when executing irregular algorithms.

Chapter 6

Dynamic Load Balancing on the *HiC*

6.1 Introduction

Ideally, every processor in a parallel computer will perform an equal amount of work when executing an algorithm. This situation will result in maximum speedup and high efficiency, provided that the problem is sufficiently large to overcome the effect of overheads. For algorithms where the data or flow of control is irregularly structured, balancing the total workload across all available processors is often a difficult problem, which must be solved at execution time, dynamically. However, time spent deciding how to balance the workload is an overhead, and should be reduced as much as possible. Ensuring that the work load is balanced across all the processors, while minimising the overheads incurred by the distribution process, is the most fundamental problem to be overcome when executing irregular algorithms on parallel computers. This chapter will demonstrate that the *HiC* and *FatHiC* interconnection networks are well suited to executing irregular algorithms which rely on dynamic load balancing strategies to

achieve efficiency.

A new, receiver initiated, dynamic work distribution algorithm for $HiC_{(k,h)}$ is described, the hierarchical search (HS) algorithm. The HS algorithm uses the hierarchical structure of the HiC as the basis for selection of target processors. This allows the HS algorithms target selection method to be simple and deterministic, while using locality to reduce both request and work transfer times. The scalability of the HS algorithm is analysed and compared with existing dynamic load balancing algorithms for hypercube.

As an example of an application of the HS algorithm, a novel application algorithm is introduced, the Parallel Network (PN) algorithm. The PN algorithm is a parallel, deterministic algorithm for finding subgraph isomorphisms from a database of attributed, directed model graphs to an attributed, directed input graph. The algorithm distributes data amongst processors so that work is generated at many processors throughout the computer. The HS algorithm then efficiently distributes work to idle processors.

The chapter is set out as follows: Section 6.2 discusses some desirable properties for a receiver initiated dynamic load balancing strategy to possess. Section 6.3 describes and analyses a new algorithm, the HS algorithm, and compares its properties and scalability with existing algorithms. Section 6.4 proposes an example application algorithm, the PN algorithm. The PN algorithm is described and its worst case complexity is analysed. The forms of parallelism within the algorithm are discussed. The advantages of using a HiC interconnection network and the HS algorithm are considered. Conclusions are drawn in Section 6.5.

6.2 Dynamic Load Balancing Strategies

Many different dynamic load balancing schemes have been proposed and used over the years. Most of them fall into one of two broad categories: sender initiated or receiver

initiated load balancing schemes. This chapter deals exclusively with receiver initiated load balancing schemes. To recap the brief description of dynamic load balancing schemes given in Section 2.4, receiver initiated schemes split work only when an idle processor requests work from a target processor with more than some threshold quantity of work. The target processor then divides its work and passes some to the requesting processor. If a processor receives a request for work when it is idle, or has too little work, then it rejects the request. If an idle processor has a request for work rejected, it selects another target and sends it a work request. A number of receiver initiated dynamic load balancing strategies are briefly described in Section 2.4. They are: nearest neighbour (NN), random polling (RP), asynchronous round robin (ARR), global round robin (GRR) and scheduler based load balancing (SB).

A desirable property of a load balancing strategy is that work requests should be evenly distributed over all processors, thereby ensuring that available work is distributed as quickly and efficiently as possible. Most receiver initiated load balancing schemes can be broadly classified as either asynchronous or synchronous. In asynchronous schemes, such as RP and ARR, each idle processor determines its own targets, independently of all other processors. In most asynchronous schemes a situation can therefore arise where a single processor is targeted by multiple idle processors simultaneously. Synchronous schemes use a central resource which allocates targets to idle processors. Synchronous schemes, such as GRR and SB, can ensure that work requests are allocated uniformly. The drawback of such schemes is that their use of a central resource creates a bottleneck, which can fundamentally limit their efficiency and scalability.

In general, the greater the distance over which work is to be transferred, the greater the transfer time. This is true even in systems which employ cut-through type routing techniques; especially in an irregular algorithm, where the longer the path taken by a packet becomes, the more likely the packet is to be blocked. It is therefore a desirable property of a load balancing strategy that idle processors should seek work in a struc-

tured way, requesting work from neighbouring processors before targeting processors at increasing distance. None of the strategies discussed has this property. The NN scheme does make use of locality, but it searches only the nearest processors. For this reason the NN scheme is very slow to balance the work load if there is a localised concentration of work.

Simplicity is also a virtue for a dynamic load balancing scheme. Strategies which rely on queues of potential targets and client-server style target selection techniques increase complexity. They must therefore offer significant advantages in other areas to compensate for the additional overhead caused by their own complexity.

6.3 A New Algorithm: Hierarchical Search HS

The fundamental difference between the various receiver initiated, dynamic load balancing strategies is the method of selection, by idle processors, of target processors from which to request work. The hierarchical structure of the *HiC* architecture provides a natural sequence for processors to follow when requesting work.

An idle processor will look for work first amongst its neighbours. The rich local connectivity allows work to be rapidly distributed at a local level, meaning the workload within a clique will be rapidly balanced. If an entire clique is idle, the processors within it will request work from processors with which they share a least common ancestor at level 1, then 2, then 3, and so on until either a targeted processor responds to the request for work by supplying some work, or processors which share a least common ancestor at level $h-1$ have rejected a request for work. In the latter case the idle processor will once again request work from its neighbours, as they may have successfully obtained work. A processor which completes its work will always start the search process from the beginning.

As an example, consider the case of a $HiC_{(4,3)}$, the PEs of which are shown in Figure 6.1. In the initial state, only PE $\langle 1,1,1\rangle$ has work. It is denoted with a W. In the first step of the HS algorithm, each idle PE with address $M = \langle P, M_{h-1}\rangle$ requests work from the PE with address $\langle P, (M_{h-1}+1 \bmod 4)\rangle$. The result of the first step is that PE $\langle 1,1,4\rangle$ receives work from PE $\langle 1,1,1\rangle$. In the second step each idle PE with address $M = \langle P, M_{h-1}\rangle$ requests work from the PE with address $\langle P, (M_{h-1}+2 \bmod 4)\rangle$. The result of the second step is that PE $\langle 1,1,2\rangle$ receives work from PE $\langle 1,1,4\rangle$ and PE $\langle 1,1,3\rangle$ receives work from PE $\langle 1,1,1\rangle$. Although each PE has three neighbours, it is sufficient for each PE to request work from only two of them in order to completely distribute work from any one PE within a clique to all the others. All PEs which are still idle then request work from their siblings, following the same order as was followed when requesting work from neighbours. That is to say, in the third step an idle PE with address $M = \langle M_0, P\rangle$ requests work from the PE with address $\langle (M_0+1 \bmod 4), P\rangle$. The result of the third step is that PEs $\langle 4,1,1\rangle$, $\langle 4,1,2\rangle$, $\langle 4,1,3\rangle$ and $\langle 4,1,4\rangle$ all receive work from their siblings $\langle 1,1,1\rangle$, $\langle 1,1,2\rangle$, $\langle 1,1,3\rangle$ and $\langle 1,1,4\rangle$. Continuing in this fashion, it can be seen that all PEs will receive work within six steps. For a $HiC_{(4,h)}$, all PEs will receive work within $2h = \Theta(\log p)$ steps.

The HS algorithm has the desirable property of simplicity. No processor need be designated as a server, nor do lists of potential targets need to be maintained. Each processor determines its own targets in a simple, deterministic manner. The overheads due to target selection are therefore very small.

The HS algorithm searches for work at the minimum possible distance from the idle processor. This minimises the overheads due to communication time. Like the NN strategy, nearest neighbours are targeted. Unlike the NN strategy, if no work is to be found locally the HS algorithm causes an idle processor to target processors which are progressively further afield, rather than waiting for work to ripple through the network. This means work can be rapidly distributed throughout the network, even from a single

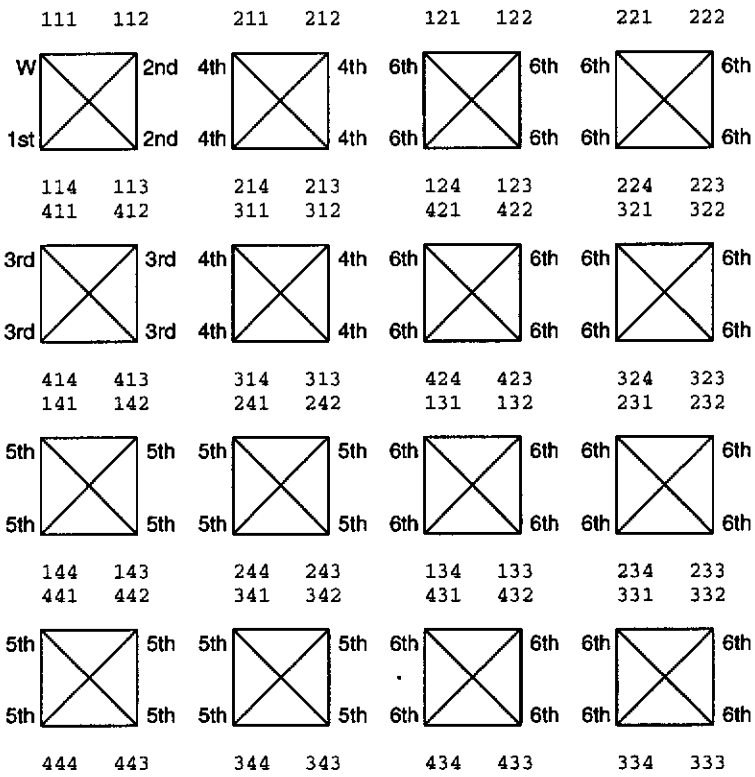


Figure 6.1: HS dynamic load balancing.

source processor.

The HS algorithm is asynchronous. This means it is possible for a single processor to be targeted multiple times at around the same time. This is not ideal, a uniform distribution of requests amongst processors is considered preferable. Synchronous strategies, such as the GRR and SB strategies, ensure a uniform distribution of requests by using a single processor to allocate all requests to target processors. The drawback of such schemes is that it creates a central resource, which can become a bottleneck. Scalability analysis performed in [Kumar and Rao(1987)] showed that the bottleneck caused by a central queue of requests dominated the overheads due to communication for the GRR and SB strategies. In the HS algorithm no central bottleneck is created, and it will be shown to be more scalable than the GRR and SB strategies.

6.3.1 Analysis

The dynamic nature of the proposed load balancing algorithm makes analysis of its probable performance difficult. Techniques used in [Kumar and Rao(1987)] provide a means of determining some useful bounds on the isoefficiency function of the algorithm. It is assumed that, at the beginning of any program execution, the total workload W is available on one processor. Since the total amount of useful work may be unknown for an irregular algorithm, W represents the maximum possible workload. When a processor receives a work request, it divides its current workload w into two parts, γw and $(1-\gamma)w$, where $\gamma \leq 0.5$. A processor will reject a request for work if it has workload $w \leq \varepsilon$, where ε is a threshold value which is too small to divide.

The unit computation time, U_{calc}, is defined as the time taken for one unit of work, and the unit communication time U_{comm} as the mean time taken to get work from one processor to another. For simplicity, constant message size is assumed. The time for a serial algorithm to perform the required work is $T_s = W \times U_{calc}$, and the time to perform the work on a parallel processor with p processors is T_p. Efficiency is therefore given by $E = T_s/pT_p$. For efficiency to be constant, T_s should grow as $\Theta(pT_p)$. As U_{calc} is constant, W should also grow as $\Theta(pT_p)$. For $T_p = \Omega(G(p))$, therefore, there is a lower bound $\Omega(pG(p))$ on the isoefficiency of the entire parallel algorithm, including the work distribution algorithm. For a $HiC_{(k,h)}$ there is a lower bound $\Omega(2h-1) = \Omega(\log p)$ for the work to be distributed to all processors, as this is the diameter of the network. Thus there is a lower bound on the execution time of $\Omega(\log p)$, and a lower bound on the isoefficiency function of $\Omega(p \log p)$. The hypercube also has a lower bound on the isoefficiency function of $\Omega(p \log p)$, while isoefficiency of load balancing on the $2D$-mesh has a lower bound of $\Omega(p\sqrt{p})$ [Kumar and Rao(1987)].

In order to establish an upper bound on the isoefficiency function, efficiency must be determined. Efficiency $E = T_s/pT_p = 1/(1 + (T_o/T_s))$. Since $T_s = W \times U_{calc} = \Theta(W)$, it remains to determine T_o. The HS algorithm executes in two stages. In the first

stage, the available work is distributed from the originating processor to all the other processors. At any given time t, some processor has the maximum workload $\max w_t$. After a target processor with workload w services a request for work, both the target and receiver processors will have a maximum workload of $(1-\gamma)w$, unless $w \leq \varepsilon$. If, at time $t+1$, every processor with work has received at least one request for work, then $\max w_{t+1} \leq (1-\gamma)\max w_t$. At time $t=1$ the maximum workload $\max w_1 = W$. The first stage takes $\Theta(\log p)$ steps, therefore at the end of the first stage the maximum workload $\max w_{\log p} \leq (1-\gamma)^{\Theta(\log p)}W$. The overhead T_{o1} for the first stage is determined below.

$$\begin{aligned} T_{o1} &= 1+2+\sum_{x=1}^{h-1} 2x2^{2x} + \sum_{x=1}^{h-1} 2x2^{2x+1} \\ &= 3+6\sum_{x=1}^{h-1} x4^x \\ &= 3+6((h-1)4^{h+1} - h4^h + 4)/(4-1)^2 \\ &= \Theta(h4^{h+1}) \\ &= \Theta(p\log p) \end{aligned}$$

In the second stage, processors which become idle request work from other processors, in order to maintain the load balance. This process continues until $\max w \leq \varepsilon$. Assume that every processor in the system receives at least one request for work in every $V(p)$ requests. Since $\max w_{\log p} \leq (1-\gamma)^{\Theta(\log p)}W$, after $V(p)$ requests the maximum workload is $\max w_{\log p+1} \leq (1-\gamma)^{\log p+1}W$, and after $xV(p)$ requests it has become $\max w_{\log p+x} \leq (1-\gamma)^{\log p+x}W$. We wish to determine x for $(1-\gamma)^{\log p+x}W = \varepsilon$. Therefore $\log p + x = \log_{1-\gamma} \varepsilon/W$ which transforms to $x = \log_{1/(1-\gamma)} W/\varepsilon - \log p$. In the second stage, therefore, there are $(\Theta(\log W) - \Theta(\log p))V(p)$ requests required, so that overhead $T_{o2} \approx U_{comm}V(p)(\Theta(\log W) - \Theta(\log p))$. For an approximation of U_{comm},

sum the distances to all possible targets and divide by the number of targets. This yields

$$\begin{aligned} U_{comm} &\approx (2 + 2\sum_{y=1}^{h-1} 2y)/2h \\ &= \Theta(h) \\ &= \Theta(\log p). \end{aligned}$$

It remains to determine $V(p)$. In the worst case the value of $V(p)$ is unbounded for the HS strategy. However, the actual amount of work assigned to any processor is unknown. In the second stage, therefore, processors become idle randomly, depending on the actual size of the work they have been assigned. Since the requesting processor is random, the target processor must also be random. In [Kumar and Rao(1987)] a value of $V(p) = \Theta(p \log p)$ is computed for the (RP) algorithm, which targets processors randomly. The second stage of the HS algorithm therefore has $V(p) = \Theta(p \log p)$. The overhead of the second stage can now be determined.

$$\begin{aligned} T_{o2} &= \Theta(\log p) \times \Theta(p \log p)(\Theta(\log W) - \Theta(\log p)) \\ &= \Theta(p \log^2 p(\log W - \log p)) \end{aligned}$$

The total overhead $T_o = T_{o1} + T_{o2}$. As overhead for the second stage asymptotically dominates the overhead for the first stage, total overhead is $T_o = \Theta(p \log^2 p(\log W - \log p))$. Since $T_s = \Theta(W)$, we can solve directly for isoefficiency.

$$\begin{aligned} W &= p \log^2 p(\log W - \log p) \\ \log W &= \log p + \log \log^2 p + \log \log(W/p) \\ W &= p \log^2 p(\log \log^2 p + \log \log(W/p)) \end{aligned}$$

Table 6.1:

Isoefficiency Functions of Dynamic Load Balancing Algorithms on Hypercube.				
(ARR)	(NN)	(GRR)	(RP)	(SB)
$\mathcal{O}(p^2 \log^2 p)$	$\Omega(p^{\log(1+1/\gamma)-1}$	$\mathcal{O}(p^2 \log p)$	$\mathcal{O}(p \log^3 p)$	$\mathcal{O}(p^2 \log p)$

Ignoring the log log term, the isoefficiency function is

$$W = \mathcal{O}(p \log^2 p \log \log^2 p).$$

Table 6.1 shows isoefficiency expressions for a number of dynamic load balancing algorithms on the hypercube, as determined in [Kumar and Rao(1987)]. Both the asynchronous and global round robin algorithms are shown to be much less scalable than the HS algorithm. The GRR algorithm has scalability determined by the bottleneck which results from use of a central target allocation scheme. The scheduler based algorithm also suffers from a bottleneck caused by contention for use of a central resource, and has the same scalability has the (GRR) algorithm. The random polling algorithm has scalability slightly inferior to the HS algorithm, though average performance could be expected to be lower as it does not take advantage of locality to reduce communication expenses. In general, the HS algorithm on the $HiC_{(4,h)}$ offers superior scalability to several dynamic load balancing algorithms on the hypercube.

6.3.2 Multiple Queues

Not all irregular algorithms involve a single processor with a quantity of work which must be distributed amongst the processors. In some algorithms work will be created at processors as the algorithm progresses. In such an algorithm there may be several sources of new work scattered throughout the parallel computer in an irregular fashion. The emphasis in the HS algorithm on targeting local processors means the distributed work queues are used to advantage, reducing the average distance of com-

munication. Dynamic load balancing schemes which cause idle processors to target processors uniformly throughout the parallel computer fail to achieve the performance benefits possible from such distributed work sources. Section 6.4 describes an irregular algorithm which generates work at processors throughout the computer.

6.4 An Example Algorithm

In order to demonstrate the use and advantages of the HS algorithm on the $HiC_{(4,h)}$, this section describes an irregular algorithm. The parallel network (PN) algorithm is a new parallel, deterministic algorithm for finding subgraph isomorphisms from a database of attributed, directed model graphs to an attributed, directed input graph [Campbell and Kumar(1998)]. The algorithm decomposes the model graphs and forms the resultant subgraphs into a number of search networks. Subgraphs common to any number of model graphs are represented only once. This approach allows rapid, parallel detection of mappings of common subgraphs onto the input graph. In parallel, all mappings found for each model graph are searched to detect complete, consistent mappings, which define subgraph isomorphisms.

When used in conjunction with the HS algorithm on a hierarchical interconnection network, the PN algorithm allows local communication to be used to advantage, reducing communication overheads and improving performance.

6.4.1 The subgraph isomorphism problem

There are many situations where, given two graphs G and H, one may wish to answer the question, does H contain a subgraph isomorphic to G? For example, subgraph isomorphism detection allows a variety of search functions to be carried out on graphs representing chemical compounds [Benstock *et al.*(1988)] [Artymiuk *et al.*(1994)], and

is used to search for models or prototypes in images [Bunke and Allermann(1983)] [Wong(1992)].

A special case of the subgraph isomorphism problem is the graph isomorphism problem; is H isomorphic to G? The graph isomorphism problem has had polynomial algorithms proposed for several special classes of graphs [Chen(1996)] [Galil *et al.*(1987)] [Hsu(1995)] [Jaja and Kosaraju(1988)]. Various special cases of the subgraph isomorphism problem are also known, for instance a polynomial algorithm for subgraph isomorphism when G is a tree and H is a forest was reported in [Matula(1978)], and another for two-connected series-parallel graphs in [Lingas and Syslo(1988)]. More recently a polynomial algorithm when G is a *bounded tree-width* graph, where $|V(G)| = \mathcal{O}(\log |V(H)|)$, was reported in [Alon *et al.*(1995)]. However the subgraph isomorphism problem in general is known to be NP-complete [Garey and Johnson(1979)].

Two broad categories of methods exist for solving the subgraph isomorphism problem.

i) Nondeterministic methods which use nonlinear optimisation techniques (or heuristic approximations thereof) to reach approximate solutions. Programs employing these methods run in less than exponential time, but may not reach a solution with a satisfactory degree of accuracy.

ii) Combinatorial algorithms which are guaranteed to yield the correct solution, but in the worst case have exponential complexity. These algorithms search the solution space, relying on bounding or pruning techniques to reduce the size of the search space to a polynomial order under suitable conditions.

The most suitable method of solution depends on the problem to be solved. Under some circumstances it may be desirable to find any one subgraph of H which is isomorphic to G, regardless of the number of subgraphs of H which may be isomorphic to G. However, searching for all possible subgraph isomorphisms is the most general problem. This implies use of combinatorial methods. A widely used combinatorial method is the

sequential tree-search algorithm described in [Ullmann(1976)], commonly known as the Ullmann algorithm.

An extension of the subgraph isomorphism problem involves searching for subgraph isomorphisms from several 'model' graphs to a single 'input' graph. Messmer and Bunke proposed a two part algorithm to solve this problem in [Messmer and Bunke(1993)]. An "off-line" part of the algorithm involved constructing a search network from subgraphs of all the model graphs. During the "online" portion of the algorithm the network is used to find successively larger subgraphs within the input graph until subgraph isomorphisms from entire model graphs are identified. This approach allowed them to take advantage of common elements within the model graphs, thereby reducing duplication of effort during the "online" search process.

As well as investigating improved serial algorithms, work has also been done on developing parallel algorithms, to allow large graph problems to be solved in reasonable time using powerful parallel computers [Rao and Kumar(1987), Costanzo *et al.*(1986), Crowl *et al.*(1994), Chen(1996)]. There are many avenues for parallelisation of the Ullmann algorithm. Even the original sequential algorithm packed binary vectors into a single word and used bitwise logic in order to achieve some level of parallelism in a single processor environment. This could be described as word parallelism. The major forms of parallelism available could be described as tree-node parallelism and branch parallelism. Tree-node parallelism uses multiple processors to speed the computation at a single tree-node of the search tree. Branch parallelism examines multiple branches of the tree in parallel. Tree-node parallelism tends to result in single branches being descended more rapidly, which can result in a single solution being found faster. Branch parallelism can result in the entire search space being covered more rapidly. The results of a theoretical and experimental study of the effect on performance of the various forms of parallelism were reported in [Crowl *et al.*(1994)]. They showed that the best choice of parallelisation depended on the number of solutions required and

the distribution of solutions within the search space, as well as the architecture of the machine on which the algorithm is implemented. In general, they found it preferable to be able to take advantage of all forms of parallelism as the occasion demanded.

The parallel network (PN) algorithm is a new parallel, deterministic algorithm for finding subgraph isomorphisms from a database of attributed, directed model graphs to an attributed, directed input graph [Campbell and Kumar(1998)]. The principles involved could, however, be easily extended to other forms of graph, particularly weighted digraphs. The PN algorithm offers several advantages. Like the algorithm of Messmer and Bunke it has an off-line and an online stage. This allows a database of model graphs to developed off-line, then matched against input graphs online without the overhead of the off-line stage. At an early stage of the PN algorithm a simple but effective check is performed to detect model graphs which cannot be subgraph isomorphic to the input graph, eliminating them and preventing wasted computation. In its final stage, the PN algorithm employs a tree search similar to that of the Ullmann algorithm. An advantage of the PN algorithm is that each tree-node incorporates more information than a tree-node in the Ullmann algorithm, which allows lookahead techniques to be employed more effectively. Also, in the PN algorithm the number of tree-nodes at each level of the tree is known in advance. This allows levels with the least nodes to be searched first, reducing the best case complexity of the search. The PN algorithm allows effective use of multiple forms of parallelisation and uses dynamic load balancing to achieve efficient use of resources under variable load conditions.

6.4.2 Algorithm design

For a set of attributed directed model graphs and an attributed directed input graph, we wish to identify any and all subgraph isomorphisms between any of the model graphs and the input graph. Informally, the PN algorithm can be described in the following way.

i) For each model graph, break it into subgraphs. Every vertex which has outgoing edges becomes the centre of a subgraph. Every subgraph consists of a central vertex, all outward edges incident to the central vertex, and the vertices they lead to. For a connected graph every vertex will be represented in at least one subgraph and for most interesting graphs many vertices will be represented in several subgraphs.

ii) Subgraphs are compiled into search networks. All subgraphs whose central vertices have the same attribute are compiled into the same search network. All subgraphs which are isomorphic with each other will be represented only once within a search network.

iii) The input graph is broken into subgraphs, in the same manner as the model graphs in step (1) above.

iv) For each input subgraph, determine which model subgraphs are subgraph isomorphic with it, using the search networks. This step determines the presence (or absence) of subgraph isomorphisms, but not the mappings θ which define the isomorphisms (recall that, for simple graphs, θ implies ϕ).

v) For each model graph, check that each subgraph is subgraph isomorphic with at least one input subgraph. If not, then no subgraph isomorphism can exist from the model graph to the input graph, so terminate the search for this model graph.

vi) For each subgraph of each model graph, determine all mappings onto those input subgraphs for which the search networks have determined a subgraph isomorphism exists.

vii) For each model graph, check for complete, consistent mappings from model subgraphs onto input subgraphs. A complete mapping contains a mapping from every model subgraph. A consistent mapping has no two vertices in the model graph mapped to a single vertex in the input graph, and no vertex in the model

graph mapped to more than one vertex in the input graph. Each complete, consistent mapping determines a subgraph isomorphism between the model graph and the input graph.

The overall operation of the PN algorithm is shown if Figure 6.2.

The algorithm is shown more formally in Algorithms 10 to 21. At the top level, Procedure PN receives as arguments a set of attributed, directed model graphs $\mathcal{M} = D_1, \ldots, D_L$, a set of search networks $\mathcal{N}$ and an attributed, directed input graph I.

Procedure PN($\mathcal{M}, \mathcal{N}, I$)
Call Off-line($\mathcal{M}, \mathcal{N}$)
Call Online($\mathcal{N}, I$)

Algorithm 10: PN algorithm

The algorithm has a pre-processing or off-line stage, comprised of steps one and two, and a runtime or online stage, comprising steps three through seven. Maximum advantage is taken of the two stage algorithm if the same set of search networks can be used repeatedly, with many different input graphs. The top level algorithm described here deals only with a single input, but the extension to multiple inputs is trivial. The off-line stage, described in Algorithm 11, simply calls Function *MakeSubs*, described in Algorithm 12, and Procedure *MakeNetworks*, described in Algorithm 13.

6.4.2.1 The Off-line algorithm

Procedure Off-line($\mathcal{M}, \mathcal{N}$)
$\mathcal{S}_{\mathcal{M}}$ = MakeSubs($\mathcal{M}$)
Call MakeNetworks($\mathcal{S}_{\mathcal{M}}, \mathcal{N}$)

Algorithm 11: Decompose model graphs into search networks

Function *MakeSubs* creates a set $\mathcal{S}_{\mathcal{M}}$ of subgraphs from the model graphs in $\mathcal{M}$. In Figure 6.3, $\mathcal{M} = \{D, E\}$ and $\mathcal{S}_{\mathcal{M}} = \{D'_1, D'_2, D'_3, E'_1, E'_2, E'_3\}$. Each subgraph D' has a vertex known as the *central vertex* $v_C(D')$, which is the tail of all the edges in the

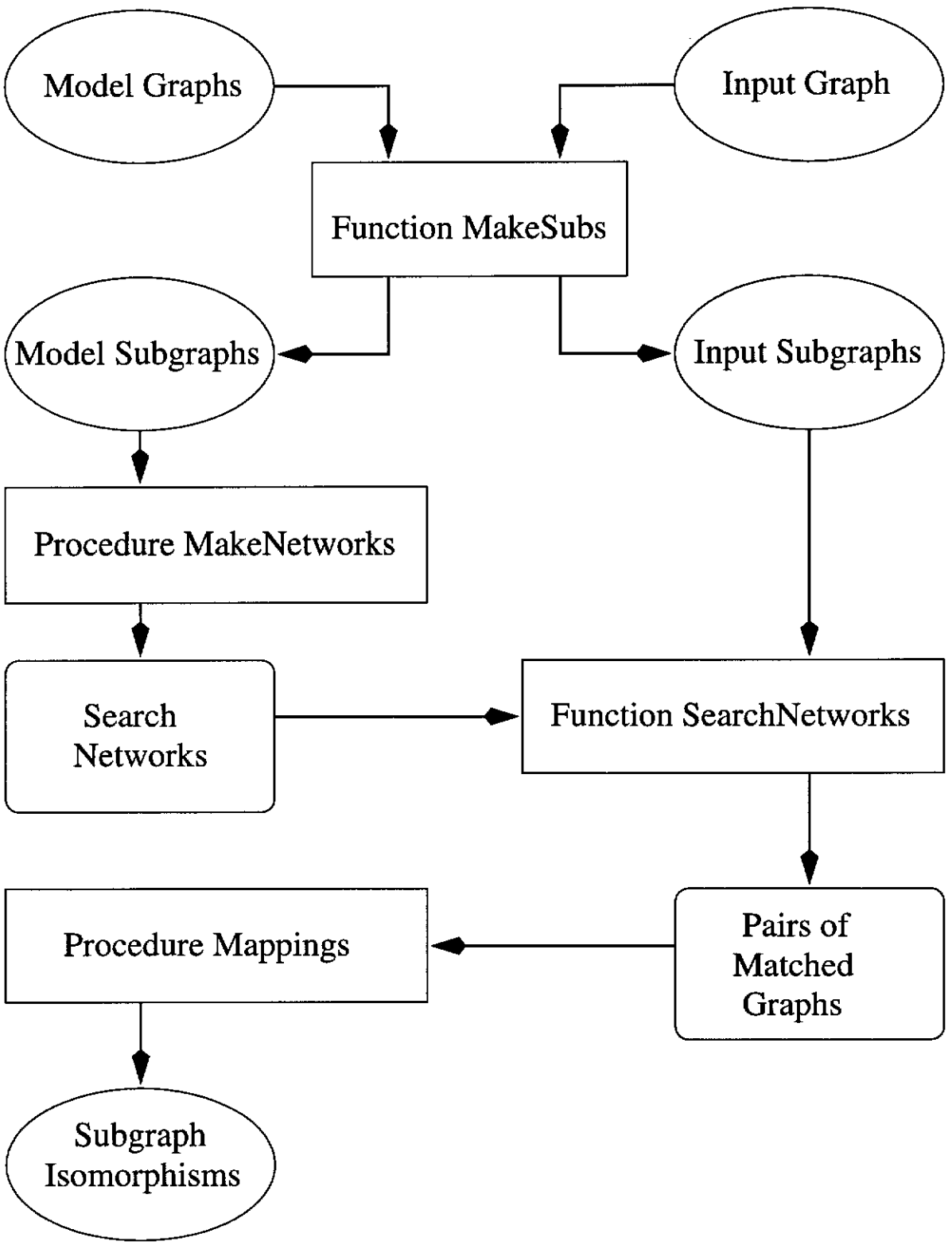


Figure 6.2: The PN algorithm.

subgraph. All other vertices in $V(D')$ are known as *satellite vertices.* For example, in Figure 6.3, subgraph D'_1 has central vertex $v1$ with attribute a. Subgraph D'_1 has $v2$ and $v3$ as satellite vertices.

Function MakeSubs($\mathcal{M}$)
$\mathcal{S} = \emptyset$
for all $D \in \mathcal{M}$ **do**
  for all $\mu \in V(D)$ **do**
    if $e \in E(D)$ such that **tail**$(e) = \mu$ **then**
      Construct graph $D' \subseteq D$ such that:
      $\mu \in V(D')$
      $e \in E(D') \iff e \in E(D)$ and **tail**$(e) = \mu$
      $\nu \in V(D') \iff e \in E(D')$ and **head**$(e) = \nu$
      $\mathcal{S} = \mathcal{S} + D'$.
Return $\mathcal{S}$

Algorithm 12: Decompose model graphs into subgraphs

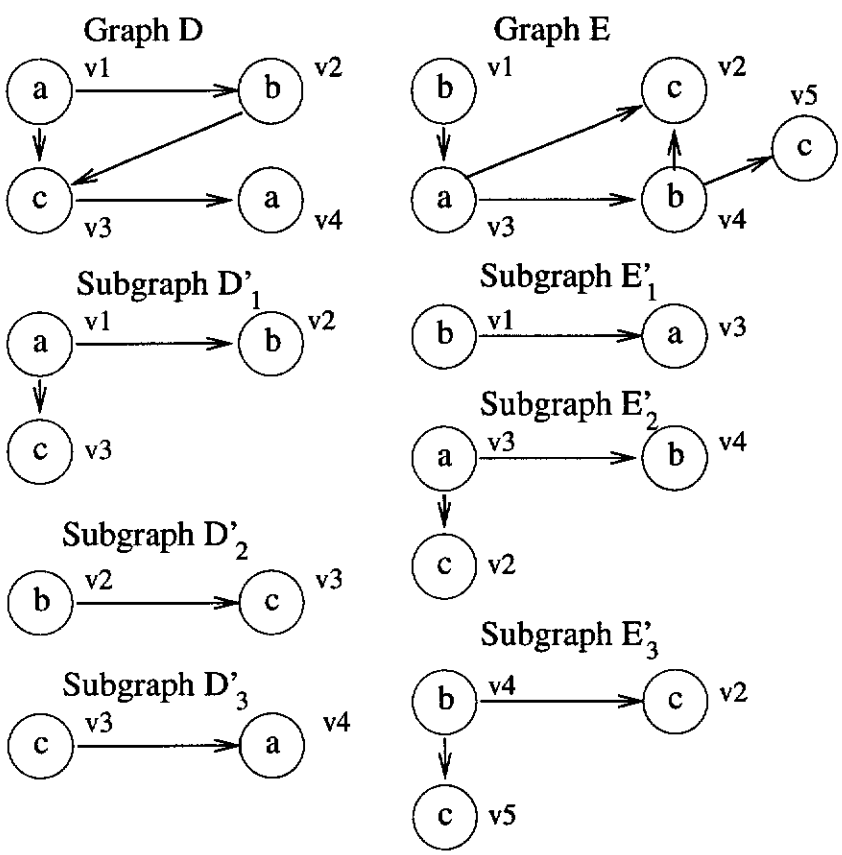


Figure 6.3: Subgraphs of graphs D and E

Procedure *MakeNetworks* transforms $\mathcal{S}_{\mathcal{M}}$ into a set of search networks $\mathcal{N}$ containing zero or more search networks η_j. Each search network $\eta_j \in \mathcal{N}$ is an attributed, directed graph which represents those subgraphs in $\mathcal{S}_{\mathcal{M}}$ having a central vertex with attribute j. As an example, Figure 6.4 shows the search networks which represent all the subgraphs from Figure 6.3.

The attribute $\alpha_{\eta_j}(\mu)$ of a vertex $\mu \in V(\eta_j)$ is an ordered pair $(\mathcal{AO}(\mu), \mathcal{D}'(\mu))$. With A representing the set of model and input graph vertex attributes, Element $\mathcal{AO}(\mu)$ is a set of ordered pairs, with each pair (att, ord) consisting of an attribute $att \in A$, and ord, the number of times that attribute att occurs at this search network vertex. The *size* of a vertex μ, given by $\mathbf{size}(\mu)$, is the sum of the values of ord for each pair in $\mathcal{AO}(\mu)$. Taken in conjunction with the known attribute j of the central vertex, and the restricted structure of the subgraphs in $\mathcal{S}_{\mathcal{M}}$, $\mathcal{AO}(\mu)$ contains sufficient information to describe an entire attributed digraph. Looking at Figure 6.4, it can be seen that vertex $v3$ of η_a has $\mathcal{AO}(v3) = \{(b,1),(c,1)\}$. Thus vertex $v3$ of η_a represents a graph with a central vertex with attribute a and two satellite vertices with attributes b and c. The second element of $\alpha_{\eta_j}(\mu)$ is $\mathcal{D}'(\mu)$; which is the set, possibly empty, of all subgraphs D' which are isomorphic to the graph represented by $\mathcal{AO}(\mu)$ and j. Looking again at vertex $v3$ of η_a in Figure 6.4, we see that two subgraphs, D'_1 and E'_2, are isomorphic to the graph $v3$ represents.

The direction of the edges in $E(\eta_j)$ indicates a parent/child relationship between vertices; if edge $(\mu,\nu) \in E(\eta_j)$, then vertex μ is a parent of vertex ν in search network η_j. If μ is a parent of ν then the graph represented by the first element of $\alpha_{\eta_j}(\mu)$ is subgraph isomorphic to the graph represented by the first element of $\alpha_{\eta_j}(\nu)$. In Figure 6.4 it can be seen that vertex $v1$ is a parent of $v3$ in η_a. Vertex $v1$ represents a graph with a central vertex with attribute a and a satellite vertex with attribute b, which is subgraph isomorphic to the graph represented by vertex $v3$.

For each search network η_j there is a set of vertices $R(\eta_j) \subseteq V(\eta_j)$ the elements of

which are *root vertices*. A root vertex has no parent vertex. The first element $\mathcal{AO}$ of the attribute of a root vertex contains only a single ordered pair (att, ord), with $ord = 1$. The element att for any root vertex μ is given by $\mathbf{att}(\mu)$. In Figure 6.4 it can be seen that vertices $v1$ and $v2$ of η_a are root vertices, with $\mathbf{att}(v1) = b$ and $\mathbf{att}(v2) = c$.

Search networks are superficially similar to those used in the Messmer and Bunke algorithm, but there are some significant differences. In the Messmer and Bunke algorithm graphs are partitioned into vertex disjoint subgraphs, whereas in the PN algorithm the subgraphs are edge disjoint. Also, the Messmer and Bunke algorithm forms a single search network from all the subgraphs of all the models. In the PN algorithm only subgraphs with a common attribute for the central vertex are combined into a search network. This results in multiple search networks being formed, up to a maximum of $|A|$, the size of the set of attributes.

Procedure MakeNetworks($\mathcal{S}, \mathcal{N}$)
for all $D' \in \mathcal{S}$ **do**
  $j = \alpha_{D'}(v_C(D'))$
  if $\eta_j \in \mathcal{N}$ **then**
    Call AddToNetwork(η_j,D')
  else
    Call CreateNewNetwork($\mathcal{N}$, D')

Algorithm 13: Combine subgraphs into search networks

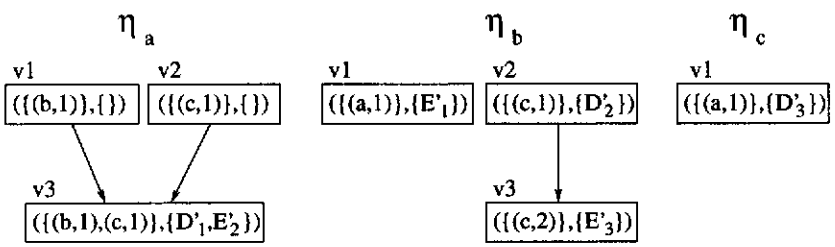


Figure 6.4: Search network for model graphs D and E

Procedure AddSubgraphToNetwork(η_j, D')
Create a search network vertex μ
$\mathcal{AO}(\mu) = \emptyset$
$\mathcal{D}'(\mu) = D'$
for all $v \in V(D')$ such that $v \neq v_C(D')$ **do**
  if $(att, ord) \in \mathcal{AO}(\mu)$ such that $att = \alpha_{D'}(v)$ **then**
    $ord = ord + 1$
  else
    $\mathcal{AO}(\mu) = \mathcal{AO}(\mu) + (\alpha_{D'}(v), 1)$
Call AddToNetwork(η_j, μ)

Algorithm 14: Combine subgraphs into an existing search network

Procedure AddToNetwork(η_j, μ)
if $\nexists (att, ord) \in \mathcal{AO}(\mu)$ such that $att = \mathbf{att}(\nu)$ for any $\nu \in R(\eta_j)$ **then**
  $V(\eta_j) = V(\eta_j) + \mu$
  Call ExpandNetwork(η_j, μ)
else
  Find $\nu \in V(\eta_j)$ such that max(**size**(ν)) and $\mathcal{AO}(\nu) \subseteq \mathcal{AO}(\mu)$.
  if **size**(ν) = **size**(μ) **then**
    $\mathcal{D}'(\nu) = \mathcal{D}'(\nu) \cup \mathcal{D}'(\mu)$
  else
    $V(\eta_j) = V(\eta_j) + \mu$.
    $E(\eta_j) = E(\eta_j) + (\nu, \mu)$
    Create a new search network vertex v
    for all $(att, ord) \in \mathcal{AO}(\mu)$ **do**
      $\mathcal{AO}(v) = \mathcal{AO}(v) + (att, ord(\mu) - ord(\nu))$
    Call AddToNetwork2(η_j, v, μ)

Algorithm 15: Combine search vertices into search networks: part 1

Procedure AddToNetwork2(η_j, μ, ω)
if $\nexists(att, ord) \in \mathcal{AO}(\mu)$ such that $att = \mathbf{att}(\nu)$ for any $\nu \in R(\eta_j)$ **then**
  $V(\eta_j) = V(\eta_j) + \mu$
  $E(\eta_j) = E(\eta_j) + (\mu, \omega)$
  Call ExpandNetwork(η_j, μ)
else
  Find $\nu \in V(\eta_j)$ such that max(**size**(ν)) and $\mathcal{AO}(\nu) \subseteq \mathcal{AO}(\mu)$.
  if **size**(ν) = **size**(μ) **then**
    $\mathcal{D}'(\nu) = \mathcal{D}'(\nu) \cup \mathcal{D}'(\mu)$
    $E(\eta_j) = E(\eta_j) + (\nu, \omega)$
  else
    $V(\eta_j) = V(\eta_j) + \mu$.
    $E(\eta_j) = E(\eta_j) + (\mu, \omega)$
    $E(\eta_j) = E(\eta_j) + (\nu, \mu)$
    Create a new search network vertex v
    for all $(att, ord) \in \mathcal{AO}(\mu)$ **do**
      $\mathcal{AO}(v) = \mathcal{AO}(v) + (att, ord(\mu) - ord(\nu))$
    Call AddToNetwork2(η_j, v, μ)

Algorithm 16: Combine search vertices into search networks: part2

Procedure CreateNewNetwork($\mathcal{N}, D'$)
$j = \alpha_{D'}(v_C(D'))$
Create a network η_j
$\mathcal{N} = \mathcal{N} + \eta_j$
Create a search network vertex μ with $\mathcal{AO}(\mu) = \emptyset$ and $\mathcal{D}'(\mu) = D'$.
for all $\nu \in V(D')$ such that $\nu \neq v_C(D')$ **do**
  if $(att, ord) \in \mathcal{AO}(\mu)$ such that $att = \alpha_{D'}(\nu)$ **then**
    $ord = ord + 1$
  else
    $\mathcal{AO}(\mu) = \mathcal{AO}(\mu) + (\alpha_{D'}(\nu), 1)$
$V(\eta_j) = V(\eta_j) + \mu$
Call ExpandNetwork(η_j, μ)

Algorithm 17: Create a search network

Procedure ExpandNetwork(η_j, μ)
Given $\mathcal{AO}(\mu) = \{(att_1, ord_1), \ldots, (att_k, ord_k)\}$
if $k > 1$ **then**
  Create a search network vertex ν
  $\mathcal{AO}(\nu) = \{(att_1, ord_1)\}$
  $\mathcal{D}'(\nu) = \emptyset$
  $V(\eta_j) = V(\eta_j) + \nu$
  $E(\eta_j) = E(\eta_j) + (\nu, \mu)$
  Create a search network vertex υ
  $\mathcal{AO}(\upsilon) = \{(att_2, ord_2), \ldots, (att_k, ord_k)\}$
  $\mathcal{D}'(\upsilon) = \emptyset$
  $V(\eta_j) = V(\eta_j) + \upsilon$
  $E(\eta_j) = E(\eta_j) + (\upsilon, \mu)$
  Call ExpandNetwork(η_j, ν)
  Call ExpandNetwork(η_j, υ)
else
  if $ord_1 > 1$ **then**
    Create a search network vertex ν
    $\mathcal{AO}(\nu) = \{(att_1, 1)\}$
    $\mathcal{D}'(\nu) = \emptyset$
    $V(\eta_j) = V(\eta_j) + \nu$
    $E(\eta_j) = E(\eta_j) + (\nu, \mu)$

Algorithm 18: Expand a search network

6.4.2.2 The Online algorithm

The algorithm of Procedure *Online* is described in Algorithm 19 for a directed, attributed input graph I and a set of search networks $\mathcal{N}$. It calls Function *MakeSubs*, described in Algorithm 12, to create a set $\mathcal{S}_I$ of subgraphs from the input graph. Figure 6.5 shows an example input graph and the subgraphs derived from it. Function *SearchNetworks*, shown in Algorithm 20, then uses the search networks in $\mathcal{N}$ to determine whether subgraph isomorphisms exist between any model subgraphs and the input subgraphs in $\mathcal{S}_I$. Subgraph isomorphisms are searched for in search networks made up of model subgraphs with the same attribute for the central vertex. For an input subgraph I' with $\alpha_{I'}(v_C(I')) = j$, each root vertex μ of η_j is *visited* once for each satellite vertex ν of I' with $\alpha_{I'}(\nu) = \mathbf{att}(\mu)$. The number of times vertex μ has been visited is given by $\mathbf{visit}(\mu)$. Once all satellite vertices of I' have been considered, the search visits all vertices of η_j which are *eligible*. Two types of vertex are eligible.

i) A vertex μ where $AO(\mu)$ contains only a single ordered pair (att, ord) with $ord > 1$ has only a single parent, the root node ν where $\mathbf{att}(\nu) = att$. Vertex μ is eligible if $ord \leq \mathbf{visit}(\nu)$.

ii) A vertex μ where $AO(\mu)$ contains more than one ordered pair has two parents. Vertex μ is eligible if both parents have been visited.

The search continues to visit eligible vertices until no more can be found. No checking for the existence of edges or vertices linking the subgraphs represented by the parent nodes is required. This means that the network can very rapidly find all model subgraphs which are subgraph isomorphic to the input subgraph. The output from Function *SearchNetworks* is a set $\mathcal{P}$ of ordered pairs of model subgraphs and input subgraphs to which they are subgraph isomorphic. For the example input graph shown in Figure 6.5 and the search networks shown in Figure 6.4 the result of SearchNetworks

is

$$\mathcal{P} = \{(D'_3, I'_1), (D'_3, I'_2), (D'_2, I'_3), (E'_3, I'_3), (D'_1, I'_4), (E'_2, I'_4)\}. \quad (6.1)$$

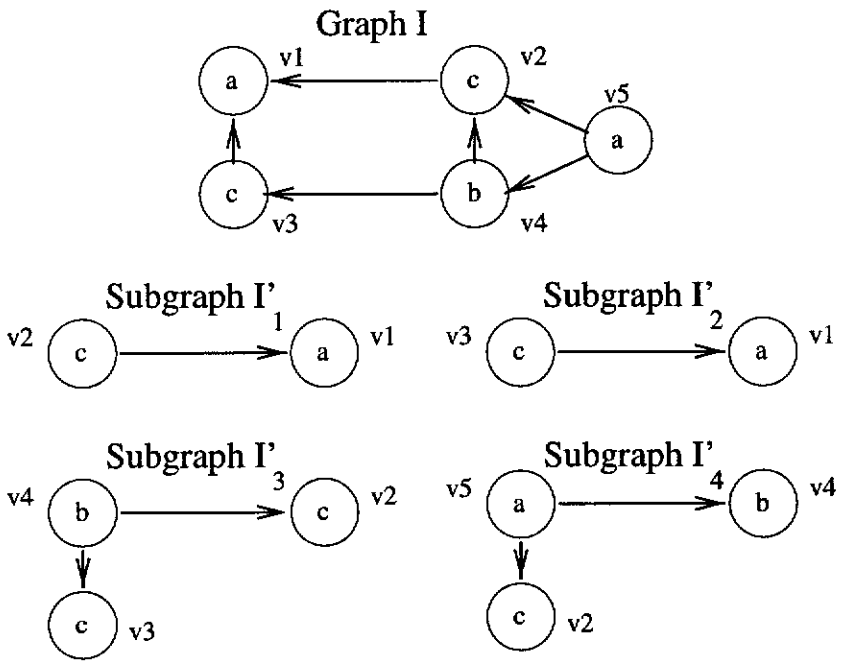


Figure 6.5: Subgraphs of input graph I

Procedure Online($\mathcal{N}, I$)
$\mathcal{S}_I$ = MakeSubs($\{I\}$)
$\mathcal{P}$ = SearchNetworks($\mathcal{S}_I, \mathcal{N}$)
Call Mappings($\mathcal{M}, \mathcal{S}_\mathcal{M}, \mathcal{P}$)

Algorithm 19: Determine subgraph isomorphisms from search networks

Procedure *Mappings*, detailed in Algorithm 21, is a three step algorithm, corresponding to steps 5, 6 and 7 in the informal description. The first step determines, for each model graph D in $\mathcal{M}$, whether all the subgraphs of D are subgraph isomorphic to at least one subgraph of the input graph I. If not then D cannot be subgraph isomorphic to I, so no further processing is required. Consider the set $\mathcal{P}$ shown in Equation 6.1, which is derived from the example model graphs shown in Figure 6.3 and the example input graph shown in Figure 6.5. For model graph D, all three subgraphs D'_1, D'_2 and D'_3 are subgraph isomorphic to at least one subgraph of I. However, for model graph E

Function SearchNetworks($\mathcal{S}, \mathcal{N}$)
$\mathcal{P} = \emptyset$
for all $I' \in \mathcal{S}$ **do**
  $j = \alpha_{I'}(v_C(I'))$
  if $\eta_j \in \mathcal{N}$ **then**
    for all $v \in V(I')$ such that $v \neq v_C(I')$ **do**
      if $\exists \mu \in R(\eta_j)$ such that **att**$(\mu) = \alpha_{I'}(v)$ **then**
        Visit μ
    for all $\mu \in V(\eta_j)$ **do**
      Given $\mathcal{AO}(\mu) = \{(att_1, ord_1), \ldots, (att_k, ord_k)\}$
      if $k = 1$ and $ord_1 > 1$ **then**
        if $(\nu, \mu) \in E(\eta_j)$ and **visits**$(\nu) \geq ord_1$ **then**
          Visit μ
    repeat
      for all $\mu \in V(\eta_j)$ **do**
        if $(\nu, \mu) \in E(\eta_j)$ and $(\upsilon, \mu) \in E(\eta_j)$ and **visits**$(\nu) \geq 1$ and **visits**$(\upsilon) \geq 1$ **then**
          Visit μ.
    until No new vertices visited
    for all $\mu \in V(\eta_j)$ which are Visited **do**
      for all $D' \in \mathcal{D}'(\mu)$ **do**
        $\mathcal{P} = \mathcal{P} + (D', I')$
Return $\mathcal{P}$

Algorithm 20: Search networks

Procedure Mappings($\mathcal{M}, \mathcal{S}_{\mathcal{M}}, \mathcal{P}$)
for all $D \in \mathcal{M}$ **do**
  INCOMPLETE = FALSE
  for all $D' \in \mathcal{S}_{\mathcal{M}}$ such that $D' \subseteq D$ **do**
    if $\nexists (D', I') \in \mathcal{P}$ **then**
      D can not be subgraph isomorphic to I
      INCOMPLETE = TRUE
    else
      for all $(D', I') \in \mathcal{P}$ **do**
        Determine all possible mappings from D' to I'
  if NOT INCOMPLETE **then**
    Perform a tree search of mappings to determine any which are complete and consistent.
    Output all complete consistent mappings found.

Algorithm 21: Create mappings between subgraphs and detect subgraph isomorphisms

subgraph E'_1 is not subgraph isomorphic to any subgraphs of the input graph, so model graph E need not be considered further.

If every subgraph of D is subgraph isomorphic to at least one subgraph of I the second stage of the algorithm is entered. The second stage determines, for each pair of subgraphs $(D', I') \in \mathcal{P}$ with $D' \subseteq D$, all possible mappings from D' to I'. In the example graphs, the mappings from the subgraphs of model D onto I are shown in Figure 6.6. Note that, while D'_3 maps onto two separate subgraphs, I'_1 and I'_2, subgraph D'_2 has two distinct mappings onto a single subgraph, I'_3. For graphs with high vertex degree, the number of mappings from a single model subgraph to a single input subgraph may be large.

Subgraph D'$_1$	1 --- 5 2 --- 4 3 --- 2	
Subgraph D'$_2$	2 --- 4 3 --- 2	2 --- 4 3 --- 3
Subgraph D'$_3$	3 --- 3 4 --- 1	3 --- 2 4 --- 1

Figure 6.6: Mappings of subgraphs of graph D onto input graph I

The third and final stage of the algorithm detects any and all combinations of mappings from subgraphs of model D to input subgraphs which form a single *complete* and *consistent* mapping. A complete mapping contains a mapping from every model subgraph to the input graph. A consistent mapping has no two vertices in the model

graph mapped to a single vertex in the input graph, and no vertex in the model graph mapped to more than one vertex in the input graph. Each complete, consistent mapping determines a subgraph isomorphism between the model graph and the input graph. Searching for complete, consistent mappings can be done using a tree search. Each level of the tree corresponds to a single model subgraph. Each tree-node at a level represents a mapping from the model subgraph to some input subgraph. Expanding a tree-node means comparing the tree-node's mapping with the mapping accumulated from previously expanded tree-nodes. If any inconsistencies occur, the branch is pruned. Otherwise the mapping is added to the accumulated mapping and the search continues down the tree. Looking at Figure 6.6, it can be seen that expanding five tree-nodes discovers a single subgraph isomorphism from D to I, given by the mapping $\theta(v1) = v5, \theta(v2) = v4, \theta(v3) = v2, \theta(v4) = v1$. This is the same basic search method used in the Ullmann algorithm, and the same lookahead techniques commonly employed when using the Ullmann algorithm can be employed here. An advantage of the PN algorithm is that each tree-node incorporates more information than a tree-node in the Ullmann algorithm, which allows lookahead techniques to be employed more effectively. Each tree-node maps not a single vertex, whose neighbours must be checked for consistency, but a group of vertices and edges whose relationships are already known. Further, the number of tree-nodes to be expanded at each level is known in advance. Putting levels with few tree-nodes toward the top of the tree may lead to more effective pruning, and can lead to rapid termination if no consistent mapping can be found.

6.4.3 Parallelism within the PN algorithm

Although designed as a parallel algorithm, the descriptions of the PN algorithm so far have included no explicit parallel statements. There are many opportunities for exploiting parallelism within the PN algorithm. In this section the major areas of

potential parallelism will be described, and the circumstances under which they are most likely to be useful will be discussed. Although the off-line stage of the algorithm is not insignificant, the online stage will generally have the most impact on overall performance. This section considers the parallelisation only of the online stage of the algorithm, shown in Algorithm 19.

Function *MakeSubs*, shown in Algorithm 12, transforms the input graph into a set of subgraphs $\mathcal{S}_I$. The transformation is essentially a binning algorithm, allocating the edges in $E(I)$ with a common originating vertex to bins. This can be done sequentially in linear time. It can also be considered an embarrassingly parallel problem, as it can readily be divided into completely separate sub-problems. For most applications, it would be expected that the number of edges $|E(I)| < 10000$, so it is likely to be effective to parallelise this part of the algorithm only with a small number of tightly coupled processors.

Function *SearchNetworks*, shown in Algorithm 20, uses the search networks in $\mathcal{N}$ to determine whether subgraph isomorphisms exist between any model subgraphs and the input subgraphs in $\mathcal{S}_\mathcal{I}$. During the off-line stage, search networks are allocated to processors. Online, each input subgraph may be allocated to a separate processor with the appropriate network. Each subgraph can be processed independently of all others. Since the number of input subgraphs in $\mathcal{S}_I$ is less than or equal to $|V(I)|$, the maximum parallelism available is of order $|V(I)|$.

Procedure *Mappings*, detailed in Algorithm 21, involves three steps for each model graph in $\mathcal{M}$. Since each model graph can be treated independently for this procedure, individual models can be allocated to separate PEs. Thus there is already parallelism of the order of $|\mathcal{M}|$ available. The first step determines, for each model graph D in $\mathcal{M}$, whether all the subgraphs of D are subgraph isomorphic to at least one subgraph of the input graph I. This step simply checks the number of input subgraphs which each model subgraph has been found subgraph isomorphic to, and can be done in linear time. The

second stage determines all possible mappings from the model subgraphs onto the input subgraphs they are known to match. Since each pair of model and input subgraphs can be searched for mappings independently, this process readily offers parallelism according to the number of subgraphs in a model and the number of input subgraphs each model subgraph matches. Beyond this, however, each search is readily parallelizable, allowing this stage of the algorithm to use as many processors as the workload efficiently allows. The third stage performs a tree search of the mappings determined in the second stage to determine any complete and consistent mappings from a single model graph D to the input graph I. No more than $|\mathcal{M}|$ tree searches are required, however it is possible to employ either branch parallelism or tree-node parallelism, or both, during this stage. As discussed earlier, it was shown in [Crowl *et al.*(1994)] that the best choice of branch or tree-node parallelisation depended on the number of solutions required and the distribution of solutions within the search space, as well as the architecture of the machine on which the algorithm is implemented.

6.4.4 Parallelising the PN algorithm on the *HiC*

The PN algorithm is a highly irregular algorithm, as the amount of computation and communication required, within the algorithm as a whole and within individual stages, varies substantially with the size and nature of the data sets used, as well as the exact output required. Under these circumstances it is difficult to map the problem onto a parallel architecture in such a way as to match the capabilities of the architecture with the requirements of the problem, since the requirements are highly variable.

The majority of the work of the PN algorithm occurs during procedure *Mappings*. As discussed in the previous section, the two main stages of procedure *Mappings* perform tree searches, and each tree search can be parallelised. Determining how to distribute the workload from the various tree searches which occur during one execution of the PN algorithm requires an efficient, dynamic load balancing algorithm. One approach to this

problem is typified by the parallel depth first search method of [Rao and Kumar(1987)], where available work is maintained on a stack and idle processors request work from the controlling processor. This has proven an effective means of distributing work in order to employ many processors without incurring prohibitive management overheads. In the PN algorithm, however, work may be created at any, or all, processors within the parallel computer. Creating a central stack of work would therefore create large communication and synchronisation overheads.

On a *HiC* network based parallel computer, the HS dynamic load balancing algorithm allows work created at any processor to be distributed rapidly amongst all idle processors in the parallel computer. However, work may be available from a number of processors with the parallel computer. Since idle processors request work first from neighbours, the HS algorithm ensures that work is likely to be obtained from the nearest available source, reducing communication time and improving performance. The HS algorithm on the *HiC* network allows the PN algorithm to function efficiently in parallel, as shown in Figure 6.7.

6.4.5 Complexity

This section analyses the worst case complexity of the PN algorithm and makes some comparisons with existing algorithms. As in the previous section only the online part of the algorithm will be considered. It is worth noting that all combinatorial methods for solving the subgraph isomorphism problem will have worst case complexity which is at least exponential. All algorithms aim to provide acceptable average case performance, generally over as wide a range of inputs as possible. Worst case complexity analysis in this case is not useful as a means of performance comparison. Rather it can be used to predict which types of input are likely to result in poor performance.

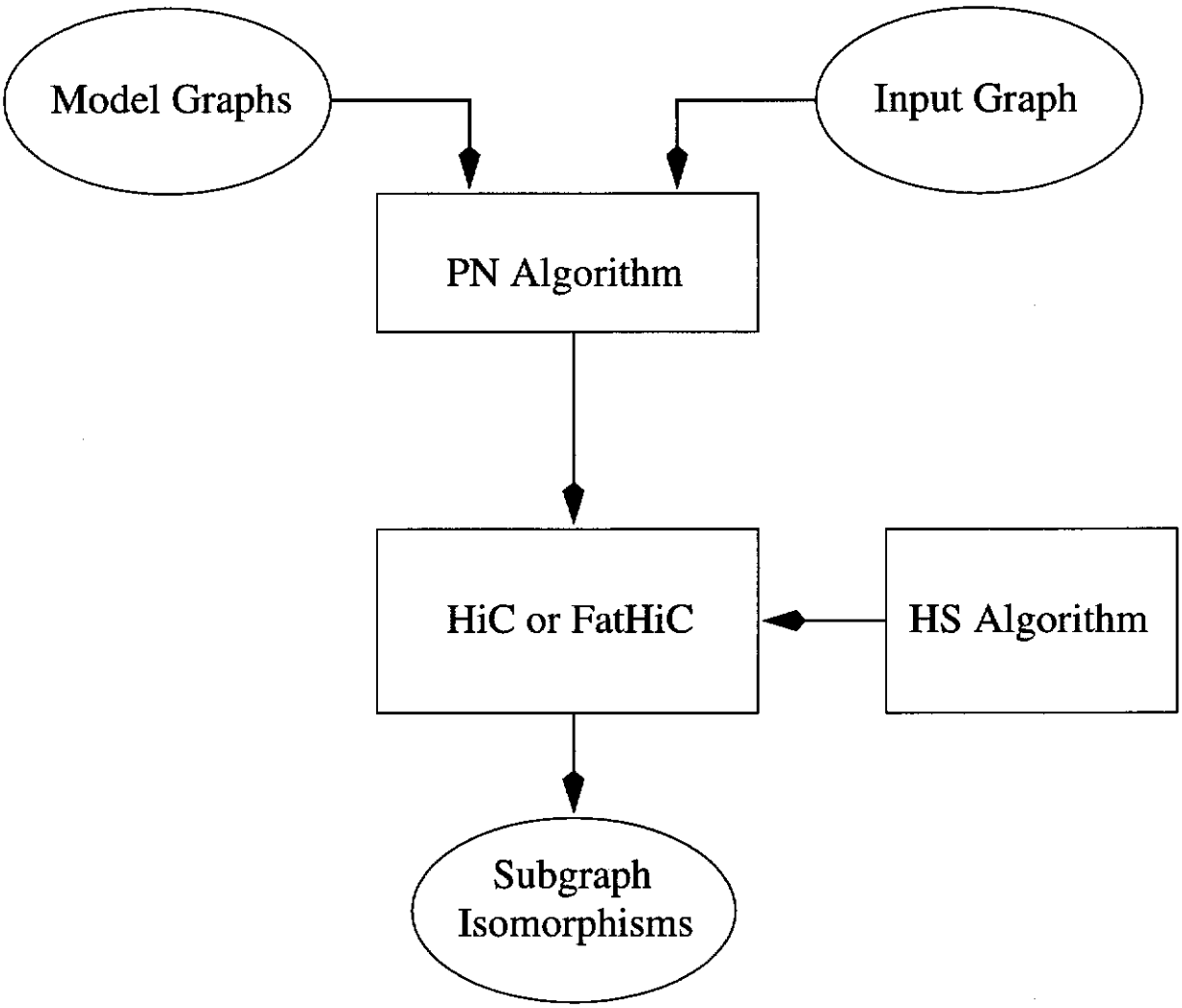


Figure 6.7: The PN and HS algorithms.

We use the following notations:

- $|V(I)|$, the number of vertices in the input graph I.
- $|E(I)|$, the number of edges in the input graph I.
- $|A|$, the number of possible attributes of a model or input graph.
- $|\mathcal{M}|$, the number of model graphs.
- $|\mathcal{S}_I|$, the number of subgraphs in the input graph.
- $|\mathcal{S}_\mathcal{M}|$, the total number of subgraphs in all model graphs.

- $|R(\eta_j)|$, the number of root vertices in search network η_j.
- $|V(\eta_j)|$, the total number of vertices in search network η_j.
- $V_m = \max(|V(D)|)\forall D \in \mathcal{M}$, the maximum number of vertices in any model graph.
- $Z_i = \max(|V(I')|)\forall I' \in \mathcal{S}_I$, the maximum size of the subgraphs in $\mathcal{S}_I$.
- $Z_m = \max(|V(D')|)\forall D' \in \mathcal{S}_\mathcal{M}$, the maximum size of the subgraphs in $\mathcal{S}_\mathcal{M}$.
- P, the number of processors used.

Function *MakeSubs*, shown in Algorithm 12, bins all the edges in I in $\mathcal{O}(|E(I)|)$ time. The maximum number of edges $|E(I)| = \mathcal{O}(|V(I)|^2)$, so *MakeSubs* is $\mathcal{O}(|V(I)|^2)$.

For input subgraph I' with $\alpha_{I'}(v_C(I')) = j$, and search network η_j, Function *Search-Networks*, shown in Algorithm 20, first searches each root vertex of η_j for each satellite vertex $\mu \in V(I')$, in time

$$\mathcal{O}(|R(\eta_j)| \times |V(I')|).$$

In the worst case $|V(I')| = Z_i$, so this stage takes time

$$\mathcal{O}(|R(\eta_j)| \times Z_i).$$

Next, eligible vertices are visited. A vertex is visited in constant time. The search network is checked iteratively until no eligible vertices are found during an entire pass. This process takes time

$$\mathcal{O}((|V(\eta_j)| - |R(\eta_j)|)^2).$$

Finally, each visited vertex μ adds the set $\mathcal{D}'(\mu)$ of model subgraphs it represents to the set $\mathcal{P}$ of pairs of subgraph isomorphic subgraphs. In the worst case $|\mathcal{S}_\mathcal{M}|$ model subgraphs will be added in $\mathcal{O}(|\mathcal{S}_\mathcal{M}|)$ time.

For each $I' \in \mathcal{S}_I$ the search is purely sequential. Therefore, for $|\mathcal{S}_I| \leq P$, the total complexity of Function *SearchNetworks* is

$$\mathcal{O}((|R(\eta_j)| \times |V(I')|) + (|V(\eta_j)| - |R(\eta_j)|)^2 + |\mathcal{S}_\mathcal{M}|).$$

For $|\mathcal{S}_I| > P$, the total complexity of Function *SearchNetworks* is

$$\mathcal{O}((|\mathcal{S}_I|/P)((|R(\eta_j)| \times |V(I')|) + (|V(\eta_j)| - |R(\eta_j)|)^2 + |\mathcal{S}_\mathcal{M}|)).$$

For a model graph $D \in \mathcal{M}$, the set $\mathcal{S}_\mathcal{M}$ of model subgraphs, and the set $\mathcal{P}$ of pairs of subgraph isomorphic subgraphs, Procedure *Mappings*, shown in Algorithm 21, first checks that each subgraph $D' \subseteq D$ occurs at least once in $\mathcal{P}$. Since the maximum number of subgraphs of a model graph is the number of vertices in the model, this takes $\mathcal{O}(|V(D)|)$ time.

The second stage determines, for each pair of subgraphs $(D', I') \in \mathcal{P}$ with $D' \subseteq D$, all possible mappings from D' to I'. Determining all mappings for a single pair of subgraphs takes time $\mathcal{O}(|V(I')|^{|V(D')|})$. If all subgraphs of all models are to be mapped to all subgraphs of the input graph,using P processors, it will take time

$$\mathcal{O}(|\mathcal{S}_\mathcal{M}| \times |\mathcal{S}_I| \times Z_i^{Z_m}/P).$$

In the worst case $|A| = 1$, all model graphs are of the same size, and both the model and input graphs are fully-connected. Under these circumstances,

$$Z_m = \mathcal{O}(|V(D)|),$$

$$V_m = \mathcal{O}(|V(D)|),$$

$$|\mathcal{S}_\mathcal{M}| = \mathcal{O}(|\mathcal{M}| \times |V(D)|),$$

$$Z_i = \mathcal{O}(|V(I)|)$$

and

$$\mathcal{S}_I = \mathcal{O}(|V(I)|).$$

The final worst case time complexity for the second stage is therefore

$$\mathcal{O}(|\mathcal{M}| \times |V(D)| \times |V(I)| \times |V(I)|^{|V(D)|}/P).$$

The third stage performs a tree search of the mappings of each subgraph of a model, searching for complete, consistent mappings from the model graph to the input graph. The complexity of tree search is given by the number of tree-nodes to expand at each level, to the power of the number of levels. The worst case conditions for stage two are also the worst case conditions for this stage. The number of levels in the search is the number of subgraphs in the model, worst case $|V(D)|$. The number of tree-nodes at a level is the number of mappings found for each subgraph in the model. Under worst case conditions the number of mappings from a single model subgraph to a single input subgraph is

$$\mathcal{O}(\frac{(|V(I)|)!}{(|V(I)| - |V(D)|)!}),$$

and every model subgraph maps onto every input subgraph where $|V(I')| \geq |V(D')|$. The number of tree-nodes at a level is therefore

$$\mathcal{O}(\frac{|V(I)|! \times |V(I)|}{(|V(I)| - |V(D)|)!}),$$

and the worst case number of tree-node expansions is

$$\mathcal{O}((\frac{|V(I)|! \times |V(I)|}{(|V(I)| - |V(D)|)!})^{|V(D)|}).$$

Finally, the search must be performed for all models using P processors, so the final

worst case complexity is

$$\mathcal{O}(\frac{|\mathcal{M}|}{P}((\frac{|V(I)|! \times |V(I)|}{(|V(I)| - |V(D)|)!})^{|V(D)|})).$$

Now looking at the complexity of the entire procedure, it can be seen that the third stage dominates the first two, and the worst case complexity for the procedure is

$$\mathcal{O}(\frac{|\mathcal{M}|}{P}((\frac{|V(I)|! \times |V(I)|}{(|V(I)| - |V(D)|)!})^{|V(D)|})).$$

The worst case occurs when all graphs have only one attribute, all graphs are fully connected, and all graphs are the same size.

The worst case complexity for the entire PN algorithm is determined by procedure *Mappings*. The worst case for the serial Ullmann algorithm is

$$\mathcal{O}(|V(I)|^{|V(D)|} \times |V(D)|^2 \times |\mathcal{M})$$

and the worst case for the algorithm of [Messmer and Bunke(1993)] is also

$$\mathcal{O}(|V(I)|^{|V(D)|} \times |V(D)|^2 \times |\mathcal{M})$$

. Thus the worst case performance of the PN algorithm is inferior to existing methods. In particular, the algorithm can be expected to perform poorly when all graphs have only one or two attributes, and are densely or fully connected. For non-extremal cases, however, the complexity appears comparable, while offering better parallelisability than the Messmer and Bunke algorithm and more efficient processing of multiple models than the Ullmann algorithm.

6.5 Conclusion

This chapter demonstrated that the HiC and $FatHiC$ interconnection networks are well suited to executing irregular algorithms which rely on dynamic load balancing strategies to achieve efficiency. Their suitability is due to their excellent local connectivity and hierarchical structure, which allows the HS dynamic load balancing algorithm to search for work on a "nearest first" basis.

The hierarchical search HS algorithm, a new, neighbourhood based, receiver initiated, dynamic work distribution algorithm for $HiC_{(k,h)}$ was described. The scalability of the HS algorithm was analysed, and compared favourably with existing dynamic load balancing algorithms for hypercube. As an example of an application algorithm which benefits from the ability of the HS algorithm to exploit local connectivity to improve performance, a novel application algorithm was introduced, the Parallel Network PN algorithm. The PN algorithm is a parallel, deterministic algorithm for finding subgraph isomorphisms from a database of attributed, directed model graphs to an attributed, directed input graph. Methods of exploiting the parallelism of the PN algorithm were described, highlighting the need for effective local distribution of work to reduce communication overheads and improve performance.

Chapter 7

Conclusion

This thesis proposed two new interconnection networks, the Hierarchical Clique (HiC) network, and a derivative, the Fat Hierarchical Clique ($FatHiC$) network. The major goal of this thesis was to establish the suitability of these networks for use in parallel computing. A minor goal of this thesis was to establish the suitability of the HiC and $FatHiC$ interconnection networks for parallel computers solving irregular problems.

The HiC interconnection network is a hierarchical network which combines the dense connectivity of a fully connected network, or clique, at the local level with the scalability of a tree for global communication. The good local connectivity allows improved performance and fault-tolerance, while the underlying tree ensures that the network has constant degree and reasonable cost. For some algorithms, high demand for communication over long distances can cause congestion in the higher levels of the tree, leading to congestion and delays in communication. The $FatHiC$ is a derivative of the HiC, which uses the principles of the Fat Tree, introduced by [Leiserson(1985)]. The $FatHiC$ uses higher bandwidth links in the upper levels of the tree to reduce or eliminate congestion, allowing significant increases in performance for algorithms which require extensive long distance communication within the parallel computer. The HiC

is an attractive, cost effective network if the algorithms to be executed on the parallel computer feature large amounts of local communication, with moderate requirements for long distance communication. The $FatHiC$ becomes cost effective if there is a requirement for large amounts of long distance communication. Both networks are well suited to the execution of irregular algorithms which require dynamic load balancing. The HS algorithm is a simple, deterministic dynamic load balancing algorithm which exploits the dense local connectivity and hierarchical structure of the HiC or $FatHiC$ networks, allowing idle processors to target processors for work requests on a 'nearest first' basis.

The topology and addressing scheme of the HiC and $FatHiC$ interconnection networks were described, as were simple, efficient message passing algorithms. These three components together allow a functional parallel computer to be based upon either interconnection network. In order to establish that the new networks are not merely functional, but also useful, it remained to establish that they offered high performance, acceptable fault-tolerance, good embedding properties and cost effectiveness.

Fundamental parameters of the networks, commonly used as indicators of performance, were derived. Diameter is slightly superior to other hierarchical networks, such as k-ary tree and GFT. The average inter-PE distance was shown to decrease with increasing values of k, at the cost of higher node degree and more links in the network. A number of standard network parameters commonly used as indicators of network fault-tolerance were derived, including degree δ, connectivity κ and link connectivity λ, as was the probability of network disconnection caused by a set of κ nodes or λ links. The fault diameter was derived; the $HiC_{(k,h)}$ was shown to be strongly resilient, with a fault diameter similar to that of the hypercube for a similar number of PEs. Lower bounds for the two terminal reliability and average two terminal reliability were calculated. The lower bound on the hypercube's average two terminal reliability was found to be higher than that of the HiC for large numbers of PEs, as a result of its much higher, and

increasing, degree. The HiC has fixed degree for greater scalability and lower cost. Simple, efficient routing algorithms were presented, and a general scheme for fault-tolerant routing was outlined. Network embeddings for the binary tree, hypercube and $2D$-mesh into the $HiC_{(4,h)}$ were described. While expansion and dilation of the embeddings was low, the congestion was high. The $FatHiC_{(k,h)}$ was shown to embed the three networks with low congestion. Results were compared with embeddings into a GFT, and the $FatHiC$ was shown to give superior embeddings.

Simulations were used to demonstrate that the proposed routing algorithms are stable up to, and beyond, saturation. Latency is very low while the network is unsaturated and throughput will continue to rise with increased uniform load even after the top layers saturate. Traffic patterns which exhibit locality of reference yield greatly improved throughput and latency. The performance of Floyd's all pairs shortest path algorithm on the HiC was studied and results were compared with results for the popular $2D$-mesh. It was shown that the HiC gives very similar performance to the mesh for this type of regular problem. Another regular problem, the *ascend/descend* class of algorithms, showed the susceptibility of the HiCs tree based structure to congestion near the root. The $FatHiC$ can be configured to eliminate the problem of congestion for *ascend/descend* class algorithms. The $FatHiC$ offered better performance than the HiC and $2D$-mesh networks for the *ascend/descend* class of algorithms. Its increased performance, however, must be balanced against the extra cost and complexity incurred in realising the extra bandwidth. In general then, the HiC and $FatHiC$ are stable platforms which can be used to solve some important regular problems as efficiently as currently accepted networks.

A novel modification of cost effectiveness factor, introduced by [Sarkar(1993)], allowed the cost effectiveness of the HiC and $FatHiC$ to be compared with that of the $2D$-mesh and GFT for two important regular problems. For Floyd's all pairs shortest path algorithm the HiC gave a cost effectiveness which was stable with increasing numbers

of processors and not greatly inferior to that of the $2D$-mesh. The $FatHiC$ was less cost effective than the HiC for Floyd's algorithm, the increased efficiency offered by improved communication was not sufficient to outweigh the extra cost of the fat links for this application. Similarly the increased connectivity of the GFT was not of sufficient use in implementing Floyd's algorithm to justify its cost. For the *ascend/descend* class of algorithms the cost effectiveness of the HiC was decreased by congestion in the interconnection network. The $FatHiC$'s gain in efficiency by eliminating the delays caused by congestion exceeded the increase in cost caused by the extra bandwidth of high level links. The $FatHiC$ was the most cost effective network considered for the *ascend/descend* class of algorithms. In general the HiC interconnection network allows cost effective solutions of some important regular problems. The HiC is competitive with the popular $2D$-mesh for such problems. It is also competitive with the GFT. Congestion in the network may reduce efficiency for some classes of problem. In such cases the $FatHiC$ may prove to be a cost effective alternative.

The HiC and $FatHiC$ interconnection networks were shown to be well suited to executing irregular algorithms which rely on dynamic load balancing strategies to achieve efficiency. Their suitability is due to their excellent local connectivity and hierarchical structure, which allows the HS algorithm to search for work on a 'nearest first' basis. The hierarchical search (HS) algorithm, a new, neighbourhood based, receiver initiated, dynamic load balancing algorithm for $HiC_{(k,h)}$ was described. The scalability of the HS algorithm was analysed, and shown to compare favourably with existing dynamic load balancing algorithms for hypercube. As an example of an application algorithm which benefits from the ability of the HS algorithm to exploit local connectivity to improve performance, a novel application algorithm was introduced, the Parallel Network (PN) algorithm. The PN algorithm is a parallel, deterministic algorithm for finding subgraph isomorphisms from a database of attributed, directed model graphs to an attributed, directed input graph. Methods of exploiting the parallelism of the PN algorithm were described, highlighting the need for effective local distribution of work to

reduce communication overheads and improve performance.

The *HiC* and *FatHiC* interconnection networks are suitable for use in parallel computing because they offer scalability and cost effective performance on regular algorithms. Their fault-tolerance and embedding properties are comparable with other tree based, hierarchical networks which have been proposed, such as the *GFT* [Öhring *et al.*(1995)]. In addition, the *HiC* and *FatHiC* networks are well suited to the execution of irregular algorithms which require dynamic load balancing, as their hierarchical structure allows the HS algorithm to provide an efficient means of distributing load.

Future work could involve investigating the suitability of the *HiC* and *FatHiC* topologies for interconnecting clusters of workstations into virtual supercomputers. Clusters are currently becoming increasingly available, effective methods of interconnecting them could allow greater power and more effective use of their resources. Further development of algorithms which efficiently utilise the *HiC* and *FatHiC* communication network topologys are another area of possible future work. General load balancing algorithms are important to allow the efficient usage of parallel systems without detailed knowledge of system details by every programmer. At the same time, parallel computing is about efficiency, and maximum efficiency comes from a tuned system of machine and algorithm. Determining how to tune algorithms for the *HiC* and *FatHiC* topologies may be a productive area of research.

Publications from this Thesis

Campbell, S. and Kumar, M. (1996). The Design and Development of the COMPS Architecture. In *Proceedings of the Australasian Computer Architecture Workshop, Monash University, Melbourne, Australia*, pages 177-187.

Campbell, S. and Kumar, M. (1996). Hierarchical Cliques: Cost Effective Multiprocessor Interconnection Network. In *IASTED International Conference on Parallel and Distributed Computing and Systems October 16-19, Chicago, Illinois - USA*, pages 134-138.

Campbell, S. and Kumar, M. (1997). COMPS: The Common Memory Message Passing System. In *Computer Architecture '96: Selected Papers of the First Australasian Conference*, pages 215-225, editor R. Pose, Springer-Verlag Singapore,

Campbell, S. and Kumar, M. (1998). A Novel Parallel Algorithm for Finding Subgraph Isomorphisms. In *Proceedings Ninth Australasian Workshop on Combinatorial Algorithms*, pages 40-51, School of Computing, Curtin University of Technology, Perth WA.

To Appear:

Campbell, S. and Kumar, M. (1999). Parallel Subgraph Matching on a Hierarchical Interconnection Network. In *Hardware for Intelligent Technologies*, editor Horia-Nicolai Teodorescu, CRC Press.

Glossary of Symbols

Bibliography

[Alon *et al.*(1995)] Alon, N., Yuster, R., and Zwick, U. (1995). Color-coding. *Journal of the Association of Computing Machinery*, **42**(4), 844–856.

[Andrews and Polychronopoulos(1991)] Andrews, J. and Polychronopoulos, C. (1991). An analytical approach to performance/cost modeling of parallel computers. *Journal of Parallel and Distributed Computing*, **12**, 343–356.

[Artymiuk *et al.*(1994)] Artymiuk, P., Grindley, H., Poirrette, A., Rice, D., Ujah, E., and Willett, P. (1994). Identification of beta-sheet motifs, of psi-loops and of patterns of acid residues in three dimensional protein structures using a subgraph isomorphism algorithm. *Journal of Chemical Information and Computer Sciences*, **34**, 54–62.

[Avizienis(1971)] Avizienis, A. (1971). Fault tolerant computing - an overview. *IEEE Computer*, **4**, 5–8.

[Ball(1980)] Ball, M. (1980). Complexity of network reliability computations. *Networks*, **10**, 153–165.

[Benstock *et al.*(1988)] Benstock, J., Berndt, D., and Agarwal, K. (1988). Graph embedding in synchem2, an expert system for organic synthesis discovery. *Discrete Applied Mathematics*, **19**, 45–63.

[Bhatt *et al.*(1992)] Bhatt, S., Chung, F., Leighton, F., and Rosenberg, A. (1992). Efficient embeddings of trees in hypercubes. *SIAM Journal of Computing*, **21**(1), 151–162.

[Boesch(1972)] Boesch, F. (1972). Lower bounds on the vulnerability of a graph. *Networks*, **2**, 329–340.

[Boesch and Felzer(1972)] Boesch, F. and Felzer, A. (1972). A general class of invulnerable graphs. *Networks*, **2**, 261–283.

[Bondy and Murty(1976)] Bondy, J. and Murty, U. (1976). *Graph Theory with Applications*. The MacMillan Press.

[Brecht and Colbourn(1986)] Brecht, T. and Colbourn, C. (1986). Improving reliability bounds in computer networks. *Networks*, **16**, 369–380.

[Brezany *et al.*(1994)] Brezany, P., Chapman, B., Ponnusamy, R., Sipkova, V., and Zima, H. (1994). Study of application algorithms with irregular distributions. Technical Report D1Z-3, Institute for Software Engineering and Parallel Systems, University of Vienna.

[Brown *et al.*(1990)] Brown, J., Colbourn, C., and Devitt, J. (1990). Network transformations and bounding network reliability. Technical report, School of Mathematics and Statistics, Curtin University of Technology, Western Australia.

[Buckley and Harary(1990)] Buckley, F. and Harary, F. (1990). *Distance in Graphs*. Addison-Wesley Publishing Company, first edition.

[Bunke and Allermann(1983)] Bunke, H. and Allermann, G. (1983). Inexact graph matching for structural pattern recognition. *Pattern Recogntion Letters*, **1**, 245–253.

[Campbell and Kumar(1996)] Campbell, S. and Kumar, M. (1996). Hierarchical cliques: Cost effective multiprocessor interconnection network. In *IASTED International Conference on Parallel and Distributed Computing and Systems October 16-19, Chicago, Illinois - USA*, pages 134–138. IASTED.

[Campbell and Kumar(1998)] Campbell, S. and Kumar, M. (1998). A novel parallel algorithm for finding subgraph isomorphisms. In *Proceedings Ninth Australasian*

Workshop on Combinatorial Algorithms, pages 40–51. School of Computing, Curtin University of Technology, Perth WA.

[Chan(1991)] Chan, M. (1991). Embedding of grids into optimal hypercubes. *SIAM Journal of Computing*, **20**(5), 297–299.

[Chen(1996)] Chen, L. (1996). Graph isomorphism and identification matrices: Parallel algorithms. *IEEE Transactions on Parallel and Distributed Systems*, **7**(3), 308–319.

[Chung *et al.*(1998)] Chung, Y., Wang, C.-L., and Prasanna, V. (1998). Parallel algorithms for perceptual grouping on distributed memory machines. *Journal of Parallel and Distributed Computing*, **50**, 123–143.

[Colbourn(1987)] Colbourn, C. (1987). *The Combinatorics of Network Reliability*. New York, Oxford University Press.

[Costanzo *et al.*(1986)] Costanzo, J., Crowl, L., Sanchis, L., and Srinivas, M. (1986). Subgraph isomorphism on the bbn butterfly multiprocessor. Technical Report 14, Dept. of Computer Science, University of Rochester.

[Crowl *et al.*(1994)] Crowl, L., Crovella, M., LeBlanc, T., and M.L.Scott (1994). The advantages of multiple parallelizations in combinatorial search. *Journal of Parallel and Distributed Computing*, **21**, 110–123.

[Dally(1990a)] Dally, W. (1990a). Network and processor architecture for message-driven computers. In R. Suaya and G. Birtwistle, editors, *VLSI and Parallel Computation*. Morgan Kaufmann Publishers Inc.

[Dally and Seitz(1986)] Dally, W. and Seitz, C. (1986). The torus routing chip. *Distributed Computing*, **1**, 187–196.

[Dally and Seitz(1987)] Dally, W. and Seitz, C. (1987). Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, **C-36**(5), 547–553.

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

[Dandamudi and Eager(1990)] Dandamudi, S. P. and Eager, D. L. (1990). Hierarchical interconnection networks for multicomputer systems. *IEEE Transactions on Computers*, **C-39**(6), 786–797.

[DeBenedictis(1982)] DeBenedictis, E. (1982). A communications operating system for the homogenous machine. Technical Report 4707, Computer Science Department, California Institute of Technology, Pasdena.

[Dutt and Hayes(1990)] Dutt, S. and Hayes, J. (1990). On designing and reconfiguring k-fault-tolerant tree architectures. *IEEE Transactions on Computers*, **39**(4), 490–503.

[Efe(1992)] Efe, K. (1992). The crossed cube architecture for parallel computation. *IEEE Transactions on Parallel and Distributed Systems*, **3**(5), 513–524.

[El-Amawy and Latifi(1991)] El-Amawy, A. and Latifi, S. (1991). Properties and performance of folded hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, **2**(1), 31–42.

[Esfahanian(1989)] Esfahanian, A.-H. (1989). Generalized measures of fault tolerance with application to n-cube networks. *IEEE Transactions on Computers*, **38**(11), 1586–1591.

[Esfahanian *et al.*(1991)] Esfahanian, A.-H., Ni, L., and Sagan, B. (1991). The twisted n-cube with application to multiprocessing. *IEEE Transactions on Computers*, **40**(1), 88–93.

[Fu and Yang(1997)] Fu, C. and Yang, T. (1997). Run-time techniques for exploiting irregular task parallelism on distributed memory architectures. *Journal of Parallel and Distributed Computing*, **42**, 143–156.

[Galil *et al.*(1987)] Galil, Z., Hoffmann, C., Luks, E., Schnorr, C., and Weber, A. (1987). An $o(n^3 \log n)$ deterministic and an $o(n^3)$ las vegas isomorphism test for trivalent graphs. *Journal of the Association of Computing Machinery*, **34**(3), 513–531.

[Garey and Johnson(1979)] Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company.

[Gupta and Kumar(1993)] Gupta, A. and Kumar, V. (1993). Performance properties of large scale parallel systems. *Journal of Parallel and Distributed Computing.*

[Hockney and Jesshope(1988)] Hockney, R. and Jesshope, C. (1988). *Parallel Computers 2: Architecture, Programming and Algorithms. 2nd edition.* Adam, Hilger, Bristol and Philadelphia.

[Hsu(1995)] Hsu, W.-L. (1995). $o(m.n)$ algorithms for the recognition and isomorphism problems on circular-arc graphs. *SIAM Journal of Computing*, **24**(3), 411–439.

[Hwang(1993)] Hwang, K. (1993). *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw-Hill. Inc., first edition.

[Hwang and Ghosh(1987)] Hwang, K. and Ghosh, J. (1987). Hypernet: A communication-efficient architecture for constructing massively parallel computers. *IEEE Transactions on Computers*, **C-36**(12), 1450–1466.

[Jaja and Kosaraju(1988)] Jaja, J. and Kosaraju, S. (1988). Parallel algorithms for planar graph isomorphism and related problems. *IEEE Transactions on Circuits and Systems*, **35**(3), 304–310.

[Kermani and Kleinrock(1979)] Kermani, P. and Kleinrock, L. (1979). Virtual cut-through: A new computer communication switching technique. *Computer Networks*, **3**, 267–286.

[Krishnamoorthy and Krishnamurthy(1987)] Krishnamoorthy, M. and Krishnamurthy, B. (1987). Fault diameter of interconnection networks. *Computing and Mathematical Applications*, **13**(5/6), 577–582.

[Kumar and Patnaik(1992)] Kumar, J. and Patnaik, L. (1992). Extended hypercube: A hierarchical interconnection network of hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, **3**(1), 45–57.

[Kumar and Rao(1987)] Kumar, V. and Rao, V. (1987). Parallel depth first search, part ii: Analysis. *International Journal of Parallel Programming*, **16**(6), 501–519.

[Kumar and Singh(1991)] Kumar, V. and Singh, V. (1991). Scalability of parallel algorithms for the all-pairs shortest-path problem. *Journal of Parallel and Distributed Computing*, **13**, 124–138.

[Kumar *et al.*(1994)] Kumar, V., Grama, A., and Rao, V. (1994). Sclable load balancing techniques for parallel computers. *Journal of Parallel and Distributed Computing*, **22**(1), 60–79.

[Lai and White(1990)] Lai, T.-H. and White, W. (1990). Mapping pyramid algorithms into hypercubes. *Journal of Parallel and Distributed Computing*, **9**, 42–54.

[Leiserson(1985)] Leiserson, C. (1985). Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, **C-34**(10), 892–900.

[Lingas and Syslo(1988)] Lingas, A. and Syslo, M. (1988). A polynomial time algorithm for subgraph isomorphism of two-connected series-parallel graphs. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming; Lecture Notes in Computer Science*, volume 317, pages 394–409. Springer.

[Lomonosov and Polesskii(1971)] Lomonosov, M. and Polesskii, V. (1971). An upper bound for the reliability of information networks. *Problems of Information Transmission*, **7**, 337–339.

[Malluhi and Bayoumi(1994)] Malluhi, Q. and Bayoumi, M. (1994). The hierarchical hypercube: A new interconnection topology for massively parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, **5**(1), 17–30.

[Matula(1978)] Matula, D. (1978). Subtree isomorphism in $o(n^{5/2})$. *Annals of Discrete Mathematics*, **2**, 91–106.

[Messmer and Bunke(1993)] Messmer, B. and Bunke, H. (1993). A network based approach to exact and inexact graph matching. Technical Report IAM-93-021, University of Bern, Institute for informatics and applied mathematics.

[Monien and Sudborough(1990)] Monien, B. and Sudborough, H. (1990). Embedding one interconnection network in another. In G. Tinhofer, E. Mayr, H. Noltemeier, and M. Syslo, editors, *Computational Graph Theory*. Springer-Verlag.

[Moreira *et al.*(1998)] Moreira, J., Naik, V., and Midkiff, S. (1998). Dynamic data distribution and processor repartitioning for irregularly structured computations. *Journal of Parallel and Distributed Computing*, **50**, 28–60.

[Öhring *et al.*(1995)] Öhring, S., Ibel, M., Das, S., and Kumar, M. (1995). On generalized fat trees. In *9th International Parallel Processing Symposium*, pages 37–44.

[Pease(1977)] Pease, M. (1977). The indirect binary n-cube microprocessor array. *IEEE Transactions on Computers*, **C-26**, 458–473.

[Petrini and Vanneschi(1997)] Petrini, F. and Vanneschi, M. (1997). k-ary n-trees: High performance networks for massively parallel architectures. In *11th International Parallel Processing Symposium*, volume 1, pages 87–93. IEEE Computer Society Press.

[Preparata and Vuillemin(1981)] Preparata, F. and Vuillemin, J. (1981). The cube-connected cycles: A versatile network for parallel computation. *Communications of the ACM*, **24**(5), 300–309.

[Provan and Ball(1983)] Provan, J. and Ball, M. (1983). The complexity of counting cuts and computing the probability that a graph is connected. *SIAM Journal of Computing*, **12**, 777–788.

[Rao and Kumar(1987)] Rao, V. and Kumar, V. (1987). Parallel depth first search, part i: Implementation. *International Journal of Parallel Programming*, **16**(6), 479–499.

[Reddaway(1973)] Reddaway, S. (1973). Dap - a distributed array processor. In *1st Annual Symposium on Computer Architecture*. IEEE/ACM.

[Saad and Schultz(1988)] Saad, Y. and Schultz, M. (1988). Topological properties of hypercubes. *IEEE Transactions on Computers*, **37**(7), 867–872.

[Sarkar(1993)] Sarkar, D. (1993). Cost and time-cost effectiveness of multiprocessing. *IEEE Transactions on Parallel and Distributed Systems*, **4**(6), 704–712.

[Seitz(1985)] Seitz, C. (1985). The cosmic cube. *Communications of the ACM*, **28**(1), 22–33.

[Seitz(1990)] Seitz, C. (1990). Concurrent architectures. In R. Suaya and G. Birtwistle, editors, *VLSI and Parallel Computation*. Morgan Kaufmann Publishers Inc.

[Slyke and Frank(1972)] Slyke, R. V. and Frank, H. (1972). Network reliability analysis: Part 1. *Networks*, **1**, 279–290.

[Squire and Palais(1963)] Squire, J. and Palais, S. (1963). Programming and design considerations of a highly parallel computer. In *Proc. AFIP Spring Joint Computer Conference*, volume 23, pages 395–400.

[Stone(1987)] Stone, H. (1987). *High-Performance Computer Architecture*. Addison-Wesley Publishing Company.

[Stout(1986)] Stout, Q. (1986). Hypercubes and pyramids. In V. Cantoni and S.Levialdi, editors, *NATO ASI Series. Vol. 25*. Springer-Verlag Berlin Heidelberg.

[Sullivan and Bashkow(1977)] Sullivan, H. and Bashkow, T. (1977). A large scale homogenous, fully distributed parallel machine. In *Proc. Fourth Symposium on Computer Architecture*, pages 105–117.

[Tien and Raghavendra(1993)] Tien, S.-B. and Raghavendra, C. (1993). Algorithms and bounds for shortest paths and diameter in faulty hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, 4(6), 713–718.

[Tzeng and Wei(1991)] Tzeng, N.-F. and Wei, S. (1991). Enhanced hypercubes. *IEEE Transactions on Computers*, **40**(3), 284–293.

[Ullmann(1976)] Ullmann, J. (1976). An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*, **23**(1), 31–42.

[Unger(1958)] Unger, S. (1958). A computer oriented towards spatial problems. *Proceedings of the Institute of Radio Engineers*, **46**, 1744–50.

[Walker(1985)] Walker, P. (1985). The transputer. *Byte Magazine*, pages 219–235.

[Wong(1992)] Wong, E. (1992). Model matching in robot vision by subgraph isomorphism. *Pattern Recognition*, **25**(3), 287–303.