Spring 5-31-1999

A new-generation class of parallel architectures and their performance evaluation

Qian Wang

New Jersey Institute of Technology

ABSTRACT

A NEW-GENERATION CLASS OF PARALLEL ARCHITECTURES AND THEIR PERFORMANCE EVALUATION

by
Qian Wang

The development of computers with hundreds or thousands of processors and capability for very high performance is absolutely essential for many computation problems, such as weather modeling, fluid dynamics, and aerodynamics. Several interconnection networks have been proposed for parallel computers. Nevertheless, the majority of them are plagued by rather poor topological properties that result in large memory latencies for DSM (Distributed Shared-Memory) computers. On the other hand, scalable networks with very good topological properties are often impossible to build because of their prohibitively high VLSI (e.g., wiring) complexity. Such a network is the generalized hypercube (GH). The GH supports full-connectivity of its nodes in each dimension and is characterized by outstanding topological properties. In addition, low-dimensional GHs have very large bisection widths. We propose in this dissertation a new class of processor interconnections, namely HOWs (Highly Overlapping Windows), that are more generic than the GH, are highly scalable, and have comparable performance. We analyze the communications capabilities of 2-D HOW systems and demonstrate that in practical cases HOW systems perform much better than binary hypercubes for important communications patterns. These properties are in addition to the good scalability and low hardware complexity of HOW systems. We present algorithms for one-to-one, one-to-all broadcasting, all-to-all broadcasting, one-to-all personalized, and all-to-all personalized communications on HOW systems. These algorithms are developed and evaluated for several communication models. In addition, we develop techniques for the efficient embedding of popular topologies, such as the ring, the torus, and

the hypercube, into 1-D and 2-D HOW systems. The objective is to show that 2-D HOW systems are not only scalable and easy to implement, but they also result in good embedding of several classical topologies.

A NEW-GENERATION CLASS OF PARALLEL ARCHITECTURES AND THEIR PERFORMANCE EVALUATION

by
Qian Wang

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Computer and Information Science

May 1999

APPROVAL PAGE

A NEW-GENERATION CLASS OF PARALLEL ARCHITECTURES AND THEIR PERFORMANCE EVALUATION

Qian Wang

5/17/99

Dr. Sotirios G. Ziavras, Dissertation Advisor — Date
Associate Professor of Electrical and Computer Engineering,
and Computer and Information Science, NJIT

5/17/99

Dr. David Nassimi, Committee Member — Date
Associate Professor of Computer and Information Science, NJIT

5/17/99

Dr. James McHugh, Committee Member — Date
Professor of Computer and Information Science, NJIT

5/17/99

Dr. Mengchu Zhou, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

5/17/99

Dr. Alex Gerbessiotis, Committee Member — Date
Assistant Professor of Computer and Information Science, NJIT

BIOGRAPHICAL SKETCH

Author: Qian Wang

Degree: Doctor of Philosophy

Date: May 1999

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer and Information Science,
  New Jersey Institute of Technology, Newark, NJ, 1999

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1993

- Bachelor of Science in Electrical Engineering,
  Huazhong University of Science and Technology, Wuhan, China, 1988

Major: Computer Science

Presentations and Publications:

Q. Wang and S.G. Ziavras, "Powerful and Feasible Processor Interconnections With an Evaluation of Their Communications Capabilities," *International Symposium on Parallel Architectures, Algorithms, and Networks,* Freemantle, Australia, June 23-25, 1999.

Q. Wang and S.G. Ziavras, "Network Embedding Techniques for a New Class of Feasible Parallel Architectures Capable of Very High Performance," *International Conference on Applied Informatics,* Innsbruck, Austria, February 15-18, 1999.

S.G. Ziavras and Q. Wang, "Robust Interprocessor Connections for Very-High Performance," in: *Robust Communication Networks: Interconnection and Survivability,* N. Dean, F. Hsu and R. Ravi (Eds.), American Mathematical Society, Rhode Island, 1999.

Q. Wang "Optical Flow Determination and Motion Analysis," Master's Thesis, New Jersey Institute of Technology, New Jersey, 1993.

To my husband Dong Liu

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Sotirios G. Ziavras, who not only served as my research supervisor, providing valuable and countless resources, insight and intuition, but also constantly gave me support, encouragement, and reassurance.

Special thanks are given to Dr. David Nassimi, Dr. James McHugh, Dr. Mengchu Zhou, Dr. Alex Gerbessiotis for actively participating in my committee.

Many of my fellow graduate students in the Computer and Information Science Department are deserving recognition for their support. I also wish to thank Leon Jololian, Karen Hare for their help over the years.

I thank my family members, Dong Liu, Guoqi Wang, Hui Yi, Xuanshi Wang, Ying Liang, Luzhong Wang, for their affectionate support, patience, and encouragement throughout the duration of this project.

This research was supported in part by the NSF/DARPA (also cosponsored by NASA) New Millennium Computing Point Design Grant ASC-9634775.

TABLE OF CONTENTS

Chapter Page

Chapter Page

Chapter Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

LIST OF TABLES

Table Page

Table Page

CHAPTER 1

INTRODUCTION

The demand for ever greater performance by many computation problems has been the driving force for the development of computers with thousands of processors. Two important aspects are expected to dominate the massively-parallel processing field. High-level parallel languages supporting a shared address space (for DSM computers) and point-to-point interconnection networks with workstation-like nodes. Near PetaFLOPS (i.e., 10^{15} floating-point operations per second) and more performance is required by many applications, such as weather modeling, simulation of physical phenomena, fluid dynamics, aerodynamics, simulation of neural networks, simulation of chips, structural analysis, real-time image processing and robotics, artificial intelligence, seismology, animation, real-time processing of large databases, etc. Dongarra pointed out in 1995 that the world's top ten technical computing sites had peak capacity of only about 850 GigaFLOPS, with each site containing hundreds of computers. The goal of 1 TeraFLOPS (i.e., 10^{12} floating-point operations per second) peak performance was reached in late 1996 with the installation of an Intel supercomputer at Sandia Laboratories.

The PetaFLOPS performance objective seems to be a distant dream primarily because of the, as currently viewed, unsurmountable difficulty in developing low-complexity, high-bisection bandwidth, and low-latency networks to interconnect thousands of processors (and remote memories in DSM systems). To quote Dally, "wires are a limiting factor because of power and delay as well as density" [6]. Several interconnection networks have been proposed for the design of massively-parallel computers, including, among others, regular meshes and tori [7], enhanced meshes [17], fat trees, (direct binary) hypercubes [9], and hypercube variations [1] [11] [12]. The hypercube dominated the high-performance computing field in the 1980's because it has good topological properties and rather rich interconnectivity that permits efficient emulation of many topologies frequently employed in the development of algorithms [9] [14]. Nevertheless, these properties come at the cost of often prohibitively high VLSI (primarily wiring) complexity due to a dramatic increase in the number of communication channels with any increase in the number of PEs (processing elements). Its high VLSI complexity is undoubtedly its dominant drawback, that limits scalability [14] and does not permit the construction of powerful, massively-parallel systems. Two nodes in the m-cube or m-D hypercube with 2^m nodes are neighbors if and only if their unique m-bit addresses differ in a single bit. The versatility of the hypercube in emulating efficiently other important topologies constitutes an incentive for the introduction of hypercube-like interconnection networks of lower complexity that, nevertheless, preserve to a large extent the former topological properties [1] [12]. Indirect implementations of hypercubes have also been proposed [8].

To support scalability, current approaches to massively-parallel processing use bounded-degree networks, such as meshes or k-ary n-cubes (i.e., tori), with low node degree (e.g., the FLASH, Cray Research MPP, Intel Paragon, and Tera computers). However, low-degree networks result in large diameter, large average internode distance, and small bisection width. Relevant approaches that employ reconfiguration to enhance the capabilities of the basic mesh architecture (e.g., reconfigurable mesh, mesh with multiple broadcasting, and mesh with separable broadcast buses) will not become feasible for massively-parallel processing in the foreseeable future because of the requirements for long clock cycles and precharged switches to facilitate the transmission of messages over long distances [17].

The high VLSI complexity problem is unbearable for generalized hypercubes (GHs). Contrary to nearest-neighbor k-ary n-cubes that form rings with k nodes in each dimension, GHs implement fully-connected systems with k nodes in each

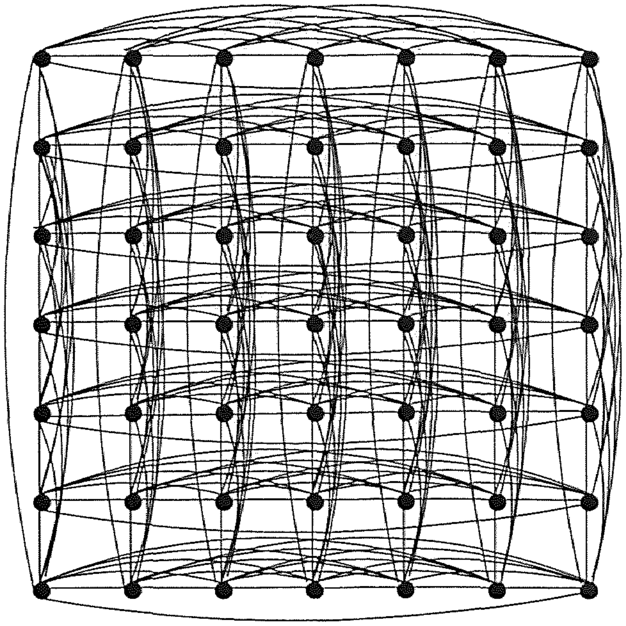


Figure 1.1 The 2-D generalized hypercube GH(7,2).

dimension [16]. The n-D (symmetric) generalized hypercube $GH(k, n)$ contains k^n nodes. The address of a node is $x_{n-1}x_{n-2}...x_1x_0$, where x_i is a radix-k digit with $0 \leq x_i \leq k-1$. This node is a neighbor to the nodes with addresses $x_{n-1}x_{n-2}...x_i'...x_1x_0$, for all $0 \leq i \leq n-1$ and $x_i' \neq x_i$. Therefore, two nodes are neighbors if and only if their n-digit addresses differ in a single digit. For the sake of simplicity, we restrict our discussion to symmetric generalized hypercubes where the nodes have the same number of neighbors in all dimensions. Therefore, each node has $k-1$ neighbors in each dimension, for a total of $n \cdot (k-1)$ neighbors per node. The n-D $GH(k, n)$ has diameter equal to only n. Figure 1.1 shows the $GH(7, 2)$ with 2 dimensions (i.e., $n = 2$) and $k = 7$. For $n = 2$ and k an even number, the diameter of the GH is only 2 and its bisection width is the immense $k^3/4$. The increased VLSI/wiring cost of GHs results in outstanding performance that permits optimal emulation of hypercubes and k-ary n-cubes, and efficient implementation of complex communication patterns [4] [3].

In order to reduce the number of communication channels in systems similar to the generalized hypercube, the spanning bus hypercube uses a shared bus for the implementation of each fully-connected subsystem in a given dimension. However, shared buses result in significant performance degradation because of the overhead imposed by the protocol that determines each time ownership of the bus. Similarly, hypergraph architectures implement all possible permutations of their nodes in each dimension by employing crossbar switches [10]. Reconfigurable generalized hypercubes interconnect all nodes in each dimension dynamically via a scalable mesh of very simple, low-cost programmable switches [15]. However, all these proposed reductions in hardware complexity may not be sufficient for very high performance computing.

To summarize, low-dimensional massively-parallel computers with full connectivity for their nodes in each dimension, such as generalized hypercubes, are very desirable because of their outstanding topological properties (e.g., extremely small diameter and average internode distance, and immense bisection width), but their electronic implementation is a Herculean task because of packaging (and primarily wiring) constraints. We propose in this dissertation a new class of interprocessor connection architectures, namely HOWs (standing for architectures with Highly Overlapping Windows), which employ the generalized hypercube [3] [16] [26] [19] with outstanding topological properties (e.g., extremely small diameter and average internode distance, and immense bisection width) as the basic building block. HOWs are also obtained from generalized hypercubes by removing some of their processor interconnections in order to reduce the wiring complexity and render them viable structures for very high-performance computing. Large generalized hypercubes have outstanding topological properties; however, they are characterized by very high wiring complexity that prohibits their implementation [11] [18] [19]. In contrast, HOWs can be viable while having simultaneously topological properties comparable to those of generalized hypercubes.

This dissertation is organized as follows. Chapter 1 introduces HOWs, a new class of parallel architectures. Chapter 2 introduces cost analysis for HOWs. Chapters 3 and 4 present the embedding of various interconnection networks into 1-D and 2-D HOW systems. Chapters 5 and 6 present and analyze communication operations for 1-D and 2-D HOW systems, respectively. Chapter 7 briefly analyzes communication operations for (direct binary) hypercubes. Finally, Chapters 8 and 9 present performance comparisons involving hypercubes (binary and generalized) and 2-D HOW systems.

1.1 The Class of HOW Architectures

The definition of the generalized hypercube network, which is the building block of HOWs, is first in order. We shall show later in this section that HOWs can also be derived from generalized hypercubes by selectively removing some of their interprocessor connections. The terms node and processor are used interchangeably. The n-D (symmetric or balanced) generalized hypercube $GH(p,n)$ with p nodes per dimension contains a total of p^n nodes [16]. The address of a node is $x_{n-1}x_{n-2}\cdots x_1x_0$, where the radix-$p$ digit x_i is $0 \leq x_i \leq p-1$ for $i = 0, 1, \cdots, n-1$. Two nodes are neighbors if and only if their n-digit addresses differ in a single radix-p digit. This generalized hypercube can be obtained from the n-D mesh by replacing the linear arrays in each dimension with fully-connected systems. Therefore, each node in the $GH(p,n)$ has $n \times (p-1)$ neighbors and its diameter is equal to just n.

Low-dimensional generalized hypercubes have very impressive bisection widths. When a network is cut into two equal halves, its bisection width is the number of edges that run between these two halves; dense/heavy communications operations can benefit from a large bisection width. For $n = 2$ and p an even number, the

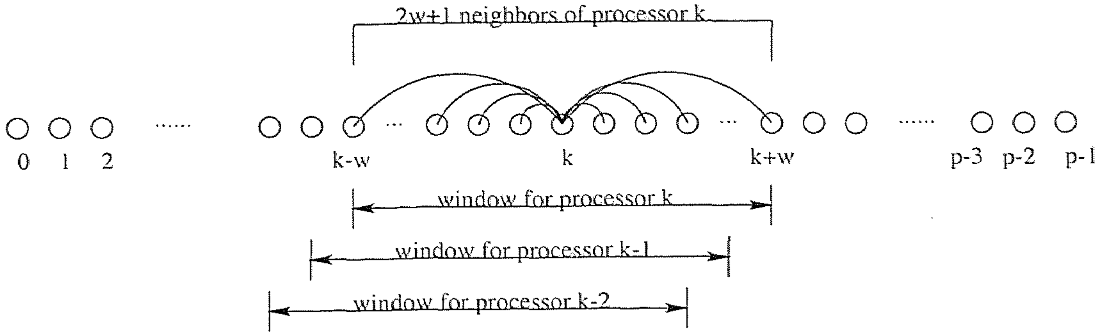


Figure 1.2 The neighbors of the node with address k in the 1-D HOW(p,w,1) system.

bisection width of the $GH(p, n)$ is the immense $p^3/4$. Also, generalized hypercubes implement efficiently very demanding communications operations, such as broadcasting and multicasting [26] [4]. Their outstanding topological properties are the result of their high node degree (that is, the large number of connections per node) which, however, has negative effect on the wiring complexity.

1.1.1 Their Structure

We first introduce the class of 1-D HOW processor interconnections [29] [30]. $HOW(p, w, 1)$ denotes a 1-D HOW system with p nodes and window size w. Each node with unique address k, where $0 \leq k \leq p-1$, is connected directly to all nodes within the windows of size w immediately to its left and right. More specifically, its neighbors have addresses $0 \leq k \pm i \leq p-1$, for all $i = 1, 2, 3, \cdots, w$. Therefore, all connections are local in this 1-D system and span up to w nodes to the left and w nodes to the right of the referenced node. Figure 1.2 shows the neighbors of a node in a 1-D HOW system.

Each processor k belongs to as many as $w + 1$ maximal-sized 1-D generalized hypercubes $GH(w + 1, 1)$ (i.e., fully-connected subsystems); they can be derived by starting with the subsystem spanning node k and all its left neighbors in the colinear representation of the $HOW(p, w, 1)$, and shifting each time the window by one position to the right until the last subsystem spans node k and all its right neighbors. Therefore, each such pair of successively-derived $GH(w+1, 1)$s have a very large overlap that forms a $GH(w, 1)$. The $HOW(p, w, 1)$ can also be derived from the $GH(p, 1)$ by removing for each node, in the colinear representation of the $GH(p, 1)$, those edges that connect it to nodes outside of the left and right windows defined by w. Therefore, existing algorithms for generalized hypercubes can be modified easily to run on HOWs because of the following reasons:

- HOWs are derived from generalized hypercubes by removing some edges.
- HOWs contain many smaller, highly-overlapping generalized hypercubes.

Not only do HOWs have reduced wiring complexity than GHs of similar size, but also the locality of processor interconnections in HOWs can be a viable solution for very high-performance computing [29] [30] [31]:

- Moore's law predicts the doubling of transistor density for chips every 18 months. Multiprocessor chips have already appeared in the market and this design concept is expected to have in the near future a very significant market share in the high performance computing field. Local intrachip processor connections, such as those required predominantly in HOWs, will then be very effective.

- Intrachip and/or local interchip connections could be implemented efficiently with current and expected electronic technologies for reasonable values of the window size w; in contrast, the global interconnections required in generalized hypercubes are much more difficult to realize. Improvements in intrachip and/or interchip interconnection technologies can increase the value of w.

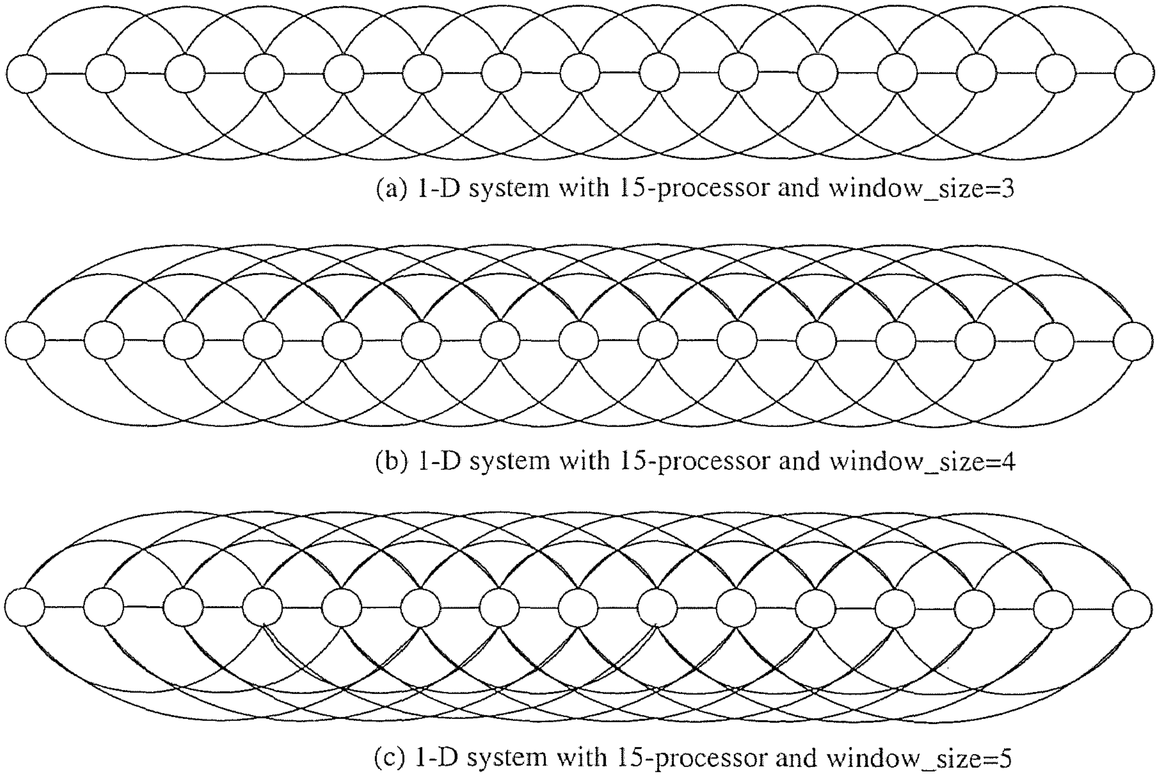
- Free-space optical interconnects are expected to become viable and commonplace in the near future for the local interconnection of chips [19]. Very substantial work is carried out in research laboratories, quite often with federal support,

> for the efficient realization of free-space interconnects within computer systems; WDM (wavelength-division multiplexing) will be employed for the transmission of multiple bits in parallel [19]. Because of the fact that chromatic dispersion becomes a major problem in WDM for distances larger than about a meter, the global interconnections required in generalized hypercubes will still be very difficult to implement. Therefore, HOWs will increase further their advantage over GHs with respect to interconnection complexity.

All of the above demonstrate that HOWs are more prone than GHs to scalability related to technological advancements.

The (symmetric) n-D $HOW(p, w, n)$ with p nodes per dimension is constructed recursively, so that each node has up to $2wn$ neighbors. A node has address $x_{n-1}x_{n-2}\cdots x_i \cdots x_1 x_0$, where x_i is a radix-p digit with $0 \leq x_i \leq p-1$ for all $i = 0, 1, \cdots, n-1$. The neighbors of this node have addresses that differ from its own address only in a single radix-p digit, that is they have addresses $x_{n-1}x_{n-2}\cdots x_i^{'} \cdots x_1 x_0$, where $1 \leq |x_i - x_i^{'}| \leq w$ for $0 \leq i \leq n-1$. This HOW system contains p^n nodes. It is important to note that such a system contains many, highly-overlapping generalized hypercubes $GH(w+1, n)$. The $HOW(p, w, n)$ can also be derived from the $GH(p, n)$ by removing in each dimension all connections for each node that do not fall into its left and right neighborhood windows defined by w. Figure 1.4 shows 2-D HOW systems containing $16, 25, 36$, and 49 processors, respectively, and having window size $w = 3$. The $HOW(4, 3, 2)$ in Figure 1.4.a is identical to the $GH(4, 2)$. In general, the $HOW(p, p-1, n)$ is identical to the $GH(p, n)$. Also, the $HOW(p, 1, n)$ is identical to the n-D mesh.

Figure 1.3 shows 1-D HOW systems containing 15 processors and having window size of 3, 4, and 5, respectively. Figure 1.4 shows the 2-D $HOW(4, 3, 2)$, $HOW(5, 3, 2)$, $HOW(6, 3, 2)$, and $HOW(7, 3, 2)$ systems containing $16, 25, 36$, and 49 processors, respectively, and having window size $w = 3$.

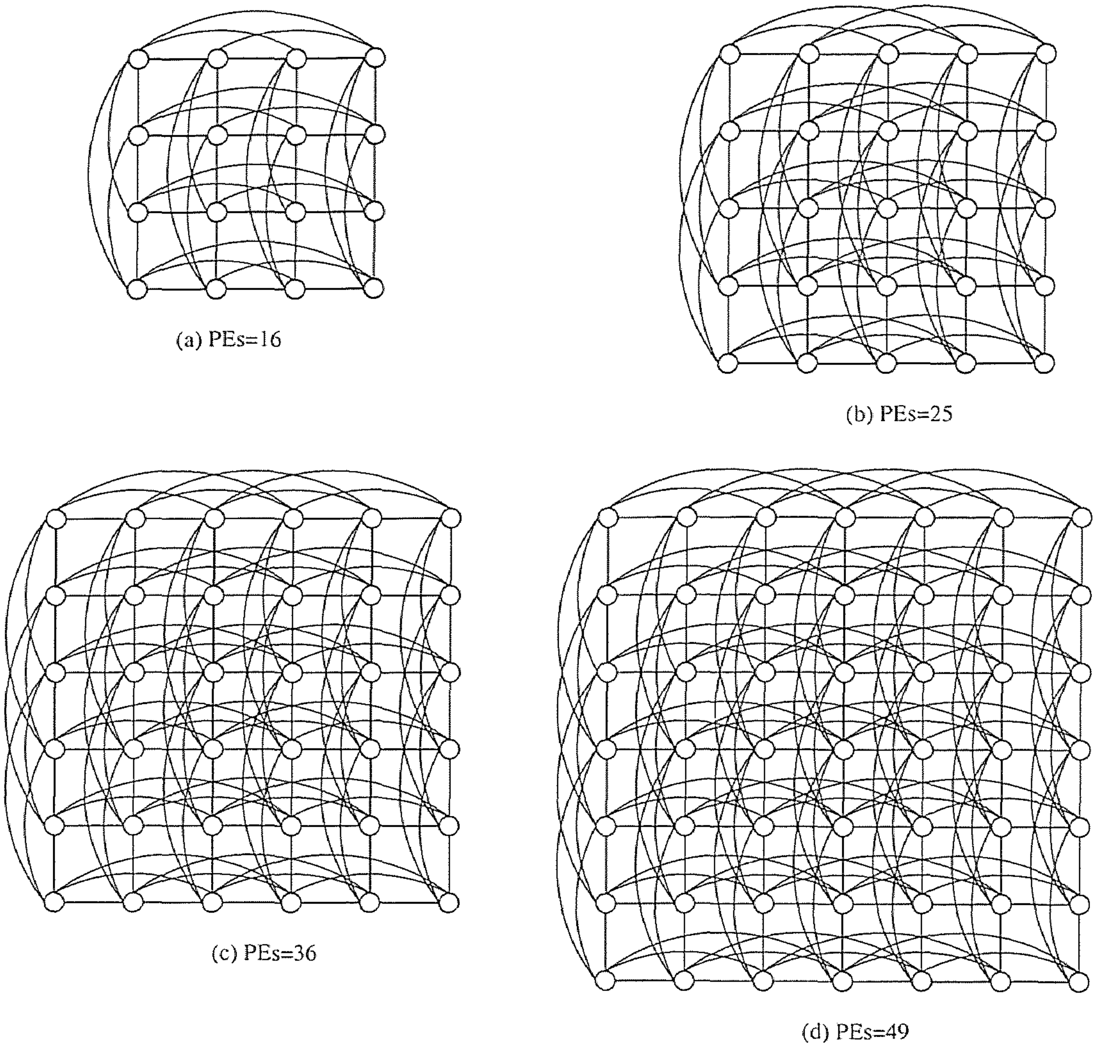


(a) 1-D system with 15-processor and window_size=3

(b) 1-D system with 15-processor and window_size=4

(c) 1-D system with 15-processor and window_size=5

Figure 1.3 1-D HOW system with 15 processors and window size of 3, 4, and 5, respectively.



(a) PEs=16

(b) PEs=25

(c) PEs=36

(d) PEs=49

Figure 1.4 Examples of 2-D HOW systems with w=3. (a) $HOW(4,3,2)$. (b) $HOW(5,3,2)$. (c) $HOW(6,3,2)$. (d) $HOW(7,3,2)$.

The next two theorems are pertinent:

Theorem 1.1.1 *The diameter of the $HOW(p, w, n)$ is $n\lceil \frac{p-1}{w} \rceil$.*

PROOF. In the worst case, a message may have to traverse all n dimensions to reach its destination. The diameter of the 1-D $HOW(p, w, 1)$ is $\lceil \frac{p-1}{w} \rceil$. It becomes $n\lceil \frac{p-1}{w} \rceil$ for the n-D $HOW(p, w, n)$. •

Theorem 1.1.2 *The number of channels in the n-D $HOW(p, w, n)$ is $np^{n-1}c_1$, where $c_1 = \frac{w}{2}(2p - w - 1)$ is the number of channels in the 1-D $HOW(p, w, 1)$.*

PROOF. The number of channels c_1 in the 1-D $HOW(p, w, 1)$ is $(p - w)w + \sum_{i=0}^{w-1} i$ or $(p-w)w+\frac{(w-1)w}{2}$ or $\frac{w}{2}(2p-w-1)$. This is because starting from the leftmost node and proceeding sequentially to the rightmost node in the colinear representation of the 1-D system, each node contributes w new channels except for the rightmost w nodes. The $i - th$ node from the right, where $0 \leq i \leq w - 1$, contributes i new channels. The proof for the n-D $HOW(p, w, n)$ is based on mathematical induction. The number of channels in the 2-D $HOW(p, w, 2)$ is $2pc_1$ because it contains p rows and p columns of 1-D $HOW(p, w, 1)$s. Let the number of channels c_{n-1} in the $(n - 1)$-D $HOW(p, w, n - 1)$ be $(n - 1)p^{n-2}c_1$. The n-D $HOW(p, w, n)$ is formed by connecting together in $HOW(p, w, 1)$ structures all nodes with the same address in p independent $HOW(p, w, n - 1)$s with p^{n-1} nodes each. Therefore, the number of channels c_n in the $HOW(p, w, n)$ is $pc_{n-1} + p^{n-1}c_1$ or $np^{n-1}c_1$. •

1.1.2 Further Implementation Issues

We will analyze the following systems and derive the equations for calculating their numbers of channels.

- the binary hypercube, (i.e. the m-cube);

- the k-ary n-cube;

- the generalized hypercube $GH(k,n)$;
- the 2-D $HOW(2^{\frac{m}{2}}, w, 2)$;
- the n-D $HOW(k,w,n)$.

Assume all systems have the same number N of processors, where $N = k^n = 2^m$. The following are the derivations for these systems.

- In the m-cube each node connects to m other nodes. Nodes share channels in pairs, so the total number of channels is $\frac{1}{2}m2^m$.
- In the k-ary n-cube each node has $2n$ neighbors and there are k^n nodes. The total number of channels is $\frac{1}{2}2nk^n = nk^n$.
- In the generalized hypercube $GH(k,n)$ each of the k nodes connects to the remaining $k-1$ nodes along one dimension. There are n dimensions and k^n nodes. The total number of channels is $n(k-1)\frac{k^n}{2}$.
- For the 2-D $HOW(2^{\frac{m}{2}}, w, 2)$, since the 1-D $HOW(k,w,1)$ is the building block we first calculate the number of channels in the 1-D HOW(k,w,1) system. For the first (starting from the left side) $k-w$ nodes, each node has w channels because the window size is w and connects to the w nodes to its right. Following this rule, no wire will be counted twice. For the rightmost w nodes, the number of channels will be $0+1+2+\cdots+w-1 = \frac{(w-1)w}{2}$. The total number of channels in the 1-D HOW(k,w,1) is $(k-w)w + \frac{(w-1)w}{2} = w((k-w)+\frac{w-1}{2}) = w(k-\frac{w+1}{2})$. For the 2-D $HOW(k,w,2)$, there are k^2 nodes, and each nodes has up to $4w$ neighbors. It can be viewed as k rows and k columns of $HOW(k,w,1)$ systems, and therefore the total number of channels is $2kw(k-\frac{w+1}{2})$.
- The n-D $HOW(k,w,n)$, contains k^n nodes and k^{n-1} 1-D $HOW(k,w,1)$ building blocks. Applying mathematical induction, we find that the total number of channels in the n-D $HOW(k,w,n)$ is $nk^{n-1}w(k-\frac{w+1}{2})$.

Table 1.1 compares the numbers of channels in the binary hypercube (i.e., m-cube), the k-ary n-cube (i.e., n-D torus), the generalized hypercube $GH(k, n)$, the 2-D $HOW(2^{\frac{m}{2}}, w, 2)$, and the n-D $HOW(k, w, n)$, all with the same number N of processors.

This dissertation focuses on 2-D HOW systems because of their simplicity, high bisection width, and ease of implementation. For a comparison, assume bidirectional data channels for full-duplex communications (i.e., simultaneous data transfers in both directions) and that $N = k^n = 2^m$ (therefore, $k = N^{1/n} = 2^{m/n}$). For an example, assume systems with $N = 16,384$ processors (i.e., $m = 14$) and 64-bit data channels; the numbers of wires in these systems are:

- $\frac{1}{2} * 14 * 2^{14} * 64 = 7 * 16384 * 64 = 7,340,032$ channels (also means 14,680,064 full-duplex bidirectional wires) for the 14-cube with diameter 14;

- $2 * 128^2 * 64 = 2,097,152$ channels (also means 4,194,304 full-duplex bidirectional wires) for the 128-ary 2-cube with diameter 128;

- $2 * 128^{2-1} * \frac{127*128}{2} * 64 = 133,169,152$ channels (also means 266,338,304 full-duplex bidirectional wires) for the $GH(128, 2)$ with diameter 2;

- $128 * 32 * (2 * 128 - 32 - 1) * 64 =$58,458,112 channels (also means 116,916,224 full-duplex bidirectional wires) for the $HOW(128, 32, 2)$ with diameter 8;

- $128 * 16 * (2 * 128 - 16 - 1) * 64 =$31,326,208 channels(also means 62,652,416 full-duplex bidirectional wires) for the $HOW(128, 16, 2)$ with diameter 16; and

- $128 * 8 * (2 * 128 - 8 - 1) * 64 =$16,187,392 (also means 32,374,784 full-duplex bidirectional wires) for the $HOW(128, 8, 2)$ with diameter 32.

For the comparative analysis of these results, we emphasize again that HOW systems with reasonable window size w are scalable, and could be implemented with

Table 1.1 Comparison of existing interconnection networks. All networks have $N = p^n = 2^m$ nodes.

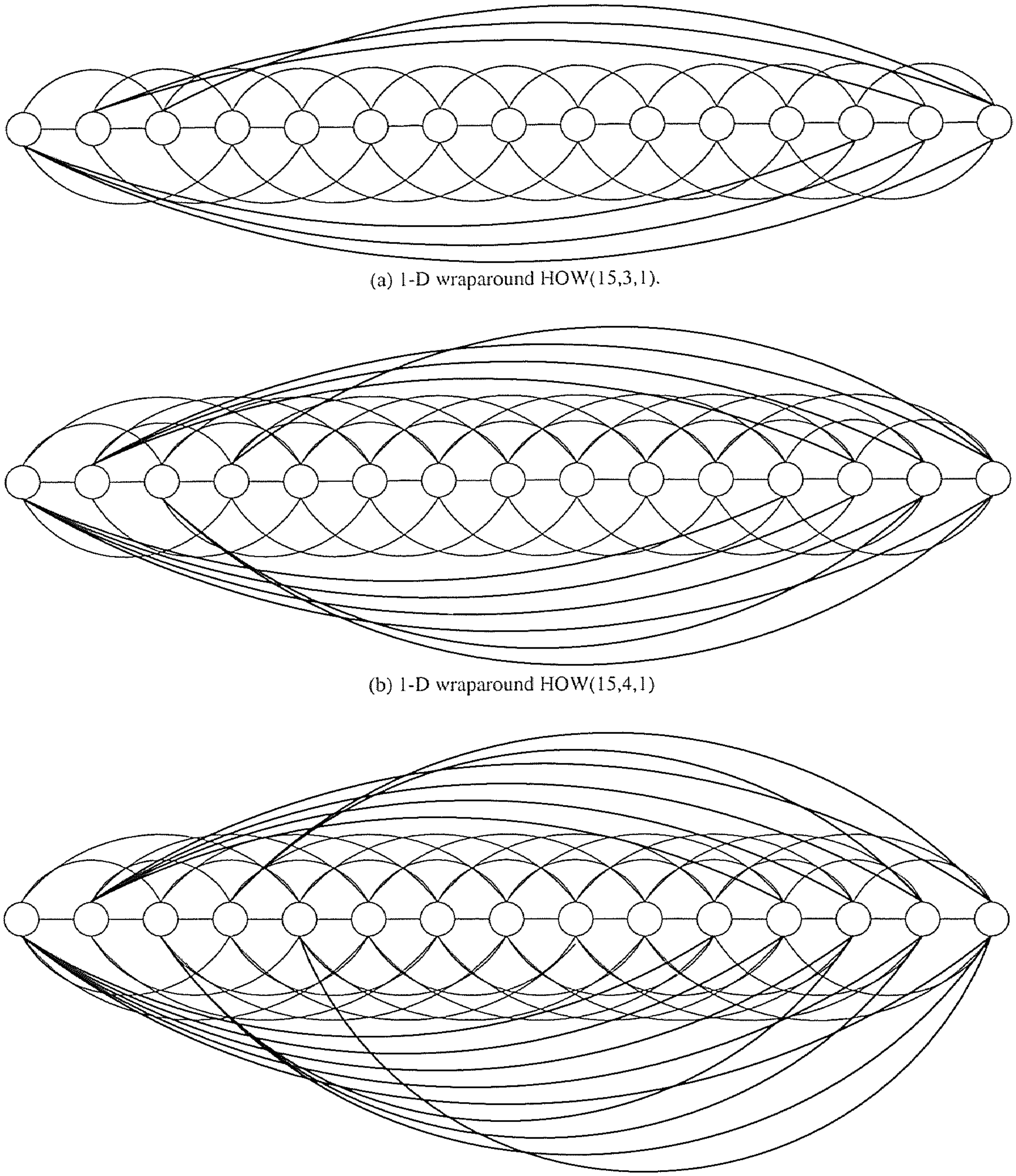
Network	Number of channels	Diameter
m-cube	$\frac{N}{2} * \log_2 N$	$m = \log_2 N = n * \log_2 p$
$N^{\frac{1}{n}}$-ary n-cube	$n * N$	$n * \lfloor \frac{N^{\frac{1}{n}}}{2} \rfloor = n * \lfloor \frac{p}{2} \rfloor$
$GH(N^{\frac{1}{n}}, n)$	$(N^{\frac{1}{n}} - 1) * n * \frac{N}{2}$	$\log_p N = n$
$HOW(\sqrt{N}, w, 2)$	$\sqrt{N} * w * (2 * \sqrt{N} - w - 1)$	$2 * \lceil \frac{\sqrt{N}-1}{w} \rceil = 2 * \lceil \frac{p-1}{w} \rceil$
$HOW(N^{\frac{1}{n}}, w, n)$	$\frac{n}{2} * N^{1-\frac{1}{n}} * w * (2 * N^{\frac{1}{n}} - w - 1)$	$n * \lceil \frac{N^{\frac{1}{n}}-1}{w} \rceil = n * \lceil \frac{p-1}{w} \rceil$

current and expected electronic and/or optical technologies because of the locality of their interconnects. In contrast, binary hypercubes are not scalable because the node degree increases with increases in the number of processors and, therefore, are difficult to build. Also, large generalized hypercubes are impossible to build because of their very large wiring complexity.

1.2 The Class of Wrap-Around HOW Architectures

Similarly to the wrap-around mesh, we introduce here wrap-around HOW architectures. For the wrap-around $HOW(k, w, 1)$ system, each node will have $2w$ neighbors, that is w neighbors to its left and w nodes to its right. Figure 1.5 shows the 1-D wrap-around $HOW(15, 3, 1)$, $HOW(15, 4, 1)$, and $HOW(15, 5, 1)$ systems. Figure 1.6 shows the 2-D wrap-around $HOW(7, 3, 2)$ system.

Each processor in the n-D wrap-around $HOW(p, w, n)$ has $2wn$ neighbors. The derivation of the total number of channels in the wrap-around $HOW(k, w, n)$ system is then as following. Because each node has $2nw$ neighbors and processors share channels in pairs, each processor contributes nw channels to the whole system. Therefore, the total number of channels is $k^{n-1} * k * nw = k^n nw$. Its diameter is half of that for the regular $HOW(k, w, n)$. Table 1.2 shows the comparison of different networks.



(a) 1-D wraparound HOW(15,3,1).

(b) 1-D wraparound HOW(15,4,1)

(c) 1-D wraparound HOW(15,5,1)

Figure 1.5 1-D wrap-around HOW systems with 15 processors and window size of 3, 4, and 5, respectively.

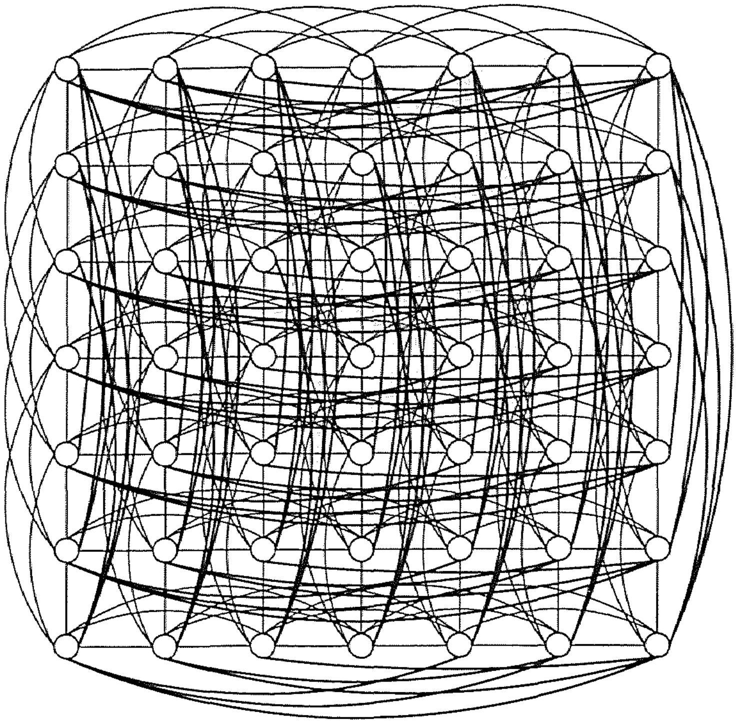


Figure 1.6 The 2-D wrap-around $HOW(7,3,2)$.

Table 1.2 Comparison of interconnection networks. All networks have $N = p^n = 2^m$ nodes.

Network	Number of channels	Diameter
m-cube	$\frac{N}{2} * \log_2 N$	$m = \log_2 N = n * \log_2 p$
$N^{\frac{1}{n}}$-ary n-cube	$n * N$	$n * \lfloor \frac{N^{\frac{1}{n}}}{2} \rfloor = n * \lfloor \frac{p}{2} \rfloor$
$GH(N^{\frac{1}{n}}, n)$	$(N^{\frac{1}{n}} - 1) * n * \frac{N}{2}$	$\log_p N = n$
$HOW(\sqrt{N}, w, 2)$	$\sqrt{N} * w * (2 * \sqrt{N} - w - 1)$	$2 * \lceil \frac{\sqrt{N}-1}{w} \rceil = 2 * \lceil \frac{p-1}{w} \rceil$
$HOW^{wrap}(\sqrt{N}, w, 2)$	$2 * w * N$	$2 * \lceil \frac{\sqrt{N}-1}{2w} \rceil = 2 * \lceil \frac{p-1}{2w} \rceil$
$HOW(N^{\frac{1}{n}}, w, n)$	$\frac{n}{2} * N^{1-\frac{1}{n}} * w * (2 * N^{\frac{1}{n}} - w - 1)$	$n * \lceil \frac{N^{\frac{1}{n}}-1}{w} \rceil = n * \lceil \frac{p-1}{w} \rceil$
$HOW^{wrap}(N^{\frac{1}{n}}, w, n)$	$n * w * N$	$n * \lceil \frac{N^{\frac{1}{n}}-1}{2w} \rceil = n * \lceil \frac{p-1}{2w} \rceil$

CHAPTER 2

COST ANALYSIS

In this chapter, a VLSI cost comparison between 1-D HOW systems and generalized hypercubes is presented. To determine the VLSI cost, we measure the number of wires and the complexity of the system based on the number of layers in the colinear layout of the circuit.

2.1 Cost Analysis for the Regular $HOW(p, w, 1)$

A VLSI cost comparison between 1-D HOWs and generalized hypercubes is presented. Since the focus of our attention in this dissertation are 2-D systems with p nodes in each dimension, this 1-D comparison is assumed to be carried out for each of the p rows and p columns in the 2-D systems (i.e., for their building blocks). The next definition is pertinent.

DEFINITION 2.1. *The crossing number of a graph is the minimum number of edge crossings needed to draw the graph in the plane* [27].

This number is related to the area needed to lay out the graph for VLSI implementation. To eliminate all edge crossings, several printed-circuit layers may have to be implemented. Not only does the number of layers affect the VLSI cost, but the thickness also of each layer contributes to the cost measure.

To determine the VLSI/wire cost, we measure the complexity of each system based on the minimum number of layers required in the colinear layout of the circuit for zero edge crossings and/or the width of each layer. In the colinear layout, all nodes in the 1-D system lie on the same straight line. The chosen rules of routing the wires for 1-D systems are:

- We consecutively number the processors 0, 1, 2, $\cdots$, $p-1$, from left to right.

- Going from left to right, for even-numbered processors the wires go to the top half of the printed-circuit board.
- For odd-numbered processors, the wires go to the bottom half of the printed-circuit board.

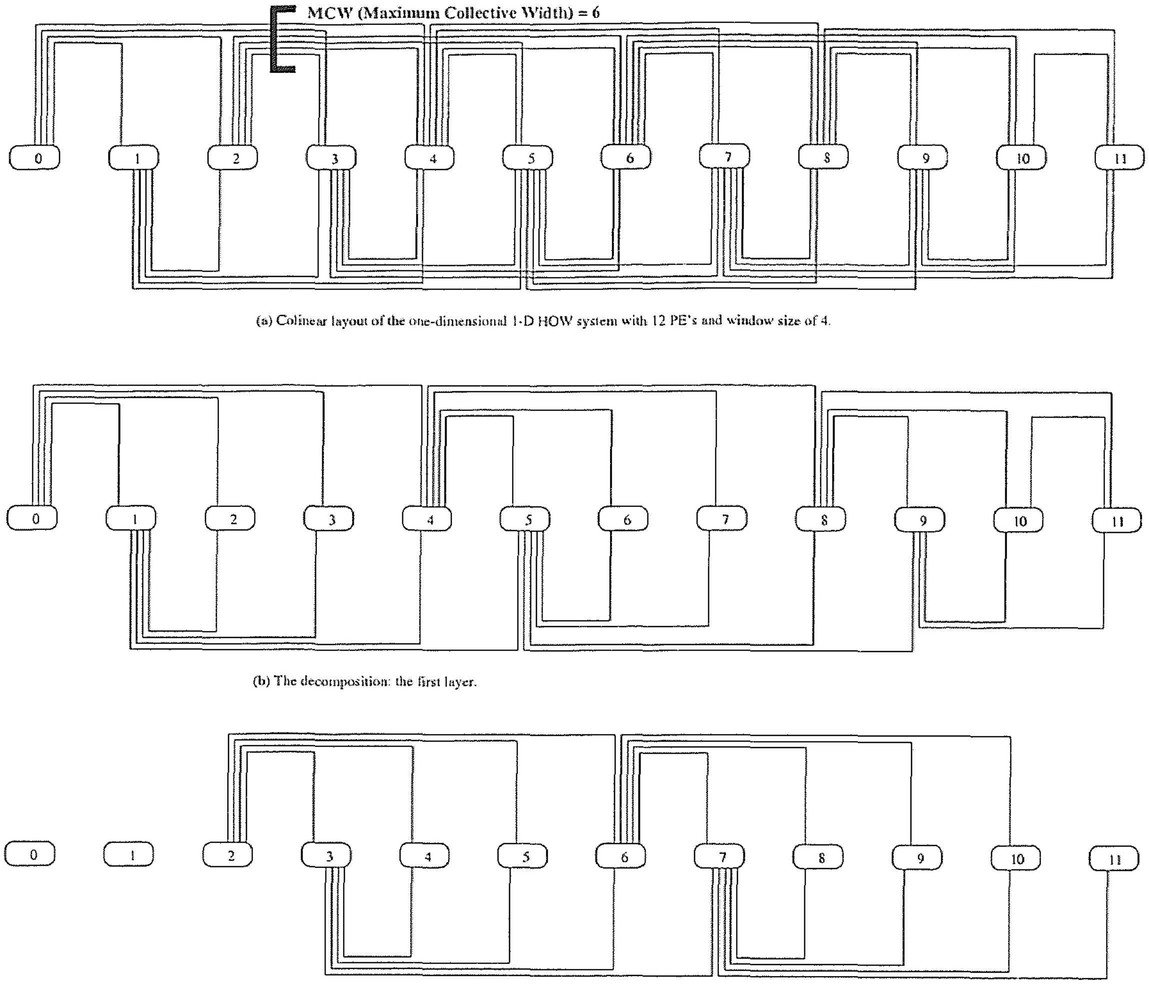
These basic rules of routing the wires minimize their maximum collective width, MCW (expressed in number of wires), in the x dimension. Figure 2.1 shows the colinear layout of the 1-D $HOW(12,4,1)$ and its brute-force decomposition for its implementation with two layers that eliminate all edge/wire crossings. However, the number of layers that eliminate all wire crossings depends on the value of w, and thus it increases with increases in the window size. For example, Figure 2.2 shows that the $HOW(12,5,1)$ requires three layers for the elimination of all wire crossings. The following theorems are pertinent.

Theorem 2.1.1 *The MCW in the colinear layout of the 1-D $HOW(p,w,1)$ with a single layer is*

$$MCW = \begin{cases} \frac{w}{2}(\frac{w}{2}+1) & \text{for even } w \\ (\frac{w+1}{2})^2 & \text{for odd } w \end{cases}$$

for practical cases with $w < \frac{p+1}{2}$. For the 1-D generalized hypercube $GH(p,1)$, the value of MCW is $(p-3)\phi + p - 1 - 2\phi^2$ with $\phi = \lfloor \frac{p-1}{4} \rfloor$.

PROOF. MCW can be determined by finding the maximum number of those wires that are located in either the upper or lower half of the layer between PE_{w-1} and PE_w. If w is even, then this maximum number corresponds to the lower half of the layer because PE_{w-1}, which is the rightmost PE in the leftmost window, is the last PE that contributes to MCW and contributes to the lower half (because it has an odd address). Therefore, we have PE_1 contributing two wires because it is connected to PE_w and PE_{w+1} outside of this leftmost window. PE_3 contributes four wires because it is connected to PE_w, PE_{w+1}, PE_{w+2} and PE_{w+3}, and so on.

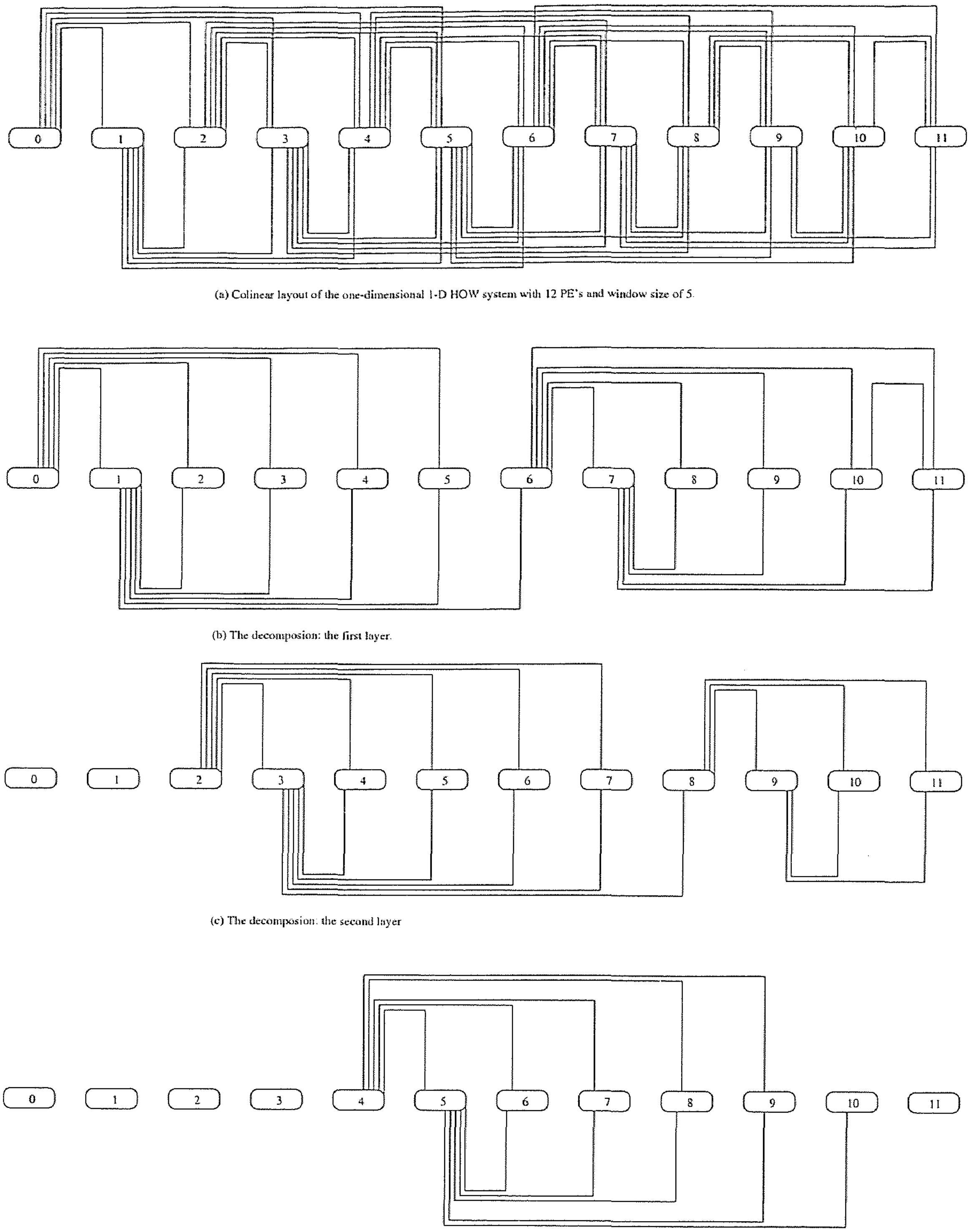


(a) Colinear layout of the one-dimensional 1-D HOW system with 12 PE's and window size of 4.

(b) The decomposition: the first layer.

(c) The decomposition: the second layer.

Figure 2.1 Colinear layout of the 1-D HOW system with 12 PEs and window size of 4, and its brute-force decomposition into printed-circuit layers.



(a) Colinear layout of the one-dimensional 1-D HOW system with 12 PE's and window size of 5.

(b) The decomposion: the first layer.

(c) The decomposion: the second layer

(d) The decomposion: the third layer.

Figure 2.2 Colinear layout of the 1-D HOW system with 12 PEs and window size of 5, and its brute-force decomposition into printed-circuit layers.

Therefore, we have $MCW = 2 + 4 + 6 + 8 + \cdots + w$, or $\sum_{i=1}^{\frac{w}{2}} 2i$ where $w/2$ is an integer or, finally, $\frac{w}{2}(\frac{w}{2} + 1)$. For odd w, however, MCW corresponds to the upper half of the layer because PE_{w-1}, which is the rightmost PE in the leftmost window, is the last PE that contributes to MCW and contributes to the upper half (because it has an even address). Therefore, we have PE_0 contributing one wire because it is connected to PE_w. PE_2 contributes three wires because it is connected to PE_w, PE_{w+1} and PE_{w+2}, and so on. Therefore, we have $MCW = 1+3+5+7+\cdots+w$, or $\sum_{i=0}^{\frac{w-1}{2}}(2i+1)$ where $\frac{w-1}{2}$ is an integer, or $2\sum_{i=1}^{\frac{w-1}{2}} i + (\frac{w-1}{2} + 1)$ or, finally, $(\frac{w+1}{2})^2$. To obtain these results, we assumed that all w wires leaving PE_{w-1} exist, and therefore $w - 1 + w < p$ or $w < \frac{p+1}{2}$. This should be expected to be the practical case for HOWs. However, the results do not cover generalized hypercubes because for them we have $w = p - 1$. Therefore, generalized hypercubes must be treated separately. Because of the symmetry in 1-D generalized hypercubes, without loss of generality we can find the MCW by focusing on the upper half of the printed-circuit. In fact, we can count the contribution of each PE in a left-to-right order. Let α be equal to $p - 1$. PE_0 contributes α wires because it is connected to α neighbors to its right. PE_2 contributes $\alpha - 4$ wires to MCW because it is connected to $\alpha - 2$ neighbors to its right and two levels of wires emanating from PE_0 can be reused (therefore, PE_2 also can use the same wire levels). Similarly, PE_4 contributes $\alpha - 8$ wires to MCW because it is connected to $\alpha - 4$ neighbors to its right and four levels of wires emanating from PE_0 can be reused. Similarly, PE_6 contributes $\alpha - 12$ wires to MCW because it is connected to $\alpha - 6$ neighbors to its right and six levels of wires emanating from PE_0 can be reused. In general, PE_i, where $i = 2j$, contributes $\alpha - 2j$ wires to MCW because it has $\alpha - j$ neighbors to its right and it can reuse j levels of wires emanating from PE_0. However, even-numbered PEs i for which $\alpha - i$ is negative or zero do not contribute to MCW. Therefore, contributing PEs have addresses $2i$, with $\alpha - 4i \geq 0$ or $i \leq \lfloor \frac{a}{4} \rfloor$. The value of MCW is then given by $\sum_{i=0}^{\phi}(\alpha - 4i)$, where $\phi = \lfloor \frac{a}{4} \rfloor$. This sum is also equal to $(p-3)\phi + p - 1 - 2\phi^2$. •

This theorem shows that HOWs have much smaller wire width (MCW) than generalized hypercubes for practical cases because this width is $O(w^2)$ and $O(p^2)$, respectively. The next theorem shows the number of printed-circuit boards (i.e., layers) required to eliminate all wire crossings when the brute-force decomposition of the type shown in Figure 2.1 is applied.

Theorem 2.1.2 *The number of layers that eliminate all wire crossings with brute-force decomposition of the $HOW(p, w, 1)$ is $\lceil \frac{w}{2} \rceil$. It becomes $1 + \lceil \frac{p-4}{2} \rceil$ for the generalized hypercube.*

PROOF. Assuming the wire routing rules defined earlier and the brute-force decomposition to produce zero wire crossings, we focus for the proof on a single window. Each layer deals with a pair of consecutive nodes within the window and there are $\lceil w/2 \rceil$ pairs. Thus, we need a total of $\lceil \frac{w}{2} \rceil$ layers for the $HOW(p, w, 1)$. For the generalized hypercube, going from left to right in the colinear representation of the system, each layer contains two successive nodes that connect to all other nodes to their right. However, up to four rightmost nodes can be combined in the last layer with zero wire crossings, and thus the total number of layers for the generalized hypercube is $1 + \lceil \frac{p-4}{2} \rceil$. •

We observe that the numbers of layers in HOWs and generalized hypercubes of similar size are $O(w)$ and $O(p)$, respectively. This is another advantage of HOWs that renders them more viable for implementation than generalized hypercubes.

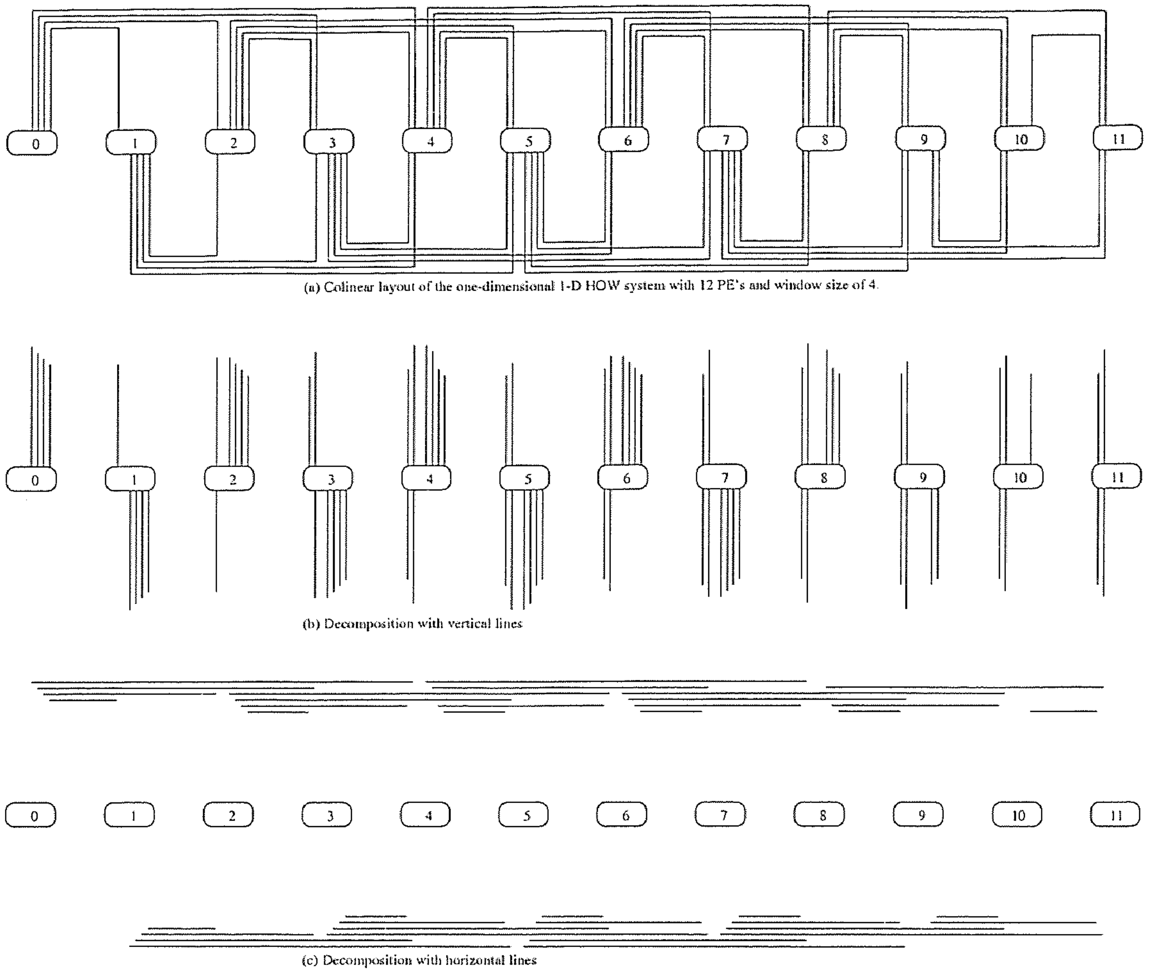
It is worth also mentioning here another wire routing technique, namely restricted routing [28], that requires only two layers for the implementation of any system represented in the 2-D space. As a result, both HOWs and generalized hypercubes require two printed-circuit layers regardless of their size. In the case of restricted routing, horizontal and vertical wire segments are laid on two different wiring layers. Figures 2.3, 2.4 and 2.5 demonstrate this technique for the $HOW(12,4,1)$, $HOW(12,5,1)$ and $GH(12,1)$ systems, respectively. Horizontal and vertical wires can then cross over each other without any electrical connection. If a connection is needed, a contact is placed at the respective intersection; these contacts contribute to the VLSI cost. Therefore, the total wiring cost with restricted routing has four components:

- The total number of wires. This number is $O(wp^2)$ and $O(p^3)$ for 2-D HOWs and GHs, respectively.
- The maximum collective width of wires, MCW (it affects the cost of the larger layer that contains the horizontal wires). This number is $O(w^2)$ and $O(p^2)$ for HOWs and GHs, respectively.
- The length of the wires. The maximum length is $O(w)$ and $O(p)$ for HOWs and GHs, respectively.
- The total number of electrical connections (contacts) between the two layers. This number is twice the total number of wires. Therefore, it is $O(wp^2)$ and $O(p^3)$ for HOWs and GHs, respectively.

Therefore, HOWs are superior to GHs even with restricted routing. We can conclude that HOWs are more prone to implementation than GHs for reasonable values of w. The following sections also show that HOWs can deliver very high performance.

2.2 Cost Analysis for the Wrap-Around $HOW(p, w, 1)$

Let us now further investigate the VLSI wire cost of HOWs with wrap-around connections. From Figures 2.6 and 2.7, it is very clear that the maximum collective width (MCW) increases with increases in the window size w. It is because in

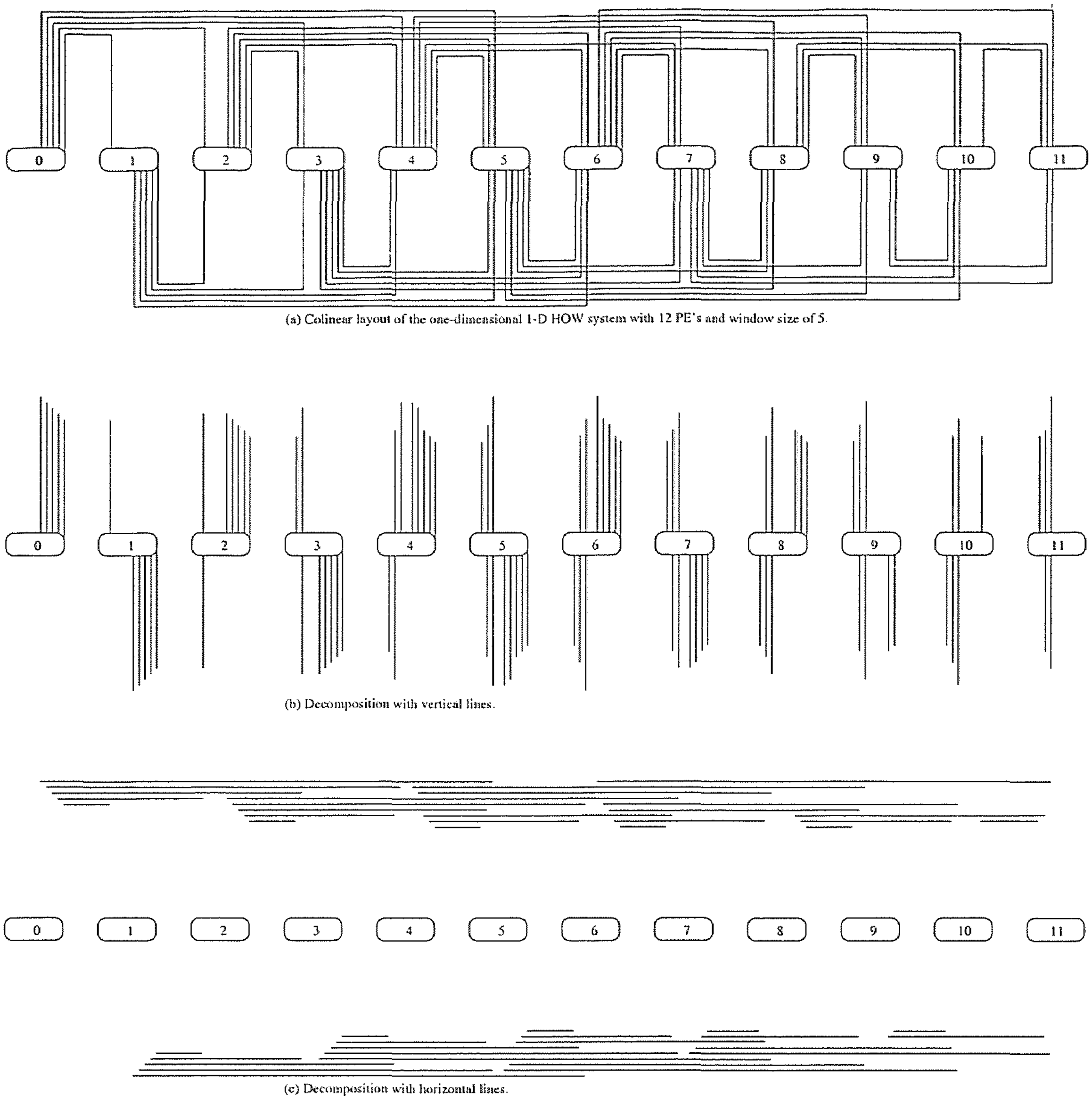


(a) Colinear layout of the one-dimensional 1-D HOW system with 12 PE's and window size of 4.

(b) Decomposition with vertical lines

(c) Decomposition with horizontal lines

Figure 2.3 Colinear layout of the 1-D HOW system with 12 PEs and window size of 4, and its decomposition into printed-circuit layers using vertical and horizontal lines.

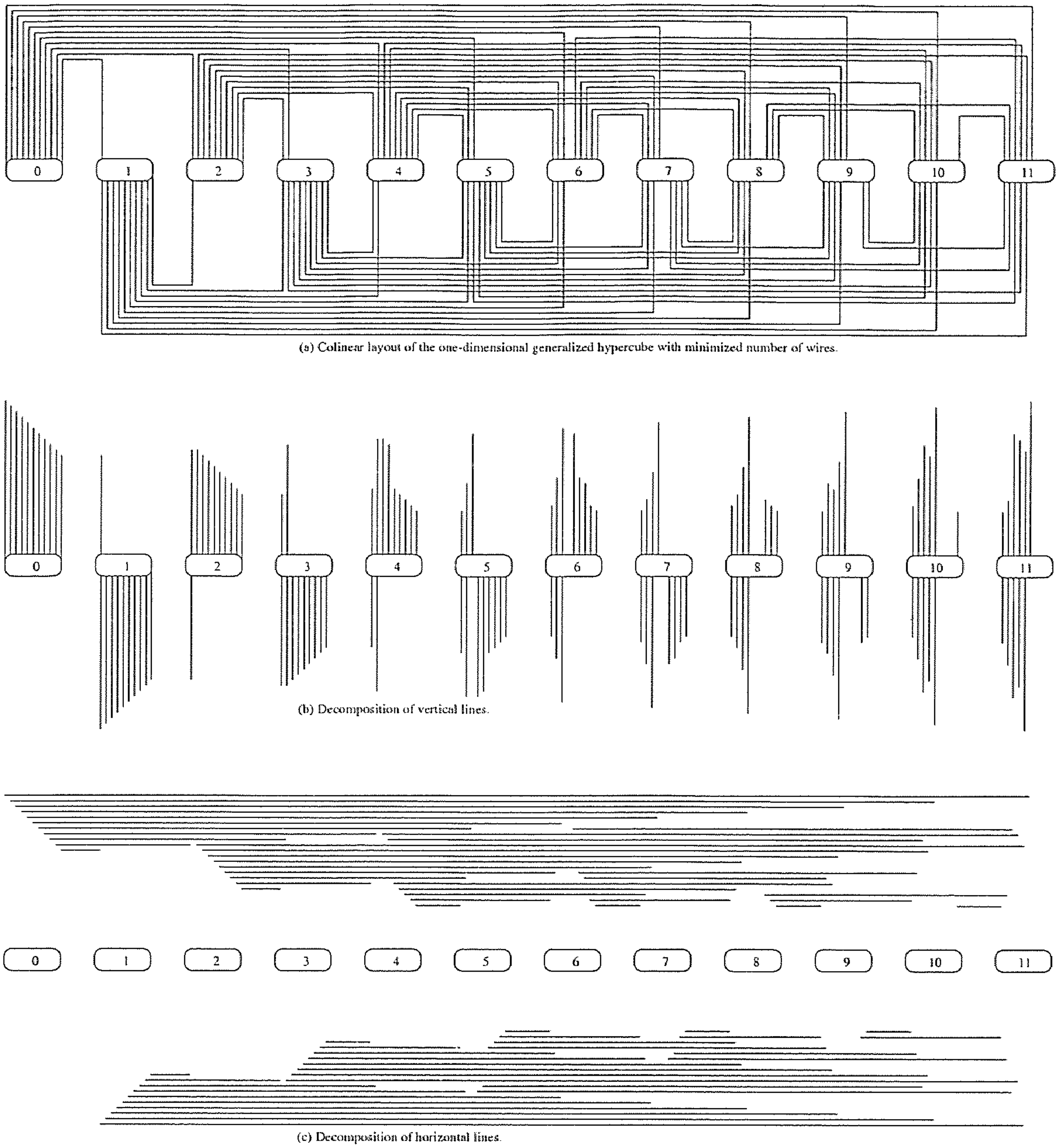


(a) Colinear layout of the one-dimensional 1-D HOW system with 12 PE's and window size of 5.

(b) Decomposition with vertical lines.

(c) Decomposition with horizontal lines.

Figure 2.4 Colinear layout of the 1-D HOW system with 12 PEs and window size of 5, and its decomposition into printed-circuit layers using vertical and horizontal lines.



(a) Colinear layout of the one-dimensional generalized hypercube with minimized number of wires.

(b) Decomposition of vertical lines.

(c) Decomposition of horizontal lines.

Figure 2.5 Colinear layout of generalized hypercube with 12 PEs, and its decomposition into printed-circuit layers using vertical and horizontal lines.

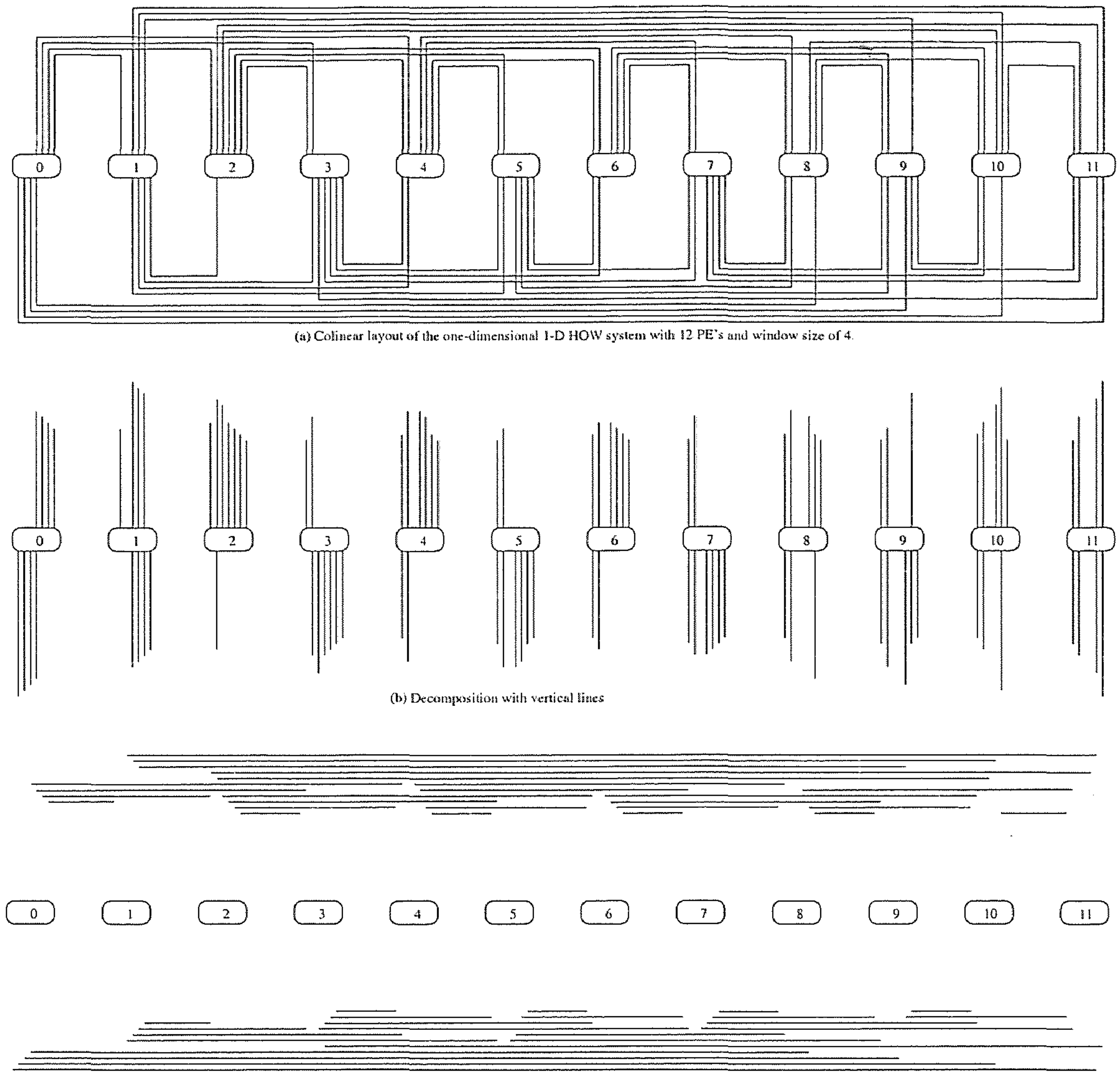
order to connect pairs of nodes belonging to the leftmost and rightmost windows, respectively, in the colinear layout, the wires will cross the entire printed-circuit plane. The number of wires needed to connect all nodes in the two opposite ends is $w + (w-1) + (w-2) + ... + 1 = \frac{w(w+1)}{2}$. Of course, we could split the wires equally between the upper and lower halves of the layer.

The following theorem determines the value of MCW.

Theorem 2.2.1 *The MCW in the colinear layout of the wrap-around 1-D HOW(p,w,1) with a single layer is*

$$MCW = \begin{cases} \frac{w}{2}(\frac{w}{2}+1) + \lceil\frac{w(w+1)}{4}\rceil & \textit{for even } w \\ (\frac{w+1}{2})^2 + \lceil\frac{w(w+1)}{4}\rceil & \textit{for odd } w \end{cases}$$

PROOF. Refer to the proof for the regular 1-D HOW(p,w,1). The total number of extra wires for the wrap-around system is $\frac{w(w+1)}{2}$. We could split the wires equally between the upper and lower halves of the layer. So, we need to add $\lceil\frac{w(w+1)}{4}\rceil$ to the equation for the regular 1-D HOW(p,w,1). •



(a) Colinear layout of the one-dimensional 1-D HOW system with 12 PE's and window size of 4.

(b) Decomposition with vertical lines

(c) Decomposition with horizontal lines

Figure 2.6 Decomposition of the 1-D wrap-around $HOW(12, 4, 1)$.

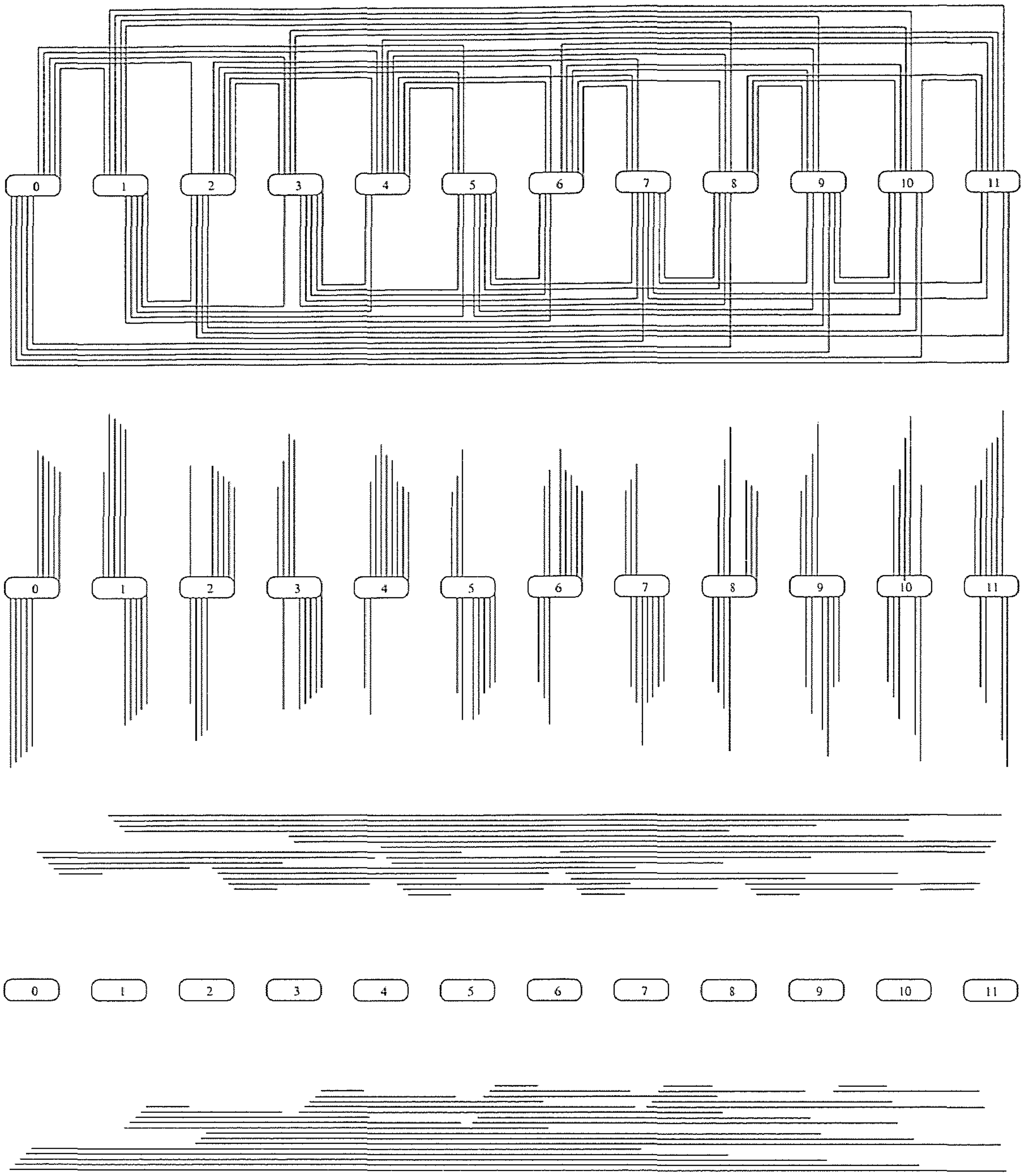


Figure 2.7 Decomposition of the 1-D wrap-around $HOW(12, 5, 1)$.

CHAPTER 3

1-D HOW SYSTEM EMBEDDINGS

In this chapter, we discuss embeddings of various widely-used interconnection networks into 1-D HOW systems. Such embeddings could prove very beneficial as HOW and related systems demonstrate significant promise in scalable parallel processing [18] [19] [29] [30] [31].

Some definitions are pertinent for the analysis of results. Given two graphs $G(V, E)$ and $G^{'}(V^{'}, E^{'})$, embedding the graph G into the graph $G^{'}$ results in the mapping of each vertex in the set V onto a vertex in the $V^{'}$ and of each edge in the set E onto an edge, or a set of edges in $E^{'}$. There are three important parameters that determine the quality of mapping a graph $G(V, E)$ onto a graph $G^{'}(V^{'}, E^{'})$.

- **Dilation** of a source edge in E: the number of edges in $E^{'}$ that the edge in E is mapped onto.

- **Congestion** of a target edge in $E^{'}$: the number of source edges mapped onto the edge in $E^{'}$.

- **Expansion**: the ratio of the number of nodes in the set V' to that in the set V.

Example: referring to the figure 3.1, there are two graph: source graph $G(4, 2)$, target graph $G^{'}(9, 8)$. The parameters are following:

- dilation of (A,B): is 5.

- dilation of (C,D): is 4.

- congestion of (K,L): is 2.

- expansion: is 9/4.

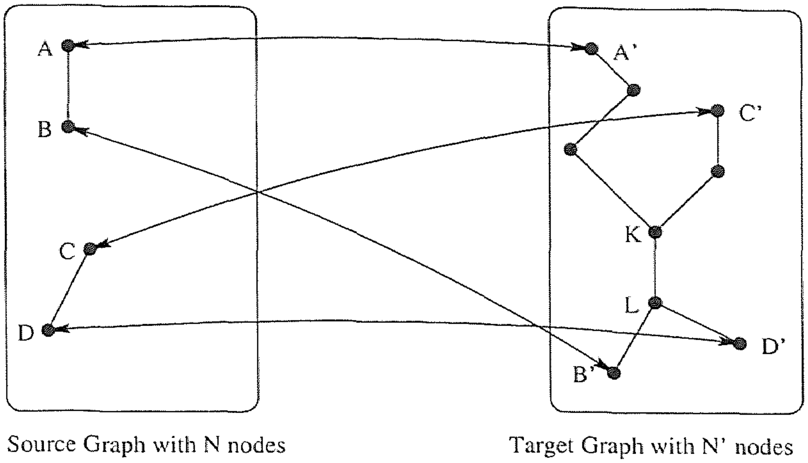


Figure 3.1 The definition of dilation, congestion and expansion.

In this dissertation, we try, if possible, to limit the scope of the discussion to cases where the expansion is one, for the sake of cost effectiveness.

3.1 Embedding a Ring into a 1-D HOW System

A ring of p nodes with addresses 0 to $p-1$ can be embedded into a 1-D HOW system with p nodes by mapping the ring processor with address i, where $i = 0, 1, 2, \ldots, p-1$, onto the distinct processor x, where $x = 0, 1, 2, \ldots, p-1$. Our embedding procedure distinguishes between even and odd addresses x, with $x = 2k$ and $x = 2k+1$, respectively, in the 1-D system and uses the function $i = G(k)$ to get the address i of the corresponding processor in the ring. The function $G(k)$ is defined as follows:

$$G(k) = \begin{cases} k & \text{if } x = 2k, \text{ for } k = 0, 1, 2, \ldots, \lfloor \frac{p-1}{2} \rfloor \\ (p-1) - k & \text{if } x = 2k+1, \text{ for } k = 0, 1, 2, \ldots, \lceil \frac{p-1}{2} \rceil - 1 \end{cases}$$

It is easy to see that this mapping technique requires a window size of at least $w = 2$ for optimal mapping (i.e., the dilation is one). Figure 3.2 illustrates the embedding of a sixteen-processor ring into a 1-D HOW system, also with sixteen processors.

For $w > 2$, we can also use several other embedding functions for optimal mapping, including the function $G^{'}(k)$ that follows:

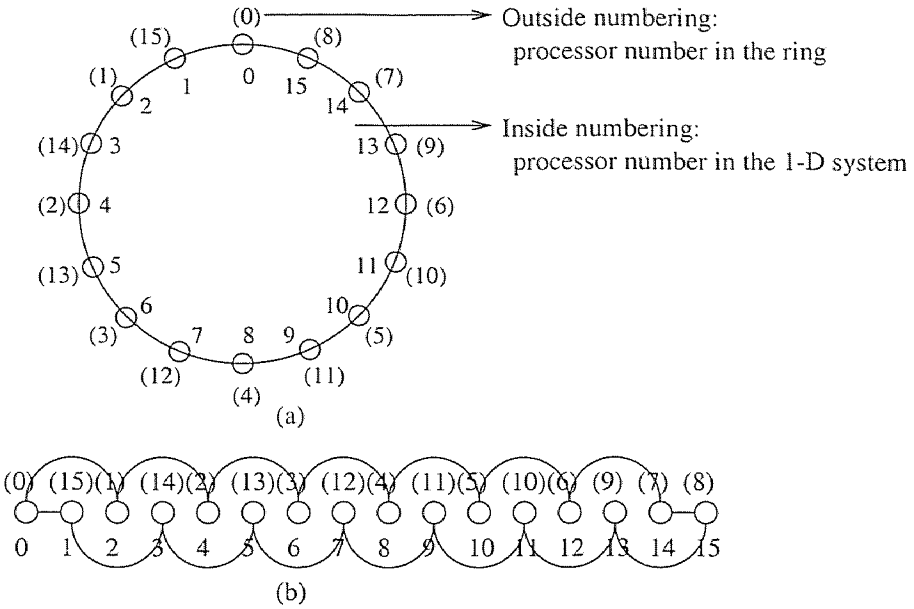


Figure 3.2 (a) A 16-processor ring and (b) its embedding into the 1-D HOW(p,w,1) system.

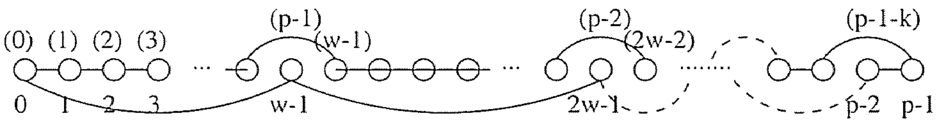


Figure 3.3 Embedding a p-processor ring into the 1-D HOW(p,w,1) system with another technique.

$$G'(k) = \begin{cases} (w-1)k + (w-j) & \text{if } x = (k+1)w - j, \text{ for } j = 2,3,4,\ldots,w \\ & \text{and } k = 0,1,2,\ldots,\lfloor\frac{p-1}{w}\rfloor \\ (p-1) - k & \text{if } x = (k+1)w - 1, \text{ for } k = 0,1,2,\ldots,\lceil\frac{p-1}{w}\rceil - 1 \end{cases}$$

Figure 3.3 illustrates the general embedding of a p-processor ring into the 1-D HOW(p,w,1) system, using the function G'. All proposed embeddings have dilation one, congestion one, and expansion one.

3.2 Embedding a 2-D Mesh into a 1-D HOW System

We present here embedding techniques for the 2-D mesh and torus topologies. The target architectures are 1-D HOW systems. In a subsequent section, we will show that much better embeddings can be derived if the target HOW systems are 2-D.

3.2.1 2-D Regular Mesh

Considering a $p \times n$ mesh with p rows and n columns, we can embed this mesh into a $(p \times n)$-processor 1-D HOW system by mapping the processor (i, j), where $i = 0,1,2,\ldots,p-1$ and $j = 0,1,2,\ldots,n-1$, of the mesh onto the processor $x = H(i,j)$, where $x = 0,1,2,\ldots,(pn-1)$ of the 1-D system. The function $H(i,j)$ is defined as follows:

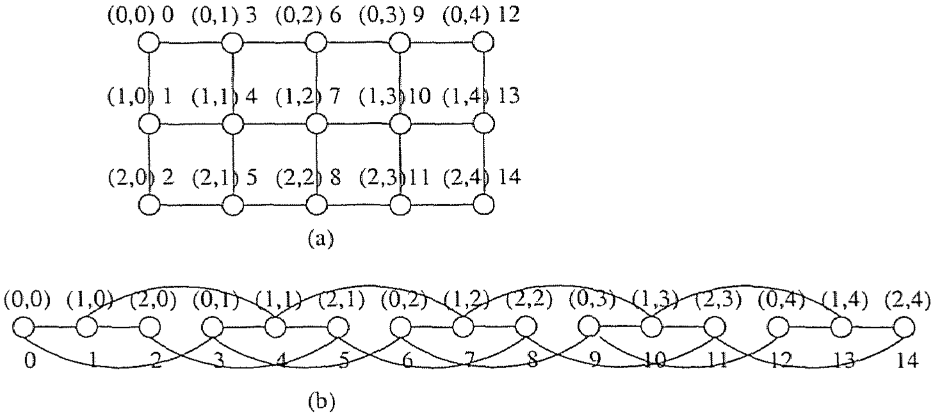
$$H(i,j) = \begin{cases} i + jp & \text{if } p \leq n, \text{ for } i = 0,1,2,\ldots,p-1 \text{ and } j = 0,1,2,\ldots,n-1 \\ ip + j & \text{if } p > n, \text{ for } i = 0,1,2,\ldots,p-1 \text{ and } j = 0,1,2,\ldots,n-1 \end{cases}$$

This mapping of a mesh onto a 1-D system has the following properties:

- If $p < n$, with column-wise mapping of mesh nodes and window size of at least p the mapping is optimal (with dilation one).

- If $p > n$, with row-wise mapping of mesh nodes and window size of at least n the mapping is optimal.

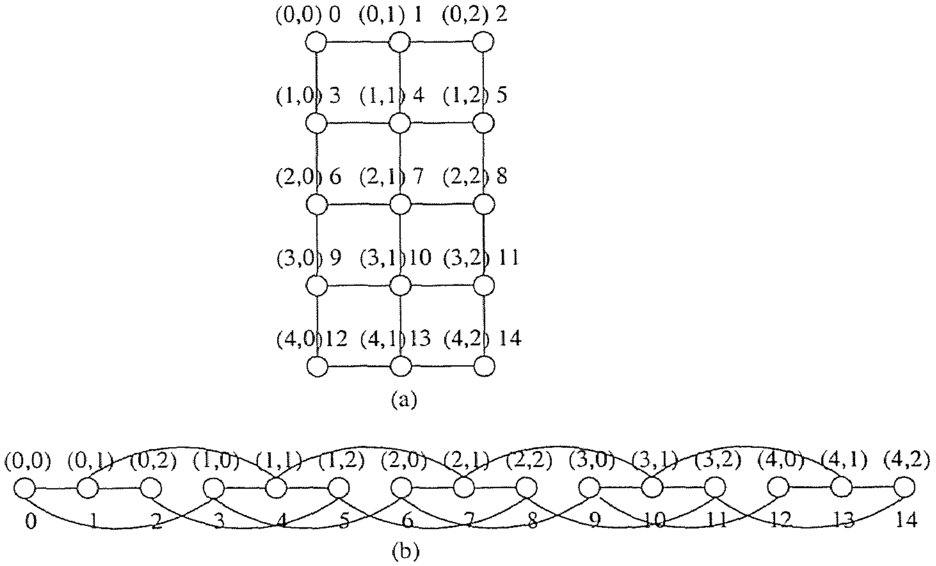
- If $p = n$, the row-wise mapping is the same as the column-wise mapping.



(a)

(b)

Figure 3.4 (a) Source 3x5 mesh and (b) its optimal embedding into the 1-D HOW(15,3,1) system.



(a)

(b)

Figure 3.5 (a) Source 5x3 mesh and (b) its optimal embedding into the 1-D HOW(15,3,1) system.

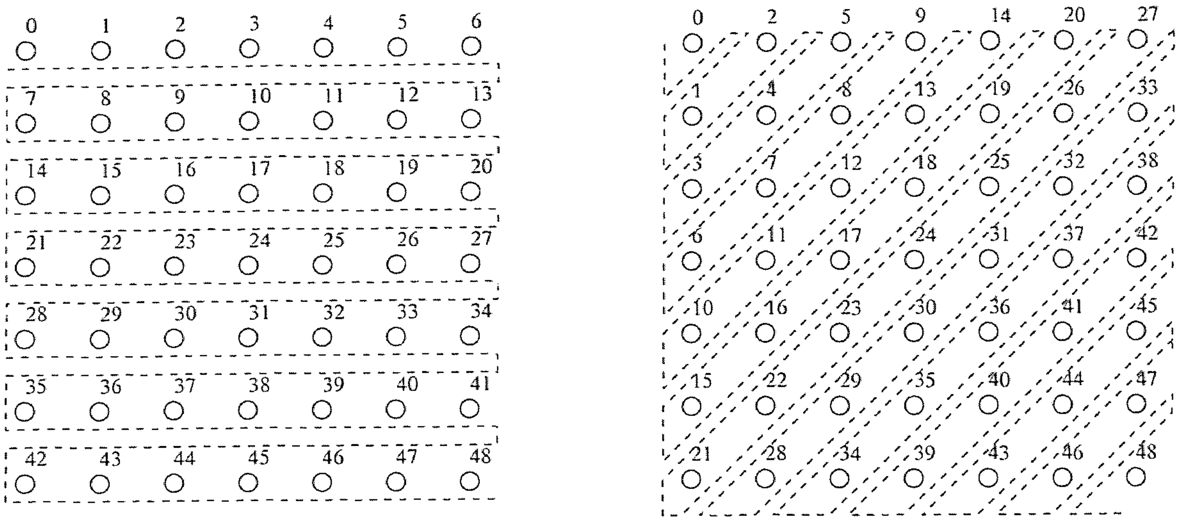
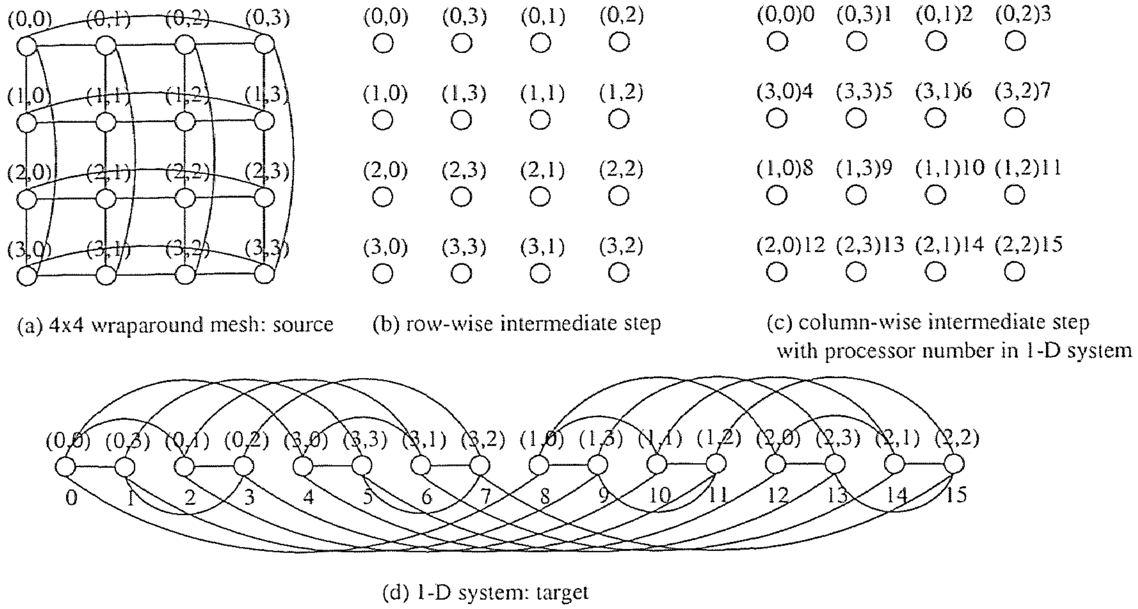


Figure 3.6 Mapping the 7 × 7 mesh in two different ways.

- The window size must be at least $p = n = min\{p, n\}$ for an optimal mapping.

Figures 3.4 and 3.5 illustrate optimal embeddings of the 3x5 and 5x3 meshes, respectively, into the 1-D HOW(15,3,1) system. In order to get an optimal mapping, the window size w should be at least equal to the $min\{p, n\}$. If $min\{p, n\} > w \geq \lfloor \frac{min\{p,n\}}{2} \rfloor$, then the mapping is suboptimal with maximum dilation two; if $\lfloor \frac{min\{p,n\}}{2} \rfloor > w \geq \lfloor \frac{min\{p,n\}}{3} \rfloor$, then the mapping is suboptimal with maximum dilation three; etc. In the general case, if $\lfloor \frac{min\{p,n\}}{m} \rfloor > w \geq \lfloor \frac{min\{p,n\}}{m+1} \rfloor$, where m is a position integer, then the embedding has maximum dilation $m + 1$. The expansion and congestion are both one.

The best mapping is not unique. For example, we can use another way to get a best mapping for the same window size. Figure 3.6 is an example to map the 7 × 7 mesh onto a 1-D system using two different ways; the first mapping applies row-major order while the second mapping is along the diagonals (i.e., along the dashed lines).



(a) 4x4 wraparound mesh: source

(b) row-wise intermediate step

(c) column-wise intermediate step with processor number in 1-D system

(d) 1-D system: target

Figure 3.7 4x4 wraparound mesh and its optimal mapping onto the 1-D HOW(16,8,1) system.

3.2.2 2-D Wraparound Mesh or Torus

Embedding a $p \times n$ wraparound mesh into a 1-D system is a natural combination and extension of embedding $(p + n)$ rings and a 2-D mesh into a 1-D system. We can embed a $p \times n$ wraparound mesh into a $(p \times n)$-processor 1-D HOW system by mapping the processor (i, j) of the torus onto the processor $(G(i, j); \text{where } j \text{ is fixed}) \| (G(i, j); \text{where } i \text{ is fixed}) \| H(i, j)$. The symbol $\|$ denotes concatenation of two different mappings onto the 1-D HOW system. The functions "G" and "H" were defined earlier for the ring and mesh embeddings.

Figure 3.7 is a step-by-step example for mapping a 4x4 wraparound mesh. This mapping of a wraparound mesh onto a 1-D system is a natural combination/extension of ring and mesh mappings, and therefore it inherits all the properties associated with the latter mappings. For example, the window size should be at least equal to $2 \times min\{p, n\}$ for optimal mapping.

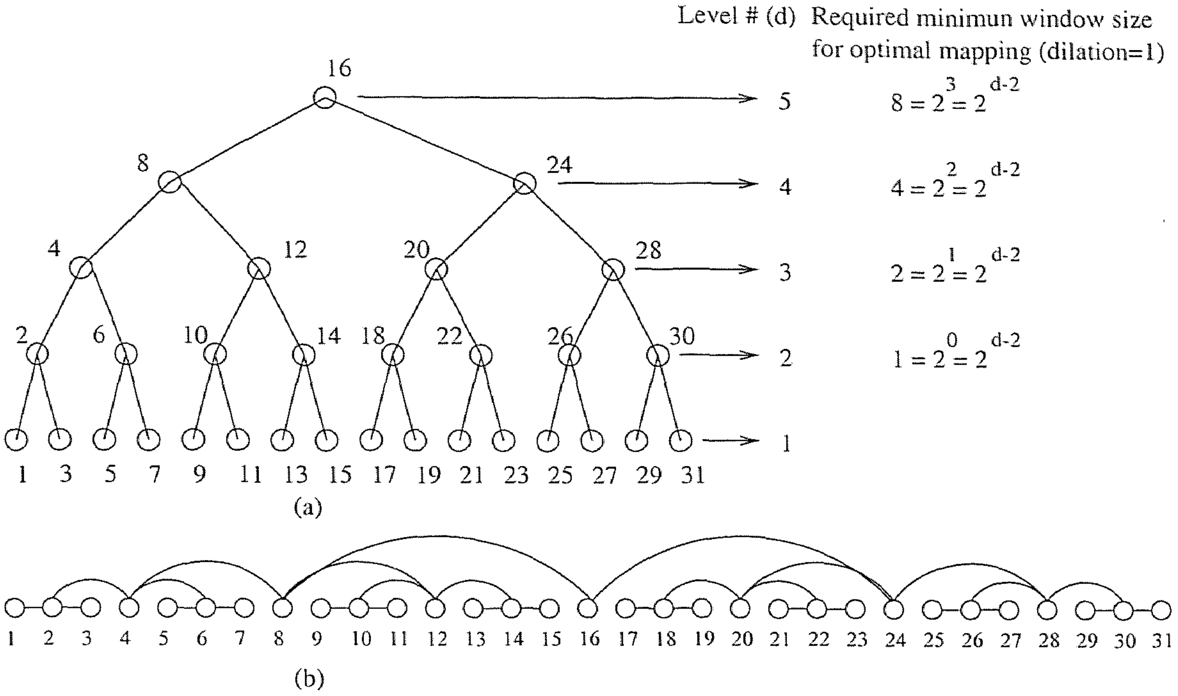


Figure 3.8 (a) A 31-processor full binary tree with depth $d = 5$ and the numbering of its nodes and (b) its optimal embedding into the 1-D HOW(31,8,1) system.

3.3 Embedding a Binary Tree into a 1-D HOW System

Binary trees can be embedded into a 1-D system in several ways. Consider a full binary tree of level d containing $2^d - 1$ processors. A good embedding can be derived by numbering the nodes of the full tree in the manner shown in the example of Figure 3.8. The number assigned to a tree node then denotes the address of the corresponding node in the HOW system.

This mapping has the following properties:

- It is a recursive mapping
- The required minimum window size for optimal mapping is 2^{d-2}, where d is the level of the full tree.

If $w < 2^{d-2}$, the mapping is suboptimal. To find the dilation for suboptimal mapping, the following proposition is pertinent.

Proposition 1. For the embedding of a full binary tree with depth d and $2^d - 1$ nodes into a 1-D HOW system with the same number of nodes, we need 2^{d-i-1} connections at distance 2^i in the 1-D linear-array configuration of the HOW system, for $i = 0, 1, 2, ..., d-2$.

Corollary 1. The maximum dilation for a suboptimal embedding is $\lceil \frac{2^{d-2}}{w} \rceil$ and corresponds to two source edges.

Corollary 2. If w is not a power of two, the maximum congestion for suboptimal embedding is one. Otherwise, for $w = 2^v$, with $v < d-2$, the maximum congestion is $d-2$ and corresponds to two target edges.

3.4 Embedding a Hypercube into a 1-D HOW System

A d-dimensional (direct binary) hypercube consists of $p = 2^d$ processors, and a $(d+1)$-dimensional hypercube is constructed by connecting together pairs of processors with the same addresses in two d-dimensional hypercubes [9] [14]. Two nodes are neighbors in the d-dimensional hypercube if and only if their unique d-bit addresses differ in a single bit [1] [9] [11].

We can embed the d-dimensional hypercube into the 1-D HOW system with 2^d nodes by mapping each hypercube node to the node with the same address in the HOW system. The window size should be at least equal to 2^{d-1} for optimal mapping. For a large value of d, the mapping should normally produce large dilation because of the large difference in the dimensionalities of the two systems. A large dimensional HOW system could produce very good results. For this reason, we avoid further analysis of this mapping for 1-D HOW systems.

Figure 3.9 shows the embedding of a 16-processor hypercube into a 16-processor 1-D HOW system.

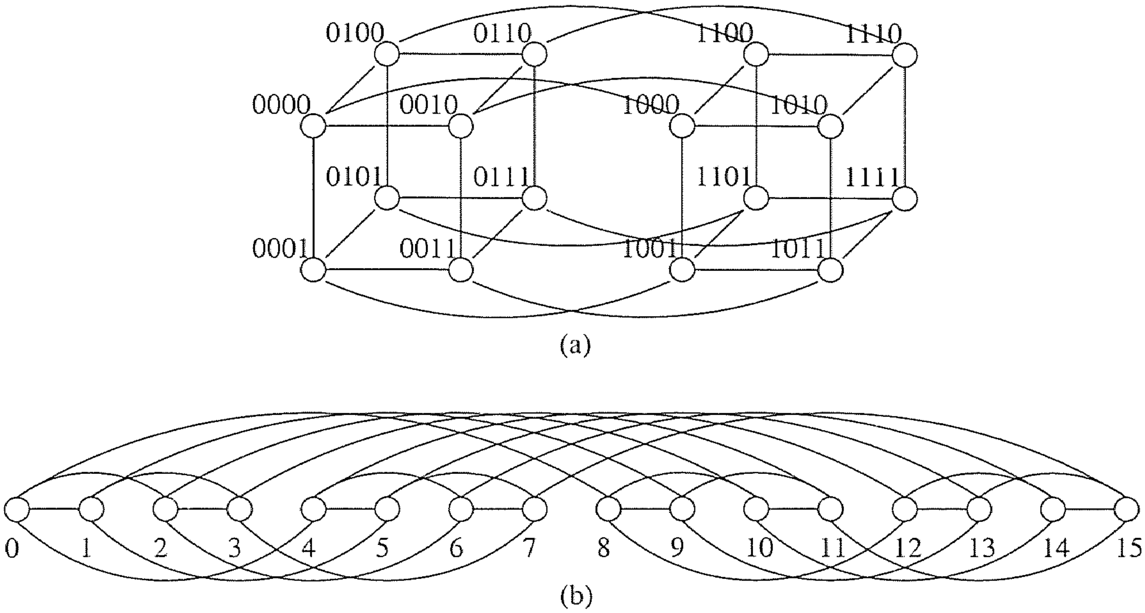


Figure 3.9 (a) A 16-processor hypercube with binary addresses for its nodes and (b) its optimal embedding into a 1-D HOW(16,8,1) system.

CHAPTER 4

2-D HOW SYSTEM EMBEDDINGS

In the following sections, we propose embeddings of various interconnection networks into 2-D HOW systems. We limit the scope of the discussion to cases in which the number of rows and the number of columns are the same, and equal to n, in the 2-D HOW system. 2-D HOW system embeddings are extensions of 1-D HOW system embeddings. Therefore, everything we discuss here is based on the preceding section.

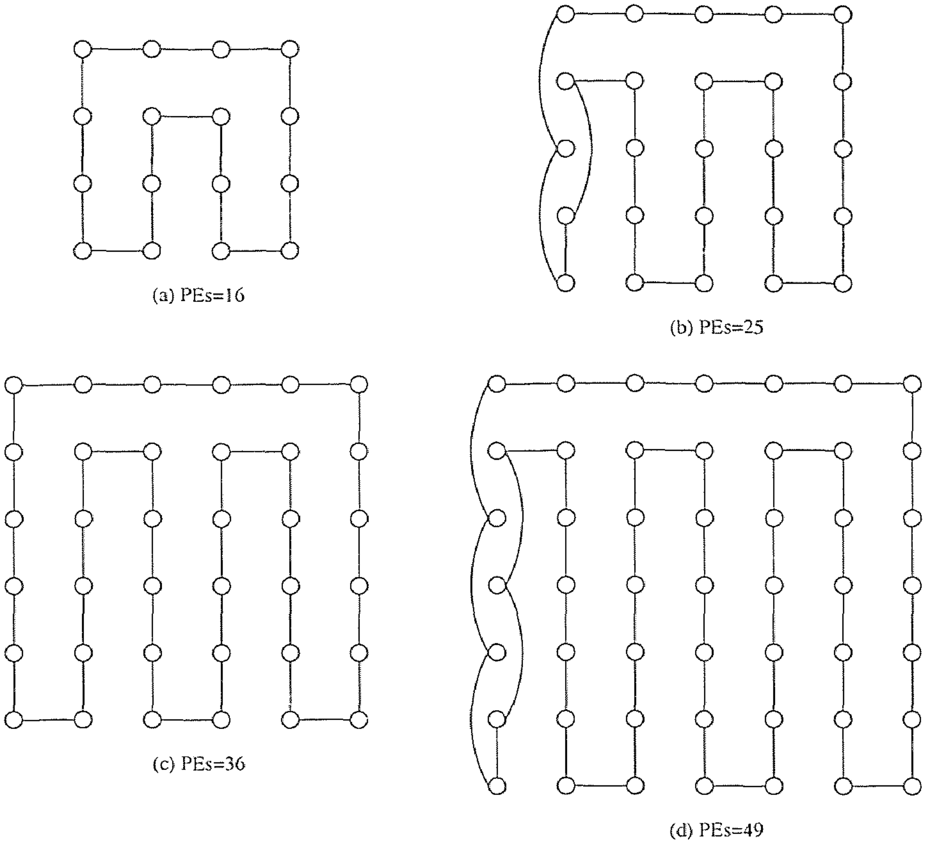
4.1 Embedding a Ring into a 2-D HOW System

An optimal ring mapping is always possible with expansion one if the window size is at least 2. We visit the nodes in a serpentine-like, column-wise way where the first column is scanned sequentially for an even number of rows. In this case, even with $w = 1$ we produce an optimal mapping. For an odd number of rows, the nodes in the first column cannot be visited sequentially, but still an optimal mapping exists for $w \geq 2$, as shown in Figure 4.1.

If the number of processors in the source graph is $(n-1)^2 < PEs < n^2$, where n is a positive integer, we just use one or more links connecting nodes at distance 2 to bypass several processors in the 2-D HOW system for optimal mapping, as shown in Figure 4.2.

4.2 Embedding a 2-D Mesh/Torus into a 2-D HOW System

It is straightforward to get an optimal mapping for the regular mesh. We apply the ring mapping for 1-D systems in individual rows and columns. An optimal mapping for wraparound edges of the torus does not exist if the target graph does not contain fully-connected rows and columns (we assume an expansion equal to one).


(a) PEs=16

(b) PEs=25

(c) PEs=36

(d) PEs=49

Figure 4.1 Embeddings of rings into 2-D HOW systems.

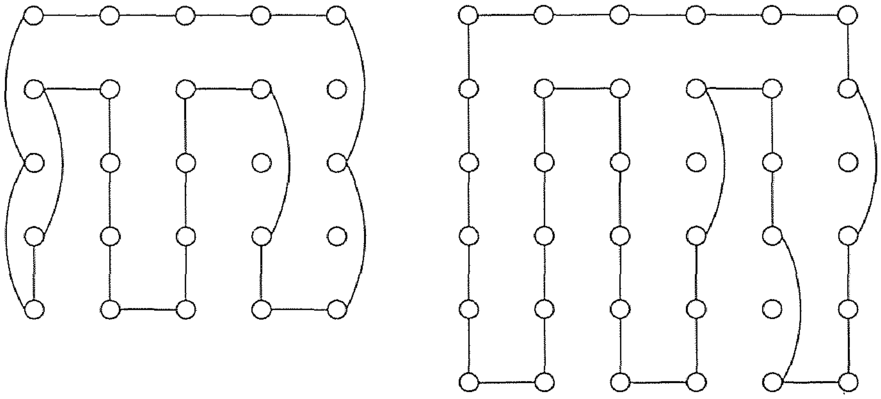
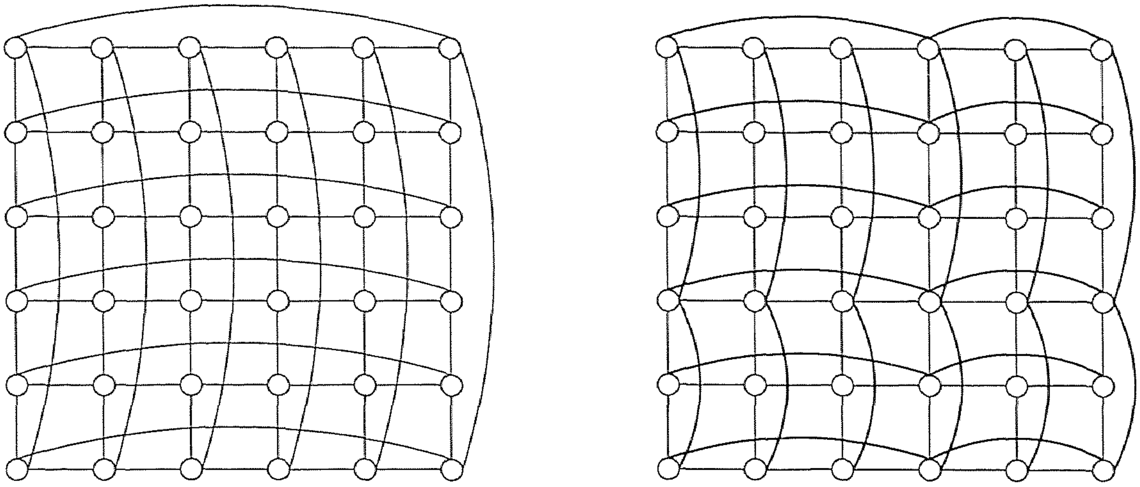


Figure 4.2 Embeddings of rings into 2-D HOW systems when the numbers of nodes in the rings are smaller than those in the HOW systems.



(a) 6x6 torus

(b) mapping onto the 2-D system

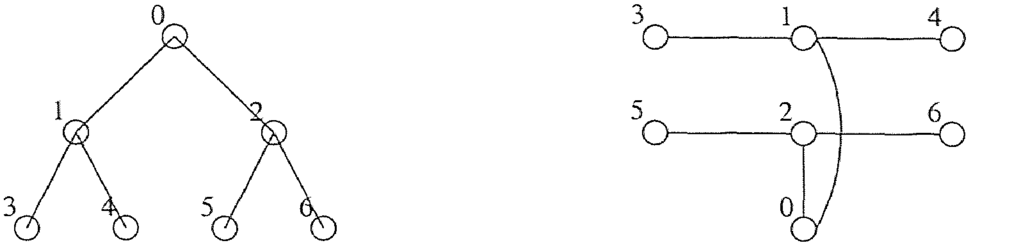
Figure 4.3 Mapping the 6 × 6 torus onto a 2-D HOW system with window size of 3. Consecutive bold segments in a row/column implement wraparound connections in the torus.

Otherwise, for the wraparound mesh (i.e., torus) we split the wraparound connections into a minimal number of segments based on the window size provided by the 2-D HOW system. Some target processors are then used not only to process data but also to forward data destined to otherwise neighbors in the torus. The target system should still be expected to perform very well for algorithms employing the torus.

The dilation of wraparound connections is then $\lceil \frac{n-1}{w} \rceil$. The mapping of torus wraparound links can always be chosen so that the maximum congestion is one, assuming that $w \geq 2$. Figure 4.3 is an example to map the 6 × 6 torus onto the 2-D HOW(6,3,2) system, using a window size of 3.

4.3 Embedding a Binary Tree into a 2-D HOW System

Binary trees can be embedded into 2-D HOW systems in several ways. Such an embedding could be used for the implementation of data reduction operations [13]. Consider a full binary tree of level l containing $2^l - 1$ processors and the 2-D



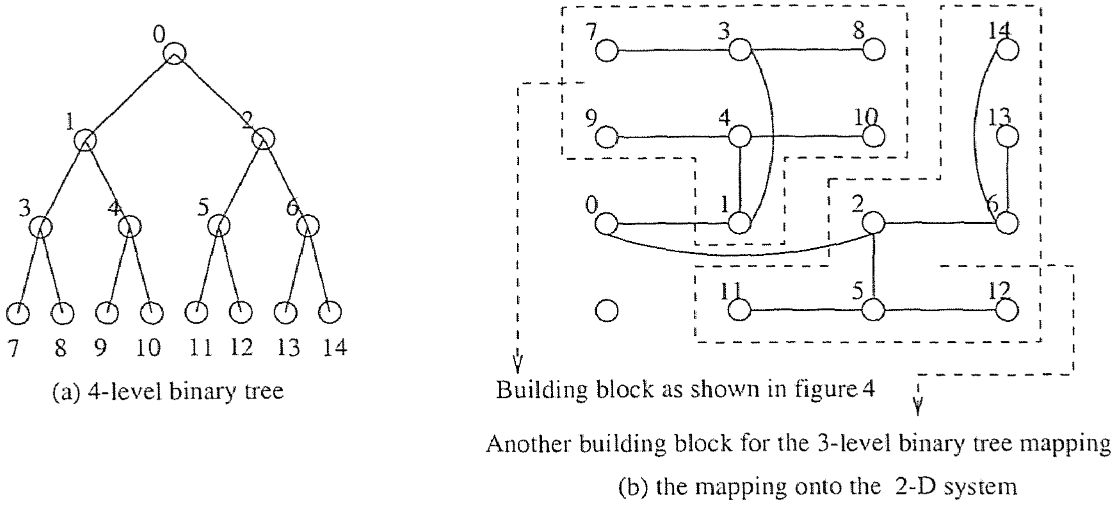
(a) 3-level binary tree

(b) the mapping onto the 2-D system

Figure 4.4 Optimal mapping of the 3-level binary tree onto the 2-D HOW(3,2,2) system.

HOW($\lceil\sqrt{2^l - 1}\rceil$, w,2) system for the smallest expansion. We assume that $w \geq 2$. The two basic building blocks used in our binary tree mapping are for the 3-level tree, and are shown in Figures 4.4 and 4.5. These two building blocks and their mirror images are employed for the mapping of larger trees. For example, Figure 4.6 shows a mapping where the building block #1 at the upper-left corner of Figure 4.5 and its three mirror images are used for the mapping of the four distinct 3-level trees containing leaves of the original 5-level tree. The mirror images are employed to minimize the distances between the roots of these trees for connections at the next level. The largest dilation of edges is 2 in this case (the reason for this is that there is no way to directly connect processor-1 and processor-4, or processor-2 and processor-6; we use two edges to connect them together as shown with the bold lines in Figure 4.6).

In general, a large binary tree of level l is viewed as four appropriately connected subtrees of level $l - 2$ for which embeddings into a 2-D HOW system are easily obtained recursively; interconnection of their roots after the embeddings are then easily derived. An example is shown in Figure 4.7. The maximum dilation is two for binary trees with an odd number of levels. Otherwise, we have optimal embeddings.


0
1
2
3
4
5
6
7 8 9 10 11 12 13 14
7
3
8
14
9
4
10
13
0
1
2
6
11
5
12
Building block as shown in figure 4
Another building block for the 3-level binary tree mapping

(a) 4-level binary tree

(b) the mapping onto the 2-D system

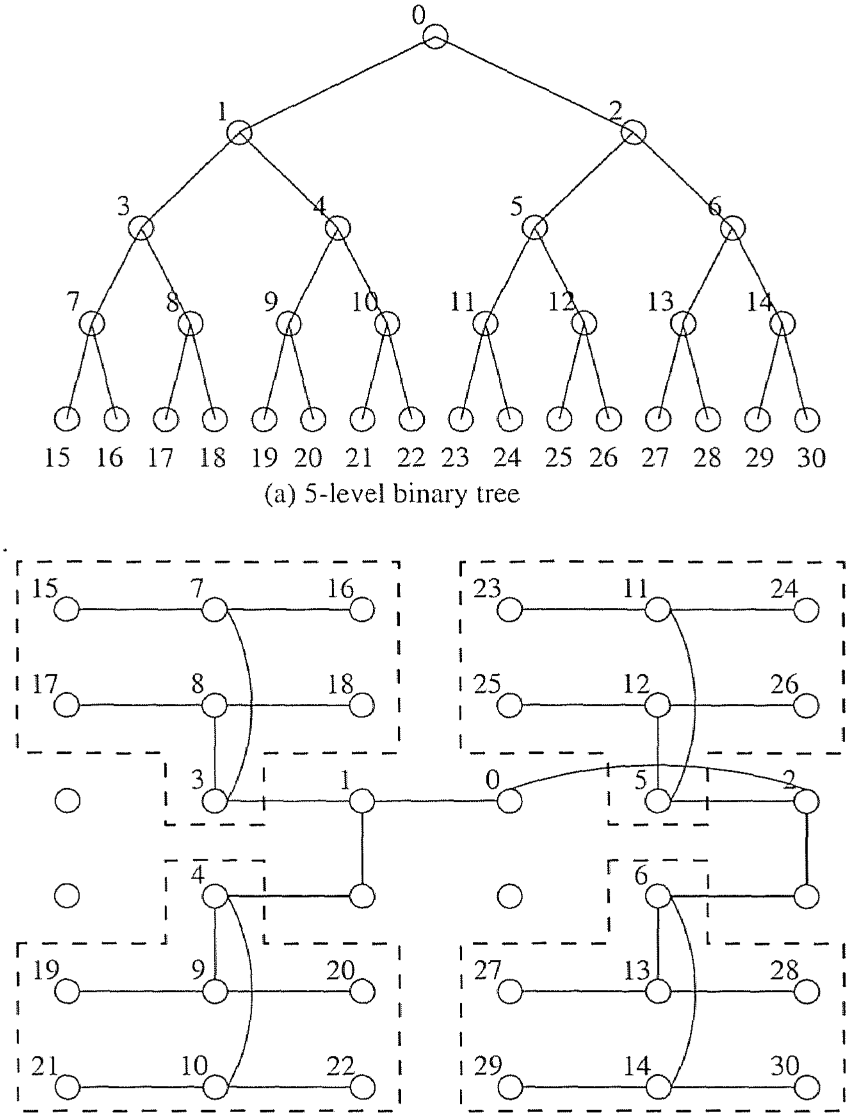
Figure 4.5 Optimal mapping of the 4-level binary tree onto the 2-D HOW(4,2,2) system. The two distinct building blocks for the mapping of 3-level binary trees are enclosed in dotted lines.

4.4 Embedding a Hypercube into a 2-D HOW System

We can embed a (direct binary) hypercube into a 2-D HOW system with two different methods, based on the desired expansion.

First, we consider the embedding of the d-D hypercube into the 2-D HOW system with $2^{\lceil \frac{d}{2} \rceil} \times 2^{\lceil \frac{d}{2} \rceil}$ nodes, corresponding to minimal expansion. We can embed this hypercube recursively as shown in Figures 4.8 and 4.9, where optimal mapping is achieved because of the large windows. This mapping is derived from the classical 2-D representation of hypercubes; optimal mapping results if $w \geq 2^{\lceil \frac{d}{2} \rceil - 1}$. In the general case, the largest dilation of edges for this mapping technique is $\lceil \frac{2^{\lceil \frac{d}{2} \rceil - 1}}{w} \rceil$. The advantage of this method is that it is very simple and easy to implement, but its disadvantage is that half of the processors are wasted when d is an odd number. The maximum congestion is one if w is not a power of two. If $w = 2^v$, with $v < \lceil \frac{d}{2} \rceil - 1$, the maximum congestion for the mapping that minimizes the maximum dilation is $(\lceil \frac{d}{2} \rceil - 1) - v + 1$ or $\lceil \frac{d}{2} \rceil - v$. The expansion is $\frac{2^{2\lceil \frac{d}{2} \rceil}}{2^d}$.

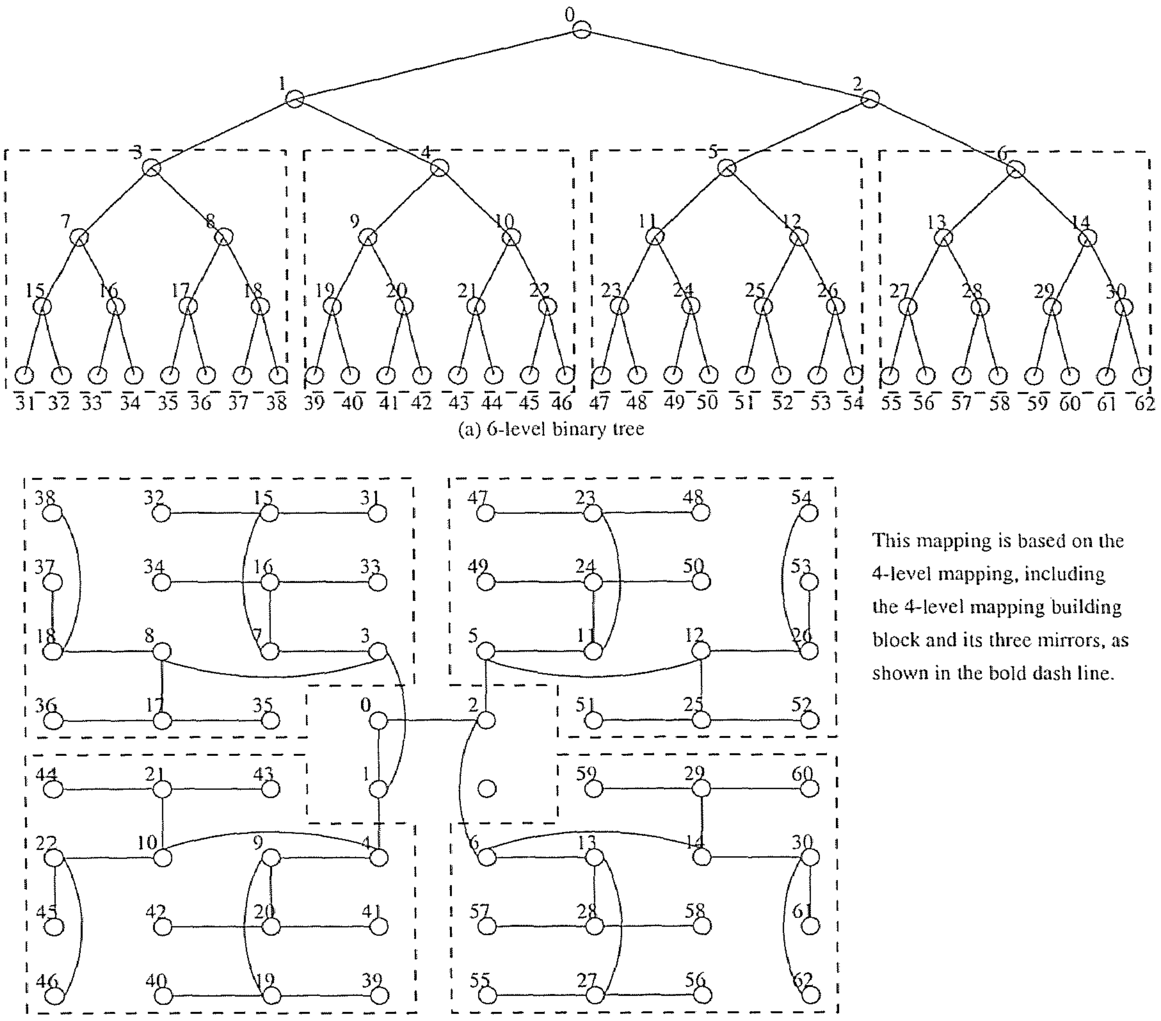
Second, in order to minimize the number of unused processors in the 2-D HOW system if d is odd, we can use another method to embed the d-D hypercube into the



(a) 5-level binary tree

(b) the mapping onto the 2-D system

Figure 4.6 Mapping the 5-level binary tree onto the 2-D HOW(6,2,2) system.

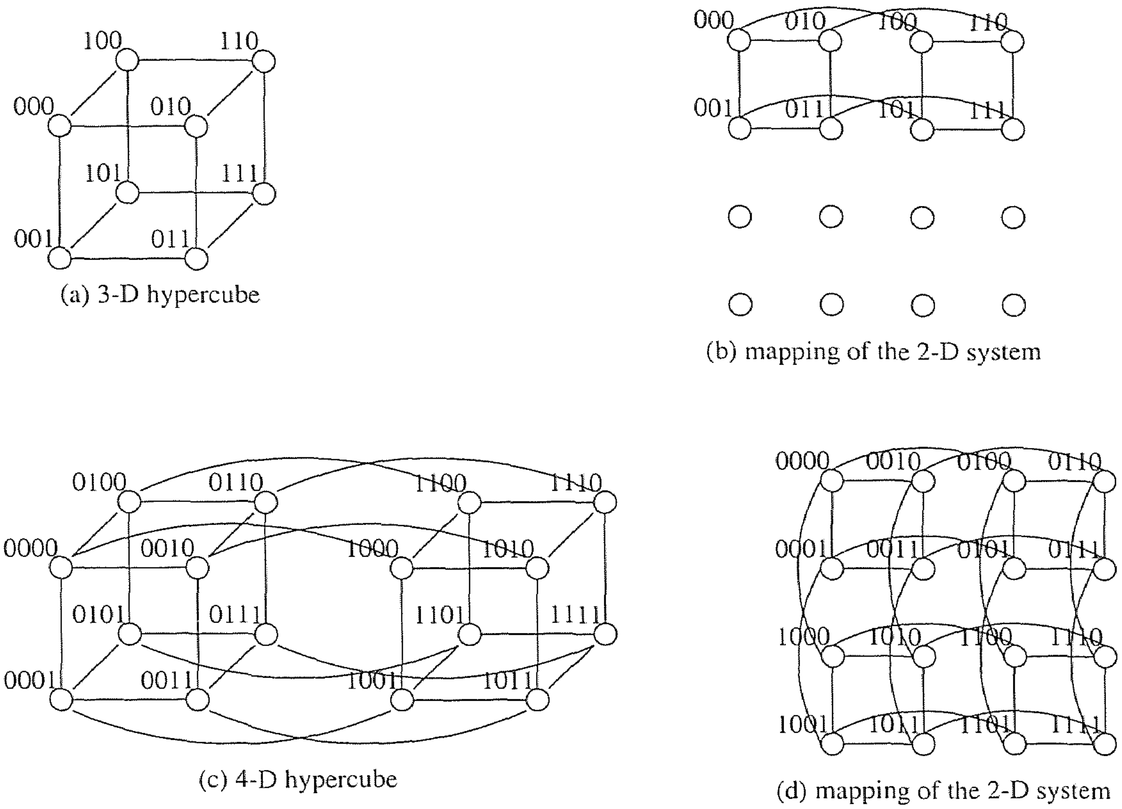


(a) 6-level binary tree

(b) the mapping onto the 2-D system

This mapping is based on the 4-level mapping, including the 4-level mapping building block and its three mirrors, as shown in the bold dash line.

Figure 4.7 Optimal mapping of the 6-level binary tree onto the 2-D HOW(8,2,2) system.



(a) 3-D hypercube

(b) mapping of the 2-D system

(c) 4-D hypercube

(d) mapping of the 2-D system

Figure 4.8 Optimal 3-D and 4-D hypercube embeddings into the HOW(4,2) system (method one).

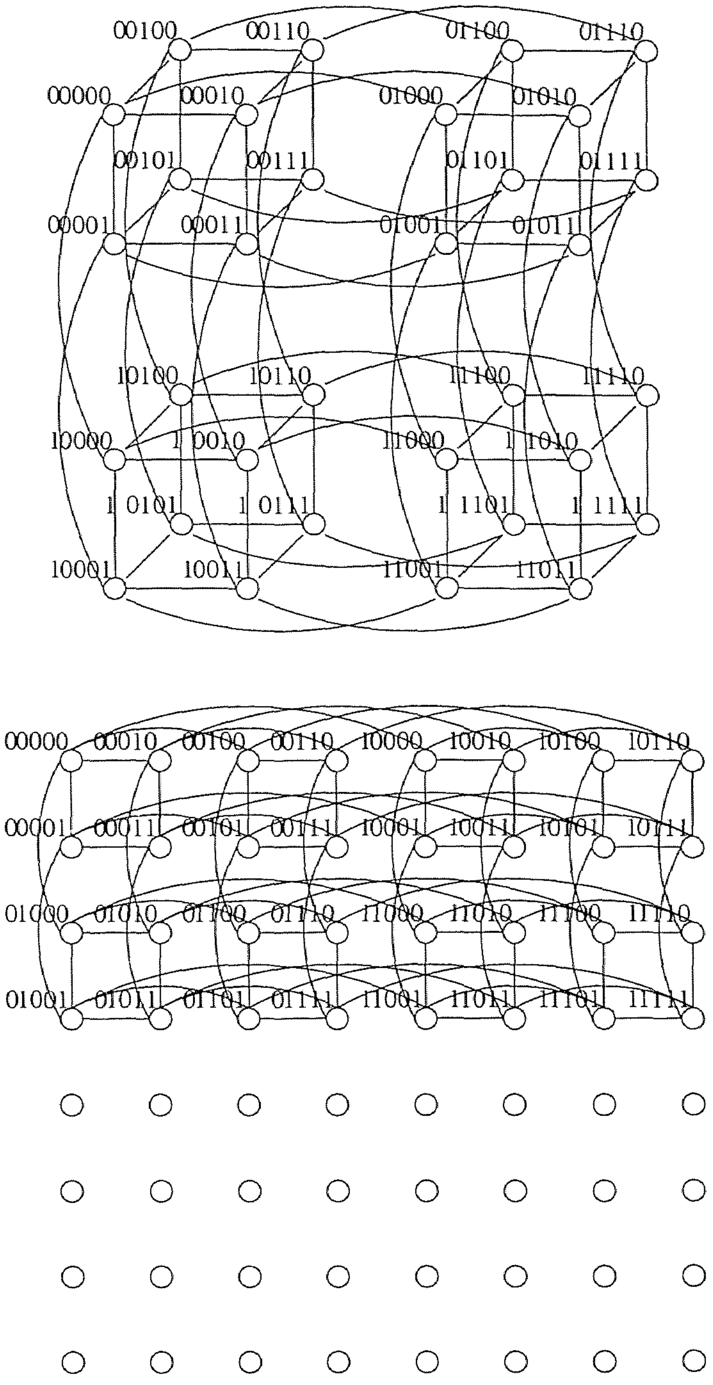


Figure 4.9 5-D hypercube and its embedding into a 2-D HOW system (method-one). Optimal mapping is derived if $w \geq 4$.

2-D HOW(k,w,2) system, where $k = 3 * 2^{\frac{d-3}{2}}$ if d is odd, or $k = 2 * 2^{\frac{d-2}{2}} = 2^{\frac{d}{2}}$ if d is even. This recursive mapping method is based on the fact that the d-D hypercube is formed from four $(d-2)$-D hypercubes. When d is even, then we use the mapping of 2^2 hypercube as the fundamental building block; when d is odd, then we use the mapping of 2^3 hypercube as the fundamental building block The advantages of this method are that we can save a lot of otherwise unused processors and that the 2-D mapping looks neater when d is an odd number. The embedding of the 3-D hypercube into the "building block" HOW(3,2,2) is used recursively. As shown in Figure 4.10, the embedding into the building block results in only one unused node. The source edges (000,100), (100,101) and (110,111) have dilation two in the building block, and the congestion is two for the target edge (110,100) —- which also means there are 3 edges with dilation two, and there is 1 edge with congestion two.. Figure 4.10 shows the embedding of a 5-D hypercube using four 3-D hypercube building blocks in HOW(3,2,2)s. This example shows that there are only four unused nodes.

In the general case, with the second method for an odd d the chosen target system is the HOW($3 \times 2^{\frac{d-3}{2}}$, w, 2) for the best mapping with minimum expansion. The expansion is actually equal to $\frac{9 \times 2^{d-3}}{2^d}$ or $\frac{9}{8}$.

Proposition 1: *Given the d-D binary hypercube, if d is even, the largest dilation of edges is* $\lceil \frac{2^{\lceil \frac{d}{2} \rceil - 1}}{w} \rceil$ *and the congestion is 1.*

Proposition 2: *If d is odd with method-two, there are* 2^{d-3} *edges with congestion 2, the largest dilation of edges is max(2,* $\lceil \frac{3(\sqrt{2^{d-3}}-1)}{w} \rceil$*), and there are at least* $3 \times 2^{d-3}$ *edges with dilation 2, and* 2^{d-3} *unused nodes.*

Lemma 1: *In the building block (3-D binary hypercube), there are* $(3 \times 2^3/2)$ *or* 12 *edges. Among them there is only one edge (node 110 to node 100) with congestion 2, there are three edges with dilation 2.*

This can be easily proven from in Figure 4.10. Only four unused nodes result with this mapping.

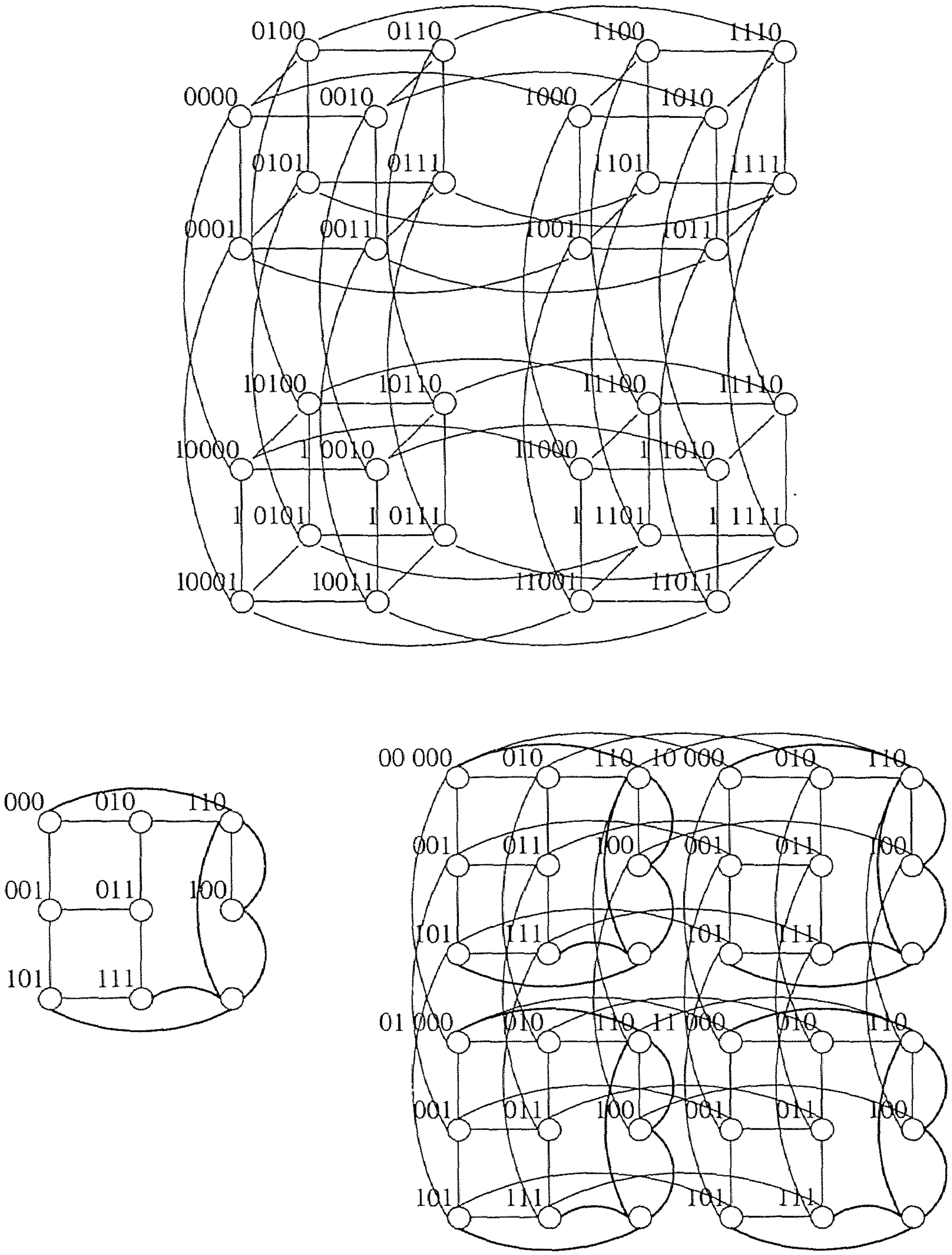


Figure 4.10 5-D hypercube embedding with the second method. This figure shows the original hypercube, the embedding of the 3-D hypercube into the building block HOW(3,2,2), and the final embedding into the 2-D HOW(6,3,2) system.

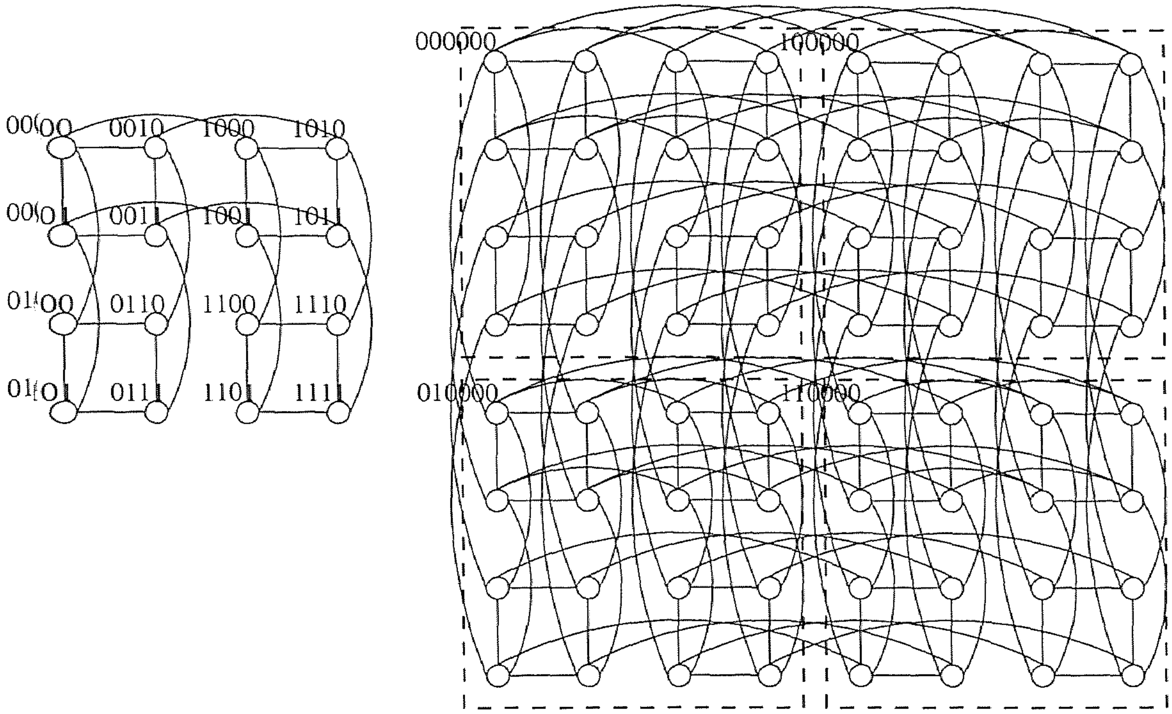


Figure 4.11 6-D hypercube embedding in a 2-D system using method two. (Actually method-two and method-one are the same for even number dimension hypercube.) This figure shows the building block and the embedding in 2-D system.

CHAPTER 5

COMMUNICATION OPERATIONS ON 1-D HOW SYSTEMS

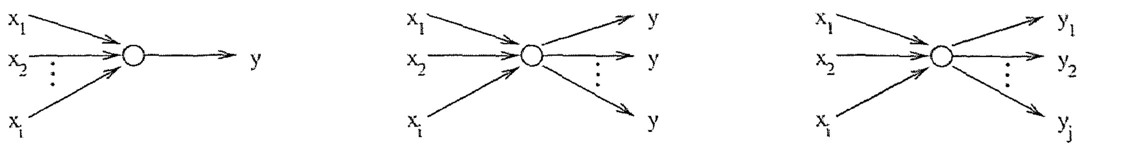
Our focus in this dissertation is 2-D HOW systems. However, 1-D HOW systems are their building blocks (BBs), and therefore we first develop communication routines for 1-D HOW systems. Before we propose algorithms for implementing various communication patterns on 1-D HOW systems, introductory material is needed to facilitate evaluation of the algorithms. The *communication latency*, that is the time consumed to communicate a message between two processors in the system, depends on the following parameters [5] [20]:

- *Startup time* (t_s): the time consumed by the sending processor. It comprises the time to prepare the message (producing the header, trailer, and error correction information), the time for the routing algorithm at the source, and the time to send the first part of the message to the appropriate communication port.

- *Per-word transfer time* (t_w): the time taken by a word to traverse a channel. If the channel bandwidth is b words per second, then each word takes time $t_w = 1/b$.

- *Combining time* (t_c): the time consumed by an intermediate node to switch a message from an input to an output port; it also includes the time to combine incoming messages, if needed, and send them to the appropriate output port.

We calculate only the time taken by a message to reach the input port of the destination. Additional time may be needed to get the data from that port. In **store-and-forward (SF) routing**, with a message traversing a path with multiple links, each intermediate processor forwards the message to the next processor in the path after it has received the entire message. To increase the utilization of communication resources and reduce communication time, **wormhole routing** divides a message



(a) model-1 with one output port

(b) model-2 with multiple ports and the same output value

(c) model-3 with multiple ports and different output values

Figure 5.1 Different output port models.

into **flits (flow-control digits)**. As the header flit advances along the chosen path, the remaining flits follow in the same path in pipelined fashion. If the header flit encounters a channel already in use, this flit is blocked until the channel becomes available [5]. Normally, the flit size coincides with the channel width. The combining time t_c is ignored in wormhole routing.

We develop algorithms under three different communication models. For all of the models, each processor can receive more than one message at a time in different input ports. These models differ in how they can use their output ports.

- **Model-1**: Each processor can use only one output port at a time.
- **Model-2**: Each processor can use multiple output ports simultaneously, as long as all output ports contain the same value.
- **Model-3**: Each processor can use multiple output ports simultaneously, and different output ports can have different values.

The architecture considered here is a 1-D system. There are three different models in a 1-D system which are used for communications, as shown in Figure 5.1.

In the following subsections we develop algorithms for various communication operations on 1-D HOW systems and derive corresponding execution times for the aforementioned models. The analysis is done each time for SF and wormhole routing, in this order. These operations are very frequently used in parallel processing [13] [4].

5.1 One-to-One Communication

This basic operation sends a message from one processor to another.

With SF routing, sending a single message containing m words takes $t_s + mt_w l + t_c(l-1)$ time, where l is the number of links traversed by the message. For a 1-D HOW system with p processors and window size w, l is at most $\lceil\frac{p-1}{w}\rceil$, and therefore the time for a single message transfer has the *upper bound* of

$$T_{one\_to\_one} = t_s + mt_w\lceil\frac{p-1}{w}\rceil + t_c(\lceil\frac{p-1}{w}\rceil - 1) = O(m\frac{p}{w})$$

assuming no contention with other messages at intermediate processors.

With wormhole routing, assume that the flit is one word, and therefore the flit transfer time is t_w. If the message traverses l links, then the header of the message takes $t_s + lt_w$ time to reach the destination. If the message is m words long, then the remaining flits will reach the destination in time $(m-1)t_w$ after the arrival of the header. Therefore, the *upper bound* is

$$T(WR)_{one\_to\_one} = t_s + t_w\lceil\frac{p-1}{w}\rceil + (m-1)t_w = O(m + \frac{p}{w})$$

For the wrap-around $HOW(p, w, 1)$. l is at most $\lceil\frac{p-1}{2w}\rceil$, and therefore it takes half of the time for the regular $HOW(p, w, 1)$ system.

Therefore the time with SF routing for a single message transfer has the *upper bound* of

$$T^{wrap}_{one\_to\_one} = t_s + mt_w\lceil\frac{p-1}{2w}\rceil + t_c(\lceil\frac{p-1}{2w}\rceil - 1) = O(m\frac{p}{w})$$

The time with wormhole routing for a single message transfer has the *upper bound* of

$$T(WR)^{wrap}_{one\_to\_one} = t_s + t_w\lceil\frac{p-1}{2w}\rceil + (m-1)t_w = O(m + \frac{p}{w})$$

5.2 One-to-All Broadcasting

One-to-all broadcasting is an operation where a single processor sends the same data of m words to all other processors. Initially, only the source processor has the data of size m that needs to be broadcast. At the termination of the procedure, there are $p-1$ copies of the initial data, one copy residing in each of the other processors. The naive way to perform one-to-all broadcasting is to sequentially send $p-1$ messages from the source to the other $p-1$ processors. For the sake of efficiency, every processor could keep a copy of the message it receives from a neighbor, and then could forward this message to one or more of its other neighbors.

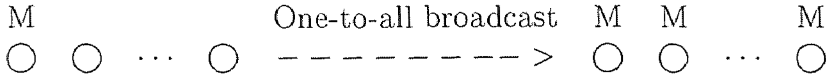


Figure 5.2 One-to-all broadcast.

5.2.1 Model-1

Since there is only one output port "available" for each processor at each transfer step, we consider two different stages. We assume that the leftmost processor is the source, for worst case timing. In the first stage, we copy the data to all processors (PEs) in the source's window of size w In the second stage, the data in the leftmost window is propagated to the right, one window size at a time.

We introduce two parameters here: s_1 represents the number of transfer steps needed to fill the first window, and s_2 represents the number of transfer steps needed in the second stage to copy the values in the first window into the remaining windows. In the first stage, the propagation doubles each time the number of PEs that receive the message, and therefore the processors within the window are assumed to form a binary tree. We have the following relations among s_1,s_2, and w:

$$s_1 = \lceil \log(w+1) \rceil$$
$$s_2 = \lceil (p - 2^{s_1})/w \rceil$$

All logarithms in this dissertation are in the base 2.

Table 5.1 The propagation rules of one-to-all broadcasting under model-1.

S	PE_{max}	PE_{total}
1	$2^{S-1} = 2^{1-1} = 1$	$\sum_{i=1}^{S} 2^{i-1} = 2^S - 1 = 1$
2	$2^{S-1} = 2^{2-1} = 2$	$\sum_{i=1}^{S} 2^{i-1} = 2^S - 1 = 3$
3	$2^{S-1} = 2^{3-1} = 4$	$\sum_{i=1}^{S} 2^{i-1} = 2^S - 1 = 7$
$\vdots$	$\vdots$	$\vdots$
s_1	$2^{S-1} = 2^{s_1-1}$	$\sum_{i=1}^{S} 2^{i-1} = 2^S - 1 = 2^{s_1} - 1$

We can also use Table 5.1 to illustrate the propagation rules to follow in the first stage. (S is the number of the transfer step in the first stage, PE_{max} is the maximum number of PEs that can receive a copy at each transfer step, and PE_{total} is the total number of PEs that have received a copy at each transfer step).

Figure 5.3 shows an example. The communication time for one-to-all broadcasting under model-1 and SF routing has the *upper bound* of

$$T_{one\_to\_all,1} = \begin{cases} t_s + mt_w\lceil\log p\rceil + t_c(\lceil\log p\rceil - 1) & = O(m\log p) \\ & \quad \text{if } (p-1) \le w \\ t_s + mt_w(s_1+s_2) + t_c(s_1+s_2-1) & = O(m\log w + m\frac{p}{w}) \\ & \quad \text{if } (p-1) > w \end{cases}$$

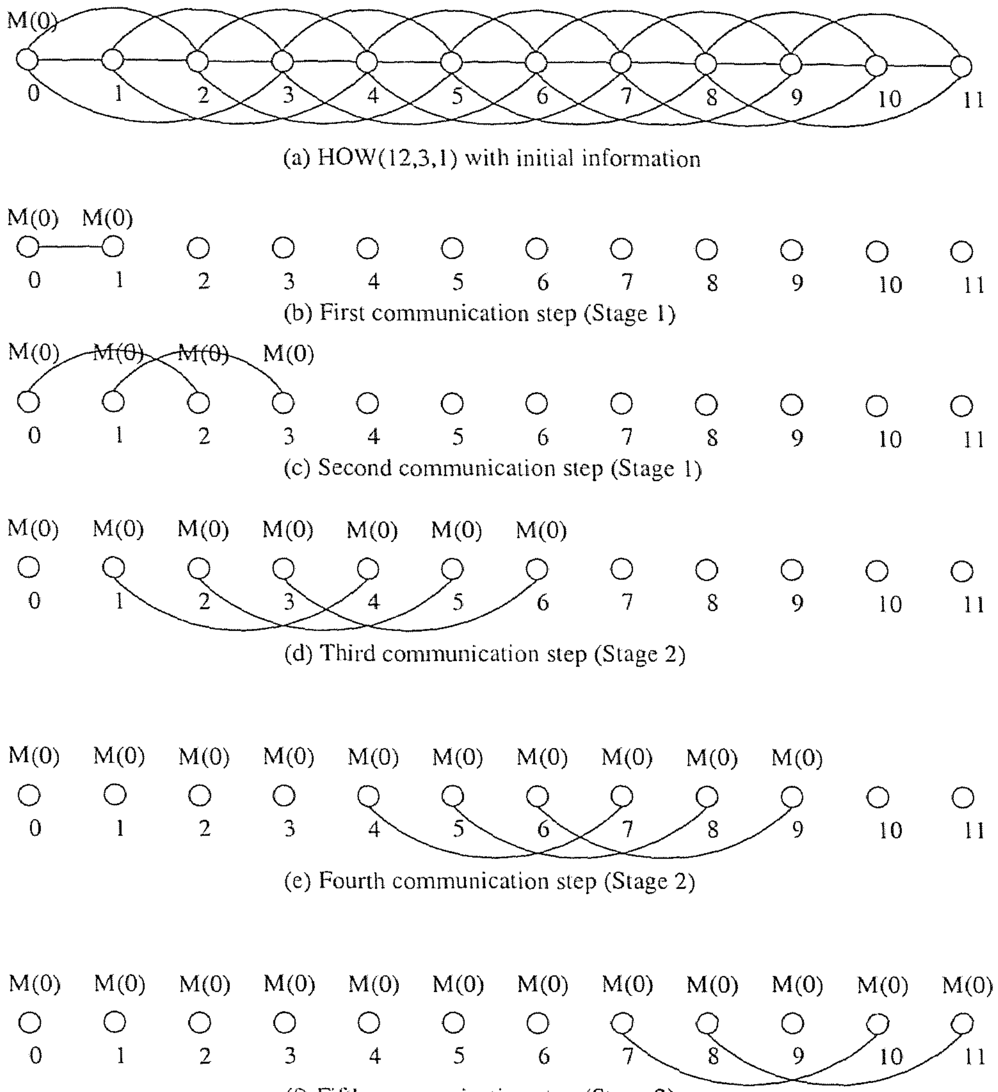
This asymptotic time is optimal.

With wormhole routing, the *upper bound* is

$$T(WR)_{one\_to\_all,1} = \begin{cases} t_s + t_w\lceil\log p\rceil + (m-1)t_w & = O(m+\log p) \\ & \quad \text{if } (p-1) \le w \\ t_s + t_w(s_1+s_2) + (m-1)t_w & = O(m+logw+\frac{p}{w}) \\ & \quad \text{if } (p-1) > w \end{cases}$$

assuming that incoming data can simultaneously be stored locally and also be transferred to the next PE in the path.

For wrap-around $HOW(p,w,1)$. It will need $s_1^{'}$ steps to fill the leftmost window and rightmost windows, which is $2 * s_1$. Also it will need $s_2^{'}$ steps which is only half of s_2 to copy the values in the leftmost and rightmost windows into the remaining



(a) HOW(12,3,1) with initial information

(b) First communication step (Stage 1)

(c) Second communication step (Stage 1)

(d) Third communication step (Stage 2)

(e) Fourth communication step (Stage 2)

(f) Fifth communication step (Stage 2)

Figure 5.3 One-to-all broadcasting under model-1 with 12 processors and window size of 3. A number in parentheses is the label of the source processor from which data has been broadcast. All communication steps are shown.

windows. Therefore,

$$\begin{array}{l} s_1^{'} = 2\lceil \log(w+1) \rceil \\ s_2^{'} = \lceil \frac{p-1-2^{s_1^{'}}}{2w} \rceil \end{array}$$

Therefore, the communication time of the wrap-around $HOW(p, w, 1)$ for one-to-all broadcasting under model-1 and SF routing has the *upper bound* of

$$T^{wrap}_{one\_to\_all,1} = \begin{cases} t_s + mt_w\lceil \log p \rceil + t_c(\lceil \log p \rceil - 1) & = & O(m \log p) \\ & & \text{if } (p-1) \leq w \\ t_s + mt_w(s_1^{'} + s_2^{'}) + t_c(s_1 + s_2^{'} - 1) & = & O(m \log w + m\frac{p}{w}) \\ & & \text{if } (p-1) > w \end{cases}$$

With wormhole routing, the *upper bound* on the communication time is

$$T(WR)^{wrap}_{one\_to\_all,1} = \begin{cases} t_s + t_w\lceil \log p \rceil + (m-1)t_w & = & O(m + \log p) \\ & & \text{if } (p-1) \leq w \\ t_s + t_w(s_1^{'} + s_2^{'}) + (m-1)t_w & = & O(m + logw + \frac{p}{w}) \\ & & \text{if } (p-1) > w \end{cases}$$

Special-case: Fully connected 1-D subsystems. For a fully connected subsystem, the procedure is similar to that for stage-1 under our model-1. Therefore,

$$T^{full}_{one\_to\_all,1} = t_s + mt_w\lceil \log p \rceil + t_c(\lceil \log p \rceil - 1) = O(m \log p)$$

With wormhole routing, the communication time is

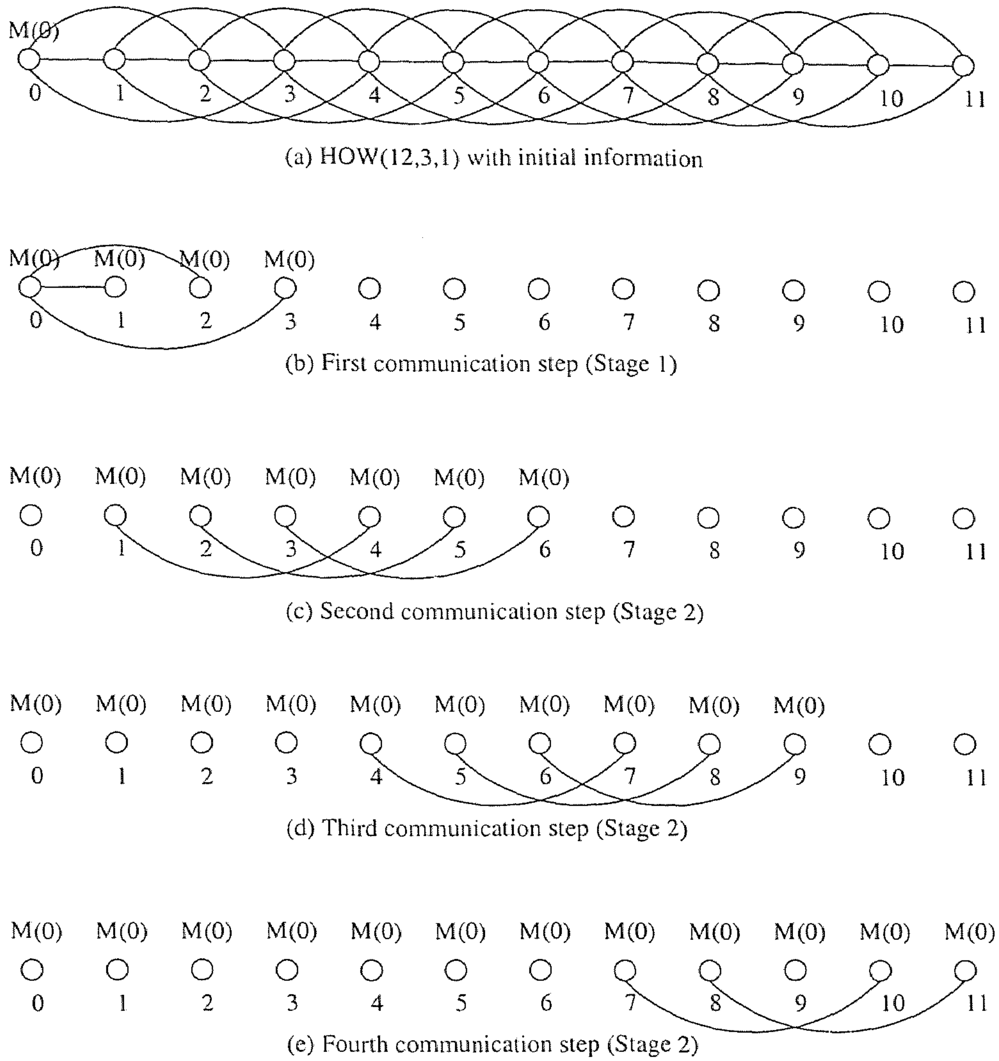
$$T(WR)^{full}_{one\_to\_all,1} = t_s + t_w\lceil \log p \rceil + (m-1)t_w = O(m + \log p)$$

5.2.2 Model-2 and Model-3

For one-to-all broadcasting, there is only one value to be sent, and therefore the procedures for this operation are identical under model-2 and model-3. Assume the leftmost PE as the source. Model-2 is not inferior to model-3 because up to w output ports are "available" to the right of each processor at each transfer step as long as these ports transfer the same value, which is the case here. The first stage now consumes one transfer step and the total number of transfer steps is $\lceil (p-1)/w \rceil$. Figure 5.4 shows an example. The communication time has the *upper bound* of

$$T_{one\_to\_all,2} = t_s + mt_w\lceil \frac{p-1}{w} \rceil + t_c(\lceil \frac{p-1}{w} \rceil - 1) = O(m\frac{p}{w})$$

This asymptotic time is optimal.



(a) HOW(12,3,1) with initial information

(b) First communication step (Stage 1)

(c) Second communication step (Stage 2)

(d) Third communication step (Stage 2)

(e) Fourth communication step (Stage 2)

Figure 5.4 One-to-all broadcasting under model-2 and model-3 with 12 processors and window size of 3. A number in parentheses is the label of the source processor from which data has been broadcast. All communication steps are shown.

With wormhole routing, the *upper bound* is

$$T(WR)_{one\_to\_all,2} = t_s + t_w\lceil\frac{p-1}{w}\rceil + (m-1)t_w = O(m+\frac{p}{w})$$

For the wrap-around $HOW(p, w, 1)$. Every node can be treated similarly, and the communication time is exactly half of that for the regular $HOW(p, w, 1)$ system.

Therefore, the communication time of the wrap-around $HOW(p, w, 1)$ for one-to-all broadcasting under model-2 and model-3 and SF routing has the *upper bound* of

$$T^{wrap}_{one\_to\_all,2} = t_s + mt_w\lceil\frac{p-1}{2w}\rceil + t_c(\lceil\frac{p-1}{2w}\rceil - 1) = O(m\frac{p}{w})$$

With wormhole routing, the *upper bound* on the communication time is

$$T(WR)^{wrap}_{one\_to\_all,2} = t_s + t_w\lceil\frac{p-1}{2w}\rceil + (m-1)t_w = O(m+\frac{p}{w})$$

Special-case: Fully connected 1-D subsystems. It is easy to get the result for the fully connected subsystem; the one-to-all broadcasting just needs one transfer step. Therefore,

$$T^{full}_{one\_to\_all,2} = t_s + mt_w = O(m)$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{one\_to\_all,2} = t_s + mt_w = O(m)$$

5.3 All-to-All Broadcasting

In all-to-all broadcasting, which is a generalization of one-to-all broadcasting, all p processors simultaneously initiate a broadcast. A processor sends the same m-word message to every other processor, but different processors may broadcast different messages.

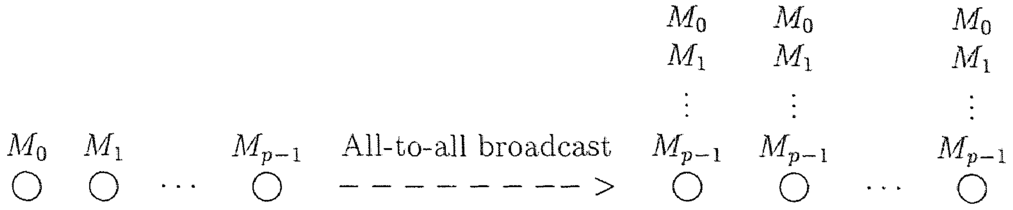


Figure 5.5 All-to-all broadcast.

5.3.1 Model-1

For model-1, there is only one output port of each processor we can use at a time. In order to let every processor pass information to a neighbor in each step, we deliberately choose those channels that form a ring, as shown in Figure 5.6. If communication is performed circularly in a single direction, then each processor receives all $(p-1)$ pieces of information from all other processors in $(p-1)$ steps. The time taken by the entire operation is

$$T_{all\_to\_all,1} = t_s + mt_w(p-1) + t_c(p-2) = O(mp)$$

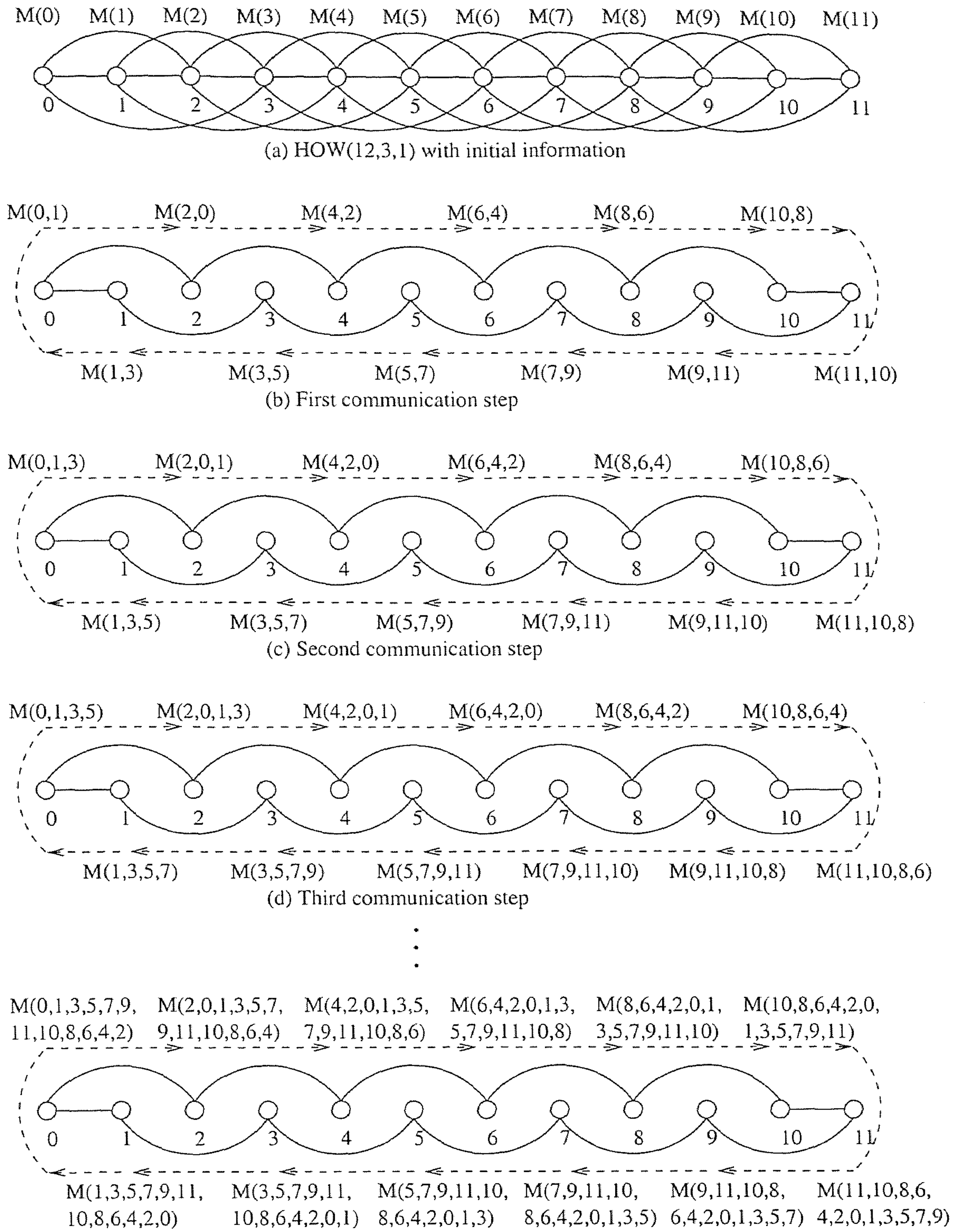
This asymptotic time is optimal because each processor can use only one output port at a time, and therefore each message must make $p-1 = O(p)$ hops.

With wormhole routing, the communication time is

$$T(WR)_{all\_to\_all,1} = t_s + mt_w(p-1) = O(mp)$$

because the header of each message is blocked at each intermediate node until the previous message has completely departed.

For the wrap-around $HOW(p, w, 1)$. Since only one cycle has to be formed in order to pass the information around, the communication time is exactly the same as that for the regular $HOW(p, w, 1)$.


M(0) M(1) M(2) M(3) M(4) M(5) M(6) M(7) M(8) M(9) M(10) M(11)
0 1 2 3 4 5 6 7 8 9 10 11
M(0,1) M(2,0) M(4,2) M(6,4) M(8,6) M(10,8)
M(1,3) M(3,5) M(5,7) M(7,9) M(9,11) M(11,10)
M(0,1,3) M(2,0,1) M(4,2,0) M(6,4,2) M(8,6,4) M(10,8,6)
M(1,3,5) M(3,5,7) M(5,7,9) M(7,9,11) M(9,11,10) M(11,10,8)
M(0,1,3,5) M(2,0,1,3) M(4,2,0,1) M(6,4,2,0) M(8,6,4,2) M(10,8,6,4)
M(1,3,5,7) M(3,5,7,9) M(5,7,9,11) M(7,9,11,10) M(9,11,10,8) M(11,10,8,6)
M(0,1,3,5,7,9, 11,10,8,6,4,2)
M(2,0,1,3,5,7, 9,11,10,8,6,4)
M(4,2,0,1,3,5, 7,9,11,10,8,6)
M(6,4,2,0,1,3, 5,7,9,11,10,8)
M(8,6,4,2,0,1, 3,5,7,9,11,10)
M(10,8,6,4,2,0, 1,3,5,7,9,11)
M(1,3,5,7,9,11, 10,8,6,4,2,0)
M(3,5,7,9,11, 10,8,6,4,2,0,1)
M(5,7,9,11,10, 8,6,4,2,0,1,3)
M(7,9,11,10, 8,6,4,2,0,1,3,5)
M(9,11,10,8, 6,4,2,0,1,3,5,7)
M(11,10,8,6, 4,2,0,1,3,5,7,9)

(a) HOW(12,3,1) with initial information

(b) First communication step

(c) Second communication step

(d) Third communication step

(e) Eleventh communication step

Figure 5.6 All-to-all broadcasting under model-1 with 12 processors and window size of 3. The numbers in parentheses for each processor are the labels of source processors from which data was received prior to the current communication step.

Special-case: Fully connected 1-D subsystems. As for a fully connected 1-D subsystem, no intermediate node will be involved in the broadcasting procedure. The time taken by the entire broadcasting procedure is

$$T^{full}_{all\_to\_all,1} = t_s + mt_w(p-1) = O(mp)$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{all\_to\_all,1} = t_s + mt_w(p-1) = O(mp)$$

5.3.2 Model-2

The broadcasting procedure follows:

- First stage: Each PE sends its message to all of its neighbors.
- Remaining stages: Assume the stage i, where $i = 1, 2, ..., \lceil \frac{p-1}{w} \rceil - 1$. In one direction, beginning from position iw and also involving all its successors, send the messages from the PEs $0, 1, ..., (p-1-iw-1)$ through all possible channels. In the other direction, beginning from position $(p-1-iw)$ and also involving all its predecessors, send the messages from the PEs $p-1, p-2, ..., (iw+1)$. If there is an overlap between these two directions, then split this stage into two steps in order to make sure that every PE sends just one value at a time. From all the messages it contains, each time a PE sends out the message received earlier from its most distant PE.

Table 5.2 shows the detailed steps involved in the broadcasting procedure for 12 *PEs* and a window of size 3. It consumes five steps. Refer to Figure 5.7 for an example. The example in Figure 5.7 is for model-3, and therefore "step" in the table stands for "stage" under model-2. However, the only difference between the two models is in the second transfer step, because there is an overlap between the two

Table 5.2 The detailed steps for all-to-all broadcasting under model-2.

p_0 (0)	p_1 (1)	p_2 (2)	p_3 (3)	p_4 (4)	p_5 (5)	p_6 (6)	p_7 (7)	p_8 (8)	p_9 (9)	p_{10} (10)	p_{11} (11)
1,2,3	0, 2,3,4	0,1, 3,4,5	0,1,2, 4,5,6	1,2,3, 5,6,7	2,3,4, 6,7,8	3,4,5, 7,8,9	4,5,6, 8,9,10	5,6,7, 9,10,11	6,7,8, 10,11	7,8,9, 11	8,9,10
				0	0,1	0,1,2	1,2,3	2,3,4	3,4,5	4,5,6	5,6,7
4,5,6	5,6,7	6,7,8	7,8,9	8,9,10	9,10,11	10,11	11				
7,8,9	8,9,10	9,10,11	10,11	11			0	0,1	0,1,2	1,2,3	2,3,4
10,11	11									0	0,1

directions; therefore, we need to split this "transfer step" into two steps for model-2. The whole procedure consumes five steps under model-2.

The total time taken by this operation is

$$T_{all\_to\_all,2} = t_s + mt_w(\lceil\frac{p-1}{w}\rceil + x) + t_c(\lceil\frac{p-1}{w}\rceil + x - 1)$$

where x is the number of stages needed to be split into two steps, and x should satisfy the condition $xw < p - 1 - xw$. So x is the largest integer less than $\frac{p-1}{2w}$. Therefore,

$$T_{all\_to\_all,2} = O(m\frac{p}{w})$$

This asymptotic time is optimal because the diameter of the system is $O(\frac{p}{w})$.

With wormhole routing, the communication time is

$$T(WR)_{all\_to\_all,2} = t_s + mt_w(\lceil\frac{p-1}{w}\rceil + x) = O(m\frac{p}{w})$$

because of message blocking on reused channels.

For the wrap-around $HOW(p,w,1)$. Every node could be treated similarly, so the number of transfer steps is $\lceil\frac{p-1}{w}\rceil$. Although each node has $2w$ neighbors, we divide w because output ports must transfer the same message. Tables 5.3 and 5.4 show detailed information for this process.

Table 5.3 The detailed steps for all-to-all broadcasting under model-2 using a wrap-around system with 16 processors.

p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13
13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1
3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2
12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11
11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10
10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3
5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4
6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5
9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8
8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6

Therefore, the communication time of the wrap-around $HOW(p, w, 1)$ for one-to-all broadcasting under model-2 and SF routing is

$$T^{wrap}_{all\_to\_all,2} = t_s + mt_w\lceil\frac{p-1}{w}\rceil + t_c(\lceil\frac{p-1}{w}\rceil - 1) = O(m\frac{p}{w})$$

With wormhole routing, the communication time is

$$T(WR)^{wrap}_{all\_to\_all,2} = t_s + mt_w\lceil\frac{p-1}{w}\rceil = O(m\frac{p}{w})$$

Special-case: Fully connected 1-D subsystems. For a fully connected subsystem, only one transfer step is needed to accomplish the broadcasting procedure.

$$T^{full}_{all\_to\_all,2} = t_s + mt_w = O(m)$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{all\_to\_all,2} = t_s + mt_w = O(m)$$

Table 5.4 The detailed steps for all-to-all broadcasting under model-2 using a wraparound system with 17 processors.

p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)
16	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	16	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
14	15	16	0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	0
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	0	1
3	4	5	6	7	8	9	10	11	12	13	14	15	16	0	1	2
13	14	15	16	0	1	2	3	4	5	6	7	8	9	10	11	12
12	13	14	15	16	0	1	2	3	4	5	6	7	8	9	10	11
11	12	13	14	15	16	0	1	2	3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11	12	13	14	15	16	0	1	2	3
5	6	7	8	9	10	11	12	13	14	15	16	0	1	2	3	4
6	7	8	9	10	11	12	13	14	15	16	0	1	2	3	4	5
10	11	12	13	14	15	16	0	1	2	3	4	5	6	7	8	9
9	10	11	12	13	14	15	16	0	1	2	3	4	5	6	7	8
8	9	10	11	12	13	14	15	16	0	1	2	3	4	5	6	7
7	8	9	10	11	12	13	14	15	16	0	1	2	3	4	5	6

Table 5.5 The detailed steps for all-to-all broadcasting under model-3 using a wrap-around system with 16 processors.

p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
14	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13
13	14	15	0	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1
3	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2
12	13	14	15	0	1	2	3	4	5	6	7	8	9	10	11
11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10
10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3
5	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4
6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5
9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8
8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
7	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6

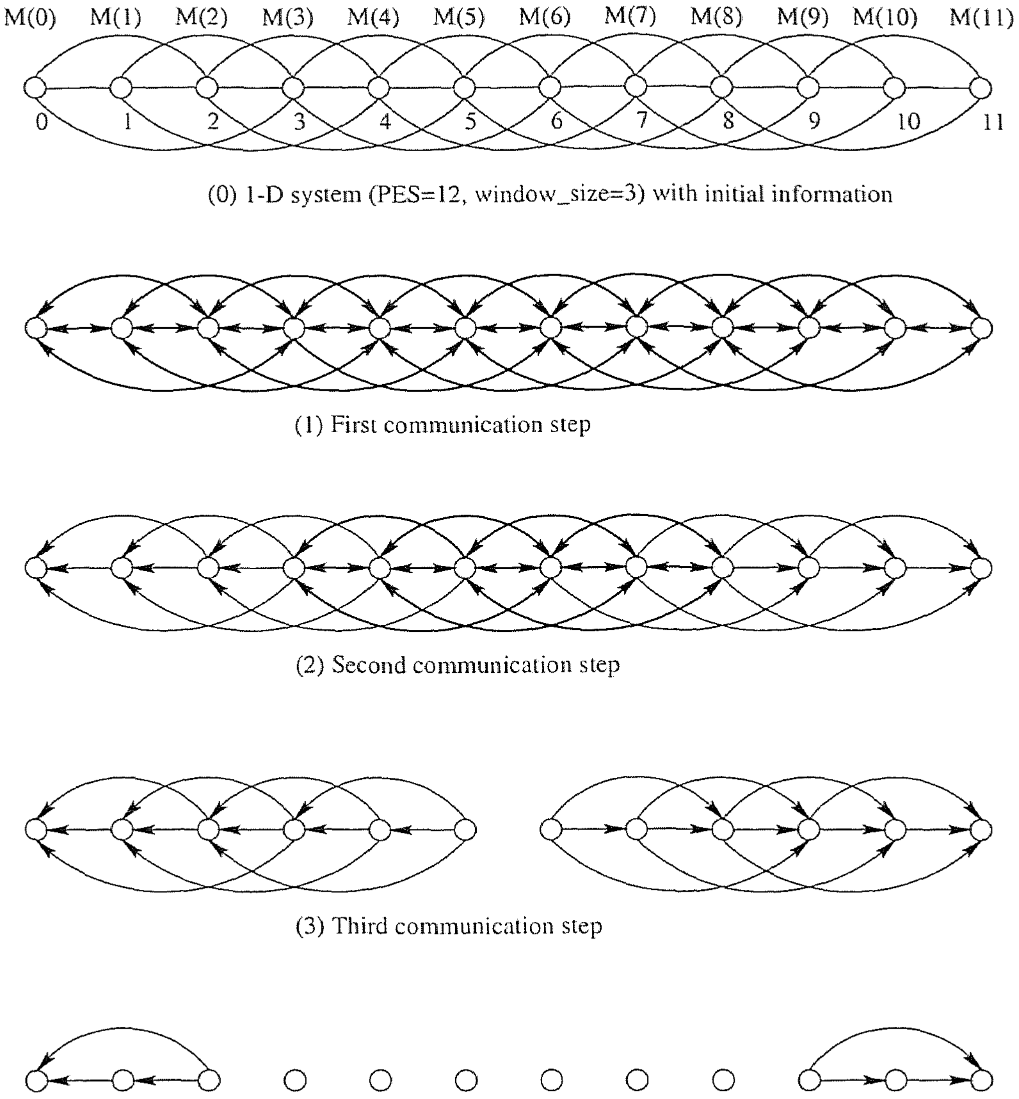
5.3.3 Model-3

This procedure is very similar to that for model-2. Since each individual processor can send different messages at the same time, we do not need to split any step, as shown in the example of Figure 5.7. The total time taken by this operation is optimal and given by

$$T_{all\_to\_all,3} = t_s + mt_w\lceil\frac{p-1}{w}\rceil + t_c(\lceil\frac{p-1}{w}\rceil - 1) = O(m\frac{p}{w})$$

With wormhole routing, the communication time is

$$T(WR)_{all\_to\_all,3} = t_s + mt_w\lceil\frac{p-1}{w}\rceil = O(m\frac{p}{w})$$



(0) 1-D system (PES=12, window_size=3) with initial information

(1) First communication step

(2) Second communication step

(3) Third communication step

(4) Fourth communication step

p_0 (0)	p_1 (1)	p_2 (2)	p_3 (3)	p_4 (4)	p_5 (5)	p_6 (6)	p_7 (7)	p_8 (8)	p_9 (9)	p_{10} (10)	p_{11} (11)
1,2,3	0, 2,3,4	0,1, 3,4,5	0,1,2, 4,5,6	1,2,3, 5,6,7	2,3,4, 6,7,8	3,4,5, 7,8,9	4,5,6, 8,9,10	5,6,7, 9,10,11	6,7,8, 10,11	7,8,9, 11	8,9,10
				0	0,1	0,1,2	1,2,3	2,3,4	3,4,5	4,5,6	5,6,7
4,5,6	5,6,7	6,7,8	7,8,9	8,9,10	9,10,11	10,11	11				
7,8,9	8,9,10	9,10,11	10,11	11			0	0,1	0,1,2	1,2,3	2,3,4
10,11	11									0	0,1

Figure 5.7 All-to-all broadcasting under model-3 with 12 processors and window size of 3. Addresses of processors from which values have been received at the end of each step are shown.

Table 5.6: The detailed steps for all-to-all broadcasting under model-3 using a wrap-around system with 17 processors.

p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)
16	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	16	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
14	15	16	0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	0
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	0	1
3	4	5	6	7	8	9	10	11	12	13	14	15	16	0	1	2
13	14	15	16	0	1	2	3	4	5	6	7	8	9	10	11	12
12	13	14	15	16	0	1	2	3	4	5	6	7	8	9	10	11
11	12	13	14	15	16	0	1	2	3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11	12	13	14	15	16	0	1	2	3
5	6	7	8	9	10	11	12	13	14	15	16	0	1	2	3	4
6	7	8	9	10	11	12	13	14	15	16	0	1	2	3	4	5
10	11	12	13	14	15	16	0	1	2	3	4	5	6	7	8	9
9	10	11	12	13	14	15	16	0	1	2	3	4	5	6	7	8
8	9	10	11	12	13	14	15	16	0	1	2	3	4	5	6	7
7	8	9	10	11	12	13	14	15	16	0	1	2	3	4	5	6

For the wrap-around $HOW(p, w, 1)$. The number of transfer steps is $\lceil \frac{p-1}{2w} \rceil$. Tables 5.5 and 5.6 show detailed information for this process.

Therefore, the communication time of the wrap-around $HOW(p, w, 1)$ for one-to-all broadcasting under model-3 and SF routing is

$$T^{wrap}_{all\_to\_all,3} = t_s + mt_w \lceil \frac{p-1}{2w} \rceil + t_c(\lceil \frac{p-1}{2w} \rceil - 1) = O(m\frac{p}{w})$$

With wormhole routing, the communication time is

$$T(WR)^{wrap}_{all\_to\_all,3} = t_s + mt_w \lceil \frac{p-1}{2w} \rceil = O(m\frac{p}{w})$$

Special-case: Fully connected 1-D subsystems. For a fully connected 1-D subsystem, the whole broadcasting procedure just needs a single transfer step.

$$T^{full}_{all\_to\_all,3} = t_s + mt_w = O(m)$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{all\_to\_all,3} = t_s + mt_w = O(m)$$

5.4 One-to-All Personalized Communication

One-to-all personalized communication is an operation where the source processor sends $(p - 1)$ unique messages, each one destined for a different processor in the system. Unlike one-to-all broadcasting, one-to-all personalized communication does not involve any duplication of data. However, the communication patterns for one-to-all broadcasting and one-to-all personalized communication are identical; only the sizes and contents of messages are different.

5.4.1 Model-1 and Model-2

Even though under model-2 each processor has multiple outports available in each step, all the outports are supposed to transport the same message. But for one-to-all personalized communication, the source processor has different messages to be

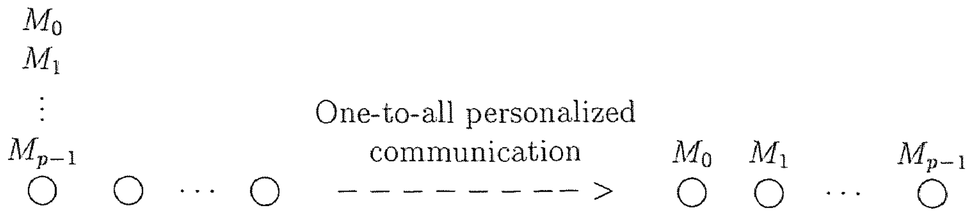


Figure 5.8 One-to-all personalized communication.

transmitted. In this case, the communication procedures are exactly the same for both model-1 and model-2. For these two models, no matter what the window size is, it will take $(p-1)$ transfer steps for this communication operation. A ring structure is used to communicate values, as shown in Figure 5.6. Messages going farther have higher priority of transmission. The total time taken by this operation is

$$T_{one\_to\_all\_pers,1} = t_s + mt_w(p-1) + t_c(\lceil\frac{p-1}{2}\rceil - 1) = O(mp)$$

This is similar to the asymptotic time consumed by the source, and therefore it is optimal. The shortest paths in the ring are chosen to reach respective destinations. For the sake of simplicity, assume that the source is p_0. To reach the PE p_x, where $1 \leq x \leq (p-1)$, the message makes $\lceil\frac{x}{2}\rceil$ hops. Assume that the source first sends out the messages destined for the odd-numbered PEs. It then transmits messages to the even-numbered PEs. Assume for the second case the PE p_x with $x = 2y$. This PE will receive its message with delay $t_c(y-1) + mt_w(y-1)$ after it was transmitted by the source. The time left for the source to complete the entire operation is $mt_w(y-1)$, because $(y-1)$ is the number of messages still to be transmitted. Therefore, the "combining time" term used in the equation is for the worst case, where $y = \lceil\frac{p-1}{2}\rceil$.

With wormhole routing, the total number of flits to be transferred by the source is $(p-1)m$. Messages going farther have higher priority of transmission. The communication time is

$$T(WR)_{one\_to\_all\_pers,1} = t_s + mt_w(p-1) = O(mp)$$

This also represents the time consumed by the source because of the pipelining of messages and the chosen priority for message transmission.

Special-case: Fully connected 1-D subsystems. Referring to the previous case, we know that even under a fully connected 1-D subsystem, we still need $(p-1)$ transfer steps. The total time taken by this operation is

$$T^{full}_{one\_to\_all\_pers,1} = t_s + mt_w(p-1) = O(mp)$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{one\_to\_all\_pers,1} = t_s + mt_w(p-1) = O(mp)$$

5.4.2 Model-3

Under model-3, the one-to-all personalized communication operation can be done as follows. For the worst case, we assume p_0 to be the source:

- First, the processor p_0 passes the w most distant messages to its w neighbors, so that a destination processor with higher address gets a message for a higher-addressed processor.

- Second, the processor p_0 similarly passes the next w most distant messages to its window, while all processors that received an intermediate message earlier pass that message to their neighbor at distance w in the next window (i.e., window to their right).

- The second step repeats until all processors receive their own message.

  Table 5.7 shows a complete example for 12 processors and window size of 3. The total time taken by this operation is

$$T_{one\_to\_all\_pers,3} = t_s + mt_w\lceil\frac{p-1}{w}\rceil + t_c(\lceil\frac{p-1}{w}\rceil - 1) = O(m\frac{p}{w})$$

which has the same asymptotic complexity with the time consumed by the source, and therefore it is optimal.

Table 5.7 The detailed steps for one-to-all personalized communication under model-3.

p_0 m_{0--11}	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}
m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8	m_9	m_{10}	m_{11}								
m_0, m_1, m_2, m_3, m_4, m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}					
m_0, m_1, m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}		
m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}

With wormhole routing, all processors receive their messages simultaneously in time $t_s + mt_w\lceil\frac{p-1}{w}\rceil$, because of message pipelining and message blocking resulting from the m-flit messages. Therefore, the total communication time is

$$T(WR)_{one\_to\_all\_pers,3} = t_s + m\lceil\frac{p-1}{w}\rceil t_w = O(m\frac{p}{w})$$

which is again optimal because it is identical to the time consumed by the source with peak utilization of its communication ports and no data duplication.

Special-case: Fully connected 1-D subsystems. For a fully connected 1-D subsystem, the entire communication operation needs just a single transfer step. Therefore,

$$T^{full}_{one\_to\_all\_pers,3} = t_s + mt_w = O(m)$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{one\_to\_all\_pers,3} = t_s + mt_w = O(m)$$

5.5 All-to-All Personalized Communication

In all-to-all personalized communication, also known as *total exchange*, each processor sends a distinct message of size m to every other processor. Unlike all-to-all broadcasting, all-to-all personalized communication does not involve any duplication of data.

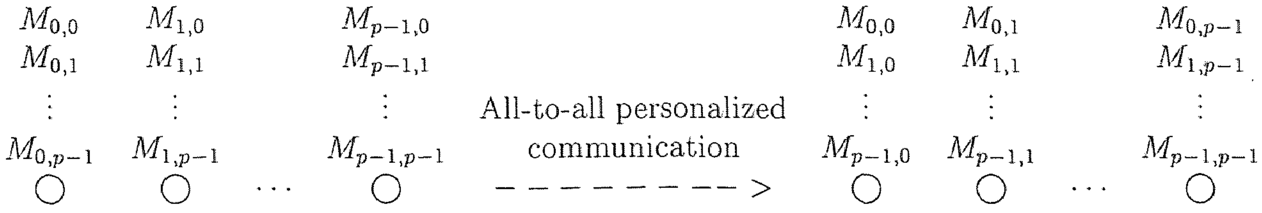

$M_{0,0}$ $M_{0,1}$ ⋮ $M_{0,p-1}$
$M_{1,0}$ $M_{1,1}$ ⋮ $M_{1,p-1}$
$M_{p-1,0}$ $M_{p-1,1}$ ⋮ $M_{p-1,p-1}$
All-to-all personalized communication
$M_{0,0}$ $M_{1,0}$ ⋮ $M_{p-1,0}$
$M_{0,1}$ $M_{1,1}$ ⋮ $M_{p-1,1}$
$M_{0,p-1}$ $M_{1,p-1}$ ⋮ $M_{p-1,p-1}$

Figure 5.9 All-to-all personalized communication.

5.5.1 Model-1 and Model-2

For all-to-all personalized communication, the source processor has different messages to be transmitted. Although model-2 has multiple outports available, all the outports are supposed to transport the same message. Therefore, the communication procedures are exactly the same for both model-1 and model-2.

We form a ring here, as in Figure 5.6. In each transfer step every processor transfers the m-word message destined for its farthest remaining processor. If only one direction in the ring is used for all transfers, then the total number of transfer steps is equal to $\sum_{i=1}^{p-1}(p-i) = \sum_{i=1}^{p-1} i = \frac{(p-1)p}{2}$. The total time taken by this operation is

$$\begin{aligned} T_{all\_to\_all\_pers,1} &= t_s + \sum_{i=1}^{p-1} mt_w(p-i) + \sum_{i=1}^{p-1} t_c(p-i-1) \\ &= t_s + mt_w\frac{(p-1)p}{2} + t_c\frac{(p-1)(p-2)}{2} \\ &= O(mp^2) \end{aligned}$$

However, for the shortest paths, and therefore for smaller communication time, both directions in the ring should be used. In this case, there are $\lceil\frac{p-1}{2}\rceil$ "large" communication stages. In the i-th "large" stage, where $i = 1, 2, ..., \lceil\frac{p-1}{2}\rceil$, each processor transmits the respective messages to the processors at the same distance i to its left and to its right, exclusively in this order. If p is even, then the $\lceil\frac{p-1}{2}\rceil$-th "large" stage implements transmissions in only one of the two directions in the ring. Therefore, the total number of transfer steps to neighbors is equal to

$$
\begin{aligned}
2\sum_{i=1}^{\lceil\frac{p-1}{2}\rceil} i-(\lceil\frac{p-1}{2}\rceil-\lfloor\frac{p-1}{2}\rfloor) &= 2\frac{1}{2}\lceil\frac{p-1}{2}\rceil(\lceil\frac{p-1}{2}\rceil+1)-(\lceil\frac{p-1}{2}\rceil-\lfloor\frac{p-1}{2}\rfloor) \\
&= \lceil\frac{p-1}{2}\rceil^2+\lfloor\frac{p-1}{2}\rfloor
\end{aligned}
$$

The total time is

$$
\begin{aligned}
T_{all\_to\_all\_pers,1} &= t_s+mt_w(\lceil\frac{p-1}{2}\rceil^2+\lfloor\frac{p-1}{2}\rfloor)+2\sum_{i=0}^{\lceil\frac{p-1}{2}\rceil-1} t_c\, i-(\lceil\frac{p-1}{2}\rceil-\lfloor\frac{p-1}{2}\rfloor)t_c \\
&= t_s+mt_w(\lceil\frac{p-1}{2}\rceil^2+\lfloor\frac{p-1}{2}\rfloor)+t_c(\lceil\frac{p-1}{2}\rceil^2-2\lceil\frac{p-1}{2}\rceil+\lfloor\frac{p-1}{2}\rfloor) \\
&= O(mp^2)
\end{aligned}
$$

which is asymptotically optimal because each processor sends out $O(p)$ messages of m words each, and the average distance traveled is $O(p)$.

With wormhole routing, the communication time is

$$
\begin{aligned}
T(WR)_{all\_to\_all\_pers,1} &= t_s+2\sum_{i=1}^{\lceil\frac{p-1}{2}\rceil} m\, t_w\, i-mt_w(\lceil\frac{p-1}{2}\rceil-\lfloor\frac{p-1}{2}\rfloor) \\
&= t_s+mt_w(\lceil\frac{p-1}{2}\rceil^2+\lfloor\frac{p-1}{2}\rfloor) \\
&= O(mp^2)
\end{aligned}
$$

Special-case: Fully connected 1-D subsystems. For a fully connected 1-D subsystem, all the processors use one port at a time to send a single message, and therefore the entire communication operation needs $(p-1)$ steps.

$$T^{full}_{all\_to\_all\_pers,1}=t_s+mt_w(p-1)=O(mp)$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{all\_to\_all\_pers,1}=t_s+mt_w(p-1)=O(mp)$$

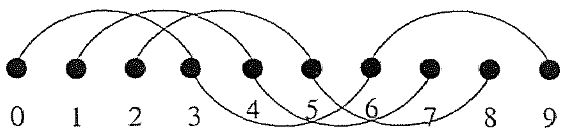


Figure 5.10 Chosen linear arrays in the $HOW(10, 3, 1)$ for all-to-all personalized communication.

5.5.2 Model-3

The all-to-all personalized communication operation involves a lot of message transfers. We will not necessarily derive the most efficient procedure here, because such a procedure can be of a very complex nature. We present a simple procedure that comprises two stages. The basic idea is to use the largest possible number of linear arrays for pipelined message transfers, with the smallest possible number of nodes per such array. Figure 5.10 shows the chosen linear arrays in the $HOW(10, 3, 1)$.

- First stage: this is the initialization stage where local transfers are employed to move messages to processors that belong to the aforementioned linear arrays. Every processor passes all relative messages to neighbors in its window(s). For a given destination message, it passes that message to its neighbor that belongs to a linear array containing that destination; if two such neighbors exist, the one closer to the destination is chosen. It takes up to $s_1 = \lceil \frac{p-1}{w} \rceil$ cycles to finish the initialization, which is the same as the maximum number of values to be sent from a processor to another one.

- Second stage: the linear arrays are used to transfer the values. There are w linear arrays to be used. We need up to $s_2 = \lceil \frac{p-1}{w} \rceil - 1$ cycles to finish the broadcasting along the linear arrays, which is the same as the maximum number of values a processor has to send in a single dimension; messages going farther have higher priority.

The total time taken by this operation is

$$T_{all\_to\_all\_pers,3} = t_s + m(s_1 + s_2)t_w + m(s_1 + s_2 - 1)t_c$$

$$\begin{aligned} &= t_s + 2\, mt_w \lceil \frac{p-1}{w} \rceil + mt_c(2\, \lceil \frac{p-1}{w} \rceil - 1) \\ &= O(m\frac{p}{w}) \end{aligned}$$

An example with 10 processors and window size equal to 3 is shown in the following tables:

With wormhole routing, the communication time is

$$T(WR)_{all\_to\_all\_pers,3} = t_s + 2\, mt_w \lceil \frac{p-1}{w} \rceil = O(m\frac{p}{w})$$

Special-case: Fully connected 1-D subsystems. For a fully connected 1-D subsystem, all the processors use all output ports sending different destined messages to their destination in one single step. The total time taken by the operation is

$$T^{full}_{all\_to\_all\_pers,3} = t_s + mt_w = O(m)$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{all\_to\_all\_pers,3} = t_s + mt_w = O(m)$$

Table 5.8 The detailed steps for all-to-all personalized communication in 1-D HOW under model-3.

Initial state with all information									
p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
{0,0}	{1,0}	{2,0}	{3,0}	{4,0}	{5,0}	{6,0}	{7,0}	{8,0}	{9,0}
{0,1}	{1,1}	{2,1}	{3,1}	{4,1}	{5,1}	{6,1}	{7,1}	{8,1}	{9,1}
{0,2}	{1,2}	{2,2}	{3,2}	{4,2}	{5,2}	{6,2}	{7,2}	{8,2}	{9,2}
{0,3}	{1,3}	{2,3}	{3,3}	{4,3}	{5,3}	{6,3}	{7,3}	{8,3}	{9,3}
{0,4}	{1,4}	{2,4}	{3,4}	{4,4}	{5,4}	{6,4}	{7,4}	{8,4}	{9,4}
{0,5}	{1,5}	{2,5}	{3,5}	{4,5}	{5,5}	{6,5}	{7,5}	{8,5}	{9,5}
{0,6}	{1,6}	{2,6}	{3,6}	{4,6}	{5,6}	{6,6}	{7,6}	{8,6}	{9,6}
{0,7}	{1,7}	{2,7}	{3,7}	{4,7}	{5,7}	{6,7}	{7,7}	{8,7}	{9,7}
{0,8}	{1,8}	{2,8}	{3,8}	{4,8}	{5,8}	{6,8}	{7,8}	{8,8}	{9,8}
{0,9}	{1,9}	{2,9}	{3,9}	{4,9}	{5,9}	{6,9}	{7,9}	{8,9}	{9,9}
Step-1: exchanging information with all connected neighbors.									
p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
{0,0}	{1,1}	{2,2}	{3,3}	{4,4}	{5,5}	{6,6}	{7,7}	{8,8}	{9,9}
right	{0,1}	{0,2}	{0,3}						
		{1,2}	{1,3}	{1,4}					
			{2,3}	{2,4}	{2,5}				
				{3,4}	{3,5}	{3,6}			
					{4,5}	{4,6}	{4,7}		
						{5,6}	{5,7}	{5,8}	
							{6,7}	{6,8}	{6,9}
								{7,8}	{7,9}
									{8,9}
left						{9,0}	{9,1}	{9,2}	
					{8,0}	{8,1}	{8,2}		
				{7,0}	{7,1}	{7,2}			
			{6,0}	{6,1}	{6,2}				
		{5,0}	{5,1}	{5,2}					
	{4,0}	{4,1}	{4,2}						
{3,0}	{3,1}	{3,2}							
{2,0}	{2,1}								
{1,0}									
{0,4}	{1,5}	{2,6}	{3,7}	{4,0}	{5,0}	{6,0}	{7,0}	{8,0}	{9,0}
{0,5}	{1,6}	{2,7}	{3,8}	{4,8}	{5,1}	{6,1}	{7,1}	{8,1}	{9,1}
{0,6}	{1,7}	{2,8}	{3,9}	{4,9}	{5,9}	{6,2}	{7,2}	{8,2}	{9,2}
{0,7}	{1,8}	{2,9}					{7,3}	{8,3}	{9,3}
{0,8}	{1,9}							{8,4}	{9,4}
{0,9}									{9,5}

Table 5.9 The detailed steps for all-to-all personalized communication in 1-D HOW under model-3.(continue-1)

Step-2: transferring the farthest messages through all connected neighbors.									
p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
{0,0}	{1,1}	{2,2}	{3,3}	{4,4}	{5,5}	{6,6}	{7,7}	{8,8}	{9,9}
{1,0}	{0,1}	{0,2}	{0,3}	{1,4}	{2,5}	{3,6}	{4,7}	{5,8}	{6,9}
{2,0}	{2,1}	{1,2}	{1,3}	{2,4}	{3,5}	{4,6}	{5,7}	{6,8}	{7,9}
{3,0}	{3,1}	{3,2}	{2,3}	{3,4}	{4,5}	{5,6}	{6,7}	{7,8}	{8,9}
	{4,1}	{4,2}	{4,3}	{5,4}	{6,5}	{7,6}	{8,7}	{9,8}	
		{5,2}	{5,3}	{6,4}	{7,5}	{8,6}	{9,7}		
right	{0,7}	{0,8}	{0,9}						
		{1,7}	{1,8}	{1,9}					
			{2,7}	{2,8}	{2,9}				
				{3,7}	{3,8}	{3,9}			
						{4,8}	{4,9}		
								{5,9}	
left						{9,0}	{9,1}	{9,2}	
					{8,0}	{8,1}	{8,2}		
				{7,0}	{7,1}	{7,2}			
			{6,0}	{6,1}	{6,2}				
		{5,0}	{5,1}						
	{4,0}								
{0,4}	{1,5}	{2,6}					{7,3}	{8,3}	{9,3}
{0,5}	{1,6}							{8,4}	{9,4}
{0,6}									{9,5}

Table 5.10 The detailed steps for all-to-all personalized communication in 1-D HOW under model-3. (continue-2)

Step-3: intermediate step to transfer information.									
p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
{0,0}	{1,1}	{2,2}	{3,3}	{4,4}	{5,5}	{6,6}	{7,7}	{8,8}	{9,9}
{1,0}	{0,1}	{0,2}	{0,3}	{1,4}	{2,5}	{3,6}	{4,7}	{5,8}	{6,9}
{2,0}	{2,1}	{1,2}	{1,3}	{2,4}	{3,5}	{4,6}	{5,7}	{6,8}	{7,9}
{3,0}	{3,1}	{3,2}	{2,3}	{3,4}	{4,5}	{5,6}	{6,7}	{7,8}	{8,9}
	{4,1}	{4,2}	{4,3}	{5,4}	{6,5}	{7,6}	{8,7}	{9,8}	
		{5,2}	{5,3}	{6,4}	{7,5}	{8,6}	{9,7}		
			{6,3}	{7,4}	{8,5}	{9,6}			
right	{0,4}	{0,5}	{0,6}						
		{1,5}	{1,6}	{0,7}					
			{2,6}	{1,7}	{0,8}				
				{2,7}	{1,8}	{0,9}			
					{3,7}	{2,8}	{1,9}		
							{3,8}	{2,9}	
								{4,8}	{3,9}
									{4,9}
									{5,9}
left						{9,3}	{9,4}	{9,5}	
					{9,2}	{8,3}	{8,4}		
				{9,1}	{8,2}	{7,3}			
			{9,0}	{8,1}	{7,2}				
		{8,0}	{7,1}	{6,2}					
	{7,0}	{6,1}							
{6,0}	{5,1}								
{5,0}									
{4,0}									

Table 5.11 The detailed steps for all-to-all personalized communication in 1-D HOW under model-3. (continue-3)

Step-4: intermediate step to transfer information.									
p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
{0,0}	{1,1}	{2,2}	{3,3}	{4,4}	{5,5}	{6,6}	{7,7}	{8,8}	{9,9}
{1,0}	{0,1}	{0,2}	{0,3}	{1,4}	{2,5}	{3,6}	{4,7}	{5,8}	{6,9}
{2,0}	{2,1}	{1,2}	{1,3}	{2,4}	{3,5}	{4,6}	{5,7}	{6,8}	{7,9}
{3,0}	{3,1}	{3,2}	{2,3}	{3,4}	{4,5}	{5,6}	{6,7}	{7,8}	{8,9}
	{4,1}	{4,2}	{4,3}	{5,4}	{6,5}	{7,6}	{8,7}	{9,8}	
		{5,2}	{5,3}	{6,4}	{7,5}	{8,6}	{9,7}		
			{6,3}	{7,4}	{8,5}	{9,6}			
{4,0}	{5,1}							{4,8}	{5,9}
{5,0}									{4,9}
{6,0}									{3,9}
right		{}	{}	{0,4}					
			{}	{1,5}	{0,5}				
				{2,6}	{1,6}	{0,6}			
					{2,7}	{1,7}	{0,7}		
						{1,8}	{3,7}	{0,8}*	
							{}	{2,8}	{0,9}
								{3,8}	{1,9}
									{2,9}
left					{9,5}	{}	{}		
				{9,4}	{8,4}	{}			
			{9,3}	{8,3}	{7,3}				
		{9,2}	{8,2}	{7,2}					
	{9,1}	{6,2}	{8,1}*						
{9,0}	{7,1}	{}							
{8,0}	{6,1}								
{7,0}									

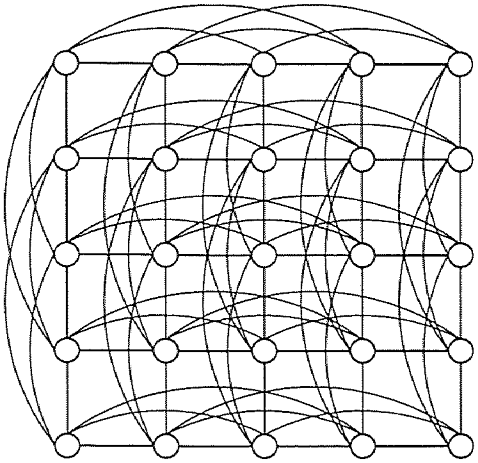
Table 5.12 The detailed steps for all-to-all personalized communication in 1-D HOW under model-3. (continue-4)

Step-5: intermediate step to transfer information.									
p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
{0,0}	{1,1}	{2,2}	{3,3}	{4,4}	{5,5}	{6,6}	{7,7}	{8,8}	{9,9}
{1,0}	{0,1}	{0,2}	{0,3}	{1,4}	{2,5}	{3,6}	{4,7}	{5,8}	{6,9}
{2,0}	{2,1}	{1,2}	{1,3}	{2,4}	{3,5}	{4,6}	{5,7}	{6,8}	{7,9}
{3,0}	{3,1}	{3,2}	{2,3}	{3,4}	{4,5}	{5,6}	{6,7}	{7,8}	{8,9}
	{4,1}	{4,2}	{4,3}	{5,4}	{6,5}	{7,6}	{8,7}	{9,8}	
		{5,2}	{5,3}	{6,4}	{7,5}	{8,6}	{9,7}		
			{6,3}	{7,4}	{8,5}	{9,6}			
{4,0}	{5,1}							{4,8}	{5,9}
{5,0}									{4,9}
{6,0}									{3,9}
{7,0}	{6,1}	{6,2}	{7,3}	{0,4}	{0,5}	{0,6}	{0,7}	{0,8}	{0,9}
{8,0}	{7,1}	{7,2}		{8,4}	{9,5}		{3,7}	{2,8}	{1,9}
{9,0}	{8,1}							{3,8}	{2,9}
right					{1,5}	{2,6}			
						{1,6}	{2,7}		
							{1,7}	{1,8}	
left			{7,3}	{8,4}					
		{7,2}	{8,3}						
	{8,1}	{8,2}							

CHAPTER 6

COMMUNICATION OPERATIONS ON 2-D HOW SYSTEMS

Assume symmetric 2-D HOW systems with p processors. The numbers for rows and columns are then $0, 1, \cdots, \sqrt{p} - 1$. For example, Figure 6.1 shows the processor addresses in the 2-D system $HOW(5, 3, 2)$.



p_{00}	p_{01}	p_{02}	p_{03}	p_{04}
p_{10}	p_{11}	p_{12}	p_{13}	p_{14}
p_{20}	p_{21}	p_{22}	p_{23}	p_{24}
p_{30}	p_{31}	p_{32}	p_{33}	p_{34}
p_{40}	p_{41}	p_{42}	p_{43}	p_{44}

Figure 6.1 Processor addresses in the $HOW(5, 3, 2)$.

6.1 One-to-One Communication

We assume, without loss of generality, that p_{00} is the source processor and that the destination is at distance l.

With SF routing, sending a single message containing m words takes $t_s + mt_w l + t_c(l-1)$ time, where l is the number of links traversed by the message. For a 2-D HOW system with a total of p processors (having $\sqrt{p}$ rows and $\sqrt{p}$ columns) and window size w, l is at most $2\lceil\frac{\sqrt{p}-1}{w}\rceil$, and therefore the time for a single message transfer has the *upper bound* of

$$T_{one\_to\_one} = t_s + 2mt_w\lceil\frac{\sqrt{p}-1}{w}\rceil + t_c(2\lceil\frac{\sqrt{p}-1}{w}\rceil - 1) = O(m\frac{\sqrt{p}}{w})$$

assuming no contention with other messages at intermediate processors.

With wormhole routing, for a single message transfer on the 2-D HOW system the *upper bound* is

$$T(WR)_{one\_to\_one} = t_s + 2t_w\lceil\frac{\sqrt{p}-1}{w}\rceil + (m-1)t_w = O(m\frac{\sqrt{p}}{w})$$

6.2 One-to-All Broadcasting

6.2.1 Model-1

For the best possible performance, we first have to determine which of the row or column window the source belongs to is closer to the center of that row or column, respectively. If it is the row window, then the source broadcasts within that row, and this is followed by broadcasting from those row PEs into all columns. Otherwise, we begin with column broadcasting. However, here we assume the worst case, where the source PE is in the first window of the corresponding 1-D HOW row and column subsystems. Using the same notations as for the 1-D HOW system, s_1 represents the number of transfer steps needed to fill the first window in this row and s_2 represents the number of transfer steps needed in the second stage to copy the values from the first window into the remaining windows of this row. We already know the following relations among s_1,s_2, and w

$$\begin{aligned} s_1 &= \lceil\log(w+1)\rceil \\ s_2 &= \lceil(\sqrt{p}-2^{s_1})/w\rceil \end{aligned}$$

This operation is done by first broadcasting within the aforementioned row and then from that row within all the columns. The communication time under model-1 with SF routing has the *upper bound*

$$T_{one\_to\_all,1} = \begin{cases} t_s + 2mt_w\lceil\log\sqrt{p}\rceil + t_c(2\lceil\log\sqrt{p}\rceil - 1) & = & O(m\log\sqrt{p}) \\ & & \text{if } (\sqrt{p}-1) \le w \\ t_s + 2mt_w(s_1+s_2) + t_c(2(s_1+s_2)-1) & = & O(m\log w + m\frac{\sqrt{p}}{w}) \\ & & \text{if } (\sqrt{p}-1) > w \end{cases}$$

With wormhole routing, the *upper bound* is

$$T(WR)_{one\_to\_all,1} = \begin{cases} t_s + 2t_w\lceil\log\sqrt{p}\rceil + (m-1)t_w = & O(m+\log\sqrt{p}) \\ & \text{if } (\sqrt{p}-1) \leq w \\ t_s + 2t_w(s_1+s_2) + (m-1)t_w = & O(m+\log w + \frac{\sqrt{p}}{w}) \\ & \text{if } (\sqrt{p}-1) > w \end{cases}$$

assuming that incoming data can be stored locally and can simultaneously be transferred to the next PE in the path.

Special-case: Fully connected 1-D subsystems. For fully connected subsystems that form a 2-D generalized hypercube, the procedure is similar to that for $(\sqrt{p}-1) = w$ under model-1.

$$T^{full}_{one\_to\_all,1} = t_s + 2mt_w\lceil\log\sqrt{p}\rceil + t_c(2\lceil\log\sqrt{p}\rceil - 1) = O(m\log\sqrt{p})$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{one\_to\_all,1} = t_s + 2t_w\lceil\log\sqrt{p}\rceil + (m-1)t_w = O(m+\log\sqrt{p})$$

6.2.2 Model-2 and Model-3

For the one-to-all broadcasting operation, there is only one value to be sent, and therefore the whole procedure for model-3 is exactly the same as that for model-2. Figure 6.2 shows two different methods used for one-to-all broadcasting. The numbers of communication steps for the two methods are the same. However, method (b) is easier to program, because it is an extension of the respective method for the 1-D HOW system. This method first broadcasts within the row and then within all columns. The *upper bound* on the total time taken by this operation is

$$T_{one\_to\_all,2} = t_s + 2mt_w\lceil\frac{\sqrt{p}-1}{w}\rceil + t_c(2\lceil\frac{\sqrt{p}-1}{w}\rceil - 1) = O(m\frac{\sqrt{p}}{w})$$

With wormhole routing, the *upper bound* is

$$T(WR)_{one\_to\_all,2} = t_s + 2t_w\lceil\frac{\sqrt{p}-1}{w}\rceil + (m-1)t_w = O(m+\frac{\sqrt{p}}{w})$$

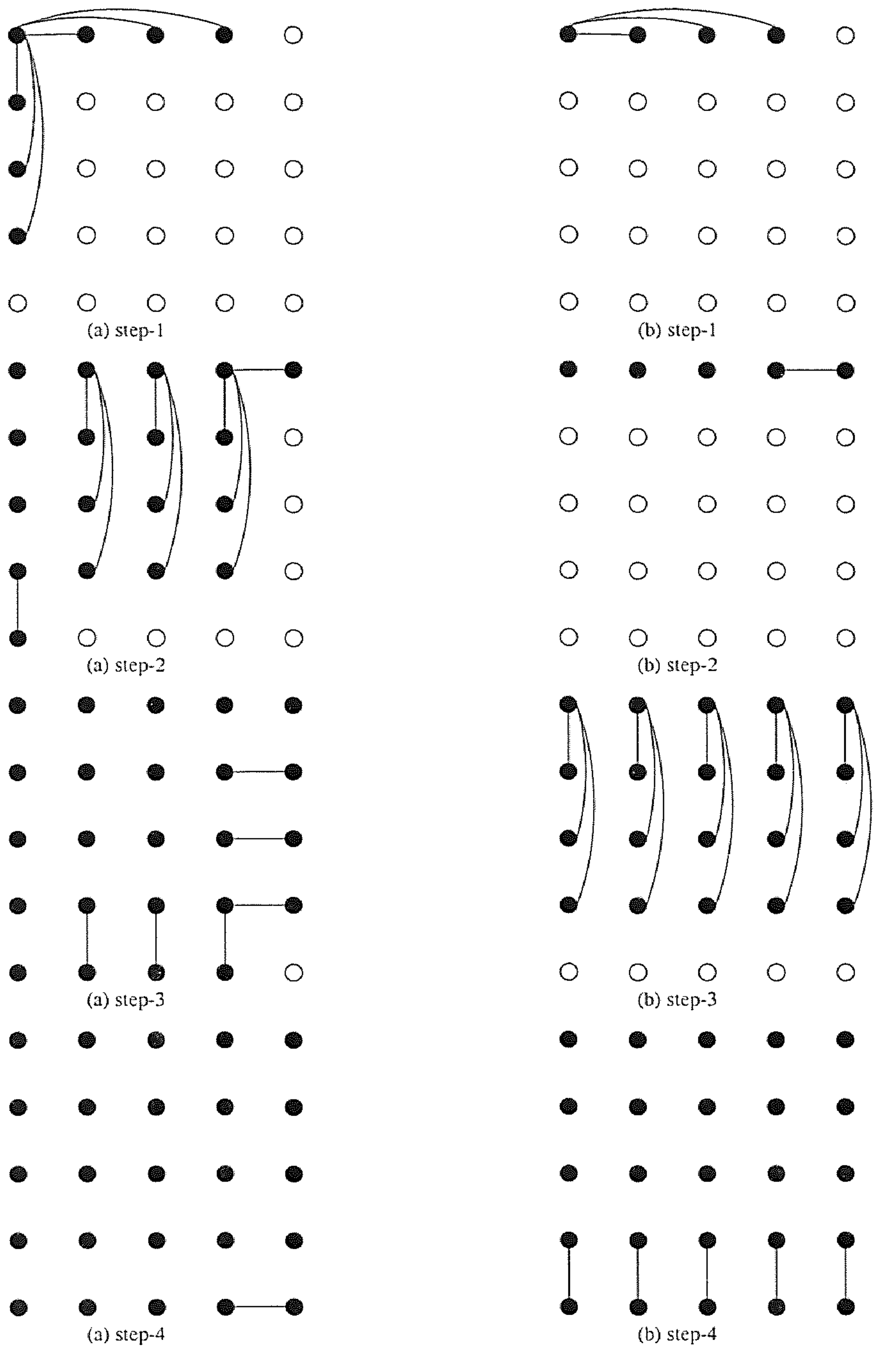


Figure 6.2 One-to-all broadcasting under model-2 and model-3 with two different methods, both of which have the same number of communication steps. A filled circle means that the current processor has already received the message broadcast by the source. All communication steps are shown here. We assume that w=3. For the worst case, we assume p_{00} to be the source.

assuming that the dimension to be traversed is changed just after the first flit is received.

Special-case: Fully connected 1-D subsystems. It is easy to see that for fully connected 1-D subsystems, one-to-all broadcasting needs just two transfer steps. Therefore,

$$T^{full}_{one\_to\_all,2} = t_s + 2mt_w + t_c = O(m)$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{one\_to\_all,2} = t_s + 2t_w + (m-1)t_w = t_s + (m+1)t_w = O(m)$$

6.3 All-to-All Broadcasting

The following table 6.1 shows the initial message state and the required final state for all-to-all broadcasting in a 5x5 system.

Table 6.1 The initial and final state of HOW(5,3,2).

Initial state of HOW(5,3,2)					Required final state				
$m_{0,0}$	$m_{0,1}$	$m_{0,2}$	$m_{0,3}$	$m_{0,4}$	MM	MM	MM	MM	MM
$m_{1,0}$	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$m_{1,4}$	MM	MM	MM	MM	MM
$m_{2,0}$	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$	$m_{2,4}$	MM	MM	MM	MM	MM
$m_{3,0}$	$m_{3,1}$	$m_{3,2}$	$m_{3,3}$	$m_{3,4}$	MM	MM	MM	MM	MM
$m_{4,0}$	$m_{4,1}$	$m_{4,2}$	$m_{4,3}$	$m_{4,4}$	MM	MM	MM	MM	MM

where each processor receives messages from all other processors, and therefore

$$MM = \left\{ \begin{array}{lllll} m_{0,0} & m_{0,1} & m_{0,2} & m_{0,3} & m_{0,4} \\ m_{1,0} & m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} \\ m_{2,0} & m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} \\ m_{3,0} & m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} \\ m_{4,0} & m_{4,1} & m_{4,2} & m_{4,3} & m_{4,4} \end{array} \right.$$

The procedure repeats many times the corresponding procedure for the 1-D HOW system. That is, processors first exchange messages along rows, so that each processor has $\sqrt{p}$ messages at the end for the processors on its own column.

Then, processors exchange their $\sqrt{p}$ messages along columns by repeating the same procedure $\sqrt{p}$ times within the columns.

6.3.1 Model-1

For model-1, there is only one output port of each processor we can use at a time. In order to let every processor pass some information to a neighbor, we deliberately choose some channels to form a ring on each row/column. We assume pipelining of messages along rows and columns.

We start with all-to-all row broadcasting that takes time $t_s + \tau = t_s + (\sqrt{p} - 1)mt_w + t_c(\sqrt{p} - 2)$, as derived for the 1-D HOW system in Subsection 3.1.1. The $\sqrt{p}$ column broadcasts then take time $\sqrt{p}\tau$, because all-to-all 1-D HOW broadcasting is repeated $\sqrt{p}$ times. The time taken by the entire operation is

$$\begin{aligned} T_{all\_to\_all,1} &= t_s + (1 + \sqrt{p})mt_w(\sqrt{p} - 1) + (1 + \sqrt{p})t_c(\sqrt{p} - 2) + t_c \\ &= t_s + (p - 1)mt_w + (p - \sqrt{p} - 1)t_c = O(mp) \end{aligned}$$

The last t_c term is for switching from row broadcasting into column broadcasting. This asymptotic time is optimal because each processor can use only one output port at a time, and therefore each message will make $O(p)$ hops to visit all $O(p)$ processors.

With wormhole routing, within each row the entire time is $t_s + m(\sqrt{p} - 1)t_w$, assuming the formation of a ring. This is because each processor starts receiving flits with the first data transfer, pipelining of messages is applied, and the total number of flits each processor receives is $m(\sqrt{p} - 1)$. Similarly, for columns the time is $m\sqrt{p}(\sqrt{p} - 1)t_w$. The total time is

$$T(WR)_{all\_to\_all,1} = t_s + m(1 + \sqrt{p})(\sqrt{p} - 1)t_w = t_s + m(p - 1)t_w = O(mp)$$

Special-case: Fully connected 1-D subsystems. As for the 1-D subsystem, there is one t_c that will be involved in the broadcasting procedure within the row and the column. The time taken by the entire broadcasting procedure is

$$T^{full}_{all\_to\_all,1} = t_s + 2mt_w(\sqrt{p} - 1) + t_c = O(m\sqrt{p})$$

assuming again two steps (row-wise and column-wise steps) in the implementation.

With wormhole routing, the communication time is still

$$T(WR)^{full}_{all\_to\_all,1} = t_s + m(\sqrt{p} + 1)t_w = O(m\sqrt{p})$$

6.3.2 Model-2

Based on the algorithm proposed for the 1-D HOW system, the total time taken by this operation is

$$T_{all\_to\_all,2} = t_s + (1+\sqrt{p})mt_w(\lceil \frac{\sqrt{p}-1}{w} \rceil + x) + t_c(1+\sqrt{p})(\lceil \frac{\sqrt{p}-1}{w} \rceil + x - 1) = O(m\frac{p}{w})$$

where x is the largest integer less than $\frac{\sqrt{p}-1}{2w}$. The algorithm for the 1-D HOW system is used $(1 + \sqrt{p})$ times, once for the rows and $\sqrt{p}$ times for the columns.

With wormhole routing, the communication time is

$$T(WR)_{all\_to\_all,2} = t_s + 2mt_w(\lceil \frac{\sqrt{p}-1}{w} \rceil + x)(1 + \sqrt{p}) = O(m\frac{p}{w})$$

Special-case: Fully connected 1-D subsystems. For the 1-D subsystem, only two transfer steps are needed to accomplish broadcasting. Therefore,

$$T^{full}_{all\_to\_all,2} = t_s + (1 + \sqrt{p})mt_w = O(m\sqrt{p})$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{all\_to\_all,2} = t_s + (1 + \sqrt{p})mt_w = O(m\sqrt{p})$$

Table 6.2 Messages received in the first two detailed steps for all-to-all broadcasting within the rows of the $HOW(5,3,2)$ system.

Initial state					Step 1					Step 2				
m_{00}	m_{01}	m_{02}	m_{03}	m_{04}	m_{01} m_{02} m_{03}	m_{00} m_{02} m_{03} m_{04}	m_{00} m_{01} m_{03} m_{04}	m_{00} m_{01} m_{02} m_{04}	m_{01} m_{02} m_{03}	m_{04}				m_{00}.
m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{11} m_{12} m_{13}	m_{10} m_{12} m_{13} m_{14}	m_{10} m_{11} m_{13} m_{14}	m_{10} m_{11} m_{12} m_{14}	m_{11} m_{12} m_{13}	m_{14}				m_{10}
m_{20}	m_{21}	m_{22}	m_{23}	m_{24}	m_{21} m_{22} m_{23}	m_{20} m_{22} m_{23} m_{24}	m_{20} m_{21} m_{23} m_{24}	m_{20} m_{21} m_{22} m_{24}	m_{21} m_{22} m_{23}	m_{24}				m_{20}
m_{30}	m_{31}	m_{32}	m_{33}	m_{34}	m_{31} m_{32} m_{33}	m_{30} m_{32} m_{33} m_{34}	m_{30} m_{31} m_{33} m_{34}	m_{30} m_{31} m_{32} m_{34}	m_{31} m_{32} m_{33}	m_{34}				m_{30}
m_{40}	m_{41}	m_{42}	m_{43}	m_{44}	m_{41} m_{42} m_{43}	m_{40} m_{42} m_{43} m_{44}	m_{40} m_{41} m_{43} m_{44}	m_{40} m_{41} m_{42} m_{44}	m_{41} m_{42} m_{43}	m_{44}				m_{40}

6.3.3 Model-3

Table 6.2 shows the first two steps involving all-to-all broadcasting under model-3. It is very similar to the procedure for model-2. Since each individual processor can send different messages at the same time, we do not need to split any stage. The total time taken by this operation is

$$T_{all\_to\_all,3} = t_s + (1+\sqrt{p})mt_w\lceil\frac{\sqrt{p}-1}{w}\rceil + t_c(1+\sqrt{p})(\lceil\frac{\sqrt{p}-1}{w}\rceil - 1) = O(m\frac{p}{w})$$

With wormhole routing, the communication time is

$$T(WR)_{all\_to\_all,3} = t_s + mt_w\lceil\frac{\sqrt{p}-1}{w}\rceil(1+\sqrt{p}) = O(m\frac{p}{w})$$

Table 6.3 The initial and final states for one-to-all personalized communication in the HOW(p,w,2).

Initial state					Required final state				
$m_{0,0}$ $m_{0,1}$ $m_{0,2}$ $m_{0,3}$ $m_{0,4}$ $m_{1,0}$ $m_{1,1}$ $m_{1,2}$ $m_{1,3}$ $m_{1,4}$ $m_{2,0}$ $m_{2,1}$ $m_{2,2}$ $m_{2,3}$ $m_{2,4}$ $m_{3,0}$ $m_{3,1}$ $m_{3,2}$ $m_{3,3}$ $m_{3,4}$ $m_{4,0}$ $m_{4,1}$ $m_{4,2}$ $m_{4,3}$ $m_{4,4}$					$m_{0,0}$	$m_{0,1}$	$m_{0,2}$	$m_{0,3}$	$m_{0,4}$
					$m_{1,0}$	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$m_{1,4}$
					$m_{2,0}$	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$	$m_{2,4}$
					$m_{3,0}$	$m_{3,1}$	$m_{3,2}$	$m_{3,3}$	$m_{3,4}$
					$m_{4,0}$	$m_{4,1}$	$m_{4,2}$	$m_{4,3}$	$m_{4,4}$

Special-case: Fully connected 2-D subsystems. For the 2-D generalized hypercube, the whole broadcasting procedure needs just two transfer steps. Therefore,

$$T^{full}_{all\_to\_all,3} = t_s + (1 + \sqrt{p})mt_w = O(m\sqrt{p})$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{all\_to\_all,3} = t_s + (1 + \sqrt{p})mt_w = O(m\sqrt{p})$$

6.4 One-to-All Personalized Communication

Table 6.3 shows the initial state and the required final state for one-to-all personalized communication in the 5 × 5 2-D $HOW(5, 3, 2)$ system. We assume, without loss of generality, that p_{00} is the source processor.

6.4.1 Model-1 and Model-2

Because of personalized data, the same procedure is applied for model-1 and model-2. Restricted by the availability of only one output port at a time for each processor, independently of the window size it will take $(\sqrt{p} - 1)$ transfer steps along a row or a column for a processor to send personalized data to all other processors. In the first phase, the source processor, assume p_{00}, passes messages within its row for all processors in the corresponding columns. Messages going farther have higher priority of transmission. This process is implemented as $\sqrt{p}$ one-to-all personalized communications within the row (i.e., 1-D HOW system). At the end of the first phase, each of the first row processors will have $\sqrt{p}$ messages. All $\sqrt{p}$ messages of each first row processor will be transferred in the second phase along the corresponding column applying again one-to-all personalized communication. The total time taken by this operation is

$$\begin{aligned} T_{one\_to\_all\_pers,1} &= t_s + (\sqrt{p}+1)mt_w(\sqrt{p}-1) + t_c(1+\sqrt{p})(\lceil\frac{\sqrt{p}-1}{2}\rceil - 1) \\ &= t_s + (p-1)mt_w + t_c(1+\sqrt{p})(\lceil\frac{\sqrt{p}-1}{2}\rceil - 1) = O(mp) \end{aligned}$$

With wormhole routing, the communication time is

$$T(WR)_{one\_to\_all\_pers,1} = t_s + (1+\sqrt{p})mt_w(\sqrt{p}-1) = t_s + mt_w(p-1) = O(mp)$$

Special-case: Fully connected 1-D subsystems. Referring to the previous case, we know that even under a fully connected 1-D subsystem, we still need $(\sqrt{p}-1)$ transfer steps along each row and each column. The total time taken by this operation is

$$T^{full}_{one\_to\_all\_pers,1} = t_s + (\sqrt{p}+1)mt_w(\sqrt{p}-1) = t_s + (p-1)mt_w = O(mp)$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{one\_to\_all\_pers,1} = t_s + 2t_w(\sqrt{p}-2) + (p-1)mt_w \quad = O(mp)$$

6.4.2 Model-3

We first send the messages that must travel the longest distance using simultaneously all column and row connections. (Note: it is a different method than that used for model-1.) Figure 6.3 shows the exact steps needed for the $HOW(5,3,2)$ system, with p_{00} being the source. The number of message transfer steps is $2\lceil\frac{\sqrt{p}-1}{w}\rceil$, the same as the diameter of the system. The *upper bound* on the total time is

$$T_{one\_to\_all\_pers,3} = t_s + 2\lceil\frac{\sqrt{p}-1}{w}\rceil mt_w(\sqrt{p}-1) + t_c(2\lceil\frac{\sqrt{p}-1}{w}\rceil - 1)(\sqrt{p}-1) = O(m\frac{p}{w})$$

which is optimal.

With wormhole routing, the *upper bound* is

$$T(WR)_{one\_to\_all\_pers,3} = t_s + mt_w 2\lceil\frac{\sqrt{p}-1}{w}\rceil(\sqrt{p}-1) = O(m\frac{p}{w})$$

Special-case: Fully connected 1-D subsystems. For the fully connected 1-D subsystem, the whole communication operation needs just two transfer steps. Therefore,

$$T^{full}_{one\_to\_all\_pers,3} = t_s + (1 + (\sqrt{p}-1))mt_w = O(m\sqrt{p})$$

With wormhole routing, the communication time is

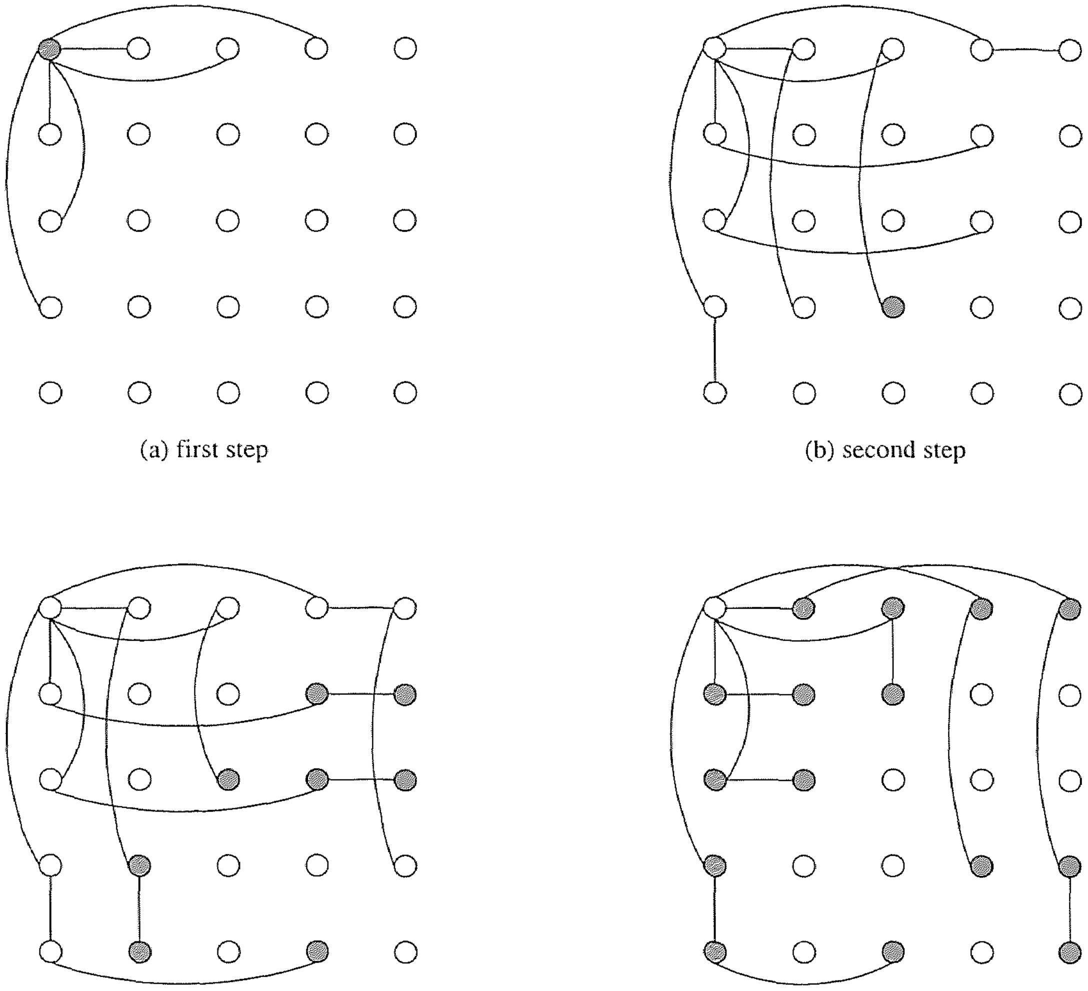
$$T(WR)^{full}_{one\_to\_all\_pers,3} = t_s + \sqrt{p}mt_w = O(m\sqrt{p})$$

6.5 All-to-All Personalized Communication

Tables 6.4 and 6.5 show the initial state and the required final result for all-to-all personalized communication in a 5 × 5 2-D system. Two phases are implemented again.

6.5.1 Model-1 and Model-2

We form rings on rows and columns. In each transfer step the message size is m words and every processor tries to transfer the message(s) destined for its farthest processor. We start with row transfers and continue with $\sqrt{p}$ all-to-all personalized


(a) first step

(b) second step

(c) third step

(d) fourth step

Figure 6.3 One-to-all personalized communication under model-3, for $w = 3$. The Cartesian coordinates of destination processors are shown as pairs of numbers. A shaded circle means that the corresponding processor has already received the personalized message sent by the source.

Table 6.4 The initial state for all-to-all personalized communication in 2-D HOW system.

Initial state of HOW(5,3,2)	
{(0,0),(0,0)},{(0,0),(0,1)},{(0,0),(0,2)},{(0,0),(0,3)},{(0,0),(0,4)}, {(0,0),(1,0)},{(0,0),(1,1)},{(0,0),(1,2)},{(0,0),(1,3)},{(0,0),(1,4)}, {(0,0),(2,0)},{(0,0),(2,1)},{(0,0),(2,2)},{(0,0),(2,3)},{(0,0),(2,4)}, {(0,0),(3,0)},{(0,0),(3,1)},{(0,0),(3,2)},{(0,0),(3,3)},{(0,0),(3,4)}, {(0,0),(4,0)},{(0,0),(4,1)},{(0,0),(4,2)},{(0,0),(4,3)},{(0,0),(4,4)}	{(1,0),(0,0)}, ...
{(1,0),(0,0)},{(1,0),(0,1)},{(1,0),(0,2)},{(1,0),(0,3)},{(1,0),(0,4)}, {(1,0),(1,0)},{(1,0),(1,1)},{(1,0),(1,2)},{(1,0),(1,3)},{(1,0),(1,4)}, {(1,0),(2,0)},{(1,0),(2,1)},{(1,0),(2,2)},{(1,0),(2,3)},{(1,0),(2,4)}, {(1,0),(3,0)},{(1,0),(3,1)},{(1,0),(3,2)},{(1,0),(3,3)},{(1,0),(3,4)}, {(1,0),(4,0)},{(1,0),(4,1)},{(1,0),(4,2)},{(1,0),(4,3)},{(1,0),(4,4)}	{(1,1),(0,0)}, ...
{(2,0),(0,0)},{(2,0),(0,1)},{(2,0),(0,2)},{(2,0),(0,3)},{(2,0),(0,4)}, {(2,0),(1,0)},{(2,0),(1,1)},{(2,0),(1,2)},{(2,0),(1,3)},{(2,0),(1,4)}, {(2,0),(2,0)},{(2,0),(2,1)},{(2,0),(2,2)},{(2,0),(2,3)},{(2,0),(2,4)}, {(2,0),(3,0)},{(2,0),(3,1)},{(2,0),(3,2)},{(2,0),(3,3)},{(2,0),(3,4)}, {(2,0),(4,0)},{(2,0),(4,1)},{(2,0),(4,2)},{(2,0),(4,3)},{(2,0),(4,4)}	{(2,1),(0,0)}, ...
{(3,0),(0,0)},{(3,0),(0,1)},{(3,0),(0,2)},{(3,0),(0,3)},{(3,0),(0,4)}, {(3,0),(1,0)},{(3,0),(1,1)},{(3,0),(1,2)},{(3,0),(1,3)},{(3,0),(1,4)}, {(3,0),(2,0)},{(3,0),(2,1)},{(3,0),(2,2)},{(3,0),(2,3)},{(3,0),(2,4)}, {(3,0),(3,0)},{(3,0),(3,1)},{(3,0),(3,2)},{(3,0),(3,3)},{(3,0),(3,4)}, {(3,0),(4,0)},{(3,0),(4,1)},{(3,0),(4,2)},{(3,0),(4,3)},{(3,0),(4,4)}	{(3,1),(0,0)}, ...
{(4,0),(0,0)},{(4,0),(0,1)},{(4,0),(0,2)},{(4,0),(0,3)},{(4,0),(0,4)}, {(4,0),(1,0)},{(4,0),(1,1)},{(4,0),(1,2)},{(4,0),(1,3)},{(4,0),(1,4)}, {(4,0),(2,0)},{(4,0),(2,1)},{(4,0),(2,2)},{(4,0),(2,3)},{(4,0),(2,4)}, {(4,0),(3,0)},{(4,0),(3,1)},{(4,0),(3,2)},{(4,0),(3,3)},{(4,0),(3,4)}, {(4,0),(4,0)},{(4,0),(4,1)},{(4,0),(4,2)},{(4,0),(4,3)},{(4,0),(4,4)}	{(4,1),(0,0)}, ...

Table 6.5 The final result for all-to-all personalized communication in a 2-D HOW system.

Required final state of HOW(5,3,2)	
{(0,0),(0,0)},{(0,1),(0,0)},{(0,2),(0,0)},{(0,3),(0,0)},{(0,4),(0,0)}, {(1,0),(0,0)},{(1,1),(0,0)},{(1,2),(0,0)},{(1,3),(0,0)},{(1,4),(0,0)}, {(2,0),(0,0)},{(2,1),(0,0)},{(2,2),(0,0)},{(2,3),(0,0)},{(2,4),(0,0)}, {(3,0),(0,0)},{(3,1),(0,0)},{(3,2),(0,0)},{(3,3),(0,0)},{(3,4),(0,0)}, {(4,0),(0,0)},{(4,1),(0,0)},{(4,2),(0,0)},{(4,3),(0,0)},{(4,4),(0,0)}	{(0,0),(0,1)}, ...
{(0,0),(1,0)},{(0,1),(1,0)},{(0,2),(1,0)},{(0,3),(1,0)},{(0,4),(1,0)}, {(1,0),(1,0)},{(1,1),(1,0)},{(1,2),(1,0)},{(1,3),(1,0)},{(1,4),(1,0)}, {(2,0),(1,0)},{(2,1),(1,0)},{(2,2),(1,0)},{(2,3),(1,0)},{(2,4),(1,0)}, {(3,0),(1,0)},{(3,1),(1,0)},{(3,2),(1,0)},{(3,3),(1,0)},{(3,4),(1,0)}, {(4,0),(1,0)},{(4,1),(1,0)},{(4,2),(1,0)},{(4,3),(1,0)},{(4,4),(1,0)}	{(0,0),(1,1)}, ...
{(0,0),(2,0)},{(0,1),(2,0)},{(0,2),(2,0)},{(0,3),(2,0)},{(0,4),(2,0)}, {(1,0),(2,0)},{(1,1),(2,0)},{(1,2),(2,0)},{(1,3),(2,0)},{(1,4),(2,0)}, {(2,0),(2,0)},{(2,1),(2,0)},{(2,2),(2,0)},{(2,3),(2,0)},{(2,4),(2,0)}, {(3,0),(2,0)},{(3,1),(2,0)},{(3,2),(2,0)},{(3,3),(2,0)},{(3,4),(2,0)}, {(4,0),(2,0)},{(4,1),(2,0)},{(4,2),(2,0)},{(4,3),(2,0)},{(4,4),(2,0)}	{(0,0),(2,1)}, ...
{(0,0),(3,0)},{(0,1),(3,0)},{(0,2),(3,0)},{(0,3),(3,0)},{(0,4),(3,0)}, {(1,0),(3,0)},{(1,1),(3,0)},{(1,2),(3,0)},{(1,3),(3,0)},{(1,4),(3,0)}, {(2,0),(3,0)},{(2,1),(3,0)},{(2,2),(3,0)},{(2,3),(3,0)},{(2,4),(3,0)}, {(3,0),(3,0)},{(3,1),(3,0)},{(3,2),(3,0)},{(3,3),(3,0)},{(3,4),(3,0)}, {(4,0),(3,0)},{(4,1),(3,0)},{(4,2),(3,0)},{(4,3),(3,0)},{(4,4),(3,0)}	{(0,0),(3,1)}, ...
{(0,0),(4,0)},{(0,1),(4,0)},{(0,2),(4,0)},{(0,3),(4,0)},{(0,4),(4,0)}, {(1,0),(4,0)},{(1,1),(4,0)},{(1,2),(4,0)},{(1,3),(4,0)},{(1,4),(4,0)}, {(2,0),(4,0)},{(2,1),(4,0)},{(2,2),(4,0)},{(2,3),(4,0)},{(2,4),(4,0)}, {(3,0),(4,0)},{(3,1),(4,0)},{(3,2),(4,0)},{(3,3),(4,0)},{(3,4),(4,0)}, {(4,0),(4,0)},{(4,1),(4,0)},{(4,2),(4,0)},{(4,3),(4,0)},{(4,4),(4,0)}	{(0,0),(4,1)}, ...

communications within columns. Based on the implementation of $(\sqrt{p}+1)$ all-to-all personalized 1-D HOW operations, we get

$$T_{all\_to\_all\_pers,1} = t_s + (\sqrt{p}+1)mt_w(\lceil\frac{\sqrt{p}-1}{2}\rceil^2 + \lfloor\frac{\sqrt{p}-1}{2}\rfloor) +$$
$$(\sqrt{p}+1)t_c(\lceil\frac{\sqrt{p}-1}{2}\rceil^2 - 2\lceil\frac{\sqrt{p}-1}{2}\rceil + \lfloor\frac{\sqrt{p}-1}{2}\rfloor) = O(mp^{3/2})$$

With wormhole routing, the communication time is

$$T(WR)_{all\_to\_all\_pers,1} = t_s + (\sqrt{p}+1)mt_w(\lceil\frac{\sqrt{p}-1}{2}\rceil^2 + \lfloor\frac{\sqrt{p}-1}{2}\rfloor) = O(mp^{3/2})$$

Special-case: Fully connected 1-D subsystems. For a fully connected 1-D system, because all the processors use one port at a time to send a single message, the total time taken is the same as that for the regular case.

The total time taken by this operation is

$$T^{full}_{all\_to\_all\_pers,1} = t_s + mt_w\frac{(p-1)\sqrt{p}}{2} + t_c(\frac{(p-1)\sqrt{p}}{2} - 1) = O(mp^{3/2})$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{all\_to\_all\_pers,1} = t_s + t_w(\sqrt{p}-1)(p-1)m = O(mp^{3/2})$$

6.5.2 Model-3

The implementation of this operation requires the following steps:

- Each processor transmits $\sqrt{p}$ values to each of the other $\sqrt{p}-1$ processors on its row, to be later distributed on the corresponding columns. At the end of this step, each processor has received $(\sqrt{p}-1) * \sqrt{p}$ messages. This operation is equivalent to $\sqrt{p}$ all-to-all personalized communications on an 1-D HOW (row).

- In this step, each processor transmits the values it received earlier and its own $\sqrt{p}-1$ values to the other processors on its column. Since $\sqrt{p}-1$ of the messages received in the first step were destined for this particular processor, the number of messages to be transmitted is $(\sqrt{p}-1)*\sqrt{p}-(\sqrt{p}-1)+(\sqrt{p}-1) = (\sqrt{p}-1)*\sqrt{p}$.

So the total number of all-to-all personalized 1-D HOW communications is $\sqrt{p}(\sqrt{p}-1)+\sqrt{p}=p$. Therefore, the total amount of time is

$$\begin{aligned} T_{all\_to\_all\_pers,3} &= t_s + p\ (2\ mt_w \lceil \frac{\sqrt{p}-1}{w} \rceil + mt_c(2\ \lceil \frac{\sqrt{p}-1}{w} \rceil - 1)) \\ &= O(m\frac{p^{3/2}}{w}) \end{aligned}$$

With wormhole routing, the time is

$$T(WR)_{all\_to\_all\_pers,3} = t_s + p\ mt_w 2 \lceil \frac{\sqrt{p}-1}{w} \rceil = O(m\frac{p^{3/2}}{w})$$

Special-case: Fully connected 1-D subsystems. For a fully connected 1-D subsystem, all the processors use all output ports sending different destined messages to all accessible processors. The total time taken by the operation is

$$T^{full}_{all\_to\_all\_pers,3} = t_s + (p-1)mt_w = O(mp)$$

With wormhole routing, the communication time is

$$T(WR)^{full}_{all\_to\_all\_pers,3} = t_s + (p-1)mt_w = O(mp)$$

CHAPTER 7
COMMUNICATION OPERATIONS ON BINARY HYPERCUBES

We compare here the performance of 2-D HOW systems with that of binary hypercubes for the studied set of communication operations. The (binary) hypercube is an interconnection network that has been widely used in parallel processing, primarily in the 1980's. A tremendous number of algorithms have been developed for this system. The d-D binary hypercube or d-cube contains 2^d nodes. Two nodes are neighbors if and only if their d-bit unique addresses differ in a single bit. A hypercube with p nodes has $(\frac{p}{2}\log p)$ edges.

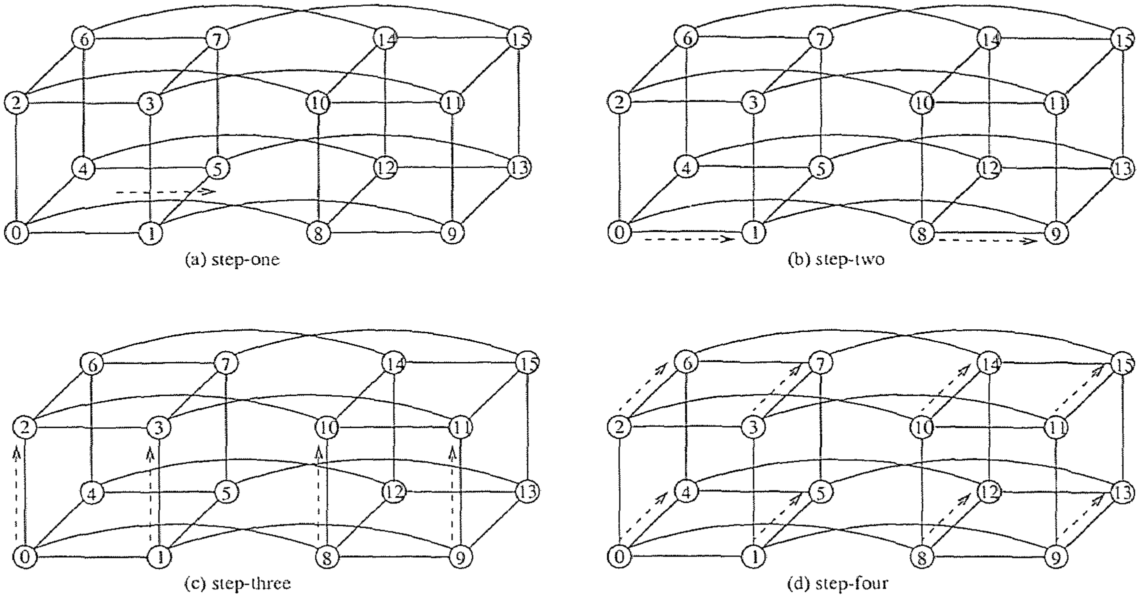
No matter what communication model we are using (such as model-1, model-2, or model-3), the number of transfer steps is the same and depends on $d = \log p$.

The examples shown in this section are for the 16-processor hypercube or 4-cube.

Of course, the one-to-all communication procedure is different from the all-to-all communication procedure. For one-to-all communication, the channels used in this communication procedure are shown in Figure 7.1. In each step, there is only one message sent along each direction. The number of channels and which channel will be used are shown in Figure 7.1.

For all-to-all communication, in each step there are $2^{\log p-1} = 2^{4-1} = 8$ channels to be used and the pairs of processors exchange their information. Of course, different channels will be used in different steps. Figure 7.2 shows the channels involved in the 4-cube for all-to-all communication.

For the sake of simplicity, we restrict our comparisons to model-3, the most powerful communication model, by also assuming the store-and-forward routing technique. In fact, the equations we derive for the hypercube are also valid under model-1 and model-2. First, we briefly evaluate communication operations for hypercubes [5]. Then, comparisons with HOW systems follow in Section 6.



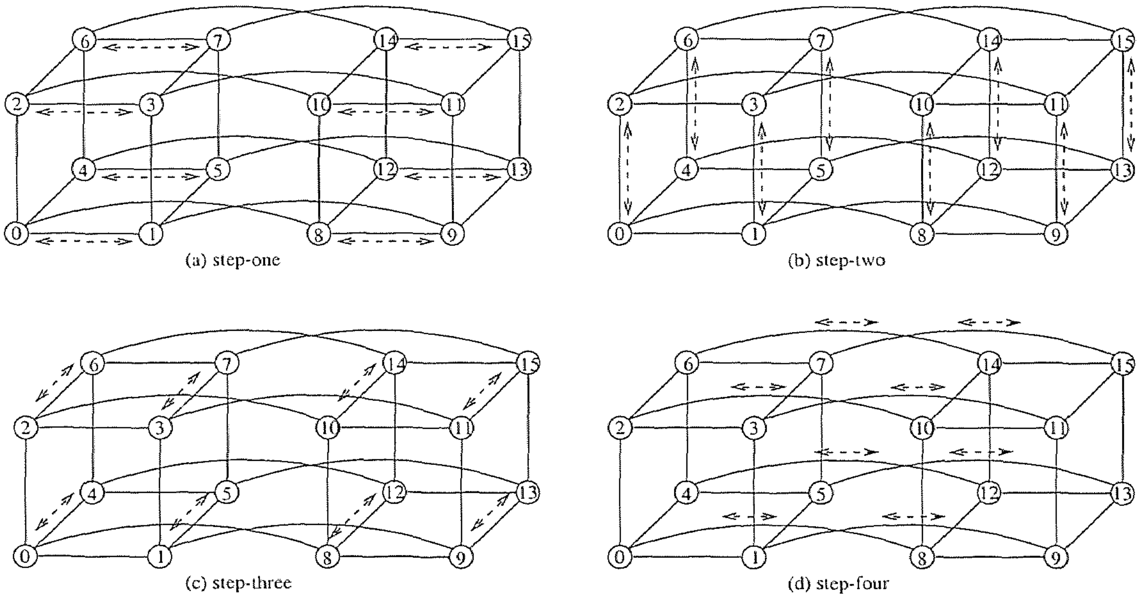
(a) step-one

(b) step-two

(c) step-three

(d) step-four

Figure 7.1 One-to-all communication procedure with 16 processors, for a hypercube system.



(a) step-one

(b) step-two

(c) step-three

(d) step-four

Figure 7.2 All-to-all communication procedure with 16 processors, for a hypercube system.

7.1 One-to-One Communication

Routing in the hypercube is carried out by first producing the XOR (exclusive-OR) result between the d-bit source and destination addresses and then routing the message in those dimensions where the bit in the XOR result is equal to 1. Two addresses may differ in up to d bits, and therefore the maximum distance is equal to $d = \log p$.

Therefore, the *upper bound* on the communication time is

$$T_{one\_to\_one} = t_s + mt_w \log p + (\log p - 1)t_c = O(m \log p)$$

7.2 One-to-All Broadcasting

The implementation of this communication operation requires the traversal of all d dimensions. Despite the fact that the order chosen for the traversal of the d dimensions does not matter, the description here assumes that this traversal starts with the highest dimension. In the first phase, the source processor sends the message to its neighbor in the $(d - 1)$-th dimension. In the second phase, the source and the processor that previously received the message send a copy to their neighbors in the $(d - 2)$-th dimension. In general, in the s-th phase, the 2^{s-1} processors that have a copy of the message send a copy to their neighbors in the $(d - s)$-th dimension, for $1 \leq s \leq d$.

The communication time required here is the same as the worst-case communication time required for one-to-one communication, the only difference being that for one-to-all broadcasting the message is stored in the intermediate nodes while for one-to-one communication the message is not stored in the intermediate nodes. Therefore,

$$T_{one\_to\_all} = t_s + mt_w \log p + (\log p - 1)t_c = O(m \log p)$$

7.3 All-to-All Broadcasting

This operation is carried out in $d = \log p$ steps. Pairs of processors exchange information in each step. Each step doubles the size of the data to be exchanged between processors in the next step because processors concatenate their current data with the data they receive. Each step i, for $i = 1, 2, ..., d$, implements communications in a different dimension i, and the size of all messages in step i is $(2^{i-1}m)$ words. The communication time is

$$\begin{aligned} T_{all\_to\_all} &= t_s + (\sum_{i=1}^{\log p} 2^{i-1}m)t_w + (\log p - 1)t_c \\ &= t_s + m(2^{\log p} - 1)t_w + (\log p - 1)t_c \\ &= t_s + m(p-1)t_w + (\log p - 1)t_c \quad = \quad O(mp) \end{aligned}$$

Table 7.1 shows the entire procedure of all-to-all broadcasting in the 4-cube.

7.4 One-to-All Personalized Communication

The communication patterns are similar to those for one-to-all broadcasting. However, the amounts of information to be exchanged in different steps differ dramatically. In step i, for $i = 1, 2, ..., d$, a processor that has received earlier data (or the source processor for $i = 1$) sends half of its data to its neighbor in dimension i; the set of 2^{d-i} values sent to that neighbor is for the 2^{d-i} processors with the higher addresses if the neighbor has a higher address (otherwise, the values are for the 2^{d-i} processors with the lower addresses). The communication time is

$$\begin{aligned} T_{one\_to\_all\_pers} &= t_s + (\sum_{i=1}^{\log p} 2^{\log p - i}m)t_w + (\log p - 1)t_c \\ &= t_s + (\sum_{i=0}^{\log p - 1} 2^i m)t_w + (\log p - 1)t_c \\ &= t_s + m(2^{\log p} - 1)t_w + (\log p - 1)t_c \\ &= t_s + m(p-1)t_w + (\log p - 1)t_c \quad = \quad O(mp) \end{aligned}$$

Table 7.2 shows the details involved in this communication.

Table 7.1 Detailed information for all-to-all broadcasting on the hypercube.

Initial state			
p_0 with message m_0	p_1 with message m_1	p_2 with message m_2	p_3 with message m_3
p_4 with message m_4	p_5 with message m_5	p_6 with message m_6	p_7 with message m_7
p_8 with message m_8	p_9 with message m_9	p_{10} with message m_{10}	p_{11} with message m_{11}
p_{12} with message m_{12}	p_{13} with message m_{13}	p_{14} with message m_{14}	p_{15} with message m_{15}
First step (among two processors with first bit difference, such as p_0 and p_1.)			
m_0, m_1	m_0, m_1	m_2, m_3	m_2, m_3
m_4, m_5	m_4, m_5	m_6, m_7	m_6, m_7
m_8, m_9	m_8, m_9	m_{10}, m_{11}	m_{10}, m_{11}
m_{12}, m_{13}	m_{12}, m_{13}	m_{14}, m_{15}	m_{14}, m_{15}
Second step (among two processors with second bit difference, such as p_0 and p_2.)			
m_0,m_1,m_2,m_3	m_0,m_1,m_2,m_3	m_0,m_1,m_2,m_3	m_0,m_1,m_2,m_3
m_4,m_5,m_6,m_7	m_4,m_5,m_6,m_7	m_4,m_5,m_6,m_7	m_4,m_5,m_6,m_7
m_8,m_9,m_{10},m_{11}	m_8,m_9,m_{10},m_{11}	m_8,m_9,m_{10},m_{11}	m_8,m_9,m_{10},m_{11}
m_{12},m_{13},m_{14},m_{15}	m_{12},m_{13},m_{14},m_{15}	m_{12},m_{13},m_{14},m_{15}	m_{12},m_{13},m_{14},m_{15}
Third step (among two processors with third bit difference, such as p_0 and p_4.)			
m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7
m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7
m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}
m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}
Fourth step (among two processors with forth bit difference, such as p_0 and p_8.)			
m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}
m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}
m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}
m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}

Table 7.2 Detailed information for one-to-all personalized communication on the hypercube.

Detail information about one-to-all personalized communication.			
Initial state			
p_0 with message m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7 m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}	p_1 with no message	p_2 with no message	p_3 with no message
p_4 with no message	p_5 with no message	p_6 with no message	p_7 with no message
p_8 with no message	p_9 with no message	p_{10} with no message	p_{11} with no message
p_{12} with no message	p_{13} with no message	p_{14} with no message	p_{15} with no message
First step: Message transfer from p_0 to p_8.			
m_0,m_1,m_2,m_3 m_4,m_5,m_6,m_7			
m_8,m_9,m_{10},m_{11} m_{12},m_{13},m_{14},m_{15}			
Second step: Message transfer from p_0 to p_1 and from p_8 to p_9.			
m_0,m_2,m_4,m_6	m_1,m_3,m_5,m_7		
m_8,m_{10},m_{12},m_{14}	m_9,m_{11},m_{13},m_{15}		
Third step: Message transfer from p_0 to p_2, from p_1 to p_3, from p_8 to p_{10}, and from p_9 to p_{11}			
m_0,m_4	m_1,m_5	m_2,m_6	m_3;m_7
m_8,m_{12}	m_9,m_{13}	m_{10},m_{14}	m_{11},m_{15}

Fourth step: Message transfer from p_0 to p_4, from p_1 to p_5,
from p_2 to p_6, and from p_3 to p_7;
from p_8 to p_{12}, from p_9 to p_{13}, from p_{10} to p_{14}, and from p_{11} to p_{15}.			

m_0	m_1	m_2	m_3
m_4	m_5	m_6	m_7
m_8	m_9	m_{10}	m_{11}
m_{12}	m_{13}	m_{14}	m_{15}

7.5 All-to-All Personalized Communication

This operation also requires $\log p$ communication steps. Each processor contains p values in each step. In step i, for $i = 1, 2, ..., d$, each processor sends half of its data to its neighbor in the i-th dimension; these data are destined for processors whose the i-th bit in the address is similar to that of the chosen neighbor. The communication time is

$$T_{all\_to\_all\_pers} \quad = \quad t_s + (\log p)(\frac{p}{2}m)t_w + (\log p - 1)t_c = O(mp \log p)$$

Tables 7.3, 7.4, and 7.5 show the details involved in this communication procedure.

Table 7.3 Detailed information for all-to-all personalized communication on the hypercube.

Detailed information for all-to-all personalized communication.			
Initial state			
p_0 with message $m_{00},m_{01},m_{02},m_{03}$ $m_{04},m_{05},m_{06},m_{07}$ $m_{08},m_{09},m_{0,10},m_{0,11}$ $m_{0,12},m_{0,13},m_{0,14},m_{0,15}$	p_1 with message $m_{10},m_{11},m_{12},m_{13}$ $m_{14},m_{15},m_{16},m_{17}$ $m_{18},m_{19},m_{1,10},m_{1,11}$ $m_{1,12},m_{1,13},m_{1,14},m_{1,15}$	p_2 with message $m_{20},m_{21},m_{22},m_{23}$ $m_{24},m_{25},m_{26},m_{27}$ $m_{28},m_{29},m_{2,10},m_{2,11}$ $m_{2,12},m_{2,13},m_{2,14},m_{2,15}$	p_3 with message $m_{30},m_{31},m_{32},m_{33}$ $m_{34},m_{35},m_{36},m_{37}$ $m_{38},m_{39},m_{3,10},m_{3,11}$ $m_{3,12},m_{3,13},m_{3,14},m_{3,15}$
p_4 with message $m_{40},m_{41},m_{42},m_{43}$ $m_{44},m_{45},m_{46},m_{47}$ $m_{48},m_{49},m_{4,10},m_{4,11}$ $m_{4,12},m_{4,13},m_{4,14},m_{4,15}$	p_5 with message $m_{50},m_{51},m_{52},m_{53}$ $m_{54},m_{55},m_{56},m_{57}$ $m_{58},m_{59},m_{5,10},m_{5,11}$ $m_{5,12},m_{5,13},m_{5,14},m_{5,15}$	p_6 with message $m_{60},m_{61},m_{62},m_{63}$ $m_{64},m_{65},m_{66},m_{67}$ $m_{68},m_{69},m_{6,10},m_{6,11}$ $m_{6,12},m_{6,13},m_{6,14},m_{6,15}$	p_7 with message $m_{70},m_{71},m_{72},m_{73}$ $m_{74},m_{75},m_{76},m_{77}$ $m_{78},m_{79},m_{7,10},m_{7,11}$ $m_{7,12},m_{7,13},m_{7,14},m_{7,15}$
p_8 with message $m_{80},m_{81},m_{82},m_{83}$ $m_{84},m_{85},m_{86},m_{87}$ $m_{88},m_{89},m_{8,10},m_{8,11}$ $m_{8,12},m_{8,13},m_{8,14},m_{8,15}$	p_9 with message $m_{90},m_{91},m_{92},m_{93}$ $m_{94},m_{95},m_{96},m_{97}$ $m_{98},m_{99},m_{9,10},m_{9,11}$ $m_{9,12},m_{9,13},m_{9,14},m_{9,15}$	p_{10} with message $m_{10,0},m_{10,1},m_{10,2},m_{10,3}$ $m_{10,4},m_{10,5},m_{10,6},m_{10,7}$ $m_{10,8},m_{10,9},m_{10,10},m_{10,11}$ $m_{10,12},m_{10,13},m_{10,14},m_{10,15}$	p_{11} with message $m_{11,0},m_{11,1},m_{11,2},m_{11,3}$ $m_{11,4},m_{11,5},m_{11,6},m_{11,7}$ $m_{11,8},m_{11,9},m_{11,10},m_{11,11}$ $m_{11,12},m_{11,13},m_{11,14},m_{11,15}$
p_{12} with message $m_{12,0},m_{12,1},m_{12,2},m_{12,3}$ $m_{12,4},m_{12,5},m_{12,6},m_{12,7}$ $m_{12,8},m_{12,9},m_{12,10},m_{12,11}$ $m_{12,12},m_{12,13},m_{12,14},m_{12,15}$	p_{13} with message $m_{13,0},m_{13,1},m_{13,2},m_{13,3}$ $m_{13,4},m_{13,5},m_{13,6},m_{13,7}$ $m_{13,8},m_{13,9},m_{13,10},m_{13,11}$ $m_{13,12},m_{13,13},m_{13,14},m_{13,15}$	p_{14} with message $m_{14,0},m_{14,1},m_{14,2},m_{14,3}$ $m_{14,4},m_{14,5},m_{14,6},m_{14,7}$ $m_{14,8},m_{14,9},m_{14,10},m_{14,11}$ $m_{14,12},m_{14,13},m_{14,14},m_{14,15}$	p_{15} with message $m_{15,0},m_{15,1},m_{15,2},m_{15,3}$ $m_{15,4},m_{15,5},m_{15,6},m_{15,7}$ $m_{15,8},m_{15,9},m_{15,10},m_{15,11}$ $m_{15,12},m_{15,13},m_{15,14},m_{15,15}$
First step (among two processors with first bit difference, such as p_0 and p_1.)			
$m_{00},m_{10},m_{02},m_{12}$ $m_{04},m_{14},m_{06},m_{16}$ $m_{08},m_{18},m_{0,10},m_{1,10}$ $m_{0,12},m_{1,12},m_{0,14},m_{1,14}$	$m_{01},m_{11},m_{03},m_{13}$ $m_{05},m_{15},m_{07},m_{17}$ $m_{09},m_{19},m_{0,11},m_{1,11}$ $m_{0,13},m_{1,13},m_{0,15},m_{1,15}$	$m_{20},m_{30},m_{22},m_{32}$ $m_{24},m_{34},m_{26},m_{36}$ $m_{28},m_{38},m_{2,10},m_{3,10}$ $m_{2,12},m_{3,12},m_{2,14},m_{3,14}$	$m_{21},m_{31},m_{23},m_{33}$ $m_{25},m_{35},m_{27},m_{37}$ $m_{29},m_{39},m_{2,11},m_{3,11}$ $m_{2,13},m_{3,13},m_{2,15},m_{3,15}$
$m_{40},m_{50},m_{42},m_{52}$ $m_{44},m_{54},m_{46},m_{56}$ $m_{48},m_{58},m_{4,10},m_{5,10}$ $m_{4,12},m_{5,12},m_{4,14},m_{5,14}$	$m_{41},m_{51},m_{43},m_{53}$ $m_{45},m_{55},m_{47},m_{57}$ $m_{49},m_{59},m_{4,11},m_{5,11}$ $m_{4,13},m_{5,13},m_{4,15},m_{5,15}$	$m_{60},m_{70},m_{62},m_{72}$ $m_{64},m_{74},m_{66},m_{76}$ $m_{68},m_{78},m_{6,10},m_{7,10}$ $m_{6,12},m_{7,12},m_{6,14},m_{7,14}$	$m_{61},m_{71},m_{63},m_{73}$ $m_{65},m_{75},m_{67},m_{77}$ $m_{69},m_{79},m_{6,11},m_{7,11}$ $m_{6,13},m_{7,13},m_{6,15},m_{7,15}$
$m_{80},m_{90},m_{82},m_{92}$ $m_{84},m_{94},m_{86},m_{96}$ $m_{88},m_{98},m_{8,10},m_{9,10}$ $m_{8,12},m_{9,12},m_{8,14},m_{9,14}$	$m_{81},m_{91},m_{83},m_{93}$ $m_{85},m_{95},m_{87},m_{97}$ $m_{89},m_{99},m_{8,11},m_{9,11}$ $m_{8,13},m_{9,13},m_{8,15},m_{9,15}$	$m_{10,0},m_{11,0},m_{10,2},m_{11,2}$ $m_{10,4},m_{11,4},m_{10,6},m_{11,6}$ $m_{10,8},m_{11,8},m_{10,10},m_{11,10}$ $m_{10,12},m_{11,12},m_{10,14},m_{11,14}$	$m_{10,1},m_{11,1},m_{10,3},m_{11,3}$ $m_{10,5},m_{11,5},m_{10,7},m_{11,7}$ $m_{10,9},m_{11,9},m_{10,11},m_{11,11}$ $m_{10,13},m_{11,13},m_{10,15},m_{11,15}$
$m_{12,0},m_{13,0},m_{12,2},m_{13,2}$ $m_{12,4},m_{13,4},m_{12,6},m_{13,6}$ $m_{12,8},m_{13,8},m_{12,10},m_{13,10}$ $m_{12,12},m_{13,12},m_{12,14},m_{13,14}$	$m_{12,1},m_{13,1},m_{12,3},m_{13,3}$ $m_{12,5},m_{13,5},m_{12,7},m_{13,7}$ $m_{12,9},m_{13,9},m_{12,11},m_{13,11}$ $m_{12,13},m_{13,13},m_{12,15},m_{13,15}$	$m_{14,0},m_{15,0},m_{14,2},m_{15,2}$ $m_{14,4},m_{15,4},m_{14,6},m_{15,6}$ $m_{14,8},m_{15,8},m_{14,10},m_{15,10}$ $m_{14,12},m_{15,12},m_{14,14},m_{15,14}$	$m_{14,1},m_{15,1},m_{14,3},m_{15,3}$ $m_{14,5},m_{15,5},m_{14,7},m_{15,7}$ $m_{14,9},m_{15,9},m_{14,11},m_{15,11}$ $m_{14,13},m_{15,13},m_{14,15},m_{15,15}$

Table 7.4 Detailed information for all-to-all personalized communication on the hypercube (continued).

<table>
<tr><td colspan="4">Second step (among two processors with second bit difference, such as p_0 and p_2.)</td></tr>
<tr>
<td>$m_{00}, m_{10}, m_{20}, m_{30}$
$m_{04}, m_{14}, m_{24}, m_{34}$
$m_{08}, m_{18}, m_{28}, m_{38}$
$m_{0,12}, m_{1,12}, m_{2,12}, m_{3,12}$</td>
<td>$m_{01}, m_{11}, m_{21}, m_{31}$
$m_{05}, m_{15}, m_{25}, m_{35}$
$m_{09}, m_{19}, m_{29}, m_{39}$
$m_{0,13}, m_{1,13}, m_{2,13}, m_{3,13}$</td>
<td>$m_{02}, m_{12}, m_{22}, m_{32}$
$m_{06}, m_{16}, m_{26}, m_{36}$
$m_{0,10}, m_{1,10}, m_{2,10}, m_{3,10}$
$m_{0,14}, m_{1,14}, m_{2,14}, m_{3,14}$</td>
<td>$m_{03}, m_{13}, m_{23}, m_{33}$
$m_{07}, m_{17}, m_{27}, m_{37}$
$m_{0,11}, m_{1,11}, m_{2,11}, m_{3,11}$
$m_{0,15}, m_{1,15}, m_{2,15}, m_{3,15}$</td>
</tr>
<tr>
<td>$m_{40}, m_{50}, m_{60}, m_{70}$
$m_{44}, m_{54}, m_{64}, m_{74}$
$m_{48}, m_{58}, m_{68}, m_{78}$
$m_{4,12}, m_{5,12}, m_{6,12}, m_{7,12}$</td>
<td>$m_{41}, m_{51}, m_{61}, m_{71}$
$m_{45}, m_{55}, m_{65}, m_{75}$
$m_{49}, m_{59}, m_{69}, m_{79}$
$m_{4,13}, m_{5,13}, m_{6,13}, m_{7,13}$</td>
<td>$m_{42}, m_{52}, m_{62}, m_{72}$
$m_{46}, m_{56}, m_{66}, m_{76}$
$m_{4,10}, m_{5,10}, m_{6,10}, m_{7,10}$
$m_{4,14}, m_{5,14}, m_{6,14}, m_{7,14}$</td>
<td>$m_{43}, m_{53}, m_{63}, m_{73}$
$m_{47}, m_{57}, m_{67}, m_{77}$
$m_{4,11}, m_{5,11}, m_{6,11}, m_{7,11}$
$m_{4,15}, m_{5,15}, m_{6,15}, m_{7,15}$</td>
</tr>
<tr>
<td>$m_{80}, m_{90}, m_{10,0}, m_{11,0}$
$m_{84}, m_{94}, m_{10,4}, m_{11,4}$
$m_{88}, m_{98}, m_{10,8}, m_{11,8}$
$m_{8,12}, m_{9,12}, m_{10,12}, m_{11,12}$</td>
<td>$m_{81}, m_{91}, m_{10,1}, m_{11,1}$
$m_{85}, m_{95}, m_{10,5}, m_{11,5}$
$m_{89}, m_{99}, m_{10,9}, m_{11,9}$
$m_{8,13}, m_{9,13}, m_{10,13}, m_{11,13}$</td>
<td>$m_{82}, m_{92}, m_{10,2}, m_{11,2}$
$m_{86}, m_{96}, m_{10,6}, m_{11,6}$
$m_{8,10}, m_{9,10}, m_{10,10}, m_{11,10}$
$m_{8,14}, m_{9,14}, m_{10,14}, m_{11,14}$</td>
<td>$m_{83}, m_{93}, m_{10,3}, m_{11,3}$
$m_{87}, m_{97}, m_{10,7}, m_{11,7}$
$m_{8,11}, m_{9,11}, m_{10,11}, m_{11,11}$
$m_{8,15}, m_{9,15}, m_{10,15}, m_{11,15}$</td>
</tr>
<tr>
<td>$m_{12,0}, m_{13,0}, m_{14,0}, m_{15,0}$
$m_{12,4}, m_{13,4}, m_{14,4}, m_{15,4}$
$m_{12,8}, m_{13,8}, m_{14,8}, m_{15,8}$
$m_{12,12}, m_{13,12}, m_{14,12}, m_{15,12}$</td>
<td>$m_{12,1}, m_{13,1}, m_{14,1}, m_{15,1}$
$m_{12,5}, m_{13,5}, m_{14,5}, m_{15,5}$
$m_{12,9}, m_{13,9}, m_{14,9}, m_{15,9}$
$m_{12,13}, m_{13,13}, m_{14,13}, m_{15,13}$</td>
<td>$m_{12,2}, m_{13,2}, m_{14,2}, m_{15,2}$
$m_{12,6}, m_{13,6}, m_{14,6}, m_{15,6}$
$m_{12,10}, m_{13,10}, m_{14,10}, m_{15,10}$
$m_{12,14}, m_{13,14}, m_{14,14}, m_{15,14}$</td>
<td>$m_{12,3}, m_{13,3}, m_{14,3}, m_{15,3}$
$m_{12,7}, m_{13,7}, m_{14,7}, m_{15,7}$
$m_{12,11}, m_{13,11}, m_{14,11}, m_{15,11}$
$m_{12,15}, m_{13,15}, m_{14,15}, m_{15,15}$</td>
</tr>
<tr><td colspan="4">Third step (among two processors with third bit difference, such as p_0 and p_4.)</td></tr>
<tr>
<td>$m_{00}, m_{10}, m_{20}, m_{30}$
$m_{40}, m_{50}, m_{60}, m_{70}$
$m_{08}, m_{18}, m_{28}, m_{38}$
$m_{48}, m_{58}, m_{68}, m_{78}$</td>
<td>$m_{01}, m_{11}, m_{21}, m_{31}$
$m_{41}, m_{51}, m_{61}, m_{71}$
$m_{09}, m_{19}, m_{29}, m_{39}$
$m_{49}, m_{59}, m_{69}, m_{79}$</td>
<td>$m_{02}, m_{12}, m_{22}, m_{32}$
$m_{42}, m_{52}, m_{62}, m_{72}$
$m_{0,10}, m_{1,10}, m_{2,10}, m_{3,10}$
$m_{4,10}, m_{5,10}, m_{6,10}, m_{7,10}$</td>
<td>$m_{03}, m_{13}, m_{23}, m_{33}$
$m_{43}, m_{53}, m_{63}, m_{73}$
$m_{0,11}, m_{1,11}, m_{2,11}, m_{3,11}$
$m_{4,11}, m_{5,11}, m_{6,11}, m_{7,11}$</td>
</tr>
<tr>
<td>$m_{04}, m_{14}, m_{24}, m_{34}$
$m_{44}, m_{54}, m_{64}, m_{74}$
$m_{0,12}, m_{1,12}, m_{2,12}, m_{3,12}$
$m_{4,12}, m_{5,12}, m_{6,12}, m_{7,12}$</td>
<td>$m_{05}, m_{15}, m_{25}, m_{35}$
$m_{45}, m_{55}, m_{65}, m_{75}$
$m_{0,13}, m_{1,13}, m_{2,13}, m_{3,13}$
$m_{4,13}, m_{5,13}, m_{6,13}, m_{7,13}$</td>
<td>$m_{06}, m_{16}, m_{26}, m_{36}$
$m_{46}, m_{56}, m_{66}, m_{76}$
$m_{0,14}, m_{1,14}, m_{2,14}, m_{3,14}$
$m_{4,14}, m_{5,14}, m_{6,14}, m_{7,14}$</td>
<td>$m_{07}, m_{17}, m_{27}, m_{37}$
$m_{47}, m_{57}, m_{67}, m_{77}$
$m_{0,15}, m_{1,15}, m_{2,15}, m_{3,15}$
$m_{4,15}, m_{5,15}, m_{6,15}, m_{7,15}$</td>
</tr>
<tr>
<td>$m_{80}, m_{90}, m_{10,0}, m_{11,0}$
$m_{12,0}, m_{13,0}, m_{14,0}, m_{15,0}$
$m_{88}, m_{98}, m_{10,8}, m_{11,8}$
$m_{12,8}, m_{13,8}, m_{14,8}, m_{15,8}$</td>
<td>$m_{81}, m_{91}, m_{10,1}, m_{11,1}$
$m_{12,1}, m_{13,1}, m_{14,1}, m_{15,1}$
$m_{89}, m_{99}, m_{10,9}, m_{11,9}$
$m_{12,9}, m_{13,9}, m_{14,9}, m_{15,9}$</td>
<td>$m_{82}, m_{92}, m_{10,2}, m_{11,2}$
$m_{12,2}, m_{13,2}, m_{14,2}, m_{15,2}$
$m_{8,10}, m_{9,10}, m_{10,10}, m_{11,10}$
$m_{12,10}, m_{13,10}, m_{14,10}, m_{15,10}$</td>
<td>$m_{83}, m_{93}, m_{10,3}, m_{11,3}$
$m_{12,3}, m_{13,3}, m_{14,3}, m_{15,3}$
$m_{8,11}, m_{9,11}, m_{10,11}, m_{11,11}$
$m_{12,11}, m_{13,11}, m_{14,11}, m_{15,11}$</td>
</tr>
<tr>
<td>$m_{84}, m_{94}, m_{10,4}, m_{11,4}$
$m_{12,4}, m_{13,4}, m_{14,4}, m_{15,4}$
$m_{8,12}, m_{9,12}, m_{10,12}, m_{11,12}$
$m_{12,12}, m_{13,12}, m_{14,12}, m_{15,12}$</td>
<td>$m_{85}, m_{95}, m_{10,5}, m_{11,5}$
$m_{12,5}, m_{13,5}, m_{14,5}, m_{15,5}$
$m_{8,13}, m_{9,13}, m_{10,13}, m_{11,13}$
$m_{12,13}, m_{13,13}, m_{14,13}, m_{15,13}$</td>
<td>$m_{86}, m_{96}, m_{10,6}, m_{11,6}$
$m_{12,6}, m_{13,6}, m_{14,6}, m_{15,6}$
$m_{8,14}, m_{9,14}, m_{10,14}, m_{11,14}$
$m_{12,14}, m_{13,14}, m_{14,14}, m_{15,14}$</td>
<td>$m_{87}, m_{97}, m_{10,7}, m_{11,7}$
$m_{12,7}, m_{13,7}, m_{14,7}, m_{15,7}$
$m_{8,15}, m_{9,15}, m_{10,15}, m_{11,15}$
$m_{12,15}, m_{13,15}, m_{14,15}, m_{15,15}$</td>
</tr>
</table>

Table 7.5 Detailed information for all-to-all personalized communication on the hypercube (continued).

Fourth step (among two processors with fourth bit difference, such as p_0 and p_8.)			
$m_{00}, m_{10}, m_{20}, m_{30}$ $m_{40}, m_{50}, m_{60}, m_{70}$ $m_{80}, m_{90}, m_{10,0}, m_{11,0}$ $m_{12,0}, m_{13,0}, m_{14,0}, m_{15,0}$	$m_{01}, m_{11}, m_{21}, m_{31}$ $m_{41}, m_{51}, m_{61}, m_{71}$ $m_{81}, m_{91}, m_{10,1}, m_{11,1}$ $m_{12,1}, m_{13,1}, m_{14,1}, m_{15,1}$	$m_{02}, m_{12}, m_{22}, m_{32}$ $m_{42}, m_{52}, m_{62}, m_{72}$ $m_{82}, m_{92}, m_{10,2}, m_{11,2}$ $m_{12,2}, m_{13,2}, m_{14,2}, m_{15,2}$	$m_{03}, m_{13}, m_{23}, m_{33}$ $m_{43}, m_{53}, m_{63}, m_{73}$ $m_{83}, m_{93}, m_{10,3}, m_{11,3}$ $m_{12,3}, m_{13,3}, m_{14,3}, m_{15,3}$
$m_{04}, m_{14}, m_{24}, m_{34}$ $m_{44}, m_{54}, m_{64}, m_{74}$ $m_{84}, m_{94}, m_{10,4}, m_{11,4}$ $m_{12,4}, m_{13,4}, m_{14,4}, m_{15,4}$	$m_{05}, m_{15}, m_{25}, m_{35}$ $m_{45}, m_{55}, m_{65}, m_{75}$ $m_{85}, m_{95}, m_{10,5}, m_{11,5}$ $m_{12,5}, m_{13,5}, m_{14,5}, m_{15,5}$	$m_{06}, m_{16}, m_{26}, m_{36}$ $m_{46}, m_{56}, m_{66}, m_{76}$ $m_{86}, m_{96}, m_{10,6}, m_{11,6}$ $m_{12,6}, m_{13,6}, m_{14,6}, m_{15,6}$	$m_{07}, m_{17}, m_{27}, m_{37}$ $m_{47}, m_{57}, m_{67}, m_{77}$ $m_{87}, m_{97}, m_{10,7}, m_{11,7}$ $m_{12,7}, m_{13,7}, m_{14,7}, m_{15,7}$
$m_{08}, m_{18}, m_{28}, m_{38}$ $m_{48}, m_{58}, m_{68}, m_{78}$ $m_{88}, m_{98}, m_{10,8}, m_{11,8}$ $m_{12,8}, m_{13,8}, m_{14,8}, m_{15,8}$	$m_{09}, m_{19}, m_{29}, m_{39}$ $m_{49}, m_{59}, m_{69}, m_{79}$ $m_{89}, m_{99}, m_{10,9}, m_{11,9}$ $m_{12,9}, m_{13,9}, m_{14,9}, m_{15,9}$	$m_{0,10}, m_{1,10}, m_{2,10}, m_{3,10}$ $m_{4,10}, m_{5,10}, m_{6,10}, m_{7,10}$ $m_{8,10}, m_{9,10}, m_{10,10}, m_{11,10}$ $m_{12,10}, m_{13,10}, m_{14,10}, m_{15,10}$	$m_{0,11}, m_{1,11}, m_{2,11}, m_{3,11}$ $m_{4,11}, m_{5,11}, m_{6,11}, m_{7,11}$ $m_{8,11}, m_{9,11}, m_{10,11}, m_{11,11}$ $m_{12,11}, m_{13,11}, m_{14,11}, m_{15,11}$
$m_{0,12}, m_{1,12}, m_{2,12}, m_{3,12}$ $m_{4,12}, m_{5,12}, m_{6,12}, m_{7,12}$ $m_{8,12}, m_{9,12}, m_{10,12}, m_{11,12}$ $m_{12,12}, m_{13,12}, m_{14,12}, m_{15,12}$	$m_{0,13}, m_{1,13}, m_{2,13}, m_{3,13}$ $m_{4,13}, m_{5,13}, m_{6,13}, m_{7,13}$ $m_{8,13}, m_{9,13}, m_{10,13}, m_{11,13}$ $m_{12,13}, m_{13,13}, m_{14,13}, m_{15,13}$	$m_{0,14}, m_{1,14}, m_{2,14}, m_{3,14}$ $m_{4,14}, m_{5,14}, m_{6,14}, m_{7,14}$ $m_{8,14}, m_{9,14}, m_{10,14}, m_{11,14}$ $m_{12,14}, m_{13,14}, m_{14,14}, m_{15,14}$	$m_{0,15}, m_{1,15}, m_{2,15}, m_{3,15}$ $m_{4,15}, m_{5,15}, m_{6,15}, m_{7,15}$ $m_{8,15}, m_{9,15}, m_{10,15}, m_{11,15}$ $m_{12,15}, m_{13,15}, m_{14,15}, m_{15,15}$

CHAPTER 8

PERFORMANCE COMPARISONS BETWEEN HOW AND BINARY HYPERCUBE SYSTEMS

In this section we compare the communications capabilities of 2-D HOW systems and hypercubes. We consider communications under model-3 which permits a processor to send out different values simultaneously using different channels, because this is often actually the case with real systems. We assume that t_w is one unit of time and that $t_s = t_c = 0$ in order to simplify the calculations.

The equations derived in the previous sections for 2-D HOW systems follow:

$$T_{one\_to\_all,3} = 2mt_w\lceil\frac{\sqrt{p}-1}{w}\rceil = O(m\frac{\sqrt{p}}{w})$$

$$T_{all\_to\_all,3} = mt_w(1+\sqrt{p})\lceil\frac{\sqrt{p}-1}{w}\rceil = O(m\frac{p}{w})$$

$$T_{one\_to\_all\_pers,3} = 2\lceil\frac{\sqrt{p}-1}{w}\rceil mt_w(\sqrt{p}-1) = O(m\frac{p}{w})$$

$$T_{all\_to\_all\_pers,3} = 2pm\lceil\frac{\sqrt{p}-1}{w}\rceil t_w = O(m\frac{p^{3/2}}{w})$$

The equations for hypercube systems are:

$$T_{one\_to\_all} = m(\log p)t_w = O(m\log p)$$

$$T_{all\_to\_all} = m(p-1)t_w = O(mp)$$

$$T_{one\_to\_all\_pers} = m(p-1)t_w = O(mp)$$

$$T_{all\_to\_all\_pers} = (\log p)\ (\frac{p}{2}m)t_w = O(mp\log p)$$

It becomes obvious that HOW systems perform asymptotically better than hypercubes in one-to-all personalized communication and all-to-all broadcasting. In the other two types of communications, the result of the comparison depends on the value of w. The remaining figures show comparative results for practical cases, where the suitability of HOW systems for very high performance computing is demonstrated further.

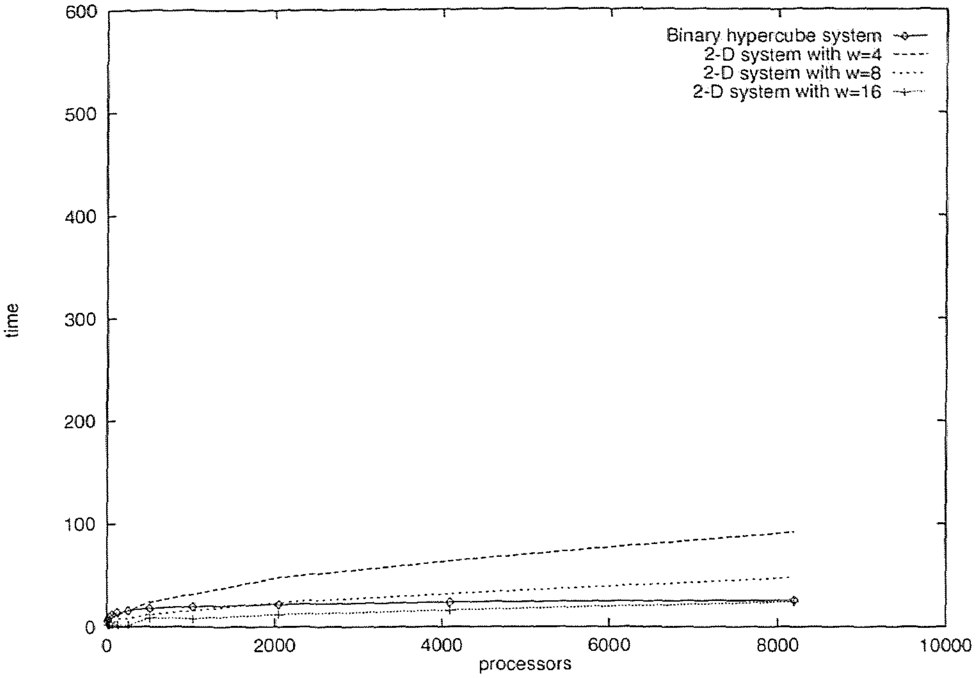


Figure 8.1 Comparisons between HOW and binary hypercube systems for one-to-all broadcasting with message size $m = 2$ words.

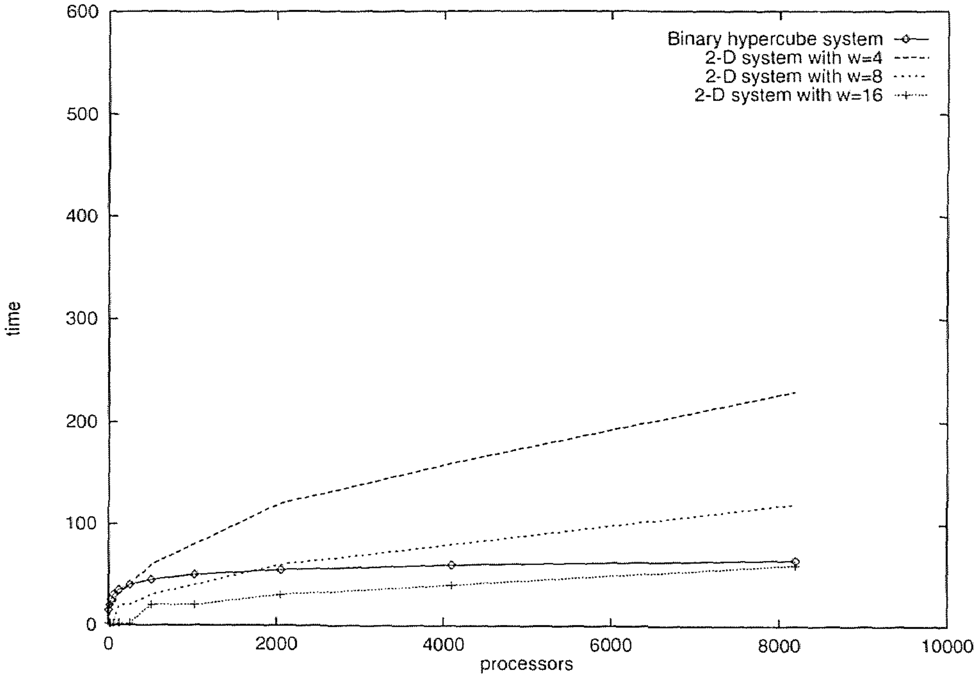


Figure 8.2 Comparisons between HOW and binary hypercube systems for one-to-all broadcasting with message size $m = 5$ words.

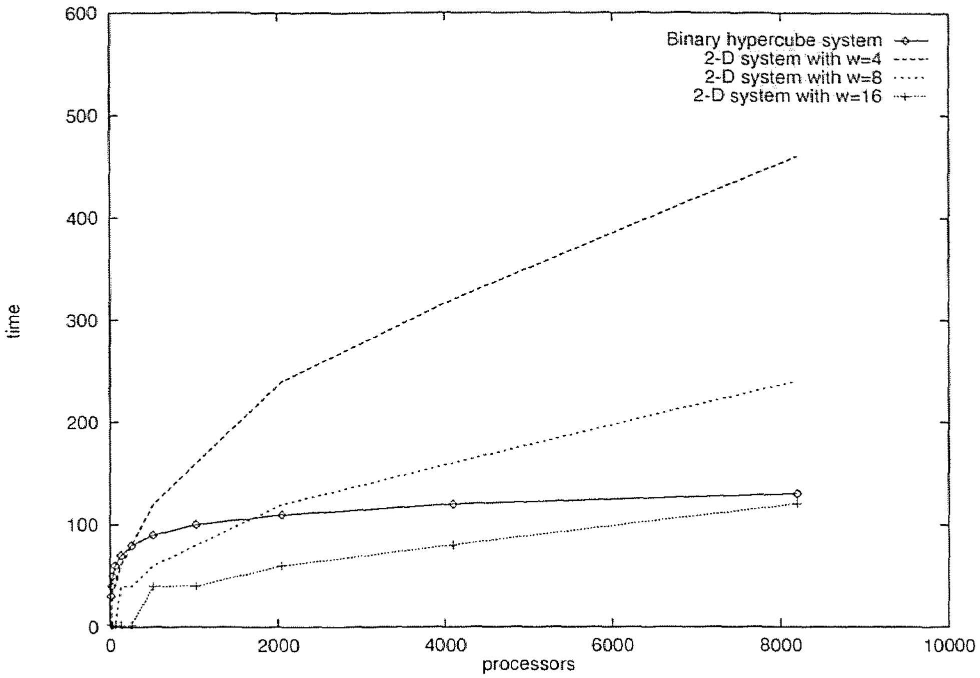


Figure 8.3 Comparisons between HOW and binary hypercube systems for one-to-all broadcasting with message size $m = 10$ words.

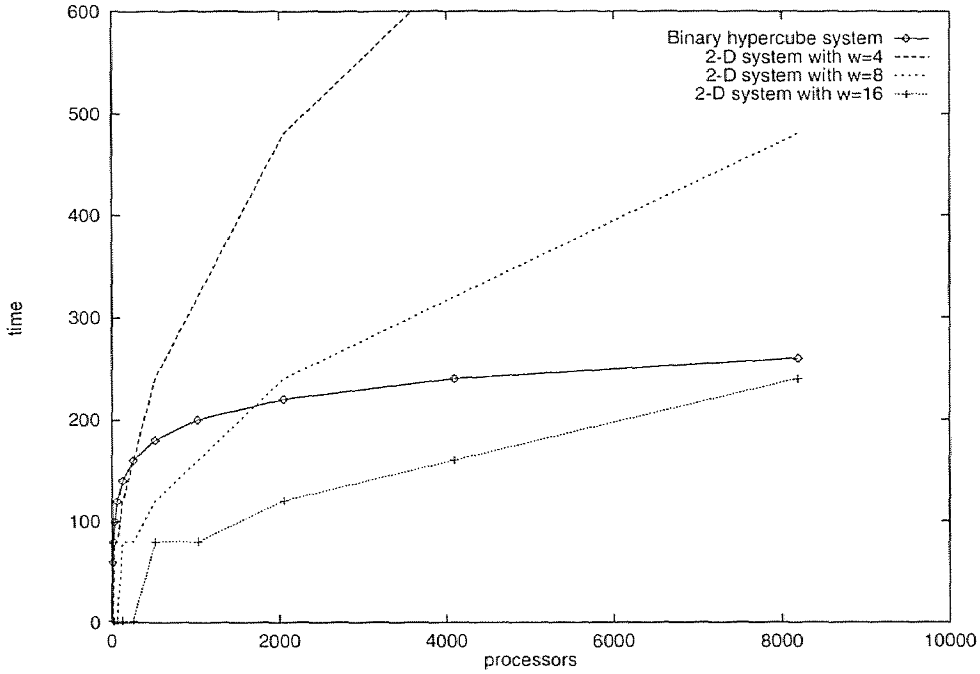


Figure 8.4 Comparisons between HOW and binary hypercube systems for one-to-all broadcasting with message size $m = 20$ words.

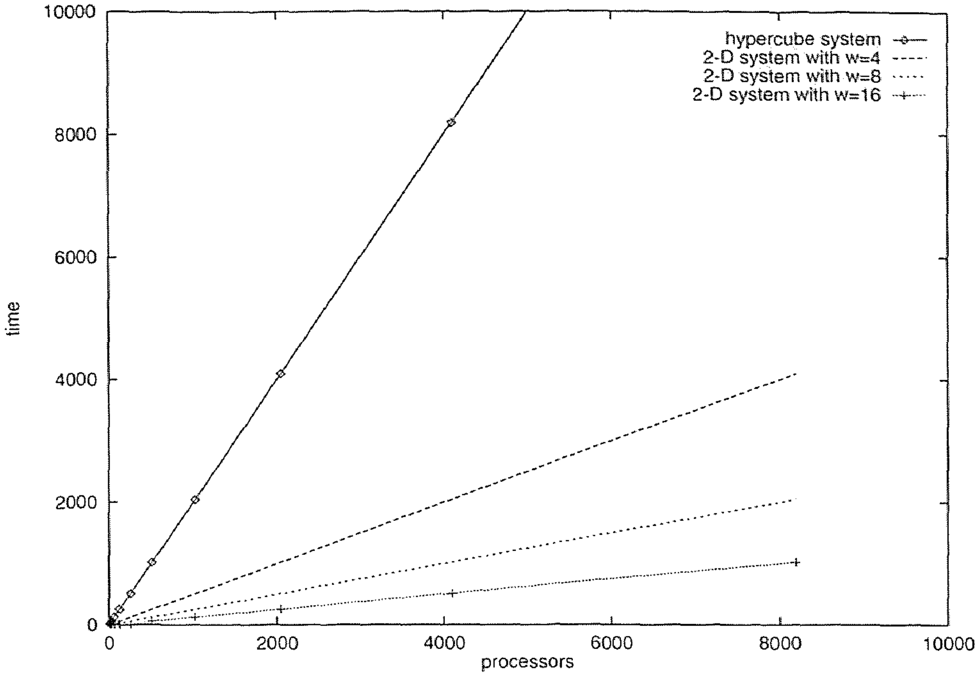


Figure 8.5 Comparisons between HOW and binary hypercube systems for all-to-all broadcasting with message size $m = 2$ words.

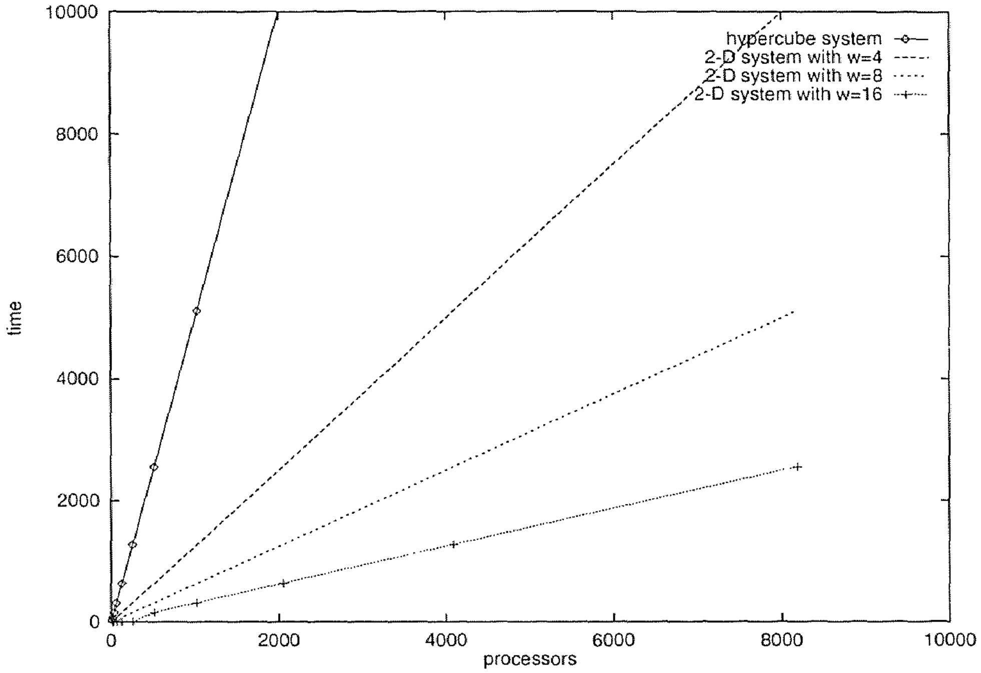


Figure 8.6 Comparisons between HOW and binary hypercube systems for all-to-all broadcasting with message size $m = 5$ words.

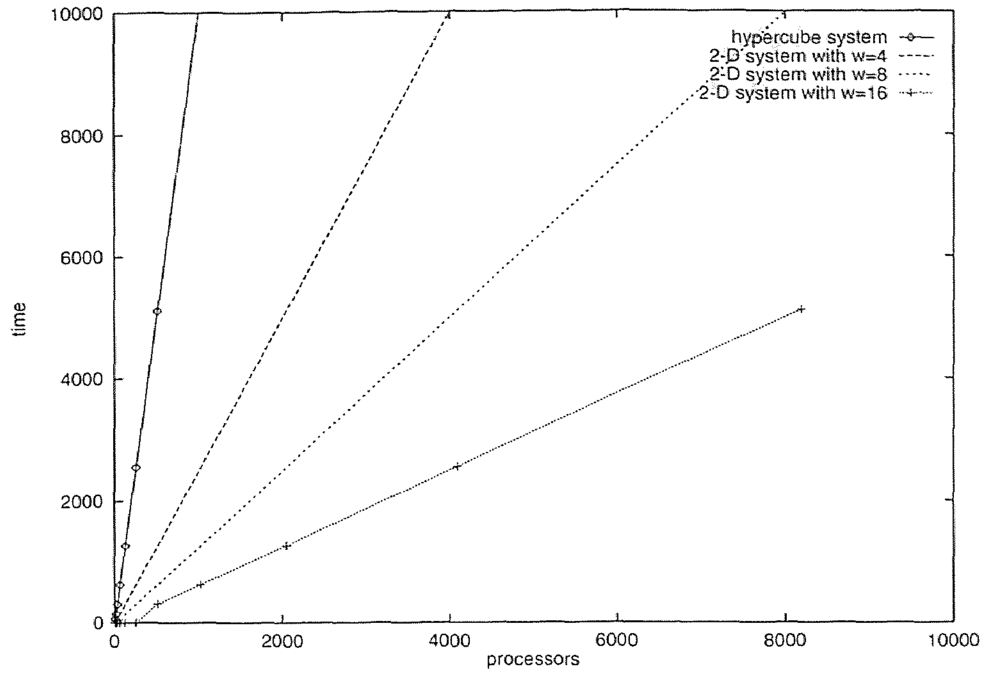


Figure 8.7 Comparisons between HOW and binary hypercube systems for all-to-all broadcasting with message size $m = 10$ words.

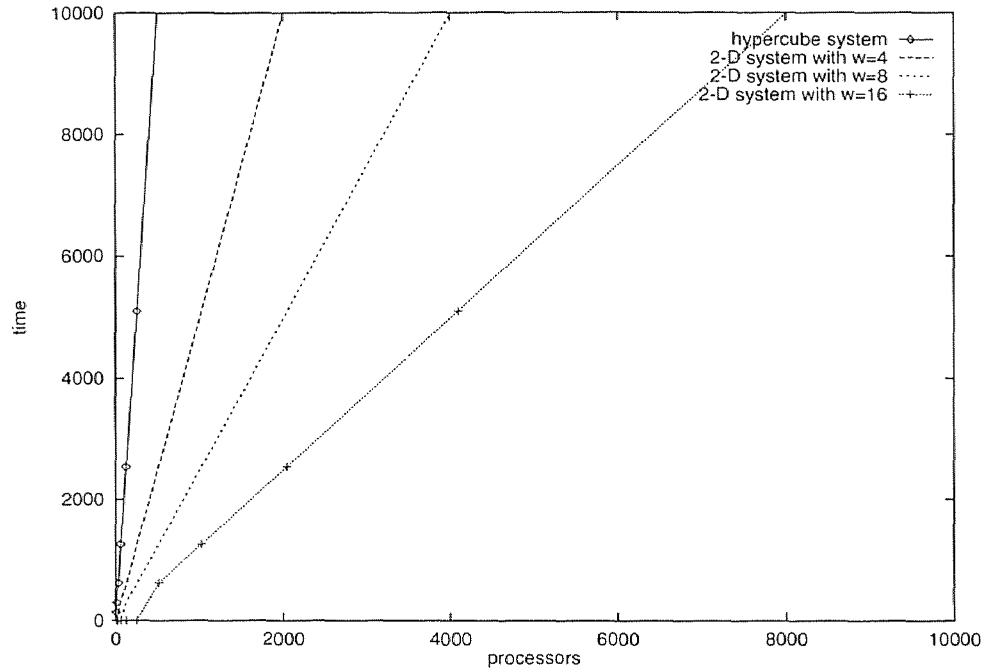


Figure 8.8 Comparisons between HOW and binary hypercube systems for all-to-all broadcasting with message size $m = 20$ words.

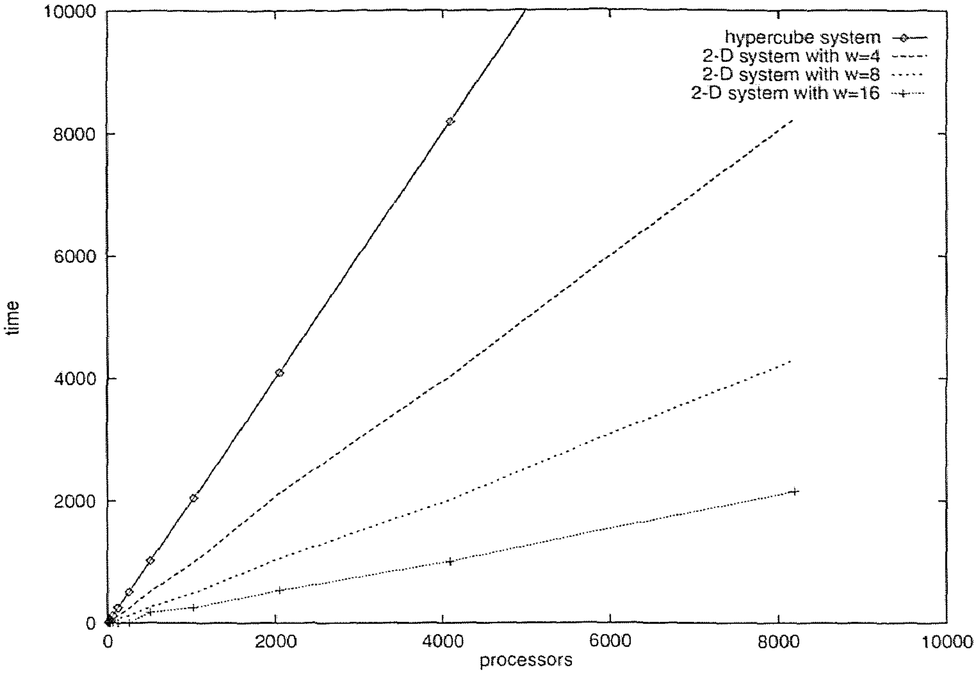


Figure 8.9 Comparisons between HOW and binary hypercube systems for one-to-all personalized communication with message size $m = 2$ words.

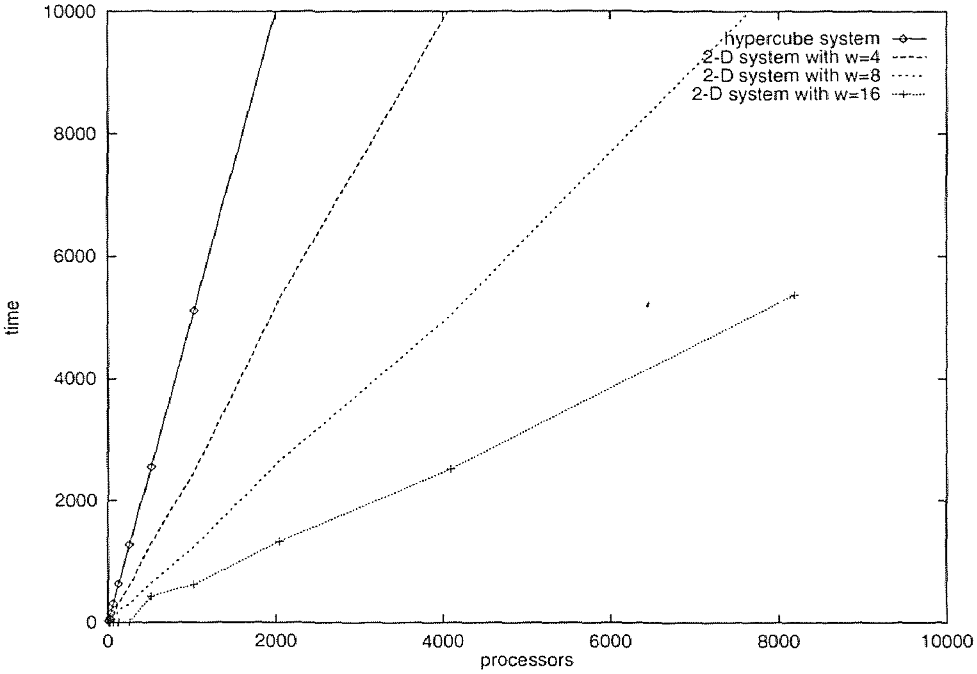


Figure 8.10 Comparisons between HOW and binary hypercube systems for one-to-all personalized communication with message size $m = 5$ words.

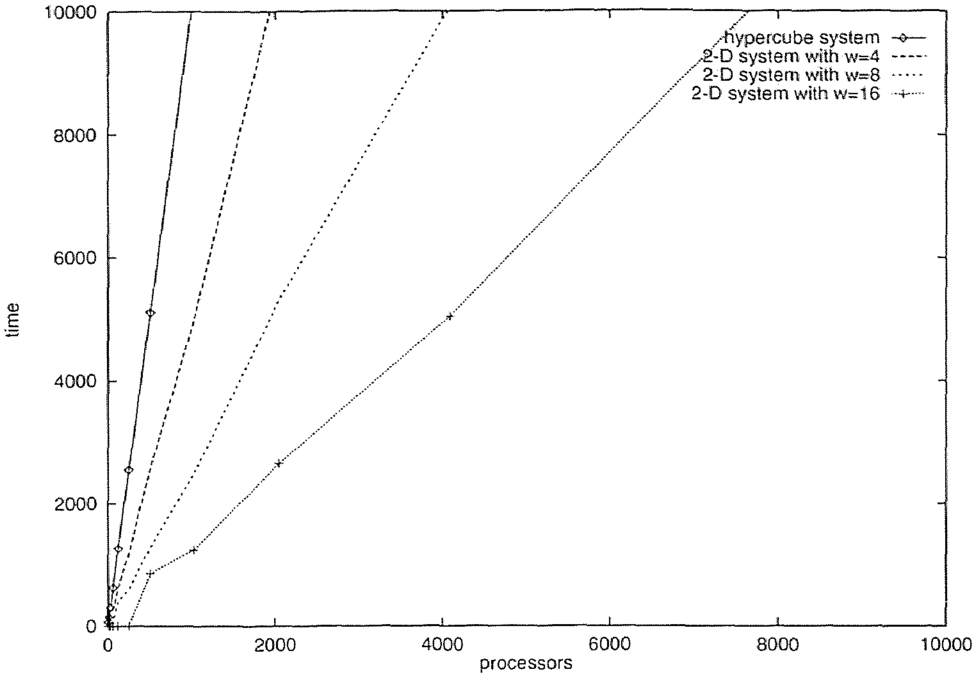


Figure 8.11 Comparisons between HOW and binary hypercube systems for one-to-all personalized communication with message size $m = 10$ words.

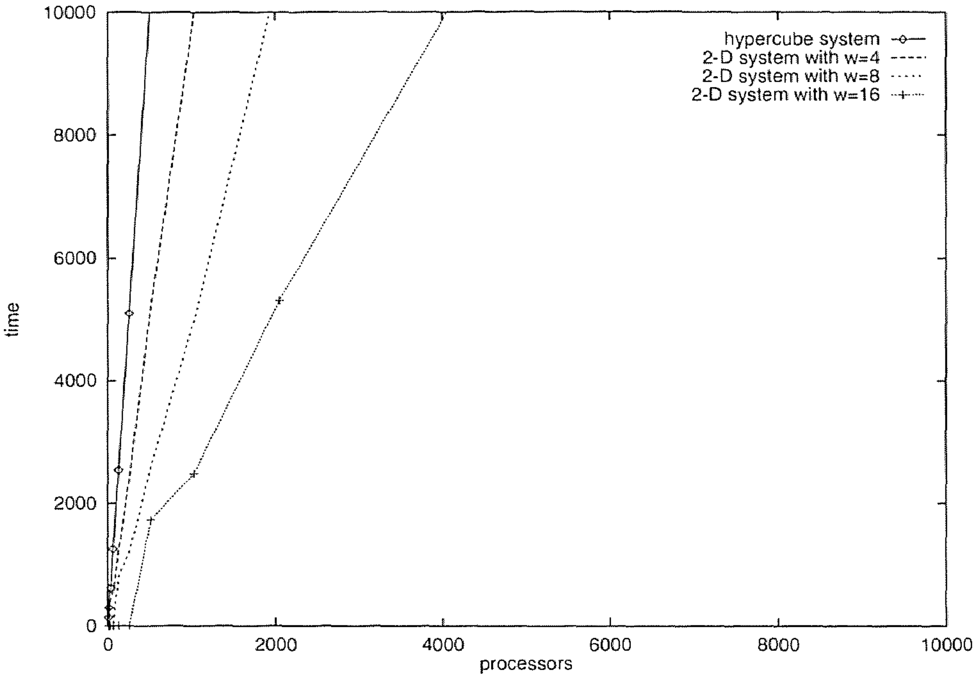


Figure 8.12 Comparisons between HOW and binary hypercube systems for one-to-all personalized communication with message size $m = 20$ words.

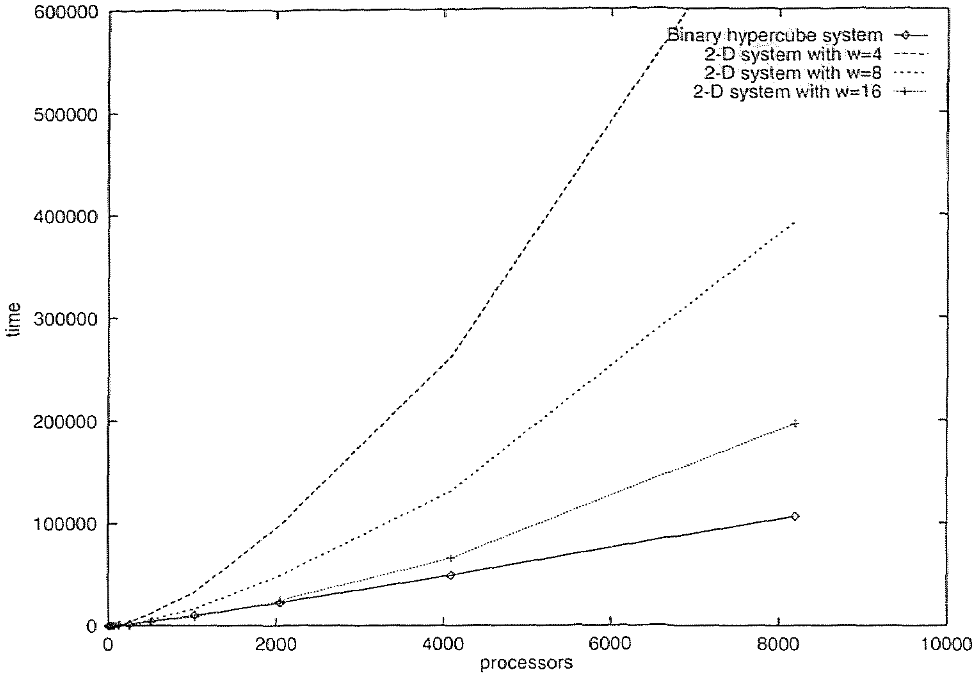


Figure 8.13 Comparisons between HOW and binary hypercube systems for all-to-all personalized communication with message size $m = 2$ words.

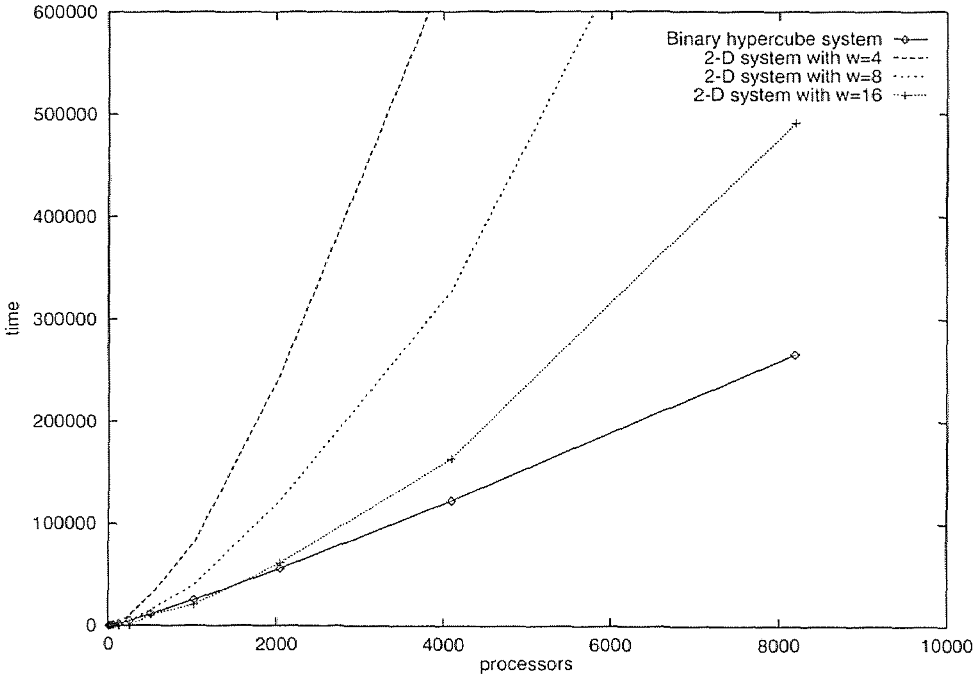


Figure 8.14 Comparisons between HOW and binary hypercube systems for all-to-all personalized communication with message size $m = 5$ words.

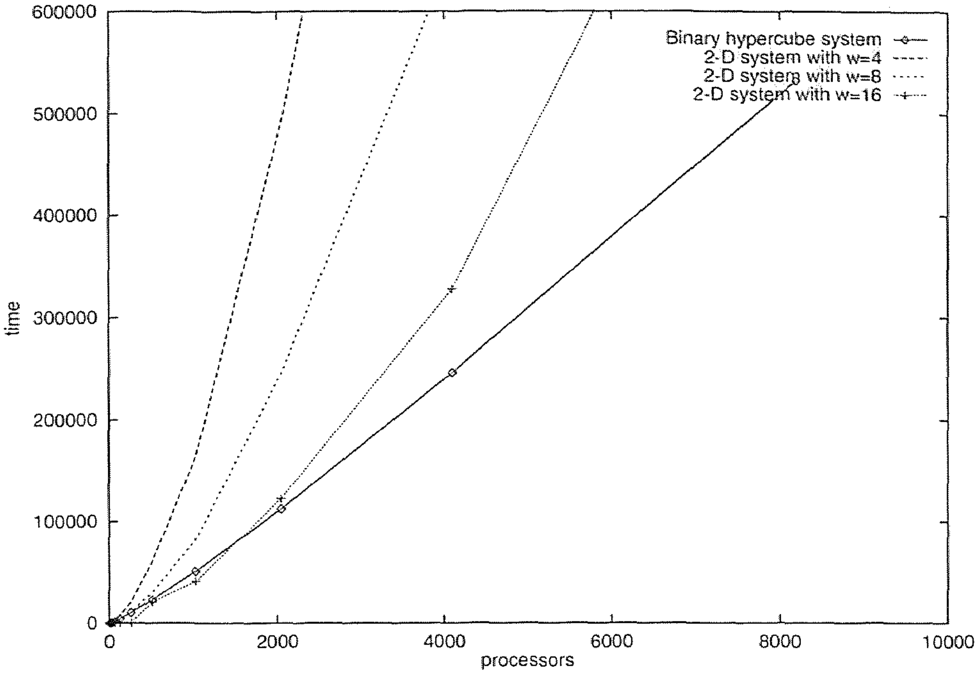


Figure 8.15 Comparisons between HOW and binary hypercube systems for all-to-all personalized communication with message size $m = 10$ words.

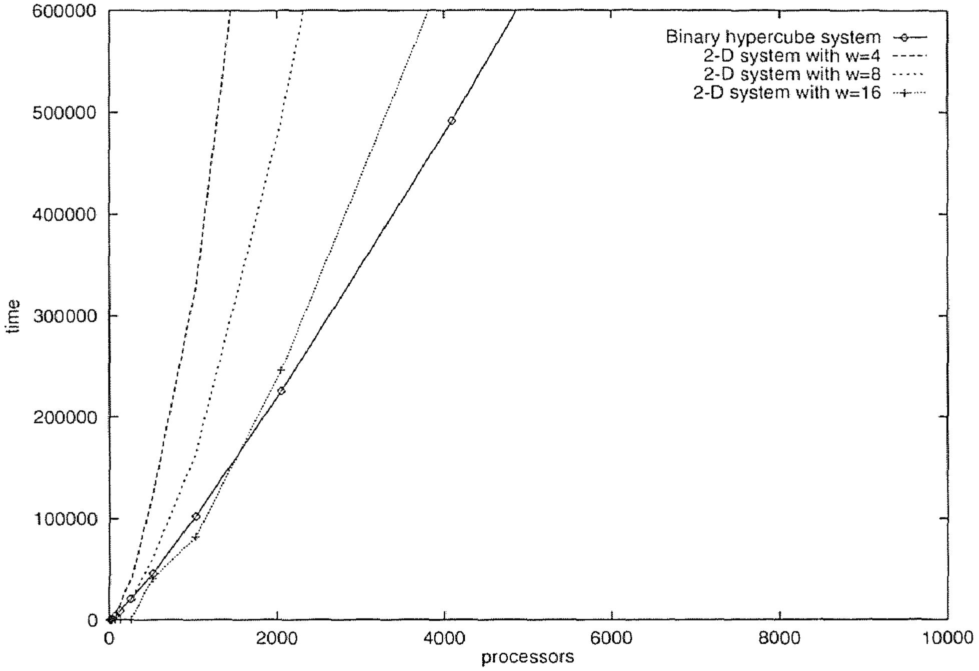


Figure 8.16 Comparisons between HOW and binary hypercube systems for all-to-all personalized communication with message size $m = 20$ words.

CHAPTER 9

PERFORMANCE COMPARISONS BETWEEN HOW AND GENERALIZED HYPERCUBE SYSTEMS

In this section we compare the communications capabilities of 2-D HOW systems and generalized hypercubes. We consider communications under model-3 which permits a processor to send out different values simultaneously using different channels, because this is often actually the case with real systems. We assume that t_w is one unit of time and that $t_s = t_c = 0$ in order to simplify the calculations.

The equations derived in the previous sections for 2-D HOW systems follow:

$$T_{one\_to\_all,3} = 2mt_w \lceil \frac{\sqrt{p}-1}{w} \rceil = O(m\frac{\sqrt{p}}{w})$$

$$T_{all\_to\_all,3} = mt_w(1+\sqrt{p}) \lceil \frac{\sqrt{p}-1}{w} \rceil = O(m\frac{p}{w})$$

$$T_{one\_to\_all\_pers,3} = 2 \lceil \frac{\sqrt{p}-1}{w} \rceil mt_w(\sqrt{p}-1) = O(m\frac{p}{w})$$

$$T_{all\_to\_all\_pers,3} = 2pm \lceil \frac{\sqrt{p}-1}{w} \rceil t_w = O(m\frac{p^{3/2}}{w})$$

The generalized hypercube is special case of our HOW system. The equations for generalized hypercube systems (or 1-D fully connected HOW subsystem) are:

$$T^{full}_{one\_to\_all,3} = 2mt_w = O(m)$$

$$T^{full}_{all\_to\_all,3} = m(1+\sqrt{p})t_w = O(m\sqrt{p})$$

$$T^{full}_{one\_to\_all\_pers,3} = m\sqrt{p}t_w = O(m\sqrt{p})$$

$$T^{full}_{all\_to\_all\_pers,3} = m(p-1)t_w = O(mp)$$

Table 9.1 Cost comparison between the $HOW(\sqrt{p}, w, 2)$ and $GH(\sqrt{p}, 2)$ systems.

	Cost Comparison			
System	one-to-all broadcasting	all-to-all broadcasting	one-to-all-pers. communication	all-to-all-pers. communication
$HOW(\sqrt{p}, w, 2)$	$O(mpw)$	$O(mp^{3/2}w)$	$O(mp^{3/2}w)$	$O(mp^2w)$
$GH(\sqrt{p}, 2)$	$O(mp^{3/2})$	$O(mp^2)$	$O(mp^2)$	$O(mp^{5/2})$

The remaining figures show comparisons between generalized hypercubes and HOW systems. It becomes obvious that generalized hypercube systems perform better than HOW systems from the communication time point of view. But the generalized hypercube has a fundamental design disadvantage. It has very large wiring complexity, as demonstrated by its *bisection width*. The bisection width is defined as the minimum number of wires that must be cut to separate the network into two equal halves [23]. A very large bisection width makes the network impossible to build. The bisection width of the $GH(k, n)$ is $O(k^{n+1})$.

It is derived as follows. The bisection width of the $GH(\sqrt{p}, 1)$ is $\lceil \frac{\sqrt{p}}{2} \rceil * \lfloor \frac{\sqrt{p}}{2} \rfloor$, because when cutting the graph into two halves the edges which connect the left $\lceil \frac{\sqrt{p}}{2} \rceil$ nodes with the right $\lfloor \frac{\sqrt{p}}{2} \rfloor$ nodes must be removed. For the $GH(\sqrt{p}, 2)$ the bisection width is $\sqrt{p} * \lceil \frac{\sqrt{p}}{2} \rceil * \lfloor \frac{\sqrt{p}}{2} \rfloor = O(\sqrt{p}\, p) = O(p^{3/2})$ and for the $GH(k, n)$ the bisection width is $k^{n-1} * \lceil \frac{k}{2} \rceil * \lfloor \frac{k}{2} \rfloor = O(k^{n+1})$.

For the 1-D $HOW(\sqrt{p}, w, 1)$ the bisection width is $1+2+3+\cdots+w = \frac{w(w+1)}{2} = O(w^2)$. For the 2-D $HOW(\sqrt{p}, w, 2)$ the bisection width is $\frac{w(w+1)}{2} * \sqrt{p} = O(\sqrt{p}\, w^2)$.

Let us define the cost of an interconnection network as the product of the "communication time" and the "bisection width". This is a reasonable cost measure because we should like to achieve small communication time with a small system complexity. Table 9.1 shows the costs of the $HOW(\sqrt{p}, w, 2)$ and the $GH(\sqrt{p}, 2)$ for $\sqrt{p} \geq w$. This table also shows that reductions in the cost are proportional to reductions in the value of w and this leads to predictability. The $HOW(\sqrt{p}, w, 2)$ outperforms the $GH(\sqrt{p}, 2)$.

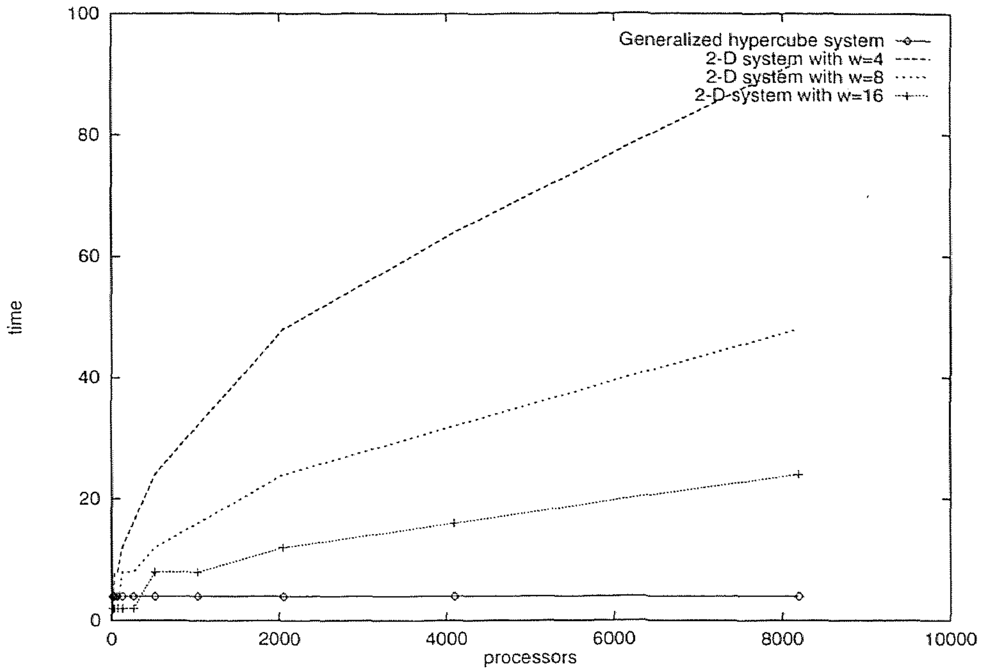


Figure 9.1 Comparisons between HOW and generalized hypercube systems for one-to-all broadcasting with message size $m = 2$ words.

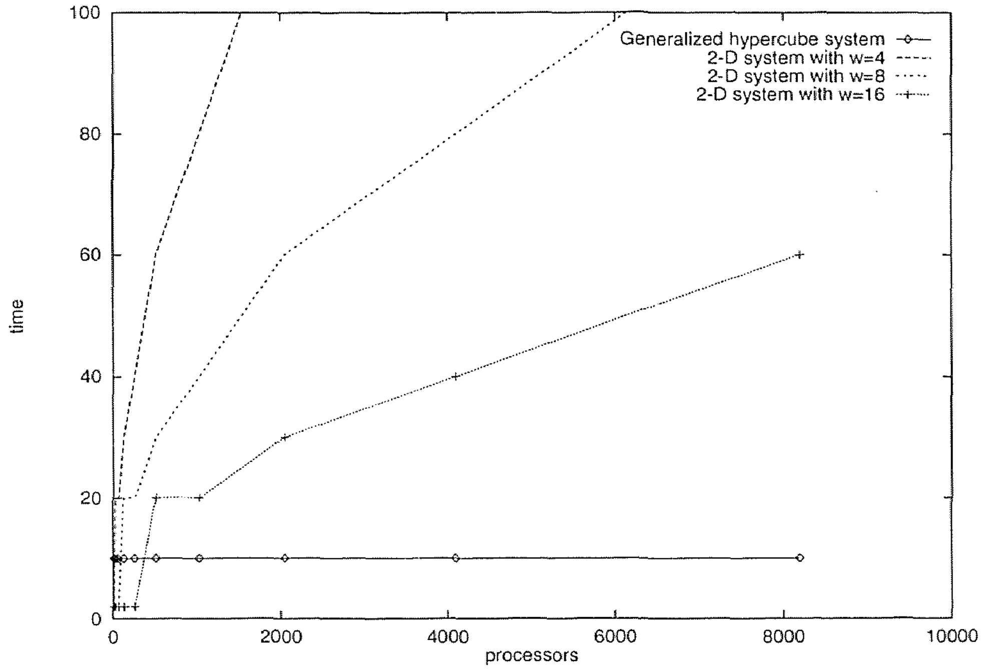


Figure 9.2 Comparisons between HOW and generalized hypercube systems for one-to-all broadcasting with message size $m = 5$ words.

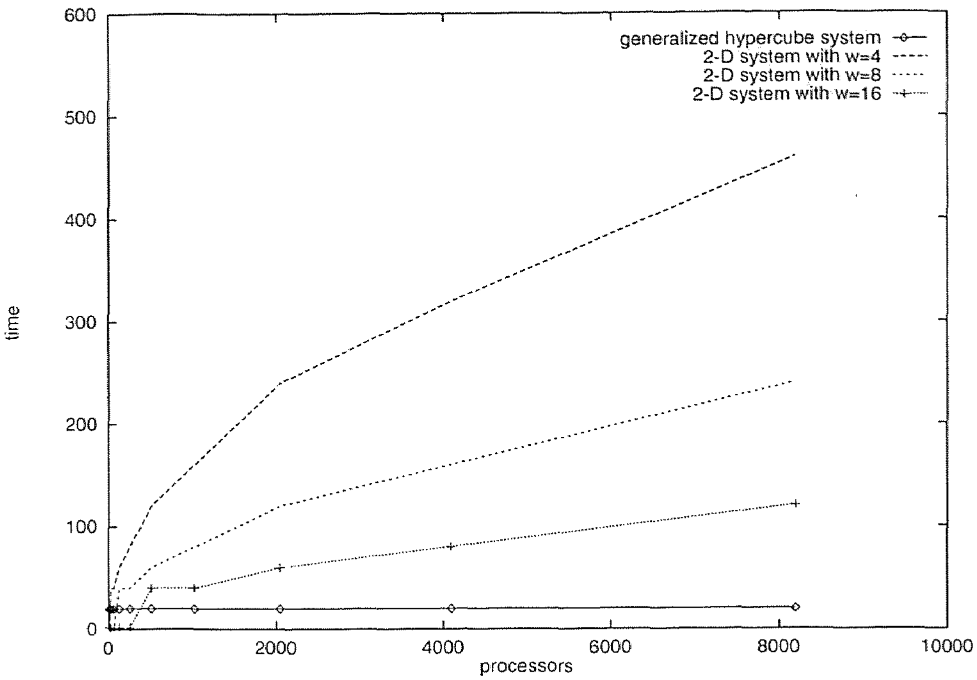


Figure 9.3 Comparisons between HOW and generalized hypercube systems for one-to-all broadcasting with message size $m = 10$ words.

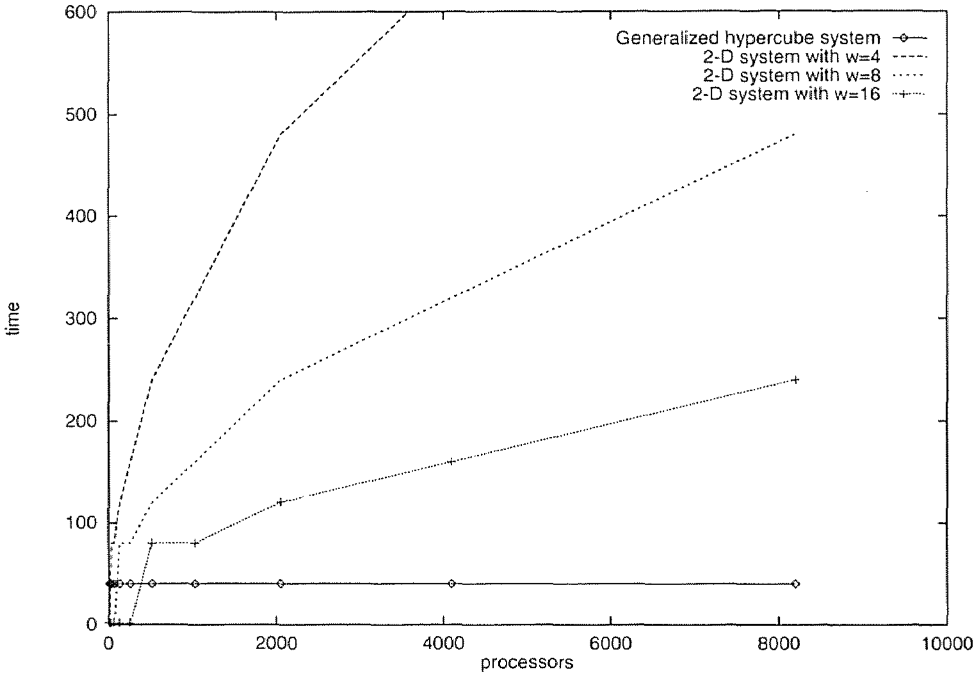


Figure 9.4 Comparisons between HOW and generalized hypercube systems for one-to-all broadcasting with message size $m = 20$ words.

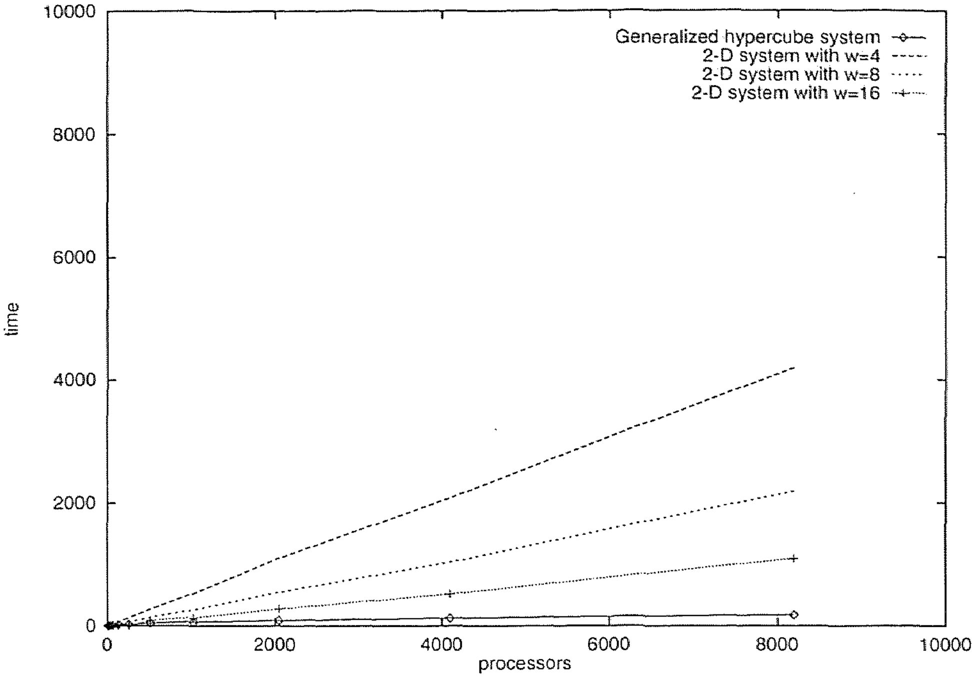


Figure 9.5 Comparisons between HOW and generalized hypercube systems for all-to-all broadcasting with message size $m = 2$ words.

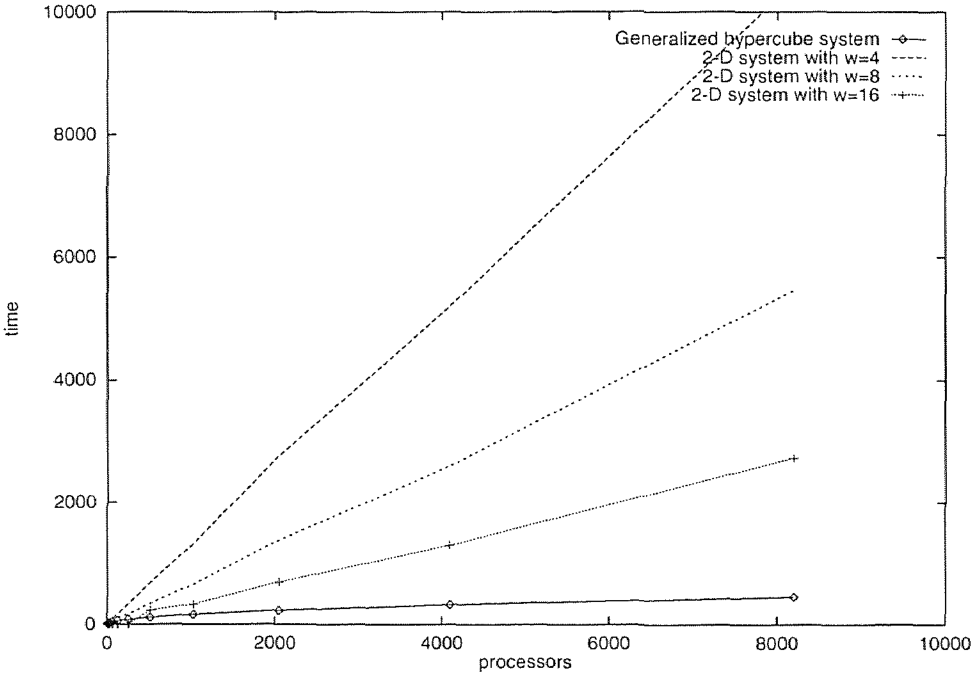


Figure 9.6 Comparisons between HOW and generalized hypercube systems for all-to-all broadcasting with message size $m = 5$ words.

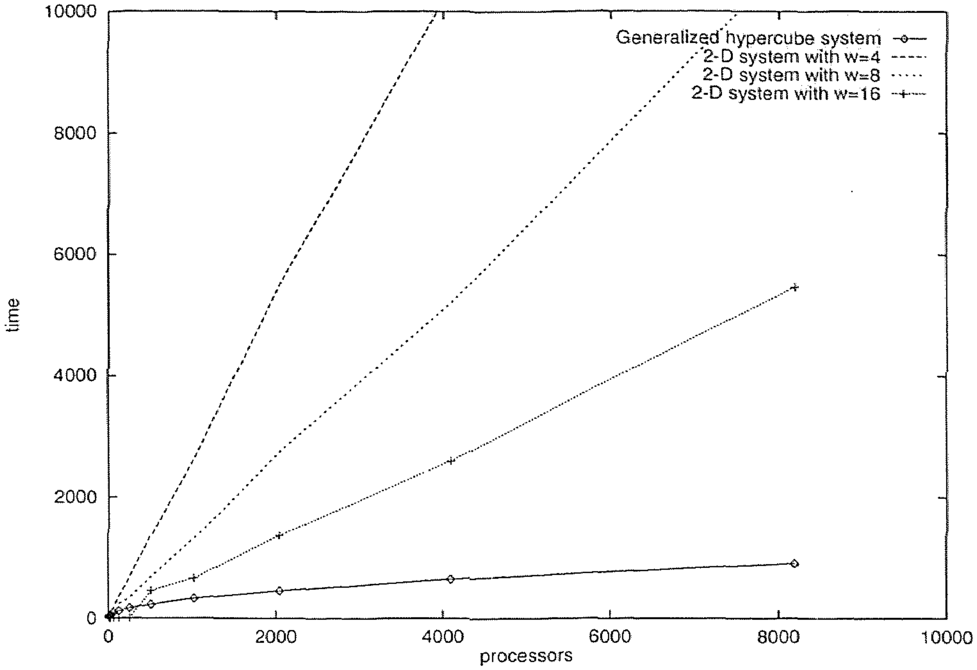


Figure 9.7 Comparisons between HOW and generalized hypercube systems for all-to-all broadcasting with message size $m = 10$ words.

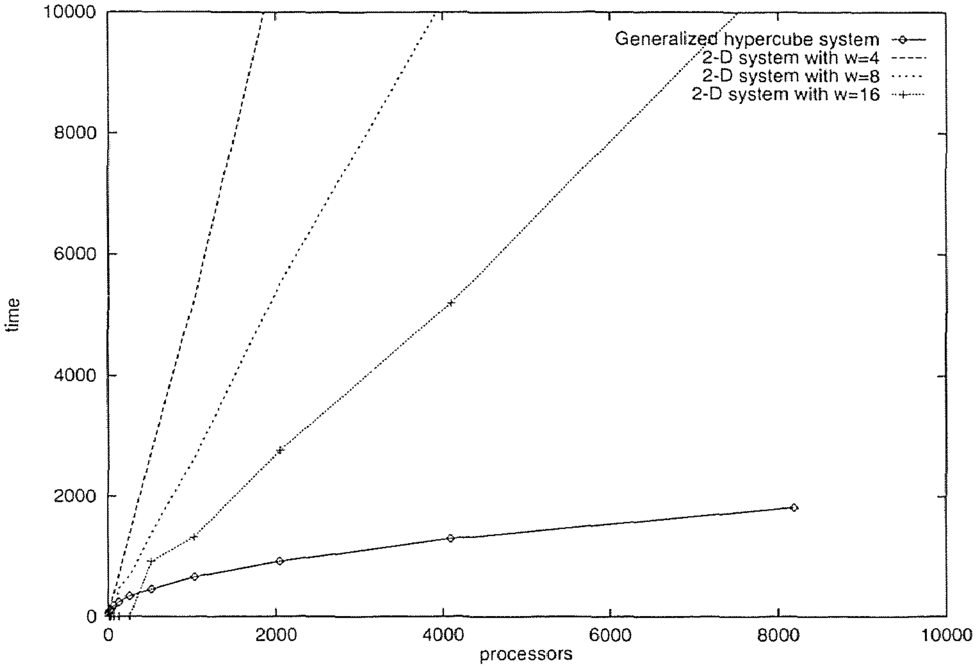


Figure 9.8 Comparisons between HOW and generalized hypercube systems for all-to-all broadcasting with message size $m = 20$ words.

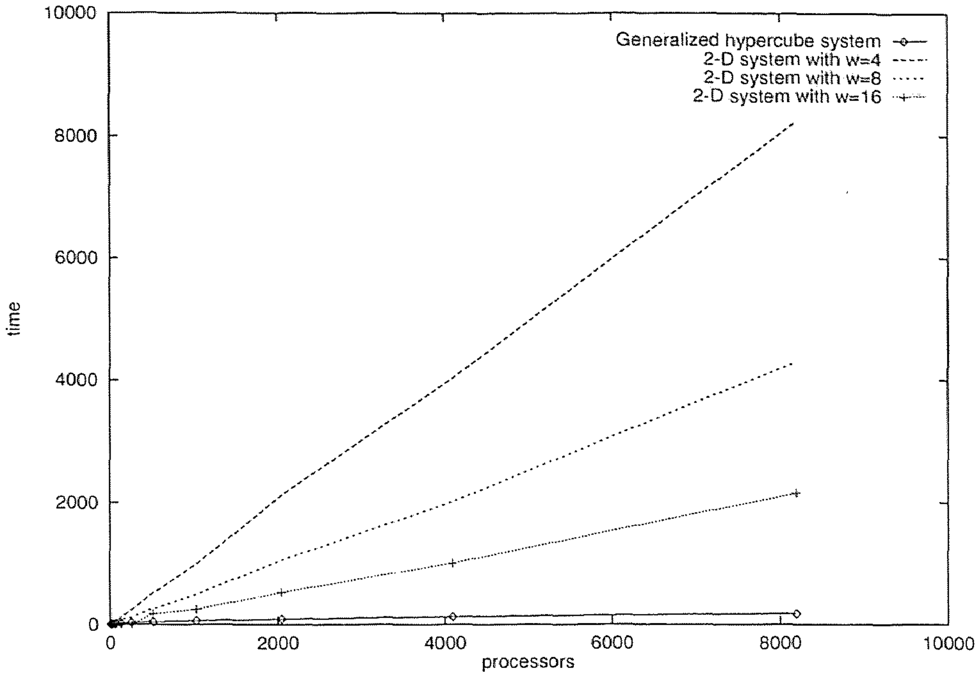


Figure 9.9 Comparisons between HOW and generalized hypercube systems for one-to-all personalized communication with message size $m = 2$ words.

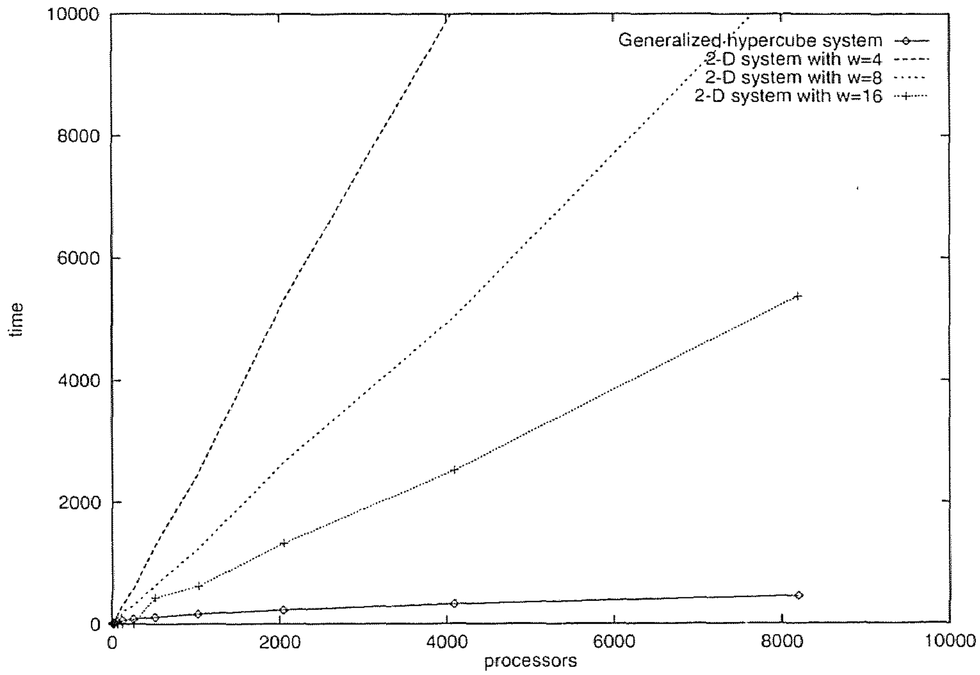


Figure 9.10 Comparisons between HOW and generalized hypercube systems for one-to-all personalized communication with message size $m = 5$ words.

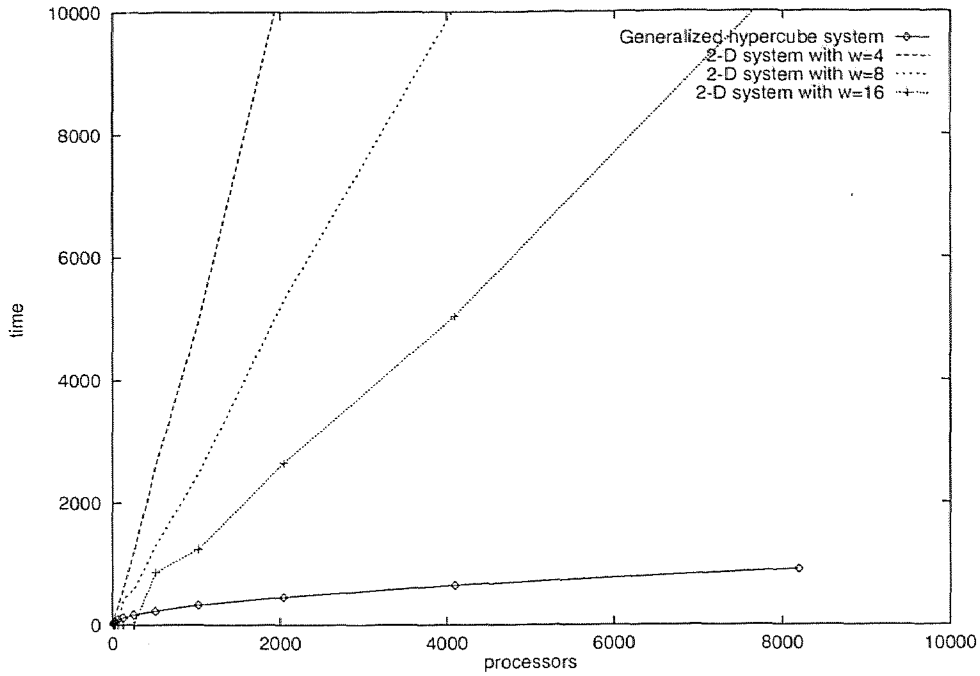


Figure 9.11 Comparisons between HOW and generalized hypercube systems for one-to-all personalized communication with message size $m = 10$ words.

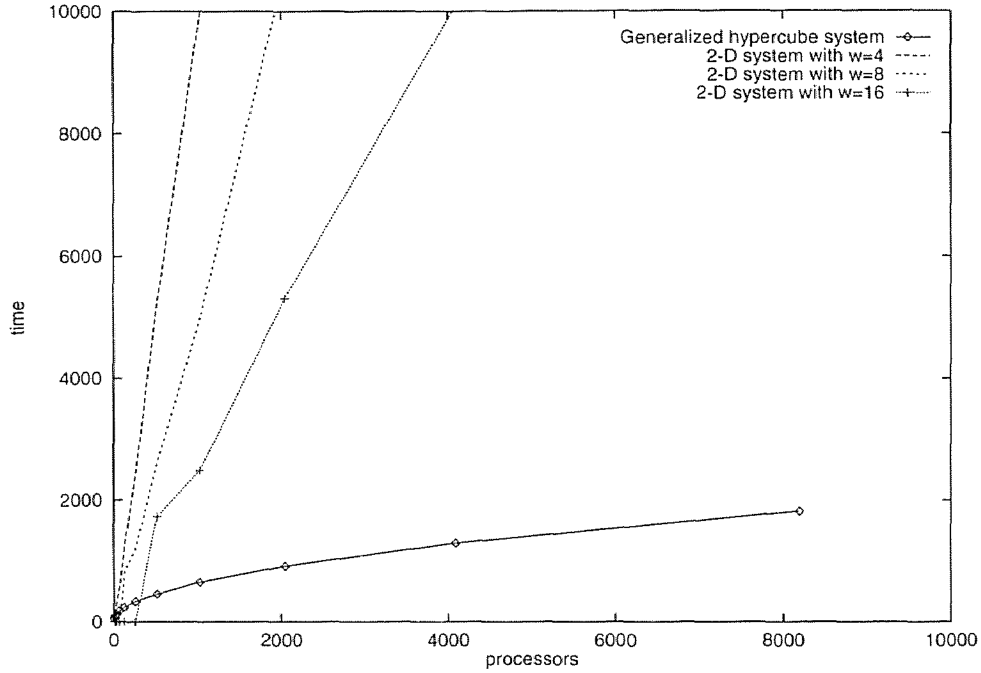


Figure 9.12 Comparisons between HOW and generalized hypercube systems for one-to-all personalized communication with message size $m = 20$ words.

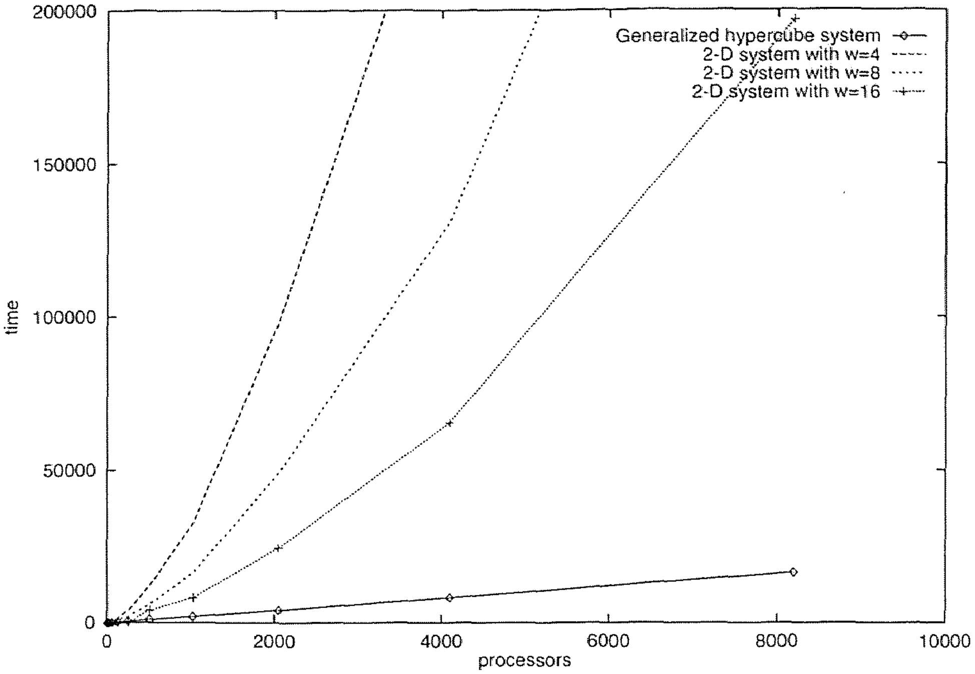


Figure 9.13 Comparisons between HOW and generalized hypercube systems for all-to-all personalized communication with message size $m = 2$ words.

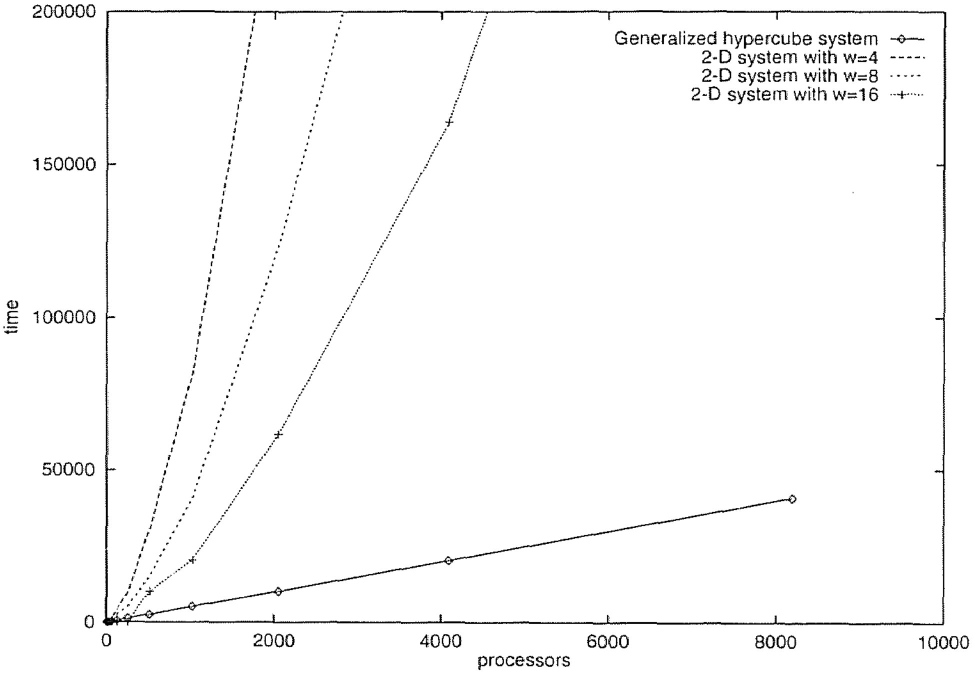


Figure 9.14 Comparisons between HOW and generalized hypercube systems for all-to-all personalized communication with message size $m = 5$ words.

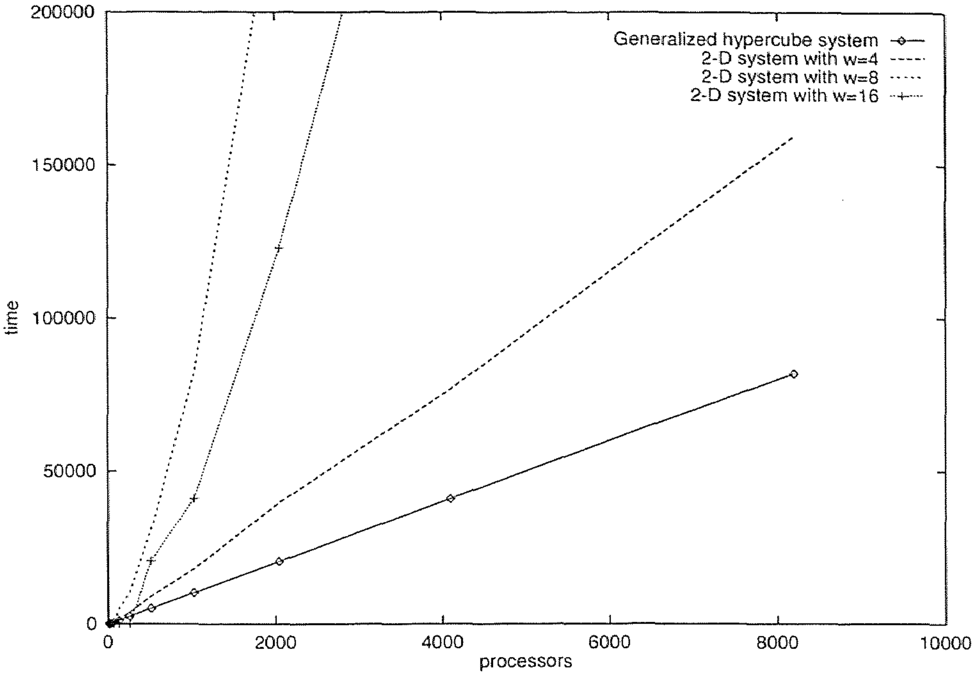


Figure 9.15 Comparisons between HOW and generalized hypercube systems for all-to-all personalized communication with message size $m = 10$ words.

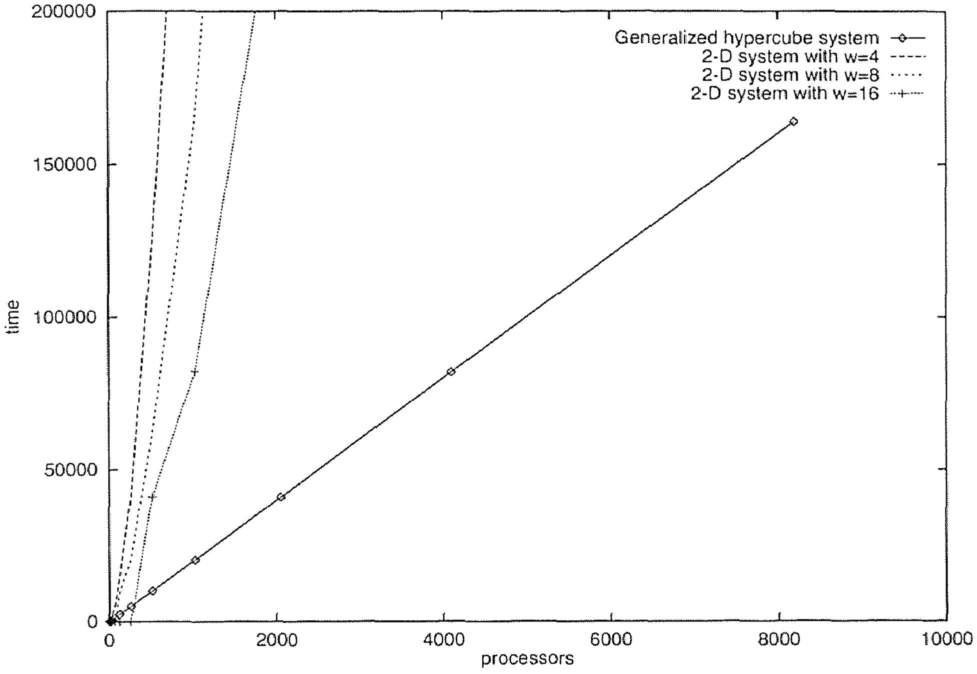


Figure 9.16 Comparisons between HOW and generalized hypercube systems for all-to-all personalized communication with message size $m = 20$ words.

CHAPTER 10

CONVERSION OF COMMUNICATIONS ALGORITHMS FOR GENERALIZED HYPERCUBES

Because the $GH_{k,n}$ is the building block of our HOW systems, it is worth trying to modify existing communications methods used for the $GH_{k,n}$. The following terms are used for constructing BST (Balanced Spanning Tree) and BSG (Balanced Spanning Subgraph) graphs [4].

DEFINITION 10.1. $GH_{k,n}$, an n-dimensional k-ary generalized hypercube, is an undirected graph of $N = k^n$ nodes, each one labeled by an n-digit number in radix k arithmetic. Each node v is connected to $n(k-1)$ other nodes with which it differs in only one digit; i.e., node $v = v_{n-1} \cdots v_{i+1} v_i v_{i-1} \cdots v_0$ is connected to nodes $v^{'} = v_{n-1} \cdots v_{i+1} v_i v_{i-1} \cdots v_0$ for all $0 \leq i \leq n-1$, $0 \leq v_i^{'} \leq k-1$, and $v_i \neq v_i^{'}$.

DEFINITION 10.2. The *translation* of a node v with respect to node s, denoted by $T_s(v)$, is defined to be the node $t = T_s(v)$, so that $t_i = (v_i + s_i) \bmod k$, for $0 \leq i \leq n-1$. The *inverse translation* of a node v with respect to node s, denoted by $T_s^{-1}(v)$, is defined to be the node $t = T_s^{-1}(v)$, so that $t_i = (v_i - s_i) \bmod k$, for $0 \leq i \leq n-1$.

DEFINITION 10.3. Consider the function r from the set $\{0, 1, \cdots, k-1\}$ to itself as follows:

$$r(i) = \begin{cases} 0 & \text{if } i = 0 \\ (i \; mod \; (k-1)) + 1 & \text{otherwise} \end{cases}$$

(Notice that r maps digit 0 to itself and the remaining digits as follows: $1- > 2- > 3- > \cdots - > k-1- > 1$.) The *rotation of a node* $v = v_{n-1} \cdots v_{i+1} v_i v_{i-1} \cdots v_0$, denoted by $R(v)$, is defined to be the node $v_{n-2} \cdots v_{i+1} v_i v_{i-1} \cdots v_0 r(v_{n-1})$.

DEFINITION 10.4. An ordered group of nodes, each one derived from its subsequent one cyclically by the application of a rotation, is called a *necklace*.

DEFINITION 10.5. The *binary correspondent* of a node v of $GH_{k,n}$ is the binary number obtained if we substitute each nonzero digit in v with the digit 1.

The generator node of a necklace is defined to be the largest among the nodes of the necklace that have the largest binary correspondent.

DEFINITION 10.6. The *displacement of a node* v, denoted by $D(v)$, is defined to be the minimum number of rotations that we have to apply on v in order to derive the generator of its necklace.

DEFINITION 10.7. The *period of a node* v, denoted by $P(v)$, is defined to be the number of nodes contained in the necklace to which it belongs.

DEFINITION 10.8. An *unfolded necklace* is an ordered group of exactly $n(k-1)$ nodes, not necessarily distinct, each one obtained from it subsequent one cyclically by the application of a rotation.

DEFINITION 10.9. A shortest path *balanced spanning tree*, rooted at node 0^n (it represents n zeros) of the $GH_{k,n}$ and denoted by BST_{0^n}, is defined through the following parent function. For node v, with $D(v) = i$, let p be the position of its first nonzero digit cyclically to the left of position $n-1-i$. Then the parent of this node in the BST_{0^n} is

$$parent^{BST_{0^n}}(v) = \begin{cases} \emptyset & \text{if } v = 0^n \\ v_{n-1}\cdots v_{p+1}0v_{p-1}\cdots v_0 & \text{if } v \neq 0 \end{cases}$$

DEFINITION 10.10. A shortest path *spanning subgraph*, rooted at node 0^n of the $GH_{k,n}$ and denoted by BSG_{0^n}, is defined through the following parent function. By $parent^{BSG_{0^n}}(v,i)$ we denote the parent of node v in the ith, where $0 \leq i \leq n(k-1)$, spanning tree of BSG_{0^n}. For node v with $D(v) = i \ mod \ P(v)$, $0 \leq i \leq n(k-1)$, let p_i be the position of its first nonzero digit cyclically to the left of position $n-1-i$:

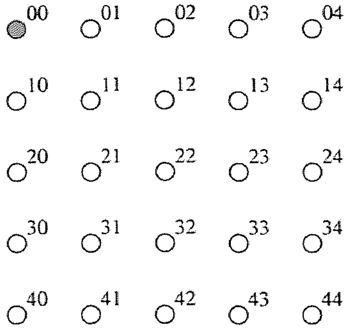
$$parent^{BSG_{0^n}}(v,i) = \begin{cases} \emptyset & \text{if } v = 0^n \\ v_{n-1}\cdots v_{p_i+1}0v_{p_i-1}\cdots v_0 & \text{if } v \neq 0 \end{cases}$$

Figures 10.1 and 10.3 show the BST_{0^2} of the $GH_{5,2}$ and the $GH_{8,2}$, respectively. The translation operation with respect to node s is applied to all the nodes of the BST_{0^n} to obtain the BST_s rooted at any node s.

Using a similar method, we can create the BST_{0^2} for the $HOW(p, w, 2)$ based on the BST_{0^2} for the $GH_{p,2}$, where $k = p$ in the $GH_{k,n}$. It is based on the fact that HOWs can be obtained from GHs by removing some edges. These steps are:

- Create the BST_{0^2} of the $GH_{p,2}$.
- Break non-connected edges in the $HOW(p, w, 2)$ which are connected in the $GH_{p,2}$, using the path which consists of all possible edges of window size w.
- If there is a conflict between intermediate nodes and leaf nodes (with the same parent), then the intermediate nodes stay where they are and the leaf nodes move to the next level.

Figures 10.2 and 10.4 show the BST_{0^2} of the $HOW(5, 3, 2)$ and the $HOW(8, 3, 2)$, respectively. Similarly, Figures 10.5 and 10.6 show the BST_{0^2} of the $HOW(8, 4, 2)$ and the $HOW(8, 5, 2)$, respectively. Shaded nodes in these figures show the procedure for the $GH_{8,2}$. According to [4], the one-to-all personalized communication consumes time $O\left(\frac{m(p-1)}{n(k-1)}\right)$ on the $GH_{k,n}$. For the $HOW(p, w, n)$, the modification of this communication procedure results in time $O\left(\frac{mp}{nw}\right)$. This is similar to what we also derived with our procedure in Chapter 6. Therefore, we do not elaborate further on the problem of modifying algorithms for the $GH_{k,n}$ from [4].



GH_5,2 k=5

d=1 01-> 10 -> 02 -> 20 -> 03 -> 30 -> 04 -> 40 ->01

d=2 11 -> 12 -> 22 -> 23 -> 33 -> 34 -> 44 -> 41 -> 11

13 -> 32 -> 24 -> 43 -> 31 -> 14 -> 42 -> 21 -> 13

The necklaces of GH_5,2

d=1 {40, 04, 30, 03, 20, 02, 10, 01}

d=2 {44, 34, 33, 23, 22, 12, 11, 41}

{42, 14, 31, 43, 24, 32, 13, 21}

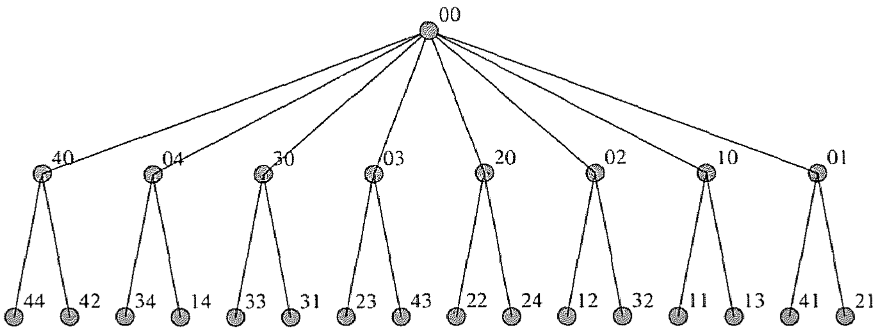


Figure 10.1 The spanning tree BST_{0^2} of the $GH_{5,2}$.

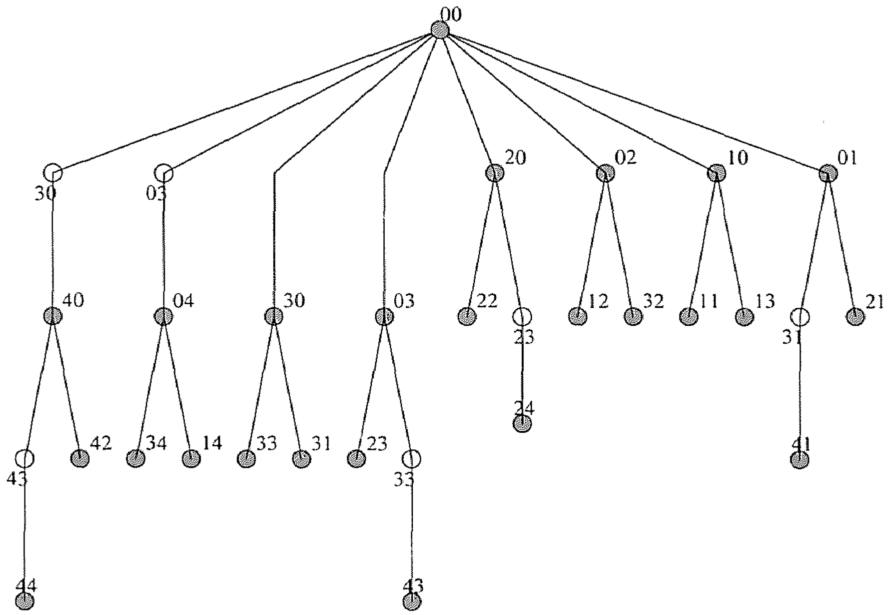
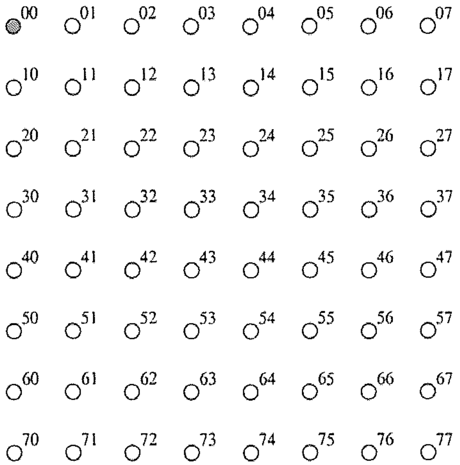


Figure 10.2 The spanning tree BST_{0^2} of the $HOW(5, 3, 2)$.



```
GH_8,2     k=8
d=1     01 -> 10 -> 02 -> 20 -> 03 -> 30 -> 04 -> 40 -> 05 -> 50 -> 06 -> 60 -> 07 -> 70
d=2     11 -> 12 -> 22 -> 23 -> 33 -> 34 -> 44 -> 45 -> 55 -> 56 -> 66 -> 67 -> 77 -> 71
        13 -> 32 -> 24 -> 43 -> 35 -> 54 -> 46 -> 65 -> 57 -> 76 -> 61 -> 17 -> 72 -> 21
        14 -> 42 -> 25 -> 53 -> 36 -> 64 -> 47 -> 75 -> 51 -> 16 -> 62 -> 27 -> 73 -> 31
        15 -> 52 -> 26 -> 63 -> 37 -> 74 -> 41

The necklaces of GH_8,2
d=1     {70, 07, 60, 06, 50, 05, 40, 04, 30, 03, 20, 02, 10, 01}
d=2     {77, 67, 66, 56, 55, 45, 44, 34, 33, 23, 22, 12, 11, 71}
        {76, 57, 65, 46, 54, 35, 43, 24, 32, 13, 21, 72, 17, 61}
        {75, 47, 64, 36, 53, 25, 42, 14, 31, 73, 27, 62, 16, 51}
        {74, 37, 63, 26, 52, 15, 41}
```

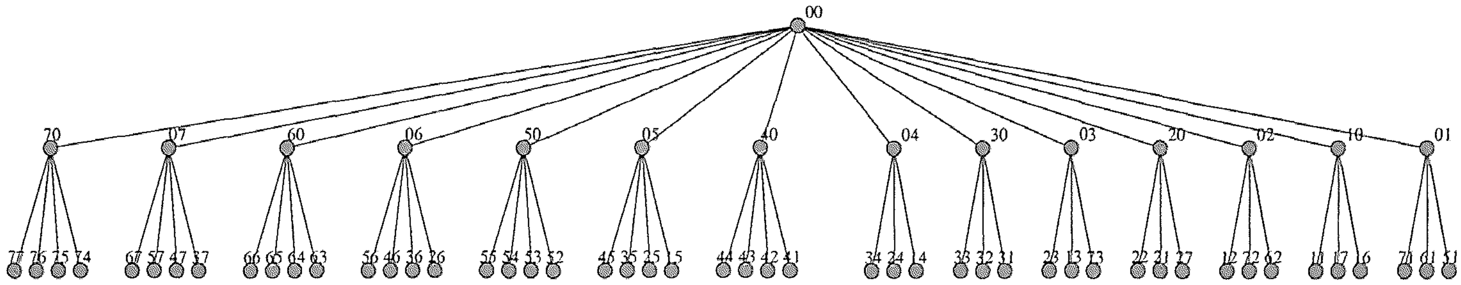


Figure 10.3 The spanning tree BST_{0^2} of the $GH_{8,2}$.

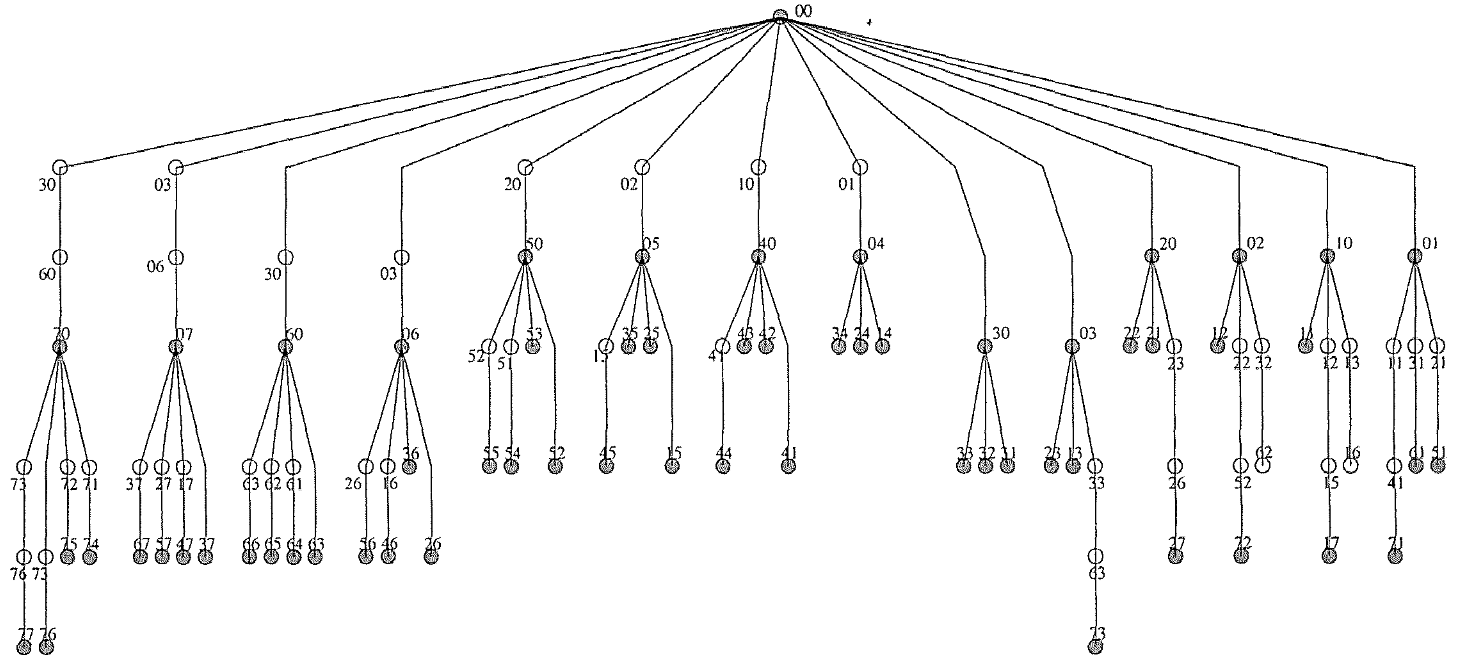


Figure 10.4 The spanning tree BST_{0^2} of the $HOW(8,3,2)$. Shaded nodes show the procedure for the GH.

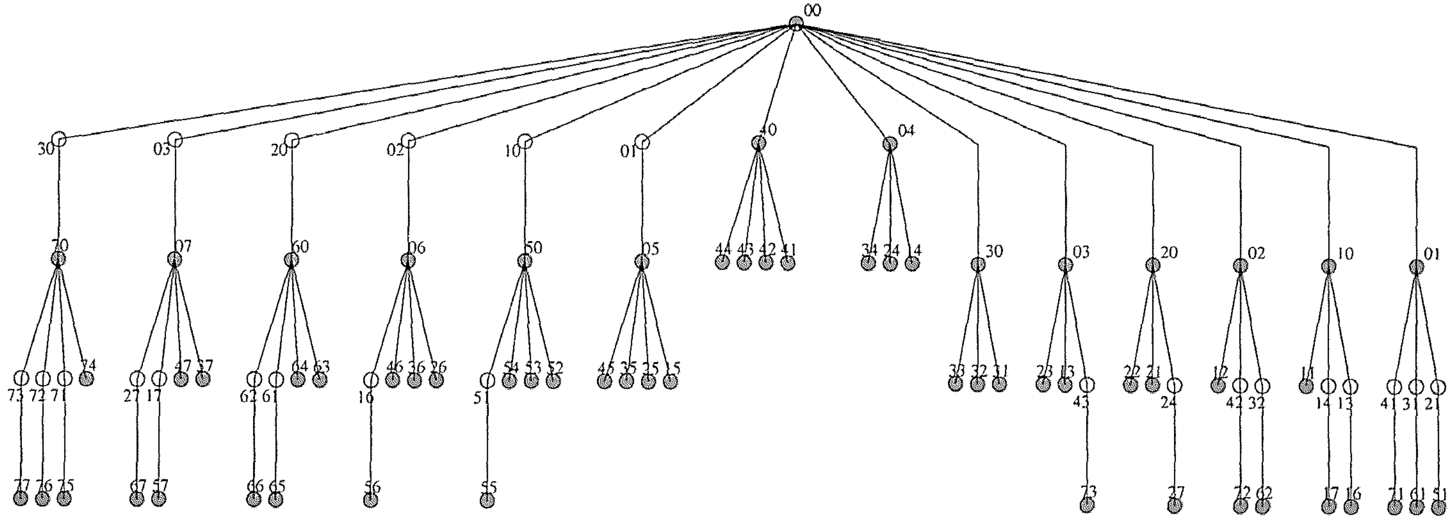


Figure 10.5 The spanning tree BST_{0^2} of the $HOW(8, 4, 2)$. Shaded nodes show the procedure for the GH.

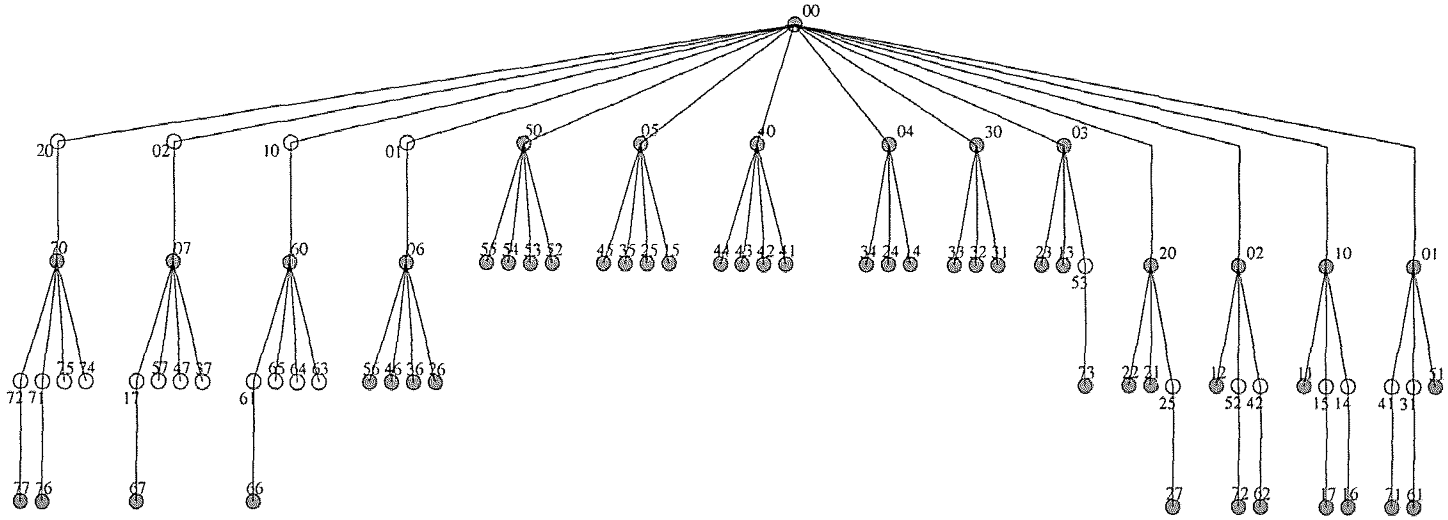


Figure 10.6 The spanning tree BST_{0^2} of the $HOW(8,5,2)$. Shaded nodes show the procedure for the GH.

CHAPTER 11

CONCLUSIONS AND FUTURE WORK

We introduced in this dissertation a new class of scalable architectures capable of very high performance. We also proposed algorithms for the implementation of various important communication operations, under frequently used communication models. We finally compared the performance of this class of architectures with that of the hypercube for the aforementioned communication operations. Our results show that not only are our architectures scalable and feasible with current technology, but also they perform better than the hypercube for several highly demanding communication operations. Of course, HOW systems perform outstandingly better than the currently popular torus systems, because of their much better topological properties.

Further work is needed on HOW systems with wrap-around connections, and on embeddings and communications operations on n-D HOW systems. Also, data reduction operations should be studied on 2-D and n-D HOW systems.

APPENDIX A

SIMULATION FOR ALL-TO-ALL PERSONALIZED COMMUNICATION ON 1-D HOWS

In all-to-all personalized communication, also known as *total exchange*, each processor sends a distinct message of size m to every other processor. It involves a lot of message transfers. We will not necessarily derive the most efficient procedure here, because such a procedure can be of a very complex nature. We present a simple procedure that comprises two stages. The basic idea here is that the first stage is initialization in which every processor exchanges related messages with its connected neighbors. The second stage is for sending related messages using the longest channel, when they are available.

The simulation code is

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

static num_of_nodes=12;
static window_size=3;

typedef struct _msg {
    int src;
    int dest;
} msg;

typedef struct _node {
    int number;                 /* node number */
    int tbl_size;
    int index;
    msg **table;
} node;

static node *all_nodes_1;
static node *all_nodes_2;
static node *current_state, *next_state;
static step;
static msg  *new_msg  (int src, int dest);
static void init_node (node *p, int n);
static void sort_node (node *p);
static void sort_all_node(void);
static void copy_all_node();
static void add_msg   (node *pNode, msg *pM);
static msg  *get_msg  (node *pNode, int i);
static void del_msg   (node *pNode, msg *pM);
static void init_all  (void);
static void print_all (void);

static void exchange_direct_node(node *p1, node *p2);

static int get_rightmost_msg (node *p,
                              msg *msg_vector[window_size]);
static int get_leftmost_msg  (node *p,
                              msg *msg_vector[window_size]);

static        msg **msg_array;
static        int *node_used;
main(int argc, char **argv)
{
    int i;
    int w;
    node *pNode1;
    node *pNode2;
    int done;

    if (argc >= 2)
        num_of_nodes = atoi(argv[1]);

    if (argc >=3)
        window_size = atoi(argv[2]);

    init_all();

    step = 0;
    print_all();

    /*first step, exchange all nodes within window_size*/
    for (i =0 ; i < num_of_nodes ; i++) {
        pNode1 = current_state+i;
        for (w = 1 ; w <= window_size; w++) {
```

```
        if ( i + w < num_of_nodes) {
            pNode2 = current_state + i+w;
            exchange_direct_node(pNode1, pNode2);
        }
    }

}

sort_all_node();
step++;
print_all();

msg_array = (msg **)malloc(sizeof(msg*)*window_size);
node_used = (int *)malloc(sizeof(int)*window_size);
while (1) {
  int dest;
  done = 1;

  copy_all_node(current_state, next_state);
  /* send msg to right */
  for (i = 0; i < num_of_nodes; i++) {
    if (get_rightmost_msg(current_state + i, msg_array)) {

      memset(node_used, 0, sizeof(int)*window_size);
      /* first try destination already within window */
      for (w = 0; w < window_size ; w++) {
        if (!msg_array[w])
            continue;
        if (msg_array[w]->dest <= i+window_size) {
          /* already with window size */
          if (!node_used[msg_array[w]->dest - i-1]) {
            del_msg(next_state + i, msg_array[w]);
            add_msg(next_state + msg_array[w]->dest,
                    msg_array[w]);
            node_used[msg_array[w]->dest - i-1] = 1;
          } else {
            int ww = w;
            while (ww < window_size) {
              if (node_used[window_size - ww -1]) {
                ww++;
                continue;
              }
              dest = i + window_size - ww;
              if (msg_array[w]->dest < dest) {
                ww++;
                continue;
              }
              if (dest < num_of_nodes) {
                del_msg(next_state + i,
                msg_array[w]);
                add_msg(next_state + dest,
                        msg_array[w]);
                node_used[window_size - ww -1] = 1;
                break;
              }
            }
          }
        }
      }
      /*
       * then try the algorithm: longest destination using
       * longest w
       */
      for (w = 0; w < window_size ; w++) {
        if (!msg_array[w])
          break;
        if (msg_array[w]->dest > i+window_size) {
          int ww = w;
          while (ww < window_size) {
            if (node_used[window_size - ww -1]) {
              ww++;
              continue;
            }
            dest = i + window_size - ww;
            if (dest < num_of_nodes) {
              del_msg(next_state + i, msg_array[w]);
              add_msg(next_state + dest, msg_array[w]);
              node_used[window_size - ww -1] = 1;
            }
            break;
          }
        }
      }
    done = 0;
  }
}

/* send msg to left */
```

```
for (i = num_of_nodes-1; i >=0; i--) {
  if (get_leftmost_msg(current_state + i, msg_array)) {
    memset(node_used, 0, sizeof(int)*window_size);
    for (w = 0; w < window_size ; w++) {
      if (!msg_array[w])
        continue;
      if (msg_array[w]->dest >= i-window_size) {
        /* already with window size */
        if (!node_used[i - msg_array[w]->dest - 1]) {
          del_msg(next_state + i, msg_array[w]);
          add_msg(next_state + msg_array[w]->dest,
                  msg_array[w]);
          node_used[i - msg_array[w]->dest - 1] = 1;
        } else {
          int ww = w;
          while (ww < window_size) {
            if (node_used[window_size - ww -1]) {
              ww++;
              continue;
            }
            dest = i - (window_size - ww);
            if (msg_array[w]->dest > dest) {
              ww++;
              continue;
            }
            if ( dest >= 0) {
              del_msg(next_state + i, msg_array[w]);
              add_msg(next_state + dest, msg_array[w]);
              node_used[window_size - ww -1] = 1;
              break;
            }
          }

        }
      }
    }
    for (w = 0; w < window_size ; w++) {
      if (!msg_array[w])
        break;
      if (msg_array[w]->dest <  i-window_size) {
        int ww = w;
        while (ww < window_size) {
          if (node_used[window_size-ww-1]) {
            ww++;
```

```
                continue;
              }
              dest = i - (window_size - ww);
              if ( dest >= 0) {
                del_msg(next_state + i, msg_array[w]);
                add_msg(next_state + dest, msg_array[w]);
                node_used[window_size - ww -1] = 1;
              }
              break;
            }
          }
        }
        done = 0;
      }

    }

    if (done)
      break;

    pNode1 = next_state;
    next_state = current_state;
    current_state = pNode1;
    sort_all_node();
    step++;
    print_all();
  }
}

static void
exchange_direct_node(node *p1, node *p2)
{
    msg *pM;
    int i;

    /* send msg from p1, to p2 */
    for (i = 0; i < p1 -> index; i++) {
        if ( p1 -> table[i]->dest == p2 -> number) {
            pM = get_msg(p1, i);
            add_msg(p2, pM);    /* send to p2 */
        }
    }
    /* send msg from p2, to p1 */
    for (i = 0; i < p2 -> index; i++) {
        if ( p2 -> table[i]->dest == p1 -> number) {
            pM = get_msg(p2, i);
            add_msg(p1, pM);    /* send to p1 */
        }
    }
}

static int cmp_msg(const void *p1, const void *p2)
{
    msg **m1 = (msg **)p1;
    msg **m2 = (msg **)p2;
    return (*m1) -> dest - (*m2) -> dest;
}
static void
sort_node(node *p)
{
    qsort(p -> table, p -> index, sizeof(msg *), cmp_msg);
}

static void
sort_all_node(void)
{
    int i;
    for (i =0 ; i < num_of_nodes ; i++)
        sort_node(current_state+i);
}

static void
copy_all_node(node *p1, node *p2)
{
    int i;
    int j;
    for (i =0 ; i < num_of_nodes ; i++) {
        p2[i].number = p1[i].number;
        p2[i].tbl_size = p1[i].tbl_size;
        p2[i].index = p1[i].index;
        for (j = 0 ; j < p1[i].index; j++)
            p2[i].table[j] = p1[i].table[j];
    }
}

static int
get_rightmost_msg(node *pNode, msg *msg_array[window_size])
{
    int my_num = pNode -> number;
    int ret;
    int i;

    int j = 0;

    /* note! the messages in node->table are sorted */
    for (i =  pNode -> index - 1; i >= 0; i--) {
        msg *pMsg = pNode->table[i];
        int distance = pMsg->dest - my_num;
        if (distance > 0) {/*this msg should send to righ*/
            msg_array[j++] = pMsg;
            if (j >= window_size)
                break;
        }
    }

    ret = j;
    while (j < window_size)
        msg_array[j++] = NULL;

    /* remove msg from node */
    for (i = 0; i < window_size; i++) {
        if (msg_array[i])
            del_msg(pNode, msg_array[i]);
    }

    return ret;
}

static int
get_leftmost_msg(node *pNode, msg *msg_array[window_size])
{
    int my_num = pNode -> number;
    int ret;
    int i;

    int j = 0;

    /* note! the messages in node->table are sorted */
    for (i = 0; i <  pNode -> index - 1; i++) {
        msg *pMsg = pNode->table[i];
        int distance = my_num - pMsg->dest;
        if (distance > 0) {/*this msg should send to left*/
```

```
            msg_array[j++] = pMsg;
            if (j >= window_size)
                break;
        }
    }

    ret = j;
    while (j < window_size)
        msg_array[j++] = NULL;

    /* remove msg from node */
    for (i = 0; i < window_size; i++) {
        if (msg_array[i])
            del_msg(pNode, msg_array[i]);
        else
            msg_array[i] = 0;
    }
    return ret;
}

static msg *
new_msg(int src, int dest)
{
    msg *ret = malloc(sizeof(msg));
    ret -> src = src;
    ret -> dest = dest;
    return ret;
}

static void
add_msg(node *pNode, msg *pMsg)
{
    pNode -> table[pNode->index] = pMsg;
    pNode -> index++;
}

static msg *
get_msg(node *pNode, int i)
{
    msg *ret;
    if (i >= pNode -> index)
        return 0;

    ret = pNode -> table[i];
    pNode -> table[i] = pNode -> table[pNode->index - 1];
    pNode -> index--;
    return ret;
}

static void
del_msg(node *pNode, msg *pMsg)
{
    int i;
    for (i =0 ; i < pNode -> index; i++) {
        if (pMsg == pNode -> table[i]) {
            pNode -> table[i] =pNode->table[pNode->index-1];
            pNode -> index--;
            return;
        }
    }
}

static void
init_node(node *pNode, int num)
{
    msg *m;
    int i;

    pNode -> number = num;
    pNode -> tbl_size = num_of_nodes*num_of_nodes;
    pNode -> table = (msg **)malloc(
                             sizeof(msg*)*pNode->tbl_size);
    pNode -> index = 0;

    for (i =0 ; i< num_of_nodes; i++) {
        m = new_msg(num, i);
        add_msg(pNode, m);
    }

}

static void
init_all(void)
{
    int i;
    all_nodes_1 = (node *)malloc(sizeof(node)*num_of_nodes);
    all_nodes_2 = (node *)malloc(sizeof(node)*num_of_nodes);
    current_state = all_nodes_1;
    next_state = all_nodes_2;

    for (i = 0; i < num_of_nodes; i++) {
        init_node(current_state + i, i);
        init_node(next_state + i, i);
    }
}

static void
print_all(void)
{
    int i;
    int j;
    int printed;

    j = 0;
    printf("Step %d\n", step);
    while (1) {
        msg *pM;
        printed = 0;
        for (i = 0; i < num_of_nodes ; i++) {
            if ( j < current_state[i].index ) {
                pM =  current_state[i].table[j];
                printf("%2d,%-2d ", pM->src,pM->dest);
                printed = 1;
            } else
                printf("      ");
        }
        j++;
        printf("\n");

        if (!printed)
            return;
    }

}
```

The running results for $HOW(10,3,1)$ and $HOW(11,4,1)$ are:

```
For HOW(10,3,1):
Step 0
 0,0   1,0   2,0   3,0   4,0   5,0   6,0   7,0   8,0   9,0
 0,1    1,1    2,1    3,1    4,1    5,1    6,1    7,1    8,1    9,1
 0,2    1,2    2,2    3,2    4,2    5,2    6,2    7,2    8,2    9,2
 0,3    1,3    2,3    3,3    4,3    5,3    6,3    7,3    8,3    9,3
 0,4    1,4    2,4    3,4    4,4    5,4    6,4    7,4    8,4    9,4
 0,5    1,5    2,5    3,5    4,5    5,5    6,5    7,5    8,5    9,5
 0,6    1,6    2,6    3,6    4,6    5,6    6,6    7,6    8,6    9,6
 0,7    1,7    2,7    3,7    4,7    5,7    6,7    7,7    8,7    9,7
 0,8    1,8    2,8    3,8    4,8    5,8    6,8    7,8    8,8    9,8
 0,9    1,9    2,9    3,9    4,9    5,9    6,9    7,9    8,9    9,9

Step 1
 0,0    0,1    0,2    0,3    4,0    5,0    6,0    7,0    8,0    9,0
 3,0    1,1    1,2    1,3    1,4    5,1    6,1    7,1    8,1    9,1
 1,0    4,1    2,2    2,3    2,4    2,5    6,2    7,2    8,2    9,2
 2,0    2,1    4,2    3,3    3,4    3,5    3,6    7,3    8,3    9,3
 0,4    3,1    3,2    4,3    4,4    4,5    4,6    4,7    8,4    9,4
 0,5    1,5    5,2    6,3    7,4    5,5    5,6    5,7    5,8    9,5
 0,6    1,6    2,6    5,3    5,4    7,5    6,6    6,7    6,8    6,9
 0,7    1,7    2,7    3,7    6,4    6,5    8,6    7,7    7,8    7,9
 0,8    1,8    2,8    3,8    4,8    8,5    7,6    8,7    8,8    8,9
 0,9    1,9    2,9    3,9    4,9    5,9    9,6    9,7    9,8    9,9

Step 2
 0,0    4,0    5,0    6,0    7,0    8,0    9,0    9,1    9,2    9,3
 3,0    1,1    1,2    5,1    6,1    7,1    8,1    8,2    8,3    9,4
 1,0    3,1    4,2    1,3    2,4    6,2    7,2    7,3    8,4    9,5
 2,0    2,1    2,2    2,3    3,4    3,5    3,6    5,7    9,8    8,9
 0,4    4,1    3,2    3,3    4,4    4,5    4,6    4,7    6,8    7,9
 0,5    0,1    5,2    6,3    7,4    5,5    5,6    7,7    5,8    9,9
 0,6    1,5    0,2    4,3    1,4    2,5    6,6    6,7    7,8    6,9
        1,6    2,6    5,3    6,4    6,5    8,6    8,7    8,8
        0,7    1,7    0,3    5,4    8,5    7,6    9,7    5,9
               0,8    2,7    3,7    7,5    9,6    4,9
                      1,8    2,8    3,8    4,8
                      0,9    1,9    2,9    3,9

Step 3
 0,0    5,1    7,0    8,0    8,1    8,2    7,3    8,4    9,5    5,9
 3,0    1,1    1,2    9,0    7,1    9,2    8,3    9,4    3,8    4,9
 1,0    3,1    4,2    1,3    9,1    7,2    9,3    3,7    4,8    3,9
 2,0    2,1    2,2    2,3    6,4    3,5    3,6    5,7    9,8    8,9
 6,0    4,1    3,2    3,3    3,4    4,5    4,6    4,7    6,8    7,9
 5,0    0,1    5,2    6,3    4,4    5,5    5,6    7,7    5,8    9,9
 4,0    6,1    0,2    4,3    1,4    2,5    6,6    6,7    7,8    6,9
```

```
        0,4    6,2    5,3    7,4    6,5    8,6    8,7    8,8
               1,5    0,3    5,4    8,5    7,6    9,7
               0,5    0,6    2,4    7,5    9,6    2,9
                      1,6    0,7    2,8    0,9
                      2,6    1,7    1,8    1,9
                             2,7    0,8

Step 4
 0,0    9,0    7,1    9,1    7,2    9,3    2,6    2,7    6,8    5,9
 3,0    1,1    1,2    9,2    8,3    9,4    5,6    4,7    3,8    4,9
 1,0    3,1    4,2    1,3    8,4    9,5    6,6    3,7    4,8    3,9
 2,0    2,1    2,2    2,3    6,4    3,5    3,6    5,7    9,8    8,9
 6,0    4,1    3,2    3,3    3,4    4,5    4,6    7,7    5,8    7,9
 5,0    0,1    5,2    6,3    4,4    5,5    8,6    8,7    8,8    9,9
 4,0    6,1    0,2    4,3    1,4    2,5    7,6    6,7    7,8    6,9
 8,0    8,1    6,2    5,3    7,4    6,5    9,6    9,7    0,8    1,9
 7,0    5,1    8,2    0,3    5,4    8,5    1,7    1,8    0,9    2,9
                      7,3    2,4    7,5    2,8
                             0,4    0,5
                             1,5    1,6
                             0,6    0,7

Step 5
 0,0    7,1    9,2    8,3    9,4    1,5    2,6    2,7    6,8    5,9
 3,0    1,1    1,2    9,3    0,4    0,5    5,6    4,7    3,8    4,9
 1,0    3,1    4,2    1,3    8,4    9,5    6,6    3,7    4,8    3,9
 2,0    2,1    2,2    2,3    6,4    3,5    3,6    5,7    9,8    8,9
 6,0    4,1    3,2    3,3    3,4    4,5    4,6    7,7    5,8    7,9
 5,0    0,1    5,2    6,3    4,4    5,5    8,6    8,7    8,8    9,9
 4,0    6,1    0,2    4,3    1,4    2,5    7,6    6,7    7,8    6,9
 8,0    8,1    6,2    5,3    7,4    6,5    9,6    9,7    0,8    1,9
 7,0    5,1    8,2    0,3    5,4    8,5    0,6    1,7    1,8    2,9
 9,0    9,1    7,2    7,3    2,4    7,5    1,6    0,7    2,8    0,9


For HOW(11,4,1)
Step 0
 0,0    1,0    2,0    3,0    4,0    5,0    6,0    7,0    8,0    9,0   10,0
 0,1    1,1    2,1    3,1    4,1    5,1    6,1    7,1    8,1    9,1   10,1
 0,2    1,2    2,2    3,2    4,2    5,2    6,2    7,2    8,2    9,2   10,2
 0,3    1,3    2,3    3,3    4,3    5,3    6,3    7,3    8,3    9,3   10,3
 0,4    1,4    2,4    3,4    4,4    5,4    6,4    7,4    8,4    9,4   10,4
 0,5    1,5    2,5    3,5    4,5    5,5    6,5    7,5    8,5    9,5   10,5
 0,6    1,6    2,6    3,6    4,6    5,6    6,6    7,6    8,6    9,6   10,6
 0,7    1,7    2,7    3,7    4,7    5,7    6,7    7,7    8,7    9,7   10,7
 0,8    1,8    2,8    3,8    4,8    5,8    6,8    7,8    8,8    9,8   10,8
 0,9    1,9    2,9    3,9    4,9    5,9    6,9    7,9    8,9    9,9   10,9
 0,10   1,10   2,10   3,10   4,10   5,10   6,10   7,10   8,10   9,10  10,10

Step 1
 0,0    0,1    0,2    0,3    0,4    5,0    6,0    7,0    8,0    9,0   10,0
 1,0    1,1    1,2    1,3    1,4    1,5    6,1    7,1    8,1    9,1   10,1
 2,0    2,1    2,2    2,3    2,4    2,5    2,6    7,2    8,2    9,2   10,2
 3,0    3,1    3,2    3,3    3,4    3,5    3,6    3,7    8,3    9,3   10,3
 4,0    4,1    4,2    4,3    4,4    4,5    4,6    4,7    4,8    9,4   10,4
 0,5    5,1    5,2    5,3    5,4    5,5    5,6    5,7    5,8    5,9   10,5
 0,6    1,6    6,2    6,3    6,4    6,5    6,6    6,7    6,8    6,9    6,10
 0,7    1,7    2,7    7,3    7,4    7,5    9,6    7,7    7,8    7,9    7,10
 0,8    1,8    2,8    3,8    8,4    8,5    7,6    9,7    8,8    8,9    8,10
 0,9    1,9    2,9    3,9    4,9    9,5    8,6    8,7    9,8    9,9    9,10
 0,10   1,10   2,10   3,10   4,10   5,10  10,6   10,7   10,8   10,9   10,10

Step 2
 0,0    5,0    6,0    7,0    8,0    9,0   10,0   10,1   10,2   10,3   10,4
 1,0    0,1    0,2    6,1    7,1    8,1    9,1    9,2    9,3    9,4   10,5
 2,0    1,1    1,2    0,3    0,4    7,2    8,2    8,3   10,8   10,9   10,10
 3,0    2,1    2,2    1,3    1,4    1,5    2,6    3,7    4,8    9,9    9,10
 4,0    3,1    3,2    2,3    2,4    2,5    3,6    4,7    5,8    5,9    8,10
 0,5    4,1    4,2    3,3    3,4    3,5    4,6    5,7    6,8    6,9    7,10
 0,6    5,1    5,2    4,3    4,4    4,5    5,6    6,7    7,8    7,9    6,10
        1,6    6,2    5,3    5,4    5,5    6,6    7,7    8,8    8,9
        0,7    1,7    6,3    6,4    6,5    9,6    9,7    9,8    5,10
               0,8    7,3    7,4    7,5    7,6    8,7    4,10
                      2,7    8,4    8,5    8,6   10,7
                      1,8    2,8    9,5   10,6    4,9
                      0,9    1,9    3,8    3,9    3,10
                             0,10   2,9    2,10
                                    1,10

Step 3
 0,0    6,1    7,2    9,0    9,1    9,2   10,4   10,5    2,8    3,9    5,10
 1,0    0,1    0,2   10,0   10,1   10,3    9,4    2,7    3,8    2,9    4,10
 2,0    1,1    1,2    0,3   10,2    9,3    2,6    3,7   10,8   10,9   10,10
 3,0    2,1    2,2    1,3    0,4    1,5    3,6    4,7    4,8    9,9    9,10
 4,0    3,1    3,2    2,3    1,4    2,5    4,6    5,7    5,8    5,9    8,10
 8,0    4,1    4,2    3,3    2,4    3,5    5,6    6,7    6,8    6,9    7,10
 7,0    5,1    5,2    4,3    3,4    4,5    6,6    7,7    7,8    7,9    6,10
```

```
   6,0   8,1   6,2   5,3   4,4   5,5   9,6   9,7   8,8   8,9  2,10
   5,0   7,1   8,2   6,3   5,4   6,5   7,6   8,7   9,8   4,9  3,10
                     7,3   6,4   7,5   8,6  10,7
                     8,3   7,4   8,5  10,6  1,10
                     0,5   8,4   9,5   0,8  0,10
                           1,6   1,7   0,9
                           0,6   0,7   1,9
                                 1,8

Step 4
   0,0  10,0  10,1  10,2   9,3   9,4   0,6   2,7   2,8   3,9  5,10
   1,0   6,1   7,2  10,3  10,4  10,5   2,6   3,7   3,8   2,9  4,10
   2,0   0,1   0,2   0,3   8,4   0,5   3,6   4,7  10,8  10,9 10,10
   3,0   1,1   1,2   1,3   0,4   1,5   4,6   5,7   4,8   9,9  9,10
   4,0   2,1   2,2   2,3   1,4   2,5   5,6   6,7   5,8   5,9  8,10
   8,0   3,1   3,2   3,3   2,4   3,5   6,6   7,7   6,8   6,9  7,10
   7,0   4,1   4,2   4,3   3,4   4,5   9,6   9,7   7,8   7,9  6,10
   6,0   5,1   5,2   5,3   4,4   5,5   7,6   8,7   8,8   8,9  2,10
   5,0   8,1   6,2   6,3   5,4   6,5   8,6  10,7   9,8   4,9  3,10
   9,0   7,1   8,2   7,3   6,4   7,5  10,6   0,7   1,8   1,9  0,10
         9,1   9,2   8,3   7,4   8,5   1,7   0,8   0,9  1,10
                                 9,5
                                 1,6

Step 5
   0,0  10,1  10,2   9,3   9,4   9,5   0,6   2,7   2,8   3,9  5,10
   1,0   6,1   7,2  10,3  10,4  10,5   2,6   3,7   3,8   2,9  4,10
   2,0   0,1   0,2   0,3   8,4   0,5   3,6   4,7  10,8  10,9 10,10
   3,0   1,1   1,2   1,3   0,4   1,5   4,6   5,7   4,8   9,9  9,10
   4,0   2,1   2,2   2,3   1,4   2,5   5,6   6,7   5,8   5,9  8,10
   8,0   3,1   3,2   3,3   2,4   3,5   6,6   7,7   6,8   6,9  7,10
   7,0   4,1   4,2   4,3   3,4   4,5   9,6   9,7   7,8   7,9  6,10
   6,0   5,1   5,2   5,3   4,4   5,5   7,6   8,7   8,8   8,9  2,10
   5,0   8,1   6,2   6,3   5,4   6,5   8,6  10,7   9,8   4,9  3,10
   9,0   7,1   8,2   7,3   6,4   7,5  10,6   0,7   1,8   1,9  0,10
  10,0   9,1   9,2   8,3   7,4   8,5   1,6   1,7   0,8   0,9  1,10
```

REFERENCES

1. S. G. Ziavras, "RH: A Versatile Family of Reduced Hypercube Interconnection Networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 11, Nov. 1994, pp. 1210-1220.

2. C. Qiao and R. Melhem, "Reducing Communication Latency with Path Multiplexing in Optically Interconnected Multiprocessor Systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 2, Feb. 1997, pp. 97-108.

3. J. K. Antonio, L. Lin, and R. C. Metzger, "Complexity of Intensive Communications on Balanced Generalized Hypercubes," *International Parallel Processing Symposium*, 1993, pp. 387-394.

4. P. Fragopoulou, S. G. Akl, and H. Meijer, "Optimal Communication Primitives on the Generalized Hypercube Network," *Journal of Parallel and Distributed Computing*, Vol. 32, 1996, pp. 173-187.

5. V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms,* Benjamin/Cummings, California, 1994.

6. W. Dally, "Network and Processor Architecture for Message-Driven Computers," in: *VLSI and Parallel Computation*, R. Suaya and G. Birtwistle (Eds.), Morgan Kaufmann, California, 1990, pp. 140-222.

7. W.J. Dally and C.L. Seitz, "The Torus Routing Chip," *Journal of Distributed Computing*, Vol. 1, No. 3, 1986, pp. 187-196.

8. M.C. Pease, III, "The Indirect Binary n-Cube Microprocessor Array," *IEEE Transactions on.Computers*, C-26(5), 1977, pp. 458-473.

9. C.L. Seitz, "Concurrent VLSI Architectures," *IEEE Transactions on Computers*, C-33(12), 1984, pp. 1247-1265.

10. T. Szymanski, " "Hypermeshes": Optical Interconnection Networks for Parallel Computing, " *Journal of Parallel and Distributed Computing*, Vol. 26, 1995, pp. 1-23.

11. L.D. Wittie, "Communication Structures for Large Networks of Multicomputers," *IEEE Transactions on Computers*, C-30(4), 1981.

12. S.G. Ziavras, "Generalized Reduced Hypercube Interconnection Networks for Massively Parallel Computers," in: *Networks for Parallel Computations*, D.F. Hsu, A. Rosenberg, and D. Sotteau (Eds.), American Mathematical Society, Rhode Island, 1995, pp. 307-325.

13. S.G. Ziavras and A. Mukherjee, "Data Broadcasting and Reduction, Prefix Computation, and Sorting on Reduced Hypercube Parallel Computers," *Parallel Computing* 22, 1996, pp. 595-606.

14. S.G. Ziavras, "On the Problem of Expanding Hypercube-Based Systems," *Journal of Parallel and Distributed Computing*, 16(1), 1992, pp. 41-53.

15. S.G. Ziavras, "Scalable Multifolded Hypercubes for Versatile Parallel Computers," *Parallel Processing Letters*, 5(2), 1995, pp. 241-250.

16. L.N. Bhuyan and D.P. Agrawal, "Generalized Hypercube and Hyperbus Structures for a Computer Network," *IEEE Transactions on Computers* 33 (4), 1984, pp. 323-333.

17. S.G. Ziavras, "Investigation of Various Mesh Architectures with Broadcast Buses for High-Performance Computing," *VLSI Design*; Special Issue High Performance Bus-Based Architectures, pp. 29-53, 1999.

18. S.G. Ziavras, H. Grebel, and A.T. Chronopoulos, "A Low-Complexity Parallel System for Gracious, Scalable Performance. Case Study for Near PetaFLOPS Computing," *6th Symposium on Frontiers Massively Parallel Computing*, Special Session New Millennium Computing Point Designs, 1996, pp. 363-370.

19. S.G. Ziavras, H. Grebel, and A.T. Chronopoulos, "A Scalable/Feasible Parallel Computer Implementing Electronic and Optical Interconnections for 156 TeraOPS Minimum Performance," *PetaFLOPS Architecture Workshop*, 1996, pp. 179-209.

20. P.T. Gaughan and S. Yalamanchili, "Adaptive Routing Protocols for Hypercube Interconnection Networks," *IEEE Computer*, May 1993, pp. 12-23.

21. P.W. Dowd, "High Performance Interprocessor Communication Through Optical Wavelength Division Multiple Access Channels," *Proceedings of International Symposium on Computer Architecture*, 1991, pp. 96-105.

22. A. Abraham, K. Padmanabhan, "Performance of Multicomputer Networks under Pin-out Constraints," *Journal of Parallel and Distributed Computing*, Vol. 12, 1991, pp. 237-248.

23. A. Agarwal, "Limits on Interconnection Network Performance," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, 1991, pp. 398-412.

24. C.D. Thompson, "Area-Time Complexity for VLSI," *Proceedings of 11th Annual ACM Symposium on Theory of Computing*, May 1979, pp. 81-88.

25. W.J. Dally, "Wire-Efficient VLSI Multiprocessor Communication Networks," *Proceedings of 1987 Stanford Conference on Advanced Research in VLSI*, MIT Press, Cambridge, MA, 1987, pp. 391-415.

26. S.G. Ziavras and S. Krishnamurthy, "Evaluating the Communications Capabilities of the Generalized Hypercube Interconnection Network," *Concurrency: Practice and Experience*, (accepted for publication).

27. J.D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Maryland, 1984.

28. P. Banerjee, *Parallel Algorithms for VLSI Computer-Aided Design*, Prentice-Hall, New Jersey, 1994.

29. Q. Wang and S.G. Ziavras "Powerful and Feasible Processor Interconnections with an Evaluation of Their Communications Capabilities," *International Symposium on Parallel Architectures, Algorithms, and Networks*, Freemantle, Australia, June 23-25 1999.

30. Q. Wang and S.G. Ziavras "Network Embedding Techniques for a New Class of Feasible Parallel Architectures Capable of Very High Performance," *International Conference on Applied Informatics*, Innsbruck, Austria, Feb. 23-25 1999.

31. S.G. Ziavras and Q. Wang, "Robust Interprocessor Connections for Very-High Performance," in: *Robust Communication Networks: Interconnection and Survivability*, N. Dean, F. Hsu and R. Ravi (Eds.), American Mathematical Society, Rhode Island, 1999.