Optimal and Heuristic Application-Aware Oblivious Routing

Michel A. Kinsy, Myong Hyon Cho, Keun Sup Shim, Mieszko Lis, G. Edward Suh, *Member*, *IEEE*, and Srinivas Devadas, *Fellow*, *IEEE*

Abstract—Conventional oblivious routing algorithms do not take into account resource requirements (e.g., bandwidth, latency) of various flows in a given application. As they are not aware of flow demands that are specific to the application, network resources can be poorly utilized and cause serious local congestion. Also, flows, or packets, may share virtual channels in an undetermined way; the effects of head-of-line blocking may result in throughput degradation. In this paper, we present a framework for application-aware routing that assures deadlock freedom under one or more virtual channels by forcing routes to conform to an acyclic channel dependence graph. In addition, we present methods to statically and efficiently allocate virtual channels to flows or packets, under oblivious routing, when there are two or more virtual channels per link. Using the application-aware routing framework, we develop and evaluate a bandwidth-sensitive oblivious routing scheme that statically determines routes considering an application's communication characteristics. Given bandwidth estimates for flows, we present a mixed integer-linear programming (MILP) approach and a heuristic approach for producing deadlock-free routes that minimize maximum channel load. Our framework can be used to produce application-aware routes that target the minimization of latency, number of flows through a link, bandwidth, or any combination thereof. Our results show that it is possible to achieve better performance than traditional deterministic and oblivious routing schemes on popular synthetic benchmarks using our bandwidth-sensitive approach. We also show that, when oblivious routing is used and there are more flows than virtual channels per link, the static assignment of virtual channels to flows can help mitigate the effects of head-of-line blocking, which may impede packets that are dynamically competing for virtual channels. We experimentally explore the performance tradeoffs of static and dynamic virtual channel allocation on bandwidth-sensitive and traditional oblivious routing methods.

Index Terms—Systems-on-chip, on-chip interconnection networks, oblivious routing, virtual channel allocation

1 Introduction

Routers can be generally classified into oblivious and adaptive [23]. In oblivious routing, the path is completely determined by the source and the destination. Deterministic routing is a subset of oblivious routing, where the same path is always chosen between a source-destination pair. Thanks to its distributed nature where each node can make its routing decisions independent of others, oblivious routing, such as dimension order routing [7], enables simple and fast router designs and is widely adopted in today's on-chip interconnection networks. On the other hand, today's oblivious routing algorithms can have difficulty with certain traffic patterns because many flows can be routed through the same link and generate heavy network congestion, even if other network resources are not being used.

In adaptive routing, given a source and a destination address, the path taken by a particular packet is dynamically adjusted depending on, for instance, network congestion. With this dynamic load balancing, adaptive routing can potentially achieve better throughput and latency compared to oblivious routing. However, adaptive routing methods face a difficult challenge in balancing router complexity with the capability to adapt. To achieve the best performance through adaptivity, a router ideally needs global knowledge of the current network status. However, due to router speed and complexity, dynamically obtaining a global and instantaneous view of the network is often impractical. As a result, adaptive routing in practice relies primarily on local knowledge, which limits its effectiveness.

In this paper, we present an application-aware oblivious routing framework that statically determines deadlock-free routes considering an application's communication characteristics. The framework supports a variety of algorithms that optimize various cost functions, for example, maximum channel load across all links when bandwidth demands of flows are known, or latency of (a subset of) routes, or a combination thereof. Our focus in this work is on bandwidth-sensitive oblivious routing, with static virtual channel allocation, which produces deadlock-free routes given rough estimates of bandwidth demands of all flows obtained through application program analysis and/or profiling. Using these estimates, an *offline* algorithm determines routes for the data transfers that maximize satisfaction of flow demand or minimize maximum channel load, while ensuring deadlock freedom. The network is then statically configured prior to runtime as processing elements are loaded with the computation code. This approach can achieve better throughput than traditional oblivious routing algorithms because routes are optimized based on the global

- M.A. Kinsy, M.H. Cho, K.S. Shim, M. Lis, and S. Devadas are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 32 Vassar Street, Cambridge, MA 02139. E-mail: {mkinsy, mhcho, ksshim, mieszko, devadas}@mit.edu.
- G.E. Suh is with the School of Electrical and Computer Engineering, Cornell University, Frank H T Rhodes Hall, Room 338, Ithaca, NY 14853. E-mail: suh@ece.cornell.edu.

Manuscript received 2 Sept. 2010; revised 21 Sept. 2011; accepted 28 Sept. 2011; published online 10 Nov. 2011.
Recommended for acceptance by R. Marculescu.
For information on obtaining reprints of this article, please send E-mail to: tc@computer.org, and reference IEEECS Log Number TC-2010-09-0490.
Digital Object Identifier no. 10.1109/TC.2011.219.

0018-9340/13/$31.00 © 2013 IEEE Published by the IEEE Computer Society

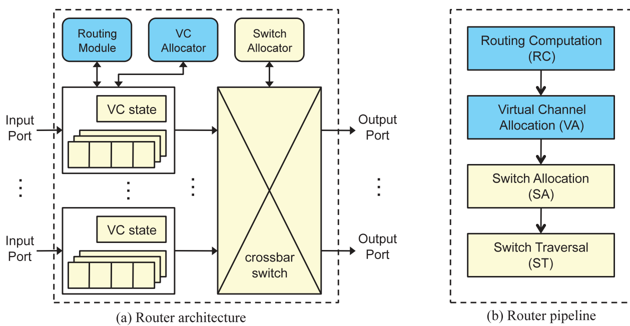


Fig. 1. Typical virtual-channel router architecture. The dark blue indicates that the modules and routing step may be modified for our approach.

knowledge of bandwidth demands. At the same time, the router remains simple because the routes are configured statically and do not change at runtime. Application-aware oblivious routing will be particularly suitable for long-running applications with predictable communication patterns. Our studies on synthetic traffic with various patterns and the H.264 decoder application show throughput improvements over traditional oblivious routing.

Section 2 describes a generic network architecture for oblivious routing, and the small augmentations required to support application-aware oblivious routing. Section 3 describes our framework for application-aware routing, and our approach to avoiding deadlock. Various algorithms for bandwidth-sensitive routing are the subjects of Section 4. Related work is summarized in Section 5. Section 6 compares our routing schemes to existing deterministic and oblivious routing algorithms. Section 7 concludes the paper.

2 Router Architecture

This section discusses the impact of our oblivious routing technique and static virtual channel allocation on the router architecture, and compares the modified architecture with standard routers for other oblivious routing algorithms. The following discussion assumes a typical virtual-channel router on a 2D mesh network as a baseline. However, we note that the proposed routing technique is largely independent of network topology and flow control mechanisms. Therefore, the same approach to routing can be applied to other network topologies and either packet-buffer or flit-buffer flow control.

2.1 Typical Virtual Channel Router

Fig. 1 illustrates a typical virtual-channel router architecture and its operation [8], [19], [26]. As shown in the figure, the data path of the router consists of buffers and a switch. The input buffers store flits while they are waiting to be forwarded to the next hop. There are often multiple input buffers for each physical channel so that flits can flow as if there are multiple "virtual" channels. When a flit is ready to move, the switch connects an input buffer to an appropriate output channel. To control the data path, the router also contains three major control modules: a router, a virtual-channel (VC) allocator, and a switch allocator. These control modules determine the next hop, the next virtual channel, and when a switch is available for each packet/flit.

The routing o peration takes four steps or phases, namely, routing (RC), virtual-channel allocation (VA), switch allocation (SA), and switch traversal (ST), which often represent one to four pipeline stages in modern virtual-channel routers. When a head flit (the first flit of a packet) arrives at an input channel, the router stores the flit in the buffer for the allocated virtual channel and determines the next hop for the packet (RC phase). Given the next hop, the router then allocates a virtual channel in the next hop (VA phase). Finally, the flit competes for a switch (SA phase) if the next hop can accept the flit, and moves to the output port (ST phase). For existing oblivious routing algorithms, such as Dimension Order Routing (DOR) [7], ROMM [22], Valiant [31], and o1turn [28], the next hop of a packet can be easily computed at each router node based on the packet's destination.

2.2 Router Architecture for Application-Aware Oblivious Routing

The router architecture to enable application-aware oblivious routing, or static virtual channel allocation, is almost identical to the typical virtual-channel router architecture. The router uses the exact data path that is described above. The main change in our routing architecture is in its routing module, where route selection is table based, as opposed to combinational logic.

For simple oblivious routing algorithms such as DOR, the baseline architecture implements the algorithm with fixed logic and dynamically allocates virtual channels to a packet. To support our routing scheme with any algorithm variant, our routing module needs table-based routing so that routes can be configured for each application. This single change is sufficient because our routing algorithms ensure that there is no cyclic dependence in routes either through route selection (cf. Section 3.1) or through static channel allocation (cf. Section 3.2). We next discuss the details of the modification.

The router must be *programmable* so that the routes for each flow can be configured depending on the application, and be *flexible* enough to support arbitrary routing paths. In order to provide programmability and flexibility, our router uses *table-based* routing where the path between a pair of nodes is stored in a routing table. Unlike cases where a simple routing algorithm is hardwired with fixed logic (algorithmic routing), the routing table can be simply reprogrammed with new routes before an execution of a new application in order to update the routing. The table-based approach also allows our routing algorithm to select almost any path from a source to a destination as long as the route can fit into the table.

Table-based routing can be realized in two different ways: source routing and node-table routing, and our routing technique can also be implemented in both styles. In the *source routing* approach, each node has a routing table that contains a route from itself to each destination node in the network. The routes are precomputed by our routing algorithms and programmed into the tables before the execution of an application. When sending a packet, the node prepends this routing information to the packet. Routers along the path can determine the output port simply by looking up the routing flits. Fig. 2a illustrates

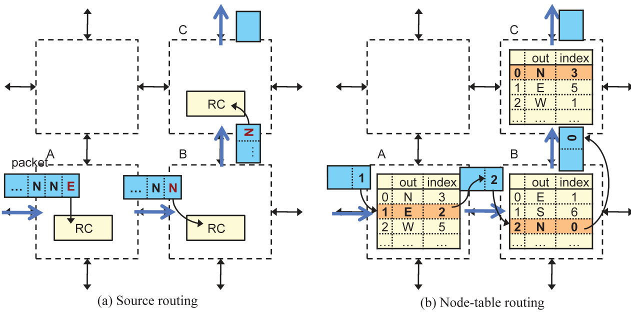


Fig. 2. The table-based routing architecture. (a) Source routing. (b) Node-table routing.

source routing where a packet is routed through nodes A, B, and C. The route corresponds to East, North, and North, which is reflected in the routing flits in the packet. Thanks to its speed and simplicity, source routing has been widely used in many router designs including the IBM SP1 [29] and the Avici TSR [6], although it leads to longer packets containing routing flits as compared to the case where the route is computed for each hop.

With *node-table* routing, the routing module of a node contains a routing table that has the output port for each flow that is routed through the node. To determine which table entry corresponds to each packet, the packet carries an index field for the current node and the routing table provides the new index for the next hop. Upon receiving a packet, a router reads its routing table to determine the proper output port and forwards the packet with the new index field from the table. Fig. 2b shows an example of node-table routing when a packet is routed through the same path with the source routing example. As shown in the figure, the incoming packet to node A contains the table index of 1. To route this packet to B (East), the entry (1) in A's routing table is set as (East, 2), indicating that the packet should be routed to East with the new index of 2. In the same way, the packet looks up the second entry in node B for routing.

Both routing methods, namely source routing and node-table routing, are widely known and have been implemented in multiple routers [29], [6], [11]. In other words, the proposed routing approach can be realized with standard routing hardware without new specialized mechanisms. Also, our routing approach will not have noticeable impact on the latency or the organization of the router pipeline.

2.3 Router Architecture for Static Virtual Channel Allocation

Statically allocating a VC to each flow simplifies the VC allocation step of the baseline router. A previous study shows that the latency of a pipelined virtual-channel router is dominated by virtual-channel allocation, when done dynamically, which incurs 15-20 FO4 delay [26]. In our scheme, VCs at each link are allocated per flow by the routing algorithm, rather than being dynamically allocated using arbiters. The router then assigns the next-hop VC in the same way as it obtains the route: with source routing, each packet carries its VC number for each hop along with its route, while in node-table routing, an entry in the routing table is augmented with the VC number for the flow. Since the router can thus obtain both the output port and the next VC number in the RC step, the primary complexity in the VA step lies in the arbitration among packets: two or more packets may be assigned the same VC simultaneously, and arbitration is needed to determine which packet will be sent first. This requires a $P \cdot V$ to 1 arbitration for each VC, where packets from P physical channels with V VCs each vie for the same VC, and is simpler than the $P \cdot V$ to V arbitration required by dynamic routing. Peh and Dally, indicate that $P \cdot V$ to 1 arbitration is about 20 percent faster than $P \cdot V$ to V arbitration (11.0 FO4 versus 13.3 FO4 with eight VCs) [26]. Static VC allocation requires additional bits in the routing table to specify the VC for each flow. For example, for eight VCs, three extra bits are required for each entry; if each routing table has 256 entries, this results in an increase of 96 bytes, still keeping the routing table accessible in a single cycle.

3 Deadlock-Free Oblivious Routing

Application-aware oblivious routing exploits application knowledge, under a given topology, to provide deadlock freedom and efficient network resource (e.g., bandwidth, virtual channel) allocation crucial to the performance of the routing algorithm. In this section, we present the framework and introduce two deadlock avoidance techniques, namely, acyclic channel dependency routing and static virtual channel allocation.

3.1 Channel Dependency-Based Deadlock Avoidance

3.1.1 Definitions and Framework

We first give standard definitions of flow networks and channel dependence graphs (CDG).

Definition 1. *Given a flow graph $G(V, E)$, where an edge $(u, v) \in E$ has capacity $c(u, v)$, the capacities $c(u, v)$ are the available bandwidths on the edge. There is a set of k data transfers or flows $K = \{K_1, K_2, \ldots, K_k\}$. $K_i = (s_i, t_i, d_i)$, where s_i and t_i are the source and sink, respectively, for connection i, and d_i is the demand. We assume $s_i \neq t_i$. We may have multiple flows with the same source and destination pairs. The flow variable i along edge (u, v) is $f_i(u, v)$. A route is a path p_i from s_i to t_i for a flow i. Edges along this path will have $f_i(u, v) > 0$, other edges will have $f_i(u, v) = 0$.*

If $f_i(u, v) > 0$, then route p_i will use both bandwidth and buffer space on the edge (u, v). The value of $f_i(u, v)$ indicates how much of the edge's bandwidth is being used by flow i. We will assume flit-buffer flow control in this paper, though our framework can be applied to other flow control schemes as well.

Definition 2. *A channel dependence graph $D(V', E')$ is derived from the flow network G as follows: each vertex in V' is an edge in G. There is an edge from $v_1 \in V'$ to $v_2 \in V'$ if a packet can flow from the edge in G associated with v_1 into the edge associated with v_2, without traversing any other edges. That is, the edges are consecutive in G.*

Fig. 3 shows a bidirectional 3×3 mesh and its associated CDG. BC and CB are edges in opposite directions from B to C and C to B, respectively. They correspond to separate vertices in the CDG. Note that the CDG has cycles.

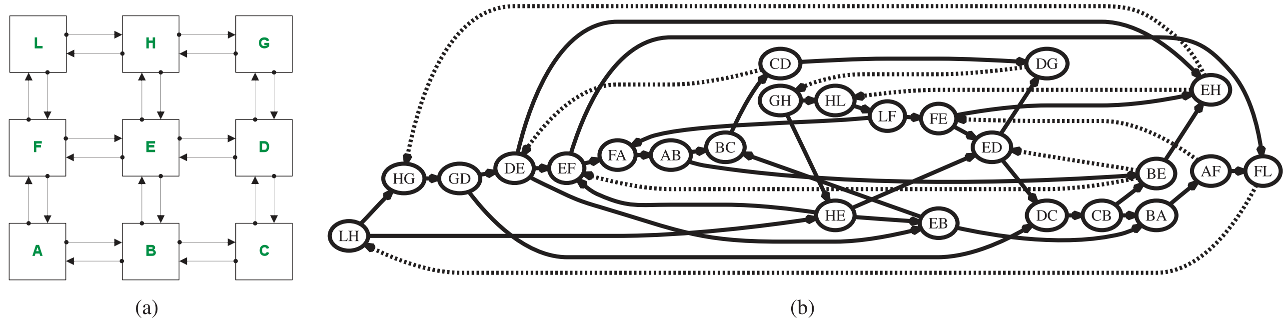


Fig. 3. (a) 3×3 mesh. (b) Channel dependence graph without 180 degree turns.

Application-aware oblivious routing follows the framework of Fig. 4. We need to ensure that the routes selected are deadlock free, and this is done by creating an acyclic CDG D_A (Step 1), deriving a flow network G_A (Step 2) and generating routes by selecting paths on G_A (Step 3). We can explore different acyclic CDG's by deleting different edges from the cyclic CDG to create different D_A's (Step 4). The best set of routes according to our cost function is chosen (Step 5).

Assuming a single virtual channel per link, if packets follow routes that conform to an acyclic channel dependence graph, then network deadlock will not occur [7]. This is also a necessary condition, provided false resource dependences do not exist [27]. Therefore, we have to restrict routing by breaking all the cycles in the CDG D associated with the network. This can be done in many ways; the turn model [13] provides a few systematic ways. Fig. 5 shows two different turn models that can be used in a 2D mesh. While the turn model was developed to enable adaptive routing, we use it to choose routes in an offline fashion for oblivious routing. For example, for the 3×3 mesh, using the North-Last model to break cycles implies removing the dotted edges in Fig. 3b, and produces the acyclic CDG of Fig. 6a. Cycles can also be broken in an *ad hoc* or random fashion as shown in Fig. 6b. Typically, a larger number of dependences need to be removed to obtain an acyclic CDG, but after route selection under this type of CDG, we may obtain a better result. We can use *any* acyclic CDG to drive an application-aware oblivious routing algorithm. Given that different CDG's may result in different qualities of routes, we can perform route selection under many different CDG's and select the best result. To generate deadlock-free routes that conform to a given acyclic CDG, a flow network is derived from the CDG, as described next.

3.1.2 Deriving a Flow Graph from an Acyclic CDG

Given source and destination network nodes s_i and t_i, respectively, for each flow i, we derive a flow graph or network G_A from an acyclic CDG D_A. We can then run our route selection algorithm on G_A, to find the "best" routes for the flows (cf. Section 4). This will have the effect of running route selection on the original flow network G corresponding to the interconnection network, but with the route conforming to D_A. If the routes for all flows conform to D_A, deadlock freedom is assured. G_A is derived from D_A as follows: D_A is copied over to G_A. We add "dummy" vertices to G_A corresponding to s_i and t_i, for each i. We add edges from s_i to all vertices in G_A that have s_i as the source node of the corresponding link. For example, if s_i is network node A in the 3×3 mesh shown in Fig. 3a, then edges are added from s_i to AB and AF. For each vertex in G_A that has t_i as the destination node of the corresponding link, we add an edge from the vertex to t_i. For example, if t_i is network node I in the 3×3 mesh shown in Fig. 3a, then edges are added from FL to t_i and from HL to t_i. These dummy vertices are primarily for convenience and are not necessary. They avoid having to find the best route from multiple vertices in G_A to one of several possible destination vertices. In our example, say that we want to find the best route in G_A starting with either AB or AF and ending at either FL or HL. Fig. 6c shows a flow network derived from the acyclic CDG of Fig. 6b, given source-destination pairs A, L and E, G.

FRAMEWORK (Flows Transfers K)
1. Create (new) acyclic CDG D_A by deleting some edges from D.
2. Transform D_A into a flow network G_A, with flows K.
3. Perform application-aware routing of flows in G_A.
4. If desired, go to Step 1.
5. Select the best set of routes found.

Fig. 4. Offline application-aware oblivious routing framework.

3.2 Virtual Channel Allocation-Based Deadlock Avoidance

3.2.1 Flow-Based Virtual Channel Allocation

When the routing algorithm generates routes that do not conform to a particular acyclic CDG or turn model, they may not be deadlock free. However, if the number of available virtual channels is two or more, and routes are minimal, we can ensure deadlock freedom via static virtual channel assignment by partitioning the flows across available virtual channels.

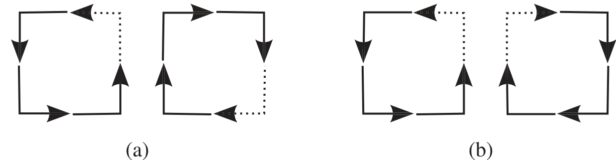


Fig. 5. (a) Turns allowed (solid) and disallowed (dotted) under the West-First turn model. (b) Turns allowed and disallowed under the North-Last turn model.

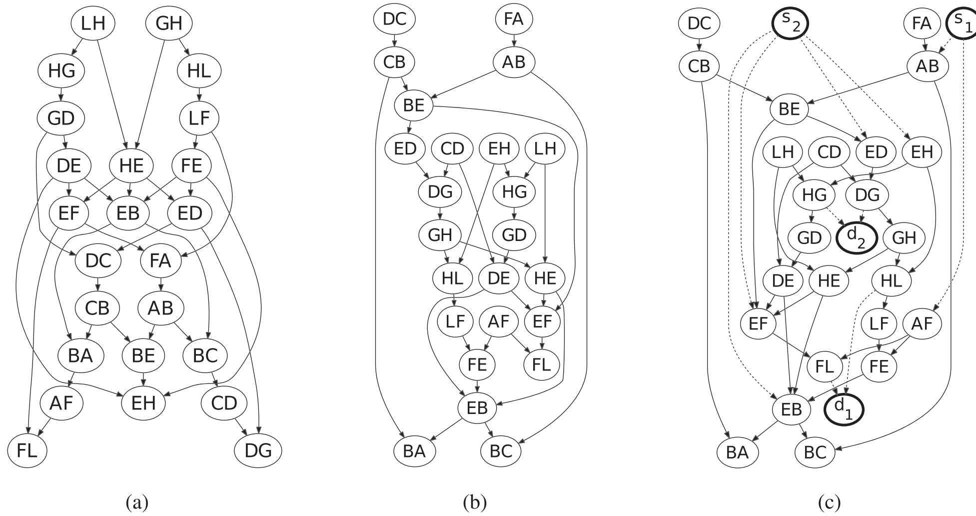


Fig. 6. (a) North-Last routing CDG without 180 degree turns: 32 edges removed. (b) Ad hoc CDG without 180 degree turns: 36 edges removed. (c) Flow network from acyclic CDG of (b) with source-destination pairs A, L and E, G.

Theorem 1. *Given a router with* ≥ 2 *VCs, and an arbitrary set of minimal routes over an* $n \times n$ *mesh, it is possible to statically allocate VCs to each flow to ensure deadlock freedom.*

Proof. Consider, without loss of generality, the case of two VCs. Fig. 7a shows the eight possible minimal routes with two types of turns each (two-turn-type routes). Given the constraint of minimality, minimal routes can only have two *types* of turns, even though they may have many more turns. Minimal routes that have one type of turn or no turns can be ignored as special cases of two-turn-type routes for the subsequent analysis. Looking at Fig. 7a, it is easy to see that minimal routes 3, 4, 5, and 8 conform to the West-First turn model (but violate the North-Last model as shown by the boxes over the violating turns), while minimal routes 1, 2, 6, and 7 conform to the North-Last turn model (but violate the West-First turn model as indicated by the circles over the illegal turns). Therefore, we can partition an arbitrary set of routes into two sets: the first conforming to the West-First turn model, and the second to the North-Last model. Note that the four single-turn-type minimal routes shown in Fig. 7b, and routes with no turns, can be placed in either set; the four other single-turn-type routes (not shown) will be forced to one of the sets. If we assign VC 1 to the first set and VC 0 to the second, no deadlock can occur. □

The proof of Theorem 1 suggests a static VC allocation strategy. Given an application and a collection of minimal routes, we create three sets of flows:

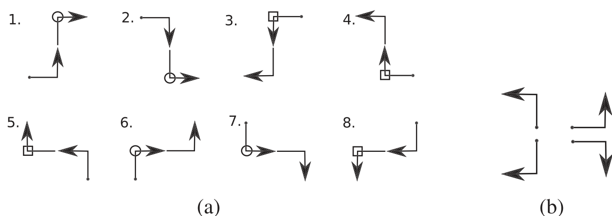


Fig. 7. (a) The eight different two-turn minimal routes on a 2D mesh. (b) The four (out of a possible eight) different one-turn routes on a 2D mesh that conform to both the West-First and North-Last turn model.

1. Flows with two-turn-type and single-turn-type routes that conform to the West-First turn model.
2. Flows with two-turn-type and single-turn-type routes that conform to the North-Last turn model.
3. Flows with single-turn-type or zero-turn-type routes that conform to both.

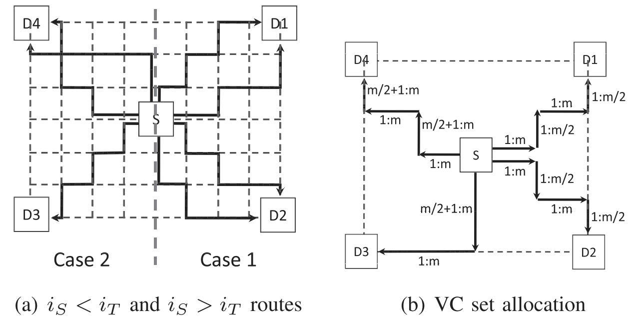
Before moving on to static VC allocation, we assign the flows in the third set to either of the first two sets, appropriately balancing the bandwidths and number of flows. Each flow in the third set is assigned to the set that has fewer flows than share links with the flow, or, if the number of shared flows is the same for both sets, to the set with fewer flows. After only two sets remain, we have *local* flexibility in determining the ratio of VCs across the two sets. The number of flows for the first set and that for the second set can be different for each link, so we must assign VCs to the two sets on a per-link basis. We follow a simple principle: at each link, split available VCs evenly into two groups associated with the two flow sets and, if unused VCs remain in exactly one group, shift the unused VCs to the other group. For example, if the number of flows in the first set is two and that for the second set is six, the VCs are divided into two groups of size (1,1), (2,2), and (2,6) for #VC=2, #VC=4, and #VC=8, respectively. Notice that for the #VC=8 case, we do not allocate four channels to the first set since it only has two flows. This localized division reduces wasted VCs, and the route is now deadlock free since the two sets of flows are assigned to disjoint groups of channels. Finally, at each link, we assign a given flow to either set, with the VC allocation within the set the same as in DOR.

3.2.2 Packet-Based Virtual Channel Allocation

The primary feature of the flow-based VC allocation approach is also its limitation, since we need prior knowledge of the flows in the application. The packet-based approach is another technique to solve the deadlock problem in oblivious routing algorithms without knowledge of flows or flow demands. In this scheme, each packet

1) Split virtual channels into two disjoint non-empty sets VC 0 and VC 1.
2) If $i_S = i_T$ or $j_S = j_T$, then an S-to-T packet q can use any VC set traversing any link (u, v).
3) If $i_S < i_T$, then an S-to-T packet q traversing a vertical link (u, v), where $i_u = i_v$, must use set VC 0.
4) If $i_S > i_T$, then an S-to-T packet q traversing a vertical link (u, v), where $i_u = i_v$, must use set VC 1.
5) For all other cases, an S-to-T packet q can use any VC set traversing any link (u, v).

Fig. 8. Deadlock-free direction-aware virtual channel allocation (DAVCA).



(a) $i_S < i_T$ and $i_S > i_T$ routes (b) VC set allocation

Fig. 9. Virtual channel assignment under DAVCA.

is routed independently of others, even if they share the same source-destination. For a class of routing algorithms, when routes are minimal, we propose a static virtual channel allocation scheme called Direction-Aware Virtual Channel Allocation (DAVCA) Fig. 8. (We note that this paper does not provide results on DAVCA.)

Direction-aware virtual channel allocation. Given an N-by-M mesh network, we assign a unique pair of coordinates $(i_u; j_u)$ to each node u, representing the position of u in X and Y dimensions on the mesh, with $0 \leq i_u < N$ and $0 \leq j_u < M$. In DAVCA, the static virtual channel assignment depends on the relative position of the source node S and destination node T. The allocation is done in the following manner:

The key advantage of DAVCA is that it guarantees deadlock freedom for *any* k-phase minimal routing with only two VCs for a 2D mesh network. For example, one can readily use DAVCA on conventional k-phase ROMM routing using two virtual channels to provide deadlock freedom, when the original algorithm calls for k VCs [22]. Fig. 9 illustrates the allocation of m virtual channels between $i_S < i_T$ and $i_S > i_T$ traffic under DAVCA.

Theorem 2. *DAVCA is deadlock free.*

Proof. To show that DAVCA is deadlock free, we invoke the turn model [13]. Fig. 5 shows two different turn models that can be used in a 2D mesh: each model disallows two of the eight possible turns, and, when all traffic in a network obeys the turn model, deadlock freedom is guaranteed. The key observation, which we made in Section 3.2.1, is that minimal-path traffic always obeys one of those two turn models: eastbound packets never turn westward, westbound packets never turn eastward, and packets between nodes on the same row or column never turn at all. Thus, westbound and eastbound routes always obey the restrictions of Figs. 5a and 5b, respectively, and placing them on different virtual networks ensures deadlock freedom. Traffic over horizontal links and traffic between nodes on the same column simultaneously conform to both models, and may use both virtual networks. Note that the correct virtual channel allocation for a packet can be determined *locally* at each switch, given only the packet's destination (encoded in its flow ID), and which ingress and virtual channel the packet arrived at. □

4 Bandwidth-Sensitive Oblivious Routing (BSOR) Algorithms

For given application characteristics, our application-aware methodology achieves higher throughput by efficiently *load-balancing* the network. For instance, when estimated bandwidths are available, we can route data traffic in a *bandwidth-sensitive* manner to avoid premature network saturation. It is widely known that a linear programming formulation can determine a lower bound on the maximum channel load [8], [30]. However, the routes given by linear programming may not be realizable on standard routers since a packet flow may need to be split across multiple paths to achieve the maximum throughput. Further, these routes may result in deadlock under a single virtual channel. A routing in which each commodity flows along a single path is called an *unsplittable flow*. Unfortunately, the unsplittable flow problem is NP-hard even for single sources [17], requiring the use of approximation algorithms or heuristics for large problems. Mixed integer-linear programming (MILP) can produce an optimal result either minimizing maximum channel load, or maximizing throughput, for problems of small size (cf. Section 4.1). We will use Dijkstra's weighted shortest-path algorithm [5] in Step 3 of Fig. 4 to heuristically select good routes for large problems (cf. Section 4.2).

4.1 Mixed Integer-Linear Programming

The capacity of an edge in G_A is the capacity of the link/vertex that the edge is incident on. For example, the edge from s_i to AB will be assigned the capacity of link/vertex AB. An edge from AB to BC will be assigned the capacity associated with link/vertex BC. Edges into destination nodes t_i have infinite capacity.

Definition 3. *Assume the specification of Definition 1. Find an assignment of flow in* $G_A(V, E)$, *i.e.,* $\forall i, \forall (u, v) \in E$ $f_i(u, v) \geq 0$, *which satisfies the constraints:*

Capacity:

$$\forall v \neq s_i,\ t_i \quad h(v) = \sum_{i=1}^{k} \sum_{(u,v)\in E} f_i(u, v) \leq c(v).$$

Flow conservation:

$$\forall i,\ \forall u \neq s_i, \quad t_i \sum_{(w,u)\in E} f_i(w, u) = \sum_{(u,w)\in E} f_i(u, w),$$

$$\forall i \sum_{(s_i,w)\in E} f_i(s_i,w) = \sum_{(w,t_i)\in E} f_i(w,t_i) = g_i.$$

Unsplittable flow:

$$\forall i,\ \forall (u,v) \in E \quad f_i(u,v) \le b_i(u,v) \cdot d_i,$$

$$\forall i,\ \forall u \sum_{(u,v)\in E} b_i(u,v) \le 1.$$

Hop Count:

$$\forall i \sum_{(u,v)\in E} b_i(u,v) \le hop_i,$$

and maximizes the total throughput, given as

$$maximize \quad S = \sum_{i=1}^{k} g_i, \tag{1}$$

or maximizes the minimal fraction of the flow of each commodity to its demand

$$maximize \quad T = \min_{1\le i\le k} \frac{g_i}{d_i}, \tag{2}$$

or minimizes the maximum channel load

$$minimize \quad U = \max_{v\in V} h(v). \tag{3}$$

The variables $b_i(u,v)$ are Boolean variables, i.e., they can take on values of 0 or 1 only. They enforce the restriction that a flow i can only take a single path from source s_i to destination t_i. They also enforce path length restrictions. hop_i is a specified constant that can be set to be equal to or greater than the minimal path length between s_i and t_i. The $f_i(u,v)$ variables can take on any positive value less than or equal to the demand d_i.

There are several interesting cost functions. If the flows are uncorrelated as in synthetic benchmarks, we can maximize total throughput given by $\sum_{i=1}^{k} g_i$. In most applications, flows are correlated, i.e., throttling one flow will affect the throughput demand of another. In this case, one possibility is to maximize the minimum fraction of flow demand satisfaction $\min_{1\le i\le k} \frac{g_i}{d_i}$ as in (2). We focus on finding the minimum maximum channel load (MCL) as in (3) because this can be done regardless of network capacity, and only knowing the relative demands of flows. The capacity constraints are dropped; instead, we set $g_i = d_i$, for all flows i. The MILP is run once for each acyclic CDG.

We note that our MILP formulation is over the *CDG* G_A rather than the original network G. This ensures deadlock freedom with a single virtual channel unlike schemes that formulate linear programs over G (e.g., [20]).

4.2 Weighted Shortest-Path-Based Algorithm

For problems of large size, in terms of network size or number of flows in the application, we select a set of routes that heuristically minimizes the number of congested links using Dijkstra's weighted shortest-path algorithm, as an alternative to the MILP formulation. The flows are ordered in terms of decreasing bandwidth demand. We run Dijkstra on a weighted version of G_A, deriving the weights from the residual capacities of each link/vertex. Consider a link e in the original network G (e.g., AB) which is a vertex in G_A. This link has a capacity $c(e)$. We create a variable for each link $\tilde{c}(e)$ which is the current residual capacity of link e. Initially, it is equal to the capacity $c(e)$, and is set to be a constant C. If a flow i is routed through this link e, we will subtract the demand d_i from the residual capacity. Residual capacity is always checked to see whether it is enough to supply the demand for the flow during routing. If there is not enough capacity, then the algorithm never chooses the link, as described below. Therefore, the residual capacity $\tilde{c}(e)$ will never be negative.

For the weighting function, we use the reciprocal of the link residual capacity which is similar to the Constraint Shortest Path First (CSPF) metric described by Walkowiak [32]. The weighting function $w(e) = \frac{1}{\tilde{c}(e)-d_i}$, except if $\tilde{c}(e) \le d_i$, then $w(e) = \infty$, and the algorithm never chooses the link. The constant C is set to be the smallest number that can provide us with routes for all flows without using ∞-weight links. The maximum channel load (MCL) from XY or YX routing gives us an upper bound for C, but in most cases, we can set C lower and still find a solution. The MILP gives us a lower bound for C. A lower C makes the algorithm more aggressively avoid congested links due to their higher weight.

The algorithm as described above assumes weights on the edges in G_A; however, the links of G which have capacities become vertices in G_A. As with the capacity, the weight of an edge in G_A is merely the weight of the link/vertex that the edge is incident on. The edges incident on t_i are always assigned a weight of 0 (they had infinite capacities in the MILP). Fig. 6c showed a flow network derived from the acyclic CDG of Fig. 6b. Weights are assigned to the edges (not shown), and we run Dijkstra's algorithm on the weighted G_A to find a minimum-weight path from A to L, or in general from an s_i to a t_i. Then, the weights are updated, and a new source-destination pair is selected to be routed. This continues until all flows are routed.

We run the same procedure for all acyclic CDGs. For each CDG, we reduce C from the XY routing MCL to the MILP MCL or until we cannot obtain a set of routes, storing the routes obtained for each value of C. We pick the set of routes with lowest MCL among all the computed routes, across all CDGs. We also compute the overall congestion level of the network by taking the product of the average excess bandwidth demand over all links multiplied by the average number of flows competing for each link. We use this congestion level as a tiebreaker when two sets of routes have the same MCL.

4.3 Bandwidth-Sensitive Oblivious Routing with Minimal Routes (BSORM)

We now describe a variation of the BSOR algorithm that targets only minimal routes and can be made deadlock free through static VC allocation as described in Section 3.2.1, when we have two or more virtual channels. Given rough estimates of bandwidths of data transfers or flows, BSORM selects routes to minimize the *maximum channel load*, i.e., the maximum bandwidth demand on any link in the network. This method works directly on the flow graph $G(V,E)$ corresponding to the network, and not on the acyclic channel dependence graph. For each flow, we select a minimal route that heuristically minimizes the maximum channel load using Dijkstra's weighted shortest-path algorithm.

We start with a weighted version of G, deriving the weights from the residual capacities of each link. We run Dijkstra's algorithm on the weighted G to find a minimum-weight path $s_i \rightsquigarrow t_i$ for a chosen flow i. The algorithm we use also keeps track of the number of hops, and finds the minimum-weight path with minimum hop count. While our weight function allows the smallest weight path to be nonminimal, the algorithm will not generate such a path. After the path is found, we check to see whether it can be replaced by one of the XY/YX routes of Fig. 7b, while keeping the same minimum weight; if so, this replacement is made, which minimizes the number of turns in the selected routes and allows greater freedom for the static VC allocation step (cf. Theorem 1). Finally, the weights are updated, and the algorithm continues on to the next flow, until all flows are routed. The resulting set of minimal routes minimizes the *maximum channel load* and the number of turns in selected paths.

4.4 Static Virtual Channel Allocation for Bandwidth-Sensitive Oblivious Routing

Conventional virtual channel (VC) routers dynamically allocate VCs to packets or head/control flits based on channel availability and/or packet/flit waiting time. Typically, any flit can compete for any VC at a link [8], and the associated arbitration is often the highest latency step [26]. Statically allocating VCs to flows can simplify the VC allocation step. Judicious separation of flows during static allocation may also reduce or eliminate head-of-line blocking and, therefore, enhance throughput, although it may result in lower utilization of available VCs because dynamic behavior is not considered.

Static allocation of virtual channels is done in a way that is decoupled from bandwidth allocation. This decoupling allows active flows to use network bandwidth more efficiently. The bandwidth allocation can be done using any flow-based oblivious routing algorithm (e.g., BSOR, BSORM, DOR). When the routing scheme guarantees deadlock freedom by means of acyclic channel dependency, like in BSOR and DOR, flows are statically assigned to virtual channels using the Least Sharing Virtual Channel Allocation (LSVCA) algorithm with no VC set distinction. In the case of VC set-based deadlock-free routing, e.g., BSORM, LSVCA is applied to each set. LSVCA consists of the steps outlined in Fig. 10.

To understand LSVCA we define the notion of entanglement of flows. A pair of flows is said to be *entangled* if the flows share at least one VC across all the links used by both flows. Prior to channel assignment, no pairs of flows are entangled, and, if the number of flows for a given link is smaller then the number of VCs, we can avoid entanglement by assigning one channel per flow. Otherwise, in order to mitigate the effects of head-of-line blocking, we allocate VCs so as to reduce the number of distinct entangled flow pairs as described in Fig. 10.

LSVCA For Deadlock-Free Routes

Given a link (u, v) and a flow $f_i(u, v) \geq 0$:

1) Check if there is a VC containing only flows that are already entangled with f_i. Once two flows share a VC somewhere, there is no advantage to assigning them to different VCs afterwards, and, if such a channel exists, it is allocated to f_i. Go to Step 5.
2) Look for empty VCs on the link; if one exists, assign it to $f_i(u, v)$, go to Step 5.
3) If some VC contains a flow entangled with $f_i(u, v)$, assign that VC to $f_i(u, v)$, go to Step 5.
4) If none of the criteria above apply, assign $f_i(u, v)$ to the VC with the fewest flows.
5) Update flow entanglement relationships to reflect the new assignment.

Repeat Step 1 through 5 for each flow at the link, and then move to the next link.

Fig. 10. Least sharing virtual channel allocation (LSVCA) for deadlock-free routes.

5 Related Work

A basic deterministic routing method is dimension order routing [7], which becomes XY routing in a 2D mesh. Necessary and sufficient conditions for deadlock-free deterministic routing were given in [7], assuming no false resource dependences. We use this condition to determine if a set of routes is deadlock free in our oblivious routing scheme.

ROMM [22] and Valiant [31] are classic oblivious routing algorithms, which are randomized in order to achieve better load distribution. In o1turn [28], Seo et al., show that simply balancing traffic between XY and YX routing can guarantee provable worst-case throughput. A weighted ordered toggle (WOT) algorithm that assumes two or more virtual channels [12] assigns XY and YX routes to source-destination pairs in a way that reduces the maximum network load for a given traffic pattern. The previous oblivious routing algorithms are either indifferent to the traffic pattern (DOR, ROMM, Valiant, o1turn) or limited to simple minimal paths (WOT). Here, we are concerned with optimizing throughput for specific applications utilizing both minimal and nonminimal paths. We compare our scheme to several oblivious algorithms in Section 6.

Classic adaptive routing schemes include the turn routing methods [13] and odd even routing [2]. In [15], a scheme that switches between deterministic and adaptive modes depending on the application is presented, where local FIFO information is used to adapt routes. Duato (e.g., [9], [10]) gives necessary and sufficient conditions for deadlock-free adaptive routing in wormhole networks. While our algorithms are not adaptive, as described in Section 4, we use the turn model to derive an acyclic channel dependence graph that drives our oblivious routing scheme. However, our scheme additionally allows *ad hoc* derivation of acyclic dependence graphs.

There has been significant effort in designing and utilizing Network-on-Chip (NoC) interconnect; see [1] for a recent survey. Many works on mapping of applications onto NoC architectures have considered the routing problem during the NoC design phase (e.g., [14], [21]). Our framework is significantly different from these works in its iterative use of shortest path algorithms on channel dependence graphs as opposed to the original network to avoid deadlock, and its applicability to standard router architectures. The NoC networks are designed and built for specific applications.

Given an application, a heuristic method to improve initial routes obtained using dimension order routing is presented in [33]. This method maintains deadlock freedom by checking to see if rerouting introduces cycles. Palesi et al. [24], [25] provide a framework and algorithms for application-specific bandwidth-aware deadlock-free *adaptive* routing. Given a set of source-destination pairs, cycles are broken in the CDG to minimize the impact on the average degree of adaptiveness. Bandwidth requirements are taken into account to spread traffic uniformly through the network. Towles et al. [30] give a multicommodity flow linear programming formulation for router algorithm design. When the linear program is optimized, deterministic algorithms that are worst-case, or average-case optimal, fall out as solutions. The routes generated can correspond to split flows. In our oblivious routing schemes, given any application, we break cycles in many different ways using the turn model or using *ad hoc* schemes, perform bandwidth-sensitive route selection on modified acyclic CDGs, and select the routes (and associated acyclic CDG) that best satisfy bandwidth constraints. Cho et al., describe bandwidth-aware routing for diastolic arrays [3] and avoid deadlock by assuming that each flow has its own private channel. Our approach is more general in that it can be used even in the case of a single virtual channel.

6 Results and Comparisons

This section evaluates the performance of our heuristic bandwidth-sensitive oblivious routing algorithms and static virtual channel allocation using synthetic and nonsynthetic traffic. Through simulation experiments, we compare BSOR and BSORM with DOR, ROMM [22], and Valiant [31]; and the performance of static VC allocation with dynamic allocation across various benchmarks.

6.1 Benchmarks

We use a set of standard synthetic traffic patterns, namely transpose, bit-complement, and shuffle, in our experiments, as well as an application benchmark corresponding to H.264 decoding, which has significantly different bandwidth demands for flows. The synthetic patterns provide basic comparisons between our routing scheme and other oblivious algorithms as they are widely used to evaluate routing algorithms. In the synthetic benchmarks, all flows have the same average bandwidth demands. H.264 is a set of flows that correspond to the traffic pattern of an H.264 decoder, with the bandwidths of the flows derived through profiling.

6.2 Results for Maximum Channel Load

We first present results on the maximum channel loads (MCL's) of various routes in Table 1. The $*$ sign indicates that the MCL value produced by BSOR or BSORM is minimal and equal to the MILP MCL on G_A. For BSOR, we used flow networks G_A's corresponding to 12 different acyclic CDGs D_A's; there are three different turn models, North-Last, West-First, and Negative-First, each with four rotations. We disallow 180 degree turns. In the case of BSORM, we had four runs, one for each rotation of the route set of Fig. 7b. For each benchmark, for both BSOR and BSORM, a *single* route corresponding to the lowest MCL shown in Table 1 was chosen and simulated. The network link capacity is set to 500 MB/second across all benchmarks.

TABLE 1
Comparison of Maximum Channel Load (MCL) in MB/Second Presented by Various Routing Algorithms

Traffic	XY	YX	ROMM	Valiant	BSOR	BSORM
transpose	175	175	200	175	75*	75*
bit-comp	100	100	400	200	100*	125
shuffle	100	100	150	200	75*	75*
H.264	214	365	336	352	124	174

6.3 Simulator Details

We use HORNET [18], a highly configurable, cycle-accurate network-on-chip simulator, to estimate the throughput and latency of each flow in the application for various oblivious routing algorithms. HORNET uses the ORION 2.0 [16] framework for its power estimation. The simulator models the router microarchitecture from Section 2.1. As discussed in Section 2, our routing scheme only requires minor changes in the router microarchitecture. Therefore, we assume an identical clock frequency for all routing algorithms. We use an 8×8 2D mesh network with one, two, four, or eight virtual channels per port. The simulator is configured to have a per-hop latency of one cycle, flit buffer size per VC of 16 flits, and link capacity of 1 flit/cycle. For each simulation, the network was warmed up for 20,000 cycles and then simulated for 100,000 cycles to collect statistics, which was enough for convergence.

6.4 Single Virtual Channel

Fig. 11 compares the BSOR Dijkstra algorithm to XY and YX for the four benchmarks. Varying the injection rate implies that the bandwidth demands change in absolute terms, but not in relative terms. Our algorithm outperforms existing oblivious routing algorithms in the transpose and shuffle benchmarks. XY and YX routes are ideal for the perfect symmetry in the bit-complement benchmark; BSOR converges to the same routes as in YX routing. In H.264, the BSOR algorithm performs better than DOR routes under moderate traffic load. Its load-balancing properties help to prevent bandwidth demands, assigned to a link, from reaching link capacity prematurely while large portions of the network remain unused or underutilized. However, when the network gets congested, throughput becomes unstable and drops. This is primarily due to unfair arbitration of flows at the physical link, where one flow blocks other flows on its path. This head-of-line blocking (HOL) is mitigated by using LSVCA on multiple virtual channels, as shown in Fig. 12a.

Fig. 12b shows how performance varies when the bandwidth demands change both in absolute and relative terms. For the example transpose, for the *same* set of routes as those used in Fig. 11a, we show results when the bandwidth of each individual flow changes by $\pm 10\%$ and $\pm 50\%$ in a random fashion. Thus, one bandwidth demand could be halved from the value that was used to compute the route, while another could increase by $1.5X$. BSOR continues to outperform the other algorithms since it spreads the load across the network better.

Figs. 13a and 13b show the packet latencies for the transpose and H.264 benchmarks for different injection rates. We define the *latency* of a packet to be the total amount of cycles spent in the network, from the emission of

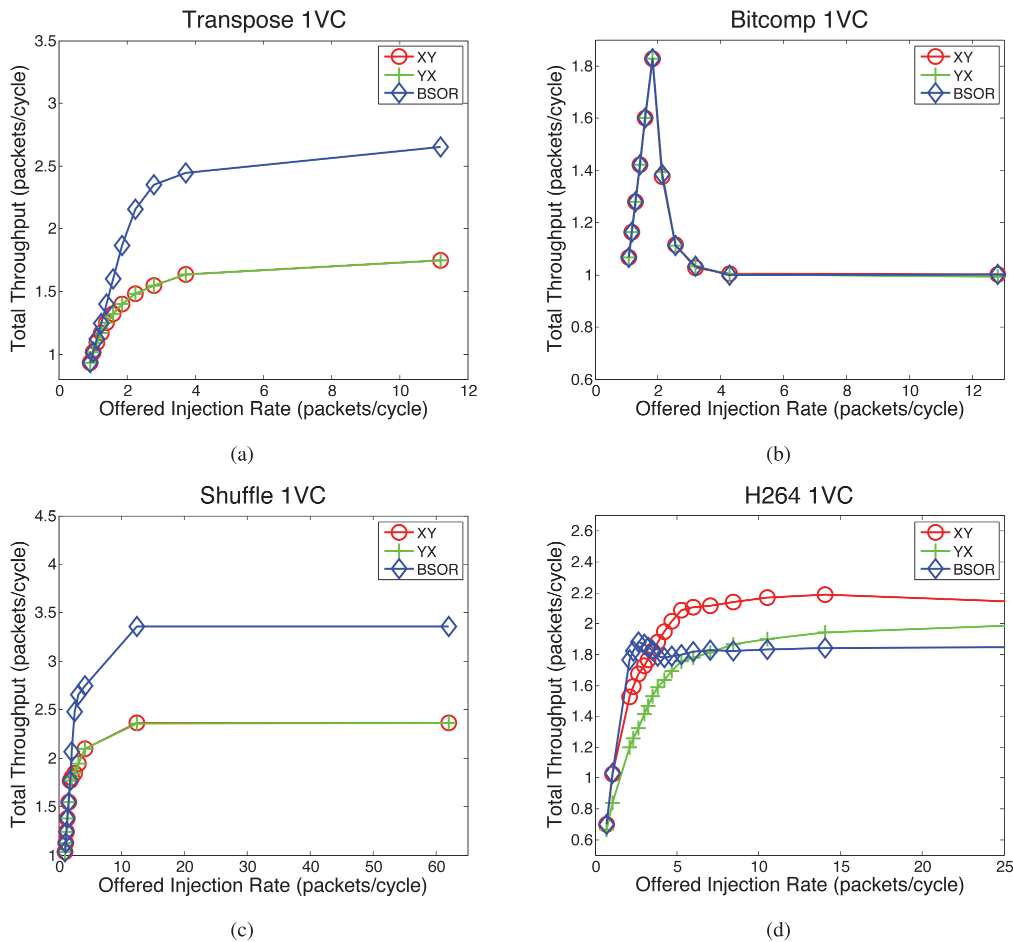


Fig. 11. Load-throughput graphs for benchmarks on a router with one virtual channel. Each graph shows the saturation throughput for various oblivious routing algorithms. (a) Transpose. (b) Bit-complement. (c) Shuffle. (d) H.264.

its header flit at the source, to the reception of its tail flit at its destination. The latency of a flow is simply the average latency of its packets. Fig. 13c illustrates the impact of VC configuration on packet latency. Throughout our experiments, the latency numbers follow the performance measured by the throughput.

Power estimation in HORNET is done through ORION 2.0, a power-performance simulator capable of providing both static and dynamic power characteristics for on-chip networks. The key takeaways from the power results are: 1) BSOR power consumption is comparable to DOR, as shown in Fig. 14, 2) For this class of oblivious routing algorithms, static link power has minimal impact on the overall system power consumption, and 3) the VC configuration of the router, which also dictates the crossbar size and arbitration scheme, is the dominant power factor, in terms of both static and dynamic power, as shown in Fig. 15.

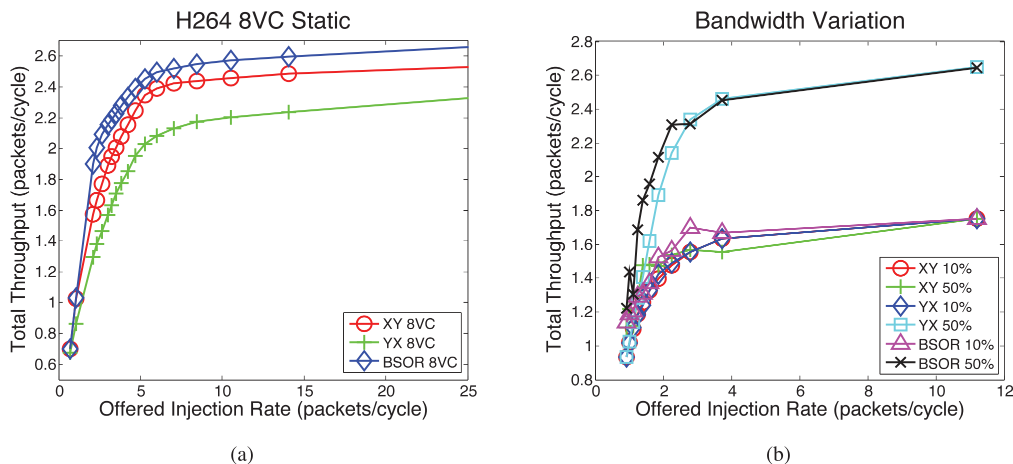
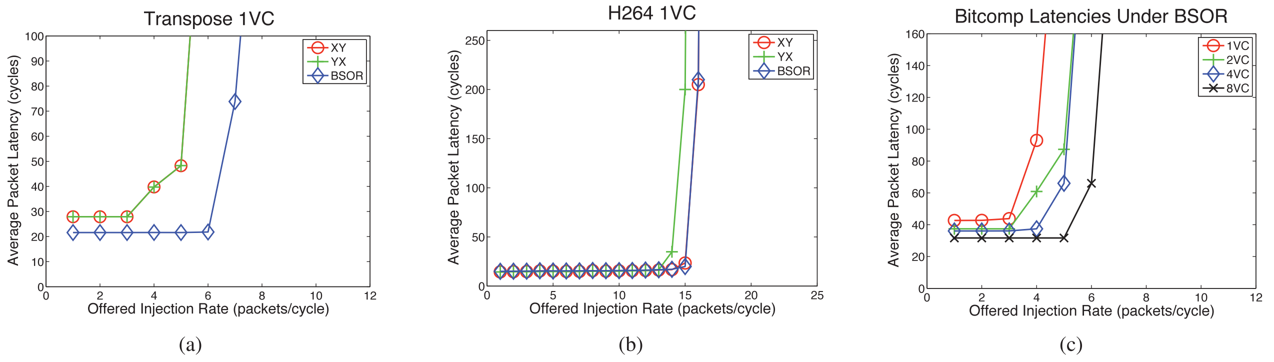
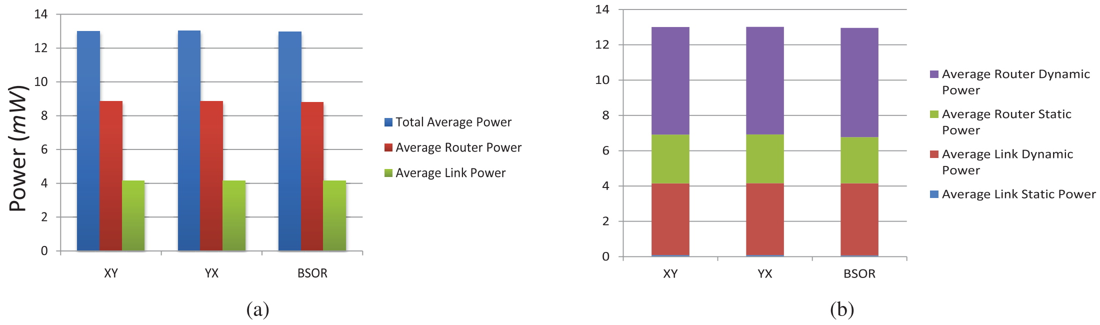


Fig. 12. (a) Load-throughput graphs for the H.264 benchmark using static VC allocation (with eight virtual channels). (b) Load-throughput graphs for the transpose benchmark (one virtual channel) when bandwidths change by $\pm 10\%$ and $\pm 50\%$ after route computation.



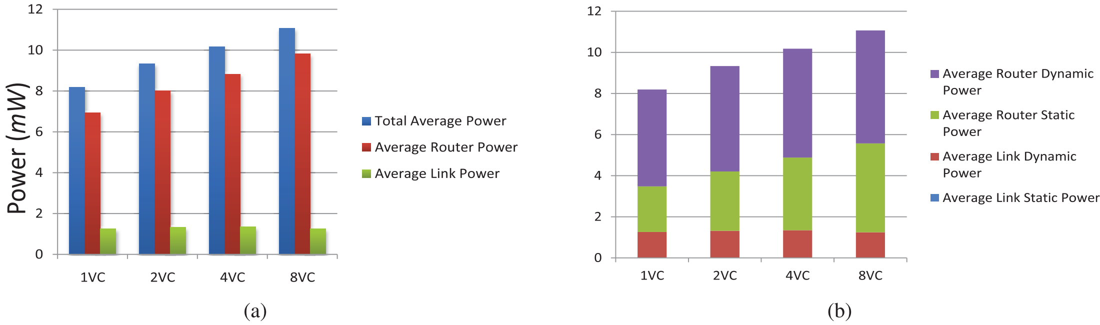
(a) (b) (c)

Fig. 13. Effects of routing algorithms and number of virtual channels on load-latency. (a) Transpose. (b) H.264. (c) Bitcomp with different VC configurations.



(a) (b)

Fig. 14. Effects of routing algorithms on power using shuffle benchmark with 1 VC. (a) Average total power. (b) Static and dynamic breakdown of the average power.



(a) (b)

Fig. 15. Effects of number of virtual channels on power using H.264 benchmark. (a) Average total power. (b) Static and dynamic breakdown of the average power.

6.5 Multiple Virtual Channels with Dynamic Allocation

We compare BSOR and BSORM with XY, YX, ROMM, and Valiant under dynamic virtual channel allocation with two virtual channels in Fig. 16. Note that ROMM and Valiant need to switch virtual channels in order to ensure deadlock freedom. BSORM performs better than BSOR for the transpose, shuffle, and H.264 benchmarks. For these benchmarks, BSORM provides better communication locality. Overall, across the various benchmarks, BSOR and BSORM show better performance than other oblivious routing algorithms. Further, their performance improvement over the other algorithms remains relatively consistent for virtual channels greater than two (not shown).

6.6 Multiple Virtual Channels with Static Allocation

For all these algorithms and across these benchmarks, static allocation performs as well or better than dynamic allocation for high injection rates by more effectively reducing head-of-line blocking effects. Figs. 17 and 18 compare static VC and dynamic VC allocations under various scenarios. Fig. 17 shows the performance of XY and YX routing with two VCs using static and dynamic VC allocation for the transpose and bit-complement benchmarks. Fig. 18 shows the performance of BSORM and XY under static and dynamic allocation for four VCs. LSVCA is used for all static VC allocations.

6.7 Discussion

BSOR and BSORM aim at finding the right tradeoff between locality and load balance. Valiant, for example, provides good load balancing to the network but at the expense of locality. In our experiments, Valiant has the longest average path length, and for applications with a large number of concurrent communications, having longer paths creates extra congestion which leads to a higher MCL and lower throughput. ROMM, an alternative to Valiant, retains locality in routing while providing some degree of load balancing. BSOR, on the other hand, provides near-optimal load

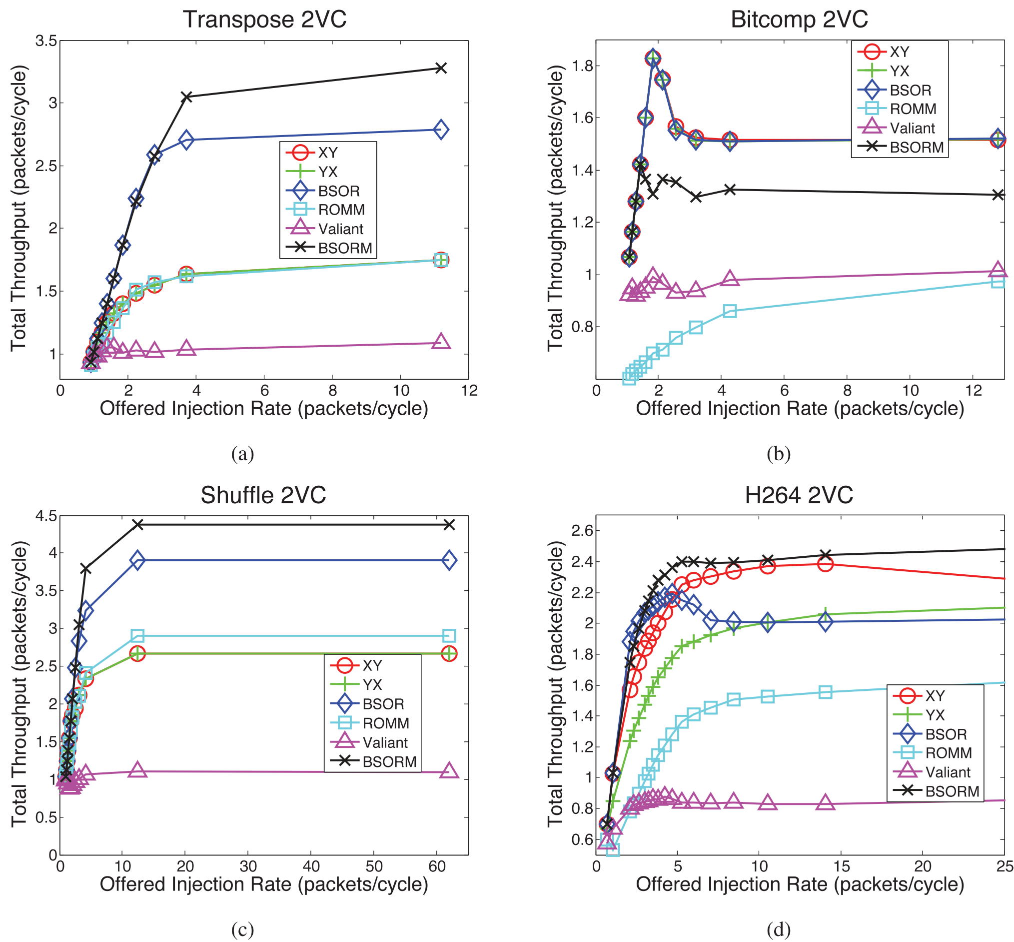


Fig. 16. Load-throughput graphs for benchmarks on a router with two virtual channels. Each graph shows the saturation throughput for various oblivious routing algorithms. (a) Transpose. (b) Bit-complement. (c) Shuffle. (d) H.264.

balancing for a given degree of locality. Depending on the application sensitivity to locality or load balance, different routes are generated. Fig. 19 shows the effect of BSOR on load distribution for the shuffle benchmark with random flow demands, where part of the traffic going through the bottleneck link is redirected to nearby links. Static virtual channel allocation, when the application communication characteristics are known, has proven to be very effective in mitigating head-of-line blocking. This is because it helps to prevent performance degradation associated with a single flow consuming multiple virtual channels and blocking other flows. Deterministic routing algorithms, like DOR, also tend to benefit from static virtual channel allocation, because on average, they have more sharers per link. Fig. 20 shows sharers per link seen in BSOR and *XY* for the shuffle benchmark with random flow demands.

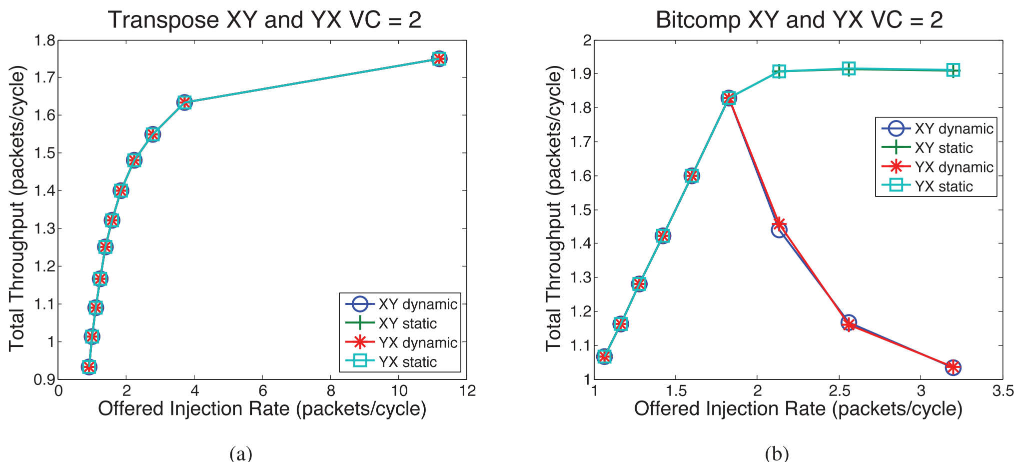


Fig. 17. Throughput for dimension order routing under static and dynamic allocation with two VCs.

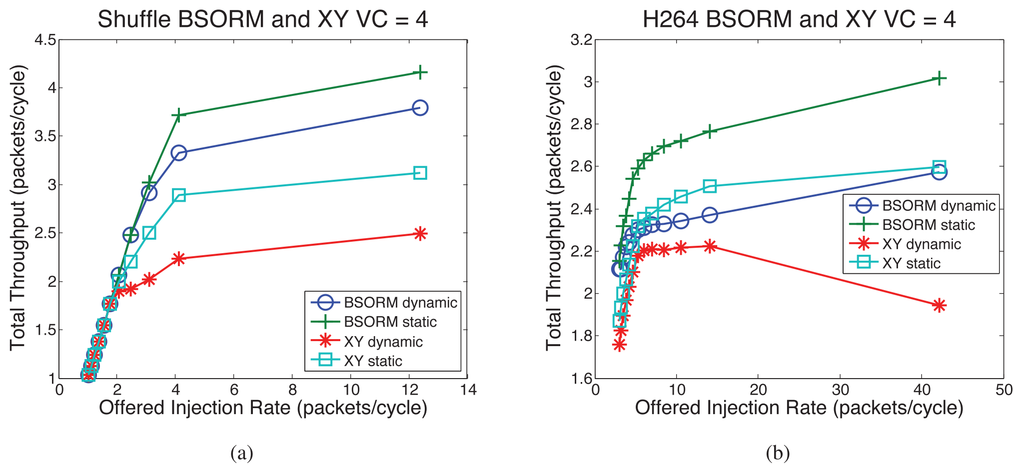


Fig. 18. Throughput for BSORM and XY under static and dynamic allocation with four VCs.

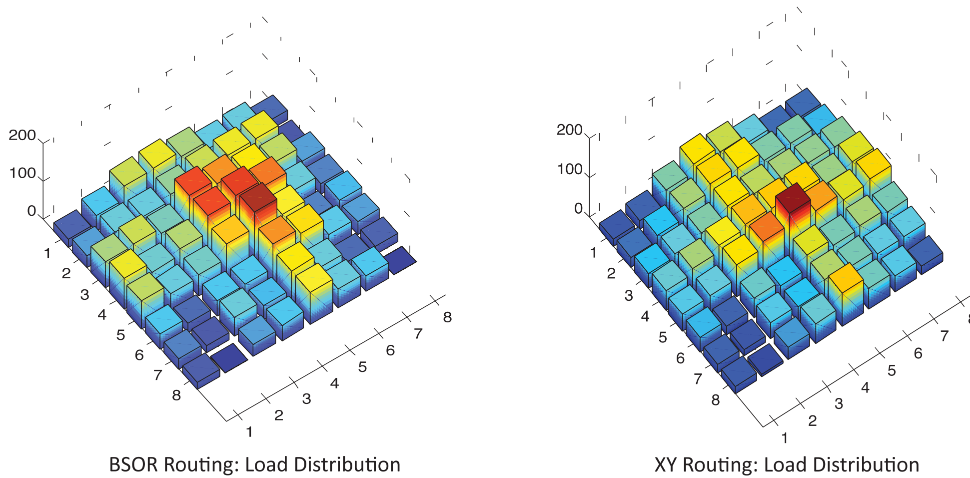


Fig. 19. BSOR and XY load distribution for random demand shuffle. XY suffers more from premature saturation.

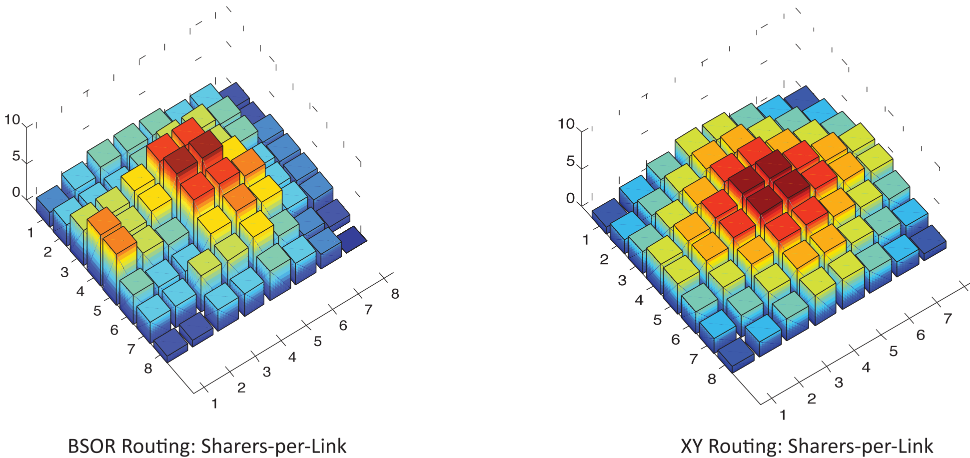


Fig. 20. BSOR and XY sharers-per-link. Static VC allocation helps both routing algorithms alleviate head-of-line blocking.

7 Conclusions

We have proposed an offline strategy to compute routes, based on knowledge of the application's data transfers, to arrive at an application-aware oblivious routing framework that does not require significant modification to standard routers. We have shown that estimates of the bandwidths demand of an application's data transfers can help improve application performance.

In the case of BSOR, a useful next step is a strategy for simultaneous acyclic CDG and route selection. We attempted to obtain a minimum channel load set of routes using the BSORM algorithm, without placing any restrictions on turns used, but placing restrictions on the minimality of the routes. It is worthwhile to investigate strategies that can eliminate the restriction of minimality, while ensuring deadlock freedom. We also examined the effects of static virtual channel allocation for oblivious

routing on the overall throughput. The main limitation of the application-aware routing framework is that we need some knowledge of the application. This does not have to necessarily be bandwidth demands, though we have focused on bandwidth in this paper. It could be knowledge of data transfers whose latency is critical to performance. These transfers can be forced to have minimal routes. Alternately, we can minimize the maximum number of flows sharing a link without knowing bandwidths. To handle bursty flows, we have proposed bandwidth-adaptive networks that contain adaptive bidirectional links and can improve the performance of conventional oblivious routing methods [4].

Acknowledgments

The authors would like to thank Derek Chiou, Joel Emer, Li-Shiuan Peh, Pengju Ren, Marten van Dijk, and David Wentzlaff for interesting discussions throughout the course of this work. We would like to acknowledge the support of Intel Corporation for providing some of the workstations used in conducting this research. This research is partially funded by NSF grant CCF-0905208.

References

[1] T. Bjerregaard and S. Mahadevan, "A Survey of Research and Practices of Network-on-Chip," *ACM Computing Surveys,* vol. 38, no. 1, article 1, 2006.

[2] G.-M. Chiu, "The Odd-Even Turn Model for Adaptive Routing," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 7, pp. 729-738, July 2000.

[3] M.H. Cho, C.-C. Cheng, M. Kinsy, G.E. Suh, and S. Devadas, "Diastolic Arrays: Throughput-Driven Reconfigurable Computing," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '08),* Nov. 2008.

[4] M.H. Cho, M. Lis, K.S. Shim, M. Kinsy, T. Wen, and S. Devadas, "Oblivious Routing in On-Chip Bandwidth-Adaptive Networks," *Proc. 18th Int'l Conf. Parallel Architecture and Compilation Techniques (PACT '09),* Sept. 2009.

[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* MIT Press, 2001.

[6] W.J. Dally, P.P. Carvey, and L.R. Dennison, "The Avici Terabit Switch/Router," *Proc. Sixth Symp. Hot Interconnects,* pp. 41-50, Aug. 1998.

[7] W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers,* vol. 36, no. 5, pp. 547-553, May 1987.

[8] W.J. Dally and B. Towles, *Principles and Practices of Interconnection Networks.* Morgan Kaufmann, 2003.

[9] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 12, pp. 1320-1331, Dec. 1993.

[10] J. Duato, "A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, no. 10, pp. 1055-1067, Oct. 1995.

[11] M. Galles, "Scalable Pipelined Interconnect for Distributed Endpoint Routing: The SGI SPIDER Chip," *Proc. Symp. Hot Interconnects,* pp. 141-146, Aug. 1996.

[12] R. Gindin, I. Cidon, and I. Keidar, "NoC-Based FPGA: Architecture and Routing," *Proc. First Int'l Symp. Networks-on-Chips (NOCS),* pp. 253-264, 2007.

[13] C.J. Glass and L.M. Ni, "The Turn Model for Adaptive Routing," *J. ACM,* vol. 41, no. 5, pp. 874-902, Sept. 1994.

[14] J. Hu and R. Marculescu, "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures," *Proc. Design, Automation and Test in Europe Conf.,* 2003.

[15] J. Hu and R. Marculescu, "DyAD: Smart Routing for Networks on Chip," *Proc. Design Automation Conf.,* June 2004.

[16] A.B. Kahng, B. Li, L.S. Peh, and K. Samadi ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration, 2009.

[17] J.M. Kleinberg, "Approximation Algorithms for Disjoint Paths Problems," PhD thesis, Massachusetts Inst. of Technology, 1996.

[18] M. Lis, P. Ren, M.H. Cho, K.S. Shim, C.W. Fletcher, O. Khan, and S. Devadas, "Scalable, Accurate Multicore Simulation in the 1000-Core Era," *Proc. IEEE Int'l Symmp. Performance Analysis of Systems and Software (ISPASS '11),* pp. 175-185, 2011.

[19] R.D. Mullins, A.F. West, and S.W. Moore, "Low-Latency Virtual-Channel Routers for On-Chip Networks," *Proc. 31st Ann. Int'l Symp. Computer Architecture (ISCA '04),* pp. 188-197, 2004.

[20] S. Murali, D. Atienz, L. Benini, and G.D. Micheli, "A Method for Routing Packets Across Multiple Paths in NoCs with In-Order Delivery and Fault-Tolerance Gaurantees," *VLSI Design,* 2007.

[21] S. Murali and G.D. Micheli, "SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs," *Proc. 41st Ann. Conf. Design Automation (DAC '04),* pp. 914-919, 2004.

[22] T. Nesson and S. Lennart Johnsson, "ROMM Routing on Mesh and Torus Networks," *Proc. Seventh Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA '95),* pp. 275-287, 1995.

[23] L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer,* vol. 26, no. 2, pp. 62-76, Feb. 1993.

[24] M. Palesi, R. Holsmark, S. Kumar, and V. Catania, "A Methodology for Design of Application Specific Deadlock-Free Routing Algorithms for NoC Systems," *Proc. Fourth Int'l Conf. Hardware/Software Codesign and System Synthesis (CODES+ISSS '06),* Oct. 2006.

[25] M. Palesi, G. Longo, S. Signorino, R. Holsmark, S. Kumar, and V. Catania, "Design of Bandwidth Aware and Congestion Avoiding Efficient Routing Algorithms for Networks-on-Chip Platforms," *Proc. ACM/IEEE Int'l Symp. Networks-on-Chip (NOCS),* pp. 97-106, 2008.

[26] L.-S. Peh and W.J. Dally, "A Delay Model and Speculative Architecture for Pipelined Routers," *Proc. Int'l Symp. High-Performance Computer Architecture (HPCA),* pp. 255-266, Jan. 2001.

[27] L. Schwiebert, "Deadlock-Free Oblivious Wormhole Routing with Cyclic Dependencies," *Proc. Ninth Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA '97),* pp. 149-158, 1997.

[28] D. Seo, A. Ali, W.-T. Lim, N. Rafique, and M. Thottethodi, "Near-Optimal Worst-Case Throughput Routing for Two-Dimensional Mesh Networks," *Proc. 32nd Ann. Int'l Symp. Computer Architecture (ISCA '05),* pp. 432-443, 2005.

[29] C.B. Stunkel, D.G. Shea, D.G. Grice, P.H. Hochschild, and M. Tsao, "The SP1 High-Performance Switch," *Proc. Scalable High Performance Computing Conf.,* pp. 150-157, May 1994.

[30] B. Towles, W.J. Dally, and S. Boyd, "Throughput-Centric Routing Algorithm Design," *Proc. 15th Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA '03),* pp. 200-209, 2003.

[31] L.G. Valiant and G.J. Brebner, "Universal Schemes for Parallel Communication," *Proc. 13th Ann. ACM Symp. Theory of Computing (STOC '81),* pp. 263-277, 1981.

[32] K. Walkowiak, "New Algorithms for the Unsplittable Flow Problem," *Proc. Int'l Conf. Computational Science and Its Applications (ICCSA),* pp. 1101-1110, 2006.

[33] X. Zhong and V. Mary, "Application-Specific Deadlock Free Wormhole Routing on Multicomputers," *Proc. Fourth Int'l PARLE Conf. Parallel Architectures and Languages Europe (PARLE '92),* pp. 193-208, 1992.

Michel A. Kinsy received the MS degree in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), the BSE degree in computer systems engineering, and the BS in computer science, both from Arizona State University. He is working toward the PhD degree at MIT. His current research interests include high-performance distributed computing platforms: reconfigurable many-core computer architectures, networks-on-chip (NoCs) routing, and domain-specific multicore system design.

Myong Hyon Cho received the bachelor's degree from Seoul National University, and the master's degree from Massachusetts Institute of Technology. He is currently working toward the PhD degree in the Department of Electrical Engineering and Computer Science at Massachusetts Institute of Technology. His current research interests include many-core computer architecture, memory subsystem, and on-chip network.

Keun Sup Shim received the BS degree in electrical engineering from KAIST, South Korea, in 2006 and the MS degree in electrical engineering and computer science from MIT in 2010. He is currently working toward the PhD degree in electrical engineering and computer science in MIT. His research interests include high-performance computer architecture, scalable many-core designs, and on-chip networks.

Mieszko Lis received the bachelor's and master's degrees from MIT. He is working toward the PhD degree at the Massachusetts Institute of Technology in computer architecture and computational biology. He accumulated extensive industry experience as a cofounder of a fabless semiconductor company and a high-level hardware synthesis startup. His current research interests include massive-scale multicores and the advanced coherent memory hierarchies required to support them.

G. Edward Suh received the BS degree in electrical engineering from Seoul National University in 1999, and SM and PhD degrees in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT) in 2001 and 2005, respectively. He is currently an assistant professor in the School of Electrical and Computer Engineering at Cornell University, where he leads the Trustworthy Systems Group in the Computer Systems Laboratory. He is a recipient of an NSF CAREER Award, an Air Force Office of Scientific Research (AFOSR) Young Investigator Program Award, and an Army Research Office (ARO) Young Investigator Program Award. His current research includes developing architectural techniques to improve security, reliability, and correctness of future computing systems. He is a member of the IEEE.

Srinivas Devadas is a professor of electrical engineering and computer science at the Massachusetts Institute of Technology (MIT), and has been on the faculty of MIT since 1988. He served as the associate head with responsibility for computer science from 2005-2011. He has worked in the areas of computer-aided design, testing, formal verification, compilers for embedded processors, computer architecture, computer security, and computational biology and has coauthored numerous papers and books in these areas. He has been a fellow of the IEEE since 1998.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.