Application-Aware Deadlock-Free Oblivious Routing

Michel Kinsy, Myong Hyon Cho, Tina Wen, Edward Suh†, Marten van Dijk, Srinivas Devadas
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA
†Department of ECE
Cornell University
Ithaca, NY

ABSTRACT

Conventional oblivious routing algorithms are either not application-aware or assume that each flow has its own private channel to ensure deadlock avoidance. We present a framework for application-aware routing that assures deadlock-freedom under one or more channels by forcing routes to conform to an acyclic channel dependence graph. Arbitrary minimal routes can be made deadlock-free through appropriate static channel allocation when two or more channels are available. 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. The heuristic algorithm is calibrated using the MILP algorithm and evaluated on a number of benchmarks through detailed network simulation. 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.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Network communications

General Terms

Algorithms, Design, Performance

Keywords

Systems-On-Chip, On-chip interconnection networks, Oblivious Routing

1. INTRODUCTION

Routers can be generally classified into oblivious and adaptive [29]. In oblivious routing, the path is completely determined by the source and the destination address. 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 from others, oblivious routing such as dimension order routing [8] 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, especially when flows have different bandwidth demands, because routes are not adjusted for different applications.

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 paper is on bandwidth-sensitive oblivious routing 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 run-time as processing elements are loaded with the computation code. This approach can achieve better thoughput than traditional oblivious routing algorithms because routes are optimized based on the global knowledge of bandwidth demands. At the same time, the router remains simple because the routes are configured statically and do not change at run-time.

Application-aware oblivious routing will be particularly suitable for long-running applications with predictable communication patterns. For example, the approach is suitable for co-processing platforms such as reconfigurable hardware, where processing elements and their interconnection network can be configured much like an FPGA to speed up a computationally-intensive task such as video compression, processor simulation, or rendering. In reconfigurable computing, a computation is spatially partitioned into processing elements (PEs) and the network traffic pattern remains relatively static as each PE performs a fixed task. Our studies on

ISCA'09, June 20–24, 2009, Austin, Texas, USA.

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. Various algorithms for bandwidth-sensitive routing are the subject of Sections 4 and 5. Related work is summarized in Section 6. Section 7 compares benchmark performance using our routing schemes to existing deterministic and oblivious routing algorithms. Section 8 concludes the paper.

2. ROUTER ARCHITECTURE

This section discusses the impact of our oblivious routing technique 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 2-D 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

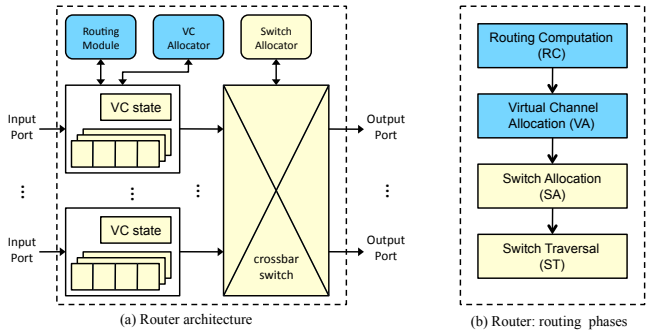


Figure 1: Typical virtual-channel router architecture. The dark blue indicates that the modules and routing step may be modified for our approach.

Figure 1 illustrates a typical virtual-channel router architecture and its operation [9, 25, 34]. As shown in the figure, the datapath 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 datapath, 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 operation 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).

Strategy	Routing mechanics
DOR, ROMM	Algorithmic: fixed logic
BSOR	Table-based: source/node-table routing
BSORM	Table-based: source/node-table routing

Table 1: Router architecture designs for routing algorithms.

For existing oblivious routing algorithms such as Dimension Ordered Routing (DOR) [8], ROMM [28], Valiant [41], and o1turn [36], the next hop of a packet can be easily computed at each router node based on the packet's destination. Moreover, these algorithms are fixed and commonly used for all types of applications and traffic patterns. As a result, traditional oblivious routing algorithms are implemented as dedicated logic in the RC phase of each router. For these routing algorithms, the RC step is quite simple and the router frequency is typically dominated by the VA step [34].

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 datapath that is described above. The main change in our routing architecture is in its routing module, which is summarized in Table 1, where BSOR and BSORM correspond to the algorithms of Section 4 and 5, respectively. 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 (cf. Section 2.2.1). This single change is sufficient because our routing algorithms ensure that there is no cyclic dependence in routes either through route selection (cf. Section 4) or through static channel allocation (cf. Section 5). We next discuss the details of the modification.

2.2.1 Programmable Routing

Our routing technique determines the routes for each flow based on an application's bandwidth requirements as well as its source and destination nodes. Additionally, to maximize the throughput, our routing algorithms can utilize any path from the source to the destination; routes may be either minimal or non-minimal. Therefore, 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 re-programmed 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 pre-computed 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

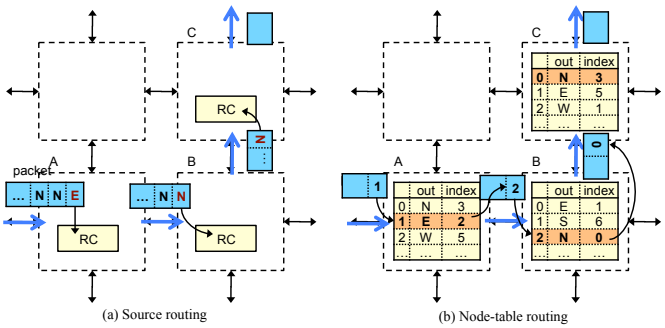


Figure 2: The table-based routing architecture. (a) Source routing. (b) Node-table routing.

path can determine the output port simply by looking up the routing flits. Figure 2(a) illustrates source routing where a packet is routed through node A, B, and C. The route corresponds to East, North, and North, which is reflected in the routing flits in the packet.

The source routing approach simplifies the router design because the routing phase (RC step) in the router now only needs to read the output port from the flit without any computation. Effectively, source routing eliminates the routing step in the router and can potentially reduce the number of pipeline stages, or clock period in case of an unpipelined router. In fact, thanks to its speed and simplicity, source routing has been widely used in many router designs including the IBM SP1 [39] and SP2 [38] and the Avici TSR [7]. On the other hand, source routing results in longer packets containing routing flits as compared to the case where the route is computed for each hop.

Instead of carrying the entire route with every packet, the nodes along the path can be programmed with routing information for relevant flows. In this *node-table* routing approach, 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. To set up the route, our routing algorithm computes a route for each flow and configures the routing tables accordingly. 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.

Figure 2(b) 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.

The router architecture for node-table routing replaces the fixed logic in the RC phase of the baseline router with a table look-up. While the table look-up can take longer than evaluating the routing logic for simple deterministic routing such as DOR, it will not change how fast a router can operate because the router's clock frequency is most often dominated by other routing phases or external factors such as a processor's frequency. A previous study shows that the latency of a pipelined virtual-channel router is dominated by virtual-channel allocation, which takes 15-20 FO4 [34]. Even if we conservatively assume that each routing table has 256 entries (256 flows), the table only takes a couple of KB; an entry needs 2 bits to represent the output port in a 2-D mesh and 8 bits for the next table index (256 entries). Therefore, a routing table will be easily accessible within a single cycle without impacting the clock frequency.

In practice, both table-based routing techniques place a restriction on the maximum number of flows that can be supported depending on the size of a routing table. In source routing, flows with an identical source-destination pair will have to share the same route unless the routing table has multiple entries for each destination. Similarly, in node-table routing, the size of each routing table limits the number of flows that can be routed through a node. Our routing algorithm can include restrictions enforced by the router hardware.

We have described two routing module designs, namely source routing and node-table routing, that can support bandwidth-sensitive oblivious routing. Both routing methods are widely known and have been implemented in multiple routers [39, 38, 7, 13]. 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 Static Virtual-Channel Allocation

If the routing algorithm statically allocates a virtual channel to each flow, the VC allocation step of the baseline router can be simplified. In this case, instead of dynamically allocating virtual channels using arbiters, the routing algorithm specifies a virtual channel at each link along the path for a flow. Then, the router can obtain the static allocation in the same way that it obtains the static route for the packet. In the source routing approach, a packet carries its virtual channel number for each hop along with its route. In the node-table routing approach, an entry in the routing table is augmented to have the virtual channel number for the flow. Therefore, the router can obtain both the output port and the next VC number at the end of the routing (RC) phase. Now, the primary complexity in the VA phase is in the arbitration amongst packets; two or more packets may request the same virtual channel simultaneously, and arbitration is required to determine which packet will be transferred first. This requires $P \cdot V$ to 1 arbitration for each virtual channel where packets from P physical channels with V virtual channels each vie for the same virtual channel. On the other hand, dynamic allocation requires $P \cdot V$ to V arbitration. A previous study indicates that the $P \cdot V$ to 1 arbitration is about 20% faster than $P \cdot V$ to V arbitration (11.0 FO4 vs. 13.3 FO4 with 8 VCs) [34].

Static allocation can potentially result in worse utilization of available virtual channels because it does not consider dynamic behavior. For example, statically allocating VC0 to flow A and VC1 to flow B may not be efficient when, for example, flow A is idle, because then different packets in flow B can potentially use both virtual channels. On the other hand, static allocation may enhance throughput through the separation or isolation of flows.

When virtual channels are dynamically allocated, we require the set of routes to conform to an acyclic CDG, which, for example, could correspond to a turn model [15] (cf. Section 3 and 4). Static allocation enables a routing algorithm to potentially choose a different set of routes compared to the dynamic allocation case by separating flows (cf. Section 5.3).

3. DEADLOCK-FREE ROUTING

3.1 Definitions and Framework

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

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. 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 (CDG) $D(V', E')$ 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.*

Figure 3 shows a bi-directional 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.

Application-aware oblivious routing follows the framework of Figure 4.

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.

Figure 4: Offline Application-Aware Oblivious Routing Framework

3.2 Creating Acyclic Channel Dependence Graphs

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).

Our framework assumes that the underlying network has been made deadlock-free. For example, a torus is made deadlock-free by applying dateline classes to each dimension [9]. Nothing needs to be done for other networks, such as meshes.

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 [8]. This is also a necessary condition provided false resource dependences do not exist [35]. 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 [15] provides a few systematic ways. 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 Figure 3(b), and produces the acyclic CDG of Figure 5(a). Cycles can also be broken in an *ad hoc* or random fashion as shown in Figure 5(b). 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.

3.3 Deriving a Flow Graph from an Acyclic CDG

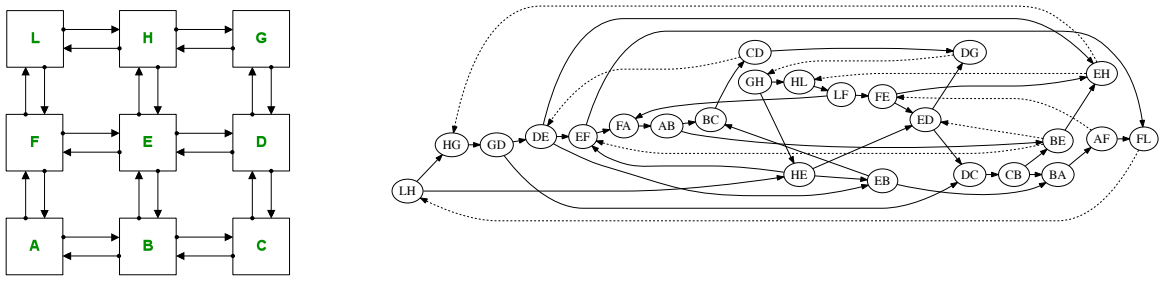
Given source and destination network nodes s_i and t_i respectively, for each flow i, we derive a flow graph or network G_A from an acyclic CDG D_A. We can then run our route selection algorithm on G_A, to find the "best" routes for the flows (cf. Section 4). This will have the effect of running route 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 Figure 3(a), 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 Figure 3(a), 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, we want to find the best route in G_A starting with either AB or AF and ending at either FL or HL.

Figure 5(c) shows a flow network derived from the acyclic CDG of Figure 5(b), given source-destination pairs $A,\ L$ and $E,\ G$.

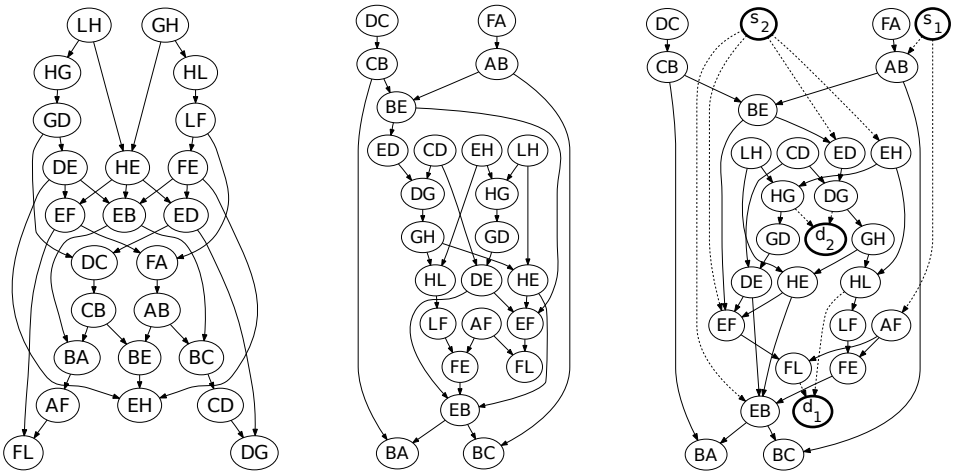
4. BANDWIDTH-SENSITIVE OBLIVIOUS ROUTING (BSOR) ALGORITHMS

It is widely known that a linear programming formulation can determine a lower bound on the maximum channel load [9, 40]. However, the routes given by linear programming may not be realizable on standard routers since a packet flow may need 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 [22], 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 [6] in Step 3 of Figure 4 to heuristically select good routes for large problems (cf. Section 4.2).



(a) (b)

Figure 3: (a) 3×3 mesh (b) Channel Dependence Graph without 180° turns.



(a) (b) (c)

Figure 5: (a) North-Last Routing CDG without 180° turns: 32 edges removed. (b) Ad Hoc CDG without 180° turns: 36 edges removed. (c) Flow network from acyclic CDG of (b) with source-destination pairs $A,\ L$ and $E,\ G$.

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 :

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

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

Unsplittable flow :

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

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

Hop Count :

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

and maximizes the total throughput, given as

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

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

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

or minimizes the maximum channel load:

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

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 the minimal path length between s_i and t_i. This will imply that only minimal paths will be considered. hop_i should be incremented by 2 or more to allow for non-minimal routing. 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 $\min_{1 \leq i \leq k} \frac{g_i}{d_i}$ as in Eqn. 2. We focus on finding the minimum maximum channel load (MCL) as in Eqn. 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., [26]).

4.2 Weighted Shortest-Path-Based Algorithm

We select a route for each flow that heuristically minimizes the number of congested links using Dijkstra's weighted shortest-path algorithm. 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. 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 CSPF metric described by Walkowiak [42]. The weighting function $w(e) = \frac{1}{\tilde{c}(e) - d_i}$, except if $\tilde{c}(e) \leq d_i$, 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). Figure 5(c) showed a flow network derived from the acyclic CDG of Figure 5(b). 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 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 amongst all the computed routes, across all CDGs. We also compute the congestion corresponding to the product of the average excess bandwidth demand over all links times the average number of flows competing for each link, and use the congestion as a tiebreaker when two sets of routes have the same MCL.

4.3 Multiple Virtual Channels

In modern routers, there are many virtual channels per link. Many virtual channel routers dynamically allocate virtual channels to packets in flows. As in the single channel case, cycles in the CDG imply that the network might deadlock. It is possible to have cycles in the CDG if an escape path is provided [11, 12]; this implies adaptive routing and we do not consider that in this paper. Therefore, our routing strategy remains unchanged for dynamic allocation of virtual channels; we guarantee routes that conform to an acyclic CDG. However, if we can statically allocate channels to flows, we can choose a more diverse set of routes, as we describe in the next section.

5. STATIC VIRTUAL CHANNEL ALLOCATION

We assume the router design described in Section 2 and that static allocation of virtual channels is supported as described in Section 2.3. We show how deadlock freedom can be assured through static virtual channel allocation subsequent to route selection. The material in this section first appeared in [37].

5.1 Turn Model

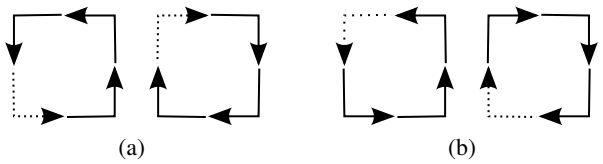


Figure 6: (a) Turns allowed (solid) and disallowed (dotted) under the West-First turn model (b) Turns allowed and disallowed under the North-Last turn model.

The turn model [15] is a systematic way of generating deadlock-free routes, as mentioned earlier. Figure 6 shows two different turn models that can be used in a 2-dimensional mesh. Each model disallows two out of eight turns. If a set of routes conform to one of the turn models, then deadlock freedom is assured with any number of virtual channels, and we use that in our framework of Section 3. The third turn model Negative-First is not shown.[1]

5.2 Deadlock-Free Minimal Routing with 2 virtual channels

We now show how any set of *minimal* routes produced using any routing method can be made deadlock-free through appropriate static virtual channel allocation. Our argument for deadlock freedom invokes the turn models of Figure 6. An arbitrary set of minimal routes may cause deadlock, since they do not necessarily conform to a particular acyclic CDG or turn model. However, if the number of available virtual channels is ≥ 2 we can perform a

[1]We have ignored the Negative-First turn model because it does not induce a flow partition (and a resultant channel allocation strategy) in combination with either of the other two turn models (cf. Theorem 1). This is true even when rotations are used.

static virtual channel assignment that ensures deadlock freedom by partitioning the flows across 2 (or more) virtual channels.

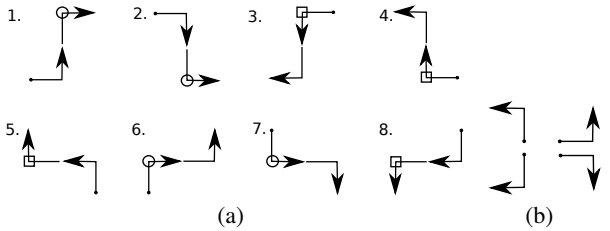


Figure 7: (a) The eight different two-turn minimal routes on a 2-dimensional mesh. (b) The four (out of a possible eight) different one-turn routes on a 2-dimensional mesh that conform to both the West-First and North-Last turn model.

THEOREM 1. *Given a router with $\geq$ 2 virtual channels, and an arbitrary set of routes over an $n \times n$ mesh, where each route is minimal, it is possible to statically allocate virtual channels to each flow to ensure deadlock freedom.*

Proof: Without loss of generality consider the case of 2 virtual channels. Figure 7(a) shows the eight possible minimal routes that use two different turns each. Of course, minimal routes that use a single turn or having no turns may also be included in the given arbitrary set of routes, but these can be ignored as special cases of the two-turn routes for the subsequent analysis. Looking at Figure 6, 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 illustrated by the boxes on the violating turns), and 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 on the violating turns). Therefore, given an arbitrary set of routes, we can partition the routes into two sets: the first set conforms to the West-First turn model, and the second to the North-Last turn model. Note that the four one-turn minimal routes shown in Figure 7(b) can be placed in either set, as can routes with zero turns. The four other one-turn routes (not shown) will be forced to one of the sets. If we assign virtual channel 1 to the first set and virtual channel 0 to the second set, we are assured freedom from deadlock. □

The proof of Theorem 1 points us toward a static virtual channel allocation strategy. We derive minimal routes using an MILP strategy or using a Dijkstra-based algorithm as we will describe in Section 5.3. Given a route for each flow, we create three sets of flows:

1. Flows with two-turn and single-turn routes that conform to the West-First turn model,
2. Flows with two-turn and single-turn routes that conform to the North-Last turn model, and
3. Flows with single-turn or no-turn routes that conform to both.

We simply assign the flows in the third set to either of the first two sets, appropriately balancing the bandwidths and number of flows across the two sets. For each flow in the third set,

1. the flow is assigned to the set which has fewer flows that share links with the flow; however,
2. if the number of flows that share links with the given flow is same for both sets, the flow is assigned to the set with fewer flows.

We will not describe the allocation strategy for more than 2 virtual channels; it is described in [37]. Our experiments in this paper primarily focus on 1 and 2 virtual channels.

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

The BSORM scheme works directly on the flow graph $G(V, E)$ corresponding to the network, *not* the flow network G_A derived from an acyclic CDG D_A as do the BSOR algorithms of Section 4. We do not need to constrain the routes in BSORM to conform to an acyclic CDG, but we require them to be minimal.

In the MILP formulation of Section 4.1, the hop count constraints are appropriately set to only allow minimal routing. Note that the MILP is formulated over G and not G_A.

In the Dijkstra formulation, we select a minimal route for each flow that heuristically minimizes the maximum channel load using Dijkstra's weighted shortest-path algorithm in a similar manner to Section 4.2. We elaborate below.

We run Dijkstra on a weighted version of G, deriving the weights from the residual capacities of each link as before. Dijkstra finds a minimum-weight path for a chosen flow i from an s_i to a t_i. The algorithm we use also keeps track of the number of hops and finds the minimum-weight path with minimum hop count. (Given our weight function, it is possible that the smallest weight path is non-minimal, but the algorithm will not generate such a path.) We check to see if the minimum-weight path can be replaced by one of the XY/YX routes of Figure 7(b). This replacement is made only if the XY/YX routes have minimum weight. This is done to minimize the number of turns in the selected routes and to give greater freedom to the flow partitioning step. Then, the weights are updated, and a new flow is selected to be routed. This continues until all flows are routed.

6. RELATED WORK

6.1 Routing and Bandwidth Allocation

A basic deterministic routing method is dimension-ordered routing (DOR) [8] which becomes XY routing in a 2-D mesh. Necessary and sufficient conditions for deadlock-free deterministic routing were given in [8] 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 [28] and Valiant [41] are classic oblivious routing algorithms, which are randomized in order to achieve better load distribution. In o1turn [36], 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 2 or more virtual channels [14] 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 non-minimal paths. We compare our scheme to several oblivious algorithms in Section 7.

Classic adaptive routing schemes include the turn routing methods [15] and odd even routing [3]. In [19] 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., [11, 12]) gives necessary and sufficient conditions for 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., [18], [27], [17]). 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 [43]. This method maintains deadlock freedom by checking to see if rerouting introduces cycles. Palesi *et al* [31, 32] 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* [40] 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 *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 [4] 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.

6.2 Virtual Channels and Router Design

Dally's virtual channels [10] allocate buffer space for virtual channels in a decoupled way from bandwidth allocation. iWarp [16] implemented virtual channels across single links. Many designs of virtual channel routers have been proposed (e.g., [25], [2], [21], [30]). Recently, express virtual channels have been proposed which skip routers along multiple-hop paths to enhance performance in a dynamic routing scheme [23]. Support for multicast channels has been proposed [20]. Our virtual channel router design is fairly standard (cf. Section 2). however, in one realization, virtual channels are allocated to flows statically, rather than dynamically.

6.3 Network Reconfiguration and Adaptivity

In this paper, we assume that the network is reconfigured prior to running the application. It is possible to integrate dynamic reconfiguration methods (e.g., [24], [33]) into the network architecture at the cost of increased hardware complexity. The new routes should satisfy deadlock freedom properties.

7. RESULTS AND COMPARISONS

This section evaluates the performance of our heuristic bandwidth-sensitive oblivious routing algorithms BSOR and BSORM. Through simulation experiments, we compare our routing scheme with dimension-order routing (DOR), and with routing schemes such as ROMM [28], and Valiant [41].

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

Table 2: Comparison of Maximum Channel Load (MCL) in MB/second presented by various routing algorithms.

7.1 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.

7.2 Results for Maximum Channel Load

We first present results on the maximum channel loads (MCL's) of various routes in Table 2.

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 4 rotations. We disallow 180° turns. For each benchmark, a *single* route corresponding to the lowest MCL and congestion (cf. Section 4.2) was chosen and simulated; this route's MCL is reported in Table 2 where a * indicates that the value is minimum as determined by the MILP on G_A. For BSORM, we use the original flow network G and the Dijkstra algorithm of Section 5.3. BSORM is run 4 times; once for each rotation of the route set of Figure 7(b). The only difference in these runs is that once a minimum-weight path is obtained for a flow, we check to see if it can be replaced by an equivalent minimum-weight single-turn path in the (rotated) set. We choose a single routing for each benchmark, corresponding to the smallest MCL across the 4 runs. Again, a * value indicates that the value is minimum as determined by the MILP on G with minimum hop constraints. Route selection requires on the order of a minute for these benchmarks.

7.3 Simulator Details

A cycle-accurate network simulator is used to estimate the throughput of each flow in the application for various oblivious routing algorithms. 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 2-D mesh network with 1, 2, 4 or 8 virtual channels per port. The simulator is configured to have a per-hop latency of 1 cycle, and the flit buffer size per VC of 16 flits. 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.

7.4 Single Virtual Channel

Figure 8 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 algo-

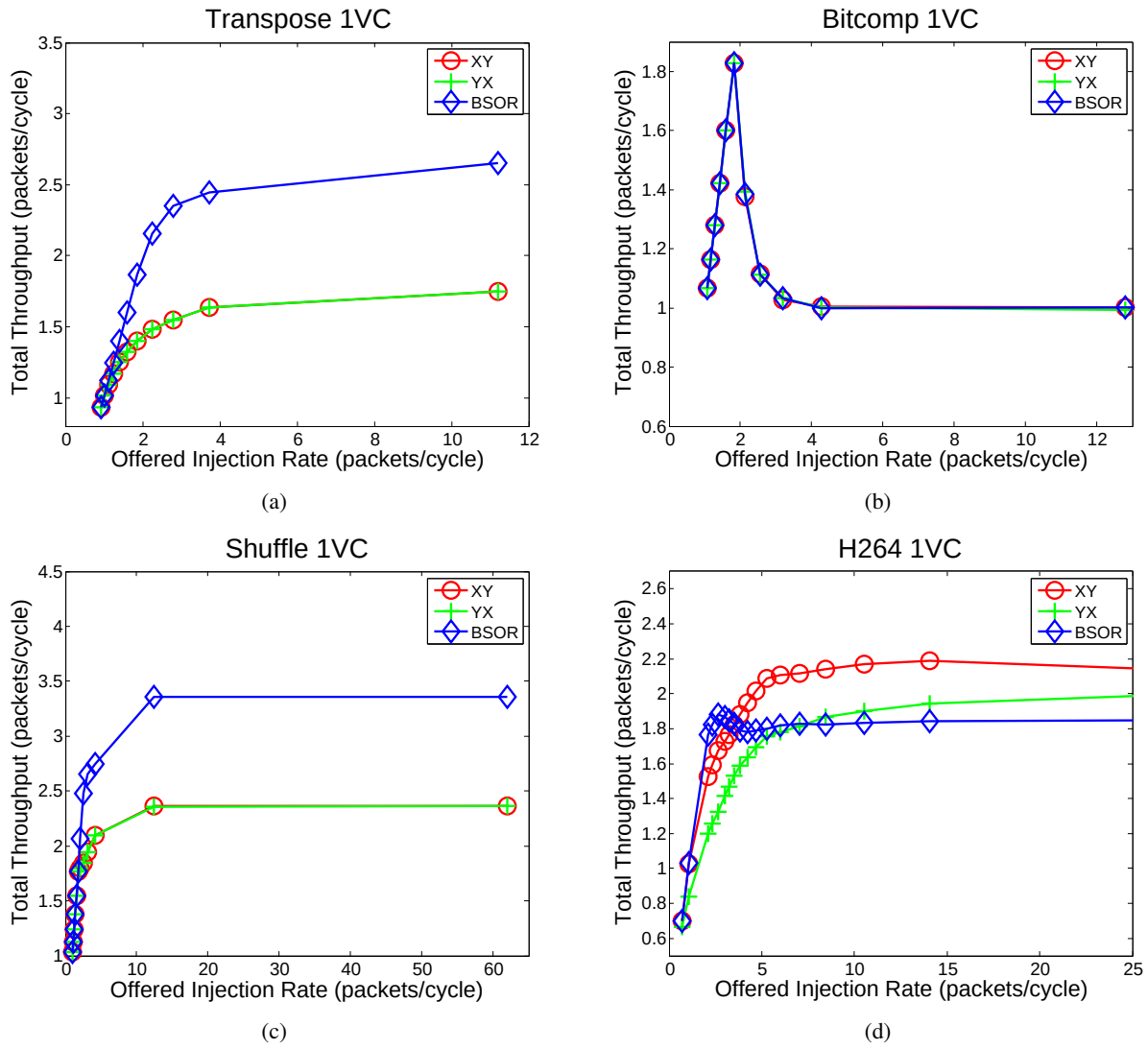


Figure 8: Load-throughput graphs for benchmarks on a router with 1 virtual channel. Each graph shows the saturation throughput for various oblivious routing algorithms. (a) transpose. (b) bit-complement. (c) shuffle. (d) H.264.

rithms in transpose and shuffle. For these benchmarks, there are multiple routes with the same minimal MCL. We employ the congestion metric of Section 4.2 as a tiebreaker to select a set of routes for each benchmark. XY-ordered and YX-ordered 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 network capacity is too small the throughput can show instability if the routes have many flows going through one or more links. This is because, when virtual channels are dynamically allocated, flow arbitration can be unfair and one flow may block other flows on its path. Unfairness causes greater performance degradation when more flows converge at nodes. The bandwidth demands of the flows are largely irrelevant when network capacity is very small. Therefore, the throughput of BSOR routes becomes lower than XY and YX routes when lots of links are congested. If network capacity is highly restricted, we should instead focus on the number of flows that go through each link. This corresponds to ignoring bandwidth demands. Figure 9 shows an alternative set of BSOR routes (BSOR2) which has higher performance than XY and YX routes at high injection rates.

If flow arbitration is fair the throughput of BSOR routes does not degrade under heavy traffic load. Static allocation of virtual channels improves fairness of flow arbitration [37], and Figure 10 shows that the BSOR set of routes with the lowest MCL consistently performs better than other oblivious routing algorithms when virtual channels are statically allocated. The routes are the same as those used in Figure 8(d).

Figure 11 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 Figure 8(a), 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's could increase by $1.5X$. BSOR

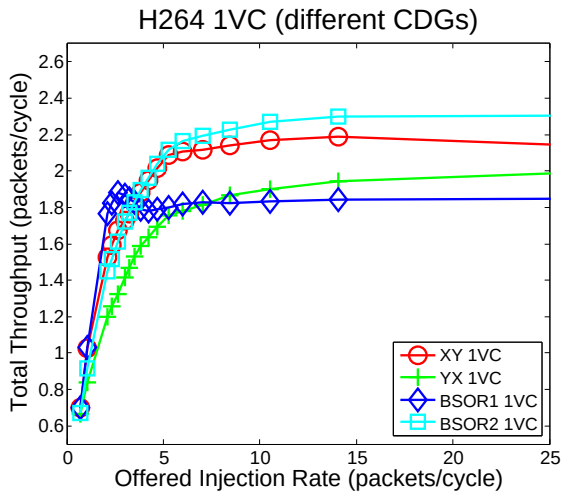


Figure 9: Load-throughput graphs for H.264 of BSOR from different CDGs.

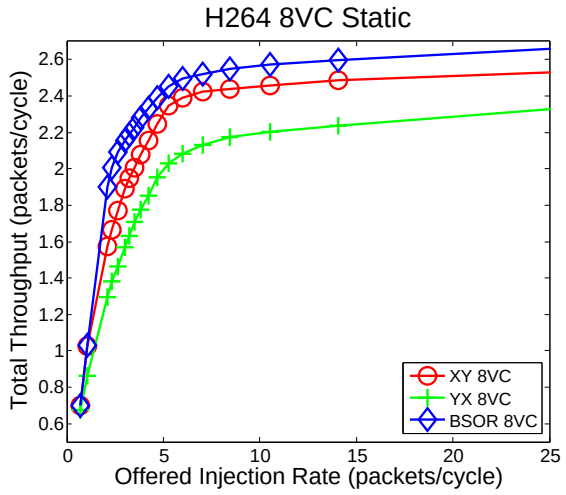


Figure 10: Load-throughput graphs for H.264 using static VC allocation (with 8 virtual channels).

continues to outperform the other algorithms since it spreads the load across the network better.

7.5 Multiple Virtual Channels

We compare BSOR with XY, YX, ROMM and Valiant under dynamic virtual channel allocation with 2 virtual channels in Figure 12. Note that ROMM and Valiant need to switch virtual channels in order to ensure deadlock-freedom. We also compare BSOR with BSORM. Note that BSORM requires static channel allocation, though the allocation, for the most part, is forced by the turns in the routes when there are only two virtual channels available. The routes produced by BSOR and BSORM are similar; however, BSORM performs better than BSOR because static allocation mitigates head-of-line blocking [37]. BSORM for bit-complement has a higher MCL than BSOR, and worse performance. The BSORM heuristic did not produce the minimum MCL YX routing for bit-complement.

The performance improvement shown, using BSOR or BSORM routing, over other oblivious routing algorithms, is relatively consistent for virtual channels greater than 2 (not shown). A more detailed comparison of static and dynamic virtual channel allocation can be found in [37].

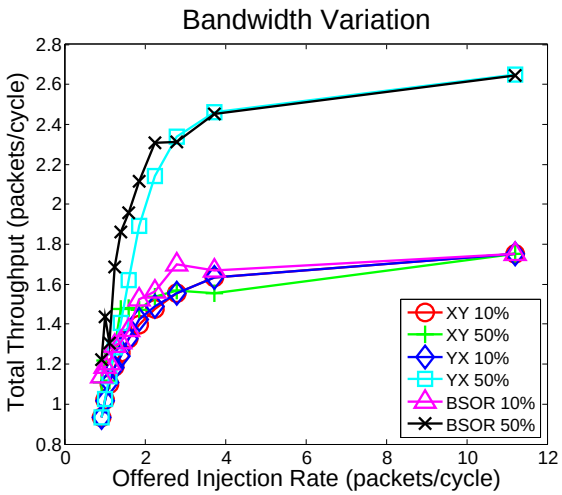


Figure 11: Load-throughput graphs for transpose (1 virtual channel) when bandwidths change by $\pm 10\%$ and $\pm 50\%$ after route computation.

8. 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 bandwidth demands 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.

The primary feature of our approach is also its limitation; 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 simply 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 [5]. Ongoing work includes evaluating BSOR and BSORM on a bandwidth-adaptive network.

Acknowledgement: We thank Derek Chiou, Joel Emer, Li-Shiuan Peh, Mieszko Lis, 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.

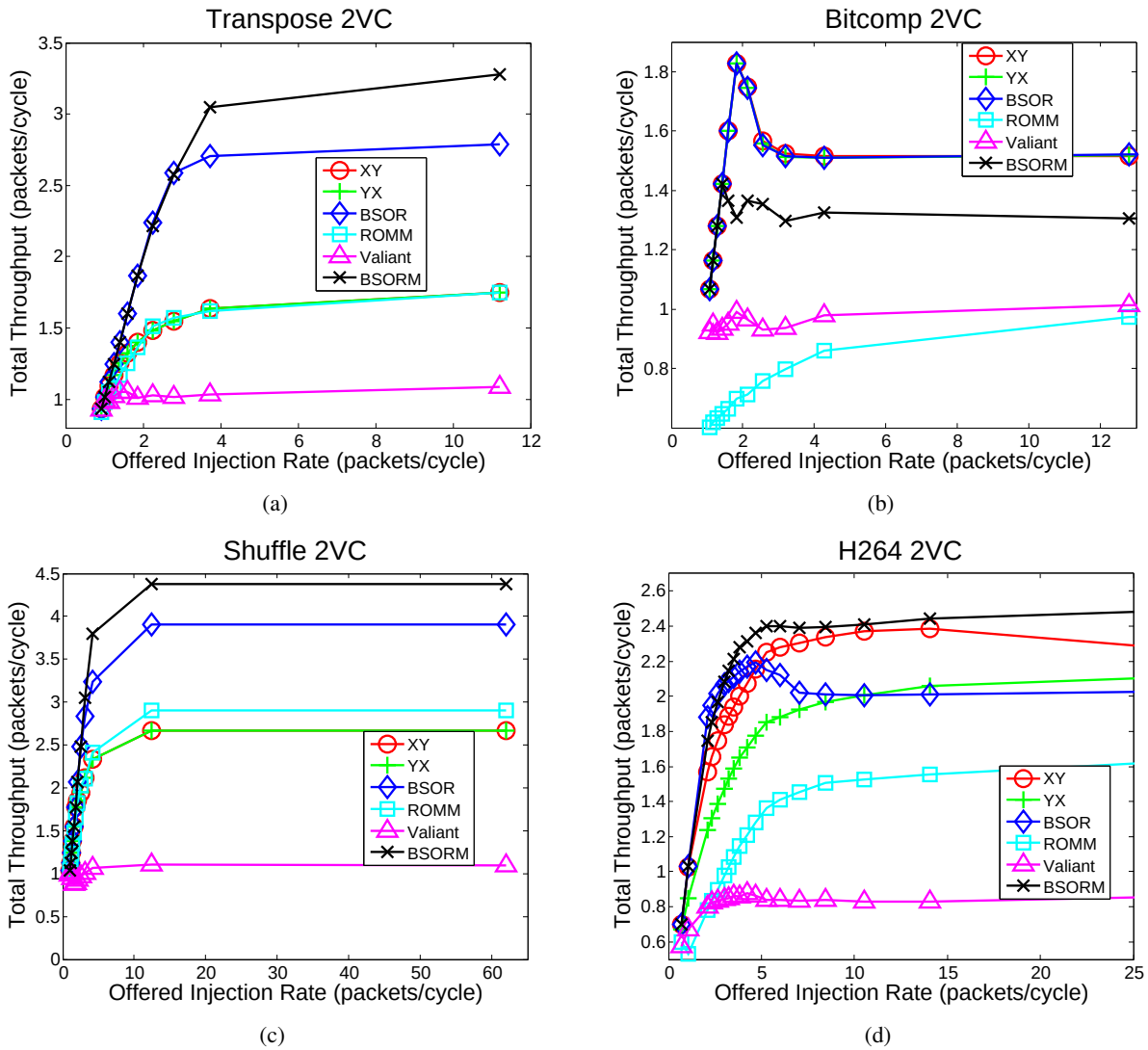


Figure 12: Load-throughput graphs for benchmarks on a router with 2 virtual channels. Each graph shows the saturation throughput for various oblivious routing algorithms. (a) transpose. (b) bit-complement. (c) shuffle. (d) H.264.

9. REFERENCES

[1] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38(1), 2006.

[2] Tobias Bjerregaard and Jens Sparsø. Virtual channel designs for guaranteeing bandwidth in asynchronous network-on-chip. In *Proceedings of the IEEE Norchip Conference (NORCHIP 2004)*. IEEE, 2004.

[3] Ge-Ming Chiu. The Odd-Even Turn Model for Adaptive Routing. *IEEE Trans. Parallel Distrib. Syst.*, 11(7):729–738, 2000.

[4] M. H. Cho, C-C. Cheng, M. Kinsy, G. E. Suh, and S. Devadas. Diastolic Arrays: Throughput-Driven Reconfigurable Computing. In *Proceedings of the Int'l Conference on Computer-Aided Design*, November 2008.

[5] M. H. Cho, M. Lis, K. S. Shim, M. Kinsy, T. Wen, and S. Devadas. Oblivious Routing in On-Chip Bandwidth-Adaptive Networks. Technical Report CSAIL-TR-2009-011 (http://hdl.handle.net/1721.1/44958), Massachusetts Institute of Technology, March 2009.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 2001.

[7] William J. Dally, P. P. Carvey, and L. R. Dennison. The Avici Terabit Switch/Router. In *Proceedings of the Symposium on Hot Interconnects*, pages 41–50, August 1998.

[8] William J. Dally and Charles L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Trans. Computers*, 36(5):547–553, 1987.

[9] William J. Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.

[10] W.J. Dally. Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 03(2):194–205, 1992.

[11] José Duato. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Trans. Parallel Distrib. Syst.*, 4(12):1320–1331, 1993.

[12] José Duato. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Trans. Parallel Distrib. Syst.*, 6(10):1055–1067, 1995.

[13] Mike Galles. Scalable Pipelined Interconnect for Distributed

Endpoint Routing: The SGI SPIDER Chip. In *Proceedings of the Symposium on Hot Interconnects*, pages 141–146, August 1996.

[14] Roman Gindin, Israel Cidon, and Idit Keidar. NoC-Based FPGA: Architecture and Routing. In *First International Symposium on Networks-on-Chips (NOCS 2007)*, pages 253–264, 2007.

[15] Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. *J. ACM*, 41(5):874–902, 1994.

[16] Thomas Gross and David R. O'Hallaron. *iWarp: anatomy of a parallel computing system*. MIT Press, Cambridge, MA, USA, 1998.

[17] Zvika Guz, Isask'har Walter, Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. Efficient link capacity and QoS design for network-on-chip. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 9–14, 2006.

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

[19] Jingcao Hu and Radu Marculescu. DyAD: Smart Routing for Networks on Chip. In *Design Automation Conference*, June 2004.

[20] Natalie Enright Jerger, Li-Shiuan Peh, and Mikko Lipasti. Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support. In *ISCA '08: Proceedings of the 35th annual international symposium on Computer architecture*, 2008.

[21] N. K. Kavaldjiev, G. J. M. Smit, and P. G. Jansen. A virtual channel router for on-chip networks. In *IEEE Int. SOC Conf., Santa Clara, California*, pages 289–293. IEEE Computer Society Press, September 2004.

[22] Jon Michael Kleinberg. *Approximation algorithms for disjoint paths problems*. PhD thesis, Massachusetts Institute of Technology, 1996. Supervisor-Michel X. Goemans.

[23] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K. Jha. Toward Ideal On-Chip Communication Using Express Virtual Channels. *IEEE Micro*, 28(1):80–90, 2008.

[24] Olav Lysne and José Duato. Fast Dynamic Reconfiguration in Irregular Networks. In *ICPP '00: Proceedings of the Proceedings of the 2000 International Conference on Parallel Processing*, page 449, 2000.

[25] Robert D. Mullins, Andrew F. West, and Simon W. Moore. Low-latency virtual-channel routers for on-chip networks. In *Proc. of the 31st Annual Intl. Symp. on Computer Architecture (ISCA)*, pages 188–197, 2004.

[26] Srinivasan Murali, David Atienz, Luca Benini, and Giovanni De Micheli. A Method for Routing Packets Across Multiple Paths in NoCs with In-Order Delivery and Fault-Tolerance Gaurantees. *VLSI Design*, 2007.

[27] Srinivasan Murali and Giovanni De Micheli. SUNMAP: a tool for automatic topology selection and generation for NoCs. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 914–919, 2004.

[28] Ted Nesson and S. Lennart Johnsson. ROMM routing on mesh and torus networks. In *Proc. 7th Annual ACM Symposium on Parallel Algorithms and Architectures SPAA'95*, pages 275–287, 1995.

[29] Lionel M. Ni and Philip K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *Computer*, 26(2):62–76, 1993.

[30] Chrysostomos A. Nicopoulos, Dongkook Park, Jongman Kim, Narayanan Vijaykrishnan, Mazin S. Yousif, and Chita R. Das. ViChaR: A dynamic virtual channel regulator for network-on-chip routers. In *Proc. of the 39th Annual Intl. Symp. on Microarchitecture (MICRO)*, 2006.

[31] M. Palesi, R. Holsmark, S. Kumar, and V. Catania. A methodology for design of application specific deadlock-free routing algorithms for NoC systems. In *Proc. Intl. Conf. on Hardware-Software Codesign and System Synthesis*, Seoul, Korea, October 2006.

[32] 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. of the ACM/IEEE Int. Symp. on Networks-on-Chip (NOCS)*, pages 97–106, 2008.

[33] Li-Shiuan Peh and William J. Dally. Flit-reservation flow control. In *In Proc. of the 6th Int. Symp. on High-Performance Computer Architecture (HPCA)*, pages 73–84, January 2000.

[34] Li-Shiuan Peh and William J. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. In *Proc. International Symposium on High-Performance Computer Architecture (HPCA)*, pages 255–266, January 2001.

[35] Loren Schwiebert. Deadlock-free oblivious wormhole routing with cyclic dependencies. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 149–158, 1997.

[36] Daeho Seo, Akif Ali, Won-Taek Lim, Nauman Rafique, and Mithuna Thottethodi. Near-Optimal Worst-Case Throughput Routing for Two-Dimensional Mesh Networks. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA 2005)*, pages 432–443, 2005.

[37] K. S. Shim, M. H. Cho, M. Kinsy, T. Wen, M. Lis, G. E. Suh, and S. Devadas. Static Virtual Channel Allocation in Oblivious Routing. In *Proceedings of the* 3^{rd} *ACM/IEEE International Symposium on Networks-on-Chip*, May 2009.

[38] Craig B. Stunkel and Peter H. Hochschild. SP2 High-Performance Switch Architecture. In *Proceedings of the Symposium on Hot Interconnects*, pages 115–121, August 1994.

[39] Craig B. Stunkel, Dennis G. Shea, Don G. Grice, Peter H. Hochschild, and Michael Tsao. The SP1 High-Performance Switch. In *Proceedings of the Scalable High Performance Computing Conference*, pages 150–157, May 1994.

[40] Brian Towles, William J. Dally, and Stephen Boyd. Throughput-centric routing algorithm design. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 200–209, 2003.

[41] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 263–277, 1981.

[42] Krzysztof Walkowiak. New Algorithms for the Unsplittable Flow Problem. In *ICCSA (2)*, volume 3981 of *Lecture Notes in Computer Science*, pages 1101–1110, 2006.

[43] Xiaoxiong Zhong and Virginia Mary Lo. Application-Specific Deadlock Free Wormhole Routing on Multicomputers. In *PARLE '92: Proceedings of the 4th International PARLE Conference on Parallel Architectures and Languages Europe*, pages 193–208, 1992.