Static virtual channel allocation in oblivious routing

The MIT Faculty has made this article openly available. ***Please share*** how this access benefits you. Your story matters.

Citation	Static virtual channel allocation in oblivious routing Keun Sup Shim; Myong Hyon Cho; Kinsy, M.; Wen, T.; Lis, M.; Suh, G.E.; Devadas, S.; Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on 10-13 May 2009 Page(s):38 - 43
As Published	http://dx.doi.org/10.1109/NOCS.2009.5071443
Publisher	Institute of Electrical and Electronics Engineers
Version	Author's final manuscript
Accessed	Fri Dec 14 19:02:21 EST 2018
Citable Link	http://hdl.handle.net/1721.1/51681
Terms of Use	Attribution-Noncommercial-Share Alike 3.0 Unported
Detailed Terms	http://creativecommons.org/licenses/by-nc-sa/3.0/

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

DSpace@MIT

Static Virtual Channel Allocation in Oblivious Routing

Keun Sup Shim Myong Hyon Cho Michel Kinsy Tina Wen Mieszko Lis G. Edward Suh* Srinivas Devadas
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

ABSTRACT

Most virtual channel routers have multiple virtual channels to mitigate the effects of head-of-line blocking. When there are more flows than virtual channels at a link, packets or flows must compete for channels, either in a dynamic way at each link or by static assignment computed before transmission starts. In this paper, we present methods that statically allocate channels to flows at each link when oblivious routing is used, and ensure deadlock freedom for arbitrary minimal routes when two or more virtual channels are available. We then experimentally explore the performance tradeoffs of static and dynamic virtual channel allocation for various oblivious routing methods, including DOR, ROMM, Valiant and a novel bandwidth-sensitive oblivious routing scheme (BSORM). Through judicious separation of flows, static allocation schemes often exceed the performance of dynamic allocation schemes.

1. INTRODUCTION

Routers may mitigate head-of-line blocking by organizing the buffer storage associated with each network channel into several small queues rather than a single deep queue [7]. Such "virtual" channels (VCs) increase hardware complexity but offer a mechanism to achieve Quality-of-Service (QoS) guarantees and performance isolation — important considerations for on-chip interconnection networks (OCINs) [14]).

Most routers in OCINs have a small number of VCs, though network routers can have large numbers of queues and channels (e.g., Avici TSR [4]). While overhead considerations tend to limit routers used in multicore or multiprocessor systems to 16 or fewer VCs, applications may have hundreds if not thousands of flows, which must compete for channels, buffer space, and bandwidth at each network link.

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 [6], and the associated arbitration is often the highest latency step [16].

Statically allocating VCs to flows can simplify the VC allocation step. Judicious separation of flows during static allocation may reduce or eliminate head-of-line blocking and so enhance throughput, but may result in worse utilization of available VCs because dynamic behavior is not considered. There has been little or no exploration of such performance tradeoffs outside of evaluating QoS guarantees; therefore, in this paper, we answer the question of whether static allocation outperforms dynamic allocation in the context of oblivious routing through performance simulation.

Exploring this tradeoff requires methods that statically allocate flows to VCs at each link and a router architecture that supports these methods. Section 2 describes modifications to a standard router architecture for table-based application-aware routing and static VC allocation; section 3 describes how algorithms such as Dimension Order Routing (DOR), ROMM [13] and Valiant [19] can assign VCs via a table-based routing architecture. The performance of these routing algorithms varies under different static allocations, and we show how this allocation can be judiciously determined. In Section 4, we describe a bandwidth-sensitive oblivious routing scheme, BSORM, which produces a set of minimal routes that attempt to minimize maximum channel load; VCs are statically allocated to optimize performance. We show how an analysis of the classical turn model [10] can be used to derive a static VC allocation scheme that assures deadlock freedom for an arbitrary set of minimal routes with ≥ 2 available VCs. Related work is summarized in Section 5. We compare static and dynamic VC allocation for DOR, ROMM, Valiant, and BSORM in Section 6, and Section 7 concludes the paper.

2. ROUTER ARCHITECTURE

2.1 Typical Virtual Channel Router

We assume a typical virtual channel (VC) router on a 2-D mesh network as a baseline [6, 12, 16], but our methods can be used independent of network topology and flow control mechanisms.

Router operation takes four steps: routing (RC), virtual channel allocation (VA), switch allocation (SA), and switch traversal (ST), often done in one to four stages in modern routers. When a head flit (the first flit of a packet) arrives at an input channel, the router stores the flit in the channel's buffer and determines the next hop for the packet (RC). The router then allocates a VC in the next hop (VA). Finally, if the next hop can accept the flit, it competes for a switch (SA) and moves to the output port (ST).

2.2 Table-Based Routing

The only architectural change required for static VC allocation and application-aware oblivious routing (see Section 4) is in the routing module. While the baseline architecture implements simple oblivious routing algorithms such as DOR via fixed logic and dynamically allocates VCs to packets, our routing module needs *table-based routing* so that routes can be configured for each application. This single change is sufficient as long as routing algorithms preclude cyclic channel dependence through route selection or VC allocation (cf. Section 4.3).

As illustrated in Figure 1, table-based routing can be realized in two different ways: source routing and node-table routing. In the *source routing* approach, each node has a routing table with a route from itself to each destination node in the network. The routes

*Cornell University

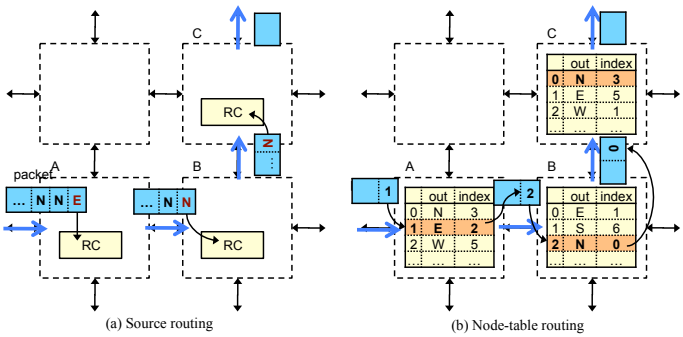


Figure 1: The table-based routing architecture (a) Source routing (b) Node-table routing

are pre-computed by a routing algorithm and written into the tables before application execution. When sending a packet, the node prepends this routing information to the packet, and routers along the path determine output ports directly from the routing flits. Figure 1(a) illustrates source routing for a packet routed through nodes A, B, and C. The route corresponds to East, North, and North, which is reflected in the routing flits.

Source routing eliminates the routing step and can potentially reduce the number of pipeline stages, but results in longer packets (with extra routing flits) compared to the case where the route is computed at each hop. To avoid this, the nodes along the path can be programmed with next-hop routing information for relevant flows. In this *node-table* routing approach, illustrated in Figure 1(b), the routing module contains a table with the output port for each flow routed through the node. The head packet carries an index into this table, which, once looked up, is replaced with the index for the next hop stored in the table entry. To set up the route, our routing algorithm computes a route for each flow and configures the routing tables accordingly.

If we conservatively assume that each routing table has 256 entries (256 flows), the table only takes a few KB: an entry needs 2 bits to represent the output port in a 2-D mesh and 8 bits for the next table index. Therefore, the table can be accessed in one cycle without impacting clock frequency.

Both routing methods are widely known and have been implemented in multiple routers (e.g., [4, 8]).

2.3 Static Virtual Channel Allocation

Statically allocating a VC to each flow simplifies the VC allocation step of the baseline router. Rather than being dynamically allocated using arbiters, VCs at each link are allocated per flow by the routing algorithm. 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 routing (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. A previous study [16] indicates that $P \cdot V$ to 1 arbitration is about 20% faster than $P \cdot V$ to V arbitration (11.0 FO4 vs. 13.3 FO4 with 8 VCs).

Static VC allocation requires additional bits in the routing table to specify the VC for each flow. For example, for 8 VCs, 3 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.

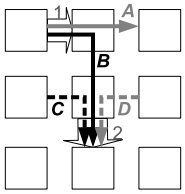
Since static allocation does not consider dynamic behavior, it can potentially result in worse utilization of available VCs; for example, statically allocating VC0 to flow A and VC1 to flow B may be inefficient when flow A is idle, because flow B might be able to use both VCs. On the other hand, static allocation can enhance throughput by separating or isolating flows (cf. Figure 2).

3. STATIC VC ALLOCATION IN OBLIVIOUS ROUTING

We assume the router design described in Section 2 with support for static VC allocation as described in Section 2.3. Since each link has multiple VCs, the assignment of channels to flows is done on a *per link* basis.

3.1 Dimension-Ordered Routing (DOR)

On a mesh, dimension-ordered routing corresponds to either XY or YX routing. Figure 2 exhibits the advantages of static allocation: four uncorrelated flows with the same demands are shown, using XY routing with four VCs. Flows B, C, and D share link 2, which becomes congested when injection rates are high; this limits the throughput of flow B to approximately one-third of the link bandwidth. If dynamic allocation is used, flow A also suffers because of head-of-line blocking when flow A is held up by flow B. If we statically allocate VCs, however, we can assign flows A and B to separate channels and utilize the full bandwidth of link 1.



	Dynamic	Static
Flow A	0.3234	0.6681
Flow B	0.3146	0.3319
Flow C	0.3365	0.3332
Flow D	0.3488	0.3349

Throughput (flits/cycle)

Figure 2: Motivation for Static Allocation

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.

Flows are assigned to VCs separately at each link. Given a link and a flow F using it, the allocation algorithm proceeds as follows:

1. Check if there is a VC containing only flows that are already entangled with F. 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's allocated to F.
2. Look for empty VCs on the link; if one exists, assign it to F.
3. If some VC contains a flow entangled with F, assign it to F.
4. If none of the criteria above apply, assign F to the VC with the fewest flows.
5. Update flow entanglement relationships to reflect the new assignment.

The process above is repeated for each flow at the given link, and the algorithm moves on to the next link.

3.2 ROMM and Valiant

The ROMM [13] and Valiant [19] routing algorithms attempt to balance network load by choosing random intermediate nodes in the network and using XY/YX routing to route first from the source to the intermediate node and then from there to the destination.

The basic algorithm for static allocation is same as for DOR. The only difference arises from the requirement that the source-to-intermediate and intermediate-to-destination subroutes not share the same VCs, in order to avoid deadlock. This reduces our allocation choices, since flows must be assigned VCs only within the particular set. While ROMM and Valiant thus require a minimum of 2 VCs, having more than 2 is desirable as it affords some freedom in allocating VCs.

4. STATIC VC ALLOCATION IN BANDWIDTH-SENSITIVE ROUTING

We now show how to select routes to minimize maximum channel load given rough estimates of flow bandwidths, and how deadlock freedom can be assured through static VC allocation subsequent to route selection. (We again assume the router design from Section 2 with the static VC allocation support of Section 2.3).

4.1 Flow graph and Turn Model

DEFINITION 1. *Let $G(V,E)$ be a flow graph where each edge $(u,v) \in E$ has a capacity $c(u,v)$ representing the available bandwidth on the edge. Let $K = \{K_1, K_2, \ldots, K_k\}$ be a set of k data transfers (or flows) where $K_i = (s_i, t_i, d_i)$ and s_i represents the source for connection i, t_i the sink (with $s_i \neq t_i$), and d_i the demand; multiple flows with the same source and destination are permitted. The flow i along an edge (u,v) is $f_i(u,v)$. A route for flow i is a path p_i from s_i to t_i; edges along this path will have $f_i(u,v) > 0$, while 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 edge (u,v); the magnitude of $f_i(u,v)$ indicates how much of the edge's bandwidth is used by flow i. Although we assume flit-buffer flow control in this paper, our techniques also apply to other flow control schemes.

With a single VC per link or dynamic VC allocation, packets routes that conform to an acyclic channel dependence graph avoid network deadlock [5]. This is also a necessary condition unless false resource dependencies exist [17].

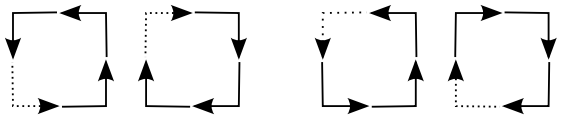


Figure 3: Turns allowed (solid) and disallowed (dotted) under (a) the West-First turn model and (b) the North-Last turn model.

Turn models [10] are a systematic way of generating deadlock-free routes, and have been used for adaptive routing. Figure 3 shows two turn models that can be used in a 2-D mesh: each model disallows two out of the eight possible turns. If a set of routes conforms to one of the turn models, then deadlock freedom is assured with any number of VCs. The third turn model, Negative-First, does not serve our purposes and so is not shown.[1]

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

We now describe a routing method that targets improved network throughput given rough estimates of flow bandwidths. We show how any set of *minimal* routes produced using any routing method can be made deadlock-free through appropriate static VC allocation (cf. Section 4.3); our argument for deadlock freedom invokes the turn models of Figure 3.

Given rough estimates of bandwidths of data transfers or flows, bandwidth-sensitive oblivious routing selects routes to minimize the *maximum channel load*, i.e., the maximum bandwidth demand on any link in the network. The method works on a flow graph $G(V,E)$ corresponding to the network; 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. Consider a link e in G with a capacity $c(e)$. We create a variable for $\tilde{c}(e)$ representing the current residual capacity of e; initially, $\tilde{c}(e)$ equals the capacity $c(e)$, and is set to be a constant C. If the residual capacity $\tilde{c}(e)$ exceeds the demand d_i of a flow i, then flow i can be routed via link e and d_i is subtracted from $\tilde{c}(e)$. Since flows are not routed through links with insufficient $\tilde{c}(e)$, no residual capacity is ever negative.

For the weighting function, we use the reciprocal of the link residual capacity, which is similar to the CSPF metric described by Walkowiak [20]: $w(e) = \frac{1}{\tilde{c}(e)-d_i}$, 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 the smallest number that provides 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, there are solutions for lower values of C; in effect, a smaller C places more weight on avoiding congested links.

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 non-minimal, 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 Figure 4(b) 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.

4.3 Deadlock-Free Static VC Allocation

Since the routes selected by the Dijkstra-based algorithm may not conform to a particular acyclic CDG or turn model, they may not be deadlock-free. If the number of available VCs exceeds 2, however, we can ensure deadlock freedom via static VC assignment by partitioning the flows across available VCs.

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.*

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

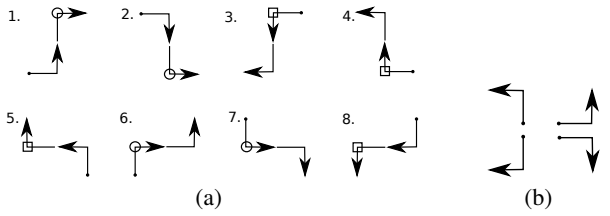


Figure 4: (a) The eight different two-turn minimal routes on a 2-D mesh. (b) The four (out of a possible eight) different one-turn routes on a 2-D mesh that conform to both the West-First and North-Last turn model.

Proof: Consider, without loss of generality, the case of 2 VCs. Figure 4(a) shows the eight possible minimal routes with two different turns each. (Minimal routes that have a single turn or no turns can be ignored as special cases of two-turn routes for the subsequent analysis). Looking at Figure 4(a), 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 one-turn minimal routes shown in Figure 4(b), and routes with no turns, can be placed in either set; the four other one-turn 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. After deriving minimal routes using the BSOR algorithm of Section 4.2, 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 zero-turn 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 that 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 2 and that for the second set is 6, 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.

5. RELATED WORK

5.1 Routing Techniques

A basic deterministic routing method is dimension ordered routing (DOR) [5] which becomes XY routing in a 2-D mesh. Necessary and sufficient conditions for deadlock-free deterministic routing were given in [5] assuming no false resource dependences.

ROMM [13] and Valiant [19] are classic oblivious routing algorithms, which are randomized in order to achieve better load distribution. In o1turn [18], 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 (VCs) assigns XY and YX routes to source-destination pairs in a way that reduces the maximum network load for a given traffic pattern [9].

Classic adaptive routing schemes include the turn routing methods [10] and odd even routing [1].

5.2 Bandwidth-Aware Routing

Palesi *et al* [15] 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. Our focus here is on oblivious routing.

Bandwidth-aware routing for diastolic arrays is described in [2]; deadlock is avoided by assuming that each flow has its own private channel. An application-aware oblivious routing (BSOR) framework for conventional routers with dynamic VC allocation and one or more VCs is presented in [11]; this framework selects possibly non-minimal routes that conform to an acyclic CDG typically derived from a turn model. In this paper, our focus is static VC allocation schemes for traditional oblivious routing methods (e.g., DOR, ROMM, Valiant) as well as for bandwidth-sensitive oblivious routing.

6. RESULTS AND COMPARISONS

This section compares the performance of static and dynamic VC allocation using synthetic traffic through simulation. We also compare our routing scheme (BSORM) with other oblivious routing algorithms like DOR, ROMM [13], and Valiant [19].

6.1 Benchmarks

We use a set of standard synthetic traffic patterns: transpose, bit-complement, and shuffle, as well as an application benchmark H.264. The synthetic patterns are widely used to evaluate routing algorithms and provide basic comparisons between our routing scheme and other oblivious algorithms; in the synthetic benchmarks, all flows have the same average bandwidth demands. H.264 is a set of flows reflecting the traffic pattern of an H.264 decoder, with flow bandwidths derived through profiling.

6.2 Simulator Details

A cycle-accurate network simulator was used to estimate the throughput of each flow in the application for various oblivious routing. We use an 8×8 2-D mesh network with 1, 2, 4 or 8 VCs per port, and we simulate a fixed packet length of 8 flits. 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.

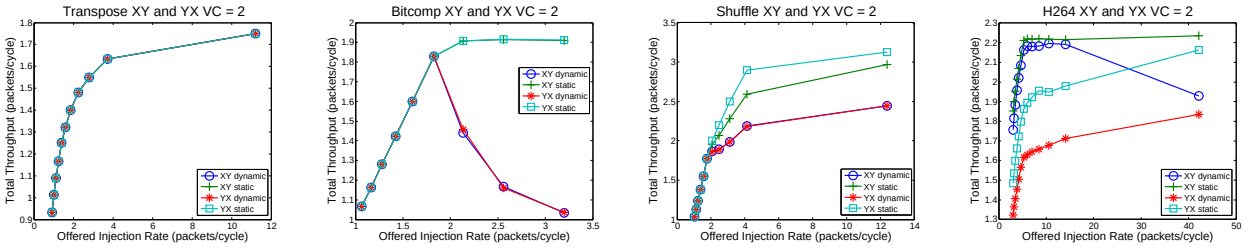


Figure 5: Throughput for dimension-ordered routing under static and dynamic allocation with 2 VCs.

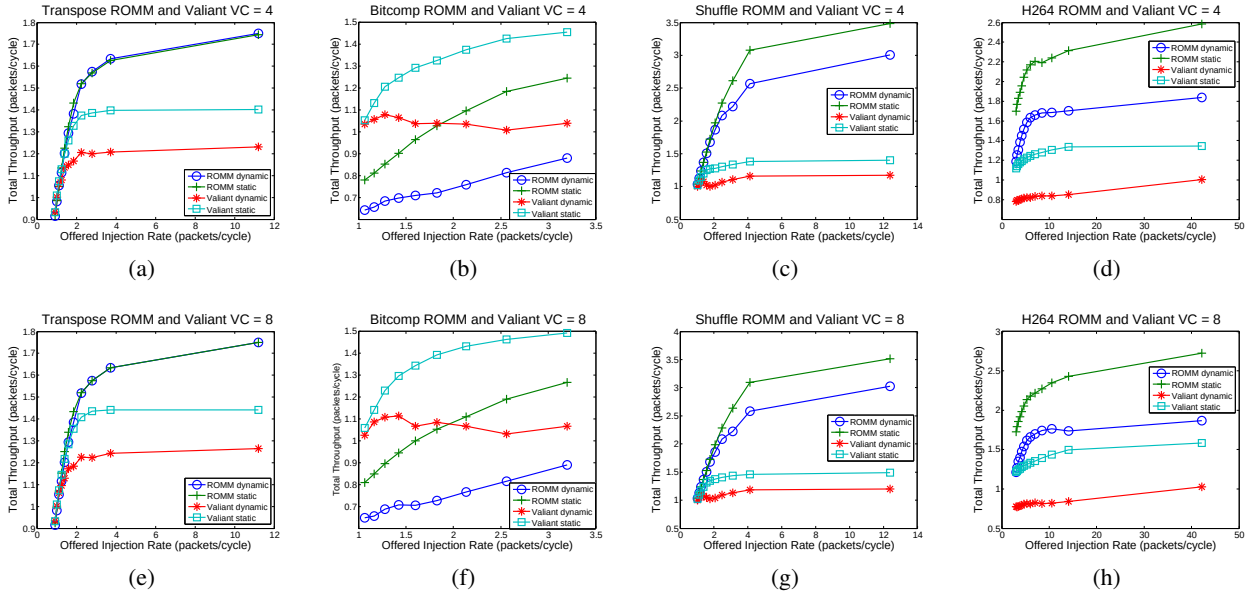


Figure 6: Throughput for ROMM and Valiant under static and dynamic allocation with 4 and 8 VCs.

6.3 DOR, ROMM and Valiant

Figure 5 shows the performance of XY and YX routing with 2 VCs for static and dynamic VC allocation for various benchmarks. Figure 6 shows the performance of ROMM and Valiant under static and dynamic allocation for 4 and 8 VCs. ROMM and Valiant routes require 2 VCs to avoid deadlock; these routes are broken into two segments, with a VC allocated to each segment. Hence static and dynamic allocation schemes differ when there are multiple VCs that can be allocated to each route segment.

For all these algorithms, static allocation performs as good or better than dynamic allocation for high injection rates by more effectively reducing head-of-line blocking effects as exemplified in Figure 2.

6.4 BSORM

Figure 7 shows the performance of the BSORM algorithm for four VCs and compares it to XY (static and dynamic) for various benchmarks. We use BSORM to obtain the routes and break these routes into two sets to avoid deadlock, as described in Section 4.3. We perform static allocation or assume dynamic allocation within each set. Figure 8 compares BSORM under static and dynamic allocation for 8 VCs. As each benchmark uses a single routing derived using BSORM, the performance differences are due only to static versus dynamic VC allocation.

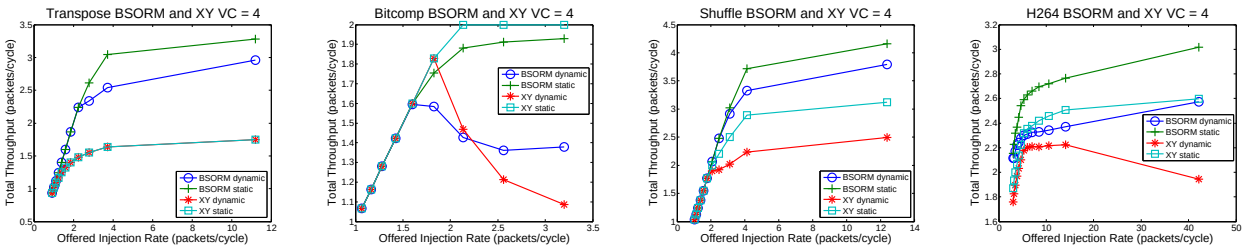
BSORM performs better than DOR on the benchmarks because the bandwidth-aware routing reduces MCL; BSORM with static allocation outperforms dynamic allocation for the same reasons as in DOR.

7. CONCLUSIONS

Our results indicate that static VC allocation often outperforms dynamic VC allocation for existing oblivious routing schemes. This is because static allocation can better reduce the effects of head-of-line blocking.

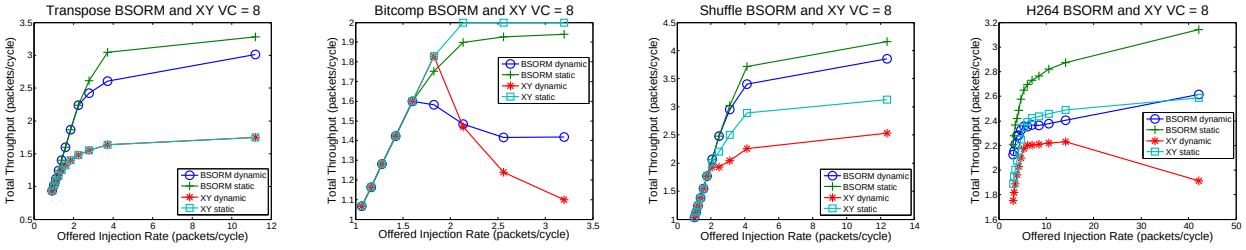
When given rough estimates of bandwidths, the BSORM algorithm provides better performance than existing oblivious routing schemes, and here too, static allocation produces as good or better results. If head-of-line blocking effects are small, maximum channel load serves as a dominant factor in determining the performance of a given route. This justifies the BSORM algorithm's minimization of the maximum channel load. Two other advantages of static allocation are that routers with static allocation can be slightly simpler than those with dynamic allocation, and static allocation assures in-order packet delivery, since two VCs in a link will never be assigned to flits/packets from the same flow.

Bandwidth-adaptive networks contain adaptive bidirectional links and can improve the performance of conventional oblivious routing methods [3]. Ongoing work includes evaluating BSORM on a bandwidth-adaptive network.



(a) (b) (c) (d)

Figure 7: Throughput for BSORM and XY under static and dynamic allocation with 4 VCs.



(a) (b) (c) (d)

Figure 8: Throughput for BSORM and XY under static and dynamic allocation with 8 VCs.

8. REFERENCES

[1] Ge-Ming Chiu. The odd-even turn model for adaptive routing. *IEEE Trans. Parallel Distrib. Syst.*, 11(7):729–738, 2000.

[2] 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.

[3] 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.

[4] 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.

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

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

[7] W.J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 03(2):194–205, 1992.

[8] 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.

[9] 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.

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

[11] Michel Kinsy, Myong Hyon Cho, Tina Wen, Edward Suh, Marten van Dijk, and Srinivas Devadas. Application-Aware Deadlock-Free Oblivious Routing. In *Proceedings of the Int'l Symposium on Computer Architecture*, June 2009.

[12] 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.

[13] 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.

[14] J. D. Owens, W. J. Dally, R. Ho, D. N. Jayasimha, S. W. Keckler, and L-S. Peh. Research Challenges for On-Chip Interconnection Networks. *IEEE Micro*, 27(5):96–108, Sept/Oct 2007.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] Krzysztof Walkowiak. New algorithms for the unsplittable flow problem. In *ICCSA (2)*, volume 3981 of *Lecture Notes in Computer Science*, pages 1101–1110, 2006.