Static and Dynamic Virtual Channel Allocation for High Performance, In-Order Communication in On-Chip Networks

by

Keun Sup Shim

Bachelor of Science, Electrical Engineering and Computer Science,
KAIST, 2006

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author .....
Department of Electrical Engineering and Computer Science
May 18, 2010

Certified by .....
Srinivas Devadas
Associate Department Head, Professor
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Theses

Static and Dynamic Virtual Channel Allocation for High Performance, In-Order Communication in On-Chip Networks

by

Keun Sup Shim

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2010, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Most routers in on-chip interconnection networks (OCINs) have multiple virtual channels (VCs) to mitigate the effects of head-of-line blocking. Multiple VCs necessitate VC allocation schemes since packets or flows must compete for channels when there are more flows than virtual channels at a link. Conventional dynamic VC allocation, however, raises two critical issues. First, it still suffers from a fair amount of head-of-line blocking since all flows can be assigned to any VC within a link. Moreover, dynamic VC allocation compromises the guarantee of in-order delivery even when used with basic variants of dimension-ordered routing, requiring large reorder buffers at the destination core or, alternatively, expensive retransmission logic.

In this thesis, we present two virtual channel allocation schemes to address these problems: Static Virtual Channel Allocation and Exclusive Dynamic Virtual Channel Allocation (EDVCA). Static VC allocation assigns channels to flows by precomputation when oblivious routing is used, and ensures deadlock freedom for arbitrary minimal routes when two or more VCs are available. EDVCA, on the other hand, is done at runtime, not requiring knowledge of traffic patterns or routes in advance. We demonstrate that both static VCA and EDVCA guarantee in-order packet delivery under single path routing, and furthermore, that they both outperform dynamic VC allocation (out-of-order) by effectively reducing head-of-line blocking. We also introduce a novel bandwidth-sensitive oblivious routing scheme (BSORM), which is deadlock-free through appropriate static VC allocation. Implementation for these schemes requires only minor, inexpensive changes to traditional oblivious dimension-ordered router architectures, more than offset by the removal of packet reorder buffers and logic.

Thesis Supervisor: Srinivas Devadas
Title: Associate Department Head, Professor

Acknowledgments

First, I would like to start by expressing my sincere gratitude to my advisor, Professor Srinivas Devadas. Throughout this research, he not only guided me with great insight, but also offered me full support to develop myself as an independent researcher. Whenever I encountered any problem, I could always make a step forward thanks to his extreme energy and brilliant ideas. Without Srini, I could never have done this.

I truly thank my collaborators on this project, Myong Hyon "Brandon" Cho, Michel Kinsy, Mieszko Lis and Tina Wen. I am especially grateful to Brandon for being a great mentor not only in research, but also in my graduate life. Michel and Tina gave me significant help in this project, and I greatly thank Mieszko for the simulator development and insightful feedbacks. It was a wonderful opportunity to work with these highly creative and talented people. I would also like to extend my deep gratitude to Samsung Scholarship, who supported me financially during my study.

I greatly appreciate my dear friends, Donghyun, Jae-Byum and Joong Bum for always being there for me. They encouraged me when I was down, shared wonderful times with me, and also were great research buddies. We've known each other for more than ten years since high school, and we are all now here in Boston as MIT graduate students. They are true friends who can understand me more than I do myself!

Finally, I cannot thank enough to my parents and family who have always been behind me 100% throughout my entire life. I would not be who I am today without their amazing love and support. I dedicate this thesis to them.

Contents

List of Figures

List of Tables

Chapter 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 [10]. 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) [26].

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 [7]). 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. When there are more flows than virtual channels at a link, packets or flows must compete for channels, thus requiring virtual channel allocation schemes.

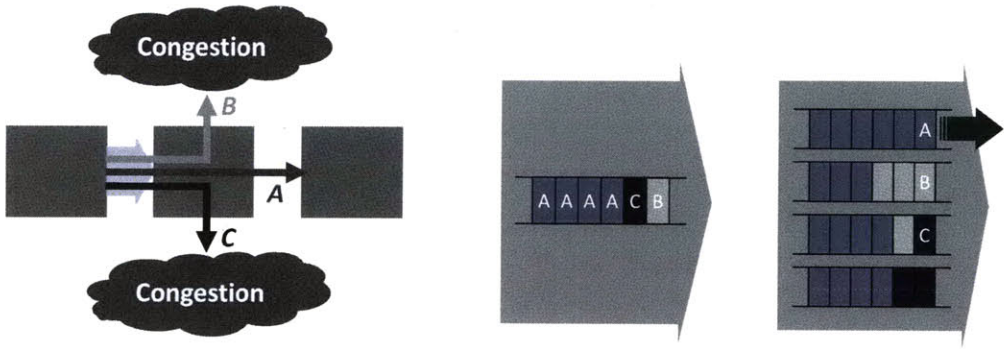
Conventional virtual channel (VC) routers dynamically allocate VCs to packets or head/control flits based on channel availability and/or packet/flit waiting time. Typically, any flit can compete for any VC at a link [9], and the associated arbitration is often the highest latency step [28].

Although this conventional VC allocation scheme under multiple virtual channels may reduce the head-of-line blocking effects somewhat, it cannot reach the performance that can be achieved by removal of head-of-line blocking. What is worse is that it creates a new problem, which is out-of-order packet delivery. We will describe in more detail these two

limitations of conventional dynamic allocation.

1.1 Head-of-Line Blocking is a Problem

Under basic dimension-order routing (DOR) without multiple virtual channels – a common approach in network-on-chip (NoC) designs – all packets follow the same path and are stored in the same buffers. Because packets from different flows are buffered in the same queues, however, a single ill-behaved flow can overwhelm other flows and effectively block them even if they are destined for a different egress port, a phenomenon known as head-of-line blocking.



(a) Flow A, B and C share the same physical link.

(b) Single VC

(c) Four VCs

Figure 1-1: Multiple virtual channels can mitigate the effects of head-of-line blocking.

Figure 1-1 illustrates the scenario that having multiple VCs can mitigate head-of-line blocking effects. Suppose flow A, B and C are sharing the same physical link and they are all headed to different destinations at a next hop. Assuming that the next hops of flow B and C are congested, the packets of flow A will also not be able to proceed due to the blocking packets of flow B and C that are ahead when there is only one queue, i.e., a single virtual channel within a link. Multiple virtual channels, however, reduce the probability of flows affecting each other by providing more possible paths. (In Figure 1-1(c), at least the packets of flow A in the top virtual channel can move forward.)

Using a standard dynamic VC allocation scheme, however, the head-of-line blocking effects are not sufficiently removed since all flows can be assigned to any VC within a

link. In the previous example, once all four VCs accept packets from flow *B* or *C*, flow *A* will again be blocked, preventing it from moving on to the next node. In this thesis, we demonstrate that through judicious separation of flows during static allocation or by introducing a restriction during dynamic allocation at runtime, we can minimize head-of-line blocking and maximize throughput.

1.2 Out-of-Order Packet Delivery is a Problem

In-order packet delivery in a network is a widely-assumed and critical abstraction for a wide range of application protocols such as file transfer protocols and optimized cache coherence protocols (e.g., [15, 16]); for example, Hennessy & Patterson begin the description of their cache coherence protocol with "first, we assume that the network provides point-to-point in-order delivery of messages" [15, p. E–7]. Implementations of direct-communication computation models such as stream computing (e.g., StreamIt [34]) also require that packets be delivered in the order they were sent, as do explicit message-passing applications. Indeed, in-order delivery is so widely taken for granted that it is often not specifically addressed.

While basic variants of dimension-ordered routing guarantee in-order delivery, improving performance by adding multiple dynamically allocated virtual channels as described in Chapter 1.1, which is a popular solution to ameliorate head-of-line blocking, allows packets from the same flow to be buffered in multiple VCs on a given link, and, in effect, creates multiple virtual paths for each flow, compromising in-order guarantees.

In Figure 1-2, one of the two channels (upper one) is blocked by some packets while the other channel (lower one) is not experiencing any blocking. Since packets from the same flow can be assigned to any VC, packets on the different channels may experience different congestion and travel times, resulting in out-of-order delivery.

This issue can be addressed by resorting to packet reordering: each packet is tagged with a sequential serial number, and any packets that arrive out of order are stored in a reorder buffer at the destination node until all of their predecessors have been received so that they can be reordered. This induces significant hardware cost, as the reorder buffers at

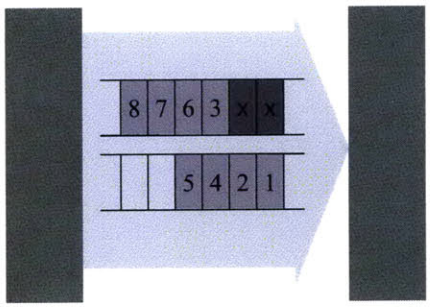
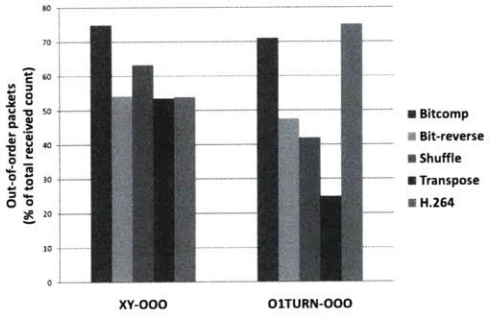
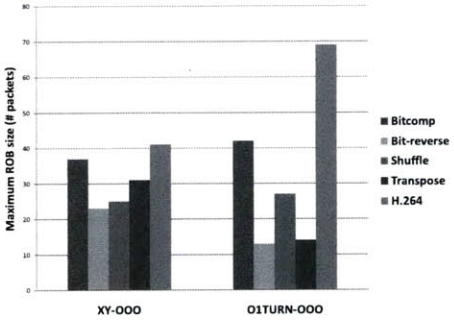


Figure 1-2: Dynamic VC allocation scheme under multiple VCs incurs out-of-order packet delivery. Numbers in the packets represent the order they are injected into the network.

each node must be quite large to ensure that all out-of-order packets can be stored, and, in the worst case, raises the specter of deadlock or requires expensive retransmission.



(a) Percentage of packets received out of order



(b) Maximum space requirement observed in any reorder buffer for any flow

Figure 1-3: Reorder costs under 4 virtual channels

Figure 1-3(a) shows the percentage of packets that arrived out of order in the experiments under five different traffic patterns for dimension-ordered routing and O1TURN under 4 virtual channels. Figure 1-3(b) shows the highest number of flits from any one flow simultaneously waiting in any reorder buffer; that is, it indicates the minimum reorder buffer size required to avoid dropping and retransmitting packets. The buffer size observations confirm that out-of-order packet arrival is a significant problem across a wide variety of loads, and shows that per-flow reorder buffers at each destination must be relatively large to avoid expensive retransmission. The high percentage of packets received out of order indicates that the reorder buffer and reordering logic must operate at close to line

rate, effectively excluding any software-level solution. Since one such buffer may have to be implemented for each flow arriving at a given destination, and the efficiency demands would require very fast storage, the cost of reorder buffer space alone in a store-and-reorder scheme would be significant.

To avoid deadlock scenarios when reorder buffers are not large enough, the network must be able to dynamically limit the number of outstanding packets of a given flow, either by dropping and resending packets, or by sending packets only after enough preceding packets have been delivered. This approach works well when the processing element (PE) at each node is significantly faster than the network (as is the case, for example, with multi-computer networks like the Internet), however, it is less appropriate for NoCs where the on-chip network fabric is very fast compared to the PE. While the reordering protocol can be implemented in hardware to improve performance in such cases, the amount and complexity of the necessary logic can be daunting.

1.3 Thesis Outline

In this thesis, we introduce a restriction on virtual channel allocation to address both head-of-line blocking and out-of-order packet delivery; any flow can reside in only one virtual channel within any given link. We propose two virtual channel allocation schemes that both implement this restriction. One implements this restriction in a static way before execution, and the other by a dynamic way during runtime: Static Virtual Channel Allocation and Exclusive Dynamic Virtual Channel Allocation (EDVCA).

To effectively reduce head-of-line blocking, static VC allocation achieves judicious separation of flows by exploiting a priori knowledge of the application's traffic patterns, resulting in significantly better throughput. Static VC allocation also ensures that each flow uses a single VC per node and packets arrive in order since each flow uses only a specific, statically allocated VC in each node. Furthermore, it provides us a method to ensure deadlock freedom for arbitrary minimal routes when two or more virtual channels are available.

Exclusive Dynamic Virtual Channel Allocation (EDVCA) is a dynamic VC allocation

scheme, which guarantees in-order delivery under dynamic VC allocation by ensuring that a flow is traveling via at most one path *at any one instant*. When combined with multi-VC dimension-ordered routing, our method guarantees deadlock-free in-order packet delivery at a fraction of the hardware cost and complexity of packet reordering approaches and without the overhead of including a serial number in each packet. Moreover, EDVCA significantly improves network performance for traffic patterns susceptible to head-of-line blocking, while offering performance equivalent to standard dynamic VC allocation on other traffic.

On average, static VC allocation schemes exceed the performance of standard dynamic VCA by 25%, and EDVCA outperforms standard dynamic VCA by 18% on our benchmarks when 4 virtual channels are used. In addition, our novel bandwidth-sensitive oblivious routing with minimal routes (BSORM), which is deadlock-free through appropriate static VC allocation, outperforms basic dimension-ordered routing by 35%.

The rest of this thesis is structured as follows. First, related work is summarized in Chapter 2. Chapter 3 describes how we statically allocate channels to flows at each link when oblivious routing is used, and also describes 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 avoid deadlock and optimize performance. We show how an analysis of the classical turn model [13] 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. In Chapter 4, we propose Exclusive Dynamic Virtual Channel Allocation (EDVCA), a VC allocation scheme which guarantees in-order delivery under dynamic VC allocation. Next, Chapter 5 details a router architecture that supports these methods, and discusses implementation costs compared to a baseline oblivious virtual channel router design. Chapter 6 offers performance analysis of static VC allocation, EDVCA and standard dynamic VC allocation via extensive cycle-accurate simulation with synthetic as well as application traffic patterns, and Chapter 7 concludes the thesis.

Chapter 2

Background and Related Work

2.1 Routing Techniques

Dimension-ordered routing (DOR) is one of the basic deterministic routing methods, and it applies to a broad class of networks, including the 2D mesh we consider here [8]. Packets traverse each dimension in a predefined order, traveling along one dimension until they have reached the destination node coordinate along that dimension, at which point they switch to the next dimension. Under a single virtual channel, DOR guarantees in-order packet delivery within each flow, but when it comes to multiple channels, implementations with dynamically allocated VCs compromise this guarantee since packets may pass each other in neighboring VCs. DOR has low implementation cost due to its hardware simplicity, but it pays the cost of poor worst-case and average-case throughput for mesh networks.

Valiant and Brebner proposed a routing scheme that randomly chooses an intermediate node every packet and employs DOR to route from the source node to the intermediate node and then from there to the destination node [36]. Since this algorithm spreads all traffic through the entire network, it achieves optimal worst-case throughput. In terms of latency and average-case behavior, however, the algorithm is likely to behave poorly, since even traffic between neighboring nodes may incur significant delays in traveling to and from the intermediate node. Moreover, different packets from the same flow will choose different intermediate nodes, i.e., different paths, resulting in out-of-order packet delivery.

ROMM [23, 24] basically applies the Valiant algorithm, but it limits the choice of inter-

mediate nodes to only within the minimum rectangle defined by the source and destination nodes, thus making it a minimal routing algorithm. Although ROMM guarantees minimum routes, its load balancing is not optimal, and it may saturate at a lower throughput than DOR in 2D torus networks [35] and 2D mesh networks [31]. While increasing the number of phases can reduce congestion, it comes at the cost of additional hardware complexity and additional virtual channels to avoid deadlock. Like Valiant, ROMM may deliver packets out of order.

In O1TURN [31], Seo *et al* show that simply balancing traffic between the XY and YX variants of DOR routing not only guarantees provable worst-case throughput but also matches average-case behavior of ROMM for both global and local traffic. Generally, it performs very well in practice with very little extra hardware cost and no transmission overhead. Since packets from the same flow may experience different congestion along the two possible routes, however, O1TURN does not guarantee in-order packet delivery.

Classic adaptive routing schemes include the turn routing methods [13] and odd-even routing [4]. These are general schemes that allow packets to take different paths through the network while ensuring deadlock freedom but do not specify the mechanism by which a particular path is selected. An adaptive routing policy determines what path a packet takes based on network congestion. Many policies have been proposed (e.g., [6, 14, 17, 32, 33]). Since adaptive routing algorithms alter the route in response to network conditions, they can also deliver packets out of order.

2.2 Bandwidth-Aware Routing

Palesi *et al* [27] 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 [5]; 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 [18]; this framework selects possibly non-minimal routes that conform to an acyclic CDG typically derived from a turn model. In this thesis, our focus is virtual channel allocation schemes for traditional oblivious routing methods (e.g., DOR, ROMM, Valiant) as well as for bandwidth-sensitive oblivious routing. For arbitrary minimal routes, we can always perform static VC allocation to avoid deadlock (assuming ≥ 2 virtual channels) (cf. Chapter 3.2).

2.3 VC Allocation

Flow-Aware Allocation (FAA) [1] reduces the effects of head-of-line blocking by allowing only one packet per virtual channel at each cycle. It works similar to EDVCA and can preserve the order of packets if used with single-path routing algorithms. However, FAA focuses only on mitigating head-of-line blocking, not on in-order point-to-point communication since it defines network flows only by destinations, regardless of sources. Dynamic virtual channel allocation schemes to improve throughput by intelligent buffer management have also been proposed (e.g., [25]). These schemes do not, however, provide in-order guarantees. It is possible to integrate EDVCA with such as those in [25].

2.4 In-Order Packet Delivery

Few routing scheme designs address out-of-order packet delivery. Within the Network-on-Chip (NoC) context, Murali et al [22] describe a multi-path in-order scheme where sequentially numbered packets belonging to a given flow are delayed at switches where distinct paths used by the same flow join (or cross). This scheme relies on a static assignment of flows to links; moreover, their re-ordering method contemplates only packets within one flow and either does not consider the possibility of deadlock when separate flows block each other or makes an unrealistic assumption of a private virtual channel for each flow.

More generally, ensuring in-order delivery via buffering out-of-order packets at the destination node and reordering them (and, if necessary, dropping and retransmitting) has

been around since the dawn of computer networking, and is employed, for example, in the Transmission Control Program [2, 3], the precursor of the ubiquitous Transmission Control Protocol [29]. TCP combines destination-buffered reordering with window-based flow control and acknowledgements piggybacked on return packets.

Chapter 3

Static VC Allocation

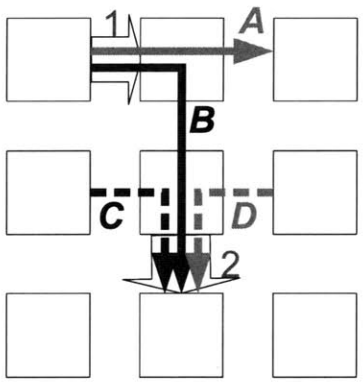
3.1 Static VC Allocation in Oblivious Routing

We assume the router design described in Chapter 5.1 with support for static VC allocation as described in Chapter 5.2. Since each link has multiple VCs, the assignment of channels to flows is done on a *per link* basis.

3.1.1 Dimension-Ordered Routing (DOR)

On a mesh, dimension-ordered routing corresponds to either XY or YX routing. Figure 3-1 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.

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 than the number of VCs, we can avoid entanglement by assigning one channel per flow. Otherwise, in order to mitigate the effects



	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 3-1: Motivation for Static Allocation

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 is 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.

3.1.2 ROMM and Valiant

The ROMM [24] and Valiant [36] 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.

While the original ROMM and Valiant choose random intermediate nodes per packet, however, we modify them to *static* ROMM and *static* Valiant; since each flow path should be defined before runtime in order to apply our entanglement framework to statically allocate VCs, we allow one randomly chosen route for each flow.

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.[1]

3.2 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 Chapter 5.1 with the static VC allocation support of Chapter 5.2).

3.2.1 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*

[1] Here, we assume 2-phase ROMM and Valiant. We require a minimum of N VCs to avoid deadlock for N-phase ROMM and Valiant.

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 thesis, our techniques also apply to other flow control schemes.

With a single VC per link or dynamic VC allocation, packet routes that conform to an acyclic channel dependence graph avoid network deadlock [8]. This is also a necessary condition unless false resource dependencies exist [30].

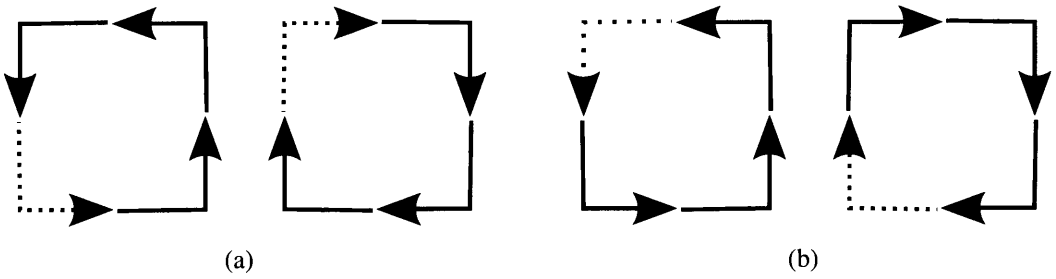


Figure 3-2: Turns allowed (solid) and disallowed (dotted) under (a) the West-First turn model and (b) the North-Last turn model.

Turn models [13] are a systematic way of generating deadlock-free routes, and have been used for adaptive routing. Figure 3-2 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.[2]

3.2.2 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. A variation of this method that restricts routes to a given

[2]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.

acyclic channel dependence graph (CDG) so as to ensure deadlock freedom under dynamic virtual channel allocation for one or more virtual channels is presented in [18]. We show how any set of *minimal* routes produced using any routing method can be made deadlock-free through appropriate static VC allocation (cf. Chapter 3.2.3); our argument for deadlock freedom invokes the turn models of Figure 3-2.

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 [37]: $w(e) = \frac{1}{\tilde{c}(e)-d_i}$, 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 3-3(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.

3.2.3 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.

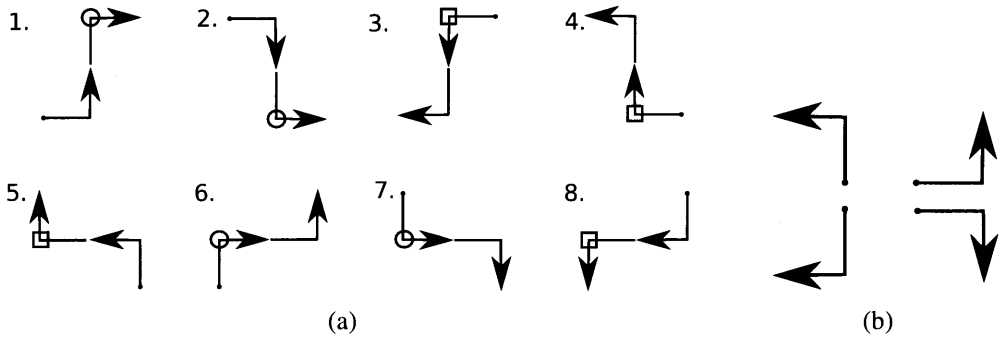


Figure 3-3: (a) The eight different two-turn minimal routes on a 2-D mesh. (b) The four (out of a possible eight) different one-turn routes on a 2-D mesh that conform to both the West-First and North-Last turn model.

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

Proof: Consider, without loss of generality, the case of 2 VCs. Figure 3-3(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 3-3(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 3-3(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 BSORM algorithm of Chapter 3.2.2, 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.

3.3 Effects

Dynamic VC allocation under multiple VCs allows packets from all flows to be queued in any VC within a link, and thus, it may not efficiently remove the effects of head-of-line blocking and also lose the guarantee of in-order packet delivery. Static VC allocation, however, can enhance throughput by separating or isolating flows (cf. Figure 3-1). The fact that each flow is allocated to exactly one VC within a link also indicates that each flow is transferred through effectively one single path across the network, ensuring in-order packet delivery.

Moreover, static VC allocation provides us methods to avoid deadlock for any minimal routes over an $n \times n$ mesh under ≥ 2 VCs. When virtual channels are dynamically allocated, we require the set of routes to conform to an acyclic channel dependence graph (CDG), which, for example, could correspond to a turn model [13]. Static allocation, on the other hand, enables a routing algorithm to choose more diverse routes compared to the dynamic allocation case by separating flows (cf. Chapter 3.2).

Statically allocating a VC to each flow also 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 [28] 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).

One of the downsides of the static scheme is that it relies on off-line route and requires knowledge of traffic patterns, which may not be possible when computation is not known

beforehand. This incurs another restriction that routes should not change during the run-time, and therefore, routing algorithms with randomness such as traditional ROMM [24] and Valiant [36] cannot be used. Another possible downside is that 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. These performance tradeoffs will be explored through extensive simulation using DARSIM [19], a cycle-accurate NoC simulator (cf. Chapter 6).

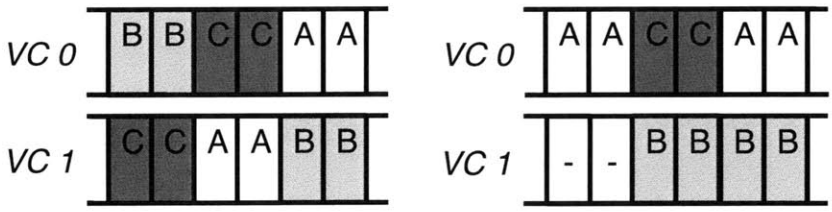
Chapter 4

Exclusive Dynamic VC Allocation (EDVCA)

As described in Chapter 1.2, standard dynamic VC allocation does not guarantee in-order packet delivery even with single-path routing when there are multiple VCs, because packets may depart out of arrival order if they are queued in different virtual channels. We solve this problem using Exclusive Dynamic Virtual Channel Allocation (EDVCA).

In a nutshell, EDVCA prevents packets from any single flow from using more than one VC at a given ingress *at any given instant*. When a snapshot of the network is examined, packets travel via a single path, which ensures in-order packet delivery and reduces head-of-line blocking. At the same time, the VC being used for a specific flow at a given ingress can change over time, and when the network state is examined over a longer period, flows may appear to be using multiple VCs at each ingress port, spreading incoming traffic among available VCs.

In what follows, we assume a standard ingress-queued virtual-channel router with wormhole routing and credit-based inter-link flow-control [9]. In such designs, each packet arriving at an ingress port is immediately queued in a VC buffer, and forwarded via four steps: route computation (RC), virtual channel allocation (VCA), switch allocation (SA), and switch traversal (ST), sometimes implemented as separate pipeline stages for efficiency. All flits in a packet are forwarded contiguously, so the first two stages (RC and VCA) only perform computation for the head flit of each packet, returning cached results for the



(a) dynamic VCA (b) EDVCA

Figure 4-1: Comparison of VCA schemes

remaining flits (See Chapter 5.1 for details).

4.1 EDVCA Logic

To achieve the goal of each flow occupying at most one VC per node, EDVCA alters the virtual channel assignment logic and the related credit update mechanism. When allocating a next-hop VC to a packet from a flow f, the following principles apply:

- if no next-hop VC contains packets from f, assign the packet to any available VC; if no VCs are available, stall the packet and try to allocate again in the next cycle (emulates dynamic VCA)

- if some next-hop VC v already contains packets from f, and v is available, assign the packet to v; if v is not available, stall the packet and try to allocate again in the next cycle.

Figure 4-1 illustrates how our scheme might allocate virtual channels for packets from three hypothetical flows, A, B, and C, for a two-VC ingress port. Traditional dynamic allocation might assign packets from each flow to both VCs, as shown in Figure 4-1(a); exclusive dynamic VCA, on the other hand, will assign packets from one flow to only one VC, as shown in Figure 4-1(b). Thus, when the third packet in flow B arrives, it is assigned VC 1 because VC 1 already has packets from B; similarly, the third packet in flow A is assigned VC 0. When the third C packet arrives, it must wait either until VC 0 has space (in which case it can go into VC 0) or until all other C packets in VC 0 have been forwarded (in which case it can go into either VC). Note that, while Figure 4-1(b) only

shows a snapshot at a particular instant and the VC assignment might change at a later time (for example, with VC 0 and VC 1 reversed), exclusive dynamic VCA will never result in the situation shown in Figure 4-1(a).

4.2 Credit Tracking and VCA Details

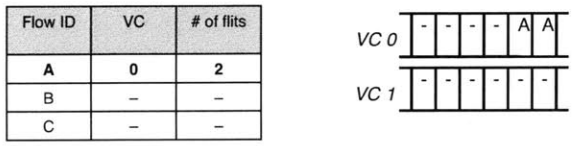
To implement EDVCA, the credit management logic has to change. In a traditional VC router, each router keeps track of the number of free slots (credits) in every next-hop VC queue in a VC Credit Table (VCT):

- when router r sends a packet to a VC queue in the next-hop router r', the credit counter for that VC is decreased to reflect the packet's arrival in the remote queue;
- when a packet departs from the next-hop VC queue, the next-hop router r' sends back a credit update signal to r identifying the relevant VC;
- when the credit update is received, r increments the credit counter for the relevant VC.

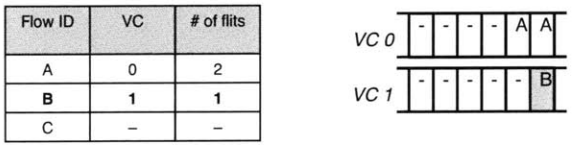
Whenever a new packet (i.e., a head flit) arrives at the head of a VC queue, the VC allocation stage assigns it to a next-hop VC using a scheduling algorithm (such as iSLIP [20]). The switch allocation logic then considers a packet's flits for crossbar traversal as long as the credit counter for the relevant remote VC queue remains positive.

EDVCA adds a Flow Assignment Table (FAT), which ensures that a flow is assigned to at most one next-hop VC at any given time. The FAT is a table with entries per flow, where the entry for each flow lists the currently assigned VC (if any), and the number of flits from that flow remaining in that VC. Figure 4-2 illustrates how such a table might be updated when packets depart from the current node for the next hop, and when they are forwarded on from the next-hop node:

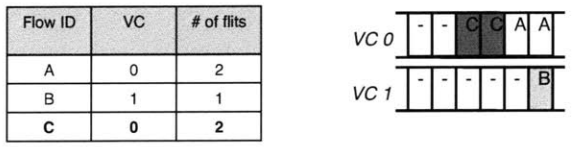
- when a flit on flow f departs for a next-hop VC v,
  - increment the # flits counter in FAT[f],



(a) Two flits in flow *A* departs for next-hop VC 0

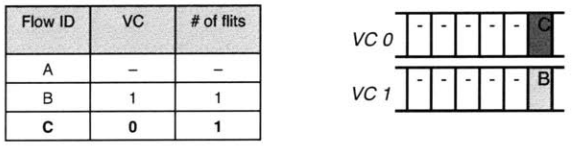


(b) One flit in flow *B* departs for next-hop VC 1



(c) Two flits in flow *C* depart for next-hop VC 0

(d) Two flits in flow *A* are forwarded from next-hop VC 0



(e) One flit in flow *C* is forwarded from next-hop VC 0

Figure 4-2: Tracking remote VC contents

  - decrement the credits for v in the VCT, and
  - send a credit update for f to the previous-hop node;

- when a credit-update message for flow f in next-hop VC v arrives,
  - decrement the # flits counter in FAT$[f]$, and
  - increment the VCT credits for v.

The VC allocation logic then queries the FAT in parallel with standard dynamic VCA, and overrides the VCA result if the flow already exists in some next-hop VC:

- choose the next available VC v according to a dynamic VC scheduling algorithm

(e.g., iSLIP);

- in parallel, query the FAT entry for flow f, FAT[f];
- if FAT[f] names a VC and # flits > 0, assign flow f to the VC in FAT[f];
- otherwise, assign flow f to v and set FAT[f] $\leftarrow$ (VC $= v$, # flits $= 0$).

4.3 Effects

Dynamic VCA gives up in-order delivery even if packets from a given flow always follow the same sequence of nodes (e.g., dimension-order routing) since it allows packets from the same flow to be buffered in different VCs, and packets in different VCs may pass each other; for example, in Figure 4-1(a), the packets from flow A could depart in any of six possible orders. EDVCA restores the in-order delivery guarantee for such routing schemes by observing that packets are still delivered in order if, within each node, all packets in the same flow are buffered in the same VC *at any given time*.[1]

The on-the-fly restriction of each flow to a single VC per node in EDVCA limits head-of-line blocking even more than resorting to multiple VCs alone. While in standard dynamic VCA a single flow can block all available VCs, EDVCA ensures that at most one VC can be blocked by any one flow; moreover, while multiple flows may be assigned to the same VC and one may still block the other, this effect is temporary, as each flow will be reassigned to a fresh VC every time all of its packets have left the next-hop VC queues.

Finally, our VC allocation scheme is also free of deadlock provided the underlying routing regime is also deadlock-free. The only new dependencies are between packets waiting to be forwarded and the next-hop VCs they are targeting, a kind of dependency already present in any kind of ingress-queued router.

In addition to EDVCA, in which each flow exclusively uses a VC at any given time, we also considered an orthogonal scheme where each VC may only contain one flow at any

[1]The in-order guarantee holds, of course, in the case of a single VC (since, at each node, all packets are buffered in the same ingress queue and so must depart in arrival order), and for static VCA like WOT [12] or BSOR [18] (because all packets *in the same flow* stay in the same queue). Unlike those schemes, however, EDVCA significantly alleviates head-of-line blocking without requiring prior knowledge on traffic patterns to build static configurations.

given time (but gives up in-order guarantees because packets from the same flow may use more than one VC), and a scheme that combines both constraints (thus guaranteeing in-order delivery). While both schemes still outperformed DOR in most of our experiments, they did not perform quite as well as EDVCA; since flows tended to monopolize VCs and cause unfairness, we judged these variants to be impractical and do not detail them here.

Chapter 5

Implementation Cost

5.1 Conventional Architecture

5.1.1 Virtual Channel Router

We assume a typical virtual channel (VC) router on a 2-D mesh network as a baseline [9, 21, 28], but our methods can be used independent of network topology and flow control mechanisms.

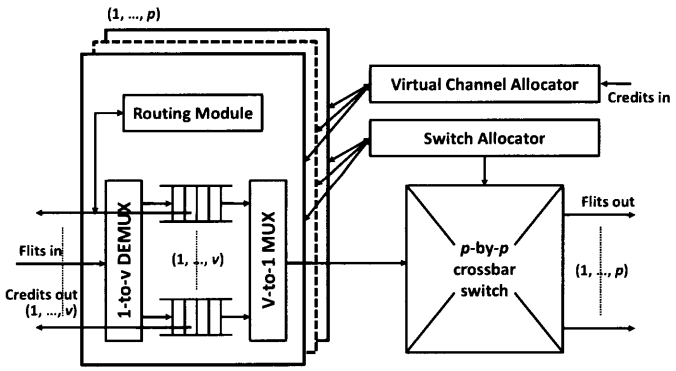


Figure 5-1: Typical virtual-channel router architecture.

Figure 5-1 illustrates a typical ingress-queued virtual-channel router with wormhole routing and credit-based inter-link flow-control [9]. The router contains three major control modules: a routing module, a VC allocator, and a switch allocator. They determine the

next hop, the next virtual channel, and when a switch is available for each flit, respectively. Once a packet arrives at an ingress port, it is queued in a VC buffer, and forwarded via four steps: route computation (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). All flits in a packet are forwarded contiguously, so the first two steps (RC and VA) are only performed for the head flit of each packet, returning cached results for the remaining flits.

5.1.2 In-Order Network

Ensuring in-order packet delivery in a fast on-chip network incurs significant additional cost except for the case of basic routing schemes (e.g., DOR) under a single virtual channel. A store-and-reorder scheme with good performance would require significant buffer space at the destination as discussed in Chapter 1.2.

If we limit the size of reorder buffer, it would not be able to hold all out-of-order packets until they get reordered, requiring retransmission logic or end-to-end flow control to ensure that the buffers at the destination cores do not overflow. This requires memory space at the source node and degrades throughput due to additional protocols.

Instead of dedicated reorder buffers at network nodes, the main memory of processing elements can be used to store out-of-order packets. If this is the case, any out-of-order packets must be removed from the network at line rate in order to prevent deadlock, which imposes the severe requirement that the memory be fast enough to keep up; equipping a processing element with enough such memory to satisfy both the reorder buffers and the applications it runs can be prohibitively expensive.

Thus, conventional implementations of in-order on-chip networks either require a large amount of fast memory at each node, or significantly degrade the performance due to expensive protocols.

5.2 Static VC Allocation

5.2.1 Table-based Routing

The only architectural change required for static VC allocation and application-aware oblivious routing (see Chapter 3.2) is in the routing module and the VC allocation 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 and VCs 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. Chapter 3.2.3). The modules that require architectural changes from a conventional virtual channel router in order to implement static VC allocation are highlighted in Figure 5-2.

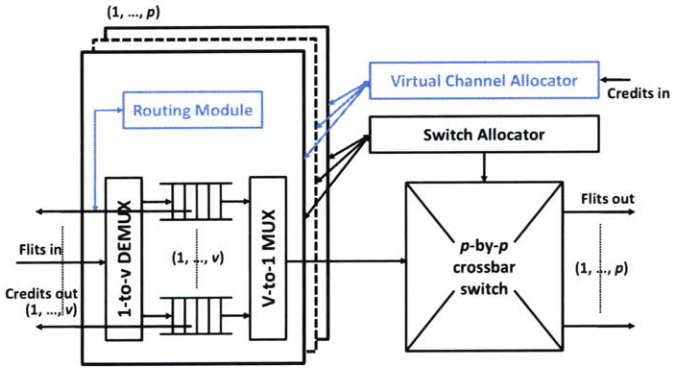


Figure 5-2: Components that require modification for static VC allocation is colored in blue. The routing module and the virtual channel allocator should be table-based.

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 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 5-3(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 5-3(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.

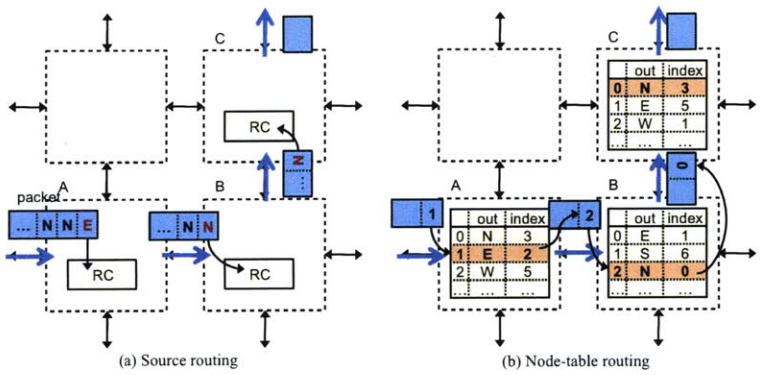


Figure 5-3: The table-based routing architecture (a) Source routing (b) Node-table routing

We have described two routing module designs, namely source routing and node-table routing. Both routing methods are widely known and have been implemented in multiple routers (e.g., [7, 11]). While the basic routing algorithms (e.g., DOR) are often implemented by the fixed logic, the router architecture based on node-table routing replaces the route computation with a table look-up. Although the table look-up can take longer than the fixed routing logic for simple routing schemes, the router's clock frequency is not affected since the latency of a pipelined virtual-channel router is mostly dominated by other routing stages such as virtual channel allocation stage.

What makes it even better is that with the table-based routing architecture and static VC allocation scheme, the VC allocation step can be much simplified. Since the routing table now holds both the routing and VC information, head flits in the route computation (RC)

stage can obtain both the output port and the next VC by a single table look-up.

The memory size required for the routing table is trivial. 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. Static VC allocation adds a few more 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; again with 256 entries, this results in an increase of 96 bytes, Therefore, the routing table can be accessed in a single cycle without impacting clock frequency.

5.3 EDVCA

5.3.1 Flow Assignment Table

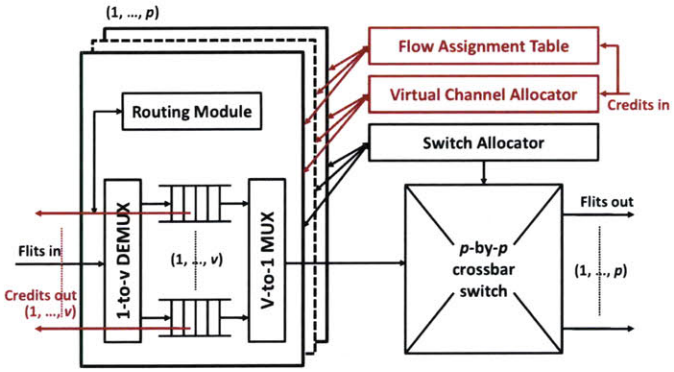


Figure 5-4: Components that need to be modified or added to implement EDVCA is colored in red.

Figure 5-4 highlights the architectural differences from a classical virtual channel router (cf. Figure 5-1) with dynamic VC allocation when there are p ports per node and v VCs per port. These include the Flow Assignment Table, credit management logic, and a VCA-stage multiplexer for EDVCA.

The main area cost is the Flow Assignment Table (FAT), which ensures that packets from a single flow will only be buffered in one VC at any snapshot; that is, during the VCA stage, the router must check whether any VCs at the next-hop node already contain the

relevant flow, and, if so, restrict itself to that VC. To accomplish this, the FAT maps each flow ID to a remote VC assignment (three bits for a 8-VC system) and a flit count (three bits for eight-flit queues per VC); the table key is a flow ID, which, assuming a flow for each source-destination pair in an 8×8 2D mesh, might be twelve bits.

While at first blush it might appear that in a system with many flows such a table might have to be quite large–for example requiring 4096 entries in an 8×8 system–observe that only a much smaller subset of flows will ever pass through any single node. Also, in our 2D mesh schemes based on DOR, this is limited to flows where either the source or the destination node are on the same row or column as the relevant transit node. In an N-by-N mesh, for example, the maximum number of flows that pass through a single node with DOR is the order of N^3. Therefore, for an 8×8 mesh with 8 eight-flit VCs, the FAT at each node requires less than 400 bytes for DOR.

Even if N is very large, the FAT size can be optimized because only flows *with any flits in the next-hop VCs* need to be tracked. This is relatively few flows: for a node with 8 eight-flit queues per ingress, at most 64 different flows can be buffered at each ingress. Also, the table can be smaller than the number of simultaneously buffered flows: if the table becomes full, the next packet for an unknown flow stalls until there is space in the table and performance degrades gracefully. In this case, each entry needs to be tagged by its flow ID.

One efficient implementation of FAT with a reduced number of entries is to use a content-addressable memory (CAM) addressed by flow ID, one for each ingress if each flow has only one entry port (e.g., in DOR), or one bigger table for each node if flows can arrive from any direction (e.g., in some adaptive routing schemes). Since the contents are keyed by the flow ID and not the next-hop node, the lookup could even be pipelined in a router with separate route and VCA stages, allowing for a slower (or larger) table. Another solution is a two-stage lookup: the first would query a per-flow "flow-renaming" table and retrieve a mapping to a "local" flow ID which would then be used to look up a FAT implemented in directly-addressable memory.

A modicum of additional delay in the VC allocation stage is associated with multiplexing between the VCA output and a successful FAT lookup (see Chapter 4.2). This, however,

is a two-way multiplexer of very few signals (two bits in a 4-VC system), and represents negligible cost.

5.3.2 Credit Update System

The only remaining overhead stems from tracking the number of flits for each flow in the next-hop VCs (Figure 4-2). This can be easily handled by slightly increasing the size of the existing credit update messages. This is because the credit update system now tracks remote VC contents *for each flow* (see Chapter 4.2), and the credit update messages must carry a flow ID instead of a VC ID (the corresponding VC IDs are locally retrieved from the FAT table). While sending flow IDs instead of VC IDs in every credit update message does increase the number of bits and therefore the bandwidth used for credit updates, the few extra wires are cheap in terms of area and do not affect the size of the crossbar switch; furthermore, if desired, the wire count can be decreased significantly by sending credit updates less often, with corresponding graceful decrease in performance.

These small overheads compare favorably with the resources and logic required to implement a typical store-and-reorder scheme for in-order packet delivery. Unlike reorder buffer size, the additional table memory does not grow with maximum packet size, and the additional VC allocation and credit update logic is much simpler than the logic needed to reorder, acknowledge, and possibly retransmit packets.

Chapter 6

Experimental Results

We have evaluated the performance of static VC allocation and EDVCA via extensive simulation on synthetic benchmarks as well as a load profile obtained from a parallel implementation of an H.264 video decoder. We also compare our routing scheme (BSORM) with DOR, which is a basic oblivious routing algorithm.

6.1 Experimental Setup

For our experiments, we used DARSIM [19], a cycle-accurate NoC simulator based on an ingress-queued virtual-channel router architecture. To avoid unfairness effects resulting from a particular combination of round-robin strategy and packet arrival rate, VCs in switch and VC allocation are considered in random order and greedily matched. To estimate the impact of out-of-order packets, we implemented a store-and-reorder strategy, although reorder buffer sizes are not limited and so retransmission is never necessary.

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.

For each simulation, the network was warmed up for 240,000 cycles and then simulated

for 960,000 cycles to collect statistics, which was enough for convergence.

Topology	8×8 2D mesh
Routing	DOR, ROMM, Valiant, O1TURN, BSORM, BSOR
VC allocation	dynamic VCA, static VCA, EDVCA
Link bandwith	1 flit/cycle
VCs per port	2, 4, 8
VC buffer size	8 flits
Avg. packet size	8 flits
Traffic workload	transpose, shuffle, bit-complement, H.264 decoder profile
Warmup cycles	240,000
Analyzed cycles	960,000

Table 6.1: Network configuration summary

Table 6.1 summarizes the configurations used for the experiments that are shown in this thesis.

6.2 Static VC Allocation and Dynamic VC Allocation

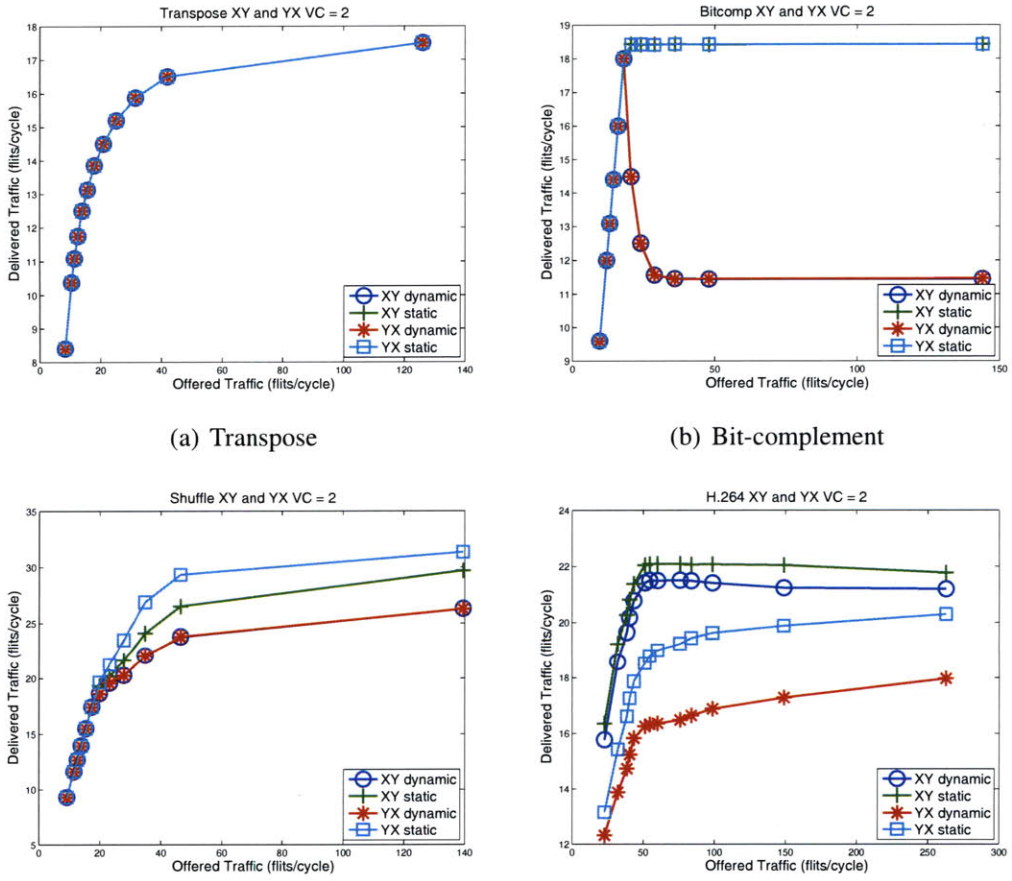
6.2.1 DOR, ROMM and Valiant

Figure 6-1 shows the performance of XY and YX routing with 2 VCs for static and dynamic VC allocation for various benchmarks. We can see that static VC allocation always 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 3-1. The performance benefit, however, depends on benchmarks; while we did not observe the throughput improvement under the transpose pattern, static VC allocation outperforms dynamic VC allocation by 61% under the bit-complement traffic pattern. This significant improvement under the bit-complement pattern is because XY and YX routing algorithms with dynamic VCA result in unstable throughput performance due to unfair scheduling caused by head-of-line blocking. The reason why static VC allocation performs the same as dynamic VC allocation for transpose is because the routes generated by XY routing for the transpose traffic pattern do not suffer from head-of-line blocking, meaning once different flows are merged, they never diverge afterwards. For this type of routes that are inherently free from

head-of-line blocking, separation of flows will not result in better performance, which is a rare case for real application traces.

Figure 6-2 and Figure 6-3 show the performance of ROMM and Valiant under static and dynamic allocation for 4 and 8 VCs, respectively. [1] 2-phase 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 and benchmarks, static VC allocation performs better than dynamic allocation by 25% on average.



(a) Transpose

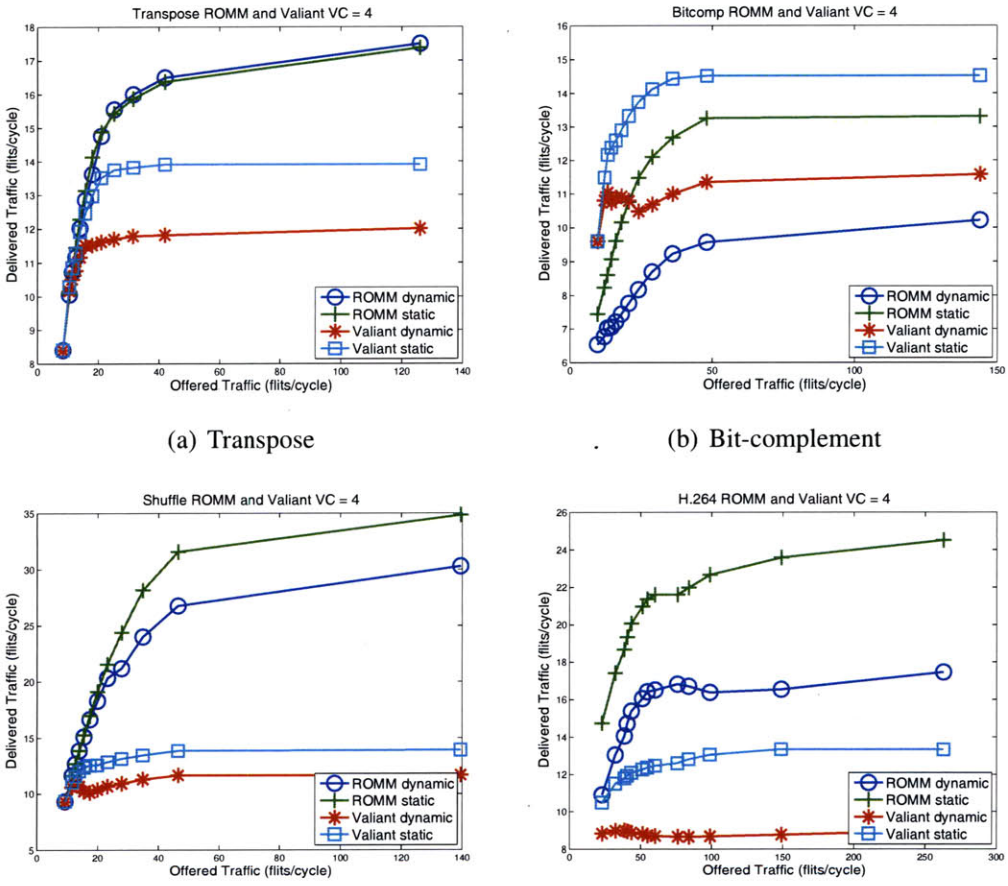
(b) Bit-complement

(c) Shuffle

(d) H.264

Figure 6-1: Throughput for DOR under static and dynamic allocation with 2 VCs

[1] We use static ROMM and static Valiant for the experiment as described in Chapter 3.1.2. For fair comparison, we use the same routes for both dynamic and static VC allocation.



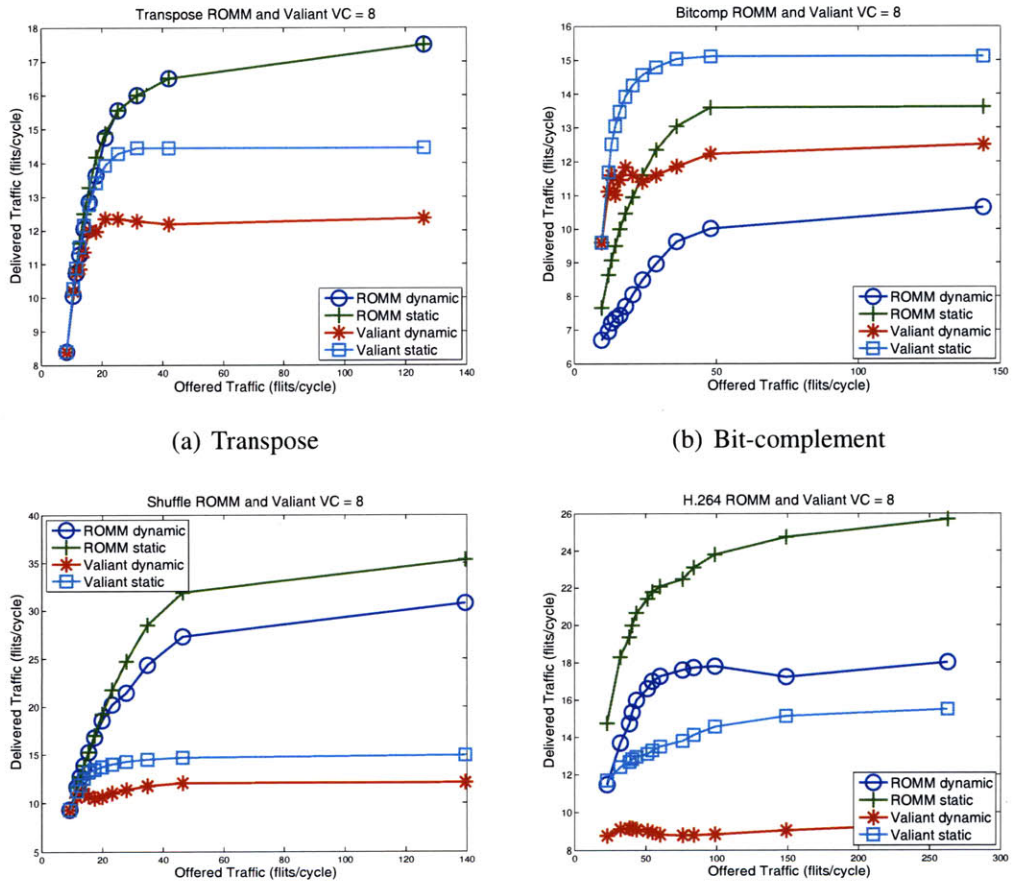
(a) Transpose

(b) Bit-complement

(c) Shuffle

(d) H.264

Figure 6-2: Throughput for ROMM and Valiant under static and dynamic allocation with 4 VCs



(a) Transpose

(b) Bit-complement

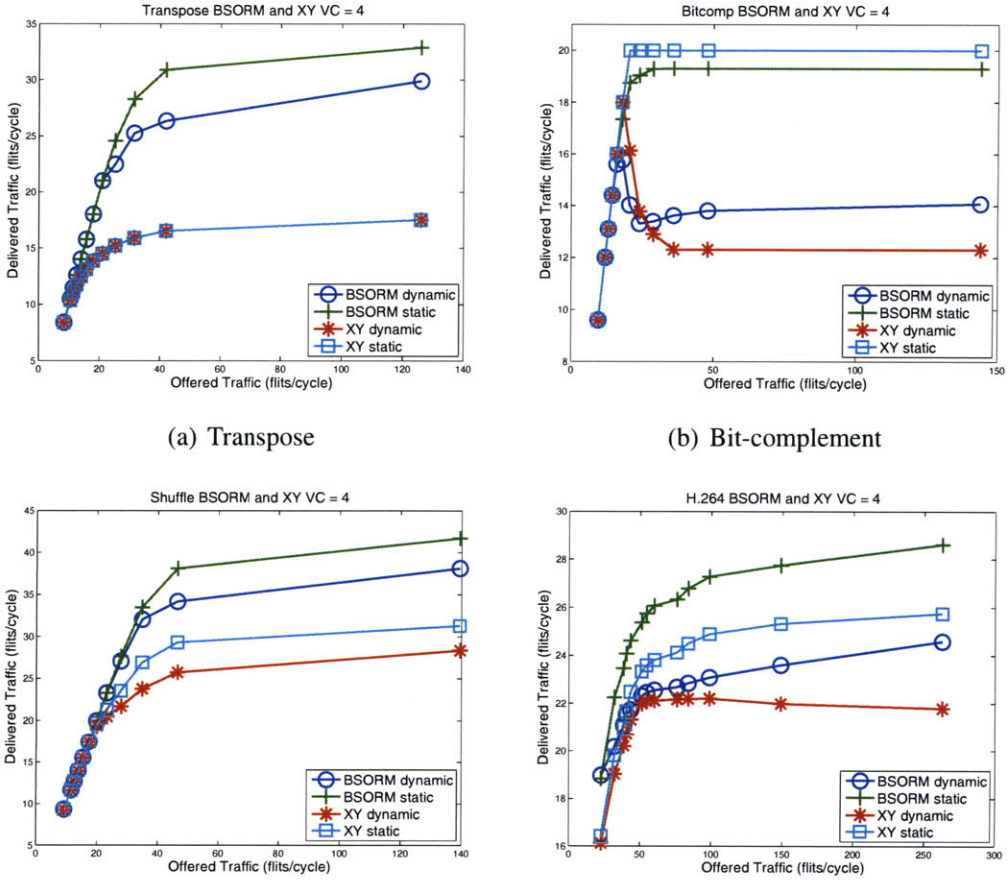
(c) Shuffle

(d) H.264

Figure 6-3: Throughput for ROMM and Valiant under static and dynamic allocation with 8 VCs

6.2.2 BSORM

Now we evaluate the performance of our newly proposed oblivious routing algorithm, BSORM. Figure 6-4 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 minimal routes, and since the routes are not deadlock-free by themselves, we should break these routes into two sets to avoid deadlock, as described in Chapter 3.2.3. We perform static allocation or assume dynamic allocation within each set.



(a) Transpose

(b) Bit-complement

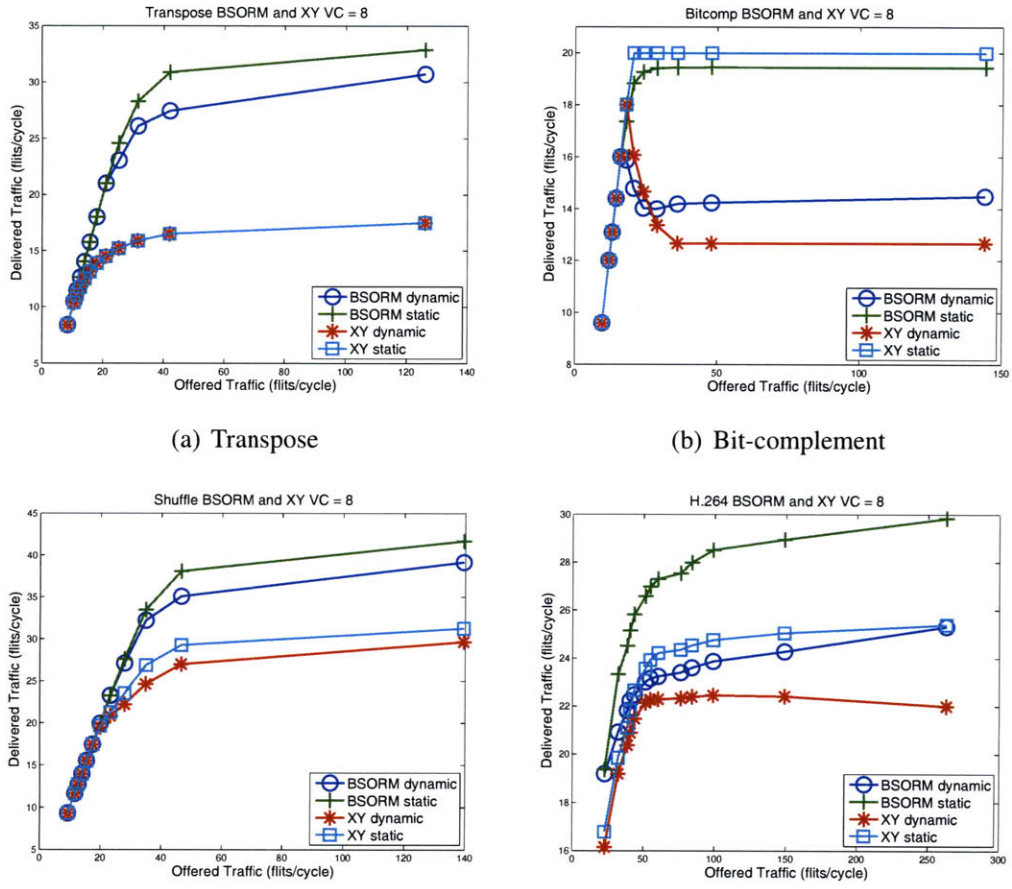
(c) Shuffle

(d) H.264

Figure 6-4: Throughput for BSORM and XY under static and dynamic allocation with 4 VCs

Figure 6-5 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.

For our benchmarks, BSORM performs better than DOR by 35% on average since the bandwidth-aware routing effectively reduces MCL. BSORM with static allocation outperforms dynamic allocation for the same reasons as in DOR.



(a) Transpose

(b) Bit-complement

(c) Shuffle

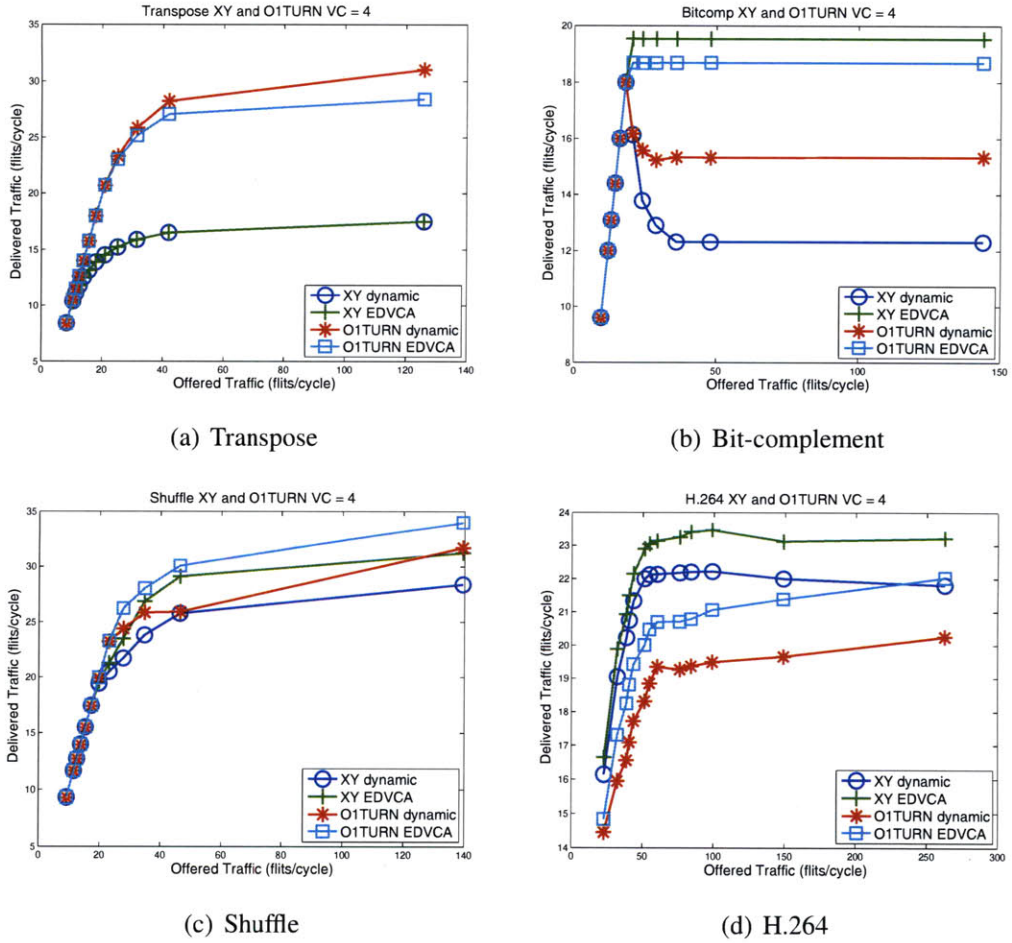
(d) H.264

Figure 6-5: Throughput for BSORM and XY under static and dynamic allocation with 8 VCs

6.3 EDVCA and Dynamic VC Allocation

6.3.1 DOR and O1TURN

In this chapter, we compare the performance of EDVCA with dynamic VC allocation under DOR–XY and O1TURN. Figure 6-6 shows the throughput for XY and O1TURN under dynamic VCA and EDVCA with 4 VCs. For all benchmarks except for transpose, we notice a uniform performance improvement of EDVCA over dynamic VC allocation. Under the bit-complement pattern, for example, EDVCA provides a significant performance benefit by reducing the effects of head-of-line blocking. These results are similar to the results from static VC allocation (DOR–XY).



(a) Transpose

(b) Bit-complement

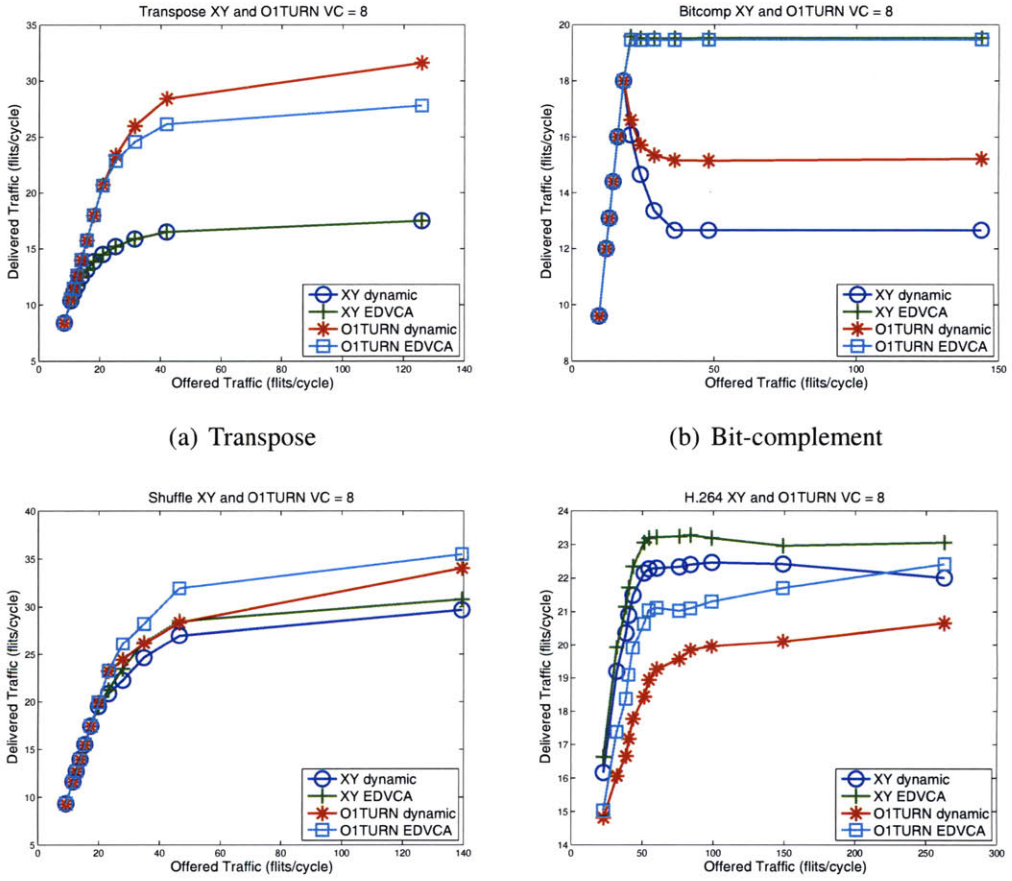
(c) Shuffle

(d) H.264

Figure 6-6: Throughput for XY and O1TURN under dynamic VCA and EDVCA with 4 VCs

In terms of in-order packet delivery, while XY with EDVCA guarantees in-order delivery, O1TURN with EDVCA can result in out-of-order packet delivery due to its path diversity. However, we believe EDVCA should be used regardless of whether in-order delivery is a requirement or not, as we observed that the throughput is uniformly improved by using EDVCA.

Results under 8 virtual channels are shown in Figure 6-7, and they show the same trends as the results under 4 VCs.



(a) Transpose

(b) Bit-complement

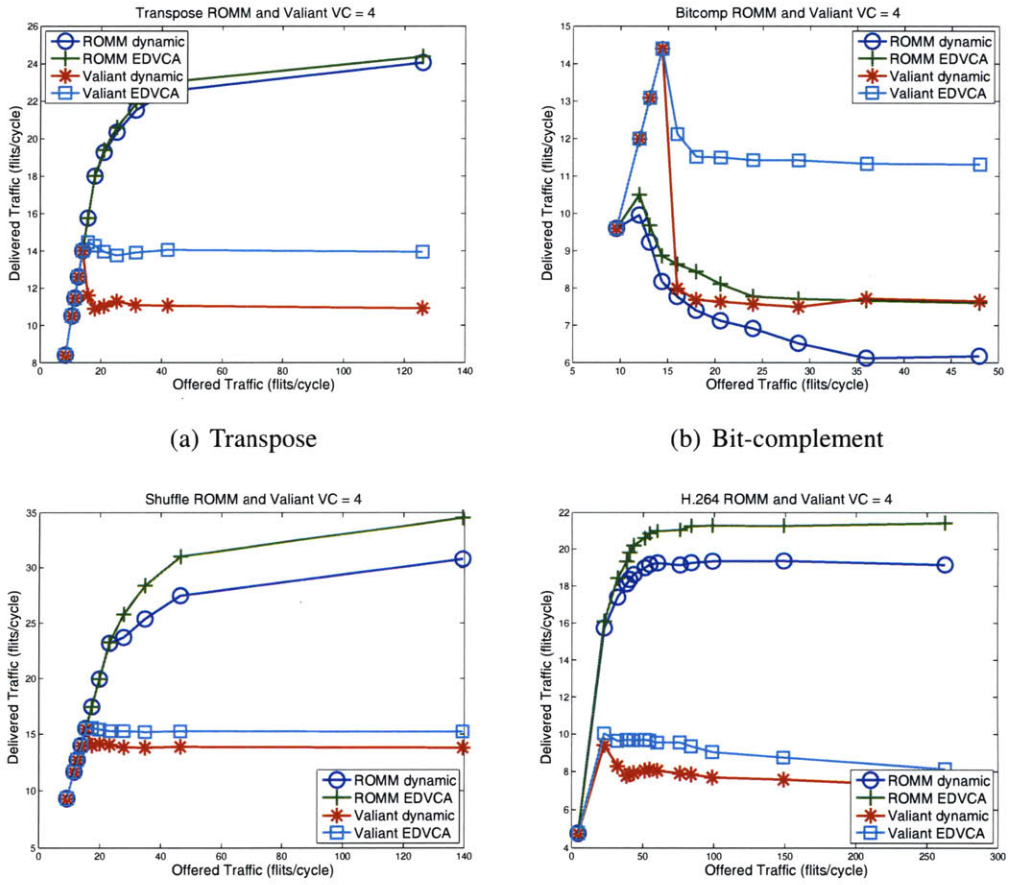
(c) Shuffle

(d) H.264

Figure 6-7: Throughput for XY and O1TURN under dynamic VCA and EDVCA with 8 VCs

6.3.2 ROMM and Valiant

Although we need to modify the original ROMM and Valiant algorithms in order to apply static VC allocation (cf. Chapter 3.1.2), EDVCA does not require any modification and can be directly applied to those algorithms since it can be done at runtime. However, ROMM and Valiant with EDVCA does not guarantee in-order delivery for the same reason as in O1TURN; they can have many different paths for the same source and destination pair by randomizing the intermediate node during execution. But again, we believe EDVCA is a better choice than dynamic VCA in terms of the throughput performance even without in-order requirement, as we can see in the results.



(a) Transpose

(b) Bit-complement

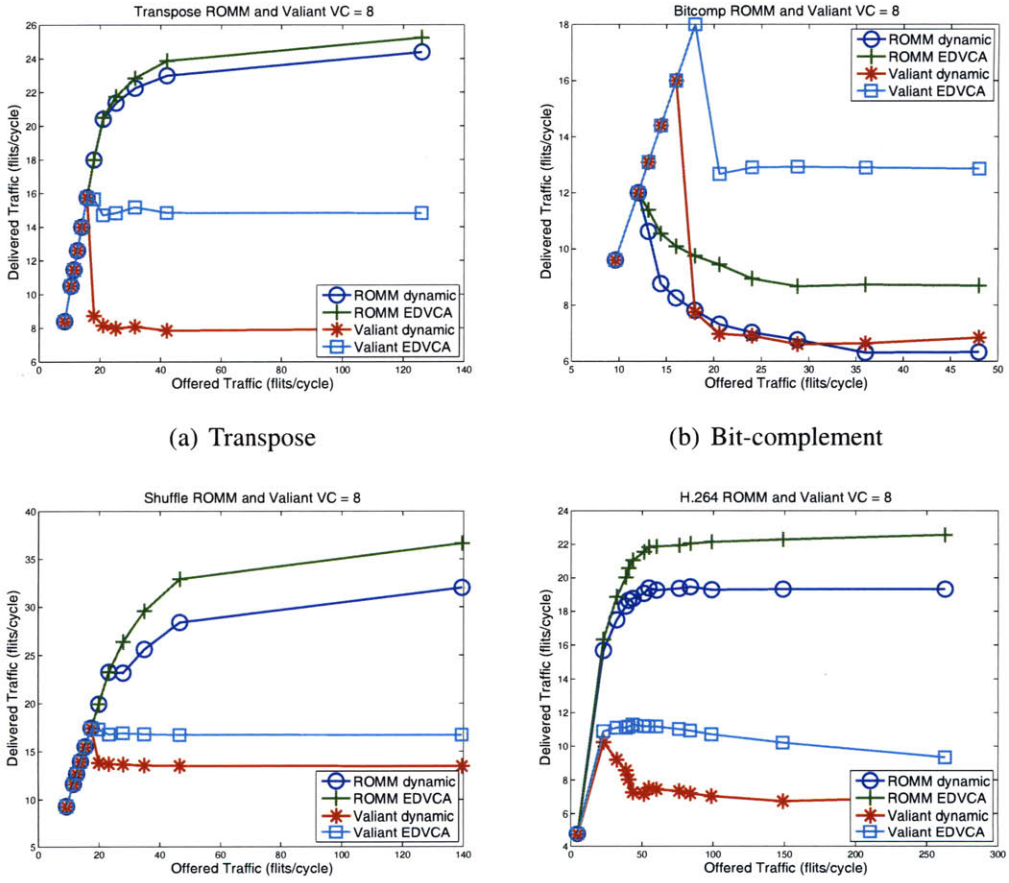
(c) Shuffle

(d) H.264

Figure 6-8: Throughput for ROMM and Valiant under dynamic VCA and EDVCA with 4 VCs

Figure 6-8 and Figure 6-9 show the performance of ROMM and Valiant under dynamic VCA and EDVCA with 4 and 8 VCs, respectively. For all traffic patterns, EDVCA outperforms dynamic VC allocation scheme for both ROMM and Valiant.

Under our benchmarks and various routing algorithms (e.g., DOR–XY, O1TURN, ROMM and Valiant), EDVCA performs better than dynamic VC allocation by 18% on average by efficiently mitigating head-of-line blocking.



(a) Transpose

(b) Bit-complement

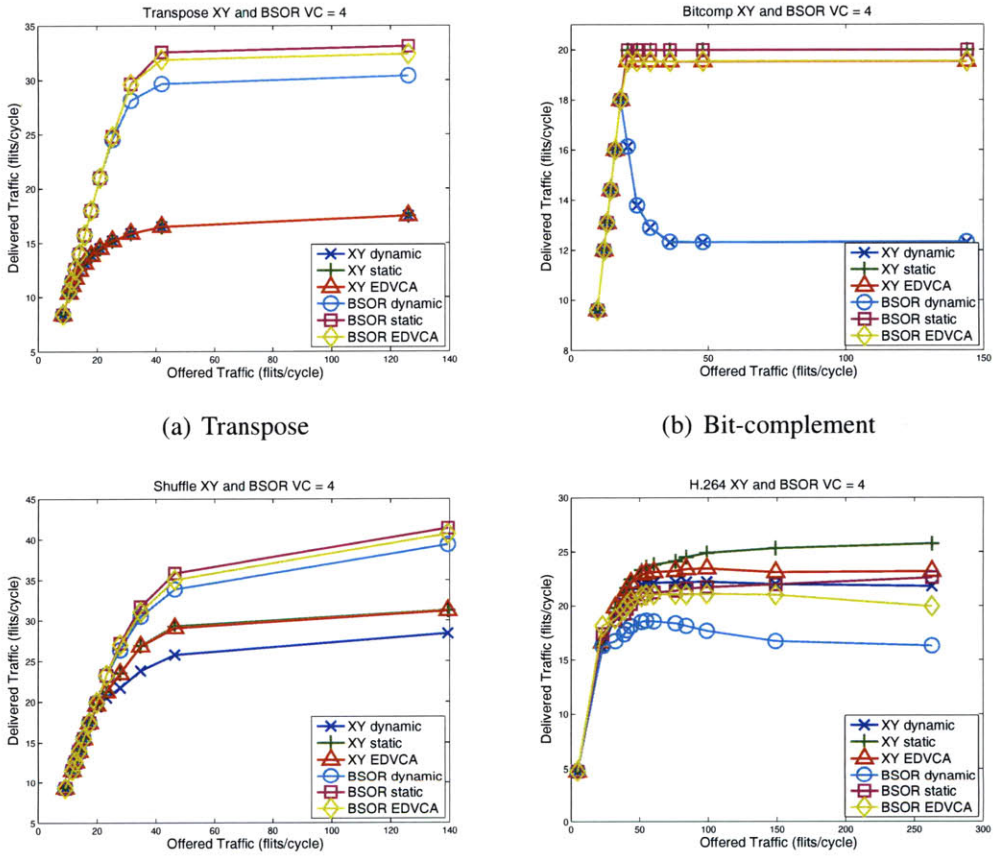
(c) Shuffle

(d) H.264

Figure 6-9: Throughput for ROMM and Valiant under dynamic VCA and EDVCA with 8 VCs

6.4 Static VC Allocation and EDVCA

Finally, we compare the performance of our two proposed VC allocation schemes: static VC allocation and EDVCA. In order to evaluate both schemes under the same routing algorithms, we have to restrict them to single-path routing, and thus we perform the evaluation under DOR–XY and BSOR, which is a bandwidth-sensitive oblivious routing algorithm that minimizes maximum channel load (MCL) while guaranteeing deadlock-freedom [18].



(a) Transpose

(b) Bit-complement

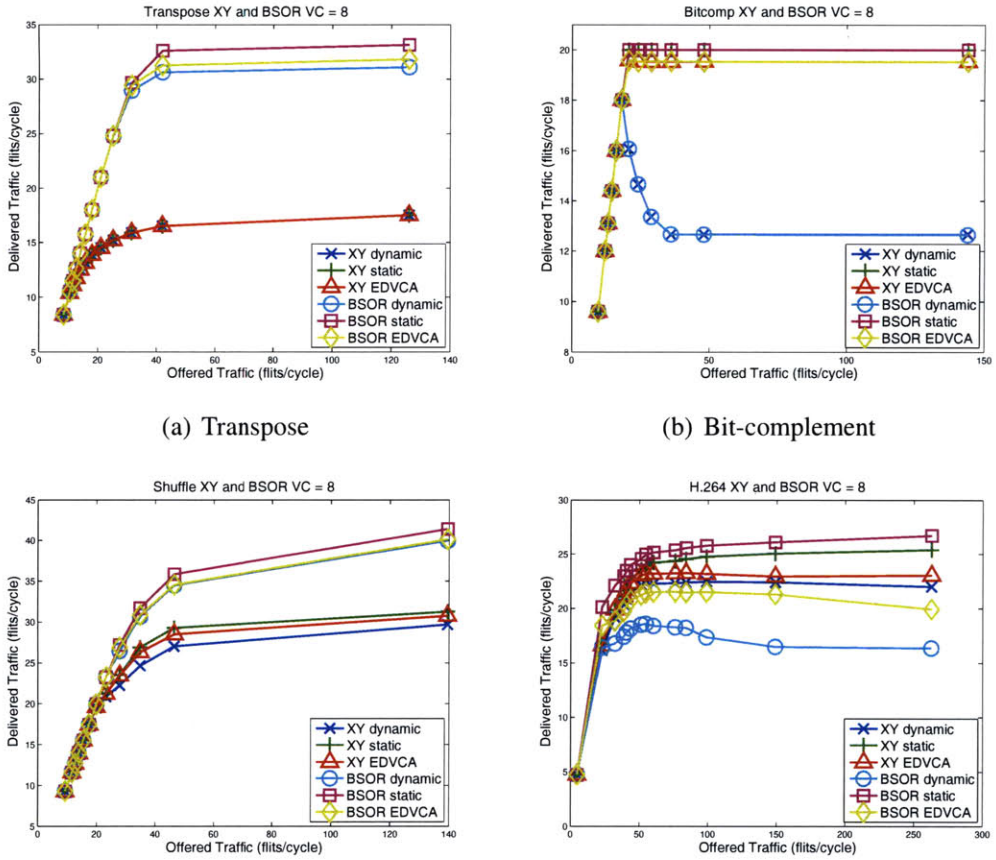
(c) Shuffle

(d) H.264

Figure 6-10: Throughput for XY and BSOR under static VCA, dynamic VCA and EDVCA with 4 VCs

Figure 6-10 shows the performance of static VCA, EDVCA, and the baseline dynamic VCA under XY and BSOR with 4 VCs (Results are similar for the case of 8 VCs, which is shown in Figure 6-11). For all traffic patterns and for both DOR–XY and BSOR, we can observe that within the same routing algorithm, static VC allocation and EDVCA always

outperform dynamic VC allocation, and between static VCA and EDVCA, static VCA always performs better than or as good as EDVCA. This indicates that we can reduce the effects of head-of-line blocking more efficiently when we have global knowledge of traffic patterns.



(a) Transpose

(b) Bit-complement

(c) Shuffle

(d) H.264

Figure 6-11: Throughput for XY and BSOR under static VCA, dynamic VCA and EDVCA with 8 VCs

Chapter 7

Conclusions

Multiple virtual channels are often considered a basic element of modern router architecture in on-chip networks for various reasons such as Quality-of-Service (QoS) guarantees, performance isolation, deadlock avoidance, etc. Conventional dynamic VC allocation under multiple VCs, however, maintains the problem of head-of-line blocking since packets from the same flow can be assigned to any VC, meaning a single congested flow can take up all VCs blocking other flows behind. What makes it worse is that it compromises the guarantee of in-order packet delivery, which is an assumption that most applications depend on. Since packets can travel across multiple different paths (equivalently, different VCs), they result in out-of-order arrival at the destination core.

In this thesis, we present two different virtual channel allocation schemes to address these problems: Static VC Allocation and Exclusive Dynamic VC Allocation (EDVCA). Static VC allocation assigns virtual channels to flows during off-line computation using the information of routes, minimizing the total amount of VC sharing among all flows. It also provides methods to avoid deadlock for an arbitrary set of minimal routes with ≥ 2 VCs, enabling us to propose a new oblivious routing scheme, BSORM. Exclusive Dynamic VCA, on the other hand, is a dynamic virtual channel allocation scheme that assigns only one VC to any single flow within a link at any instance without requiring any priori knowledge.

Our results show that static VC allocation and EDVCA both effectively reduce head-

of-line blocking, often significantly improving throughput compared to dynamic VC allocation scheme for existing oblivious routing algorithms. Moreover, the restriction on VC allocation that the same flow can reside in only one VC within any link at a time gives each flow effectively a single path from source to destination, guaranteeing in-order packet delivery.

We also demonstrate that when rough estimates of bandwidths are given, 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.

Although static VC allocation and EDVCA are realizing a similar restriction on flow assignment, they are different in that static VC allocation is done by precomputation before execution, while EDVCA assigns VCs to flows at runtime. From this fact, a computer architect now has architectural choices depending on the target application of the chip he/she wants to build. In case of application-specific chips, in which case we can easily obtain routes information in advance, we may want to use static VC allocation to achieve the best throughput by minimizing head-of-line blocking. When designing general-purpose CPUs, however, it is unrealistic to statically assign all flows in advance, and thus we can use EDVCA that performs almost as well as static VC allocation without the need of priori knowledge. Again, when used with single-path routing algorithms, they both guarantee in-order packet delivery. Our proposed schemes require only minor, inexpensive changes to traditional oblivious dimension-order router architectures, and can even lower the hardware cost by obviating the need for expensive buffers and retransmission logic.

There are many avenues of research that can be pursued further. For example, our static VC allocation and the BSORM algorithm do not consider time-varying behavior in terms of traffic. When traffic varies significantly over time during execution, it is critical to take the dynamic feature into account. Evaluating our schemes with real applications will provide us with more insights about on-chip networks. Future work also includes implementing these schemes in hardware.

Bibliography

[1] Arnab Banerjee and Simon Moore. Flow-Aware Allocation for On-Chip Networks. In *Proceedings of the* 3^{rd} *ACM/IEEE International Symposium on Networks-on-Chip*, pages 183–192, May 2009.

[2] V. G. Cerf, Y. Dalal, and C. Sunshine. Specification of Internet Transmission Control Program. RFC 675, Internet Engineering Task Force, December 1974.

[3] V. G. Cerf and R. E. Kahn. A Protocol for Packet Network Communication. *IEEE Trans. Comm.*, 22:637–648, May 1974.

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

[5] 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.

[6] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 04(4):466–475, 1993.

[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] 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.

[12] 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.

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

[14] P. Gratz, B. Grot, and S. W. Keckler. Regional Congestion Awareness for Load Balance in Networks-on-Chip. In *In Proc. of the 14th Int. Symp. on High-Performance Computer Architecture (HPCA)*, pages 203–214, February 2008.

[15] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2nd edition, September 1996.

[16] Natalie D. Enright Jerger, Li-Shiuan Peh, and Mikko H. Lipasti. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In *MICRO '08: Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 35–46, Washington, DC, USA, 2008. IEEE Computer Society.

[17] H. J. Kim, D. Park, T. Theocharides, C. Das, and V. Narayanan. A Low Latency Router Supporting Adaptivity for On-Chip Interconnects. In *Proceedings of Design Automation Conference*, pages 559–564, June 2005.

[18] 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*, pages 208–219, June 2009.

[19] Mieszko Lis, Keun Sup Shim, Myong Hyon Cho, Pengju Ren, Omer Khan, and Srinivas Devadas. Darsim: a parallel cycle-level noc simulator. In *Proceedings of the 6th Annual Workshop on Modeling, Benchmarking and Simulation (MoBS)*, June 2010.

[20] Nick McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE Trans. Networking*, 7:188–201, April 1999.

[21] 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.

[22] S. Murali, D. Atienza, L. Benini, and G. De Micheli. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. In *Proceedings of DAC 2006*, pages 845–848, July 2006.

[23] Ted Nesson and S. Lennart Johnsson. ROMM Routing: A Class of Efficient Minimal Routing Algorithms. In *in Proc. Parallel Computer Routing and Communication Workshop*, pages 185–199, 1994.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] J. Postel. Transmission Control Protocol. RFC 793, Internet Engineering Task Force, September 1981.

[30] 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.

[31] 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.

[32] Arjun Singh, William J. Dally, Amit K. Gupta, and Brian Towles. GOAL: a load-balanced adaptive routing algorithm for torus networks. *SIGARCH Comput. Archit. News*, 31(2):194–205, 2003.

[33] Arjun Singh, William J. Dally, Brian Towles, and Amit K. Gupta. Globally Adaptive Load-Balanced Routing on Tori. *IEEE Comput. Archit. Lett.*, 3(1), 2004.

[34] William Thies, Michał Karczmarek, and Saman P. Amarasinghe. StreamIt: A Language for Streaming Applications. In *Proceedings of the International Conference on Compiler Construction*, LCNS, pages 179–196, Grenoble, France, 2002.

[35] Brian Towles and William J. Dally. Worst-case traffic for oblivious routing functions. In *SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 1–8, 2002.

[36] 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.

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