ROMM Routing on Mesh and Torus Networks

ROMM Routing on Mesh and Torus Networks

Ted Nesson
S. Lennart Johnsson

TR-08-95

May 1995

Parallel Computing Research Group

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

To appear in *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, July 16-18, 1995, Santa Barbara, California.

ROMM Routing on Mesh and Torus Networks

Ted Nesson and S. Lennart Johnsson
Aiken Computation Laboratory
Harvard University
Cambridge, MA 02138
{nesson,johnsson}@das.harvard.edu

Abstract

ROMM is a class of Randomized, Oblivious, Multi–phase, Minimal routing algorithms. ROMM routing offers a potential for improved performance compared to both fully randomized algorithms and deterministic oblivious algorithms, under both light and heavy loads. ROMM routing also offers close to best case performance for many common routing problems. In previous work, these claims were supported by extensive simulations on binary cube networks [30, 31]. Here we present analytical and empirical results for ROMM routing on wormhole routed mesh and torus networks. Our simulations show that ROMM algorithms can perform several representative routing tasks 1.5 to 3 times faster than fully randomized algorithms, for medium–sized networks. Furthermore, ROMM algorithms are always competitive with deterministic, oblivious routing, and in some cases, up to 2 times faster.

1 Introduction

Efficient data motion has been critical in high performance computing as long as computers have been in existence. Massively parallel computers use a sparse interconnection network between processing nodes with local memories. Minimizing the potential for high congestion of communication links is an important goal in the design of routing algorithms and networks. Moreover, the packaging technology introduces constraints on the data motion bandwidth at module boundaries, such as chips and boards.

These constraints suggest that preserving locality of reference with careful data allocation and minimizing network load by using minimal algorithms are desirable objectives. However, it is important to note that minimal algorithms and locality preserving data mappings do not always guarantee the best routing times. In fact, for certain problems, non–minimal routing algorithms and data allocations which do not preserve locality can utilize more of the available network bandwidth and cause less communication link congestion, without the corresponding increase in demand for network bandwidth.

In this paper, we report on a class of minimal algorithms designed to keep the risk of severe congestion low for arbitrary routing patterns, yet to behave like optimal algorithms for some important representative routing patterns. ROMM (Randomized, Oblivious, Multi–phase, Minimal) routing was first presented in [30, 31]. In the earlier work, we examined the behavior of ROMM algorithms on n–dimensional binary cube networks using store–and–forward routing [44]. In this work, we study the behavior of ROMM algorithms on mesh and torus networks using wormhole routing [10].

The outline of this paper is as follows. Section 2 discusses related work. Section 3 provides basic definitions and terminology. Section 4 presents ROMM routing for mesh and torus networks. Section 5 analyzes the behavior of ROMM routing for some representative problems. Section 6 provides simulation results which show that ROMM routing, for these representative problems, performs better than fully randomized routing and close to best case routing.

2 Related Work

Message routing algorithms can be classified as *oblivious*, *adaptive*, or *customized*. In oblivious algorithms, the paths taken by messages depend only on the source and destination of the messages. In adaptive algorithms, the path taken is influenced by other messages, usually in an attempt to minimize the total routing time. In customized algorithms, message routing is based on a global knowledge of the problem. In a *randomized* algorithm, the path selection, the scheduling of messages, or both, use randomization. In a *deterministic* algorithm, randomization is not used. A routing algorithm is *minimal* if all messages follow shortest paths from source to destination. A good survey of message routing algorithms is provided in [25].

Any viable routing algorithm must ensure freedom from deadlock and livelock. The minimum buffer capacity in the nodes and the buffer management required to achieve these properties depend on whether *wormhole* [10], *virtual cut–through* [22], or *store–and–forward* [44] routing is used. The requirements for store–and–forward and virtual cut–through routing have been studied extensively [6, 7, 16, 29]. For concurrent use of all channels of a node, wormhole routing requires a buffer for each virtual channel, which is a more stringent requirement than what is required for virtual cut–through and store–and–forward routing. For these routing techniques deadlock and livelock can be guaranteed even if buffers are shared between channels. The atomic unit being routed is however typically considerably smaller in wormhole routing than in the other two techniques, and the total buffer space required for deadlock and livelock freedom is in most cases significantly less than for the other two techniques. Hence, a great deal of effort has been devoted in recent years to develop both analytical models as well as efficient implementations of wormhole routing. For instance, wormhole technology, designed originally for massively parallel computers, is currently also being applied to high–speed local area networks [3]. A survey of wormhole routing can be found in [33].

The Caltech Mosaic [42], CMU iWarp [5], Cray T3D [35], Intel Paragon, MIT Alewife [2], MIT J–Machine [34], and Stanford DASH [27] all use deterministic, oblivious routing. Such routing is relatively easy to implement and usually performs well, though performance may decay rapidly under heavy load. Also, these algorithms have poor worst case behavior[1]. For *any* N–node, degree d network, there exist permutations for which any deterministic, oblivious routing strategy requires time $\Omega(\sqrt{N}/d)$ [21].

Valiant and Brebner proposed two–phase randomized routing in which a random node is used as an intermediate destination [48, 49]. The algorithm can route *any* permutation on the n–dimensional binary cube in $O(n)$ steps with high probability. Tsantilas showed that the constant factor is approximately two [47]. The disadvantages of the algorithm are that all data locality is lost and buffers of size $O(n)$ are required in each node. Data locality is of particular interest in inhomogeneous networks. For instance, non–uniform channel widths may be introduced by packaging constraints. Variants which use fixed size buffers have been proposed [28]. In the Connection Machine CM–5 [45], randomization is employed in the first routing phase in the fat–tree network [26]. An improvement of the routing time by approximately a factor of two can be achieved by randomizing the allocation of the address space, as suggested by Ranade et al. [39, 40]. This approach is being implemented by Abolhassan et al. [1]. Fixed size buffers suffice, but data locality is still lost. Randomized allocation is an option for some gather/scatter related operations on Connection Machine systems [46].

Minimal and non–minimal deterministic, adaptive, routing algorithms, have been proposed by many researchers [9, 11, 12, 14, 15, 23, 32, 38, 37]. A survey of adaptive routing for binary cubes can be found in [13]. Deterministic, adaptive routing is used on the Connection Machine models CM–1, CM–2, and CM–200 [17]. The Chaos router [24] is a randomized, adaptive router, which allows messages to follow random shortest paths from source to destination, but misroutes messages when congestion occurs. A chip implementation of the Chaos router has been made [4]. However, questions about the effectiveness of adaptive routing, in general, have been raised [36].

Customized algorithms exist for many routing problems, such as frequently occurring permutations and gather/scatter operations, on a variety of networks. Advanced compilers may detect instances of these problems and generate the appropriate machine code, or call functions in a library such as the Connection Machine Scientific Software Library [46].

Unlike all of the routing algorithms described above, ROMM routing is randomized, oblivious, and minimal. Furthermore, ROMM routing is suitable for store–and–forward, virtual cut–through, and wormhole routed networks, unlike some of the adaptive techniques described above, which are not applicable to wormhole networks. In this paper, we examine ROMM routing on wormhole routed mesh and torus networks.

[1]For example, dimension–order routing performs poorly when routing transpose permutations on 2–dimensional meshes.

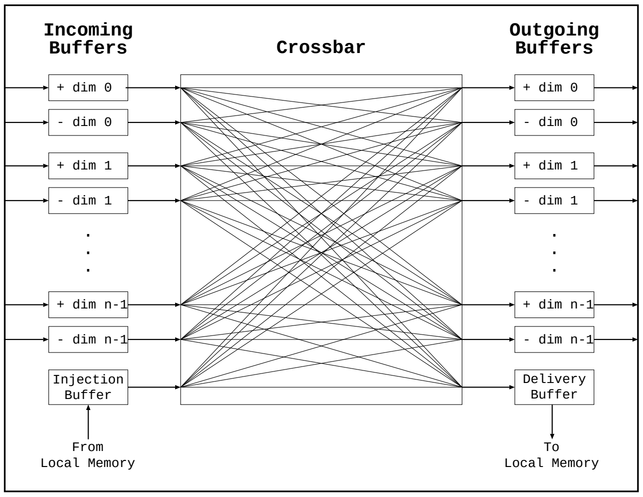

Figure 1: High–level architecture of a network interface for a generic processor node in an n–dimensional network. Except for the links to and from the processor node's local memory, each buffer is associated with a link in one of the n dimensions, in an upward or downward direction.

3 Preliminaries

Massively parallel computers are constructed from processor nodes linked together by an interconnection network. A *processor node* consists of three major elements: a processing element, a local memory, and a network interface. An *interconnection network* is a graph in which the vertices are the processor nodes and the edges are the communication links[2]. Figure 1 shows the main features of the network interface for a generic processor node.

Processor nodes communicate with each other by sending messages. The atomic unit being communicated in wormhole routing is a *flit* (flow control digits). Messages are divided into flits, with the flits from a message (the "worm") possibly spread out across multiple nodes during routing. But only the header flit contains routing information. Hence, all flits from the same message must follow the same path, and flits from different messages cannot be interleaved. In order to avoid excessively long worms, messages may be divided into packets, which in turn are divided into flits. Each packet forms a worm, with different worms possibly following different routing paths.

Definition 1 (n-dimensional Mesh Network) This network is defined by a set of extents, $K_0, K_1, \ldots, K_{n-1}$ and has $N = K_0 \times K_1 \times \ldots \times K_{n-1}$ nodes. Nodes are labeled by n–tuples, with values corresponding to the node's offset in each dimension with respect to node $(0, 0, \ldots, 0)$. Node $X = (x_0, x_1, \ldots, x_{n-1})$ is a valid node if $0 \leq x_i < K_i$, for $0 \leq i < n$. In dimension i, $0 \leq i < n$, the connectivity for

[2]Throughout this paper, we will use the terms *link* and *edge* interchangeably.

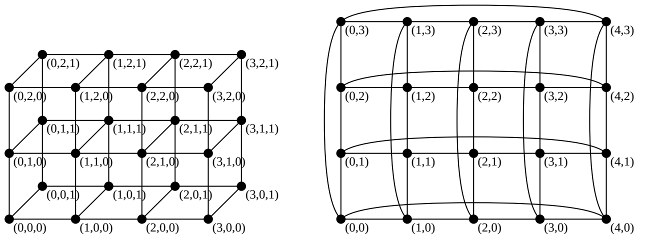

Figure 2: A $4 \times 3 \times 2$ mesh network and a 4×3 torus network.

node X is:

$$X \to \begin{cases} (x_0, x_1, \ldots, x_i + 1, \ldots, x_{n-2}, x_{n-1}) & \text{if } x_i < K_i - 1 \\ (x_0, x_1, \ldots, x_i - 1, \ldots, x_{n-2}, x_{n-1}) & \text{if } x_i > 0 \end{cases}$$

Definition 2 (n-dimensional Torus Network) This network is identical to the mesh, except for the connectivity. Unlike the mesh, the torus provides wrap–around links, connecting every node to $2n$ neighbors. In dimension i, $0 \le i < n$, the connectivity for node X is:

$$X \to \begin{cases} (x_0, x_1, \ldots, (x_i + 1) \bmod K_i, \ldots, x_{n-2}, x_{n-1}) \\ (x_0, x_1, \ldots, (x_i - 1) \bmod K_i, \ldots, x_{n-2}, x_{n-1}) \end{cases}$$

Figure 2 provides examples of mesh and torus networks. Note that meshes for which $K_i = 2$, for $0 \le i < n$, are also known as binary cubes or hypercubes. In a mesh, the distance (in links) between node $X = (x_0, x_1, \ldots, x_{n-1})$ and $Y = (y_0, y_1, \ldots, y_{n-1})$ is $\sum_{i=0}^{n-1} |y_i - x_i|$. In a torus, the distance is $\sum_{i=0}^{n-1} \min(|y_i - x_i|, K_i - |y_i - x_i|)$. Because distance does not capture many of the characteristics of the path of a message, we introduce another metric, the *routing set* for a message.

Definition 3 (Routing Set) The routing set for a message is an n-tuple of the displacements in each dimension of a mesh or torus required for a message to reach its destination. Let $\mathcal{R} = (\mathcal{R}_0, \mathcal{R}_1, \ldots, \mathcal{R}_{n-1})$ be the routing set for a message originating at node $X = (x_0, x_1, \ldots, x_{n-1})$ destined for node $Y = (y_0, y_1, \ldots, y_{n-1})$. For a mesh, $\mathcal{R}_i = (y_i - x_i)$. For a torus, the computation of $\mathcal{R}_i$ is more complex, due to the wrap–around links:

$$\mathcal{R}_i = \begin{cases} (y_i - x_i) & \text{if } |y_i - x_i| \le \lfloor K_i/2 \rfloor \\ (y_i - x_i - K_i) & \text{if } |y_i - x_i| > \lfloor K_i/2 \rfloor \text{ and } y_i > x_i \\ (y_i - x_i + K_i) & \text{if } |y_i - x_i| > \lfloor K_i/2 \rfloor \text{ and } y_i \le x_i \end{cases}$$

The *magnitude* $\|\mathcal{R}\|$ of a routing set is defined as $\|\mathcal{R}\| \overset{\text{def}}{=} \sum_{i=0}^{n-1} |\mathcal{R}_i|$. Note that $\|\mathcal{R}\|$ is also the distance from X to Y.

The *cardinality* $|\mathcal{R}|$ of a routing set is defined as the number of dimensions for which $|\mathcal{R}_i| \ne 0$, $0 \le i < n$.

The most well–known deterministic, oblivious routing algorithm is dimension–order routing (DimOrder). In DimOrder routing messages are moved from source to destination by correcting the displacements in ascending dimension order[3]. On 2–dimensional meshes, it is called X–Y routing; on binary cubes, it is called e–Cube routing [43]. The best known randomized, oblivious routing algorithm is Valiant–Brebner routing (Valiant) [48, 49]. Valiant is a two–phase algorithm which, in the first phase, uses DimOrder to route the message from its source to a random node, and in the second phase, uses DimOrder to route the message from the random node to its destination. The algorithm is obviously non–minimal, but despite its increased demands for network bandwidth compared to minimal algorithms it has been shown to decrease the likelihood of edge–congestion by effectively converting the original routing problem into two random routing problems.

[3] Any fixed order is valid, though ascending order is typically used.

To ensure the correctness of ROMM algorithms, we will employ *virtual channels*:

Definition 4 (Virtual Channels) A virtual channel consists of a unidirectional communications link between adjacent nodes together with an input and output buffer. Virtual channels may share physical communications links, but the buffers are private. Virtual channels can be used to eliminate deadlock [10] and to improve performance [8].

4 ROMM Routing

Although many routing algorithms have been proposed, DimOrder is still used in many parallel systems, primarily because it is easy to implement and usually exhibits good performance. Unfortunately, DimOrder has poor worst case behavior and degrades under heavy loads. In contrast, Valiant routing prevents worst case behavior by employing randomization to convert a routing problem into two random, average case problems. This almost always results in non–minimal paths and higher demand for network bandwidth. But, the use of additional network edges through non–minimal routing may well outweigh the increased demand and hence reduce the total routing time, as for instance Gray–to–binary code conversion on a binary cube [18, 20]. Unfortunately, for many problems, Valiant routing results in longer routing times.

Randomized, Oblivious, Multi–phase, Minimal (ROMM) routing offers a compromise. ROMM algorithms are minimal, like DimOrder, and randomized, like Valiant. Furthermore, ROMM algorithms are oblivious, which eliminates the complexity often associated with adaptive approaches. Because ROMM algorithms use only minimal paths, they cannot offer as much randomization for many problems as a fully randomized algorithm like Valiant. However, the lower degree of randomness is compensated by only using minimal paths.

The degree of randomness in ROMM algorithms is controlled through the number of randomly selected nodes along a minimal path a message is required to pass through (the precise details will be described shortly). A p–Phase ROMM algorithm has $p - 1$ randomly selected nodes $Z_1, Z_2, \ldots, Z_{p-1}$ between source and destination[4]. The $p-1$ intermediate nodes are selected individually for each message. During each phase, a subset of the displacements in the routing set of a message are decreased in magnitude.

During the i^{th} phase ($0 \le i < p$), the message is routed from node Z_i to node Z_{i+1} using DimOrder routing. $\mathcal{R}^i = (\mathcal{R}_0^i, \mathcal{R}_1^i, \ldots, \mathcal{R}_{n-1}^i)$ denotes the routing set for the i^{th} phase (i.e., routing from Z_i to Z_{i+1}) with $\mathcal{R}_j = \sum_{i=0}^{p-1} \mathcal{R}_j^i$, for $0 \le j < n$ ($\mathcal{R}_j$ is the displacement from source to destination in the j^{th} dimension. Figure 3 shows a possible 3–Phase path through a 2–dimensional mesh.

[4] We will use Z_0 and Z_p as shorthand for the source and destination nodes throughout this paper.

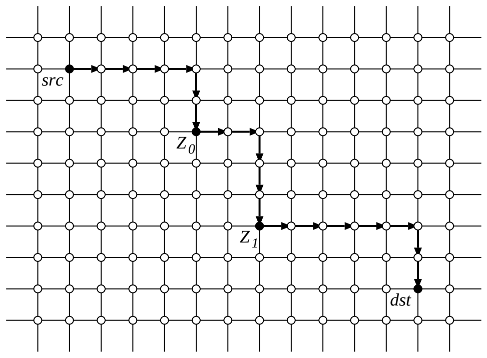

Figure 3: A possible path from *src* to *dst* when using 3–PHASE ROMM routing on a 2–dimensional mesh network.

In the next two sections, we prove some properties of ROMM routing and discuss several randomization issues.

4.1 Deadlock, Livelock, and Starvation

Theorem 1 p virtual channels per physical link are necessary and sufficient to ensure deadlock freedom for p–PHASE ROMM routing on a wormhole routed mesh network.

Proof (Informal) To demonstrate deadlock freedom for an oblivious wormhole algorithm, we must show that the buffer dependency graph is acyclic [10]. It is straightforward to show that DIMORDER routing on an n–dimensional mesh is deadlock free, using only one virtual channel. We have p virtual channels for p phases, so we simply require messages in phase i to use virtual channel i. Since this eliminates resource dependencies across phases and DIMORDER is used in each phase, it follows that p–PHASE is deadlock–free. If fewer than p virtual channels are available, deadlocked configurations can be constructed easily. ∎

Remark The proof of Theorem 1 implies that the message header contains phase information, in addition to any other routing information required for the algorithm.

Theorem 2 $2p$ virtual channels per physical link are necessary and sufficient to ensure deadlock freedom for p–PHASE ROMM routing on a wormhole routed torus network.

The proof of Theorem 2 is similar to the proof of Theorem 2. However, DIMORDER routing is not deadlock–free on a torus network using one virtual channel, due to the wrap–around links. To make DIMORDER deadlock–free, one must use two virtual channels per physical link, partitioning the physical channels into two groups to break the cyclic dependency. Then, p–PHASE ROMM can be made deadlock free using the same approach as before, and it follows that $2p$ virtual channels are necessary and sufficient for deadlock freedom.

Remark p (or $2p$) virtual channels also suffice to make p–PHASE store–and–forward and virtual cut–through ROMM routing deadlock–free on a mesh (or torus) network. This result can be proved with arguments similar to the ones above [10].

Theorem 3 Deadlock–free p–PHASE ROMM routing is livelock–free on mesh and torus networks.

Proof Since the intermediate destinations for a message fall along a minimal path from source to destination and DIMORDER routing is used in each phase, p–PHASE ROMM routing is minimal. Theorems 1 and 2 guarantee that a message moves, and minimality guarantees that it moves closer to its destination. Hence, there is no livelock. ∎

To prevent starvation, we require a mechanism to ensure timely access to shared resources for all messages. The resources of concern are the physical links, and the outgoing buffers (including the delivery buffers).

Theorem 4 p–PHASE ROMM routing can be made starvation–free on mesh and torus networks using a round–robin resource allocation algorithm at each node.

Proof (Informal) Servicing the virtual channel buffers associated with a physical link in cyclic order ensures fairness among multiple virtual channels sharing a physical link. In mapping incoming and injection buffers to outgoing buffers, we can guarantee fair service to every message by cycling through the outgoing buffers and maintaining for each outgoing buffer, a cyclic ordering on the incoming buffers with the injection buffers included, starting after the last incoming (or injection) buffer that used the outgoing buffer. This scheme ensures service for all flits eventually, whether they are waiting to move from an incoming buffer to an outgoing (or delivery) buffer or waiting for transmission on a physical link. ∎

4.2 Randomization Issues

The amount of randomization introduced by a ROMM algorithm is directly related to p, the number of phases. If, for a given message, $p = \text{DISTANCE}(src, dst) - 1$, the intermediate destinations can be used to define *any* minimal path from *src* to *dst*. However, as p increases, additional resources are required. One cost is the memory and logic associated with p virtual channels. Another is the cost of managing $p - 1$ intermediate destinations. These may be carried in the message (which increases the header size) or computed dynamically at the start of a phase (which increases routing logic). Hence, there is a trade off between the degree of randomization and resource requirements. In this section, we explore these issues and propose a methodology designed to minimize the resource requirements and maximize performance.

In previous work [30, 31], we examined several ways to randomly select the intermediate destinations for store–and–forward ROMM routing on a binary cube. Analytical and empirical results suggested that it is desirable to make the magnitudes of the routing sets for each phase about the same, maximizing the number of different paths available to a message. However, there are several issues involved in translating this observation to a mesh or torus network.

One important issue is the bias with respect to edge–load introduced by using DIMORDER routing in each phase. On a mesh or torus network with randomly selected intermediate nodes and DIMORDER routing used in each phase, most messages will depart the source node in dimension 0 (see Figure 3). In fact, all but $\frac{N}{K_0}$ of the N possible destinations begin routing in dimension 0, since these are the only nodes that do not include dimension 0. Since $K_0 = 2$ for a binary cube, this bias is less severe on the binary cube. The heavy use of dimension 0 for the first routing step creates contention when messages in the injection buffers attempt to enter the network. A similar problem occurs in dimension $n-1$ for messages in the network trying to enter the delivery buffers.

The above problem is of major concern only if routing delays are caused primarily by contention at injection or delivery. Our simulations indeed suggest that this form of contention is the primary problem in mesh and torus wormhole routing. It is our conjecture that with high source and destination node contention, there is not a sufficiently large number of messages in the network to cause network induced contention (contention for network resources caused by interference among messages with different sources and destinations). The random selection of intermediate nodes described below, at injection attempts to distribute messages evenly among the dimensions that must be routed in order to move from source to destination.

For wormhole routing, a full random selection of the Z_i's may cause worms to become "tangled" and reduce the chances that worms may follow each other nicely in "marching order". A negative impact on performance of many turns have been observed by others [41] as well as in our own simulations. The randomization scheme described below for ROMM routing yields a minimal number of turns when the number of phases is at most equal to the number of dimensions to be routed.

For p–PHASE ROMM routing on an n–dimensional mesh or torus, intermediate nodes are chosen with the intent to balance the edge–load at the source and destination nodes, and also with the intent to minimize the number of turns for a given number of phases p. For $p \leq d$, where d is the maximum cardinality of *any* routing set, we accomplish this task by restricting intermediate nodes to be randomly selected from the set of "corner" nodes of the minimal submesh defined by the source and destination nodes. The intermediate corner nodes are selected by evenly and randomly distributing the d elements of the routing set of a message among the p phases. Thus, in fact, only corner nodes that evenly (assuming p divides d) divide the number of submesh edges between source and destination are allowed as candidate intermediate nodes. All corner nodes are eligible when $p = d$. For example, when using 3–PHASE ROMM routing on a 6–dimensional mesh or torus, a possible selection of routing sets for the three phases is

$$\begin{aligned}\mathcal{R}^0 &= (\ 0,\ \ 0, \mathcal{R}_2,\ \ 0, \mathcal{R}_4,\ \ 0)\\ \mathcal{R}^1 &= (\ 0, \mathcal{R}_1,\ \ 0,\ \ 0,\ \ 0, \mathcal{R}_5)\\ \mathcal{R}^2 &= (\mathcal{R}_0,\ \ 0,\ \ 0, \mathcal{R}_3,\ \ 0,\ \ 0).\end{aligned}$$

If the number of phases exceeds the number of dimensions being routed, then some of the $\mathcal{R}_i$'s are split. In this case, nodes other than corner nodes are also allowed as intermediate nodes, and the number of turns between source and destination is no longer minimal. Note however, that the additional intermediate nodes allowed when $p > d$ are contained within the minimal submesh defined by the source and destination nodes.

For example, when using 4–PHASE on a 3–dimensional mesh or torus, the routing sets could be

$$\begin{aligned}\mathcal{R}^0 &= (\ 0,\ \ 0, \mathcal{R}_2')\\ \mathcal{R}^1 &= (\ 0, \mathcal{R}_1,\ \ 0)\\ \mathcal{R}^2 &= (\mathcal{R}_0,\ \ 0,\ \ 0)\\ \mathcal{R}^3 &= (\ 0,\ \ 0, \mathcal{R}_2''),\end{aligned}$$

where $\mathcal{R}_2' + \mathcal{R}_2'' = \mathcal{R}_2$.

Finally, we specify a few other parameters of our randomization scheme. It is important to note that the ROMM randomization scheme is applied to every message independently. Hence, two messages with the same source and destination will not necessarily follow the same path. Moreover, whether or not one or more $\mathcal{R}_i$'s are split in our analysis and simulation results is determined by d, the maximum cardinality of the routing set for any message. Hence, even though the number of phases may exceed the cardinality of the routing set for a particular message, no splitting of the elements of the routing set is made for that message (as long as $p \leq d$). Messages for which the cardinality of their routing sets is less than p, simply experiences some null phases in which the message does not move. Such null phases do not affect the correctness of ROMM routing. Moreover, our splitting scheme splits a dimension at most $\lceil \frac{p}{d} \rceil$ times. Dimensions to be split are selected such that no $\mathcal{R}_i^j = 0$ is generated as long as there exists one dimension to be split within the constraint that no dimension is split more than $\lceil \frac{p}{d} \rceil$ times.

It is interesting to note that for a mesh or torus network 2–PHASE and VALIANT both have two phases, but 2–PHASE has a highly constrained set of intermediate nodes compared VALIANT since it only allows corner nodes at half distance measured in dimensions on a minimal submesh as intermediate nodes, compared to *any* node for VALIANT.

5 Performance Analysis

To assess the performance of ROMM algorithms on mesh and torus networks, we have selected four representative routing tasks. These are bit–complement permutations, transpose permutations, single–random problems (same random destination for every message originating from a given node), and full–random problems (individual random destinations for every message).

Bit–complement is an operation that may occur in distributed matrix multiplication [19], as well as when performing vector reversals. Bit–complement is also interesting because it is highly symmetric and places a substantial load on the links in the network. Matrix transpositions occur often in scientific and engineering programming. Although the total link load for routing this permutation is lower than that of bit–complement, the loads are not well balanced when DIMORDER routing is used. Figure 4 provides examples of bit–complement and transpose permutations on mesh networks[5]. Single–random tasks attempt to model general routing of large blocks of information by selecting, for each node, a random destination and routing all messages from that node to the selected destination. Full–random destinations model what many other researchers have done by selecting, for each message, a random destination.

For the purposes of our analyses and simulations, we assume that every node injects L messages (except, of course, for any node whose destination is itself). We refer to L as the network load. Furthermore, we allow for the possibility that sources and destinations may exhibit locality and that a routing task may require routing in only d of the n dimensions. When $d < n$, minimal algorithms can have a significant advantage over non–minimal ones.

[5]For transpose permutations in more than 2–dimensions, we assume $K_i = K_{i+n/2}$, for $0 \leq i < n/2$.

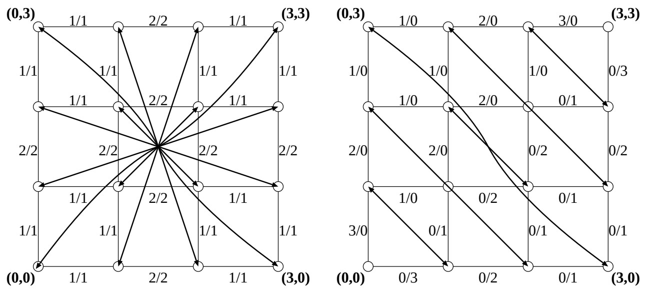

Figure 4: Bit–complement (left) and transpose (right) routing patterns on 4×4 and mesh networks. The arrows indicate the permutation. The numbers indicate the total number of messages traversing the link in each direction ($+X/-X$ or $+Y/-Y$), using DimOrder routing, with one message injected per node.

To better understand the characteristics of these routing tasks, we have analyzed the average and maximum path lengths, the number of paths available, and the maximum edge–loads for the different algorithms[6].

Table 1 shows the average and maximum possible path lengths on meshes. Average path length multiplied by the load and the message size measured in flits is the total network bandwidth required, in flits. For minimal algorithms, bit–complement has the highest demand, while the others are equal. Compared to any minimal algorithm, Valiant doubles the expected bandwidth required for all of these tasks when $d = n$, except for bit–complement for which the expected bandwidth required for Valiant is the same as for the other algorithms. When $d < n$, then the bandwidth requirements for Valiant are more than double that of the other algorithms for all problems, except bit–complement for which Valiant still has higher routing requirements than the other algorithms. Note though that the bandwidth available for Valiant is also higher than for the other algorithms, especially when $d < n$.

Under light load, the maximum possible path length can determine the performance. Valiant may increase the maximum path length substantially for all of the tasks, except bit–complement when $d = n$.

In Table 2, we determine the number of different paths available to a message with routing set $\mathcal{R}$. DimOrder routing provides only one path, while p–Phase ROMM and Valiant provide substantially more paths. Whether p–Phase or Valiant provides the most paths depends upon the mesh shape (i.e., the K_i's), d, the number of dimensions to be routed, n, the dimensions of the mesh, and p. Note that the number of *edge–disjoint* paths is a more interesting measure than the number of different paths, but edge–disjoint paths are considerably harder to determine. As an example, the number of paths accessible for $d = n$ in a 4096 node square mesh, cubic mesh, and binary cube are given in Table 3.

Finally, Table 4 describes the expected maximum edge–loads (in messages) for the different problems and algorithms, on mesh networks. This is a useful performance metric, because it is a lower bound on routing time, albeit not a tight one. The maximum expected edge–load

[6] These analytical results are *expected* values whenever the task or algorithm is randomized.

Routing Task	**Average Path Length**	
	Minimal Routing	Valiant Routing
Bit–Comp.	$\sum_{i=0}^{d-1} \frac{K_i}{2}$	$2 \cdot \sum_{i=0}^{n-1} \frac{{K_i}^2 - 1}{3K_i}$
Trans.	$\sum_{i=0}^{d-1} \frac{{K_i}^2 - 1}{3K_i}$	$2 \cdot \sum_{i=0}^{n-1} \frac{{K_i}^2 - 1}{3K_i}$
Single–Rand.	$\sum_{i=0}^{d-1} \frac{{K_i}^2 - 1}{3K_i}$	$2 \cdot \sum_{i=0}^{n-1} \frac{{K_i}^2 - 1}{3K_i}$
Full–Rand.	$\sum_{i=0}^{d-1} \frac{{K_i}^2 - 1}{3K_i}$	$2 \cdot \sum_{i=0}^{n-1} \frac{{K_i}^2 - 1}{3K_i}$

Routing Task	**Maximum Path Length**	
	Minimal Routing	Valiant Routing
Bit–Comp.	$\sum_{i=0}^{d-1} (K_i - 1)$	$\sum_{i=0}^{d-1} (K_i - 1) + \sum_{i=d}^{n-1} 2(K_i - 1)$
Trans.	$\sum_{i=0}^{d-1} (K_i - 1)$	$\sum_{i=0}^{d-1} (2K_i - 3) + \sum_{i=d}^{n-1} 2(K_i - 1)$
Single–Rand.	$\sum_{i=0}^{d-1} (K_i - 1)$	$\sum_{i=0}^{d-1} 2(K_i - 1) + \sum_{i=d}^{n-1} 2(K_i - 1)$
Full–Rand.	$\sum_{i=0}^{d-1} (K_i - 1)$	$\sum_{i=0}^{d-1} 2(K_i - 1) + \sum_{i=d}^{n-1} 2(K_i - 1)$

Table 1: Average path length and maximum possible path length on mesh networks using minimal or Valiant routing, for bit–complement, transpose, single–random, and full–random problems.

Algorithm	**Number of Paths**
DimOrder	1
p–Phase $(1 < p \leq d)$	$\frac{d!}{\left(\lfloor \frac{d}{p} \rfloor !\right)^{p - d \bmod p} \cdot \left(\lceil \frac{d}{p} \rceil !\right)^{d \bmod p}}$
p–Phase $(d < p < 2d)$	$\geq \sum_{j=0}^{p-d} \left((d\text{-}j)! \times j! \times \sum_{s_1 \neq \ldots \neq s_j} (\mathcal{R}_{s_1}\text{-}1) \cdot \ldots \cdot (\mathcal{R}_{s_j}\text{-}1) \right)$
Valiant	$\prod_{i=0}^{n} K_i - \sum_{i=0}^{d} \lvert \mathcal{R}_i \rvert$

Table 2: Number of paths available to a message with routing set $\mathcal{R}$ using DimOrder, p–Phase ($p \leq d$), and Valiant. We assume $\mathcal{R}_i \neq 0$, for $0 \leq i < d$.

Mesh Shape	2–Phase	4–Phase	6–Phase	Valiant
$n = 2, K_i = 64$	2	$\geq 4{,}097$	—	3,970
$n = 3, K_i = 16$	3	≥ 96	$\geq 83{,}766$	4,051
$n = 12, K_i = 2$	924	369,600	7,484,400	4,084

Table 3: The number of paths available to a message for the various routing algorithms on several 4096 processor networks, for a longest path in each network.

for bit–complement is unaffected by the choice of algorithm. Random problems are unaffected by the choice of oblivious minimal algorithm as expected [39], but the maximum expected edge–load is doubled if Valiant is used. Finally, we note that transpose permutations have a high maximum edge–load of about $\sqrt{N}$ when DimOrder is used, while for routing in d dimensions d–Phase ROMM and Valiant have a lower expected maximum edge–load. For example, when routing transpose permutations on a 16×16 mesh, DimOrder, 2–Phase, and Valiant have maximum edge–loads of $15L$, $7.5L$, and $7.5L$ messages, respectively. For a $4 \times 4 \times 4 \times 4$ mesh, DimOrder, 4–Phase, and Valiant have expected maximum edge–loads of $12L$, $3L$, and $1.5L$ messages, respectively. Note that the formulas in the table indicate that Valiant has the smallest expected maximum edge–load when there are more than 2 dimensions and the extents of the mesh are large. Also, for a fixed number of nodes, d–Phase will have the lowest expected maximum edge–load when d is large (and the mesh becomes hypercube like).

6 Simulation Results

Using a simulator, we have collected data for 16×16 mesh and torus networks, and for a $4 \times 4 \times 4$ torus network for all of the routing tasks. We have also collected data for transpose and single–random routing on a 32×32 mesh. For our simulations, we assume that each displacement $\mathcal{R}_i^j$ consumes a flit. Hence, the message header for p–Phase requires p flits. The header size can be reduced at the expense of more clever encodings, for instance by only including nonzero $\mathcal{R}_i^j$'s for a message in its header, or by (dynamically) eliminating header fields for displacements as they are achieved. Hence, in our simulations, all of the ROMM algorithms are being charged more buffer space and time for the header than necessary, biasing these results *against* ROMM routing.

The network resources for the algorithms we simulated are summarized for the mesh in Table 5. The number of virtual channels for the torus networks are twice the number for the mesh networks to prevent deadlock. For fairness, we equalized the number of virtual channels for DimOrder, 2–Phase and Valiant by using two lanes [8] for DimOrder, while a single lane was used for 2–Phase and Valiant. 4–Phase and 3–Phase were given unfair number of buffers in that they were given four (three) buffers for each physical link (compared to two for all the other algorithms) as required for deadlock freedom. Moreover, in order to accommodate the entire message header, the buffer depth was four for 4–Phase (three for 3–Phase) compared to two for the other algorithms. Hence, in all, 4–Phase was given four times the buffer space of the other algorithms with which it was compared. The torus network was given twice the buffer capacity of the corresponding mesh simulation, since twice the number of virtual channels per physical link are

Net.	**Alg.**	**VC's Req.**	**VC's Used**	**In Buf**	**Buf/ Node**	**Hdr. Size**
16×16 mesh	DimOrder	1	2	2	24	1
	2–Phase	2	2	2	24	2
	4–Phase	4	4	4	80	4
	Valiant	2	2	2	24	2
16×16 torus	DimOrder	2	4	2	48	1
	2–Phase	4	4	2	48	2
	4–Phase	8	8	4	160	4
	Valiant	4	4	2	48	2
$4 \times 4 \times 4$ torus	DimOrder	4	4	3	96	1
	2–Phase	4	4	3	96	2
	3–Phase	6	6	3	144	3
	Valiant	4	4	3	96	2

Data Size is always 15 and **Out Buf** depth is always 1.

Table 5: Simulation configuration parameters, including number of virtual channels required and used, flit depths of incoming and outgoing buffers, total flit storage required per node, and flit size of message header and payload.

required for deadlock freedom. Round–robin scheduling was used for buffer and switch scheduling, as described in Theorem 4. Two injection and two delivery lanes were used. The network links and the links across the router crossbar can transfer one flit per simulated cycle. All data points reflect averages over 32 runs.

The results of the 2–dimensional simulations are plotted in Figures 5 and 6. The torus performs better than the mesh, as expected, since torus networks offer twice the bisection width of mesh networks.

For bit–complement permutations, the analyses in Section 5 suggest that all of the minimal algorithms require the same bandwidth and have the same expected maximum edge–load. Valiant requires 33% more bandwidth but has the same expected maximum edge–load. The results generally agree with the analyses, except that Valiant is more than 33% slower and 4–Phase is substantially worse than 2–Phase and DimOrder. The main problem is that both Valiant and 4–Phase split displacements across phases which results in driving more messages towards the center of the mesh or torus, the most congested area for this task (see Figure 4). Furthermore, the routes used by these algorithms involve many turns which may cause more contention amongst worms (i.e., "tangling"). Finally, the larger headers account for a small increase in routing time with respect to DimOrder.

For transpose permutations, which have an unbalanced edge–load under DimOrder routing, we observe that 2–Phase is the best algorithm, with performance approximately double that of DimOrder. For injection(ejection) limited routing, 2–Phase is expected to complete in about half the time of DimOrder. For higher dimensional meshes the advantage of d–Phase is expected to be more significant. 4–Phase does slightly better than DimOrder (and worse than 2–Phase) on the mesh, but slightly worse on the torus. The balancing of edge–loads at injection is the same for 2–Phase and 4–Phase. Moreover, 4–Phase tends to drive messages into the minimal submesh defined by the source and destination nodes by allowing extra turns compared to 2–Phase that only use the "bounding" edges of the minimal submesh. Since edge–loads accumulate towards two of the corners in 2–Phase to become $\frac{1}{2}(\sqrt{N} - 1)$ for an N node

Algorithm	**Routing Task**		
	Bit–Complement	Transpose	Single/Full–Random
DimOrder	$L \times \max_{0 \le i < d}(\frac{K_i}{2})$	$L \times \max \left(\begin{array}{c} (K_0 - 1) \cdot \prod_{i=1}^{d/2-1} K_i, \\ (K_{d/2-1} - 1) \cdot \prod_{i=0}^{d/2-2} K_i \end{array} \right)$	$L \times \max_{0 \le i < d}(\frac{K_i}{4})$
p–Phase	$L \times \max_{0 \le i < d}(\frac{K_i}{2})$	$L \times \sum_{j=1}^{d/2} \left[\frac{1}{2j} \times (K-1)^j \times \prod_{i=1}^{j-1} \left(\frac{d-2i}{2j-i} \right) \right]$	$L \times \max_{0 \le i < d}(\frac{K_i}{4})$
Valiant	$L \times \max_{0 \le i < n}(\frac{K_i}{2})$	$L \times \max_{0 \le i < n}(\frac{K_i - 1}{2})$	$L \times \max_{0 \le i < n}(\frac{K_i}{2})$

Table 4: Expected maximum edge loads, in messages, for different routing tasks and algorithms on mesh networks. Note that the p–Phase result for transpose assumes $p = d$ and $K_i = K$, for $0 \le i < d/2 - 1$.

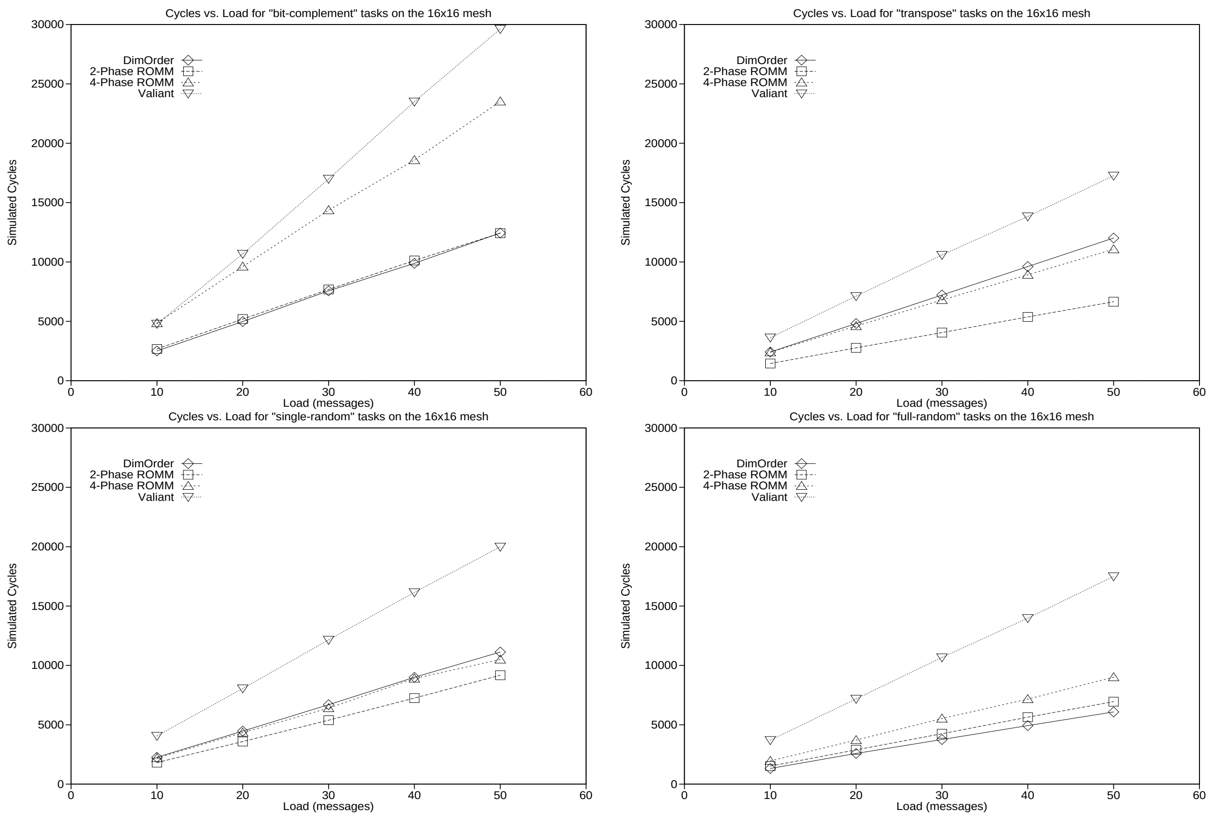

Figure 5: Graphs of simulated cycles vs. load for 16×16 mesh networks.

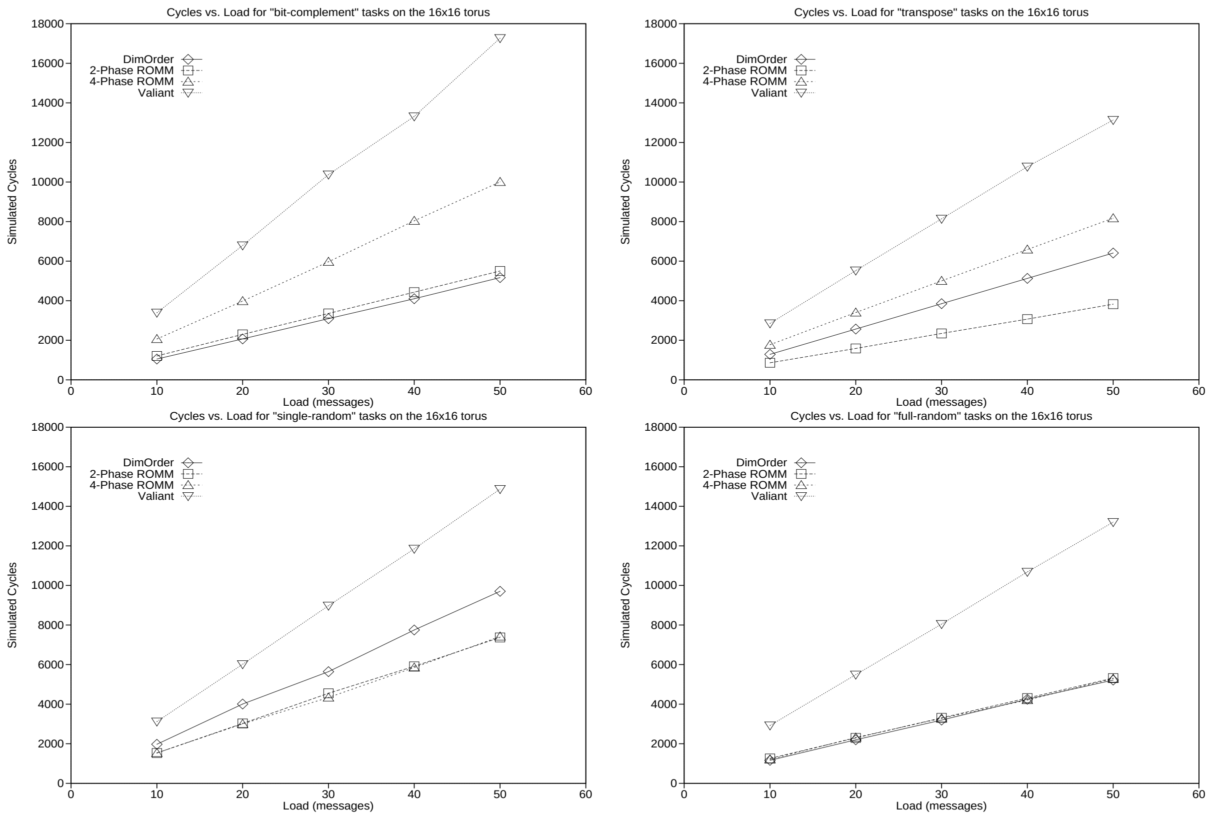

Figure 6: Graphs of simulated cycles vs. load for 16×16 torus networks.

square mesh (see Figure 4), moving messages into the minimal submesh is expected to be advantageous with respect to routing time. However, the increased header size (increasing the message size by about 12% in our simulations) and the increased "tangling" of worms clearly more than offset the expected gains in our simulations, despite the additional buffering resources (four times that of 2–PHASE). For large meshes, the relative merits of 4–PHASE may increase, since the header size will not increase (measured in displacements) and the "corner effect" grows as $\sqrt{N}$. VALIANT does not measurably improve the edge–load at injection compared to DIMORDER. Though VALIANT routing drives messages into the mesh and away from the most congested corners, this expected benefit is clearly more than offset by "tangling" or other congestion effects for the simulated mesh size. As for 4–PHASE, the relative merits of VALIANT are expected to grow with the mesh size. It is worth noting that VALIANT and p–PHASE routing will have substantially lower edge–loads than DIMORDER when routing transpose permutations on higher dimensional networks, and under heavy load may outperform ROMM algorithms on such networks [30, 31].

For single–random tasks, 2–PHASE and 4–PHASE are expected to perform better than DIMORDER and VALIANT assuming edge–load at injection(ejection) is the performance limiting factor. Our simulation results suggest that indeed holds for both mesh and torus networks. 4–PHASE, as for the transpose case, seem handicapped by the "tangling" of worms and increased header size despite its higher buffering capacity. VALIANT is not competitive, since there is no real benefit of randomizing a random problem, and the required bandwidth is doubled.

For full–random tasks, we expect similar performance from all of the minimal algorithms. On the mesh, DIMORDER is the best, with 2–PHASE close behind, presumably due to its larger header. 4–PHASE lags farther behind, for the reasons discussed earlier. VALIANT doubles the required bandwidth, and hence, takes much longer. On the torus, the minimal algorithms are nearly indistinguishable. In this case, both 2–PHASE and 4–PHASE are competitive, presumably due to the shorter distances in a torus and the message dispersal effects discussed earlier.

The results of the 3–dimensional torus simulations are plotted in Figure 7. For bit–complement problems, DIMORDER is best, with 2–PHASE and 3–PHASE lagging behind by about a factor of two, and VALIANT worse by a factor of close to four. The larger message header of the ROMM algorithms penalizes their performance somewhat compared to DIMORDER. But, it is our conjecture that most of the performance differential is due to the fact that the randomization of dimensions disrupts the balanced load of DIMORDER for this problem and, given the very short extents of the mesh in all dimensions, the load imbalance caused by randomization is relatively more significant than in the 16×16 square mesh. In fact, for the $4 \times 4 \times 4$ mesh at most two messages contend for an edge in DIMORDER routing and the relative impact of load–imbalance is very significant. For the 16×16 mesh up to eight messages may contend for an edge in DIMORDER routing and the relative impact of load–imbalance caused by the randomization in ROMM routing is expected to be small. Transpose does not map easily to 3–dimensional networks, so it was not tested. For single–random tasks on the 3–dimensional torus, we ex-

Net.	Algorithm	Task			
		bitc.	trans.	s–rand.	f–rand.
16 × 16 mesh	DimOrder	248	240	223	119
	2–Phase	245	130	184	136
	4–Phase	463	217	212	176
	Valiant	625	340	400	344
16 × 16 torus	DimOrder	103	128	192	102
	2–Phase	107	74	146	101
	4–Phase	198	160	146	101
	Valiant	343	258	293	258
4 × 4 × 4 torus	DimOrder	16	—	63	22
	2–Phase	30	—	48	29
	3–Phase	32	—	46	28
	Valiant	63	—	73	62

Table 6: Performance of the various algorithms, for the tasks and networks tested. Measurements are in cycles per message.

pect ROMM routing to outperform DimOrder routing for injection(ejection) limited routing. We also expect Valiant to be more competitive than on the two–dimensional mesh. This is precisely what we observe in the simulation results. For full–random tasks, ROMM algorithms are expected to provide only a small advantage over DimOrder due to the very short extents of the axis in the three–dimensional mesh. In fact, the better balancing of edge–loads at injection by 2–Phase and 3–Phase do not make up for the larger headers. However, on a larger three–dimensional mesh the merits of ROMM routing are expected to be noticeable.

Table 6 summarizes the performance of the algorithms, for the tasks and networks tested. For the 2–dimensional networks, 2–Phase performs 2 to 3 times better than Valiant, in nearly every case. Furthermore, it is always competitive with DimOrder, and in some cases, nearly twice as fast. For the 3–dimensional torus, 2–Phase and 3–Phase are 1.5 to 2 times faster than Valiant and competitive with, or better than, DimOrder, except for the bit–complement case. However, the three–dimensional network is very small and the relative merits of ROMM routing not well exposed.

Table 7 provides a comparison of expected maximum edge–loads with actual routing times for transpose tasks. The routing times for 2–Phase and DimOrder are close to the time predicted by the maximum edge–load, which suggest that ROMM routing's advantage should scale with network size. The performance of Valiant is not predicted well by the bounds in the table. However, Valiant still has the potential to be the best algorithm for some problems on large, high dimensional networks. In Figure 8, we plot routing time against network size, for transpose and single–random tasks. In the figure, the plots on the right are scaled by the network bisection width. As predicted, the behavior observed on 256 node meshes scales, at least up to 1024 nodes.

7 Conclusions

We have presented ROMM routing, a class of Randomized, Oblivious, Multi–phase, Minimal routing algorithms. Our analysis and simulation results show that ROMM routing for comparable network resources has the potential to offer better performance than DimOrder routing by decreas-

Algorithm	Expected Maximum Edge Load (flits)	Cycles Required
DimOrder	12,000	12,017
2–Phase	6,375	6,652
Valiant	6,375	17,264

Table 7: A comparison of the expected maximum edge load and the actual routing time for transpose permutations on a 16 × 16 mesh network, for routing 50 messages.

ing the contention for network resources. The minimality of ROMM routing is an advantage compared to Valiant routing for many problems and networks. ROMM routing outperformed Valiant routing in all our simulations. Moreover, ROMM routing tends to balance the edge–loads at sources and destinations, as seen with the transpose and single–random tasks. The advantages of ROMM routing should scale to large network sizes, since the maximum edge–load tends to increase with network size, but less rapidly for ROMM routing than DimOrder routing.

Freedom from deadlock for p–Phase ROMM routing is achievable using p virtual channels per physical channel on mesh networks and $2p$ virtual channels per physical channel on torus networks. These results suggest that the simplicity and efficiency of ROMM routing makes it a viable alternative to deterministic and adaptive methods. Furthermore, compared to Valiant routing, ROMM routing is of particular interest in non–homogeneous networks since the minimality, combined with mappings of the index space to memory locations preserving locality of reference with respect to low capacity links or bisections, may reduce the message traffic across those links or bisections.

The strong performance of ROMM for single–random tasks is of interest, because these problems are representative general routing of large blocks of information. Finally, we note that randomizing the order in which the dimensions are traversed appears more useful than simply selecting random destinations along a minimal path.

In this paper, we have explored only a small set of the design parameters in ROMM routing. There are many potentially worthwhile choices of intermediate nodes. Furthermore, routing algorithms other than DimOrder, possibly random ones, could be used in each phase. Such algorithms could improve performance by eliminating the dimension bias introduced by DimOrder. Buffer scheduling policies other than round–robin are also of interest. We have only made a preliminary study of the impact of buffer depth, which indicates that there is a monotonic performance improvement with increased buffer size. Other important aspects of wormhole routing in the context of ROMM routing as well as for other routing schemes is the effect of message size and the behavior in routing with mixed sizes. Yet another issue the requires further study is the conjectured "tangling" of worms.

8 Acknowledgements

We would like to acknowledge the support of the National Center for Supercomputing Applications, the Navy Research Laboratory, Washington D.C., the Massachusetts Institute of Technology, and the Northeast Parallel Applications Center for providing access to Connection Machine systems. We

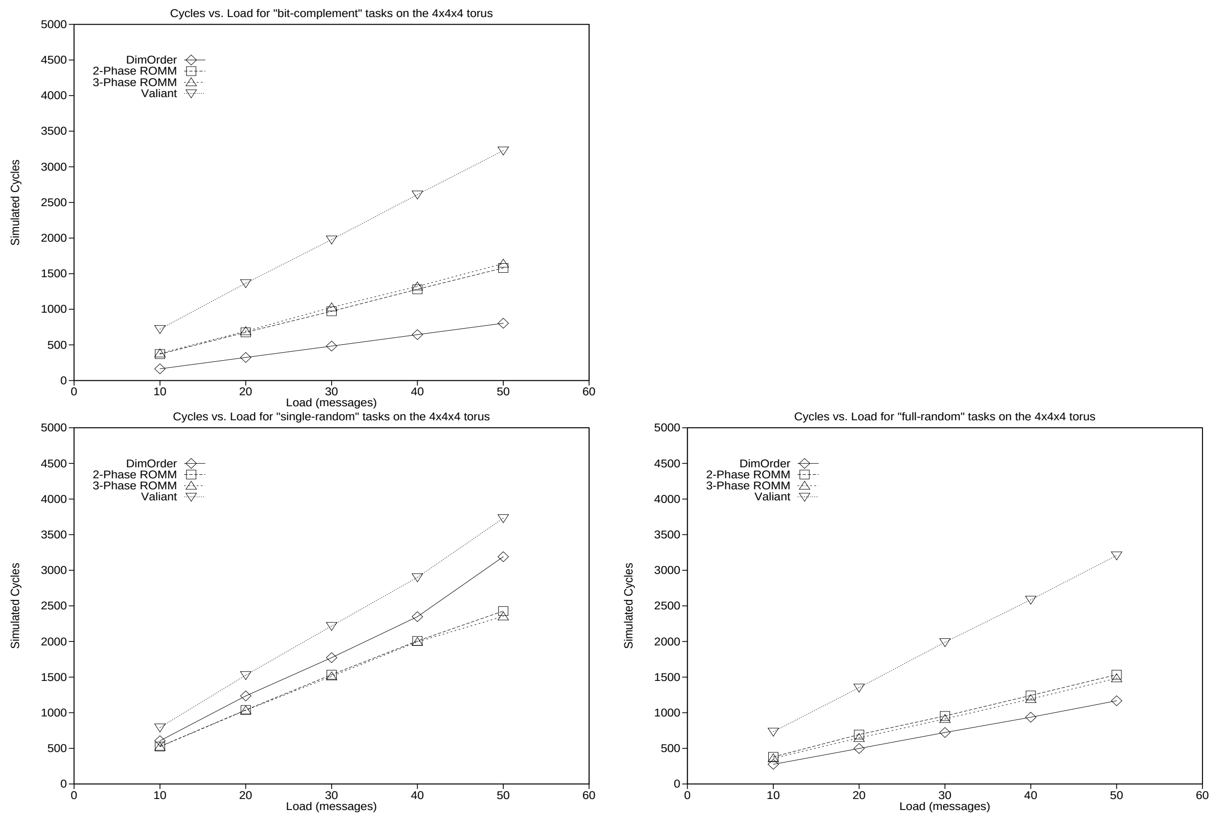

Figure 7: Graphs of simulated cycles vs. load for $4 \times 4 \times 4$ torus networks.

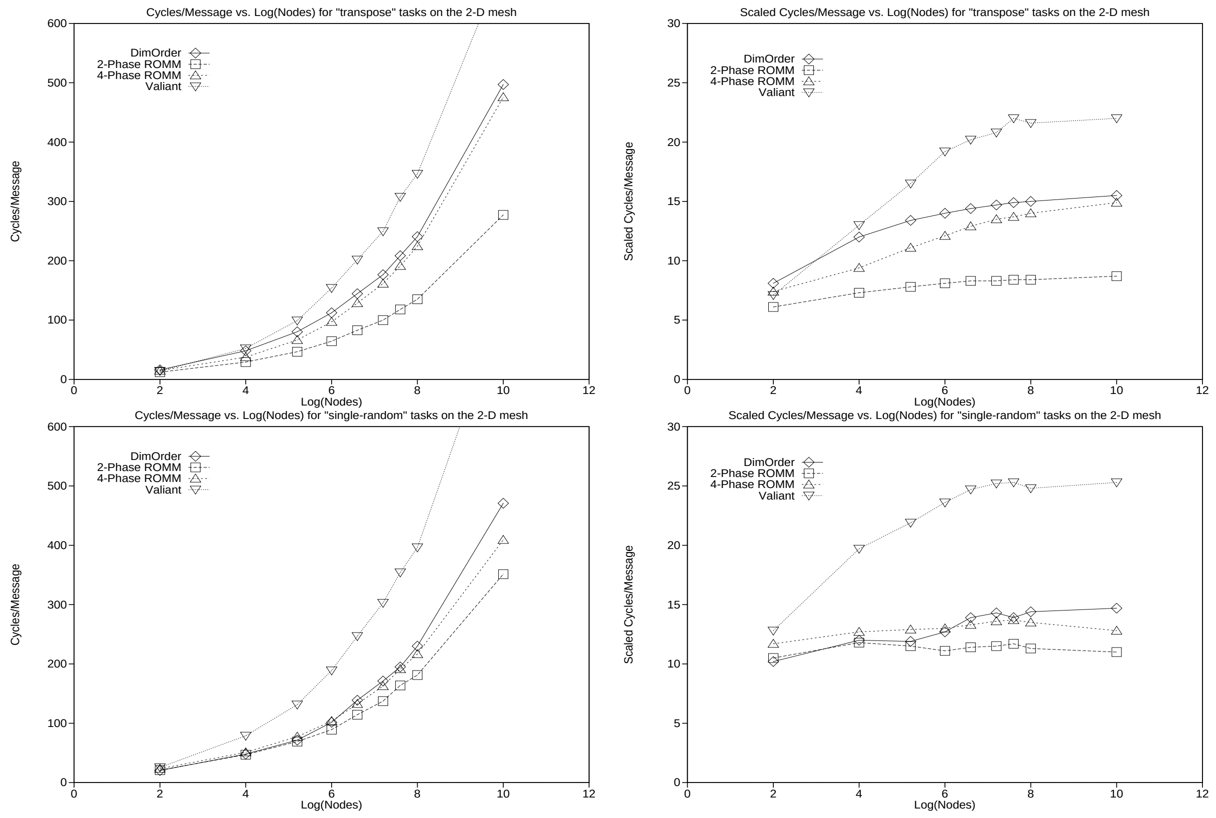

Figure 8: Graphs of cycles/message vs. mesh size, for transpose and single–random on square meshes.

would also like to acknowledge the support of the Office of Naval Research through grant N00014-93-1-0192 and the Air Force Office of Scientific Research through grant F49620-93-1-0480.

References

[1] F. Abolhassan, R. Drefenstedt, J. Keller, W. J. Paul, and D. Scheerer. On the Physical Design of PRAMs. *Computer Journal*, 36(8):756–762, December 1993.

[2] A. Agarwal, D. Chaiken, G. D'Souza, K. Johnson, D. Kranz, J. Kubiatowicz, K. Kurihara, B.-H. Lim, G. Maa, D. Nussbaum, M. Parkin, and D. Yeung. The MIT Alewife Machine: A Large–Scale Distributed–Memory Multiprocessor. Technical Report MIT/LCS TM 454, Massachusetts Institute of Technology, Cambridge, MA, 1991. WWW URL is *http://cag-www.lcs.mit.edu*.

[3] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. Seizovic, and W.-K. Su. Myrinet – A Gigabit–per–second Local–Area Network. *IEEE MICRO*, February 1995. To appear.

[4] K. Bolding, S.-C. Cheung, S.-E. Choi, C. Ebeling, S. Hassoun, T. A. Ngo, and R. Wille. The Chaos Router Chip: Design and Implementation of an Adaptive Router. In *Proceedings of VLSI '93*, September 1993.

[5] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P. S. Tseng, J. Sutton, J. Urbanski, and J. Webb. iWARP: An Integrated Solution to High–Speed Parallel Computing. In *Proceedings of the Supercomputing Conference*, pages 330–338. IEEE, November 1988.

[6] R. Cypher and L. Gravano. Requirements for Deadlock–Free Adaptive, Packet Routing. In *11th ACM Symposium on Principles of Distributed Computing*, 1992. Also to appear in SIAM Journal of Computing.

[7] R. Cypher and L. Gravano. Storage–Efficient, Deadlock–Free Packet Routing Algorithms for Torus Networks. *IEEE Trans. on Computers*, 43(12):1376–1385, December 1994.

[8] W. J. Dally. Virtual–Channel Flow Control. *IEEE Trans. on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[9] W. J. Dally and H. Aoki. Deadlock–Free Adaptive Routing in Multicomputer Networks Using Virtual Channels. *IEEE Trans. on Parallel and Distributed Systems*, 4(4):466–475, April 1993.

[10] W. J. Dally and C. L. Seitz. Deadlock–Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Trans. on Computers*, C-36(5):547–553, May 1987.

[11] J. Duato. A New Theory of Deadlock–Free Adaptive Routing in Wormhole Networks. *IEEE Trans. on Parallel and Distributed Systems*, 4(12):1320–1331, December 1993.

[12] J. Duato and P. López. Performance Evaluation of Adaptive Routing Algorithms for k–ary n–cubes. In *Proc. of the 1994 Parallel Computer Routing and Communication Workshop*. Springer-Verlag, May 1994.

[13] P. T. Gaughan and S. Yalamanchili. Adaptive Routing Protocols for Hypercube Interconnection Networks. *IEEE COMPUTER*, 26(5):12–23, May 1993.

[14] C. J. Glass and L. M. Ni. The Turn Model for Adaptive Routing. *Journal of the ACM*, 41(5):874–902, September 1994.

[15] L. Gravano, G. D. Pifarré, P. E. Berman, and J. L. C. Sanz. Adaptive Deadlock– and Livelock–Free Routing With All Minimal Paths in Torus Networks. *IEEE Trans. on Parallel and Distributed Systems*, 5(12):1233–1251, December 1994.

[16] K. D. Günther. Prevention of Deadlocks in Packet–Switched Data Transport Systems. *IEEE Trans. on Communications*, 29(4):512–524, April 1981.

[17] W. Daniel Hillis. *The Connection Machine*. MIT Press, Cambridge, MA, 1985.

[18] Ching-Tien Ho, M.T. Raghunath, and S. Lennart Johnsson. An Efficient Algorithm for Gray–to–Binary Permutation on Hypercubes. *Journal of Parallel and Distributed Computing*, 20(1):114–120, 1994.

[19] S. Lennart Johnsson. Minimizing the Communication Time for Matrix Multiplication on Multiprocessors. *Parallel Computing*, 19(11):1235–1257, 1993.

[20] S. Lennart Johnsson and Ching-Tien Ho. On the Conversion between Binary Code and Binary-Reflected Gray code. Technical Report TR-20-91, Center for Research in Computing Technology, Harvard University, Cambridge, MA, July 1993. To appear in IEEE Transactions on Computers. WWW URL is *http://www.das.harvard.edu/aiken.html*.

[21] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight Bounds for Oblivious Routing in the Hypercube. In *Proc. of the 2nd Annual ACM Symp. on Parallel Algorithms and Architectures*, pages 31–36. ACM Press, July 1990.

[22] P. Kermani and L. Kleinrock. Virtual Cut–Through: A New Computer Communication Switching Technique. *Computer Networks*, 3:267–286, 1979.

[23] S. Konstantinidou. Adaptive, Minimal Routing in Hypercubes. In *Proceedings of the 6th MIT Conference on Advanced Research in VLSI*, pages 139–153. MIT Press, 1990.

[24] S. Konstantinidou and L. Snyder. The Chaos Router. In *IEEE Trans. on Computers*. IEEE, December 1994.

[25] T. Leighton. Methods for Message Routing in Parallel Machines. In *Proc. of the 24th Annual ACM Symp. on the Theory of Computing*, pages 77–96. ACM Press, May 1992.

[26] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak. The Network Architecture of the Connection Machine CM–5. In *Proc. of the 4th Annual ACM Symp. on Parallel Algorithms and Architectures*, pages 272–285. ACM Press, July 1992.

[27] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy. The DASH Prototype: Implementation and Performance. In *Proc. of the 19th International Symp. on Computer Architecture*, pages 92–103. IEEE, May 1992.

[28] Y.-D. Lyuu. *An Information Dispersal Approach to Issues in Parallel Processing.* PhD thesis, Harvard University, 1990.

[29] P. M. Merlin and P. J. Schweitzer. Deadlock Avoidance in Store–and–Forward Networks — I: Store–and–Forward Deadlock. *IEEE Trans. on Communications*, COM-28(3):345–354, March 1980.

[30] T. Nesson and S. L. Johnsson. ROMM Routing: A Class of Efficient Minimal Routing Algorithms. In *Proc. of the 1994 Parallel Computer Routing and Communication Workshop.* Springer-Verlag, May 1994.

[31] T. Nesson and S. L. Johnsson. ROMM Routing: A Class of Efficient Minimal Routing Algorithms. Technical Report TR-21-94, Center for Research in Computing Technology, Harvard University, Cambridge, MA, August 1994. WWW URL is *http://www.das.harvard.edu/aiken.html*.

[32] J. Y. Ngai and C. L. Seitz. A Framework for Adaptive Routing in Multicomputer Networks. In *Proc. of the 1st Annual ACM Symp. on Parallel Algorithms and Architectures*, pages 1–9. ACM Press, June 1989.

[33] L. M. Ni and P. K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE COMPUTER*, 26(2):62–76, Feb 1993.

[34] M. D. Noakes, D. A. Wallach, and W. J. Dally. The J–Machine Multicomputer: An Architectural Evaluation. In *Proc. of the 20th International Symp. on Computer Architecture.* IEEE, May 1993. WWW URL is *http://cag-www.lcs.mit.edu*.

[35] W. Oed. The Cray Research Massively Parallel Processor System CRAY T3D. WWW URL is *http://www.cray.com*, November 1993.

[36] M. J. Pertel. A Critique of Adaptive Routing. Technical Report CS-TR-92-06, Dept. of Computer Science, California Institute of Technology, Pasadena, CA, 1992.

[37] G. D. Pifarré, L. Gravano, G. Denicolay, and J. L. C. Sanz. Adaptive Deadlock– and Livelock–Free Routing in the Hypercube Network. *IEEE Trans. on Parallel and Distributed Systems*, 5(11):1121–1139, November 1994.

[38] G. D. Pifarré, L. Gravano, S. A. Felperin, and J. L. C. Sanz. Fully Adaptive Minimal Deadlock–Free Packet Routing in Hypercubes, Meshes, and Other Networks: Algorithms and Simulations. *IEEE Trans. on Parallel and Distributed Systems*, 5(3):247–263, March 1994.

[39] Abhiram Ranade. How to emulate shared memory. In *Proceedings of the 28th Annual Symposium on the Foundations of Computer Science*, pages 185–194. IEEE Computer Society, October 1987.

[40] Abhiram G. Ranade, Sandeep N. Bhatt, and S. Lennart Johnsson. The Fluent abstract machine. In *Advanced Research in VLSI, Proceedings of the fifth MIT VLSI Conference*, pages 71–93. MIT Press, 1988.

[41] J. Rexford. Personal Communication, May 1994. Seattle, WA.

[42] C. L. Seitz, N. J. Boden, J. Seizovic, and W.-K. Su. The Design of the Mosaic C Multicomputer. In *Proceedings of the University of Washington Symposium on Integrated Systems.* MIT Press, 1993.

[43] H. Sullivan and T. R. Brashkow. A Large Scale Homogeneous Machine. In *Proc. of the 4th International Symp. on Computer Architecture*, pages 105–124. IEEE, 1977.

[44] A. S. Tanenbaum. *Computer Networks.* Prentice Hall, Englewood Cliffs, New Jersey, 2nd edition, 1989.

[45] Thinking Machines Corp. *CM–5 Technical Summary*, 1991.

[46] Thinking Machines Corp. *CMSSL for CM Fortran, Version 3.1*, 1993.

[47] A. M. Tsantilas. A Refined Analysis of the Valiant–Brebner Algorithm. Technical Report TR-22-89, Center for Research in Computing Technology, Harvard University, Cambridge, MA, 1989.

[48] L. G. Valiant. A Scheme for Fast Parallel Communication. *SIAM Journal on Computing*, 11(2):350–361, May 1982.

[49] L. G. Valiant and G. J. Brebner. Universal Schemes for Parallel Communication. In *Proc. of the 13th Annual ACM Symp. on the Theory of Computing*, pages 263–277. ACM Press, May 1981.