Fully-Adaptive Minimal Deadlock-Free Packet Routing in Hypercubes, Meshes, and Other Networks

Gustavo D. Pifarré *†‡
e-mail: pifarre@buevm2.vnet.ibm.com

Luis Gravano *†‡
e-mail: gravano@buevm2.vnet.ibm.com

Sergio A. Felperin *†‡
e-mail: felperin@buevm2.vnet.ibm.com

Jorge L. C. Sanz ‡†
e-mail: sanz@ibm.com

Abstract

This paper deals with the problem of packet-switched routing in parallel machines. Several new routing algorithms for different interconnection networks are presented. While the new techniques apply to a wide variety of networks, routing algorithms will be shown for the hypercube, the 2-dimensional mesh, and the shuffle-exchange. The techniques presented for hypercubes and meshes are fully-adaptive and minimal. A similar technique can be devised for tori. A fully-adaptive and minimal routing is one in which *all* possible minimal paths between a source and a destination are of potential use at the time a message is injected into the network. Minimal paths followed by messages ultimately depend on the local congestion encountered in each node of the network. In the shuffle-exchange network, the routing scheme also exhibits adaptivity but paths could be up to $3 \log N$ long for an N node machine. The shuffle-exchange algorithm is the first adaptive and deadlock-free method that requires a small (and independent of N) number of buffers and queues in the routing nodes for that network.

Furthermore, all of the new techniques are completely free of deadlock situations. In dynamic message injection models, the routing methods are also ensured to be free of livelock if messages competing for resources are handled with fairness.

In contrast to other approaches in which adaptivity, deadlock and livelock freedom can be guaranteed at the expense of complex architectures, the algorithms presented in this paper require a very moderate amount of routing hardware. In particular, it will be shown that only two central queues per routing node of the network are necessary for the cases of the 2-dimensional mesh and the hypercube, and four queues for the shuffle-exchange.

This paper demonstrates that "hanging" an interconnection network from a node [Gun81, MS80, BGSS89, Kon90] is a convenient methodology for creating and visualizing routing functions and understanding deadlock-free policies for queue utilization. In some cases, interconnections can be *hung* from an arbitrary node, producing new interesting routing functions [PFGS91]. While the methods presented in this paper are for packet routing, some generalizations are possible for worm-hole routing on 2-dimensional tori [GPS91].

In addition, simulation results corresponding to hypercubes of up to $16K$ nodes are reported for both static and dynamic injection models.

* ESLAI, Escuela Superior Latino Americana de Informática, CC 3193,(1000) Buenos Aires, Argentina.

† Computer Research and Advanced Applications Group, IBM Argentina, Ing. E. Butti 275, (1300) Buenos Aires, Argentina.

‡ Computer Science Dept., IBM Almaden Research Center, San José, California.

1 Introduction.

Message routing in large interconnection networks has attracted a great deal of interest in recent years. Different underlying machine models have been used [DS86a], [RBJ88, Ran85], [Upf89, LM89], [Val88], [KS90], [NS], [Hil85]. Some fundamental distinctions among routing algorithms involve the length of the messages injected

in the network, the static or dynamic nature of the injection model, special assumptions on the semantic of the messages, architecture of the network and router, degree of synchronization in the hardware, and others.

In terms of message length, several issues have been studied concerning the ways to handle long messages (of potentially unknown size) and very short messages (typically of 100 bits). Recently, new techniques and architectures have been proposed based on worm-hole routing [DS86a], [DS86b], and packet-switched routing [KS90]. In between packet-routing and worm-hole lie some hybrid approaches, as the virtual cut-through technique [KK79].

Two subjects of long-standing interest in routing are deadlock and livelock freedom. Techniques that perform without deadlocks or livelocks have been shown on different models. Some algorithms succeed in accomplishing deadlock-free or livelock-free routing only in a probabilistic sense [KS90], [Pip84]. In other algorithms, deadlock freedom is guaranteed in a deterministic sense [DS86a], [Kon90], [RBJ88, Ran85], [LMR88], [Gel81], [Gun81, MS80].

Several techniques have been developed that avoid deadlock by defining an ordering on the critical resources, and allowing each message to progress throughout the network by occupying resources in a strictly monotonic fashion. The central idea for avoiding deadlock in the works of [DS86a], [RBJ88, Ran85], [Kon90], [BGSS89], [Gun81, MS80], and others is to order the use of resources potentially intervening in the generation of deadlocks. This idea results in the generation of a directed acyclic graph (DAG) of the resources. All DAG-based methods can be used for both worm-hole and packet routing. This methodology has been used by the authors of this paper to create a wide variety of new adaptive routing methods for hypercubes, meshes, shuffle-exchanges, cube-connected cycles, and other networks [PFGS91]. Some of the DAG's proposed in [PFGS91] will be utilized in this paper.

Most known techniques that completely avoid livelock and deadlock situations do that at the expense of some hardware resources. These hardware resources will increase with the degree of adaptivity desired in the routing of the messages. In the work of [DS86a], moderate resources are proposed for practical deterministic deadlock freedom on some networks, but routing techniques are oblivious. On the other hand, in [Kon90], an adaptive method for routing in the hypercube is proposed. This method performs well on simulations involving up to $16K$ nodes.

Some methods may become impractical for efficient routing on large interconnection networks due to either the amount of work done during routing or the required architecture resources in a node. The recent work reported in [KS90] shows a striking reduction of hardware resources by providing an adaptive deadlock-free routing algorithm dubbed *Chaos*. The method has a non-zero probability that a message will not reach its destination after t routing steps, for an arbitrary t. However, this probability tends to zero as t approaches infinity. Furthermore, the technique in [KS90] applies only to packet routing and paths followed by the messages are not necessarily minimal.

Restricting the set of available paths in the network to a subset suitably chosen is a common way to reduce the hardware resources necessary for deadlock-free routing. When stringent restrictions are applied, oblivious algorithms or methods with partial adaptivity will be obtained. This class of routing algorithms has been studied thoroughly for meshes and tori [Lei90]. On the other hand, if few restrictions are imposed on the set of possible routes generated by a routing function, impractical algorithms may result. For example, the structured buffer pool [Gun81, MS80] guarantees deadlock freedom by adding all necessary resources so that a DAG is obtained. This will result in an excessive amount of hardware necessary in a routing node and this situation will not be improved by allowing messages to depart from the DAG routes if queue space is available [MS80].

A *fully-adaptive minimal* routing scheme is one in which all possible minimal paths between a source and a destination are of potential use at the time messages are injected into the network. Paths followed by the messages depend on the traffic congestion found in the nodes of the network. For example, the minimal routing functions presented in [BGSS89] and [Kon90] are *not* fully-adaptive because several minimal routes are not allowed to take place. Full-adaptivity is a feature from which one can hope to obtain the best possible performance if no source of randomization is used. Full-adaptivity has been used by Upfal in [Upf89] to produce a deterministic optimal algorithm for routing in the multibutterfly. Multibutterflies are extremely rich in terms of the number of minimal paths between any pair of nodes.

Optimal performance cannot be obtained in some networks if oblivious routing is used. This involves both deterministic performance [BH82] and even probabilistic performance if only minimal paths are used [Val82]. On the other hand, fully-adaptive minimality with bounded-size queues has the potential of providing practical performance. Furthermore, finding determin-

istic and probabilistic bounds for static models of packet injection in adaptive routing is still an open problem for all cube-type networks.

A fully-adaptive, minimal, deadlock-free worm-hole routing algorithm for the 2-dimensional mesh has been described in [Ni90, Ni91]. Routing algorithms for worm-hole routing on general k-ary n-cubes with these characteristics have been presented in [LH91]. Recent progress done by three of the authors of this paper [GPS91] includes an algorithm for fully-adaptive minimal, deadlock- and livelock-free, worm-hole routing on 2-dimensional tori that uses fewer resources than the algorithm in [LH91] for this network. This technique is believed to be practical for the involved interconnections and the routing model because of its very moderate hardware resources, fully-adaptive minimality, deterministic assurance of deadlock and livelock freedom, and promising performance for different injection models. Also, in [GPS91] both minimal and non-minimal adaptive, deadlock- and livelock-free worm-hole routing algorithms for the hypercube have been presented.

In this paper, a number of algorithms for packet routing are shown. These techniques are fully-adaptive minimal (except for the one for the shuffle-exchange, which is not fully-adaptive), deadlock- and livelock-free and require a very moderate amount of resources in the routing nodes. The new methods are presented for hypercubes, meshes, and shuffle-exchange networks.

The organization of this paper is as follows. In Section 2, some terminology and concepts concerning static and dynamic deadlock freedom will be introduced. In Sections 3, 4, and 5, the main results of this paper are presented. In these sections, algorithms for fully-adaptive routing on hypercubes, 2-dimensional meshes, and for adaptive routing on shuffle-exchange networks will be shown. In Section 6, the functional designs of the routing node for the above three interconnections are shown. These designs give emphasis to the number of buffers sharing a physical link, and the operation and number of central queues in the node. In Section 7, the results obtained from the simulations involving hypercubes of up to $16K$ nodes are presented. Simulations on higher-dimensional hypercubes and other topologies will be reported soon.

2 Adaptive Routing and Deadlock Freedom: definitions and terminology.

In packet routing, the critical resources are the queues used to store the messages during their way towards their destinations. Deadlock will arise if and only if there exists a set of full queues occupied by messages such that all of these messages need a slot of a queue that belongs to the set in order to continue their way toward their destinations.

Each node of the network will have associated with it a certain number of queues. Each node has a pair of distinct queues, namely the injection and the delivery queues. Messages will be injected in the injection queue, and they will be consumed from the delivery queue. The routing function will be expressed in terms of the queues of each node. The set of delivery queues of all the network will be referred to as $DelivQ$. Notice that each delivery queue identifies a unique node of the network. The set of injection queues will be referred to as $InjectQ$

Each message has a destination associated with it, given by the function $Dest$: Messages $\rightarrow$ $DelivQ$.

A total routing function $\mathcal{R}$: $Queues \times DelivQ \rightarrow \mathcal{P}(Queues)$ is such that $\mathcal{R}(q,d)$ indicates which are the next possible hops of a message with destination d that is currently in q. Possibly, a delivery queue d may not be reachable from a given non-delivery, non-injection queue q. In such a case, $\mathcal{R}(q,d)$ should be equal to $\emptyset$. $\mathcal{R}$ has to verify the following constraints:

1. if $q_2 \in \mathcal{R}(q_1,d)$, then the node to which q_2 belongs is at most one hop away in the network from q_1's.
2. $\mathcal{R}$ builds a non-empty set of paths from any injection queue to any delivery queue. Furthermore, as paths are built by selecting locally each hop among the possible ones, $\mathcal{R}$ must guarantee that no message will get stuck at a dead-end. These two conditions are expressed in the following one. Let r be an injection queue and d a delivery queue. If $q_0 q_1 \ldots q_p$ is a path in D such that $q_0 = r$, and $q_{i+1} \in \mathcal{R}(q_i,d)\ \forall 0 \leq i < p$, then, there exists a path $q_p q_{p+1} \ldots q_k$ in D such that $q_k = d$, and $q_{j+1} \in \mathcal{R}(q_j,d)\ \forall p \leq j < k$.

The *queue dependency graph* (QDG) corresponding to a set of queues Q and a routing function $\mathcal{R}$ is a directed graph such that its set of vertices is Q and there exists an edge from q_i to q_j $(q_i, q_j \in Q)$ iff there exist an injection queue s and a delivery queue d such that $\mathcal{R}$

builds a route from s to d passing through both q_i and q_j, and $q_j \in \mathcal{R}\ (q_i, d)$. (This definition is related to the one presented in [DS86a] regarding *virtual channels.*) Clearly, if the QDG corresponding to a set of queues and a routing function is acyclic (i.e. it is a DAG), then, the greedy routing algorithm resulting from $\mathcal{R}$ is deadlock free.

Let $D = (Q, A_s)$ be an (acyclic) queue dependency graph. Then, Q is the set of queues and A_s the set of links between the queues. Let $d^+_{(Q,A_s)}(q) \triangleq \{q' \in Q : (q,q') \in A_s\}$ be the set of *direct successors* of q. Whenever there is no ambiguity, the subscripts will be dropped.

Every non-delivery queue has finite (independent of the size of the network) size. The delivery queues of D will have infinite size, to model the fact that messages are eventually consumed at them. $Level(q)$ is the length of the longest path between any member of *InjectQ* and q. For every q, $Level(q)$ is finite because D is acyclic.

In previous work, routing functions are built such that the resulting QDG's are acyclic. Although this condition is sufficient to guarantee deadlock freedom, it is too strong, and can be relaxed: the queue dependency graph has to be *dynamically* acyclic, i.e. cyclic wait must not arise in a dynamic environment [MS80].

This paper uses a model for such dynamically acyclic queue dependency graphs in the generation of practical routing algorithms for hypercubes, meshes, and shuffle-exchanges.

Let $A_d \subset Q \times Q$ be a set such that $A_s \cap A_d = \emptyset$, and, if $(q_1, q_2) \in A_d$, then q_2 is at most one hop away from q_1 in the network. Furthermore, it must hold that, if $(q_1, q_2) \in A_d$, then $q_1 \notin DelivQ$, and $q_2 \notin InjectQ$. This means that in the extended graph to be defined below, injection and delivery queues continue to have only that function. Although it is not necessary, it will be required that if $(q,q') \in A_d$ then $Level(q) \geq Level(q')$. This is not a restriction because if $Level(q) < Level(q')$ then (q,q') can be included in A_s, and D will still be acyclic. Now, let $\tilde{D} = (Q, A_s \cup A_d)$ be the extension of D by A_d. Sometimes, D will be called the underlying DAG of $\tilde{D}$. Note that $\tilde{D}$ is not necessarily a DAG. In the following, A_s will be called the *static link set* and A_d will be called the *dynamic link set*. Let $\tilde{\mathcal{R}}$ be a routing function on $\tilde{D}$, observing the following conditions: $\forall q, q' \in Q, d \in DelivQ$:

1. If $q' \in \tilde{\mathcal{R}}\ (q,d)$, then $(q,q') \in A_s \cup A_d$.
2. $\mathcal{R}\ (q,d) \subseteq \tilde{\mathcal{R}}\ (q,d)$.
3. If $q' \in \tilde{\mathcal{R}}\ (q,d)$ and $q' \notin \mathcal{R}\ (q,d)$ then $\mathcal{R}\ (q',d) \neq \emptyset$.

   This means that if a message can be routed along a dynamic link, it will still have the possibility of taking a static link as a next step towards its destination. Therefore, at any moment, every message has a static-link path that takes it to its destination. In other words, every message will be able to progress towards its target queue through the underlying DAG.

Let $\tilde{\mathcal{R}}$ be a routing function and $\tilde{D}$ be the QDG associated with it. Furthermore, suppose that D is the underlying DAG of $\tilde{D}$. The following greedy algorithm can be used to route messages over $\tilde{D}$ from the injection to the delivery queues.

```
Route(q)
/* q is the queue executing the algorithm */
(01)  select q' ∈ d+_D̃ (q) :   ( not Full (q')
      and q' ∈ R̃ (q, Dest (Head (q))))
(02)  Insert (Head (q), q')
(03)  RemoveHead (q)
```

It is supposed that once a q finds and selects some q' verifying the condition in line (01) it gains the access to a place in q', and can execute lines (02) and (03) of the algorithm above. Note that `select` may return a q' satisfying condition in line (01) according to any criterion, as long as it does so if the set of queues satisfying (01) is not empty.

The proof of the deadlock freedom of this algorithm is easy, and it can be found in [PGFS91].

3 Hypercube Algorithm.

In this Section, a fully-adaptive minimal routing algorithm for the hypercube will be presented. A routing function will be built that uses dynamic links. So, the QDG associated with this routing function will have cycles. As said above, this routing function should be regarded as an extension of an acyclic routing function (i.e. a routing function whose QDG is acyclic) so as to guarantee that the routing algorithm is acyclic. Next, this *underlying* routing function, and how to extend it to achieve the final one will be described.

The routing function that results from routing over the hypercube as hung from node $0 \ldots 0$ will be used as the underlying acyclic function. This routing algorithm has been presented in [BGSS89], for implementing virtual barriers on the hypercube. A similar idea has been

used in [Kon90] for implementing a minimal adaptive routing algorithm on the hypercube. The idea on which this *hanging* algorithm is based is the following. The algorithm consists of two phases. In phase A, each message travels as moving downwards through the network, always moving towards its destination, as much as possible. So, in this phase, each message starts heading to node $1 \ldots 1$ (which happens to be the node that is opposite to node $0 \ldots 0$). So, in phase A, each message turns the incorrect 0s in the address of its source node into 1s.

In phase B, every message arrives at its destination by following an upwards path. In this phase, messages move towards node $0 \ldots 0$ again. So, in this phase, each message turns the incorrect 1s of its source address into 0s. Therefore, all the required corrections are terminated at the end of this phase. Consequently, each message arrives at its destination.

The following implementation of this algorithm is such that the corresponding QDG is acyclic. Each node n should have two queues, $q_{A,n}$ (associated with phase A), and $q_{B,n}$ (associated with phase B), as well as an injection queue i_n and a delivery queue d_n, as discussed above. During the first phase, messages move through the q_A queues of the nodes they visit. When a message switches phase, it has to start moving through the q_B queues of the nodes visited. The QDG resulting from this implementation is acyclic. Therefore, the algorithm associated with it is deadlock-free. See Figure 1 for the QDG of a 3-hypercube, in which the injection and delivery queues have not been drawn.

As messages are forced to correct first the incorrect 0s into 1s and only afterwards the incorrect 1s into 0s, congestion around node $1 \ldots 1$ is likely to take place.

Now, dynamic links will be added to the QDG in such a way that they will allow messages to change incorrect 1s into 0s while being in phase A if the message finds place in the q_A queue of the corresponding node, at a certain moment. The resulting algorithm, which is deadlock- free (see Section 2), is the following. Each message is injected, and starts moving through the q_A queues of the different nodes it visits (phase A) while it has any 0 to correct into 1. After performing the last 0 to 1 correction, the message will enter the q_B queue of the corresponding node, and will start doing the 1 to 0 corrections needed until it arrives at its destination node.

With the queue policy just outlined, the resulting routing algorithm is deadlock-free, and allows each message to wait for correcting any of the possible dimensions it has to correct. Consequently, there will be no particular congestion near node $1 \ldots 1$ as in the previous algorithm, as messages are allowed to move upwards even if they are in phase A, as a result of the newly added dynamic links. This algorithm requires only two queues per node, plus the injection and delivery queues, and is fully-adaptive.

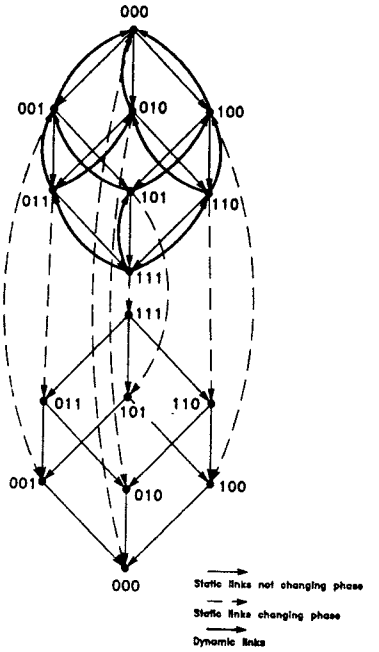

Figure 1: A 3-hypercube hung from node 000 with dynamic links.

The routing function. In the following, $\mathcal{E}^i(k)$ is the number that has the same binary representation as k but for the i^{th} digit, k_i.

Formally, the routing function is the following:

$$\tilde{\mathcal{R}}\ (i_s, d_m) = \begin{cases} \{q_{A,s}\} \\ \quad \text{if } \exists j : s_j \neq m_j \text{ and } s_j = 0 \\ \{q_{B,s}\} \\ \quad \text{otherwise} \end{cases}$$

$$\tilde{\mathcal{R}}\ (q_{A,n}, d_m) = \begin{cases} \{q_{A,\mathcal{E}^t(n)} : n_t \neq m_t\} \\ \quad \text{if } \exists j : n_j \neq m_j \text{ and } n_j = 0 \\ \{q_{B,n}\} \\ \quad \text{if } n \neq m \text{ and} \\ \quad\quad \forall j : (n_j \neq m_j \Rightarrow n_j = 1) \\ \{d_m\} \\ \quad \text{if } n = m \end{cases}$$

$$\tilde{\mathcal{R}}\ (q_{B,n}, d_m) = \begin{cases} \{q_{B,\mathcal{E}^t(n)} : n_t \neq m_t\} \\ \quad \text{if } n \neq m \\ \{d_m\} \\ \quad \text{if } n = m \end{cases}$$

Then, the following theorem can be easily proved.

Theorem 1 *The routing algorithm just described for the hypercube is fully-adaptive, minimal, deadlock- and livclock-free, and can be implemented using 2 queues per node, plus an injection and a delivery queue per node.*

Simulation results of this algorithm for hypercubes of up to $16K$ nodes are reported in Section 7.

4 Mesh Algorithm.

A routing function for the mesh will be presented here in terms of the ideas of dynamic links. The scheme is minimal and deadlock free. Although the following description focuses on 2-dimensional meshes, the technique can be easily generalized for k-dimensional meshes, for any arbitrary k.

The key idea is to have two phases: in phase A the messages approach to their destination visiting nodes in such a way that if a message passes from (x, y) to (x', y') in one routing step, then $x < x'$ or $y < y'$. In phase B, messages visit nodes with lower number instead of those with higher number. In other words, the mesh is hung from node $(0, 0)$ in phase A and the messages visit nodes with higher level, where the level of (x, y) is $x + y$. In phase B, the mesh is hung from node $(n - 1, n - 1)$ and the nodes are visited in decreasing level order. A message changes from phase A to phase B if it has nothing to correct in phase A. This scheme can be implemented using two queues in each node, q_A for phase A messages and q_B for phase B messages. In this way, the scheme is deadlock free, because the queue dependency graph is acyclic.

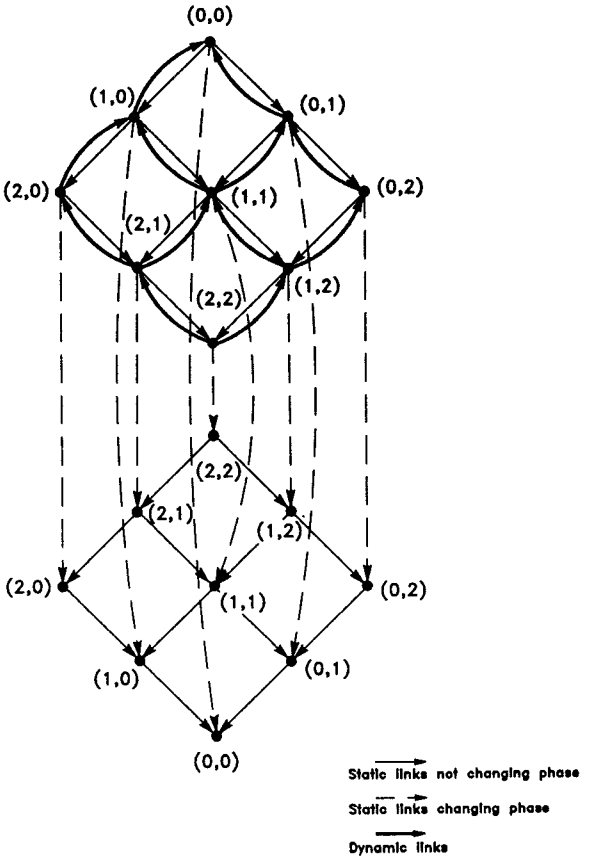

Figure 2: A 3-mesh hung from node $(0, 0)$ with dynamic links.

The routing function. [1]

$$\mathcal{R}\left(i_{(x,y)}, d_{(z,w)}\right) = \begin{cases} q_{A,(x,y)} \\ \quad \text{if } z > x \text{ or } w > y \\ q_{B,(x,y)} \\ \quad \text{if } z \leq x \text{ and } w \leq y \end{cases}$$

[1] In this section, $\mathcal{R}\ (a, b)$ is the set of all the right members satisfying the associated condition involving a and b. The same applies to the definition of $\tilde{\mathcal{R}}$ below.

$$\mathcal{R}\ (q_{A,(x,y)}, d_{(z,w)}) = \begin{cases} d_{(x,y)} \\ \quad \text{if } x = z \text{ and } y = w \\ q_{A,(x+1,y)} \\ \quad \text{if } z > x \\ q_{A,(x,y+1)} \\ \quad \text{if } w > y \\ q_{B,(x,y)} \\ \quad \text{if } z \leq x \text{ and } w \leq y \end{cases}$$

$$\mathcal{R}\ (q_{B,(x,y)}, d_{(z,w)}) = \begin{cases} d_{(x,y)} \\ \quad \text{if } x = z \text{ and } y = w \\ q_{B,(x,y-1)} \\ \quad \text{if } w < y \\ q_{B,(x-1,y)} \\ \quad \text{if } z < x \end{cases}$$

This routing function allows some degree of adaptivity. But suppose that some message starts from node (x, y) towards its destination (v, w), and let $v < x$ and $w > y$. Following the function above, this message has only one path, namely correct its second dimension, change phase and correct its first dimension. So, it has no adaptivity at all.

In the following, this scheme will be extended to a fully adaptive one, that is still deadlock free and uses the same number of queues.

This is done by allowing every message in phase A to pass to queue q_A of any neighboring node (and not only to those of higher level) *if it still has some descending path to pass through*. The phase change mechanism is the same as in the previous scheme. In phase B, the messages still have to go through ascending paths.

The routing function.

$$\tilde{\mathcal{R}}\ (i_{(x,y)}, d_{(z,w)}) = \begin{cases} q_{A,(x,y)} \\ \quad \text{if } z > x \text{ or } w > y \\ q_{B,(x,y)} \\ \quad \text{if } z \leq x \text{ and } w \leq y \end{cases}$$

$$\tilde{\mathcal{R}}\ (q_{A,(x,y)}, d_{(z,w)}) = \begin{cases} d_{(x,y)} \\ \quad \text{if } x = z \text{ and } y = w \\ q_{A,(x+1,y)} \\ \quad \text{if } z > x \\ q_{A,(x-1,y)} \\ \quad \text{if } z < x \text{ and } w > y \\ q_{A,(x,y+1)} \\ \quad \text{if } w > y \\ q_{A,(x,y-1)} \\ \quad \text{if } z > x \text{ and } w < y \\ q_{B,(x,y)} \\ \quad \text{if } z \leq x \text{ and } w \leq y \end{cases}$$

$$\tilde{\mathcal{R}}\ (q_{B,(x,y)}, d_{(z,w)}) = \begin{cases} d_{(x,y)} \\ \quad \text{if } x = z \text{ and } y = w \\ q_{B,(x,y-1)} \\ \quad \text{if } w < y \\ q_{B,(x-1,y)} \\ \quad \text{if } z < x \end{cases}$$

This new scheme is more adaptive than the first one described above. It can be implemented using only two queues, one for each phase, and it is still deadlock free. This can be proved using the ideas of dynamic links exposed in Section 2. Note that in the first phase the routing function $\tilde{\mathcal{R}}$ is defined as if the mesh were not hung.

Then, the following theorem can be easily proved.

Theorem 2 *The routing algorithm just described for the mesh is fully-adaptive, minimal, deadlock- and livelock-free, and can be implemented using 2 queues per node, plus an injection and a delivery queue per node.*

A fully-adaptive and minimal routing technique for packet-switching over tori can be achieved using 4 queues per node (plus an injection and delivery queue per node) following an idea similar to the one presented in [GPS91] for worm-hole routing over tori.

5 Shuffle-Exchange Algorithm.

In [PFGS91], a deadlock-free routing technique for the shuffle-exchange network using only a constant number of virtual channels per link has been presented. Next, a description of a modification of that technique for packet switching is given, followed by a possible extension using dynamic links to achieve adaptivity.

First, consider a 2^n-node shuffle-exchange network as without the exchange links. Each connected component of the graph will be called a *shuffle cycle*. Note that every node in a shuffle cycle has the same number of 1s in its binary address. Then, the *level* of a shuffle cycle can be defined as the number of ones in the address of any of its nodes. The idea of the algorithm is to break the shuffle cycles using the technique presented in [DS86a] in the context of worm-hole routing, and then, visit the cycles so as to avoid deadlock. Any node of a cycle can be chosen to break it.

The routing strategy can be defined in two phases. In the first one, messages can move from one shuffle cycle to another whenever the new cycle has higher level. In the second phase, messages visit the shuffles cycles in

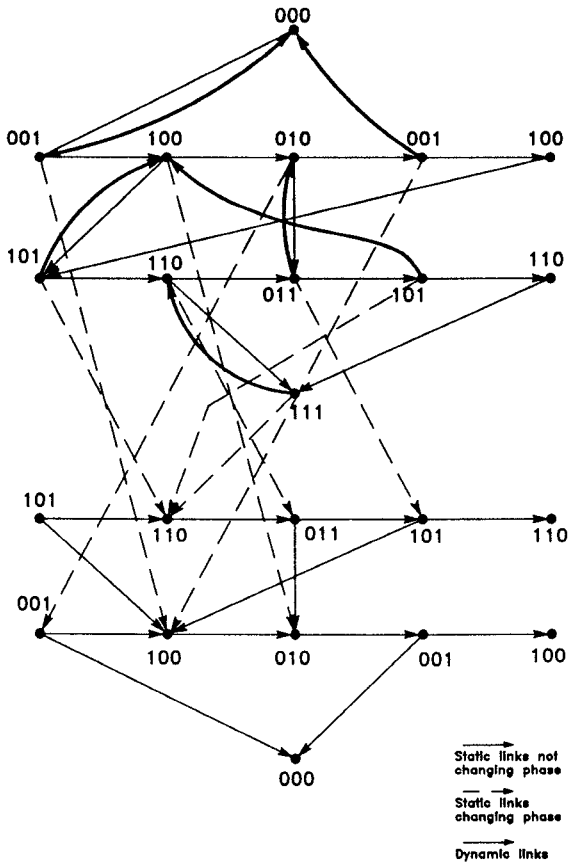

Figure 3: A 3-shuffle-exchange hung from node 000 with dynamic links.

decreasing order with respect to their level. The routing algorithm consists of visiting the dimensions of the address to correct twice, once in each phase. In each phase, dimensions are visited using the shuffle links. Consequently, every path has at most $3n$ steps: at most $2n$ shuffle steps and at most n exchange steps (see Figure 3).

After going through a shuffle link, every message has to know which dimension of the destination corresponds to the current least significant bit so as to know whether the least significant bit has to be corrected or not. So, each message must record the number of shuffle links it has already traversed. This is necessary to compare the least significant bit of the current node address with the corresponding bit of the destination address so as to decide what to do as the next step. If these bits disagree, that dimension will have to be corrected at that step or not depending on the phase the message is in. In the first phase, a dimension will be corrected if it has to be changed from 0 to 1. Note that this restriction implies that the new cycle has higher level. In the second phase, the reverse direction of the exchange links is used. Only will a change from 1 to 0 be allowed. So, the level of the cycles that are visited decreases during the second phase.

The routing function that has just been described needs only two queues per node for breaking the shuffle cycles. It is necessary to break the shuffle cycles twice: once for each phase. Therefore, each node will have 4 queues, and an injection and a delivery queue.

The messages can either be consumed as soon as they arrive at their destinations for the first time, or when arriving at their destinations after finishing the $2n$ shuffle transitions.

Next, the modification of the routing function described above by adding dynamic links is presented. Basically, the main change introduced is that a message will be allowed to traverse an exchange link that corrects the current dimension from 1 to 0 even if the message is in its first phase. In other words, a message will be allowed to correct a 1 to 0 if it happens to find place to do it during the first phase. If not, that dimension will have to be changed during the second phase. As a result of these changes, the resulting routing algorithm is adaptive, as a given message may take alternative paths as a consequence of local congestion: e.g. it may or may not correct a 1 into a 0 during the first phase (see Figure 3). See [PGFS91] for a formal and more detailed definition of the routing function.

Then, the following theorem can be easily proved.

Theorem 3 *The routing algorithm just described for the 2^n-node shuffle-exchange network is adaptive, deadlock- and livelock-free, and can be implemented using 4 queues per node, plus an injection and a delivery queue per node. Furthermore, the route each message takes has at most $3n$ steps.*

6 The design of the node.

In this Section, a possible node model to implement the routing algorithms presented in Sections 3, 4, and 5 will be presented. The node models are a modification of the one presented in [Kon90] to implement a partially adaptive routing algorithm for the hypercube.

As described above, a message can move from a queue to another queue following two types of transitions, either through dynamic links or through static links, following the terminology used in Section 2. There exists a dynamic or static link between a pair of queues only if these queues are at distance at most one in the physical network, i.e. if either the queues are in the same

node or at adjacent nodes (nodes connected by a physical link in the network). If $(q_1, q_2) \in A_s \cup A_d$, then q_2 will receive messages from q_1. So, there must exist a physical connection between q_1 and q_2. If q_1 and q_2 belong to adjacent nodes, then this physical connection is the physical link between the two nodes. On the other hand, if q_1 and q_2 belong to the same node, then, there must exist an internal connection between the two queues so as to allow internal passage of message within the nodes. Given a queue q, some fair policy must be implemented so as to guarantee fair access to q to all the resources that may want to access q.

Each node will have both an injection and a delivery queue, as explained above, as well as all the queues used by the routing algorithm. Each physical link will have associated with it input and output buffers. In general, there will be two types of buffers associated with each physical link: those associated with dynamic links and those with static links. Consider link j, incident to nodes n and n'. If traffic corresponding to dynamic links can enter node n from node n' through link j, then link j will have an input buffer in node n and an output buffer in node n' associated with the dynamic transitions. So, when a message wants to go out of node n' through link j via a dynamic transition, it will be placed in the output buffer corresponding to dynamic traffic of link j and it will arrive at the input buffer in node n that is associated with both dynamic transitions and link j. If traffic corresponding to static links can enter queue q in node n through link j, then link j will have an input buffer associated with queue q in node n and an output buffer associated with queue q in node n'. So, if some queue q' in node n' wants to send a message through link j to queue q via a static transition, then it will place the message in the output buffer corresponding to link j and queue q in node n'.

Figures 4, 5 and 6 illustrate the node model corresponding to the algorithms presented in Sections 3, 4 and 5, respectively. A more detailed description of these models can be found in [PGFS91].

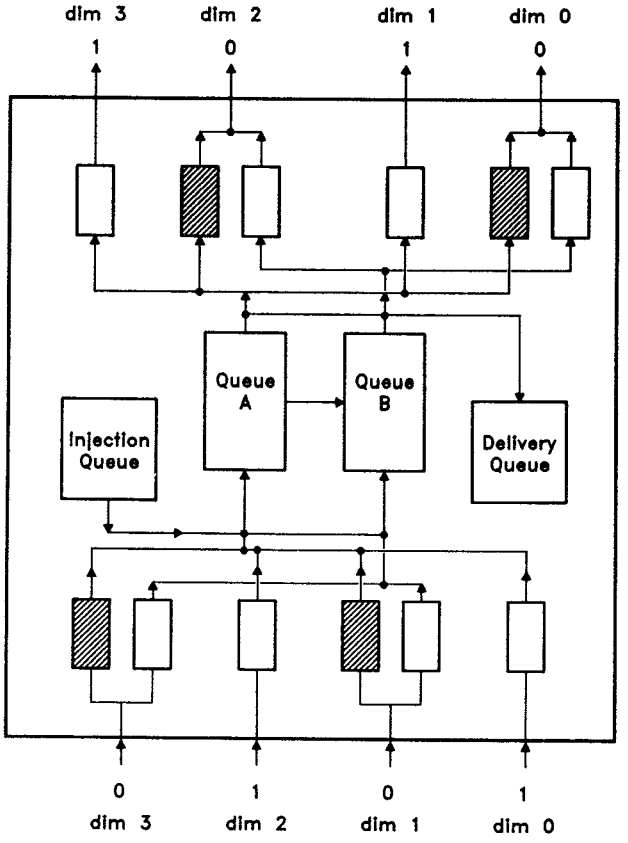

Figure 4: Node 0101 of the 4-Hypercube.

7 Simulation of the algorithm.

In this Section, simulations results of the routing algorithm proposed will be shown for the hypercube.

7.1 The activity of the node.

In the simulations presented below, each node is supposed to have an injection queue of size 1. The size of

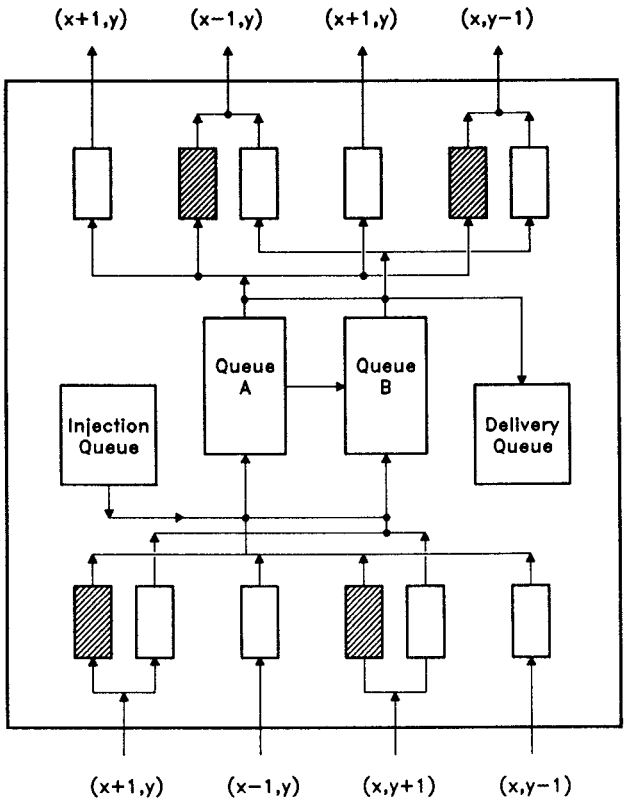

Figure 5: The node for the Mesh.

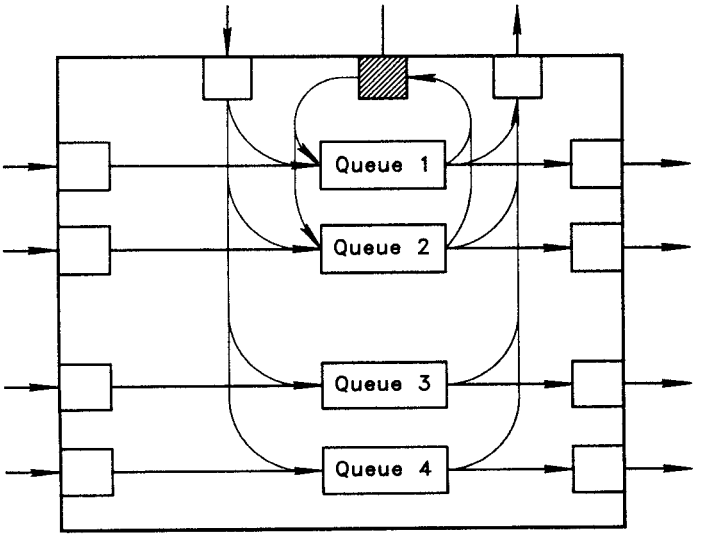

Figure 6: The node for the Shuffle-Exchange.

q_A and q_B queues has been fixed arbitrarily to 5. The idea is to have a queue size that *does not* change with the size of the network. Following the node description given in Section 6, nodes and links should work in a synchronous manner, but are independent of each other. So, each routing cycle consists of one node cycle and one link cycle. In the node cycle, each node fills its output buffers from low to high dimensions, taking messages from the queues in FIFO order. This means that if two messages want to enter the same buffer, the first one in the queue in FIFO order will get it. Then, the node reads its input buffers and its injection buffer and moves their messages to the required queues, if there is place to do so. This is carried out in a fair way. It should be noted that it takes a message at least *two* routing steps to go through a node: one to go from the input (or injection) buffer to some queue and another to go from the queue to the output buffer.

During the link cycle, each link tries to send a packet in each direction. Note that some links have associated two output buffers. As only one packet can use the link during a single cycle, packets may have to wait in the output buffers. Of course, a packet can go through a link only if the corresponding input buffer (on the other side of the link) is empty.

The simulations that have been performed show the behavior of the algorithm for many different injection models and communication patterns, described by the following parameters:

- **Injection Model:** The injection model can be either dynamic or static.
  - In the *dynamic* model, each node tries to inject a message in the network in every cycle with some probability λ. The simulations have been run for $\lambda = 1$. The dynamic injection of randomly destined packets models the situation in which the nodes communicate with each other independently. The dynamic injection of messages with the same destination is a very useful pattern of communication that models a coarse-grain parallel program with structured communication patterns. In the first case, the destinations of messages from a given source are chosen at random. In the second case, a permutation σ is chosen in advance, and every node i selects $\sigma(i)$ as destination for every message it injects into the network.
  - The injection is *static* if each node has an a priori fixed number of packets to inject in the network. The simulations have been run for

both 1 and $\log N$ packets at each node.

For both models, the average latency and maximum latency messages suffer is measured. For dynamic injection, also the effective injection rate is measured. The effective injection rate is defined as the ratio between the number of times the nodes succeeded in injecting messages into the network and the number of times the nodes attempted to inject.

- **Communication Pattern:** The following communication patterns have been tried:
  - **Random Routing:** The destination of each message is chosen randomly[2]. It should be noted that this pattern of communication does not necessarily generate permutations.
  - **Complement:** The destination of each message is the node whose binary address is the complement of the address of its origin.
  - **Transpose:** If $n = \log N$ is even, the transpose of the binary address $b_{n-1}, \ldots, b_{\frac{n}{2}}, b_{\frac{n}{2}-1}, \ldots, b_0$ is $b_{\frac{n}{2}-1}, \ldots, b_0, b_{n-1}, \ldots, b_{\frac{n}{2}}$. If n is odd, its central bit remains unchanged and the address is modified as before.
  - **Leveled Permutation:** It has been defined that the level of a node is its Hamming weight. A leveled permutation is one in which every node sends messages to a node in its same level. In [FCS90] it has been reported that congestion may arise for this sort of permutations in an oblivious routing technique where minimal paths are chosen at random.

In tables 1 to 8 the results of the simulations that have been performed for static injection are presented. Tables 9 to 12 show the results for dynamic injection. It should be emphasized that node activities are considered to take two time cycles. In the tables, N is the number of nodes of the hypercube, n is the number of dimensions of the hypercube, L_{avg} is the average latency of the messages, L_{max} is the maximum latency any packet experienced, and I_r (%) is the effective injection rate.

8 Acknowledgments.

We would like to thank S. Konstantinidou, C. T. Ho, J. Bruck and R. Cypher for their many suggestions and comments.

[2]Node p will choose the destination of every message it injects with uniform probability over the set $V - \{p\}$.

References

[BGSS89] Y. Birk, P.B. Gibbons, D. Soroker, and J.L.C. Sanz. A simple mechanism for efficient barrier synchronization in MIMD machines. RJ 7078 (67141) Computer Science, IBM Almaden Research Center, October 1989.

[BH82] A. Borodin and J.E. Hopcroft. Routing, Merging and Sorting on Parallel Models of Computation. In *Symposium on Theory of Computing*, pages 338–344, 1982.

[DS86a] W. Dally and C. Seitz. Deadlock-free routing in multiprocessor interconnection network. 5206:TR:86, Computer Science Department, California Institute of Technology, 1986.

[DS86b] W. J. Dally and C. L. Seitz. The Torus Routing Chip. *Distributed Computing*, (1):187–196, 1986.

[FCS90] M.L. Fulgham, R. Cypher, and J.L.C. Sanz. A comparison of SIMD hypercube routing strategies. RJ 7722 (71587), IBM Almaden Research Center, 1990.

[Gel81] D. Gelernter. A DAG-based algorithm for prevention of store-and-forward deadlock in packet networks. *IEEE Transactions on Computers*, c-30:709–715, October 1981.

[GPS91] L. Gravano, G.D. Pifarré, and J.L.C. Sanz. Adaptive Worm-hole Routing in Tori and Hypercubes. TR:91-10, IBM Argentina - CRAAG, March 1991.

[Gun81] K.D. Gunther. Prevention of deadlocks in packet-switched data transport system. *IEEE Transactions on Communications*, com-29(4), April 1981.

[Hil85] D. Hillis. *The Connection Machine*. The MIT Press, 1985.

[KK79] P. Kermani and L. Kleinrock. Virtual Cut-Through: A new computer communication switching technique. *Computer Networks*, (3):267–286, 1979.

[Kon90] S. Konstantinidou. Adaptive, minimal routing in hypercube. In *6th. MIT Conference on Advanced Research in VLSI*, pages 139–153, 1990.

[KS90] S. Konstantinidou and L. Snyder. The Chaos router: A practical application of randomization in network routing. In *2nd. Annual ACM SPAA*, pages 21–30, 1990.

[Lei90] T. Leighton. Average Case Analysis of Greedy Routing Algorithms on Arrays. In *SPAA*, 1990.

[LH91] D.H. Linder and J.C. Harden. An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes. *IEEE Transactions on Computers*, 40(1):2–12, January 1991.

[LM89] T. Leighton and B. Maggs. Expanders might be practical: Fast algorithms for routing around faults on multibutterflies. In IEEE, editor, 30^{th} *Annual Symposium on Foundations of Computer Science*, pages 384–389, October 1989.

[LMR88] T. Leighton, B. Maggs, and S. Rao. Universal packet routing algorithms. 1988.

[MS80] P.M. Merlin and P.J. Schweitzer. Deadlock avoidance in store-and-forward networks. 1: Store-and-forward deadlock. *IEEE Transactions on Communications*, 28(3), March 1980.

[Ni90] L.M. Ni. Communication Issues in Multicomputers. In *Proceedings of the First Workshop on Parallel Processing, Taiwan*, 1990.

[Ni91] L.M. Ni, February 1991. Personal Communication.

[NS] J.Y. Ngai and C.L. Seitz. A framework for adaptive routing. 5246:TR:87, Computer Science Department, California Institute of Technology.

[PFGS91] G.D. Pifarré, S.A. Felperin, L. Gravano, and J.L.C. Sanz. New techniques for combination, adaptivity, deadlock-freedom and synchronization in massively parallel routing. In Preparation, 1991.

[PGFS91] G.D. Pifarré, L. Gravano, S.A. Felperin, and J.L.C. Sanz. Fully-Adaptive Minimal Deadlock-Free Packet Routing in Hypercubes, Meshes, and Other Networks. Technical report, IBM Almaden Research Center, 1991.

[Pip84] N. Pippenger. Parallel communication with limited buffers. In *Foundations of Computer Science*, pages 127 – 136, 1984.

Table 1: Random Routing, 1 packet.

n	N	L_{avg}	L_{max}
10	1024	10.96	19
11	2048	12.09	21
12	4096	13.08	25
13	8192	14.03	27
14	16384	15.04	29

Table 2: Complement, 1 packet.

n	N	L_{avg}	L_{max}
10	1024	21	21
11	2048	23	23
12	4096	25	25
13	8192	27	27
14	16384	29	29

[Ran85] A.G. Ranade. How to emulate shared memory. In *Foundations of Computer Science*, pages 185 – 194, 1985.

[RBJ88] A.G. Ranade, S.N. Bhat, and S.L. Johnson. The Fluent Abstract Machine. In J. Allen and F.T. Leighton, editors, *Fifth MIT conference on advanced research in VLSI*, pages 71 – 93. The MIT press, March 1988.

[Upf89] E. Upfal. An $\mathbf{O}(\log N)$ deterministic packet routing scheme. In 21^{st} *Annual ACM-SIGACT Symposium on Theory of Computing*, May 1989.

[Val82] L. G. Valiant. Optimality of a two-phase strategy for routing in interconnection networks. March 1982.

[Val88] L.G. Valiant. General purpose parallel architectures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. North-Holland, 1988.

Table 3: Transpose, 1 packet.

n	N	L_{avg}	L_{max}
10	1024	11.09	21
11	2048	11.09	21
12	4096	13.13	25
13	8192	13.13	25
14	16384	15.23	29

Table 4: Leveled Permutation, 1 packet.

n	N	L_{avg}	L_{max}
10	1024	10.10	21
11	2048	10.98	21
12	4096	12.06	25
13	8192	13.07	25
14	16384	14.03	29

Table 5: Random Routing, n packets.

n	N	L_{avg}	L_{max}
10	1024	11.33	22
11	2048	12.52	25
12	4096	13.76	27
13	8192	15.02	30
14	16394	16.54	32

Table 6: Complement, n packets.

n	N	L_{avg}	L_{max}
10	1024	21	21
11	2048	24.99	30
12	4096	28.61	35
13	8192	32.74	39
14	16384	36.23	44

Table 7: Transpose, n packets.

n	N	L_{avg}	L_{max}
10	1024	12.27	26
11	2048	12.40	32
12	4096	16.01	37
13	8192	16.22	36
14	16384	20.49	43

Table 8: Leveled Permutation, n packets.

n	N	L_{avg}	L_{max}
10	1024	10.78	23
11	2048	11.77	25
12	4096	13.17	28
13	8192	14.60	32
14	16384	16.03	37

Table 9: Random Routing, $\lambda = 1$.

n	N	L_{avg}	L_{max}	I_r (%)
10	1024	12.10	30	93
11	2048	13.47	35	89
12	4096	15.01	37	85
13	8192	16.58	44	81
14	16364	18.30	49	76

Table 10: Complement, $\lambda = 1$.

n	N	L_{avg}	L_{max}	I_r (%)
10	1024	33.32	52	55
11	2048	39.29	58	49
12	4096	45.60	68	45
13	8192	52.87	79	41
14	16384	60.70	90	38

Table 11: Transpose, $\lambda = 1$.

n	N	L_{avg}	L_{max}	I_r (%)
10	1024	14.67	36	83
11	2048	14.67	36	83
12	4096	15.78	49	73
13	8192	20.31	54	71
14	16384	27.33	66	61

Table 12: Leveled Permutation, $\lambda = 1$.

n	N	L_{avg}	L_{max}	I_r (%)
9	512	11.28	37	94
10	1024	12.47	43	91
11	2048	13.50	48	89
12	4096	15.17	56	84
13	8192	16.91	53	80
14	16384	18.46	57	75