Viable Architectures for High-Performance Computing
*

Sotirios G. Ziavras **, Qian Wang **, Paraskevi Papathanasiou ***
** Departments of Electrical and Computer Engineering, and Computer Science
New Jersey Institute of Technology
Newark, NJ 07102, USA

*** DataLine Computer Institute
Piraeus, 18900 Greece

Address for Correspondence
Professor Sotirios G. Ziavras
Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, New Jersey 07102
USA

Phone: (973) 596-5651
Fax: (973) 596-5680
Email: ziavras@njit.edu

*The work presented in this research was supported in part jointly by NSF and DARPA under the New Millennium Computing Point Design Grant ASC-9634775.

Abstract

Existing interprocessor connection networks are often plagued by poor topological properties that result in large memory latencies for DSM (Distributed Shared-Memory) computers or multicomputers. 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 all its nodes in each dimension and is characterized by outstanding topological properties. Also, low-dimensional GHs have very large bisection widths. We present here the class of HOW (Highly-Overlapping Windows) networks, which are capable of lower complexity than GHs, comparable performance, and better scalability. HOWs are obtained from GHs by uniformly removing edges to produce feasible systems of lower wiring complexity. Resulting systems contain numerous highly-overlapping GHs of smaller size. The GH, the binary hypercube, and the mesh all belong to this new class of interconnections. In practical cases, HOWs have higher bisection width than tori with similar node and channel costs. Also, HOWs have a very large degree of fault tolerance. This paper focuses on 2-D HOW systems. We analyze the hardware cost of HOWs, present graph embeddings and communications algorithms for HOWs, carry out performance comparisons with binary hypercubes and GHs, and simulate HOWs under heavy communication loads. Our results show the suitability of HOWs for very high-performance computing.

Keywords: parallel architecture, interconnection network, graph embedding, performance comparison, generalized hypercube, fault tolerant routing.

1 INTRODUCTION

The demand for ever greater performance by many computation problems has been the driving force for the development of computers with hundreds or thousands of processors. Near PetaFlops (i.e., 10^{15} floating-point operations per second) and more performance is required by many applications. The high-performance computing field generally targets applications that require performance at the many Giga-Flops or PetaFlops range. The *High-End Computing Panel of the President's Information Technology Advisory Committee (PITAC)* in its 1999 report emphasizes that innovations are required in high-end systems to deliver PetaFlops performance [14]. Several high-performance computers have been developed in recent years, such as the SGI Mountain Blue capable of 3 TeraFLOPS, and the IBM ASCI Blue [22] and ASCI White; the latter machine is capable of about 12 trillion calculations per second. IBM is also expected to deliver in 2005 the IBM Blue Gene/L for simulating phenomena such as aging of materials, fires, and explosions, at the 200 TeraFlops rate. The ultimate IBM objective in this field is to build a one PetaFlops machine for the life sciences by the end of this decade [3]. The common objective of the HOW networks and the latter efforts is to support the implementation of computers with thousands of processors. Our work differs from these other efforts because we give primarily emphasis to the implementation, with current or future technologies (e.g., optics), of scalable interconnection networks. Our systems are scalable because they can easily take advantage of not only ever increasing

wiring and/or channel densities (i.e., the number of channels per mm^2) but also of more powerful processing nodes that reduce the volume occupied by machines. The saved volume can then be used to increase each processors degree of connectivity.

Very high performance is very difficult to be achieved or sustained 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. To quote Dally, "wires are a limiting factor because of power and delay as well as density" [10]. Several interconnection networks have been proposed including, among others, regular meshes and tori [11, 9], enhanced meshes [37], (direct binary) hypercubes [26], hypernets [16], fat trees [19], and hypercube variations [36, 31, 32]. The hypercube dominated the high-performance computing field in the 1980's because of good topological properties and rich interconnectivity that permits efficient emulation of many other topologies [26, 34]. 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 [34] and does not permit the construction of powerful systems. 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 (i.e., better scalability) that preserve to a large extent the former's topological properties [36, 32].

The current trend in the high-performance computing field is to build distributed shared-memory (DSM) computers where the main interconnection network connects clusters of SMP (Shared-Memory Processor) subsystems [8, 21, 24]. The underlying message-passing (i.e., distributed) interconnection network in DSM machines also serves as the vehicle for remote memory accesses. The proposed HOWs are also applicable to this case. Therefore, the terms processor and node (very often implying an SMP cluster) will be used interchangeably from now on in this paper. Moore's law predicts the doubling of transistor density for chips about every 18 months. As a consequence, shared-memory multiprocessor chips are slowly appearing in the market; they may eventually serve as the nodes of high-performance computing machines.

To support scalability, several approaches to massively-parallel processing use bounded-degree networks, such as meshes or p-ary n-cubes (i.e., tori), with low node degree (e.g., the FLASH [20], Cray T3E [25], and Tera computers). For example, hypernets are constructed hierarchically with identical cubes, trees, or buses as the basic bulding blocks [16]. Their focus is on maintaining a constant node degree with increases in machine size. Nevertheless, 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 [37].

The high VLSI (wiring) complexity problem is unbearable for generalized hypercubes (GHs). Contrary to nearest-neighbor p-ary n-cubes that form rings with p nodes in each dimension, GHs implement fully-connected systems with p nodes in each dimension [7]. The n-D (symmetric) generalized hypercube $GH(p,n)$ contains p^n nodes. The address of a node is $x_{n-1}x_{n-2}...x_1x_0$, where each x_i is a radix-p digit with $0 \leq x_i \leq p-1$. 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 $p-1$ neighbors in each dimension, for a total of $n(p-1)$ neighbors per node. The n-D $GH(p,n)$ has diameter equal to only n. Low-dimensional generalized hypercubes have very impressive bisection widths. When a network is cut into two equal halves (with the same number of nodes), its bisection width is the number of edges that run between these two halves [2]; dense/heavy communications operations can benefit from a large bisection width. For $n=2$ and p an even number, the bisection width of the $GH(p,2)$ is the immense $p^3/4$. The outstanding topological properties of GHs 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. The increased VLSI/wiring cost of GHs results in outstanding performance that permits optimal emulation of hypercubes and p-ary n-cubes, and efficient implementation of complex communication patterns [4, 13, 40].

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 [12]. Reconfigurable generalized hypercubes interconnect all nodes in each dimension dynamically via a scalable mesh of very simple, low-cost programmable switches [35]. However, all these proposed reductions in hardware complexity may not be sufficient for sustained high performance computing.

Fat trees have received major attention because it has been proved that, for a given amount of communications hardware a fat tree is nearly the best routing network (universality theorem) [19]; that is, the fat tree can simulate any other network containing the same amount of communications hardware in time that may be greater by a polylogarithmic factor. The fat tree structure is based on the mesh-of-trees network [18]. The processors of a fat tree are represented by the leaves of a complete binary tree; all internal nodes in the binary tree are switches. The number of wires (and bandwidth) connecting neighboring nodes in the fat tree increases exponentially with each new level, as the root is approached. It is worth noting that the butterly network is so versatile that it can be redrawn to look like a fat tree.

TRIPS (The Tera-op Reliable Intelligently adaptive Processing System) is a hierarchical

system composed of multiple chips; each chip is composed of eight processors and several memory elements [23]. Each processor has a Grid Processor Architecture (GPA) consisting of an 8x8 array of ALUs, a local register file, local instruction and data caches, and control circuits. The grid processor and the memory arrays are configurable. The objective of TRIPS is to scale well with technology, similarly to our objective for HOWs. Similarly, the Raw architecture emphasizes parallelism within a processor chip that contains simple configurable tiles interconnected in a large structure visible directly by the compiler [29]. Therefore, the Raw hardware can be customized by the compiler to the specific application. Only short wires are used in Raw, contrary to HOWs that employ short intra- and longer inter-chip connections for substantial scalability. Each simple tile in Raw contains instruction and data memories, an ALU, and a register file. Contrary to HOWs, however, where the objective is to scale well within and between processor chips in order to support very high-performance computing, the TRIPS and Raw architectures emphasize scalability within processor chips. Each node in a HOW system could actually be a scalable TRIPS or Raw processor. Therefore, HOWs and TRIPS/Raw complement each other in the areas of technology and architecture scalability.

To summarize, low-dimensional massively-parallel architectures with full connectivity for their nodes in each dimension, such as generalized hypercubes, are very desirable because of their outstanding topological properties, but their electronic implementation is a Herculean task because of packaging (and primarily wiring) constraints. We propose in this paper a new class of interprocessor connection architectures, namely HOWs (standing for architectures with Highly-Overlapping Windows), which employ the generalized hypercube [4, 7, 39, 40] with outstanding topological properties (e.g., extremely small diameter and average internode distance, and immense bisection width) as their basic building block. HOWs are also obtained from generalized hypercubes by removing some of their processor interconnections in order to reduce their 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. In contrast, HOWs can be viable while having simultaneously topological properties comparable to those of generalized hypercubes.

This paper focuses on 2-D HOWs because of their potential for ease of implementation and their large bisection width. It is organized as follows. Section 2 introduces the architecture of HOWs and presents a simple, fault-tolerant routing algorithm to find shortest paths. Section 3 discusses cost analysis for HOWs. Section 4 presents the embedding of various interconnection networks into 2-D HOW systems. Section 5 presents and analyzes communication operations for 2-D HOW systems. Finally, Section 6 presents performance comparisons involving hypercubes (binary and generalized) and 2-D HOW systems.

2 THE CLASS OF HOW ARCHITECTURES

HOWs are designed recursively. We first introduce the class of 1-D HOW node interconnections. $HOW(p, w, 1)$ denotes the 1-D HOW system with p sequentially-numbered 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 are all the nodes with addresses $0 \leq k \pm i \leq p-1$, for $i = 1, 2, 3, \cdots, w$. 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.

Each node 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.

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 $x_{n-1}x_{n-2}\cdots x_i \cdots x_1x_0$, where x_i is a radix-p digit with $0 \leq x_i \leq p-1$, for all $i = 0, 1, \cdots, n-1$. This node has neighbors with addresses that differ from its own address only in a single radix-p digit, that is they have addresses $x_{n-1}x_{n-2}\cdots x_i^{'} \cdots x_1x_0$, so that $1 \leq |x_i - x_i^{'}| \leq w$ for all $i = 0, 1, \cdots, n-1$. This HOW system contains p^n nodes. It 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 shows the $HOW(7, 3, 2)$. *The $HOW(p, p-1, n)$ is identical to the $GH(p, n)$, the $HOW(p, 1, n)$ is identical to the n-D mesh, and the $HOW(2, 1, n)$ is identical to the n-D binary hypercube.* This paper focuses on 2-D HOWs because of their simplicity, high bisection width, and ease of implementation.

Not only do HOWs have reduced wiring complexity than GHs of similar size, but also the locality of node interconnections in HOWs can prove a viable solution for very high-performance computing:

- 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

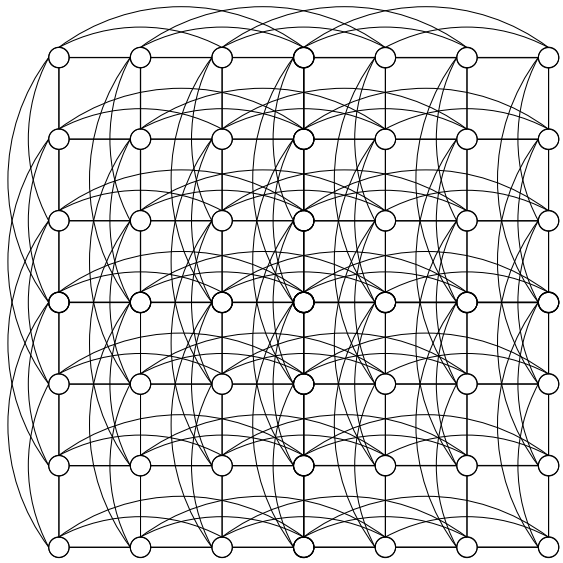


Figure 1: The 2-D $HOW(7, 3, 2)$ with $w = 3$.

more difficult to realize. Improvements in intrachip and/or interchip interconnection technologies can increase the value of w, so HOWs are technology scalable.

- Free-space optical interconnects are expected to become viable in the near future for the local interconnection of chips [39]. 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 [39]. 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 also demonstrate that HOWs are more prone than GHs to scalability related to technological advancements. Current COTS chips have about 1000 pins. Assuming 64-bit unidirectional channels and about 800 of these pins available exclusively for data transfers, we can have $w = 6$. Around the year 2010, COTS chips will have about 3000 pins (Semiconductor Industry Association-SIA prediction). Assuming about 2600 of these pins exclusively for data transfers, we will have $w = 20$. ASIC chips can generally have up to double the number of pins on COTS chips, and therefore for ASIC chips we can currently have $w = 12$.

The mesh, binary hypercube, and generalized hypercube are extremes in the spectrum of networks that HOWs are proposed for. They correspond to HOWs with $w = 1$, $p = 2$ (of course, we can only have $w = 1$ in this case), and $w = p - 1$, respectively. Therefore, the proposed HOWs are compared primarily against these extreme cases, that correspond to widely used networks in theory and practice. Some comparisons with the fat tree network

are included as well. For a comparative evaluation of these networks, a trade-off analysis must be carried out between performance and cost. As pointed out in [8], network topology has been a point of major debate primarily because "different positions make sense under different cost models and the technology keeps changing." Using *HOW* architectures, we can take advantage of wiring and/or optical technologies to ever increase the window size for ever higher performance, while supporting program portability.

*HOW*s can have very good topological properties and reasonable cost. The following proposition and theorem are pertinent.

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

Proof. 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. *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 $np^{n-1}c_1$. □

2.1 Routing in HOWs and Support for Fault Tolerance

A major requirement for any new interconnection network is to support ease of routing. We show in this Subsection that HOWs adhere to this requirement. For the sake of illustration, let us first describe a simple routing algorithm for the 1-D $HOW(p, w, 1)$. This routing algorithm is distributed and chooses a shortest path. Assume that the source and destination addresses are A and B, respectively. Routing is carried out in $\lambda = \lceil\frac{|B-A|}{w}\rceil$ steps. Assume that the required direction of transfer is shown by the value of the parameter $sign(B - A)$; it is $+1$ if $B - A \geq 0$, otherwise it is -1. Also, R is the remainder of the division $|B - A|/w$.

The randomized routing algorithm that results in a shortest path is as follows. The current node A sends the message to any node with address $A + sign(B - A) * \tau$, where $R \leq \tau \leq w$. Therefore, the node chooses any of these $w - R + 1$ neighbors. Because of these potentially multiple choices and the distributed nature of the routing algorithm, multiple paths of minimum length may be available for the transmission of a message.

Proposition 2. For $R > 0$, the number of multiple paths of minimum length for the transmission of a message from the source address A to the destination address B in the

$HOW(p, w, 1)$ is

$$\lambda + \lfloor \frac{w-R}{2} \rfloor \lambda (\lambda - 1) - \alpha$$

where

$$\alpha = \begin{cases} 0 & \text{if } \frac{w-R}{2} \text{ is not an integer} \\ \frac{\lambda(\lambda-1)}{2} & \text{otherwise} \end{cases}$$

There is a single path of minimum length for $R = 0$.

Proof. There exists a minimum length path that first follows $\lfloor \frac{|B-A|}{w} \rfloor$ links of length w and up to one link of length R in the 1-D representation of the HOW system. For $R > 0$, the link of length R can be chosen, however, in any phase during the transfer; that is, it can be selected either at the source or at any other intermediate node in the path, without increasing the total distance travelled; this distance is λ hops. The total number of such paths is equal to λ, thus the first term of the equation. Each remaining path of minimum length contains anywhere two links of length $R+i$ and $w-i$ with all its other links being of length w, for all integers i with $0 < i \leq \lceil \frac{w-R}{2} \rceil$. The total number of permutations of these pairs of links in the path for a given value of i is $\lambda(\lambda - 1)$, i assumes $\lfloor \frac{w-R}{2} \rfloor$ possible values, and pairs of paths are identical for $i = (w-R)/2$ (if it is an integer). Therefore, the second and third terms of the equation represent these additional paths. □

For example, in the $HOW(64, 7, 1)$ there exist 22 paths of minimum length 4 between any pair of nodes whose addresses differ by 24; 55 paths of minimum length 5 between pairs differing by 29; 117 paths of minimum length 9 between pairs differing by 59; and 32 paths of minimum length 4 between pairs differing by 23. For a multidimensional *HOW* system, the total number of shortest paths is given by the product of the numbers of shortest paths in all dimensions. This may be a very large number in practical cases. Thus, *HOW* systems are highly fault tolerant. The routing algorithm is deadlock free if we adhere to dimension-order routing. Since a large number of shortest paths exist for most pairs of nodes, faulty links or nodes can be easily avoided with minimal overhead. For pairs of nodes corresponding to $R = 0$, there exist numerous alternative paths that increase the shortest distance by just one.

3 COST ANALYSIS

Our main objective here is to show that HOWs have reduced VLSI/wire complexity compared to generalized hypercubes, despite the fact that they can deliver comparable performance (as shown in subsequent sections). This reduced complexity of HOWs is investigated with respect to the number of channel-wires and the complexity of the VLSI/wire layout. We also compare the numbers of channel-wires in HOWs with those of other popular interconnection networks, for systems of comparable size. Additionally, we compare the bisection width of HOWs with that of the torus, a popular network. Another major interconnection network nowadays is the fat tree. Therefore, a comparison is also made with the latter network, despite that it is actually an indirect network (since it employs switches) and, therefore, a

comparison of topological properties is not really easy. The following proposition shows the complexity of the fat tree.

Proposition 3. Assume that the channel attached to each processor (leaf node) in the binary fat tree consists of k wires and that the bandwidth of individual channels doubles for each immediately higher level. If the total number of processors is $N = 2^m$ and the switches support all permutations of the attached channels, then the fat tree contains mNk wires and $Nk(5N - 8)$ binary switches. Binary switches can assume only two states, namely "open" and "close."

Proof. The fat tree has $m + 1$ levels and the total number of wires between any pair of consecutive levels is equal to Nk. Therefore, the total number of wires that implement the channels is equal to mNk. The switch in the apex is connected to $2^{m-1}k$ wires to its left and $2^{m-1}k$ wires to its right, for a total of $2^m k$ or Nk wires. To be able to implement all possible permutations of its inputs, it is a crossbar switch with N inputs and N outputs, where the width of each input or output is k. This crossbar switch is implemented with N^2k binary switches. Level i (at distance i from the leaves), where $1 \leq i \leq m - 1$, contains 2^{m-i} crossbar switches; each crossbar switch has 2^{i+1} inputs and the same number of outputs, where each line contains k wires. Thus, level i contains a total of $2^{m-i}2^{2(i+1)}k$ or $4N2^ik$ binary switches. As a result, the total number of binary switches in the fat tree is equal to $N^2k + \sum_{i=1}^{m-1} 4N2^ik$ or $Nk(5N - 8)$. □

Table 1 compares the numbers of channels in the binary hypercube, the p-ary n-cube (i.e., n-D torus), the generalized hypercube $GH(p, n)$, the binary fat tree, the 2-D HOW, and the n-D HOW, all with the same number $N = p^n$ of nodes. This table shows that the number of channels in the $HOW(p, w, n)$ is smaller than that in the $GH(p, n)$, in the $O(.)$ notation, if $O(w) < O(N^{\frac{1}{n}})$ or $O(w) < O(p)$. As an example, assume systems with $N = 16,384$ nodes (i.e., $m = 14$) and 64-bit data channels with two sets of 64 wires for full-duplex bidirectional data transfers. The total numbers of wires for inter-node data transfers are:

- 14,680,064 wires for the 14-cube with diameter 14;
- 4,194,304 wires for the 128-ary 2-cube (i.e., 2-D torus) with diameter 128;
- 266,338,304 wires for the 2-D $GH(128, 2)$ with diameter 2;
- 29,360,128 wires for the 15-level binary fat tree with 1.72×10^{11} binary switches and diameter 28;
- 116,916,224 wires for the $HOW(128, 32, 2)$ with diameter 8;
- 62,652,416 wires for the $HOW(128, 16, 2)$ with diameter 16; and
- 32,374,784 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 current and

Table 1: Comparison of interconnection networks. All networks have $N = p^n$ nodes.

Network	Number of channels	Diameter
$\log_2 N$-cube	$\frac{N}{2} * \log_2 N$	$\log_2 N = n * \log_2 p$
$N^{\frac{1}{n}}$-ary n-cube	$n * N$	$n * \lfloor \frac{N^{\frac{1}{n}}}{2} \rfloor = n * \lfloor \frac{p}{2} \rfloor$
$GH(N^{\frac{1}{n}}, n)$	$(N^{\frac{1}{n}} - 1) * n * \frac{N}{2}$	$\log_p N = n$
$(1 + \log_2 N) - level \;\; fat \;\; tree$	$2 * N * \log_2 N \quad and$ $2 * N * (5N - 8) \;\; channel - wide$ $binary \;\; switches$	$2 * \log_2 N$
$HOW(\sqrt{N}, w, 2)$	$\sqrt{N} * w * (2 * \sqrt{N} - w - 1)$	$2 * \lceil \frac{\sqrt{N}-1}{w} \rceil = 2 * \lceil \frac{p-1}{w} \rceil$
$HOW(N^{\frac{1}{n}}, w, n)$	$\frac{n}{2} * N^{1-\frac{1}{n}} * w * (2 * N^{\frac{1}{n}} - w - 1)$	$n * \lceil \frac{N^{\frac{1}{n}}-1}{w} \rceil = n * \lceil \frac{p-1}{w} \rceil$

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 nodes and, therefore, are difficult to build. Also, large generalized hypercubes are impossible to build because of their very large wiring complexity and their global interconnectivity. Finally, fat trees of large size are very difficult to build because they contain very large numbers of binary switches. When wraparound connections also are included, the diameter of the $HOW(p, w, n)$ is reduced by half to $n\lceil \frac{p-1}{2w} \rceil$. However, we do not study here HOW systems with wraparound connections because we want to facilitate the implementation even of large-dimensional HOW systems (using local interconnects only). In addition, the diameter does not change in the asymptotic $O(.)$ notation because of the wraparound connections.

Let us also compare the bisection widths of the torus, the HOW with wraparound connections, and the generalized hypercube. We have included the torus in this comparison because it is easy to construct, and many real computers contain this network. A very large bisection width may make the network difficult to build but it also reduces the probabilty for communication bottlenecks. The bisection width of the $GH(k, n)$ is $O(k^{n+1})$; it is $O(p^{3/2})$ for the $GH(\sqrt{p}, 2)$. For the 1-D $HOW(\sqrt{p}, w, 1)$ with wraparound connections, the bisection width is $2(1 + 2 + 3 + \cdots + w) = w(w + 1) = O(w^2)$. For the 2-D $HOW(\sqrt{p}, w, 2)$ with wraparound connections, the bisection width is $w(w + 1)\sqrt{p} = O(\sqrt{p}\, w^2)$. Finally, the bisection width of the binary fat tree with the same number of nodes is $p/2$ at the expense of $O(p \log_2 p)$ wires and $O(p^2)$ binary switches (versus $O(pw)$ and zero, respectively, for the 2-D HOW).

Let us now compare in detail the bisection width of 2-D HOWs (the main focus of this paper) with that of tori, for networks comparable in cost. Assume the symmetric 2-D $HOW(\sqrt{p}, w, 2)$ with p nodes (and wraparound connections in each dimension) and the symmetric m-D torus with the same number $p = \beta^m$ of nodes. Thus, $\beta = p^{1/m}$. For the two

systems to contain the same number of wires, we need $4w = 2m$ or $m = 2w$. The bisection width of the torus is β^{m-1}. For the bisection width of the HOW to be larger than or equal to that of the torus, we need to have $w(w+1)\sqrt{p} \geq p^{1-\frac{1}{2w}}$ or $p \leq [w(w+1)]^{\frac{2w}{w-1}}$. Table 2 shows the maximum number, as a function of w, of nodes required for the HOW to have better bisection width than the torus. We can infer from this table that, in practical cases HOWs have better bisection width than tori of the same cost.

Table 2: Maximum number p of nodes required for the HOW with wraparound connections to have higher, or equal to, bisection width than the comparable torus (with the same number of nodes and the same number of connections).

w	Maximum number of nodes (p)	w	Maximum number of nodes (p)
2	10	3	1,728
4	2,947	5	4,929
6	7,866	7	11,997
8	17,592	9	24,948
10	34,389	11	46,266
12	60,953	13	78,853
14	100,392	15	126,023
16	156,223	17	191,495
18	232,366	19	279,389
20	333,142	21	394,228

The study of further issues related to the VLSI/wire cost is now in order. A VLSI cost comparison between 1-D HOWs and generalized hypercubes is presented. Since the focus of our attention is 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 block). The next definition is pertinent.

Definition 1. *The crossing number of a graph is the minimum number of edge crossings needed to draw the graph in the plane* [28].

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 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: (a) We consecutively number the nodes 0, 1, 2, $\cdots$, $p-1$, from left to right. (b) Going from left to right, for even-numbered nodes the wires go to the top half of the printed-circuit board. (c) For odd-numbered nodes, the wires go to the bottom half of the printed-circuit board. These rules of routing minimize the maximum collective

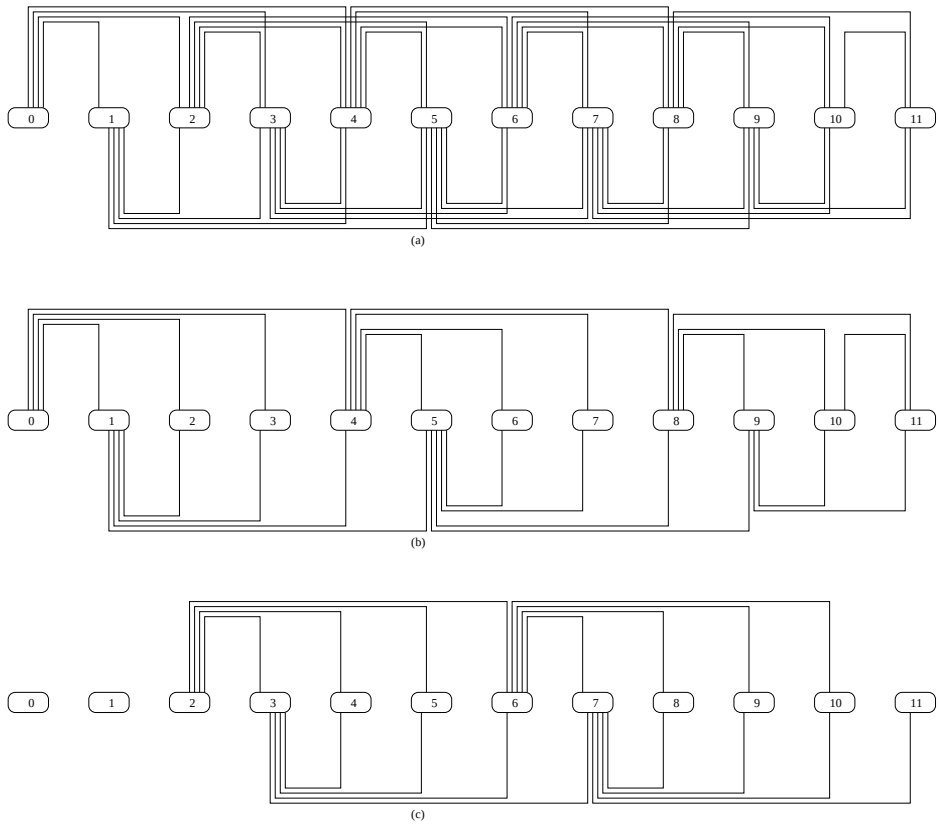


Figure 2: Colinear layout of the 1-D HOW system with 12 PEs and window size of 4, and its brute-force decomposition into printed-circuit layers.

width, MCW (expressed in number of wires), in the y (vertical) dimension. Figure 2 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 w, and thus it increases with increases in the window size. The following theorems are pertinent.

Theorem 2. *The MCW in the colinear layout of the 1-D $HOW(p, w, 1)$ with a single layer is*

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

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. The MCW is 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. 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 $i\leq\lfloor\frac{\alpha}{4}\rfloor$. The value of MCW is then given by $\sum_{i=0}^{\phi}(\alpha-4i)$, where $\phi=\lfloor\frac{\alpha}{4}\rfloor$. This sum is also equal to $(p-3)\phi+p-1-2\phi^2$. □

Therefore, HOWs have much smaller wire width (MCW) than generalized hypercubes in 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 is applied.

Theorem 3. *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 on 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. Let us now deal with another wire routing technique, namely restricted routing [5], 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. Horizontal and vertical wire segments are laid on two different wiring layers. Figure 3 demonstrates this technique for the $HOW(12, 4, 1)$. 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. This number is $O(w^2)$ and $O(p^2)$ for HOWs and GHs, respectively.
- The length of the wires. The maximum length of horizontal wires 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 double 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.

4 EMBEDDINGS INTO 2-D HOWS

We propose embeddings of popular interconnection networks into 2-D HOW systems. Such embeddings could prove very beneficial as HOW and related systems demonstrate significant promise in scalable parallel processing [30]. 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 V' and of each edge in the set E onto an edge, or a set of edges (a path) in E'. There are three important parameters that determine the quality of mapping. The *dilation* of a source edge in E is the number of edges in E' (the length of the path) that this edge from E is mapped onto. The *congestion* of a target edge in E' is the number of source edges mapped onto this edge in E'. The *expansion* is the ratio of the number of nodes in the set V' to that in V. We try, whenever possible, to limit the scope of the discussion to cases where the expansion is one, for the sake of cost effectiveness.

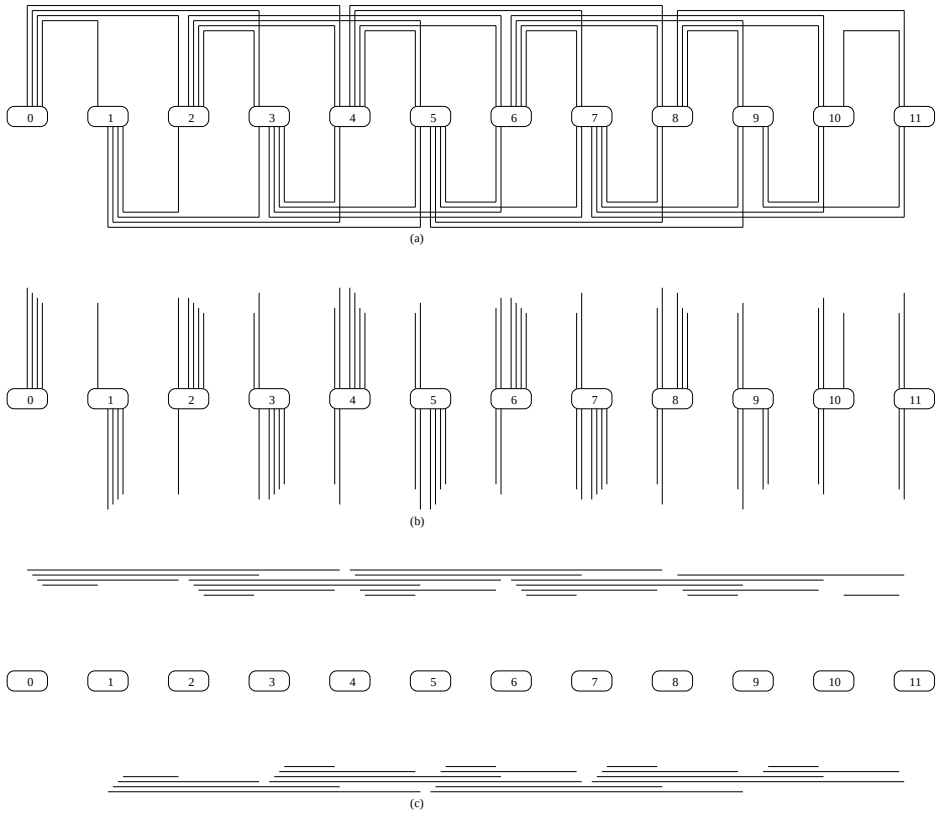


Figure 3: Colinear layout of the 1-D HOW system with 12 PEs and window size of 4, and its decomposition into printed-circuit layers using vertical and horizontal lines (restricted routing).

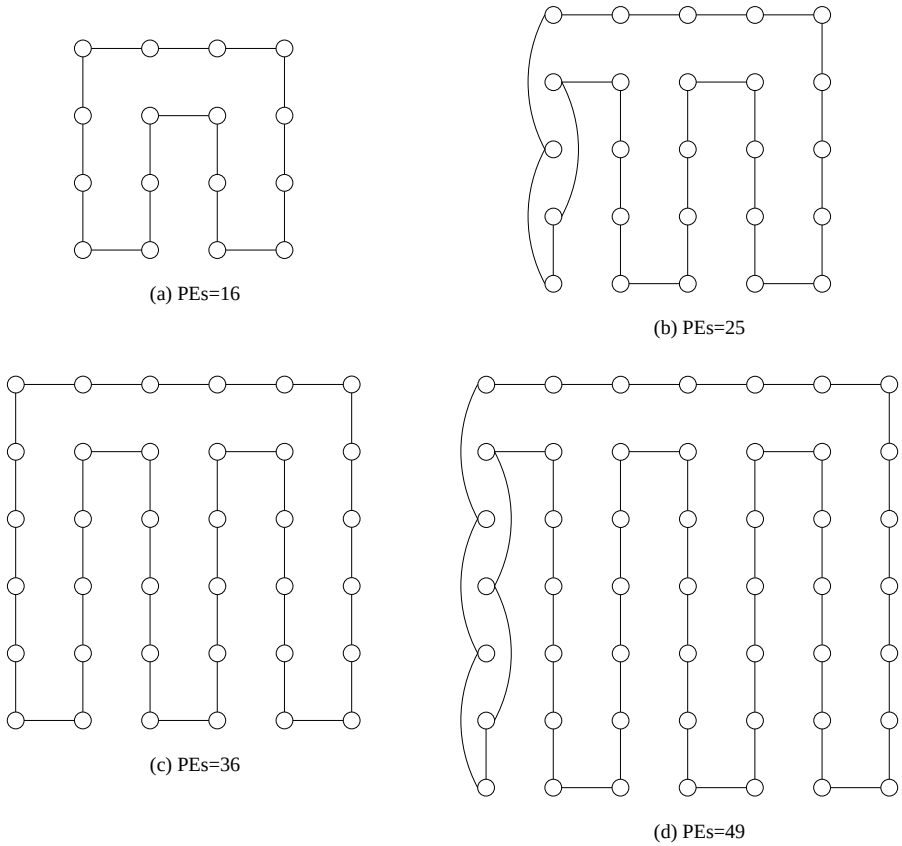


Figure 4: Optimal embeddings of rings into 2-D HOW systems.

4.1 Embedding a Ring

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 on the first column cannot be visited sequentially, but still an optimal mapping exists for $w \geq 2$, as shown in Figure 4.

If the ring has fewer nodes than the HOW system, we just use one or more links connecting nodes at distance two to bypass several nodes in the 2-D HOW system for optimal mapping, as shown in Figure 5.

4.2 Embedding a Binary Tree

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 [33]. Consider a full binary tree of depth d containing $2^d - 1$ nodes and the 2-D $HOW(\lceil\sqrt{2^d - 1}\rceil, w, 2)$ system for the smallest expansion. We assume that $w \geq 2$. The two basic building blocks used in our optimal binary tree mapping are for the 3-level tree, and are shown in Figures 6 and 7. These two building blocks and their mirror images are employed for the mapping of larger trees. For example, Figure 8 shows a mapping where the building block at the

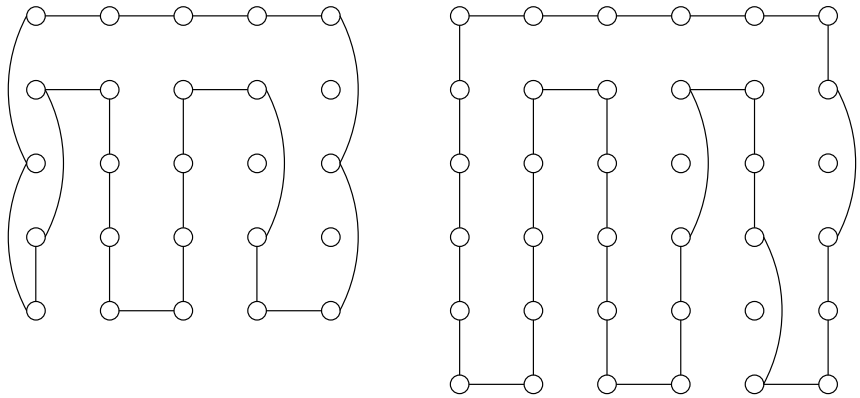
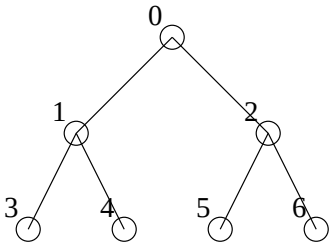
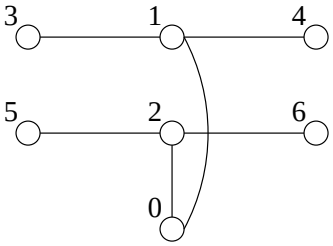


Figure 5: Embedding rings into 2-D HOW systems when the numbers of nodes in the rings are smaller than those in the HOW systems.



(a) 3-level binary tree



(b) the mapping onto the 2-D system

Figure 6: Optimal mapping of the 3-level binary tree onto the 2-D $HOW(3, 2, 2)$ system.

upper-left corner of Figure 7 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 two in this case (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 8). In general, a large binary tree of depth d is viewed as four appropriately connected subtrees of depth $d-2$ 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 9. The maximum dilation is two for binary trees with an odd number of levels. For an even number of levels, we have optimal embeddings.

4.3 Embedding a Binary Hypercube

Based on the desired expansion, we can embed a (direct binary) hypercube into a 2-D HOW system with two different methods. 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. We can embed this hypercube recursively as shown in Figure 10, where optimal mapping is derived for large windows. This mapping is derived from the classical 2-D representation of hypercubes; optimal mapping results for

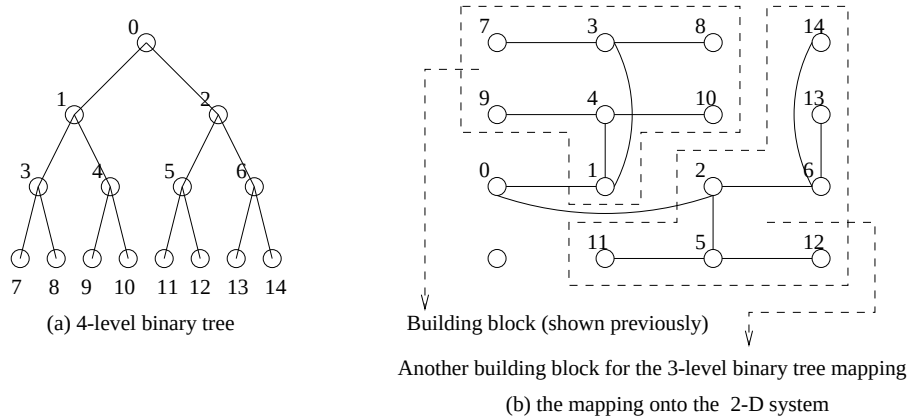


Figure 7: Optimal mapping of the 4-level binary tree onto the 2-D $HOW(4,2,2)$ system. The two distinct building blocks for the mapping of 3-level binary trees are enclosed in dotted lines.

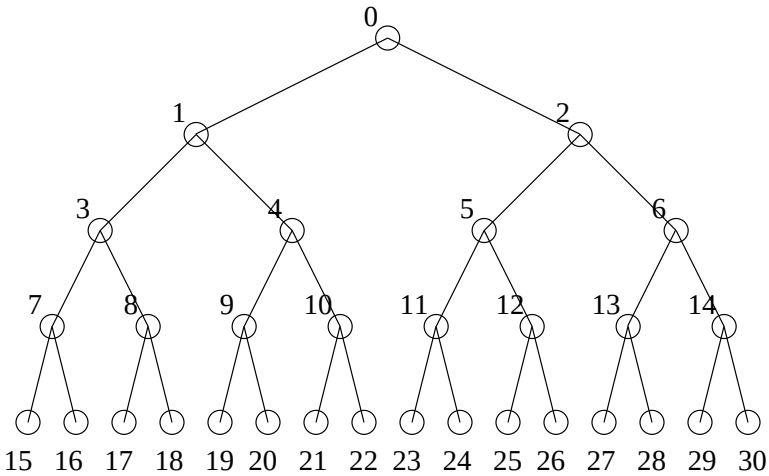
$w \geq 2^{\lceil \frac{d}{2} \rceil - 1}$. In the general case, the largest dilation of edges is $\lceil \frac{2^{\lceil \frac{d}{2} \rceil - 1}}{w} \rceil$. This method is very simple and easy to implement, but half of the nodes are wasted (i.e., the expansion is two) when d is an odd number. The maximum 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 $\lceil \frac{d}{2} \rceil - v$.

Second, in order to minimize the expansion (i.e., the number of unused HOW nodes) for an odd d, we can use another method to embed the d-D hypercube into the 2-D $HOW(p,w,2)$ system. This recursive mapping method is based on the fact that the d-D hypercube is formed from four $(d-2)$-D hypercubes. The embedding of the 3-D hypercube into the "building block" $HOW(3,2,2)$ is used recursively. As shown in Figure 11, the embedding into the building block results in only one unused node. Also, 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). Figure 11 shows the embedding of a 5-D hypercube using four 3-D hypercube building blocks mapped onto $HOW(3,2,2)$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 $HOW(3 \times 2^m, w, 2)$ for the best mapping with minimum expansion, where $m = \frac{d-3}{2}$. The expansion is equal to $\frac{9 \times 2^{2m}}{2^d}$ or $\frac{9}{8}$.

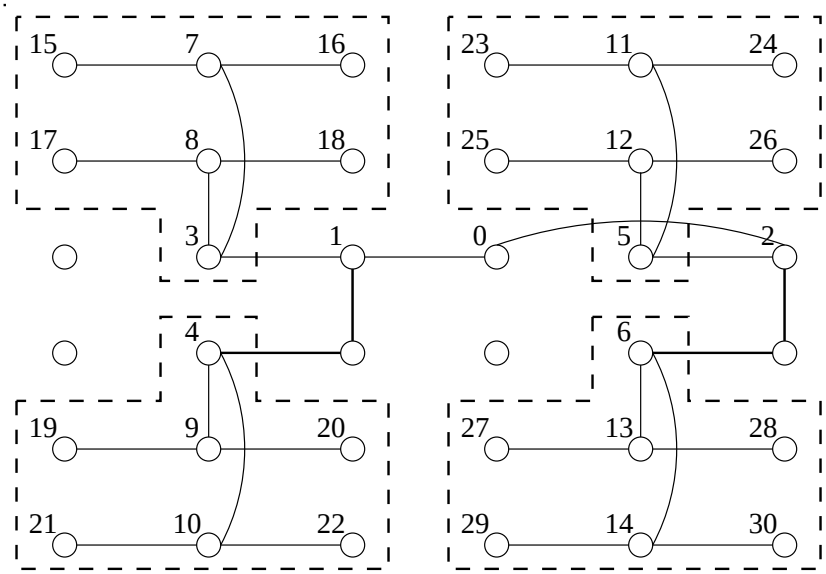
5 COMMUNICATION OPERATIONS ON 2-D HOWS

The *communication latency*, that is the time consumed to communicate a message between two nodes in the system, depends on the following parameters [17, 15, 27]:

- *Startup time (t_s)*: the time consumed by the sending node. 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 word of the message to the appropriate output communication port.

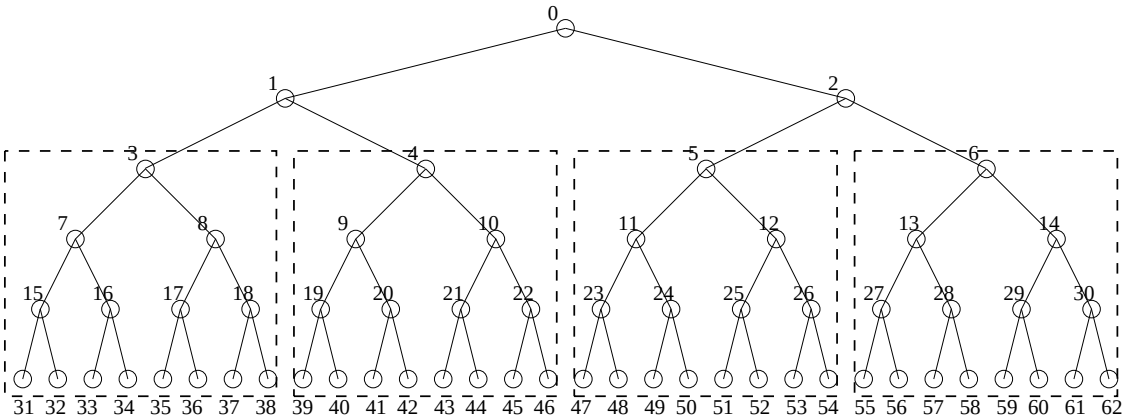


(a) 5-level binary tree

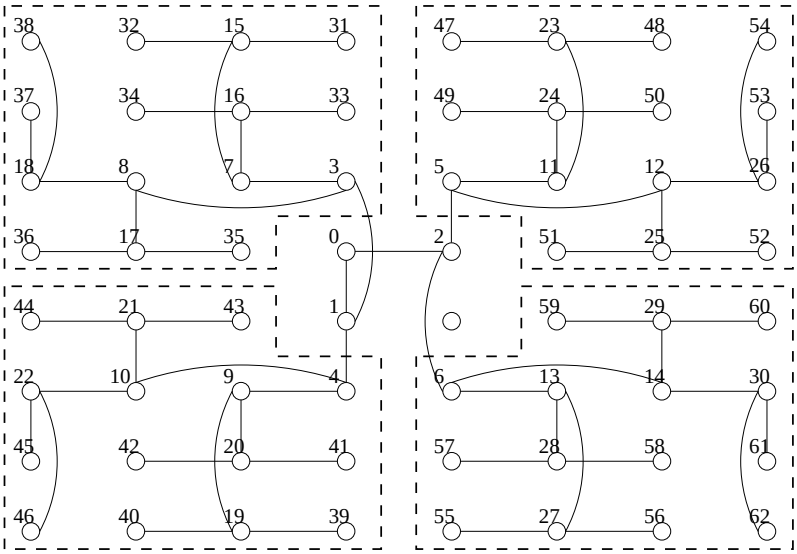


(b) the mapping onto the 2-D system

Figure 8: Mapping the 5-level binary tree onto the 2-D $HOW(6, 2, 2)$ system.



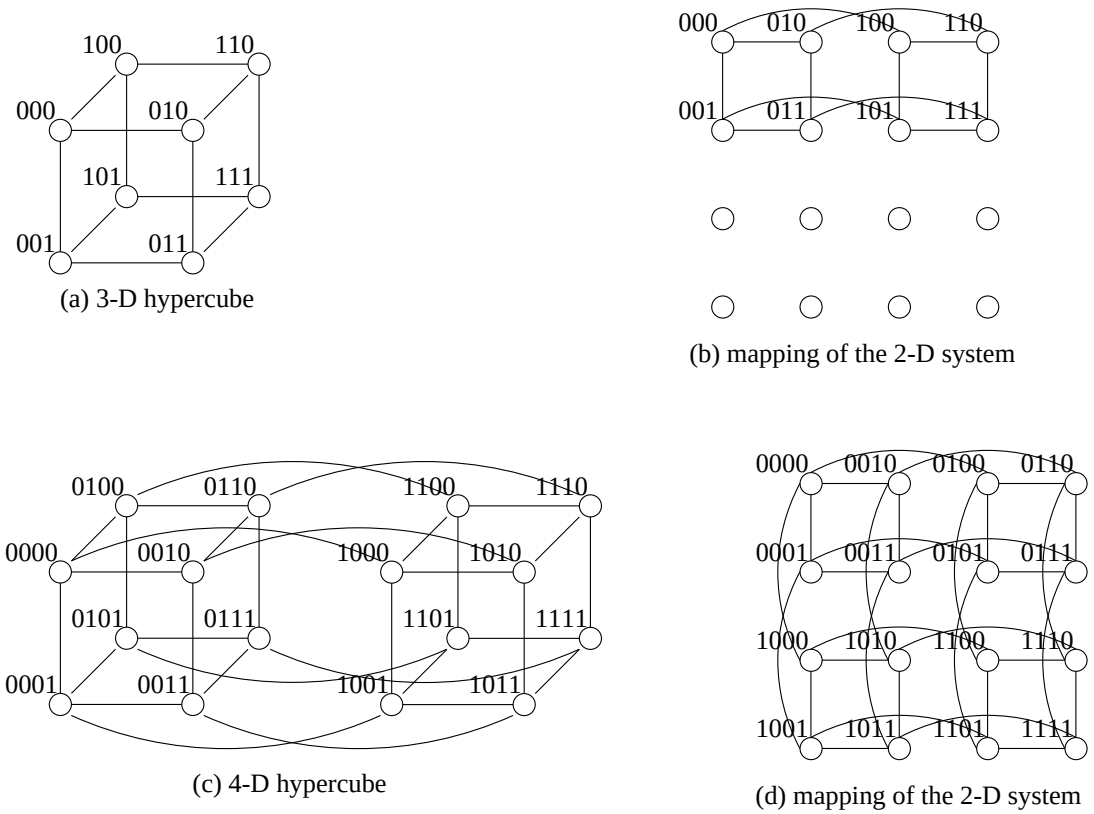
(a) 6-level binary tree



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.

(b) the mapping onto the 2-D system

Figure 9: 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 10: 3-D and 4-D hypercube embeddings into the $HOW(4, w, 2)$ system with optimal dilation (method one).

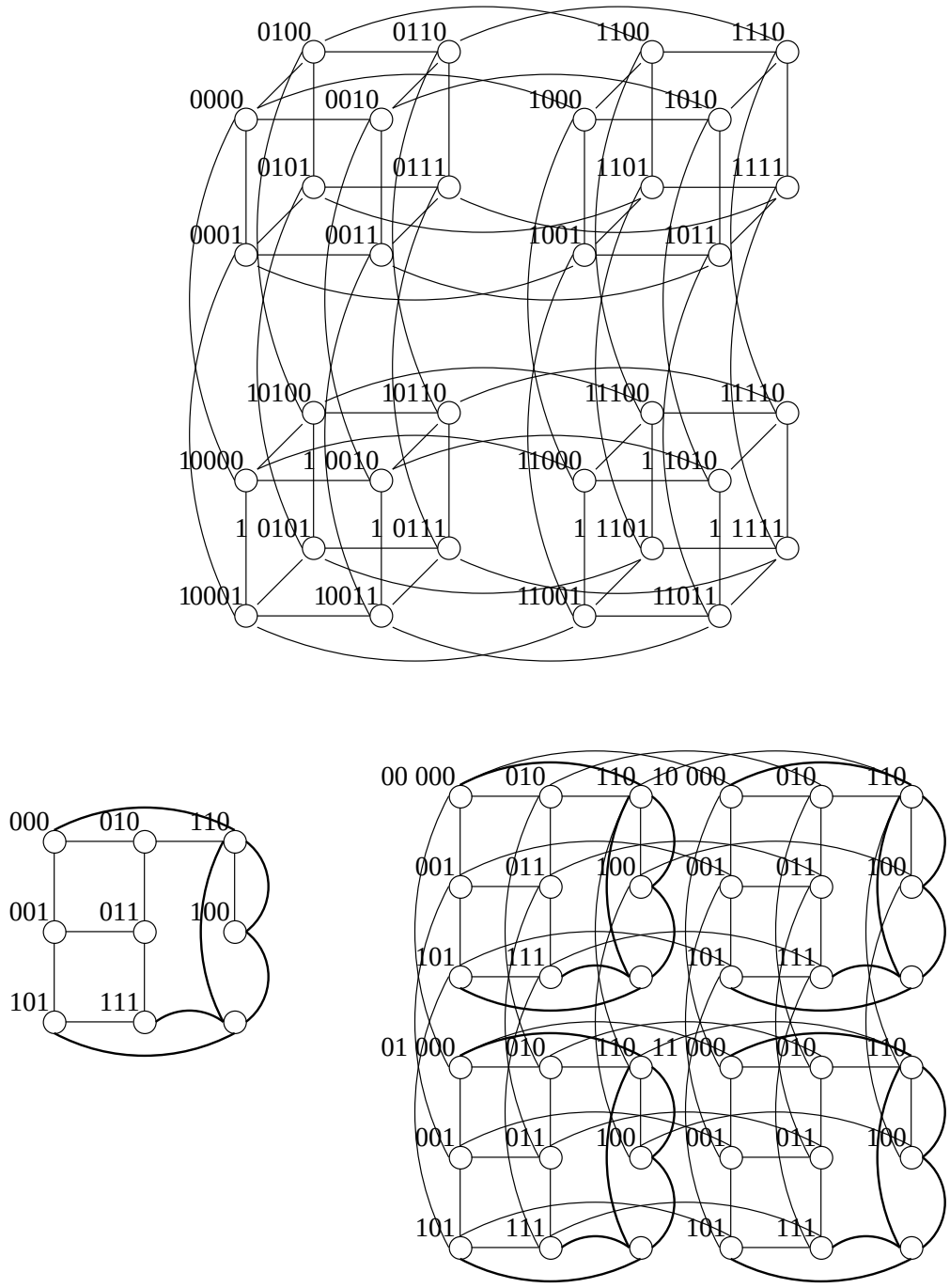


Figure 11: 5-D hypercube embedding with the second method. This figure shows the original hypercube, the embedding of the 3-D hypercube into the building block $HOW(3, 2, 2)$, and the final embedding into the 2-D $HOW(6, 3, 2)$ system.

- *Per-word channel transfer time (t_w)*: the time taken by a word to traverse a channel.
- *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 node forwards the message to the next node in the path after it has received the entire message. To reduce the communication time, *wormhole routing* divides a message into flits (flow-control digits) [11]. As the header flit advances along the chosen path, the remaining flits follow in a pipelined fashion. If the header flit encounters a channel already in use, all flits are blocked until the channel becomes available [17]. Normally, the flit size coincides with the channel width. The combining time t_c is ignored in wormhole routing.

In the following subsections we develop algorithms for the communication model where *each node can use simultaneously all of its input and output ports with different values coming/leaving into/from the ports.* This is often actually the case with real systems [6]. The analysis is done here primarily for SF routing because we can often derive the time for wormhole routing just by substituting zero for t_c. We also assume symmetric 2-D HOW systems with p node and m-word messages. $p_{i,j}$ will represent the node on row i and column j of the $HOW(\sqrt{p}, w, 2)$, where $i, j = 0, 1, 2, \cdots, \sqrt{p} - 1$.

5.1 One-to-One Communication

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 [30]. For $\sqrt{p}$ rows, $\sqrt{p}$ 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

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

With wormhole routing, the *upper bound* is

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

5.2 One-to-All Broadcasting

We first broadcast within the row of the source PE and then within all columns. The *upper bound* on the total time taken by this operation is

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

With wormhole routing, the *upper bound* is

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

assuming that the dimension to be traversed is changed just after the first flit is received. These times are asymptotically optimal because the diameter of the system is $O(\frac{\sqrt{p}}{w})$.

5.3 All-to-All Broadcasting

Nodes first exchange messages along rows, so that each node has $\sqrt{p}$ messages at the end for the nodes on its own column. Then, nodes broadcast their $\sqrt{p}$ messages along columns by repeating the same procedure $\sqrt{p}$ times within the columns. We try to use all possible channels available to pass messages. In the first stage of the broadcasting procedure for individual rows or columns, each PE sends its message to all of its neighbors. Then, in stage i, where $i = 1, 2, ..., \lceil\frac{\sqrt{p}-1}{w}\rceil - 1$ the following procedure is implemented. Assume, without loss of generality, the first row. In one direction, beginning from position iw and also involving all its successors, the PEs $0, 1, ..., (\sqrt{p} - 1 - iw - 1)$ send the messages via all their channels. In the other direction, beginning from position $(\sqrt{p} - 1 - iw)$ and also involving all its predecessors, the PEs $\sqrt{p} - 1, \sqrt{p} - 2, ..., (iw + 1)$ send their messages. If there is an overlap between these two directions, then we 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. The total time taken by this operation is given by

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

This time is optimal because each PE receives $O(mp)$ words, with up to $O(w)$ words at a time.

5.4 One-to-All Personalized Communication

Unlike one-to-all broadcasting, one-to-all personalized communication does not involve any duplication of data. We give higher priority to messages that must travel longer distances and all column and row connections for a PE are used simultaneously. It will take up to $\lceil\frac{\sqrt{p}-1}{w}\rceil m(\sqrt{p}-1)$ steps to send all messages for the last column on the respective row PE, and up to $\lceil\frac{\sqrt{p}-1}{w}\rceil m$ steps on that column. The *upper bound* on the total time is

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

which is optimal (it is the same as the time consumed by the source PE).

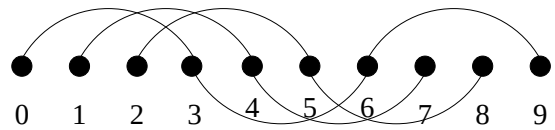


Figure 12: Linear arrays in the $HOW(10, 3, 1)$ for all-to-all personalized communication.

5.5 All-to-All Personalized Communication

This operation is also known as total exchange. Let us begin with the 1-D $HOW(\sqrt{p}, w, 1)$ operation because it will be extended later for 2-D HOWs. This operation involves a lot of message transfers. 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 12 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 nodes that belong to the aforementioned linear arrays. Every node passes all relevant 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 $s_1 = \lceil \frac{\sqrt{p}-1}{w} \rceil$ cycles to finish the initialization, which is the same as the maximum number of values to be sent from a node 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 $s_2 = \lceil \frac{\sqrt{p}-1}{w} \rceil - 1$ cycles to finish the broadcastings along the linear arrays, which is the same as the maximum number of values a node has to send; messages going farther have higher priority.

The total time taken by this operation is

$$\begin{aligned} T^{1D}_{all\_to\_all\_pers} &= t_s + m(s_1 + s_2)t_w + m(s_1 + s_2 - 1)t_c \\ &= t_s + (2\lceil \frac{\sqrt{p}-1}{w} \rceil - 1)mt_w + 2(\lceil \frac{\sqrt{p}-1}{w} \rceil - 1)mt_c = O(m\frac{\sqrt{p}}{w}) \end{aligned}$$

This time is optimal because the diameter of the 1-D system is $O(\frac{\sqrt{p}}{w})$.

The 2-D HOW implementation requires the following steps:

- Each node transmits $\sqrt{p}$ values to each of the other $\sqrt{p} - 1$ nodes on its row, to be later distributed on the corresponding columns. At the end of this step, each node 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 node transmits the values it received earlier and its own $\sqrt{p}-1$ values to the other nodes on its column. Since $\sqrt{p} - 1$ of the messages received in the first step were destined for this particular node, the number of messages to be transmitted is $(\sqrt{p} - 1)\sqrt{p} - (\sqrt{p} - 1) + (\sqrt{p} - 1) = (\sqrt{p} - 1)\sqrt{p}$.

Table 3: Exact communication times.

	Communication Times			
System	one-to-all broadcasting	all-to-all broadcasting	one-to-all-pers. communication	all-to-all-pers. communication
$HOW(\sqrt{p}, w, 2)$	$2m\lceil\frac{\sqrt{p}-1}{w}\rceil$	$m(1+\sqrt{p})\lceil\frac{\sqrt{p}-1}{w}\rceil$	$\lceil\frac{\sqrt{p}-1}{w}\rceil m\sqrt{p}$	$(2\lceil\frac{\sqrt{p}-1}{w}\rceil-1)mp$
$\log_2 p$-cube	$m(\log p)$	$m(p-1)$	$m(p-1)$	$(\log p)\ (\frac{p}{2}m)$
$GH(\sqrt{p}, 2)$	$2m$	$m(1+\sqrt{p})$	$m\sqrt{p}$	mp

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 time is

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

This time is close to the lower bound which is $O(\frac{mp}{w})$; each PE sends out $O(mp)$ words at the rate of $O(w)$ words at a time.

6 PERFORMANCE COMPARISONS

Assume that t_w is one unit of time, $t_s = 0$, and $t_c = 0$, in order to simplify the calculations. Table 3 then shows exact communications times for the three systems. The communications times for the generalized hypercube are derived from those for HOWs by assuming that $w = \sqrt{p} - 1$. The remaining figures show relevant performance comparisons. The results prove that HOWs often perform better than binary hypercubes and comparably to GHs for large values of w.

We compare the communications capabilities of 2-D HOWs, binary hypercubes [17], and 2-D GHs, all with the same number p of nodes. For the sake of simplicity, we assume store-and-forward routing. Table 4 also summarizes the performance of these systems and compares them using as cost measure the product of the "communication time" and the "node pin-out" [1]; systems with lower cost value are preferrable. The node pin-out for a network is the number of wires per node; it is the product of the node degree and the channel width (we assume constant width here). It is a very widely used measure of the VLSI cost. The cost of implementing these communications operations is asymptotically identical for HOWs and GHs; this is very important as HOWs are much easier to implement than GHs. Therefore, HOWs are proven viable networks in the field of very high performance computing.

It becomes obvious that GHs perform better than HOWs from the communication time point of view. But the GH has a fundamental design disadvantage. Let us now redefine

Table 4: Cost comparison of networks. Cost= (communication time) * (node pin-out).

Network	2-D HOW	Binary Hypercube	2-D Generalized Hypercube
Pin-out	$O(w)$	$O(\log p)$	$O(\sqrt{p})$
$T_{one\_to\_one}$	$O(m\frac{\sqrt{p}}{w})$	$O(m \log p)$	$O(m)$
$T_{one\_to\_all}$	$O(m\frac{\sqrt{p}}{w})$	$O(m \log p)$	$O(m)$
$T_{all\_to\_all}$	$O(m\frac{p}{w})$	$O(mp)$	$O(m\sqrt{p})$
$T_{one\_to\_all\_pers}$	$O(m\frac{p}{w})$	$O(mp)$	$O(m\sqrt{p})$
$T_{all\_to\_all\_pers}$	$O(m\frac{p^{3/2}}{w})$	$O(mp \log p)$	$O(mp)$
$cost_{one\_to\_one}$	$O(m\sqrt{p})$	$O(mlog^2p)$	$O(m\sqrt{p})$
$cost_{one\_to\_all}$	$O(m\sqrt{p})$	$O(mlog^2p)$	$O(m\sqrt{p})$
$cost_{all\_to\_all}$	$O(mp)$	$O(mp \log p)$	$O(mp)$
$cost_{one\_to\_all\_pers}$	$O(mp)$	$O(mp \log p)$	$O(mp)$
$cost_{all\_to\_all\_pers}$	$O(mp^{3/2})$	$O(mplog^2p)$	$O(mp^{3/2})$

Table 5: Cost comparison between the $HOW(\sqrt{p}, w, 2)$ and $GH(\sqrt{p}, 2)$ systems. Cost= (communication time) * (bisection width).

	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 cost of an interconnection network as the product of the "communication time" and the "bisection width". This is another reasonable cost measure because we should like to achieve small communication time with a small VLSI system complexity. Table 5 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 of HOWs are proportional to reductions in the value of w and this leads to predictability in their design.

We also carried out computer simulations to test the robustness of the proposed architecture. The simulation results are shown in Figure 17. The total time shown is expressed in number of communication cycles. A single communication cycle is consumed for the transmission of a message between two neighboring nodes. All messages have the same size. The uniform distribution is used to determine the location of message initiators. All messages are generated in cycle number zero and dimension-order routing is applied. Each message buffer can hold up to five messages in all simulation runs. Each case was simulated 40 times and the average time is shown in the figure. To test the architectures under many different loads and random communication patterns, the destinations are chosen randomly without

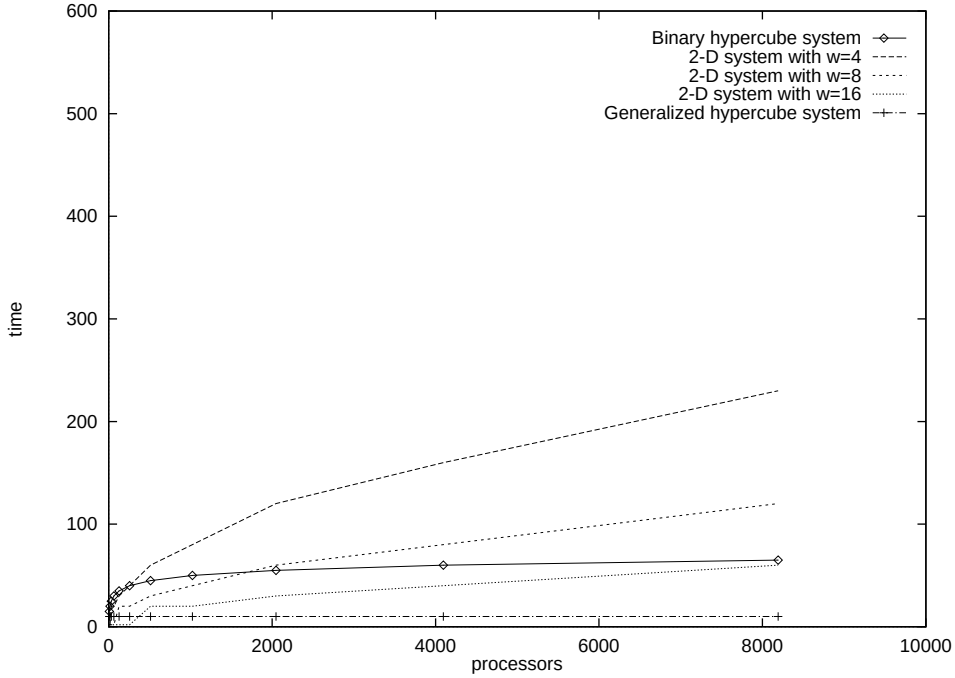


Figure 13: Comparisons for one-to-all broadcasting with message size $m = 5$ words.

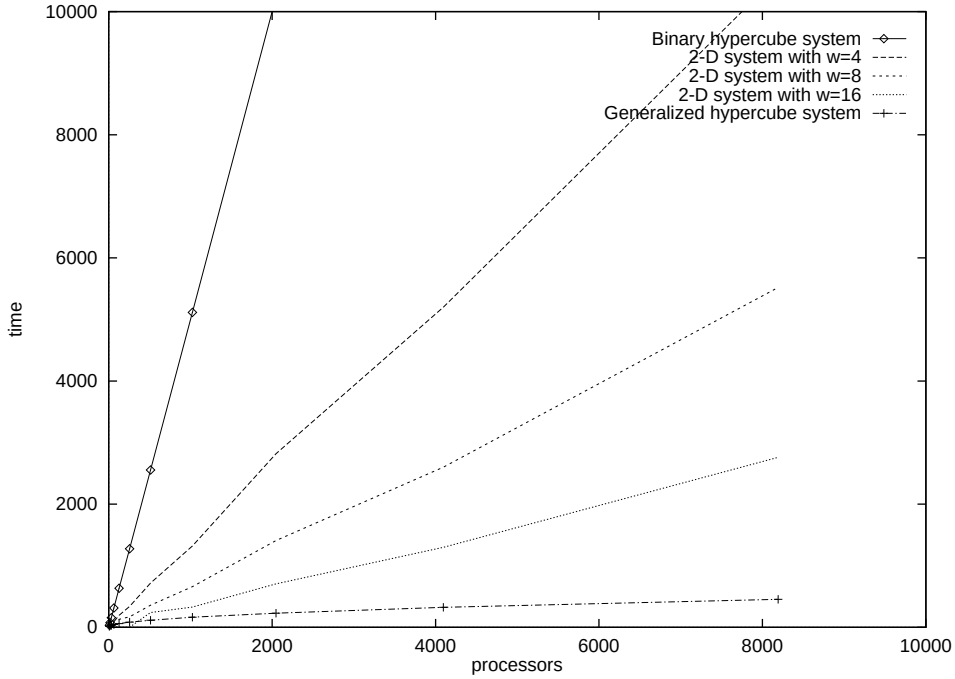


Figure 14: Comparisons for all-to-all broadcasting with $m = 5$.

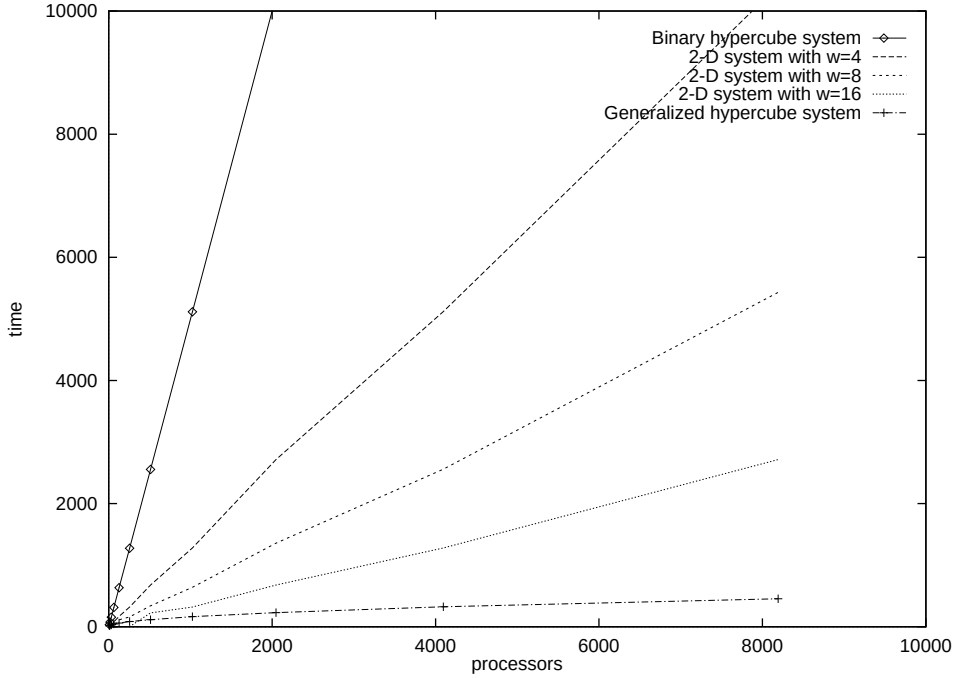


Figure 15: Comparisons for one-to-all personalized communication with $m = 5$.

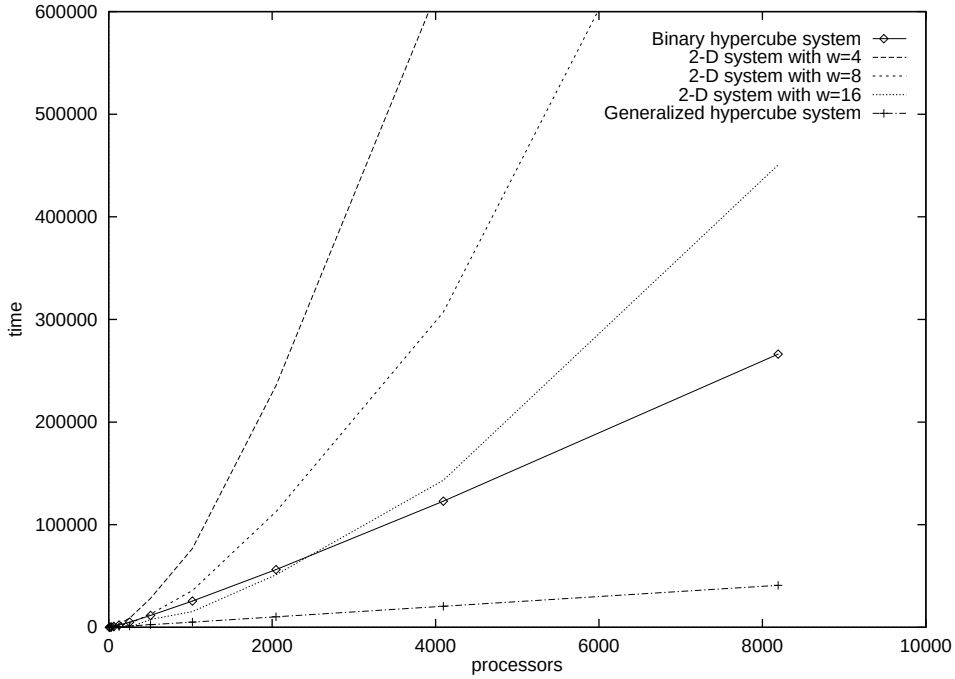


Figure 16: Comparisons for all-to-all personalized communication with $m = 5$.

R	w	M	T	%	R	w	M	T	%	R	w	M	T	%
	3	14	4.81	21.88		**3**	114	16.27	19.79		**3**	201	22.21	19.63
	4	14	3.81	21.88		**4**	114	12.2	19.79		**4**	201	16.32	19.63
	3	22	5.12	34.38		**5**	114	10.93	19.79		**5**	201	14.6	19.63
8	**4**	22	3.85	34.38		**3**	187	16.41	32.46		**6**	201	13.13	19.63
	3	31	5.3	48.43		**4**	187	12.4	32.46		**3**	251	22.24	24.51
	4	31	4.33	48.43	**24**	**5**	187	10.37	32.46		**4**	251	16.27	24.51
	3	59	5.36	92.19		**3**	279	16.76	48.43		**5**	251	14.26	24.51
	4	59	4.33	92.19		**4**	279	12.73	48.43	**32**	**6**	251	12.97	24.51
	3	51	8.41	19.92		**5**	279	10.61	48.43		**3**	334	25.41	32.62
	4	51	6.8	19.92		**3**	555	19.8	96.35		**4**	334	17.1	32.62
	3	83	9.06	32.42		**4**	555	14.51	96.35		**5**	334	15.13	32.62
16	**4**	83	7.13	32.42		**5**	555	12.8	96.35		**6**	334	13.17	32.62
	3	123	9.65	48.04		**5**	1347	22.88	32.89		**3**	499	22.86	48.73
	4	123	7.96	48.04	**64**	**7**	1347	17.3	32.89		**4**	499	17.35	48.73
	3	243	12.74	94.92		**8**	1347	15.7	32.89		**5**	499	15.6	48.73
	4	243	8.73	94.92		**8**	2019	20.28	49.29		**6**	499	13.87	48.73

R: num. rows/ columns **M:** num. of messages **%:** % of senders
w: size of window **T:** total time

Figure 17: Simulation results.

assuming any specific distribution. Random communication patterns are much more demanding of symmetric networks than regular patterns, such as permutations. Simulations are shown here under various loads, where the number of sending nodes ranges from 19.63% to 96.35% of the total number of nodes. Therefore, the behavior of the proposed architecture is also tested under very heavy loads. We can observe that the proposed architecture yields outstanding performance. In the worst cases, the total time is comparable to the diameter of the system. Therefore, the simulation results also support our claim that HOWs are capable of delivering outstanding performance.

7 CONCLUSIONS

We introduced a class of scalable architectures, namely HOWs, which are capable of very high performance. We proved that HOWs have lower cost than fat trees of similar size and higher bisection width than tori of similar cost. We also proposed graph embedding algorithms and demonstrated the implementation of various important communications operations. We also compared the performance of this class of architectures with those of the binary and generalized hypercubes for the aforementioned communications operations. Our results show that not only are our architectures scalable and feasible with current and

expected technologies, but also they perform better than the binary hypercube and comparably to the generalized hypercube for several highly demanding communications operations. Simulation results also show that HOWs can deliver very good performance under highly demanding communication loads.

Acknowledgment. The authors would like to thank the anonymous referees for invaluable comments and suggestions.

References

[1] Abraham, A. and Padmanabhan, K. (1991)Performance of Multicomputer Networks Under Pin-out Constraints. J. Paral. Distr. Comput., 12, 237-248.

[2] Agarwal, A. (1991) Limits on Interconnection Network Performance. IEEE Trans. Paral. Distr. Syst., 2, 398-412.

[3] Allen, F. et. al. (2001) Blue Gene: A Vision for Protein Science Using a PetaFlop Supercomputer. IBM Systems J., 40, 310-327.

[4] Antonio, J.K., Lin, L. and Metzger, R.C. (1993) Complexity of Intensive Communications on Balanced Generalized Hypercubes. Proceedings of Intern. Paral. Proces. Symp. (IPPS 93), Newport Beach, CA, April, pp. 387-394. IEEE Press, USA.

[5] Banerjee, P. (1994) Parallel Algorithms for VLSI Computer-Aided Design. Prentice-Hall.

[6] Bar-Noy, A. and Ho, C.-T. (1999) Broadcasting Multiple Messages in the Multiport Model. IEEE Trans. Paral. Distr. Syst., 10, 500-508.

[7] Bhuyan, L.N. and Agrawal, D.P. (1984) Generalized Hypercube and Hyperbus Structures for a Computer Network. IEEE Trans. Comput., 33, 323-333.

[8] Culler, D.E. and Singh, J.P. (1999) Parallel Computer Architecture, A Hardware/Software Approach. Morgan Kaufmann Publ., San Francisco.

[9] Dally, W.J. (1990) Performance Analysis of k-ary n-Cube Interconnection Networks. IEEE Trans. Comput., 39, 775-785.

[10] Dally, W. (1990) Network and Processor Architecture for Message-Driven Computers. In Suaya, R. and Birtwistle, G. (eds), VLSI and Parallel Computation. Morgan Kauffman, 140-222.

[11] Dally, W.J. and Seitz, C.L. (1986) The Torus Routing Chip. Distr. Comput., 1, 187-196.

[12] Dowd, P.W. (1991) High Performance Interprocessor Communication Through Optical Wavelength Division Multiple Access Channels. Proceedings of Inter. Symp. Comput. Arch. (ISCA 91), Toronto, Canada, 27-30 May, pp. 96-105. ACM Press, USA.

[13] Fragopoulou, P., Akl, S.G and Meijer, M. (1996) Optimal Communication Primitives on the Generalized Hypercube Network. J. Paral. Distr. Comput., 32, 173-187.

[14] Findings and Recommendations. High-End Computing Panel of the President's Information Technology Advisory Committee (PITAC), http://www.ccic.gov/ac/high-end-17Feb99.pdf.

[15] Gaughan, P.T. and Yalamanchili, S. (1993) Adaptive Routing Protocols for Hypercube Interconnection Networks. IEEE Comput., 12-23.

[16] Hwang, K. and Ghosh, J. (1987) Hypernet: A Communication-Efficient Architecture for Constructing Massively Parallel Computers. IEEE Trans. Comput., 1450-1466.

[17] Kumar, V., Grama, A., Gupta, A. and Karypis, G. (1994) Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/Cummings, CA.

[18] Leighton, F.T. (1983) Complexity Issues in VLSI, MIT Press, Cambridge, MA.

[19] Leiserson, C.E. (1985) Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing. IEEE Trans. Comput., C-34, 892-901.

[20] Lenoski, D. et al. (1992) The Stanford FLASH Multiprocessor. IEEE Comput., 3, 63-79.

[21] Li, X., Ziavras, S.G. and Manikopoulos, C.N. (1996) Parallel DSP Algorithms on TurboNet: An Experimental Hybrid Message-Passing/Shared-Memory Architecture. Conc. Pract. Exper., 8, 387-411.

[22] Moreira, J.E. et al (1999) A Gang-Scheduling System for ASCI Blue-Pacific. Proceedings of 7th Intern. Conf. High-Perf. Computing Networking Europe (HPCN Europe '99), Amsterdam, The Netherlands, 12-14 April, pp. 831-840. Springer, Germany.

[23] Nagarajan, R., Sankaralingam, K., Burger, D.C. and S.W. Keckler, S.W. (2001) A Design Space Evaluation of Grid Processor Architectures. Proceedings of 34th Annual International Symposium on Microarchitecture (MICRO 2001), Albuquerque, New Mexico, Austin, Texas, December, pp. 40-51. IEEE Press,USA.

[24] Patterson, D.A. et al. (1997) A Case for Intelligent RAM. IEEE Micro, 17, 34-44.

[25] Scott, S.L. and Thorson, G. (1996) The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. Proceedings of Hot Interconnects IV Symposium, Stanford, California, August, pp. 147-156. IEEE Computer Society, USA.

[26] Seitz, C.L. (1984) Concurrent VLSI Architectures. IEEE Trans. Comput., C-33, 1247-1265.

[27] Suh, Y.-J. and Yalamanchili, S. (1998) All-To-All Communication with Minimum Start-Up Costs in 2D/3D Tori and Meshes. IEEE Trans. Paral. Distr. Syst., 9, 442-458.

[28] Ullman, J.D. (1984) Computational Aspects of VLSI, Computer Science Press, Rockville, MD.

[29] Waingold, E., et al. (1997) Baring It All to Software: Raw Machines. IEEE Computer, 30, 86-93.

[30] Wang, Q. and Ziavras, S.G. (1999) Powerful and Feasible Processor Interconnections With an Evaluation of Their Communications Capabilities. Proceedings of Intern. Symp. Paral. Arch. Alg. Net. (ISPAN 99), Fremandle, Australia, 23-25 June, pp. 222-227. IEEE Computer Society, USA.

[31] Wittie, L.D. (1981) Communication Structures for Large Networks of Multicomputers. IEEE Trans. Comput., C-30.

[32] Ziavras, S.G. (1995) Generalized Reduced Hypercube Interconnection Networks for Massively Parallel Computers. In Hsu, D.F., Rosenberg, A. and Sotteau, D. (eds), Networks for Parallel Computations, American Mathematical Society, Rhode Island, 307-325.

[33] Ziavras, S.G. and Mukherjee, A. (1996) Data Broadcasting and Reduction, Prefix Computation, and Sorting on Reduced Hypercube Parallel Computers. Paral. Comput., 22, 595-606.

[34] Ziavras, S.G. (1992) On the Problem of Expanding Hypercube-Based Systems. J. Paral. Distr. Comput., 16, 41-53.

[35] Ziavras, S.G. (1995) Scalable Multifolded Hypercubes for Versatile Parallel Computers. Paral. Proc. Letts., 5, 241-250.

[36] Ziavras, S.G. (1994) RH: A Versatile Family of Reduced Hypercube Interconnection Networks. IEEE Trans. Paral. Distr. Systems, 5, 1210-1220.

[37] Ziavras, S.G. (1999) Investigation of Various Mesh Architectures with Broadcast Buses for High-Performance Computing. VLSI Design J., 9, 29-54.

[38] Ziavras, S.G., Grebel, H. and Chronopoulos, A.T. (1996) A Low-Complexity Parallel System for Gracious, Scalable Performance. Case Study for Near PetaFLOPS Computing. Proceedings of 6th Symp. Frontiers Massively Parallel Computation, Special Session New Millennium Computing Point Designs, Annapolis, Maryland, 27-31 October, pp. 363-370. IEEE Computer Society, USA.

[39] Ziavras, S.G., Grebel, H. and Chronopoulos, A.T. (1996) A Scalable/Feasible Parallel Computer Implementing Electronic and Optical Interconnections for 156 TeraOPS Minimum Performance. Proceedings of PetaFLOPS Architecture Workshop, Oxnard, California, 22-25 April, pp. 235-266. NSF, USA.

[40] Ziavras, S.G. and Krishnamurthy, S. (1999) Evaluating the Communications Capabilities of the Generalized Hypercube Interconnection Network. Conc. Pract. Exper., 11, 281-300.