RELIABLE LOW LATENCY I/O IN TORUS BASED INTERCONNECTION NETWORKS

A Thesis

by

BABATUNDE AZEEZ

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2005

Major Subject: Computer Engineering

RELIABLE LOW LATENCY I/O IN TORUS BASED INTERCONNECTION NETWORKS

A Thesis

by

BABATUNDE AZEEZ

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2005

Major Subject: Computer Engineering

ABSTRACT

Reliable Low Latency I/O in Torus Based

Interconnection Networks. (December 2005)

Babatunde Azeez, B.Sc., Obafemi Awolowo University, Nigeria

Chair of Advisory Committee: Dr. Eun Jung Kim

In today's high performance computing environment I/O remains the main bottleneck in achieving the optimal performance expected of the ever improving processor and memory technologies. Interconnection networks therefore combines processing units, system I/O and high speed switch network fabric into a new paradigm of I/O based network. It decouples the system into computational and I/O interconnections each allowing "any-to-any" communications among processors and I/O devices unlike the shared model in bus architecture. The computational interconnection, a network of processing units (*compute-nodes)*, is used for inter-processor communication in carrying out computation tasks, while the I/O interconnection manages the transfer of I/O requests between the compute-nodes and the I/O or storage media through some dedicated I/O processing units *(I /O-nodes).* Considering the special functions performed by the I/O nodes, their placement and reliability become important issues in improving the overall performance of the interconnection system.

This thesis focuses on design and topological placement of I/O-nodes in torus based interconnection networks, with the aim of reducing I/O communication latency between compute-nodes and I/O-nodes even in the presence of faulty I/O-nodes. We propose an efficient and scalable relaxed quasi-perfect placement scheme using Lee distance error correction code such that compute-nodes are at distance-*t* or at most distance-*t+1* from an I/O-node for a given *t*. This scheme provides a better and optimal alternative placement than quasi perfect placement when perfect placement cannot be found for a particular torus. Furthermore, in the occurrence of faulty I/O-nodes, the placement scheme is also used in determining other alternative I/O-nodes for rerouting I/O traffic from affected compute-nodes with minimal slowdown. In order to guarantee the quality of service required of inter-processor communication, a scheduling algorithm was developed at the router level to prioritize message forwarding according to inter-process and I/O messages with the former given higher priority.

Our simulation results show that relaxed quasi-perfect outperforms quasi-perfect and the conventional I/O placement (where I/O nodes are concentrated at the base of the torus interconnection) with little degradation in inter-process communication performance. Also the fault tolerant redirection scheme provides a minimal slowdown, especially when the number of faulty I/O nodes is less than half of the initial available I/O nodes.

DEDICATION

To Allah, my wife Shukrah and my son Jafar.

ACKNOWLEDGEMENTS

I wish to express by sincere gratitude to my advisor Dr. Eun Jung Kim for her guidance and mentorship. I am deeply indebted to her as her help, stimulating suggestions and encouragement helped me in all the time of research for and writing of this thesis. I am also grateful to my other committee members Dr A.L.N Reddy and Dr. Hank Walker for their invaluable advice and suggestions on this thesis work. My special thanks goes to all members of High Performance Computer Lab especially Hogil Kim for his enormous contributions to this thesis work. He has always been there for me during any brainstorming and idea formulation sessions. Special thanks to Dr. Bart Childs, Elena Rodriguez for helping out with graduate advising and various administrative issues and to my honorable mentor Dr. Oloso.

I shall for ever remain indebted to my parents and in-laws for supporting me financially and otherwise throughout my master program. I will like to thank my dear friends, Idris Bello, Musodiq Bello, Ananth Kini, Niyi Olajide, Salah Aly, Abdullahi Abdulrahman, Ahmad Aden, Deji Coker for their useful comments and moral supports in making this thesis a reality.

Finally, I thank Shukrah Adigun and Jafar Azeez for everything they are to me.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

It is a known fact in today's high performance computing environment that I/O remains the main bottleneck in achieving the optimal performance expected of the ever improving processor and memory technologies. Traditional I/O uses bus architecture that allows processing units to communicate with memory or I/O devices using low latency load/store semantic across a shared bandwidth. This shared bandwidth can only support limited devices over a short distance with poor scalability, poor availability and minimal fault tolerance.

Interconnection networks combines processing units, system I/O and high speed switch network fabric into a new paradigm of I/O based network. In such high performance computers, the system decouples into computational and I/O interconnections each allowing "any-to-any" communications among processors and I/O devices unlike the shared model in bus architecture. It permits a large array of devices to be connected in a manner that provides high scalability, topological flexibility, fault tolerance and high availability. Communications can be processor-to-processor or processor-to-I/O devices. It is characterized by topologies such as mesh, torus, tree and butterfly as found in systems such as massively parallel processors or parallel computers, and clusters.

1.1. Problem Statement

As we enter a new era of high performance computing, where system-on-chip technology is taking over interconnection design by integrating all server components on a single chip, the effect is a sporadic increase in the computing power with interconnection of ten's of thousands of compute nodes unlike before. Direct communication between compute nodes and I/O devices becomes impracticable with high peak-to-average bandwidth ratio of I/O. Thus, *I/O nodes* are usually embedded to act as a proxy for compute-nodes in communicating with the I/O subsystems for transfer of I/O messages. In many parallel and cluster computers such as IBM BlueGene [1, 22], Cray3T [21], MIT J-Machine [18], torus based interconnection network has become the widely accepted.

The journal model is *IEEE Transactions on Parallel and Distributed Systems.*

The design of the interconnection in compute and I/O nodes in such systems falls basically into two classes. I/O nodes are either placed within the computational torus network as found in [18,21] or interconnected as a separate I/O communication network [1,22]. While integrating the I/O nodes within the networks might affect the computational performance due to link sharing, creating separate I/O nodes interconnection will lead to performance degradation of I/O traffic due to longer path and single point of failure for the I/O nodes. However, careful placement of I/O nodes within a torus network can drastically improve I/O performance and reduce the effect on computation performance while providing cost-effective and energy efficient system [11, 14]. This challenge can otherwise be referred to as *I/O embedding or placement problem* [20].

Furthermore, as the interconnection size grows in computation and I/O communication, the system becomes more complex and the probability of node failure also becomes very high. Thus, to sustain its level of performance and reliability, it is important for the system to be able to withstand a substantial number of faults with less alteration of the computational and I/O communication design. Most researches have focused on providing fault tolerant computational system with little regards to I/O communication reliability [4,7,13]. The general approach involves either using redundant processing nodes to replace the faulty nodes or bypass faulty processing nodes without adding extra redundant processing nodes. Although I/O nodes ratio is small compared to the compute nodes but failure of any of them may affect the performance of the system as a whole since I/O messages needed for computation are either lost or become unavailable. To provide fault tolerant I/O nodes communication, addition of redundant I/O nodes may not be applicable as in the case of processing nodes since there is a constrain on the number of I/O nodes, making the bypass of faulty I/O node a more practical solution. Thus providing an efficient low latency I/O messages redirection scheme with little performance slowdown in lieu of faulty I/O nodes becomes a crucial challenge. This will be otherwise referred to as *I/O reliability problem*.

1.2. Objectives

In this thesis we investigate four intertwined issues for I/O nodes embedding and reliability in torus interconnection networks.

1. Analyze and show that though embedding I/O nodes within torus interconnection, may affect the computational performance due to link sharing between process and I/O traffic, the performance degradation is little compared to the improvement in I/O performance if the placement of I/O nodes is efficiently done.
2. Present relaxed quasi-perfect placement, an efficient and flexible scheme for I/O nodes placement in torus interconnection using Lee-error correction codes. Lee codes had been applied in perfect distance-*t* and quasi perfect distance-*t* for in certain torus interconnection [5,2,19]. The goals are to provide a flexible optimal alternative placement when perfect distance-*t* is not possible and when available I/O nodes are below the minimum bound required for perfect placement in any two or three dimensional torus network.
3. Implement priority scheduler at the router level. This is to reduce the performance degradation that may occur when sharing links between process-to-process and I/O communication. The scheduling algorithm uses the pipeline router and virtual channel concept to prioritize between process-to-process and process-to-I/O traffic, with the former given higher priority. Thus the highly latency sensitive process-to-process messages is guaranteed quality of service similar to having separate dedicated links.
4. Provide I/O reliability in terms of maintaining system I/O communication in event of I/O nodes failure. The scheme applied here further makes use of the unique properties of I/O placement using Lee-error correction codes in providing all affected compute-nodes with uniformly distributed minimal alternative paths to other healthy IO-nodes.

1.3. Related Work

The family of torus interconnection such as *k*-ary *n*-cube and mixed radix have been used for the design of several high performance especially the parallel computers. These include the MIT J-Machine [18], Cray T3-D/T3E/XT3 [21], and IBM Blue Gene [22]. In Cray systems, the compute-nodes and I/O nodes are interconnected using the 3-D-torus network. The network is used for both inter-processor communication and for process-to-I/O communication. An interconnection of I/O nodes is concentrated along one edge of the 3-D-torus. Any compute node can send message to any I/O nodes for connecting I/O nodes while all other planes are used for interconnecting the compute-nodes. Only the compute-nodes in the plane adjacent to the I/O nodes have links that are directly connected to the I/O nodes. All other compute-nodes take several links or hop counts to communicate with the I/O nodes.

MIT J-machine system, uses a 3-D mesh having unconnected links at the edges of the interconnection. The compute-nodes and I/O nodes are also embedded in the 3-D-mesh in a similar manner with Cray system. However, the I/O nodes are made up of I/O cards which use bus architecture for direct access to the I/O devices such as array of disk instead of I/O interconnection. Thus compute-nodes do not have uniform access to the IO-nodes and incur high latency due to absence of the wrap around link in torus.

IBM Bluegene [1, 22], a massive parallel system of 64K nodes based on a new architecture that exploits system-on-a-chip technology to deliver high processing power uses different interconnection for the compute-nodes and the I/O nodes. The compute-nodes are connected using a point-to-point 3-D torus network while global tree interconnection for collective operations, form a separate network that connects the compute nodes and the I/O-nodes. A group of process nodes called a p-set is logically assigned to an I/O-node for external I/O communication. Thus process-to-I/O communication traffic uses an entirely different interconnection. The I/O nodes use a Gigabit Ethernet network to communicate to the external storage system. A major design concern with Bluegene is the single centralized access to the storage system through the I/O nodes which can lead to performance degradation and single point of I/O failure.

A symmetric hypernet was proposed in [12], addressing the I/O performance in hypercube interconnection by creating separate links for data communication and I/O communication with the aim of preventing any conflict between I/O messages and process messages and any possible congestion at the link level. All compute-nodes in cube were connected to an I/O node in such a way that each compute-node is adjacent to the I/O node.

Reddy et al. [20], proposed architecture for embedding I/O in hypercube interconnection using Hamming error correcting codes to obtain a perfect adjacency placement of I/O nodes. A perfect adjacency will allow each compute node to be adjacent to an I/O node for efficient communication between the I/O nodes and compute-nodes. It was also shown that the links can be shared efficiently between the processor and I/O traffic with little performance degradation. However, hamming error correction code is a binary code that shares similar properties with hypercube interconnection rather than torus. The existence and construction of perfect *d*-dominating set for 2-dimensional torus graph using minimum number of vertex was described [17] . A dominating set of a graph is perfect if each vertex of the set is dominated by exactly one vertex in the set.

Several topological properties of torus network have been defined using Lee distance error correcting codes [6], making it a natural metric to use in resource placement and fault tolerance routing algorithm in such interconnection. A perfect distance-t placement of resources in torus network where a non resource node is at exactly distance-*t* to a resource node was described in [5] using Lee distance error code. Furthermore, they proved that perfect distance-t is not possible in all torus network especially 2D and 3D torus [2] and proposed a quasi-perfect distance-t placement where a non-resource node is at most a distance-*t+1* to a resource node. However, quasi-perfect cannot also be generalized as an alternative for all torus and it is non-optimal with number of resource node less than the minimum bound required due to some conditions in its definition. In this thesis, we propose relaxed quasi-perfect placement strategies by relaxing some conditions in formulating a generalize algorithm for I/O nodes placement in torus configuration using the known perfect placement configurations.

Fault tolerance embedding in torus based systems has been extensively researched [4, 13, 7] with main focus on compute-nodes and links failure. The approaches used in providing fault-tolerance generally fall into two categories. The first approach tries to maintain system capability by bypassing any faulty resources with a reasonable slowdown while the second approach uses redundant resource as a replacement for any failed resource. A solution to spare processor nodes placement problem for torus based network was presented in [4] using Lee distance error codes. Spare processors are placed such that they are at distance-t to the non-spare processor. It provided an optimal 1-hop spare processor node placement in multi-dimensional torus and *t*-hop placement for 2D torus. Also, a strongly fault-tolerant design for *k*-ary *n*-cube torus was described in [13] using $(k/j)^n$ spare nodes, where each spare processor node is connected to a regular processor node and each spare processor node are again interconnected in (k/j)-ary *n*-cube or an *n*-dimensional hypercube. Joshua et. al. [11] used the concept of creating subcube in *n*-dimensional hypercube interconnection to show that with certain number of faulty processor nodes, the system can still carryout its computation with constant slowdown without additional redundant nodes. In this thesis we consider the issue of maintain fault tolerance with respect to I/O nodes failure. Our approach maintains the system I/O communication capabilities by bypassing faulty I/O nodes with constant slowdown. We apply the unique properties of perfect and relaxed quasi perfect I/O placement in redirection and routing of messages evenly to other alternative I/O nodes thereby avoiding possible link congestion with minimal additional latencies.

1.4. Organization of Thesis

The rest of the thesis is organized as follows. In section 2, we provide a brief background on I/O interconnection design and I/O nodes placement. Topology properties of torus interconnection and Lee error correction codes and how Lee code can be applied to I/O nodes placement is also described in this section. We propose relaxed quasi-perfect I/O node placement as a flexible and efficient alternative scheme when no perfect placement can be found in certain torus in section 3. Furthermore, in this section, a priority scheduling schemes implemented at the router level is described to reduce the effect of sharing links between I/O and process traffic on computational performance. Section 4 introduces some inherent properties of perfect and relaxed quasi perfect placement that can be used in redirecting I/O traffic with minimal slowdown when some I/O nodes becomes faulty. In section 5, we described our simulation environment with analysis of performance results obtained. Finally, we present conclusion and recommended future work in section 6.

2. BACKGROUND

2.1. I/O Interconnection Architecture

I/O interconnection manages the communication between processing units and I/O or storage media. It is characterized by high ratio of peak-to-average bandwidth [9]. Thus high level parallelization has been exploited to increase the throughput of I/O traffic by concentrating I/O requests from compute nodes to some dedicated I/O nodes, each with an I/O processor and memory to form an internal I/O subsystem. Internal I/O subsystem can be viewed as a collection of I/O nodes each managing and providing I/O access to a storage media [11] while they connect to other compute nodes within the interconnection. They allow parallel data transfer between compute-nodes and I/O nodes and handle request with lower latency using low-latency message passing protocols. In other words the concept of internal I/O further breaks the I/O interconnection into two levels. The first level of I/O occur across the computational interconnection through dedicated I/O nodes while the second occur by taking the whole interconnection as a whole and connecting it to an external storage network [14,10] as shown in Figure 1.

Our focus is on the internal I/O interconnection rather than the external I/O or storage network. Internal I/O interconnection architectural design basically falls into two main disparate ends according to how they connect to the computational interconnection. They are the integrated and separate I/O interconnections.

2.1.1. Separate I/O Interconnection

In this I/O interconnection design, I/O network is separated completely from the computational network to avoid link sharing between I/O traffic and process traffic. The rationale is to prevent any performance degradation that may occur during execution of application due to overlapping of process and I/O traffic. It is believed in such situations inter I/O traffic may adversely affect the latency sensitive process traffic performance. Generally, each I/O node in the separate network is dedicated to a particular subset of compute nodes in the computational interconnection as found in systems such as NCUBE [11] and Bluegene [22].

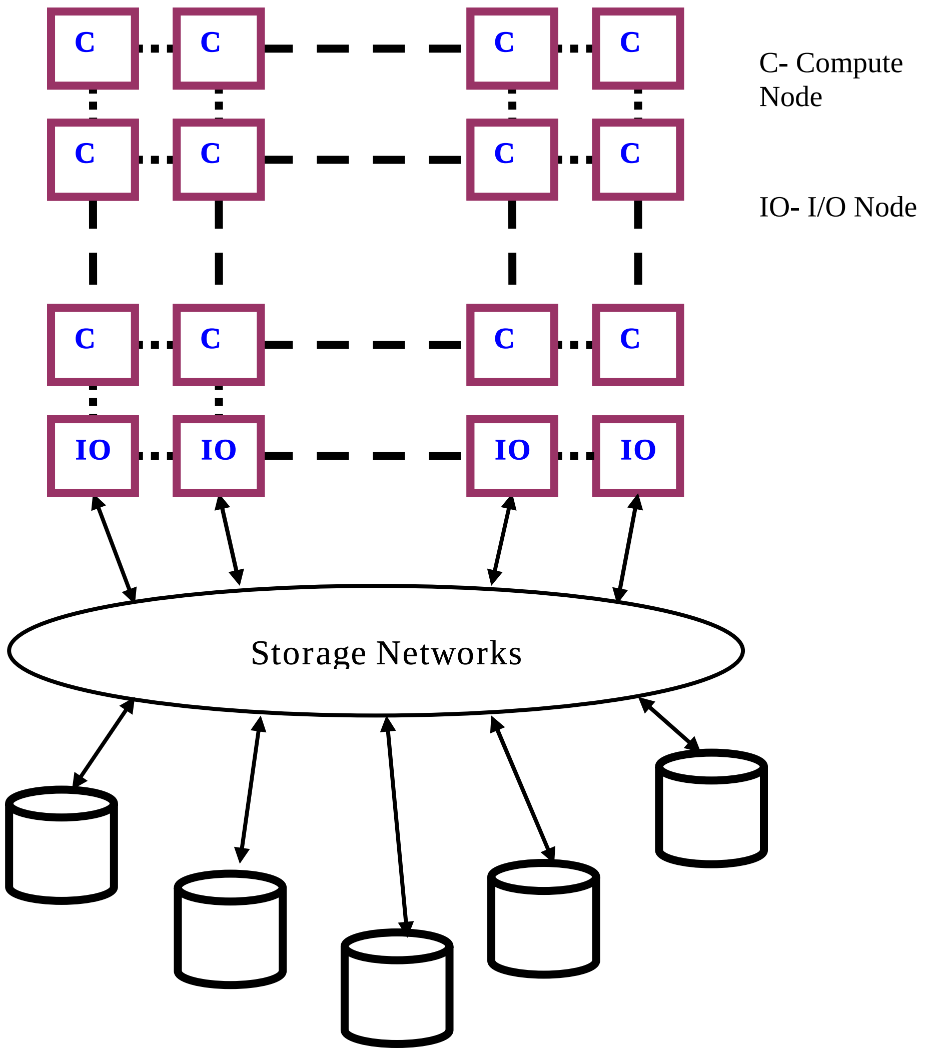


Figure 1. Integrated I/O interconnection with base I/O node placement.

Some major problems with this design include high cost/performance, power consumption due to additional hardware such as extra routers required for creating separate network. Also depending on the interconnection used for the I/O, the average hop count of compute nodes to the I/O node may increase affecting the overall I/O performance. Since each I/O node is dedicated to a subset of compute nodes, it becomes the limiting factor due to single point of failure.

2.1.2. Integrated I/O Interconnection

Here, I/O nodes are embedded within the computational interconnection networks, allowing the physical links to be shared between the process traffic and I/O traffic. A reason for this is that overlapping does not usually occur between the two traffics when running an application and in cases where they do, the performance degradation of process traffic is minimal [20]. Furthermore, integrating the two traffics in the same network reduces the complexity in design and management. Also it provides a relatively low cost/performance since it does not require additional interconnection hardware such has extra router for creating a new network. Fault tolerant schemes can be easily implemented since the failure of any I/O nodes only requires the compute node I/O traffic to be redirected to other I/O nodes all within the same network without substantial effect on the performance. Besides, as power dissipation in interconnection networks is becoming a big concern, creating another separate level of interconnection just for I/O traffic will increase the overall power dissipation in the system due to additional hardware components for separate network. However, one major concern for better I/O performance in this interconnection is the placement of I/O nodes among the compute nodes.

2.1.3. I/O Nodes Placement

Communication latency, bandwidth and contention often depend on the relative position of the end point communication in integrated I/O interconnection [14]. Therefore the relative position of I/O nodes in the interconnection can have a significant impact on the performance of the I/O subsystem. Generally in most commercially available systems, I/O nodes are usually placed or clustered at the base or end plane of the interconnection especially in torus based interconnection similar to one in Figure 1. We refer to this kind of placement the *base-I/O placement*. The problem with this arrangement is that all compute nodes do not have relatively short distance or hop count to the I/O nodes. Only those compute nodes adjacent to I/O nodes have shortest path with the other compute nodes having several hop counts to access I/O nodes depending on their location. Moreover, random movement of I/O traffic across the whole interconnection adds to the degradation of the process traffic since contention is likely to occur at every channel in the network.

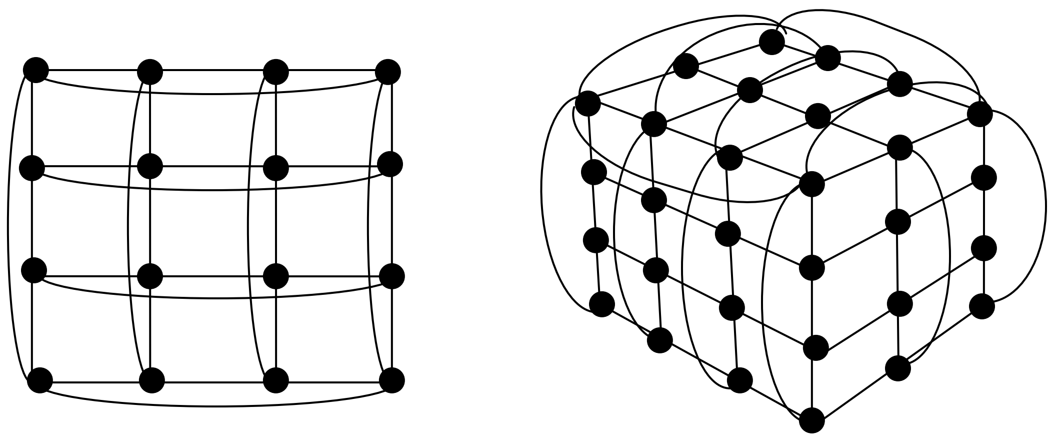
However, careful embedding of the I/O nodes within the computational interconnection can greatly reduce the I/O latency by allowing all compute nodes have almost the same shortest distance or hop count to I/O nodes. A direct implication is that many I/O traffic will be directed towards some channels associated with the I/O nodes while several other physical links not used for I/O traffic are made available to process traffic thereby reducing contention that may occur due to sharing links between the I/O and process traffic.

2.2. Torus Interconnection

Torus is a class of regular network that is strictly orthogonal as it can be arranged in such a way that it produces a displacement in a single dimension and every node has at least one link across each dimension. Torus network can be referred to as a *k*-ary *n*-cube Q_k^n or a mixed radix ($T_{k_1 k_2 \ldots k_n}$). A *k*-ary *n*-cube has equal number of nodes radix *k* in each *n* dimension as shown in Figure 2. In a mixed radix torus, at least one or more of the radix for *n* dimension are not the same. Figure 2(a) and 2(b) show a 4-ary 2-cube (Q_4^2) with equal radix on the x and y dimension and a mixed radix ($T_{4,4,3}$) in which the radix along the x and y dimension are of equal radix 4 but the z dimension as a different radix of 3.
A *2*-dimensional torus is a *k*-ary *2*-cube (Q_k^2) or a mixed radix ($T_{k_1 k_2}$) while 3D torus is a 3-dimensional ($T_{k_1 k_2 k_3}$) mixed radix *k* or a *k*-ary *3*-cube (Q_k^3). Here we assume $k \geq 3$. In terms of performance, the throughput of torus interconnect is usually limited by the bisection bandwidth B that cut the network into half, $B = \frac{4N}{k}$.

It has a small average minimum hop count between two nodes of the network given by

$$H_{av} = \begin{cases} \frac{nk}{4} & k - even \\ n\left(\frac{k}{4}\right) - \frac{1}{4} & k - odd \end{cases}$$



(a) 4-ary 2-cube. (b) 4X4X3 Mixed Torus.

Figure 2. Various torus interconnections.

A major limitation of torus interconnection is the high diameter. The diameter of a network is the maximum number of nodes that must be transversed to send a message to any node along a shortest path. For a 2-dimensional torus the diameter is $N^{\frac{1}{2}}$ where N is the total number of nodes in the torus while a 3-dimensional torus has it as $\frac{nk}{2}$. However, under realistic packaging, it has been shown that low dimensional torus such as 3-dimensional torus outperforms other types of interconnection such as hypercube [9], with better cost/performance trade off and a more scalable structure. Thus two and three dimensional torus has been the de facto in today's high performance computing as employed in [22, 21,18].

2.3. Lee Distance Error Correction Codes

Error detection and correction codes are used mainly in preserving information bits sent over a communication channel. Information bits are encoded using some metrics for adding redundant bits to the information to form a code or *codewords* with capability of detecting or correcting *t*-errors at the decoding end of the information transmission. A major metric used in such encoding and decoding is the Hamming metric especially for binary finite set. Lee-metric [16], on the other hand is more applicable when the set of information digits are non-binary finite set.

2.3.1. Lee Metric

Lee Metric [6, 16] defines two set of metric that can is applicable for finite fields >2, the Lee distance and Lee weight.

Lee Weight

Consider a vector space over V_k^n, where $\hat{V}$ a subspace in V_k^n is represented as $\hat{V} = v_{n-1}v_{n-2}...v_0$ for $0 < i < n-1$.

The Lee weight of an integer v_i

$$W_{lee}(v_i) = \min(v_i, k - v_i)$$

Then Lee weight of the vector space is

$$W_{lee}(\hat{V}) = \sum_{i=0}^{n-1} W_{lee}(v_i)$$

$$= \sum_{i=0}^{n-1} \min(v_i, k - v_i) \quad . \tag{2.1}$$

Lee Distance

Just as in Hamming Distance, Lee distance between two codes or vector points in a vector space over V_k^n, is the Lee weight of their digit-wise difference *modulo k* of a particular finite field set *k.*

Let $\hat{V}$ and $\hat{U}$ be two vector points over the vector space V_k^n

Where $\hat{V} = (v_{n-1}v_{n-2}...v_0)$ and $\hat{U} = (u_{n-1}u_{n-2}...u_0)$ $\ni$ $v_i, u_i \in$ $\{0,1,2,...,k-1\}$.

Then the Lee distance between $\hat{V}$ and $\hat{U}$ is

$$D_{lee}(\hat{V}, \hat{U}) = W_{lee}(\hat{V}, \hat{U})$$

$$= \sum_{i=0}^{n-1} \min(v_i - ui, ui - vi) \bmod k$$

$$= \sum_{i=0}^{n-1} \min((v_i - u_i)(\bmod k), (u_i - v_i)(\bmod k)) . \tag{2.2}$$

2.3.2. Lee Metric and Torus Interconnection

A *k*-ary *n*-cube, (Q_k^n), is defined as an *n*-dimensional vector space with equal radix of length k. It can be constructed as a cross product of *k-ary* cycle Q_k such that,

$$Q_k^n = \underbrace{Q_k \otimes Q_k \otimes ... \otimes Q_k}_{n} . \tag{2.3}$$

Each node $\hat{X}$ in the vector space Q_k^n can be represented as an *n-digit* radix k vector with,

$$\hat{X} = (x_{n-1}x_{n-2},...,x_0) \;\; \forall \quad x_i \in \{0,1,2,...,k-1\}$$

and the total number of nodes is K^n.

A mixed radix torus $T_{k_1k_2...k_n}$ can also be defined in vector notation as an *n*-dimensional vector space with mixed radix k_i for each dimension where $0 \le i \le n-1$ and all k_i are not of the same length. The construction is also similar to but with different values of k_i such that,

$$T_{k_1,k_2,...,k_n} = Q_{k_1} \otimes Q_{k_n} \otimes ... \otimes Q_{k_n} . \tag{2.4}$$

The vector space contains $k_1 \times k_2 \times ... \times k_n$ nodes. Each node is represented as an *n-digit mixed* radix k_i with address

$$\hat{X} = (x_{n-1}x_{n-2},...,x_0) \;\; \forall \;\; x_i \in \{0,1,2,...,k_i - 1\} .$$

Furthermore, two nodes in *n*-dimensional torus Q_k^n or $T_{k_1k_2...k_n}$ are directly connected or have an edge between them if their addresses differ by $\pm 1(\bmod k)$ or $\pm 1(\bmod k_i)$ respectively in exactly one digit. Using Lee metric, we can define an edge between two nodes $\hat{X}$ and $\hat{Y}$ in a Q_k^n torus interconnection as,

$$D_{lee}(\hat{X},\hat{Y}) = \sum_{i=0}^{n-1} \min((x_i - y_i)(\bmod k),(x_i - y_i)(\bmod k)) = \pm 1(\bmod k) = 1$$

and for $T_{k_1k_2...k_n}$ torus as

$$D_{lee}(\hat{X},\hat{Y}) = \sum_{i=0}^{n-1} \min((x_i - y_i)(\bmod k_i),(x_i - y_i)(\bmod k_i)) = \pm 1(\bmod k_i) = 1.$$

Thus $n-dimensional$ torus each node shares an edge with two other nodes on each dimensional axis for a total of $2n$ node degree incidental on any node.

2.3.3. Lee Distance Codes Generation

The linear code using Lee metric can be defined as a 3-tuple $(n,k,d_{\min})_s$

where,

n = length of the codewords

k = number of information digits

$d_{\min}$ = minimum Lee distance between codewords

s = finite field set $\mathrm{F_s}$ where $s > 2$

$n-k =$ length of the check digit

With this, a set of codewords C can be generated each of length n where all codewords are closed under addition and multiplication operation such that for any $c_i, c_j, c_k \in C$,

$$c_k = c_i + c_j \text{ and } -c_i, -c_j, -c_k \in C$$

Also, the minimum Lee distance among the codewords for detecting and correcting *t*-errors is

$$D_{lee}(c_i, c_j) = \textit{smallest } W_{lee}(c) \text{ of any non-zero codewords}$$

$$= d_{\min} = 2t+1$$

Linear block code can be constructed using a $k \times n$ generator matrix G as the basis for generating the codewords such that there are *k* rows of information digit and *n* column of codewords with

$$c = \hat{k} \cdot G$$

In order to correct any information digit with *t*-errors, an $(n-k)\times n$ parity check matrix H is used such that

$$c \cdot H^T = 0 \quad \text{iff } c \text{ is a codeword}$$

since

$$H = \begin{pmatrix} P & I_{n-k} \end{pmatrix} \text{ and } G = \begin{pmatrix} I_k & -P^T \end{pmatrix}$$

where *P is* $(n-k) \times k$ matrix and *I* an identity matrix

However, if a transmitted code $c \cdot H^T \neq 0$ then the code is in error and can be corrected by selecting a codeword at Lee distance-*t* to it.

2.3.4. Lee Codes and I/O Node Placement in Torus

An important application of Lee code is that a set of codewords of length *n* represented in radix form can be generated over a linear vector space Q_k^n such that the minimum distance between them is *d* and the distance between the non codewords and codeword is the error capability *t* of the generated code. In other word, by representing a torus interconnection network in a vector space and generating set of codewords in such vector space with error capability *t*, we can make the codewords the location of I/O nodes and the non-codewords the location of compute nodes so that any compute node is at least a distance-*t* to an I/O node. It is worth noting that though we have restricted ourselves to I/O, the concept is generally applicable to any other resource node in a torus network.

With this concept, several efficient I/O nodes placement can be developed in a torus network. I/O nodes placement can be a perfect distance-*t*, quasi-perfect distance-*t*, *j*-adjacency or *j*-adjacency-distance-*t* as defined below.

Definition 1. A placement is said to be perfect distance-t when each compute node is *exactly* at a distance-t to an I/O node and no two I/O nodes are adjacent to each other.

Definition 2. A placement is quasi-perfect (QP) distance-t if a compute node is at exactly distance-t or at most distance t+1 from an I/O node [3,12].

Definition 3. In a *j*-adjacency placement, a compute node is adjacent to *j*-I/O nodes. It is also said to be perfect if each compute node is adjacent to *j*-I/O nodes with no I/O nodes adjacent to each other.

Definition 4. A placement can also be constructed by a combination of *j*-adjacency and distance-t so that a computes node is at a distance-*t* and adjacent to *j*-I/O nodes.

Even though these placement are efficient, it as be shown [5, 2] that they are only possible in certain torus configuration and thus can not be generalized for all types of *n*-

dimensional torus. Table 1 gives a summary of some known perfect placement for I/O node placement. Note that these are the tiling block size and multiple blocks can be used to build perfect distance-*t* placement in larger torus.

TABLE 1
Summary of known Perfect distance-t Placement.

Dimension (n)	Distance (t = 1)	Distance (t = 2)	Distance (t = 3)	Distance (t = 4)	Distance (t)
1	T_3	T_5	T_7	T_9	T_{2t+1}
2	$T_{5,5}$	$T_{13,13}$	$T_{25,25}$	$T_{41,41}$	T_{2t^2+2t+1}
3	$T_{7,7,7}$	-	-	-	-

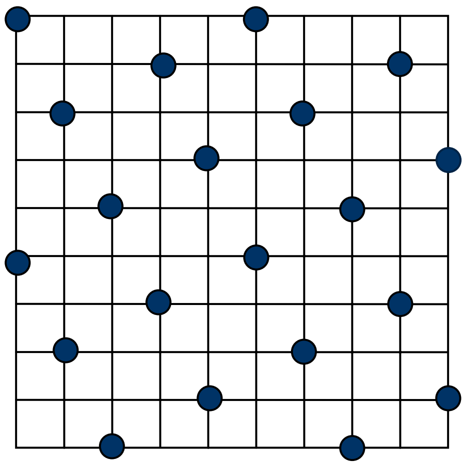
3. RELAXED QUASI-PERFECT (RQP) I/O PLACEMENT IN TORUS

Perfect distance placement is by far the most efficient I/O placement strategy for distributing I/O nodes among compute nodes, since each compute node is guaranteed uniform and constant hop-count to I/O nodes for sending their I/O traffic in torus based interconnection networks. However, depending on the value of *t*, such perfect placements are only possible in certain torus network. For example there exists a perfect distance-1 in Q_{10}^2 but not in Q_8^2. Quasi-perfect on the other hand suffer two main limitations

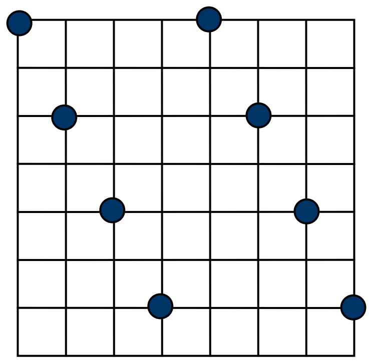
1. It can not be generalized for all torus interconnection without perfect distance placement.
2. It is non optimal due to lower number of I/O nodes than in perfect placement.

Consider for example a perfect distance-1 placement in Q_{10}^2 (100 nodes) requiring a lower bound of (20 I/O nodes) using equation 3.1, and a quasi-perfect distance-1 placement in Q_8^2 (64 nodes) with lower bound of (8 I/O nodes) as shown in Figure 3. Clearly, the placement in Q_8^2 is not optimal because if a perfect distance-1 was to exist for Q_8^2 then its minimal bound should have been, 13 I/O nodes. The reason for this is that a necessary condition for quasi-perfect requires no compute node to be at a distance-*t* or less to more than one I/O node. In essence, if a and b are two I/O nodes, and S_a and S_b are the set of compute nodes at distance-*t* or less to *a* and *b* respectively, then $S_a \cap S_b = \phi$. So even though it might be an alternative, because of this constrain, the number of I/O nodes are reduced below the minimum bound.

Therefore, for practical implementation of these theories in designing a low latency I/O with minimal hop-count in torus based networks, where other factors determines the size of the network to be used rather an I/O placement strategy, it becomes imperative to be relaxed about some of the conditions without loss of generality.



(a) Perfect distance-1 in Q_{10}^{2}.



(b) Quasi Perfect distance-1 in Q_{8}^{2}.

Figure 3: Perfect and quasi-perfect placements.

In this section, we present a relaxed-quasi perfect (RQP) distance-*t* placement as an alternative where there exist no perfect distance placements in Q_k^n or $T_{k_1k_2\ldots k_n}$ and applied it to situations when distance-*t* placement is required but not available. We also considered how to efficiently place I/O nodes when the number of available I/O nodes is below the minimum bound for perfect placement. Finally, we present a priority scheduling algorithm to be implemented at the router level to reduce the effect of sharing links between I/O and processor traffic in the event of an overlap.

3.1. Relaxed Quasi-Perfect (RQP)

Definition 5.

Relaxed quasi-perfect (RQP) distance-*t* can be defined as an I/O placement strategy with the maximum numbers of compute nodes at a distance-*t* to *one or more I/O nodes* while the remaining compute nodes are at most distance-*t+1* to some other I/O nodes. In other word let *a* and *b* be two I/O nodes, and S_a and S_b be the set of computes nodes at distance-*t* or less from each I/O node but not more than distance *t+1* to *a* and *b*, then $S_a \cap S_b \neq \phi$, with some compute nodes at distance-t or less to both *a* and *b*.

The idea is to build RQP for any given *k*-size torus from already known perfect-distance-*t* placement in an *n*-dimensional torus. Thus a necessary condition for the existence of RQP is that there must exist at least a perfect distance-*t* for any integer value of *t* in an n-dimensional torus. A RQP can then be constructed for a torus T_1 by building a perfect distance-*t* placement in a torus T_2 larger than T_1 but with the same *n*-dimension and by eliminating appropriate radix positions in each dimension of T_2 the required torus T_1 is formed. The RQP placement in T_1 will then have a maximum number of compute nodes at distance-*t* or less to I/O nodes and the remaining computes nodes can be assigned to any of the I/O nodes with distance-*t+1*. The advantages of this approach are

1) The number of I/O nodes required is either equal to the minimum bound M for such perfect distance-*t*, to be shown by Theorem 1.
2) Since some compute nodes are at distance-*t* or less to more than one I/O node, it will be very useful for fault tolerant I/O nodes placement and when there are limited number of I/O nodes.

In the following subsections we give an analytical model for constructing RQP placement for a *k*-ary *n*-cube and a mixed radix torus interconnection. The analysis is with respect to 2 and 3-dimensional torus though we generalized it into an *n*-dimensional some cases where there exist perfect placements.

3.1.1. Perfect Distance-t K-ary n-cube and Mixed Radix-k Torus Network

Here we first give a general analysis for construction of a perfect distance-t, a necessary condition for RQP in *k*-ary *n*-cube and mixed radix-*k* torus.

Let Q_k^n be an *n*-dimensional vector space with each node address represented as $\hat{X} = x_{n-1}x_{n-2}...x_0$ where $0 \leq x_i \leq k-1$.

Also the minimum bound *M* necessary for distance-t placement [6] be

$$M \geq \frac{N}{p} \qquad (3.1)$$

where $p = \left(1 + \sum_{i=1}^{\min(t,n)} 2^i \binom{n}{i}\binom{t}{i}\right)$ (3.2)

and $N = K^n$ or $k_1 \times k_2 \times ... \times k_n$

The value p is otherwise known as the volume of a *packing sphere* with radius t. A radius-t packing sphere is the set of compute node within a distance t or less from an I/O node. For a *2*-dimensional torus, equation (3.2) is reducible to $p = 2t^2 + 2t + 1$ [5] and for *3*-dimensional torus, $p = \frac{(2t)(2t+1)(t+1)}{3} + 2t + 1$ [2].

K-ary n-cube

In a k-ary n-cube Q_k^n there exists a perfect distance-*t* placement if *k* is divisible by *p*. Furthermore, if *k*=*p*, then Q_k^n is the tiling or smallest size for which a perfect distance-*t* can be generated and any other $Q_{k_2}^n$ for which $k_2 > k$ but divisible by *k* or *p* can also be formed by replicating the Q_k^n, $\frac{k_2}{k}$ times along each dimension $1 \le i \le n$ to form a perfect distance-*t* placement from the definition of Q_k^n in equation 2.1.

We can generate the I/O node locations for perfect distance-*t* in Q_k^n by using a check matrix *H* (for 2-dimensional torus, $H = \begin{pmatrix}2t^2 & t\end{pmatrix}$ [5]) such that set *C* of all nodes $\hat{X}$ for which;

$$\hat{X} \cdot H^T \equiv 0 \bmod k$$

are considered I/O nodes. Also for $Q_{k_2}^n$, the set C_2 of all nodes $\hat{X}$ with $\hat{X} \cdot H^T \equiv 0 \bmod k_1$, forms the I/O node placement where $\hat{X}$ are node orthogonal to the placements in Q_k^n.

Mixed radix $T_{k_1k_2...k_n}$ Torus

It is clear from section 2 that a mixed radix $T_{k_1k_2...k_n}$ can be formed by cross product of all Q_{k_i} where ($1 \le i \le n$). Therefore, there exist perfect distance-*t* placements also in a mixed radix torus if each k_i is divisible by *p*. Assuming *k*=*p* for a particular perfect distance-*t*, then we can obtain a perfect distance-t placement in $T_{k_1k_2...k_n}$, by constructing a perfect distance-*t* in Q_k^n and replicating the placement $\frac{k_i}{k}$ times along each dimension *I* of the torus network. The result is a set of I/O nodes location *C₂* for all $\hat{X}$ nodes orthogonal to the placement in Q_k^n that is $\hat{X} \cdot H^T \equiv 0 \bmod k_i$.

3.1.2. Relaxed Quasi-Perfect Distance-t Placement in K-ary n-cube

Now we consider a torus $Q_{k_1}^n$ where k_1 is not divisible by *p*. Also let $Q_{k_2}^n$ be another k_2-ary *n*-cube torus where

$$k_2 = \left(\left\lceil \frac{k_1}{p} \right\rceil \times p \right) \text{is greater than } k_1.$$

We show how a relaxed quasi perfect placement can be formed from the perfect distance-t placement in $Q_{k_2}^n$.

1) First we construct a perfect-distance-*t* in $Q_{k_2}^n$ such that k_2 is divisible by *p* as described in the previous subsection. C_2 is the set of all I/O nodes $\hat{x}$ for which $\hat{X} \cdot H^T \equiv 0 (\bmod k_2)$, where H is a check matrix.
2) By eliminating each $i^{th} - radix$ for all $i \geq k_1$ along each dimension of $Q_{k_2}^n$, a $Q_{k_1}^n$ relaxed quasi perfect distance-*t* is formed with a set of I/O nodes C_1 such that for each $\hat{X} = (x_{n-1}x_{n-2}...x_0) \in C_1$, there is no x_i greater than k_1.
3) In set C_1 of I/O nodes, a maximum number of compute nodes are at distance-*t* from some I/O nodes.
4) The remaining compute-nodes are then reassigned to some I/O nodes in set C_1 with distance-*t+1*.

Some of the properties of RQP are summarized in the following theorems and corollary.

Theorem 1: *In a Relaxed quasi-perfect placement the minimum bound for the number of I/O nodes is M where M is the minimum the minimum bound in a perfect distance-t.*

Proof.

This is obvious from equation 3.1 and 3.2.

Theorem 2: *In a relaxed quasi-perfect distance-t placement, there exist some I/O nodes with distance r<d between them where, d= 2t +1 is the minimum distance for perfect distance-t placement.*

Proof.

Consider a relaxed quasi-perfect distance-1 placement in a torus-A network. This is constructed by first building a perfect distance-1 torus network larger torus-T and eliminating some radix positions greater than the radix of torus-T along each dimension. This means that some I/O nodes in the placement will have some of their corresponding compute nodes in the larger torus network eliminated from torus-T network. Similarly, some compute nodes in the torus-T will have their I/O nodes with distance-*t* removed. This is especially true for all I/O and compute nodes at the edges of torus-T. Due to the wrap-around edges, the minimum distance between some two I/O nodes will be reduced by 1. □

Corollary. *In a relaxed quasi perfect distance-t placement, there exist some compute nodes at a distance t to more than one I/O node.*

Proof.

This is implied from Theorem 2 since the minimum distance between some I/O nodes will become less than *d*. □

3.1.3. RQP in Mixed Radix Torus $T_{k_1 k_2 \ldots k_n}$

As in the case of a perfect distance-t placement in $T_{k_1 k_2 \ldots k_n}$, there exists no perfect distance-t in $T_{k_1 k_2 \ldots k_n}$ when at least one of radix $k_1, k_2, \ldots, k_n$ is not divisible by *p*. We can construct a RQP in $T_{k_1 k_2 \ldots k_n}$ by forming a perfect distance-*t* in torus $T_{q_1 q_2 \ldots q_n}$

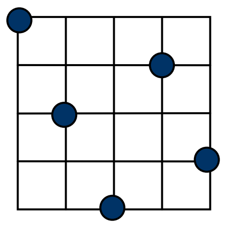
where $q_i = \left(\left\lceil \frac{k_i}{k} \right\rceil \times k \right)$ for $1 \le i \le n$.

and C_q the set of I/O nodes placement. Following the same procedure as *k*-ary *n*-cube, quasi perfect distance-t I/O placement is generated by forming a set C_j with all

$\hat{X} = (x_{n-1}x_{n-2}...x_0) \in C_q$ having $x_i \geq k_i$ along dimension-i in $T_{k_1k_2...k_n}$ are eliminated. All other analysis remains the same as *k*-ary *n*-cube.

3.1.4. Analysis with RQP Distance-1 in 8 ary -2 cube (2D)

In a *2*-dimensional torus network, it as been shown that the only existing perfect distance-1 placement are in a configuration with all radix divisible by $k = p = 5$ [5] with Q_5^2 (5-ary2-cube), being the smallest from which all other perfect-one placement can be constructed. Therefore there exist no perfect-one placement in Q_8^2 (8-ary 2-cube), since 8 is not divisible by 5. We therefore can construct an optimal relaxed perfect-distance-1 I/O nodes placement as described earlier. Using Lee error correction scheme, a perfect distance-1 is generated over Q_5^2 with I/O node locations orthogonal to H^T where $H = (1 \quad 3)$. The I/O nodes locations are the set C = {21, 00, 42, 31, 12} as shown in Figure 4-(a). The blue circle represents the I/O nodes.


(a) Perfect distance-1 in Q_5^2.

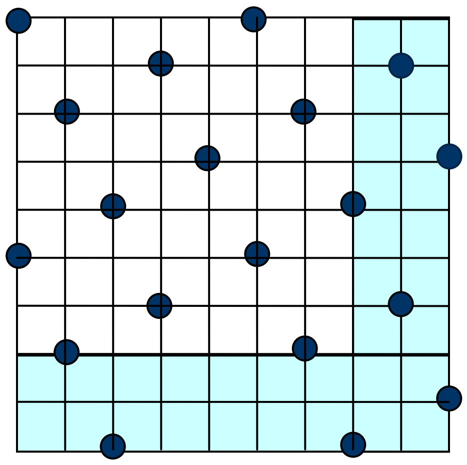

(b) Elimination some rows and column in Perfect distance-1 Q_{10}^2.

Figure 4. Perfect distance-1 construction in Q_5^2 and Q_{10}^2.

Furthermore since $k_1 = 8 > 5$, perfect distance-1 placement for a larger Q_{10}^2 (10-ary 2-cube) by replicating the I/O node locations in Q_5^2 twice on the x and y dimension as shown in Figure 4-(b). By eliminating the 8th and 9th radix in Q_{10}^2 , Q_8^2 with a relaxed quasi-perfect distance-1 is formed all I/O nodes address (xy) having either x or y ≤ 8 as shown in Figures 4-(b) and 5. The remaining compute nodes in square are distance-2 to some other I/O nodes. The lower bound for the required number of I/O nodes is $\left\lceil \frac{8^2}{5} \right\rceil = 13$.

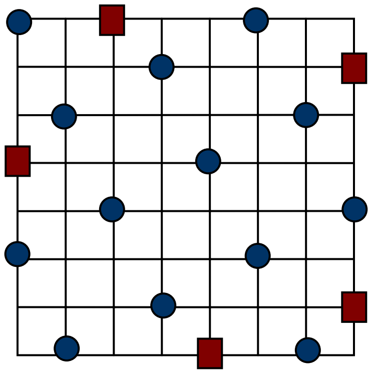


Figure 5: Relaxed Quasi Perfect distance-1 in Q_8^2.

3.1.5. Analysis with RQP Distance-1 in Q_4^3 (4-ary 3-cube)

In a 3-dimensional torus network, the only existing perfect-1 placement are configurations with all radix divisible $p = 7$ with Q_7^3 (7-ary 3-cube), being the tiling size from which all other perfect-1 placement can be constructed. In fact it has been shown that no other perfect-distance-*t* placements with $t \geq 2$ for any 3-dimensional torus [2]. Thus, we cannot find a perfect distance-1 placement for Q_4^3. In order to obtain a relaxed quasi-perfect distance-1 placement therefore, we need only obtain the perfect distance-*1* I/O location for Q_7^3 since it is a larger network than Q_4^3. The perfect distance-*1* placement is generated using a check matrix $H = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$ [2] with a set of 49 I/O locations as shown in Figure 6.

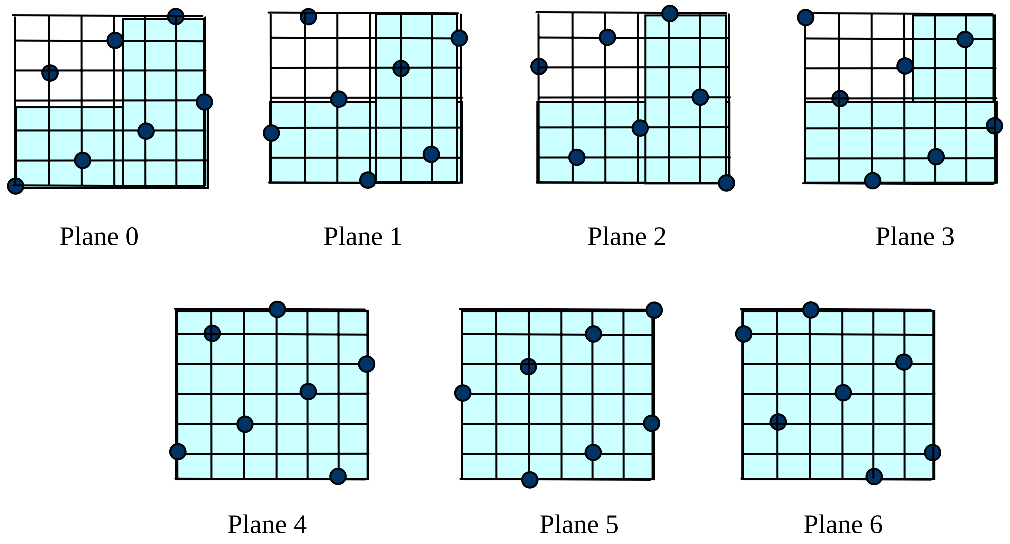


Figure 6: Perfect distance-1 in Q_7^3.

A relaxed quasi-perfect distance-1 placement for Q_4^3 is then constructed by eliminating the all planes ≥ 4 in each dimension. In each remaining planes, the corresponding radix positions ≥ 4 is also eliminated. The resulting RQP placement is as shown in Figure 7, with the remaining computes nodes in rectangular boxes at distance-2 to some other I/O nodes.

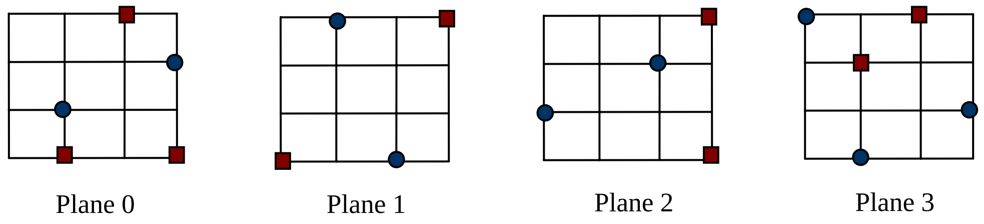


Figure 7. Relaxed quasi perfect distance-1 in Q_4^3.

It is worth noting from the definition of RQP that no two I/O nodes are to be adjacent to each other. This enables a large number of compute nodes to be at distance-*t* to I/O nodes. However, this is not the case for torus $Q_{k_1}^n$ configuration such that $(k_1 \bmod k = 1)$. The last radix will always have the same I/O position as the first radix and due to the wrap around, the two I/O nodes will be adjacent to each other. In order to conform to our definition, RQP can be formed for such $Q_{k_1}^n$ by using $Q_{k_3}^n$ such that $\left(k_3 = \frac{k_1}{2}\right)$. The RQP constructed for $Q_{k_3}^n$ can be replicated twice along each dimension to form $Q_{k_1}^n$. For example to form a relaxed quasi-perfect distance-1 I/O placement in Q_6^2, RQP is first developed for Q_3^2 and replicated twice to form Q_6^2 on each dimension as shown in Figure 8. Note that this method is applicable to other torus configuration without loss of generality.

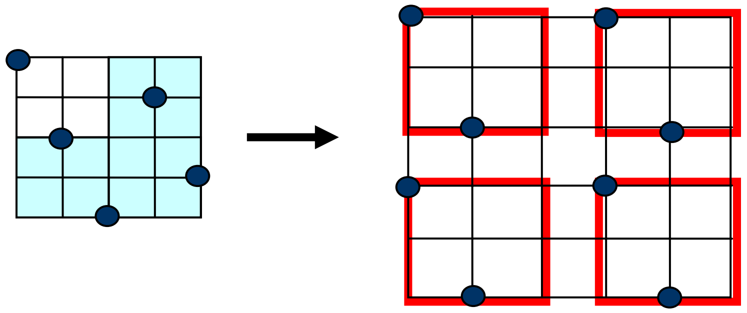


Figure 8. Relaxed quasi perfect distance-1 in Q_6^2.

Summary of Algorithm for RQP Distance-t Placement in *k*-ary *n*-cube Q^n_k

Let $\hat{X} = x_{n-1}x_{n-2}...x_0$ represents any node in vector notation over Q_k^n torus, where $0 \le x_i \le k-1$

p = volume of packing sphere for perfect distance-*t* placement defined in equation 3.2

C_1 = $\{ \hat{Y} | \hat{Y}$ is an I/O placement for RQP in $Q_{k_1}^n \}$, where k_1 is not divisible by p.

C_2 = $\{ \hat{Z} | \hat{Z}$ is an I/O placement for perfect distance-1 in $Q_{k_2}^n \}$, where k_2 is divisible by p.

H = a check matrix for perfect distance-t placement.

D_{lee} = Lee distance between two nodes

Input: (k, n, t)

```
1. Construct a perfect distance-t placement for $Q_{k_2}^n$ such that $k_2 = \left\lceil \frac{k_1}{p} \right\rceil \times p$
        ***if*** $(\hat{X} \cdot H^T) \equiv 0 (\bmod p)$ ***then*** $\hat{X} \in C_2$
2. Determine $\hat{Y} \in C_1$
        ***for*** each $\hat{Z} \in C_2$
          ***for*** i = 0 to n-1 do
            ***if*** $z_i \ge k_1$ ***then*** $\hat{Z} \notin C_1$ and ***break***
            ***else*** continue;
          ***end for***
          add $\hat{Z}$ to $C_1$
        ***end for***
3. Assign compute-nodes to I/O node
        ***for*** each $\hat{Y} \in C_1$
            ***if*** $D_{lee}(\hat{X}, \hat{Y}) \le t$ ***then*** mark and assign compute node $\hat{X}$ to $\hat{Y}$
        ***end for***
        ***for*** each unmarked compute node $\hat{X}$
              determine $\hat{Y} \in C_1$ such that $D_{lee}(\hat{X}, \hat{Y}) \le t+1$
              ***if*** number of $\hat{Y}$ is greater than 1
                ***then*** assign $\hat{X}$ to $\hat{Y}$ with the minimum number of compute node assignments
              ***else*** assign $\hat{X}$ to $\hat{Y}$
          ***end for***

**4.** Return Placement
```

Summary of Algorithm for RQP Distance-t I/O Placement in Torus T$_{\mathbf{k1k2 \cdots kn}}$

Let $\hat{X} = x_{n-1}x_{n-2}...x_0$ represents any node in vector notation over Q_k^n torus, where $0 \le x_i \le k-1$

p = volume of packing sphere for perfect distance-*t* placement defined in equation 3.2.

C_1 = { $\hat{Y} | \hat{Y}$ is an I/O placement for RQP in $T_{k_1k_2...k_n}$ }, where k_i is not divisible by *p*.

C_2 = { $\hat{Z}$ | $\hat{Z}$ is an I/O placement for perfect distance-1 in $T_{q_1q_2...q_n}$ }, where q_i is divisible by *p*.

H = a check matrix for perfect distance-t placement.

D_{lee} = Lee distance between two nodes

Input: $(k_1, k_2, ...k_n, t)$

```
1. Construct a perfect distance-t placement for $T_{q_1q_2...q_n}$ such that $q_i = \left(\left\lceil \frac{k_i}{p} \right\rceil\right) \times p$
        if ( $\hat{X} \cdot H^T$ ) $\equiv 0 (\bmod p)$ then $\hat{X} \in C_2$
2. Determine $\hat{Y} \in C_1$
        for each $\hat{Z} \in C_2$
          for  i = 0 to n-1  do
            if $z_i \ge k_i$ then $\hat{Z} \notin C_1$ and break
            else continue;
          end for
          add $\hat{Z}$ to $C_1$
        end for
3. Assign compute-nodes to I/O node
        for each $\hat{Y} \in C_1$
            if $D_{lee}(\hat{X}, \hat{Y}) \le t$ then mark and assign compute node $\hat{X}$ to $\hat{Y}$
        end for
        for each unmarked compute node $\hat{X}$
              determine $\hat{Y} \in C_1$ such that $D_{lee}(\hat{X}, \hat{Y}) \le t + 1$
              if number of $\hat{Y}$ is greater than 1
                then assign $\hat{X}$ to $\hat{Y}$ with the minimum number of compute node assignments
              else assign $\hat{X}$ to $\hat{Y}$
          end for

4. Return Placement
```

3.2. I/O Placement with a Given Number of I/O Nodes

In the previous subsection, we described relaxed quasi perfect distance I/O placement given a value for *t* and particular torus network configuration. However, there might be some situations where there is constrain on the number of available I/O nodes. In particular we consider the problem of I/O placement in torus when the number of available I/O nodes is less than the required minimum bound for the size of a torus network using the relaxed quasi-perfect distance-*t* placement explained earlier.

Consider a torus configuration T (Q_k^n or $T_{k_1 k_2 \ldots k_n}$) with a given number of I/O nodes *W*. Also let t be a positive integer such that

M_t is the smallest integer greater than or equal to *W*

and

M_{t+1} is the largest integer less than or equal to *W*

where

M_t and M_{t+1} are the minimum bounds on the number of required I/O nodes for a perfect distance-*t* or *t+1* from equation 2.1 assuming $t < k/2$ for some radix-*k* in T.

If M_t is closer to *W* than M_{t+1} that is

$$|M_t - W| < |M_{t+1} - W| \qquad (3.3)$$

then we can construct a perfect or relaxed quasi-perfect distance-*t* placement for T and remove some I/O node locations to obtain *W* I/O nodes placement .

Otherwise, if M_{t+1} is closer to *W* than M_{t+1}, with

$$|M_t - W| > |M_{t+1} - W| \qquad (3.4)$$

a perfect or relaxed quasi-perfect distance-*t+1* placement can be constructed for T with addition of some I/O node locations to obtain *W* I/O nodes placement .

However, if

$$|M_t - W| = |M_{t+1} - W| \qquad (3.5)$$

then either the addition or deletion procedure can be used. In what follows, we describe the procedures for adding or deleting I/O node locations in a perfect or relaxed quasi perfect I/O placement.

Case I: Deletion Procedure (When M_t is closer to W)

1) Construct a perfect distance-*t* if each radix *k* of the given torus *T* is divisible by p.
2) Otherwise construct a relaxed quasi-perfect distance-*t* I/O node placement.
3) In a perfect distance-*t* placement, delete some I/O node locations starting from the location with the smallest radix address. For each instance of a deleted I/O node location, *mark* all I/O node locations at distance *t+1* from the compute nodes at distance-*t* to the deleted I/O node as *undeletable*. This will help to maintain minimal distance placement.
4) For each unmarked I/O nodes, repeat the same procedure as in (3) until the remaining number of I/O nodes is equal to *W*.
5) With a relaxed quasi-perfect placement, there are some pairs of I/O nodes with distance between them less than $d = 2t + 1$ the minimum distance between two I/O nodes from Theorem 2. For each pair, delete one of the I/O node locations and mark all I/O node locations at distance t+1 from the compute nodes at distance-*t* to the deleted I/O node as undeletable. Repeat this process until the remaining number of I/O nodes is equal to *W*.
6) However, if the number of remaining I/O nodes for both perfect and relaxed quasi perfect placement is still greater than *W*, then repeat procedure (3) on the remaining marked I/O node locations until it is equal to *W*.

Case II: Addition Procedure (When M_{t+1} is closer to W)

1) Construct a perfect distance-*t+1* if each radix *k* of torus T is divisible by p.
2) Otherwise construct a relaxed quasi-perfect distance-t I/O node placement.
3) For a perfect distance-*t+1* placement, first mark all compute nodes locations at distance-*t+1* to an I/O node location. Starting from the compute node with the lowest node address, make a marked compute node an I/O node location if it is at distance-*t* or less to *2n-1*other marked compute nodes locations, where n is the dimension of the torus.

4) Unmark any marked compute node locations at distance-*t* to the added I/O node location.
5) Continue the process until the number of I/O nodes location equal *W* otherwise, repeat the process with distance-*t*.
6) In a relaxed quasi-perfect distance-*t+1*, there are some compute node at distance-*t+2* to an I/O node. Mark all such compute nodes.
7) Make any marked compute node an I/O node location if none of the other marked compute nodes are at distance-*t+2* to it.
8) Unmark any marked compute node with at most distance-*t+1* to such I/O node location. Continue procedure (7) and (8) until no compute node can be marked.
9) If the number of I/O node locations is still less than W, then repeat (3) to (5).

Figures 9 and 10 illustrate the deletion and addition procedure using Q_{10}^{2} with *W=16* and *W*=12 respectively. With *t=1* and *t=2*, the minimum bound are 20 and 8 I/O nodes. The deletion procedure is used since 16 is closer to 20 . In Figure 9-(a), a perfect distance-1 is first constructed with 20 I/O nodes. Starting with I/O node (0, 0), all I/O nodes in red circles are deleted to form the placement in Figure 10-(b).

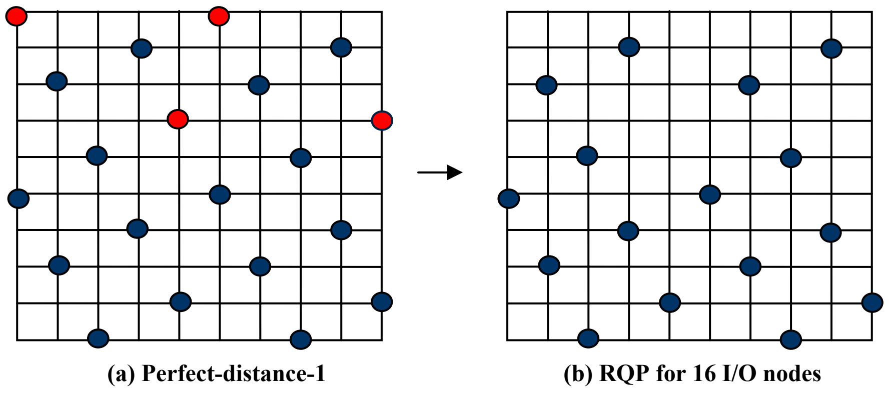


Figure 9. I/O placement Q_{10}^{2} using deletion procedure with 16 I/O nodes.

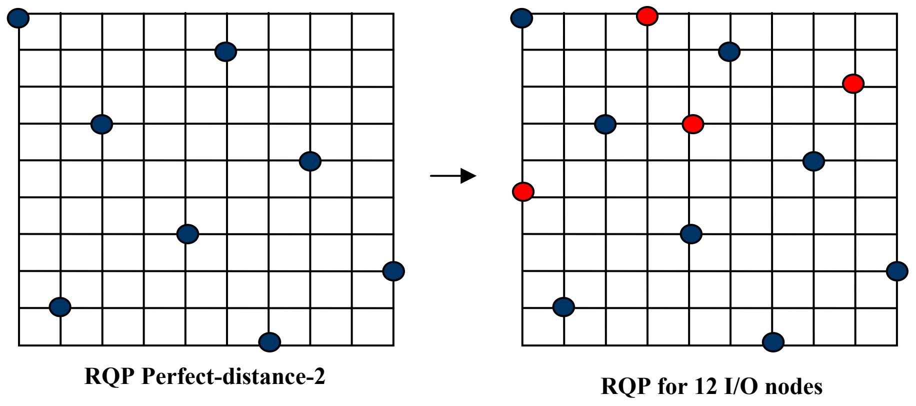


Figure 10. I/O placement Q_{10}^{2} with 12 I/O nodes.

Likewise, the addition procedure is used when *W=12* by first constructing relaxed quasi-perfect distance-2 with 8 I/O nodes. Some compute nodes locations with distance-3 are made I/O node locations until no compute node is at distance-3. Afterward, the procedure is applied to compute nodes with distance-2 to form the final placement. It is worth noting that only the deletion procedure is applicable for 3-dimensional torus since perfect distance-1 is the only possible perfect placement that exists.

Summary of Algorithm Placement with Given a Number of I/O Nodes in Torus

Let W = number of available I/O nodes

N = total number of nodes in the given torus.

p = volume of packing sphere for perfect distance-t placement defined in equation 3.2.

Input: (k,n,W)

Procedure:

1. Determine value of t and t+1 for which

$$M_t = \frac{N}{P_t} \text{ is the smallest integer greater than } W$$

$$\text{and } M_{t+1} = \frac{N}{P_{t+1}} \text{ is the largest integer lesser than } W$$

2. ***if*** $|M_t - W| \leq |M_{t+1} - W|$ ***then*** call the Deletion procedure

   else if $|M_t - W| > |M_{t+1} - W|$ ***then*** Call the Addition procedure

3. Return Placement

3.3. Priority Scheduling

It has been shown that the processor and I/O traffics in general do not usually overlap especially in scientific computation and in situation where they do such as in transaction applications the performance degradation is not very much [20]. However, in this subsection we consider the issue of applying a priority scheduling algorithm at the router level to reduce the effect of the I/O traffic on the performance of the processor communication by giving higher priority to processor traffic.

As stated earlier, a node is considered as having a processor and a router for direct connection in the torus network as shown in Figure 1. The router design and architecture that we propose here is a 5 stage pipelined router [4]. It uses flit-level flow control in which messages are broken down into flits. The flits are further classified into header, middle, and tail flits. The header flit carries the necessary information for routing decisions. The basic architecture of this router is as shown in Figure 11.

There are *n*-physical ports in the router with $n \times n$ crossbar. Each physical port has *m* virtual channels (VC) for deadlock freedom, adaptive routing capabilities and to allow different message types travel the network at the same time. The operation of this pipeline router is as follows. The first stage of the pipeline is the functional unit for synchronization of the incoming flits. The synchronized flits are de-multiplexed and directed to their respective virtual channels (VC) for decoding. The routing decisions and arbitration for the correct crossbar are done in the stage 2 and 3. Only the header flits pass through these stages for the necessary routing decision while the middle and tail flits skip these stages for the fourth stage. Flits get routed to the correct crossbar outport at the fourth stage. The fifth stage of the pipeline router performs necessary buffering of flits from the crossbar and multiplexes the physical channel among multiple VCs and performs hand-shaking and synchronization with the import of the next router in the interconnection network.

The priority scheduler is implemented at the inport of the crossbar. At startup, specific numbers of virtual channels are assigned to the two message types (processor and I/O) depending on bandwidth requirement and message ratio with process given higher

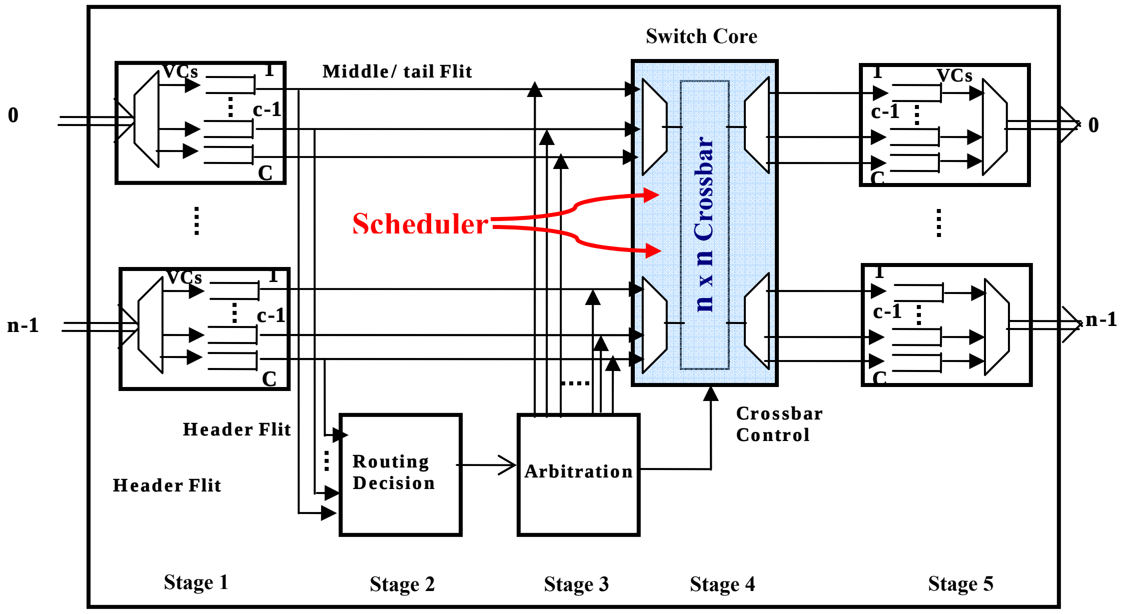


Figure 11. A five stage pipeline router with priority scheduler.

bandwidth allocation. At the inport of the crossbar, the priority scheduler first check all the VCs assigned to process traffic for any incoming messages. Available processor messages are service on a first come first serve basis. Afterwards, I/O traffic are then serviced. In order to avoid starvation for I/O messages, they are always serviced after a predetermined number of process messages using a counter. However, if no process messages are available in their respective VCs, the I/O messages are automatically serviced. We can thus guarantee that the process traffic performs is not really affected by sharing the link with I/O messages.

4. FAULT TOLERANT I/O DESIGN

In this section, we concentrate on developing a fault tolerant scheme with respect to I/O nodes in torus interconnection to minimize additional I/O latencies that may be incurred as a result of failure. Our aim is to apply perfect and relaxed quasi perfect placement policies as the case may be in reconfiguring the network so that other healthy I/O nodes with the shortest distance or hop count can be located for rerouting of I/O traffic from affected compute nodes. At the same time we try to avoid channel overloading by distributing the I/O traffic with equal probability across all possible I/O nodes using deterministic routing.

4.1. I/O Faulty Model

In the design of the fault model, we presume that some diagnosis techniques had already been employed in detecting faulty I/O nodes and concentrate on how to reconfigure I/O communication. A faulty I/O is characterized by the following assumptions;

- Fault occurs only in the I/O nodes. By this we mean the I/O processor can fail while the router is still active.
- No fault in all the links associated with a failed I/O node
- Fault can be transient or permanent.
- Perfect or relaxed quasi perfect distance-t is used for I/O placement

Furthermore, we define *default I/O nodes* as the I/O nodes with compute nodes at a distance-*t* or less in a perfect placement or at most distance *t+1* in a relaxed quasi-perfect distance-t. *Alternative I/O nodes* are I/O nodes with the next shortest distance from a compute node apart from the default.

4.2. Determining Alternative I/O Node

We basically exploit on some of the unique properties of perfect distance and relaxed quasi perfect distance I/O placement, in providing a minimal path to other healthy I/O nodes within the torus network. In what follows, we analyze some of these properties and how it can be applied in reconfiguration of I/O in the event of faults. Perfect distance-t and RQP, provides some level of redundancies, a key element in the design fault tolerant system.

Theorem 3. *In a perfect distance-t I/O placement, a compute node with the same radix position as its default I/O node along any dimension and at a distance 0<r<t to it, has a shortest distance d–r to n alternative I/O nodes, where n is the dimension of the torus, and d is the minimum lee distance between the I/O nodes.*

Proof.
Consider a perfect distance-t placement in a *k*-ary *n*-cube or mixed torus with a set of I/O node locations generated using Lee distance error code. This placement forms a linear code with *t*-error capability and a minimum distance *d = 2t +1* between two non-zero I/O node locations. Also we know that each node in a torus network has *2n* node degree incidental on it with two nodes adjacent to it along each dimension. It implies that any I/O node has *2n* other I/O nodes at minimum distance *d* from itself along each dimension. Since each dimension has two opposite directions, then we have *n* I/O nodes in each direction. Thus any compute node at distance *r* to an I/O node along the same radix position will also be at distance-(*d*-*r*) to the other n I/O nodes in that direction. The correctness of the theorem can be demonstrated using a perfect distance-1 placement in a Q_5^2. □

Figure 12 shows the five I/O nodes required for this perfect distance-1 placement. Taking I/O node J (1,2) as a reference point, we notice that each of the four remaining I/O nodes I, K, L, M has exactly Lee distance 3, the minimum distance for perfect distance-1. J has two I/O nodes with distance-3 at each side. Compute node X (2, 2) share the same radix position 2 with its default I/O node-J along the x-dimension and at distance-1 to J therefore, it also has a distance (3-1=2) to two alternate I/O nodes L (3,1) and K (2,4) as shown with the arrow.

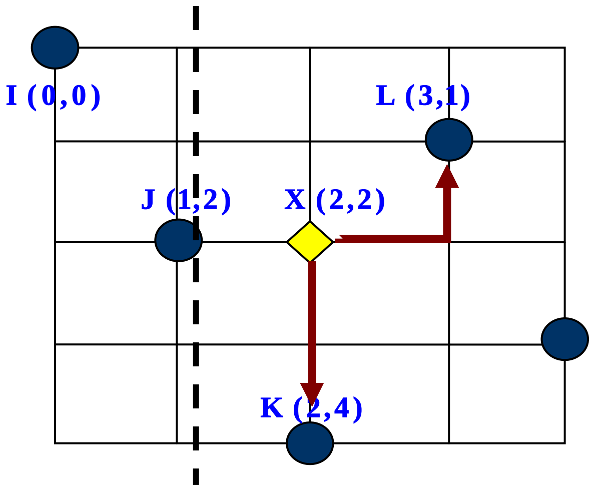


Figure 12. Redundancy in perfect distance-1 for Q_5^2.

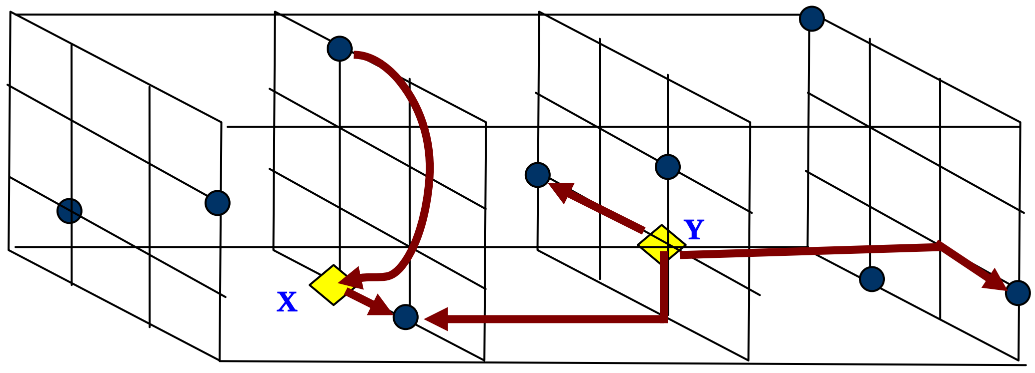


Figure 13. RQP distance-1 redundancies in Q_4^3.

Relaxed Quasi perfect distance-*t* also exhibits the same redundancy properties as described in Theorem 3. In addition, RQP also provide extra redundancies using Theorem 2 and Corollary 1 since, two I/O nodes may have a minimum distance *d-1* such that a compute node between such I/O nodes is at a distance-*t* to more than one I/O node. Hence if any of the I/O nodes fails, then such compute node can still use the other I/O available I/O nodes. Figure 13 shows RQP distance-1 in Q_4^3 with a compute node X at distance-1 or adjacent to two I/O nodes and compute node Y at distance-2 to three other I/O nodes.

The algorithm to determine the alternate I/O nodes for each of the affected compute nodes is given as follows;

```
Let H = check matrix
$\hat{Y} = (y_{n-1}y_{n-2}...y_0)$ be the location of an affected compute node
$\hat{X} = (x_{n-1}x_{n-2}...x_0)$ be the location of nodes at lee distance-(d-r) i.e. $D_{lee}(\hat{X},\hat{Y}) = d - r$
 where d is the minimum distance between I/O nodes and
     r is  the distance of an affected node to its default I/O node
Input ($\hat{Y}$) Affected compute node
Procedure:
  for each $\hat{X}$ orthogonal to $H^T$ ∍ $\hat{X} \cdot H^T = 0$
      mark $\hat{X}$ as a possible alternate I/O node
      if $\hat{X}$ is along the same radix as the faulty I/O node
        then the number of such possible $\hat{X}$ = n.
          Select a $\hat{X}$ as alternative I/O node using deterministic routing concept
      else
              $\hat{X}$ is the alternate I/O node
   end for
  Return: alternate I/O node.
```

4.3. Selecting among Multiple Alternate I/O Nodes

Careful selection I/O node when multiple alternatives exist is very important for efficient load balancing at the channels and I/O nodes. We can achieve this by redirecting all I/O traffic from affected compute nodes to alternate I/O nodes using deterministic routing algorithm concept. A deterministic routing such as dimension-order, (X-Y) routing in 2-dimensional or (e-cube) in 3-dimensional torus usually take a path from the source to the destination by reducing an offset in one dimension to zero before considering the offset of the next dimension in increasing or decreasing order.

Whenever, a compute node has more than one alternate I/O nodes, it should select an I/O node having the same radix position in the lowest dimension (X). Otherwise if none exist, the next higher dimension (Y) is checked and so on. When no I/O nodes falls on the same radix, then another round of comparison is done with the next radix positions adjacent to the current radix position on which the compute node is located, until a match is found. A detailed analysis is as shown in Figures 14 and 15 using relaxed quasi-perfect distance-1 for Q_8^2 torus. In Figure 14, a faulty I/O node L (4, 3) in red circle causes all the four adjacent compute nodes in yellow square to redirect their I/O traffic. Each of the compute nodes has two possible alternate I/O nodes at distance-2 as described in the previous subsection. Consider the compute node A (4, 2) with distance-2 to alternate I/O nodes I (3,1) and J (5,2). Using deterministic routing scheme, we first compare the x-dimension of the two I/O nodes to see if any one has the same radix with the compute node by checking the x. coordinate values of the address of the two nodes. J is finally selected as the next alternate I/O node for A since they are both on radix position 2 of the x-dimension. Compute node B (3, 3) also has alternate I/O nodes I (3,1) and K (2,4). On checking the x-dimension, none of the coordinate values is equal to 3 meaning they are not on the same radix position along x. However, I/O node-I fall along the same radix-3 on the y-dimension, therefore, *I* is chosen as the next alternate node for sending its I/O traffic.

The same procedure is repeated for the remaining compute nodes. When there is only one available alternative I/O node, an affected compute node simply select the only available next shortest I/O node. We show such situation in Figure 15. Here, I/O node-J is now faulty requiring redirection for all the connected compute nodes. First of all, each of the four adjacent nodes is redirected to the next alternative I/O nodes with distance-2 using the same procedure as Figure 14. However, compute nodes A (4, 2) and C (6,3) has only one available alternative I/O node due to the two faulty I/O nodes J and L. Therefore, compute node A initially assigned to I/O node-J at distance-2 in Figure 15 is now reassigned to its other available I/O node-I shown with brown arrow still maintain its distance-2. Also compute-node C should have selected I/O node L (4, 3) using the deterministic approach but since it is already faulty, it selects the only available I/O node M (7, 2) at distance-2. More occurrence of I/O node failure may lead to compute nodes selecting some other I/O nodes at distance-3 or greater depending on the failure rate.

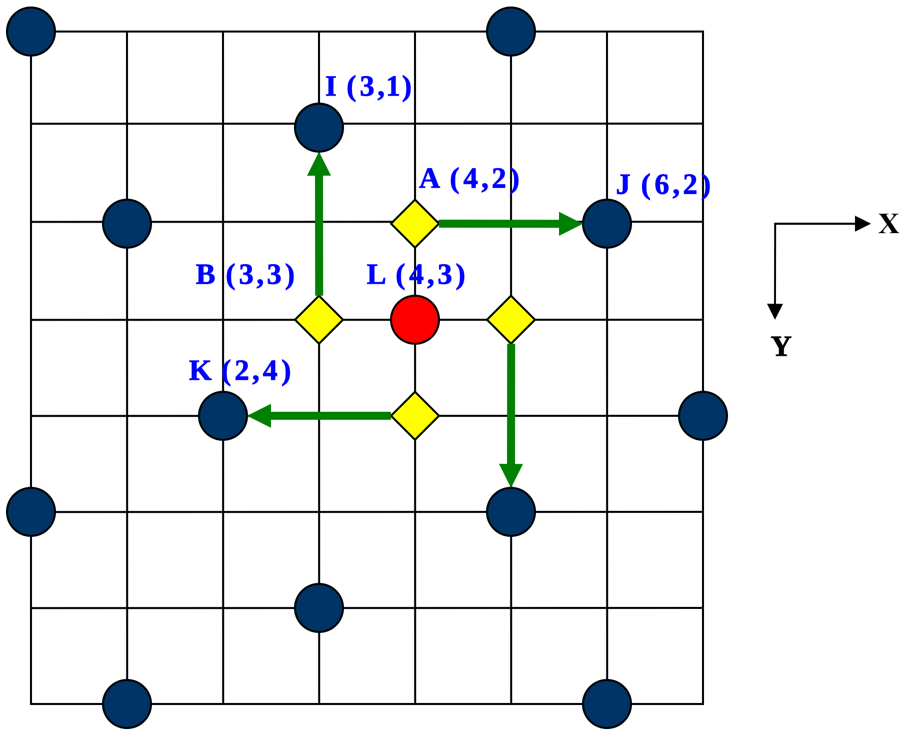


Figure 14. I/O node selection with multiple available alternate I/O nodes.

4.4. Local I/O Nodes Status Update

Status update information allows the compute nodes to keep track of which I/O nodes are healthy or faulty. This will prevent sending I/O traffic to unavailable I/O nodes. Status information updates can be done globally or locally. The global update involves broadcasting all I/O nodes status update to all compute nodes in the interconnection while local updates are restricted to computes nodes in a faulty region. We consider using localized fault information updates to minimize traffic. When an I/O node fails, all the compute nodes at distance-t or less for perfect distance-t or at most distance-($t+1$) in relaxed quasi perfect is assumed to be aware of the failure.

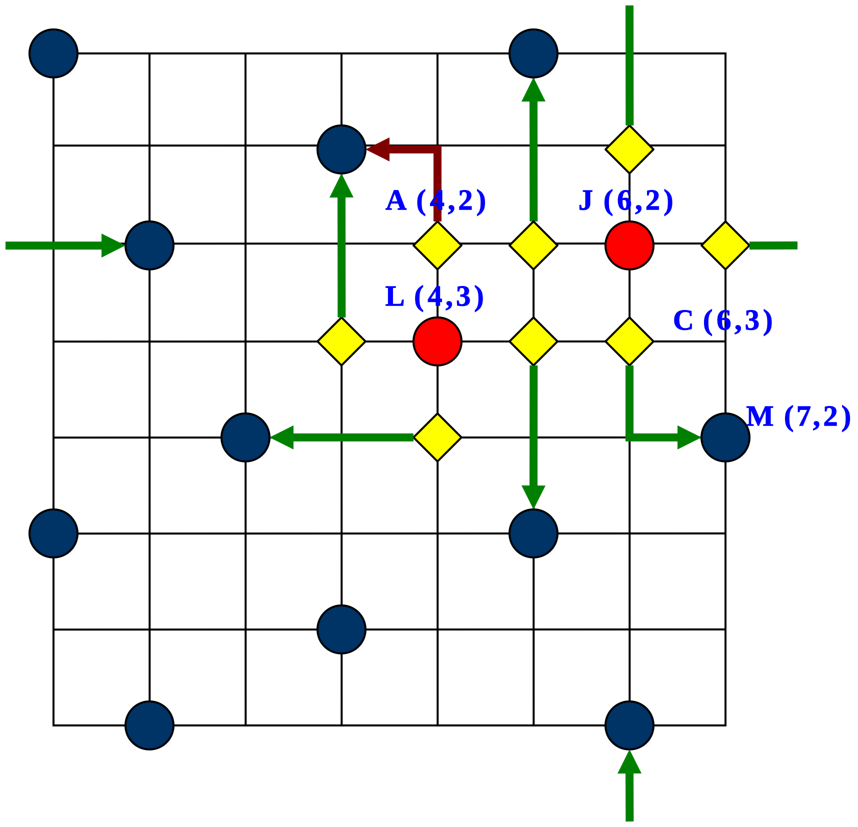


Figure 15: I/O node selection with more than two faulty I/O nodes.

The compute nodes at the boundary distance-*t* or *t+1* from the faulty I/O node notify their corresponding adjacent compute nodes assigned to some other I/O nodes of the I/O node failure. In most cases, these neighboring compute nodes will have the faulty I/O node as an alternative. A localized fault region is formed around the faulty I/O nodes covering all directly affected compute nodes and their neighbors.

Figure 16-(a) and (b) illustrate the status update mechanism using a perfect distance-1 in Q_{10}^2 torus. The red and blue circles represent faulty and healthy I/O nodes. The yellow diamond nodes are the directly affected I/O nodes while the black rectangle ones are the neighboring computes nodes. In Figure 16-(a), all the four compute nodes at distance-1 to the faulty I/O nodes notify their adjacent neighbors of their default I/O node failure. Each of the adjacent neighbor nodes has the faulty I/O node as an alternate with distance-2, should their default I/O nodes fails. The computes nodes outside the faulty region are not affected and therefore not updated since none of them is at a distance-2 to the faulty I/O node. With the status update, when any of the neighbors default I/O node becomes faulty as in Figure 16-(b), their I/O traffic will be forwarded to the other alternate healthy I/O node and another fault region shown in grey is formed again. The directly affected computes nodes now use the other alternate I/O nodes with distance-2 rather than the faulty one in the green region as shown with the red arrow. This cycle continues as more I/O nodes fails with the faulty region covering more computes nodes and some region overlapping each other depending on the location of the fault until a stage when very few I/O nodes are available.

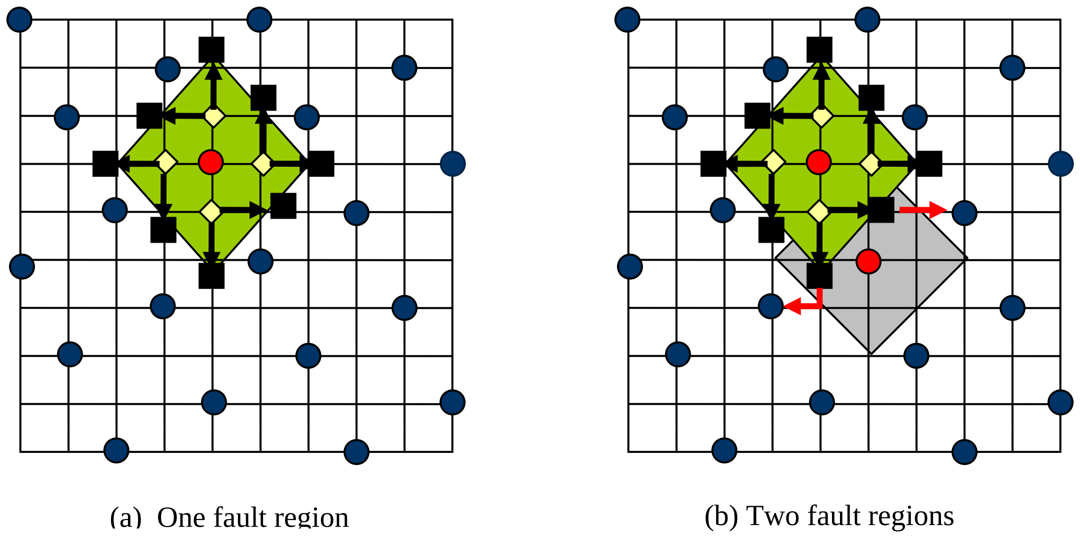


Figure 16. Fault region generation and neighbor update mechanism.

At that point, all computes nodes might have global information about the status of all the I/O nodes. Note also that the distance to alternate healthy I/O nodes increases too as the more I/O nodes fail. If any of the faulty I/O nodes ever recover as in transient failure, the same principle is applied in notifying the neighboring compute nodes of any changes in the status information.

5. SIMULATION ENVIRONMENT AND RESULTS

5.1. Simulation Setup and Environment

In order to analyze the effect of RQP I/O nodes placement in a torus network, we adapted a cycle based simulator developed in [15] for our purpose. The simulator provides the flexibility to simulate any kind of interconnection topology. It is written using CSIM libraries [8].

The simulator is made up of the message generation, the router, and the interconnection modules as shown in Figure 17. The message generation module acts like a processor and basically generates messages. It generates two types of messages, the process and I/O messages. Both messages are generated at a given inter-arrival times depending on the message size, message ratio and offered input load and follow the exponential distribution. The offered input load is the number of messages (flits) delivered per cycle. The messages are injected into the router module that performs the actual message forwarding along the path to the destination.

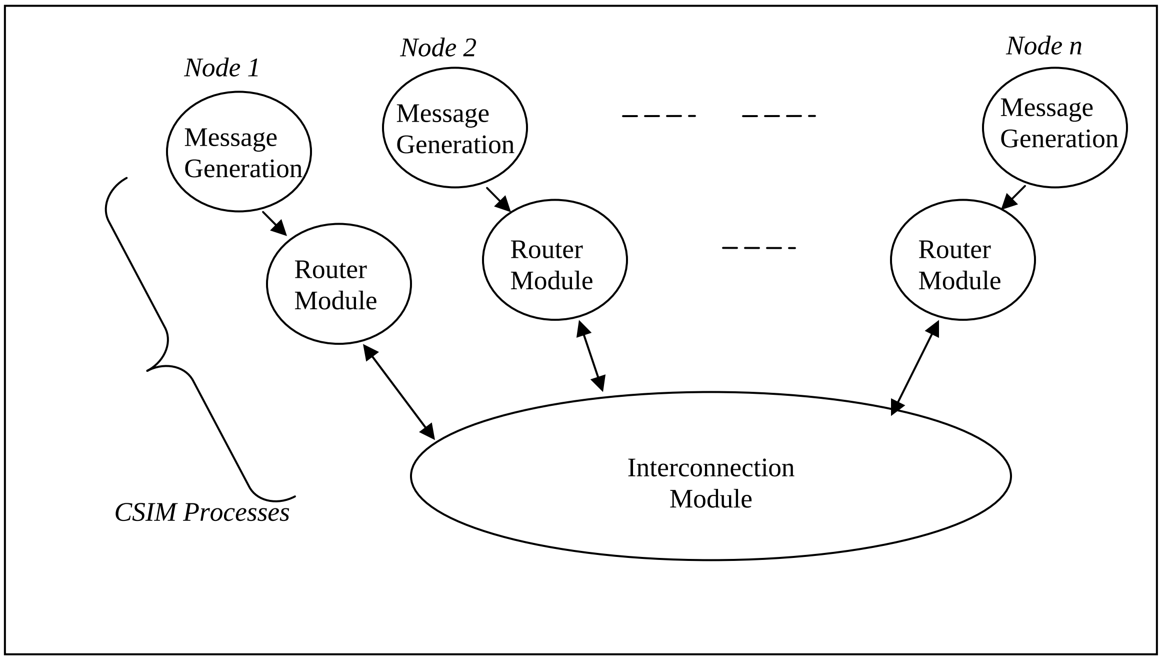


Figure 17. Simulation setup modules.

The torus network is implemented in the interconnection module. As a whole the simulator can be abstracted as in the Figure 1 with the node (compute or I/O) represented as a collection of processor and router using the processor generation and router module.

Processor message destination is determined using random generation function while I/O message destination depends on the I/O placement strategy. Messages are passed from one stage of the interconnection to another using flit flow control mechanism assuming the unit of transfer is flit/cycle. All the flits belonging to a message are reassembled back at the destination nodes. The flow control also uses credit based packet switching where messages are only received at the next stage in the interconnection when there are enough buffer spaces to accommodate a whole message. In this case, the header flit first checks for available buffer allocation otherwise, such messages are block until another cycle with enough buffer spaces. The important output parameter measured in our experiment is the average network latency for both process and I/O traffic. For each message, latency is measured from the time the first flit is generated by the source node to the time the last flit of a message is received at the destination. The average network latency is thus the overall latency over all messages for each traffic type.

5.2. Simulation Results and Analysis

The main parameters used in our simulation are as shown in Table 2 below.

TABLE 2
Simulation Parameters.

Network Size	8X8, 4X8, 4X4X4 Torus
Physical Link Bandwidth	2.5 Gbps
Number of Physical Links	5 or 7
No. of Virtual channels / Physical Link	8
Flits Size	256 bits
Process message size	32 flits
I/O message size	128 flits
Switching mechanism	Packet switching
Deterministic routing algorithm	X-Y or e-cube

5.2.1. Relaxed Quasi Perfect versus Quasi Perfect

In order to measure the effectiveness of our scheme (RQP) compared to quasi-perfect placement, we simulated a Q_8^2 (8X8) torus for RQP and QP distance-1. Each I/O placement strategy was simulated at constant 10% and 20% I/O ratio while varying the offered input load. The result obtained is as shown in Figure 18.

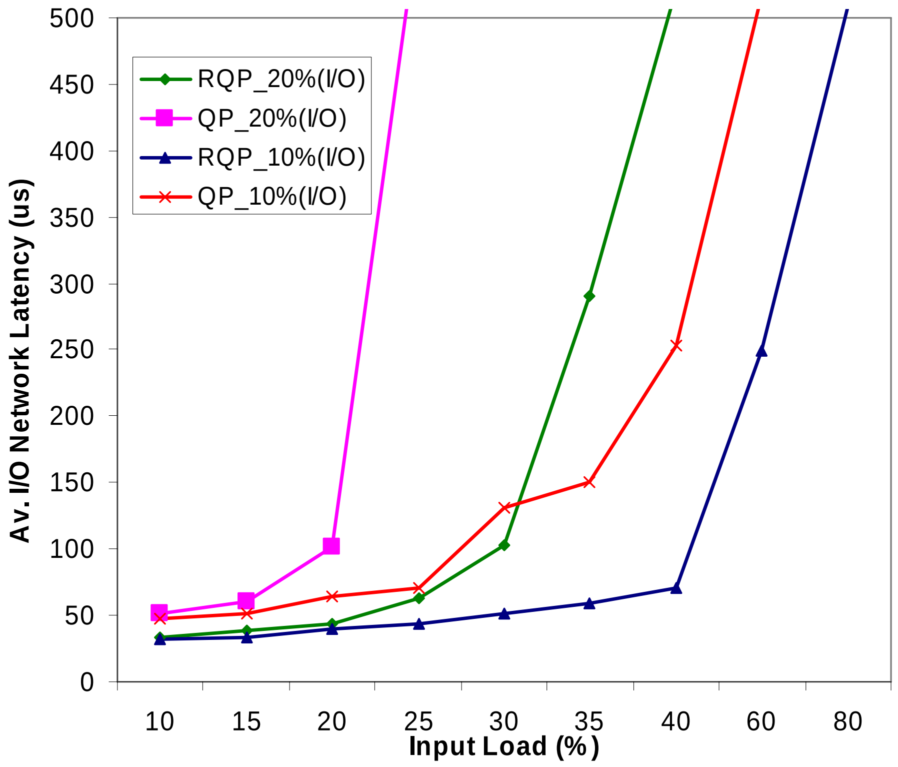


Figure 18. RQP versus QP distance-1 I/O in Q_8^2 varying input load.

At constant 10% I/O ratio, the difference in the average I/O network latency for both schemes is small at low input load. However, as the input load increases, there is a clear gap between RQP and QP with RQP maintaining lower average latency until it reaches the saturation point. RQP is also able to achieve higher offered load (60%) than QP with (45%). At 20% I/O ratio, the difference is clearer with RQP maintaining lower average latency while QP quickly enters saturation at 25% input load compared to 40% in RQP.

5.2.2. Relaxed Quasi Perfect and Base I/O Placement.

As mentioned in the introduction, base I/O placement is the conventional I/O node placement found in commercial high performance computers with I/O node concentrated at the base radix or plane of torus network as shown in Figure 1.

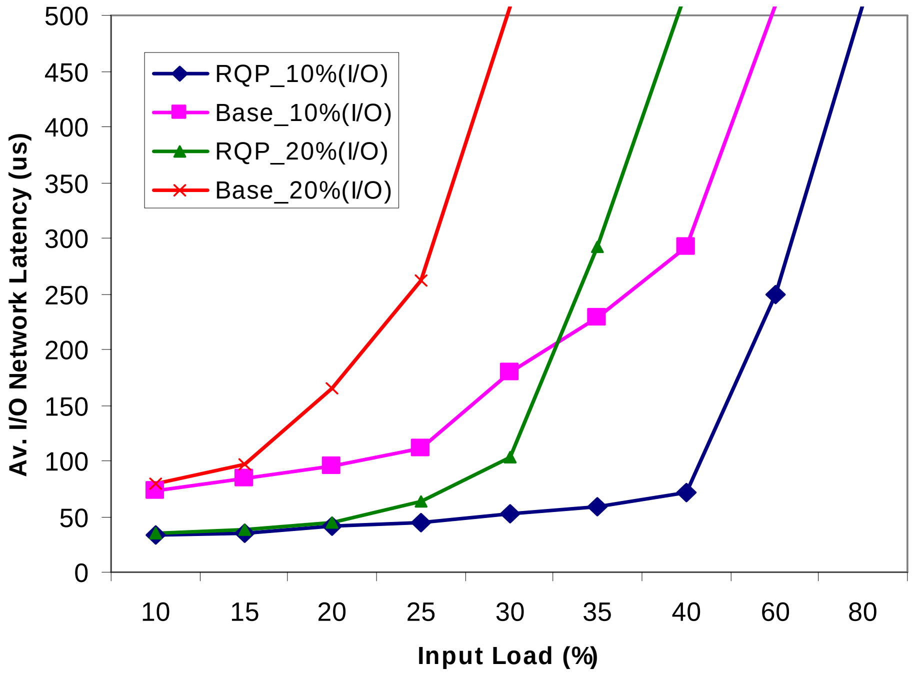


Figure 19. RQP versus base distance-1 in Q_8^2 varying input load.

We analyze the performance benefit of careful placement of I/O nodes within the torus network using RQP especially for those networks without perfect-distance-*t* placement. Figure 19 shows the effect of RQP on the average network latency compared to the Base placement in Q_8^2 (8X8) torus at constant I/O ratio 10% and 20% as the input load is increased. Clearly, RQP out performs the base placement with substantial lower average I/O network latency even at low input load. The base placement becomes saturated at lower 50% and 30% of offered input load at constant 10% and 20% I/O ratio compare to RQP. Similar comparison was performed for smaller T_{8X4} (32 nodes) and 3-dimensional

Q_4^3 (4X4X4) torus with distance-1 placement as shown in Figures 20 and 21. For the smaller (8X4) torus in Figure 20, at 5% and 10% I/O ratio, RQP still outperforms base placement in maintaining lower average I/O latency and sustained input load. However, a general trend here is that the latency is higher with little performance difference between RQP and base than in 8X8 torus. A reason for this result is that there are fewer nodes injecting more messages injected to the network than in 8X8 torus, leading to higher contention. Hence, it implies that the network size has an effect on the performance with RQP showing a outperforming base as the size becomes larger.

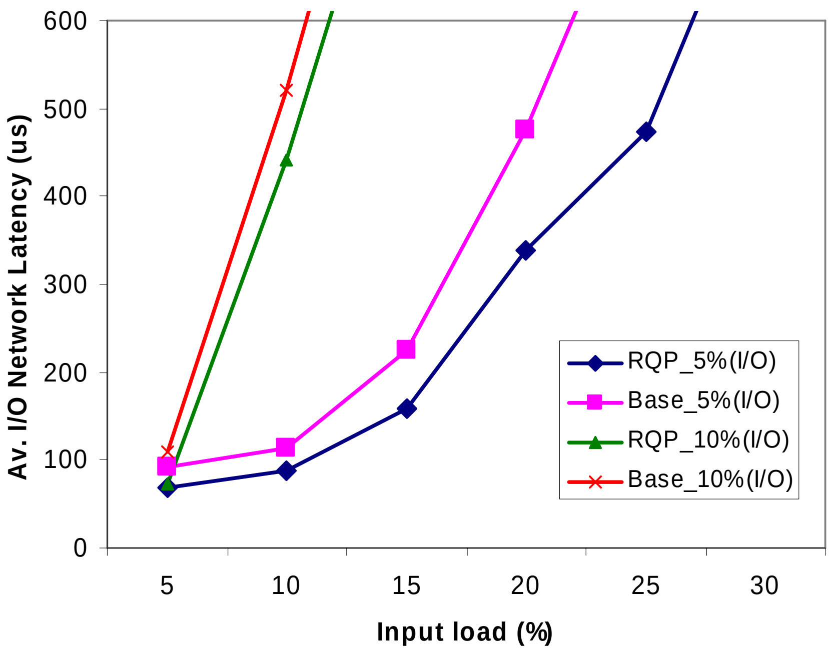


Figure 20. RQP versus base distance-1 in $T_{8X4,}$ (8X4) torus varying input load.

Figure 21 with 4X4X4 torus shows similar trend in performance as in 8X8 torus with RQP maintaining lower average I/O network latency. Also for both RQP and base placement, the sustained offered input load is generally higher than in 8X8 torus. This is expected since the degree of the network is higher in 4X4X4 torus.

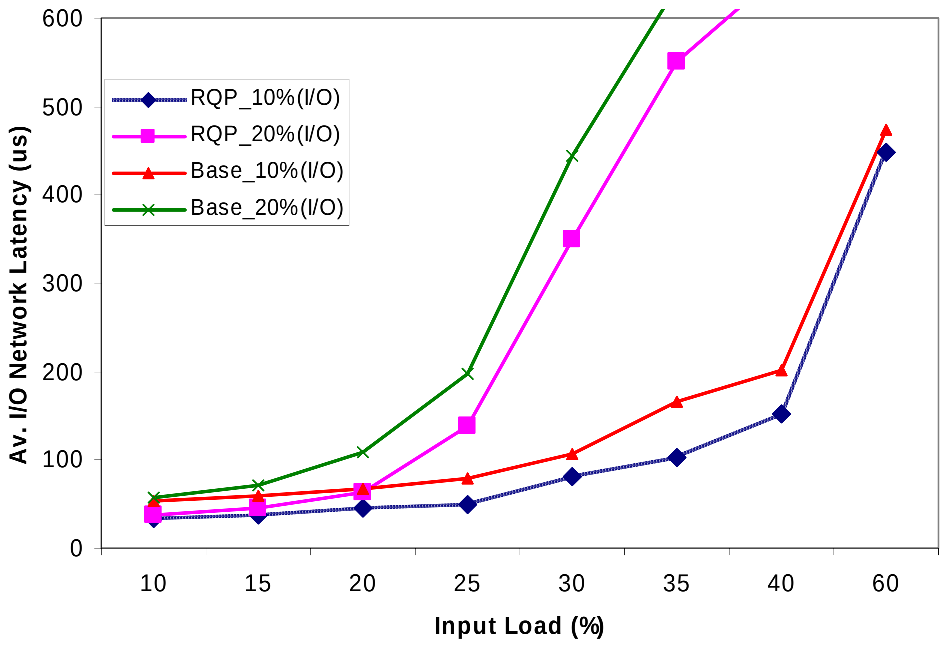


Figure 21. RQP versus base distance-1 in Q_4^3, (4X4X4) torus varying input load.

We also observe the effect of I/O ratio on the performance of RQP, at constant input load for 8X8 and 4X4X4 torus. Figure 22 shows the results obtained for RQP distance-1 in 8X8 at 15% and 25% input load. Increasing the I/O ratio at 15% input load has little effect on the average network latency of RQP but the effect is significant for a base placement especially at high I/O ratio of above 40%. At higher input load 25% there is a general linear increase in the latency for both schemes with RQP still maintaining lower latency. A similar result is observed with a 3-dimensional Q_4^3 (64 nodes) torus as shown in Figure 23. But at higher input load of 25%, the effect of increasing I/O ratio is more significant than in the two dimensional torus.

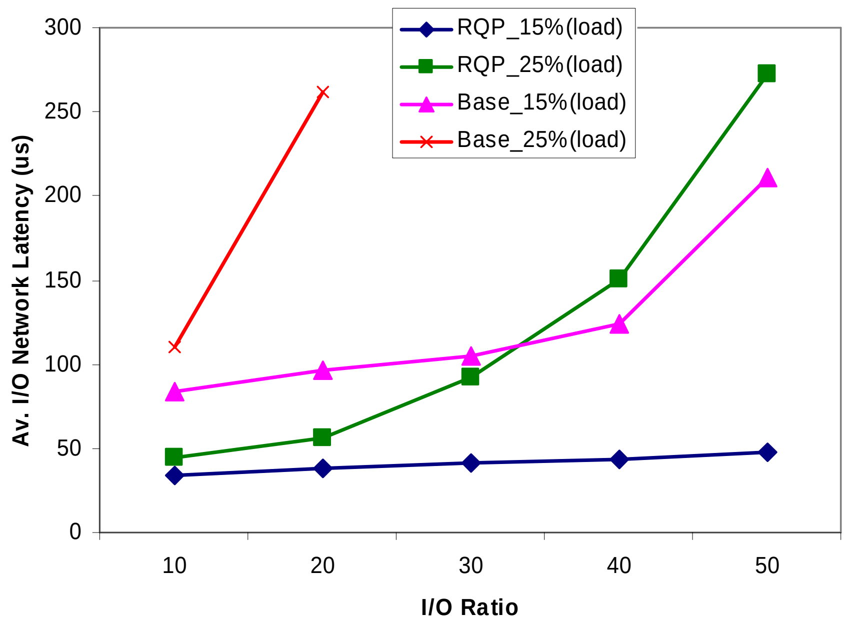


Figure 22. RQPdistance-1versus base in Q_8^2 (8X8) torus varying I/O ratio.

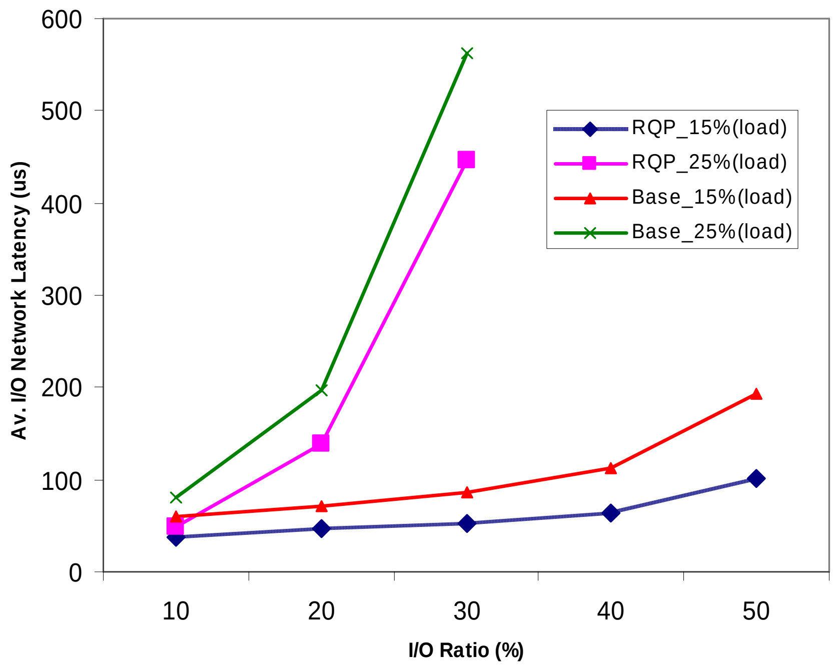


Figure 23. RQP distance-1 versus base in Q_4^3 (4X4X4) varying I/O ratio.

5.2.3. Effect of I/O Placement with Given Number of I/O Nodes

When the number of available I/O nodes is less than the required minimum bound for RQP, then we expect the average latency to increase. However, if the number is within the minimum bound, RQP can still provide a lower latency placement. In order to measure the effectiveness of RQP under this condition, we compared the slowdown obtained in Q_4^3 when the number of 8 and 7 available I/O nodes against the 9 I/O nodes minimum bound as shown in Figure 24 and 25 respectively. Figure 24 compares these placements at constant 10% I/O ratio while the input load is varied. RQP with 8 I/O nodes achieves almost the same performance as with the normal RQP placement but the difference becomes obvious as with 7 I/O nodes. Both base placement with 7 and 8 I/O performed worse with higher average latency and saturates at lower input load.

In Figure 25, the average I/O latency for RQP (RQP-8) and Base (Base-8) placement are compared the normal RQP placement. At different I/O ratio we observe a general linear increase in the average latency as the I/O ratio increases across all configurations as explained in previous results. The RQP_8 I/O shows a slightly constant increase in average latency compared with the normal RQP. On the other hand, there is a significant increase in the average latency in base_8 I/O placement compared to normal RQP. The difference between the RQP-8 and Base-8 tend to be smaller as the I/O rate increases since contention at the I/O node becomes a limiting factor. The figure also shows a similar result with 7 I/O nodes. While the RQP-7 stills out performs Base-7 in providing lower average latency compared to the normal RQP, the slowdown tends to be higher than in RQP-8. This is expected because, the more the number of I/O nodes deviates from the minimum bound for RQP in distance-1, the more the number of compute nodes at distance-2 or greater to an I/O node and eventually, at a stage, it becomes a RQP distance-2 or even distance-3.

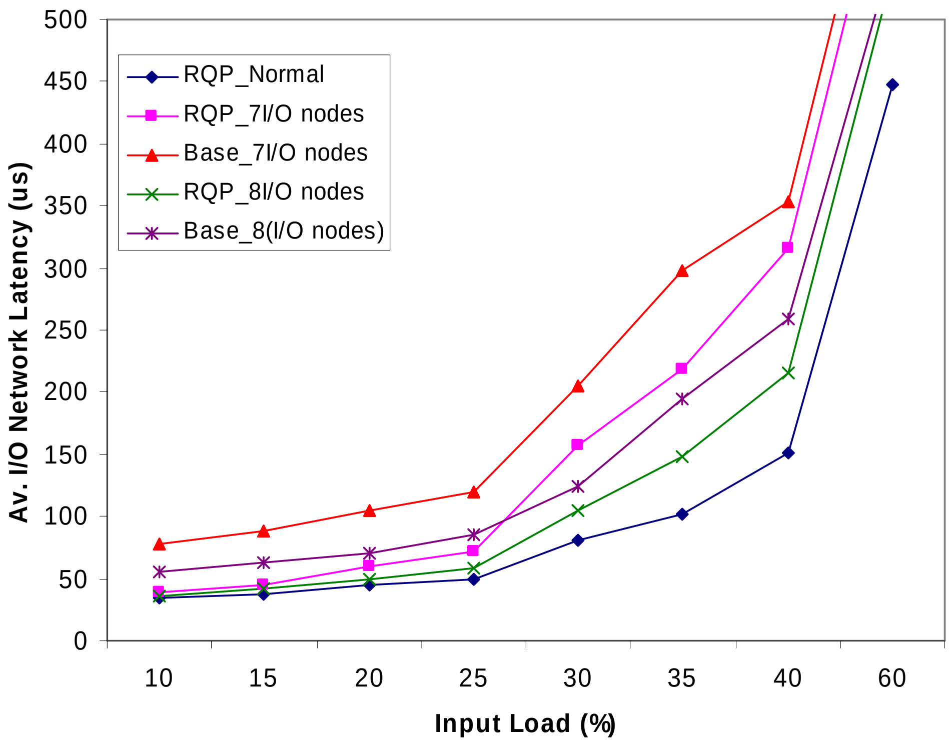


Figure 24. Normal RQP distance-1 against 7 and 8 I/O nodes in Q_4^3 varying input load.

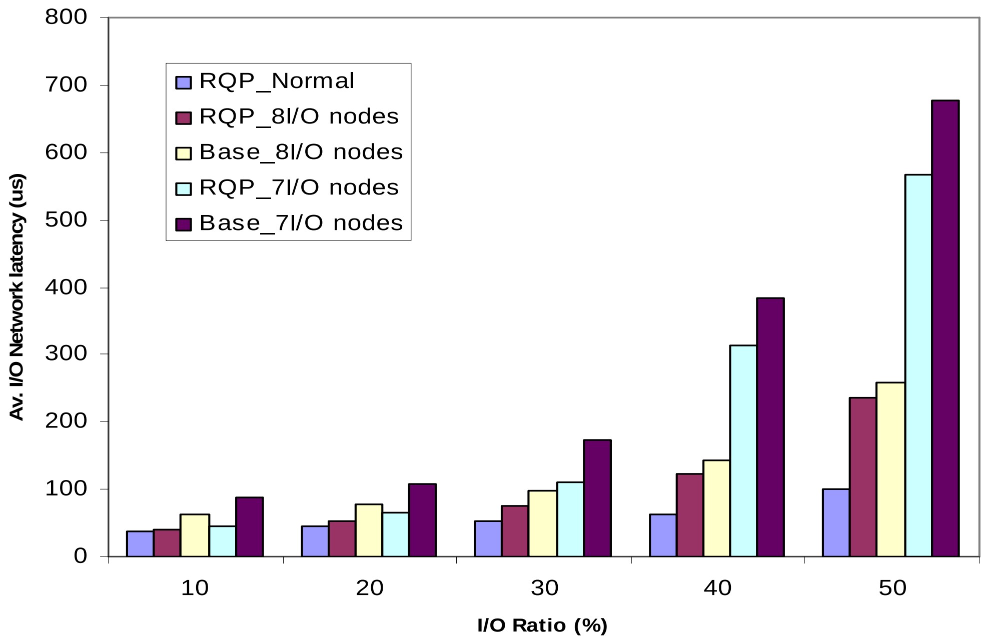


Figure 25. Normal RQP distance-1 against 7 and 8 I/O nodes in Q_4^3 varying I/O ratio.

It is worth noting that if the number of available I/O nodes is so small compared to the minimum bound, the RQP I/O placement strategy or any other placement such as perfect or quasi-perfect might have the same performance as the base because of contention. In essence I/O placement becomes insignificant in the overall reduction in the I/O latency.

5.2.4. Effect of Priority Scheduling on Processor Communication Performance

We simulated a Q_4^3 torus using RQP distance-1 placement with and without priority scheduler at 20 % input load while varying the I/O ratio. The result obtained is as shown in Figure 26.

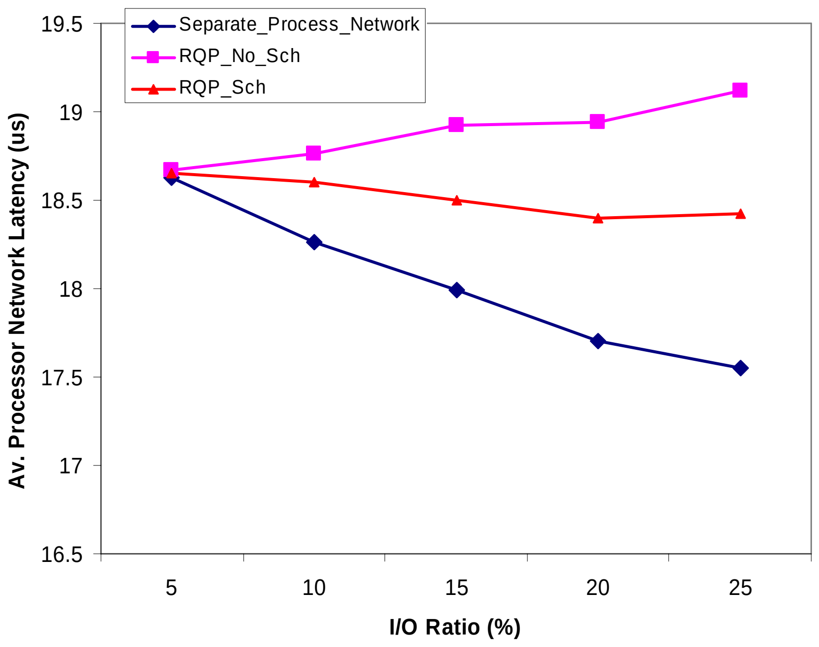


Figure 26: Effect of scheduler on inter-processor performance.

In a separate processor network, processor traffic do not share links with I/O traffic. The average network latency for this network, show a linear decrease as the I/O ratio increases. At a very low I/O ratio, processor ratio is very high causing an increase in the latency due to contention among the processor messages. But the latency decreases with increase in I/O ratio. Unlike separate processor network, the processor communication performance in RQP I/O placement with no priority scheduler (RQP_No_Sch) tends to maintain the average latency as the I/O ratio increases. At higher I/O ratio, the difference becomes clearer. With application of a priority scheduler to the RQP placement, the average processor latency is reduced considerably at higher I/O ratio. It is important to note that at very low I/O ratio, the scheduler has little or no effect on the performance since traffic is dominated by processor messages. A general trend that can be observed is that though sharing links between I/O and process traffic might affect computational performance only at high I/O messages ratio or extremely highly data intensive application with substantial overlap which is rare in real implementation. Otherwise, there is a slight deterioration in performance which can be reduced by giving higher priority to process traffic whenever there is contention.

5.2.5. I/O Placement and Fault Tolerance

We also analyzed the effect of our fault tolerance scheme based on perfect or relaxed quasi perfect I/O Placement in minimizing the average I/O network latency in the presence of faulty I/O nodes. A Q_4^3 torus network was simulated with RQP distance-1 placement while I/O nodes were randomly made faulty or unavailable. We compared the slowdown obtained with 1, 2, 3, and 4 faulty I/O nodes relative to when there are no faulty I/O nodes as the I/O ratio increases. The effect of increasing the input load on the slowdown is shown in Figure 27. Just as the case in Figure 24, with 1, 2 or 3 faulty I/O nodes, the slowdown is minimal but as it increases to 4 faulty I/O nodes the slowdown becomes significant. Also when I/O ratio is varied at constant 15% input load, minimal slowdown is observed with up to 3 faulty I/O nodes as shown in Figure 28. In general, we can say the fault tolerant redirection scheme provides a minimal slowdown, when the number of available I/O nodes is greater than half the required minimum lower bound for a distance-t placement.

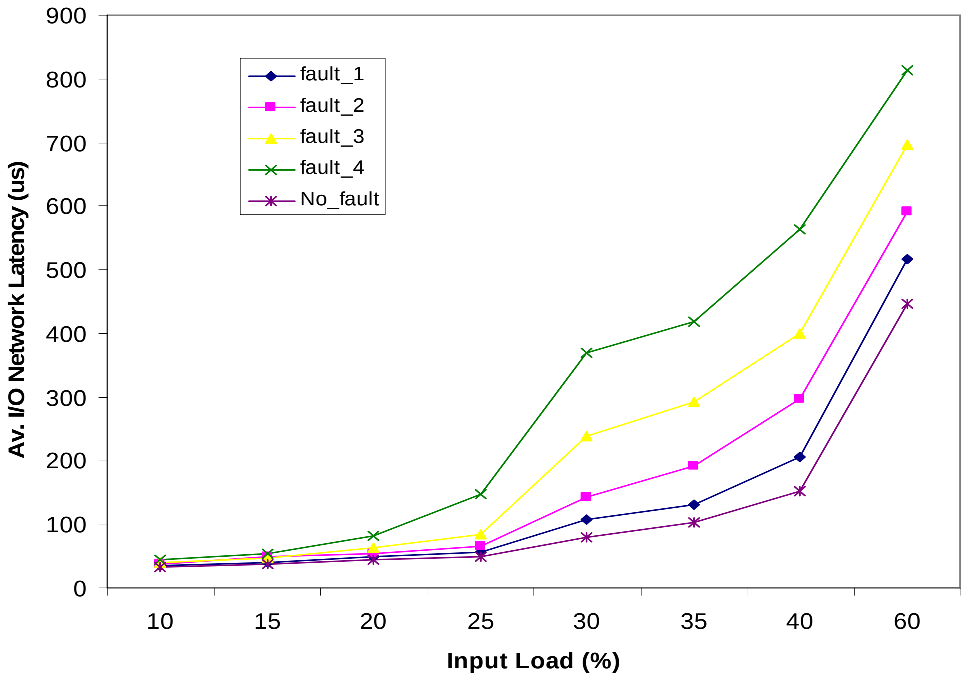


Figure 27. Slowdown effect in RQP for Q_4^3 due to faulty I/O nodes varying input load.

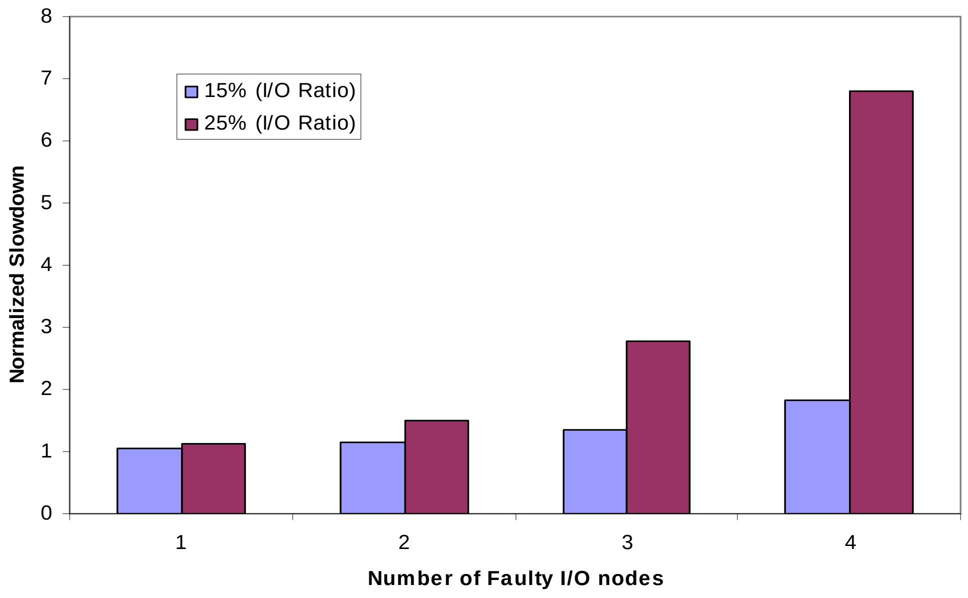


Figure 28: Slowdown in RQP for Q_4^3 due to faulty I/O nodes varying I/O ratio.

6. CONCLUSION

In this thesis we analyze the issue of careful I/O node placement in torus based interconnection as an important aspect in reducing I/O latency in an I/O interconnection architecture where both compute nodes and I/O nodes are integrated in the same network sharing the same links. We proposed relaxed quasi-perfect (RQP) distance-*t* I/O placement as an alternative when there exist no perfect distance-t placement for a particular torus network. We further analyzed the advantage of RQP against the quasi-perfect distance-*t* placement as a better optimal placement. We also showed how to construct I/O placement in situation where the number of available I/O nodes are less than the minimal bound required for a perfect distance-t placement. In order to reduce any performance degradation of processor traffic as a result of sharing links with I/O traffic, we implemented a priority scheduler with the former given higher priority. An I/O traffic redirection scheme using RQP or perfect distance-*t* placement was also presented to provide minimal slowdown in the event of I/O nodes failure.

Based on our analysis and simulation results RQP out performs both QP and the conventional I/O placement strategy where I/O nodes are concentrated at the base of the torus interconnection. In general, the RQP provides a lower I/O network latency with higher sustainable applied input load as contention due to input load and I/O ratio and the torus network size increases,. Substantial improvement in processor communication was also observed with the implementation of a priority scheduler especially when the ratio of I/O is very high. Finally, the fault tolerant schemes provided a minimal slowdown especially when the number of faulty I/O nodes is more than half of the lower bound on the number of required I/O nodes for a particular distance-t placement.

As a future work, further improvement is required for the fault tolerant model by considering other factors such as links or router failure to provide a more realistic robust tolerance schemes. In these cases, creating an interconnection with redundant I/O nodes and links using a *j*-adjacency distance-*t* where compute nodes are adjacent to *j* I/O nodes and at a distance-*t* to it may be alternative solution.

REFERENCES

[1] G. Almasi. “An Overview of the BlueGene/L System Software Organization,” *Proc. Euro-Par’03 Conf. Parallel and Distributed Computing*, pp.543–555, August 2003.

[2] B.F. AlMohammad, B. Bose, “On Resource Placements in 3D Tori”, *Fifth World Multi-Conference on Systemics, Cybernetics and Informatics*, vol. 5, pp. 96–101, July 2001.

[3] B.F. Almohammad, B. Bose. “Resource Placement in 2D Tori,” *International Parallel and Distributed Processing Symposium*, pp. 431-438, March 1998.

[4] M. Bae and B. Bose, “Spare Processor Allocation for Fault Tolerance in Torus-Based Multicomputers”, *Proc. 26th Int’l Symp. Fault-Tolerant Computing*, pp. 282-290, June 1996.

[5] M. Bae, B. Bose, “Resource Placement in Torus Based Networks,” *IEEE International Parallel Processing Symposium*, pp. 327–331, April 1996.

[6] B. Bose, B. Broeg, K. Younggeun, Y. Ashir, “Lee Distance and Topological Properties of k-ary n-cubes”, *IEEE Transactions on Computer*, vol. 44., no. 8, pp. 1021-1030, August 1995.

[7] J. Bruck, R. Cypher, and D. Soroker, “Tolerating Faults in Hypercubes using Subcube Partitioning,” *IEEE Transaction Computing*, vol. 41, no. 5, pp. 599-605, May 1992.

[8] CSIM, *User’s Guide: CSIM18 Simulation Engine (C version)*, Mesquite Software Inc., 1998.

[9] W.J. Dally, B. Towles, *Principles and Practices of Interconnection Networks*, 1st edition, Morgan Kaufmann Publications, 2004.

[10] D.G. Feitelson, P.F. Corbett, S.J. Baylor, and Y. Hsu. “Parallel I/O Subsystems in Massively Parallel Supercomputers”, *IEEE Parallel and Distributed Technology*, vol. 3, pp. 33-47, Sept.1995.

[11] J. Ghosh, K.D. Goveas, and J.T. Draper. “Performance Evaluation of Parallel I/O Subsystem for Hypercube Multicomputers”, *Journal of Parallel and Distributed Computing*, vol.17, pp. 90-106, Jan.1993.

[12] K. Hwang and J. Ghosh, “Hypernet: A Communication-Efficient Architecture for Constructing Massively Parallel Computers, “*IEEE Transaction on Computing*, vol. C-36, pp. 1450-1466, Dec. 1987.

[13] B.A. Izadi, F Ozguner, “Enhanced Cluster k-Ary n-Cube,A Fault-Tolerant Multiprocessor”, *IEEE Transactions On Computers*, vol. 52, no. 11, Nov. 2003.

[14] R. Jain, J. Werth, J.C. Browne. *Input/Ouput in Parallel and Distributed Computer Systems*, Kluwer Academic Publishers, 1996.

[15] E.J. Kim, K. H. Yum, C. R. Das, M. Yousif and J. Duato, "Performance Enhancement Techniques for InfiniBandTM Architecture," *Proc. of the Ninth International Symposium on High-Performance Computer Architecture (HPCA-9)*, pp. 253-262, Feb. 2003.

[16] C.Y. Lee, "Some Properties of Nonbinary Error-correcting Codes”, *IEEE Trans. Information Theory*, vol. 4, no. 2, pp. 77-82, June 1958.

[17] M. Livingston, Q.F. Stout. “Perfect Dominating Sets”, *Congressus Numerantum,* vol. 79, pp. 187-203, 1990.

[18] M.D. Noakes, D.A. Walach, W.J. Dally, “The J-Machine Multicomputer: an architectural evaluation”, *20th International Symposium on Computer Architecture*, pp. 224–235, May 1993.

[19] P. Ramanathan, S. Chalasani, “Resource Placement with Multiple Adjacency Constraints in k-ary n-cubes”, *IEEE Transaction Parallel Distributed Systems,* vol.6, no.5, pp. 511-519, May 1995.

[20] A.L.N Reddy, P. Banerjee, “Design, Analysis, and Simulation of I/O Architectures for Hypercube Multiprocessors**,** *IEEE Trans. Parallel Distributed Systems*, vol.1, no.2, pp. 140-151, April 1990.

[21] S. Scott, G. Thorson, “The Cray T3E Network: Adaptive Routing in High Performance 3D torus”, *Proc. of Symposium on High Performance Interconnects (HOT Interconnects IV)*, pp.147-156, August 1996.

[22] The BlueGene/L Team. “An Overview of the BlueGene/L Supercomputer”, *SC 2000 High Performance Networking and Computing*, pp. 1-22, Nov.2002.

[23] K. H. Yum, E. J. Kim, and C. R. Das, " QoS Provisioning in Clusters: An Investigation of Router and NIC Design," *Proc. of the 28th Annual International Symposium on Computer Architecture (ISCA 2001),* pp.120-129, June 2001.

VITA

CONTACT DETAILS	**Babatunde Azeez** Department of Computer Science, Texas A&M University College Station, TX-77843-3112. tuniks@tamu.edu	
EDUCATION	**M.S.**, **Computer Engineering** Texas A&M University (TAMU) College Station, TX	**December 2005**
	B.Sc., **Computer Engineering** Obafemi Awolowo University Ile-Ife, Nigeria	**July 2002**