TxComm: Transforming Stream Communication for Load Balance, Efficiency, and Fault-tolerance in Networks-on-Chip

Ahmed Hamdy Abdel-Gawad
Electrical and Computer Engineering, Purdue University, aabdelga@purdue.edu

Mithuna Thottethodi
School of Electrical and Computer Engineering, Purdue University, mithuna@purdue.edu

TxComm: Transforming Stream Communication for Load Balance, Efficiency, and Fault-tolerance in Networks-on-Chip

Ahmed Hamdy Abdel-Gawad

Mithuna Thottethodi

TR-ECE-11-11

May 9, 2011

School of Electrical and Computer Engineering

1285 Electrical Engineering Building

Purdue University

West Lafayette, IN 47907-1285

TxComm: Transforming Stream Communication for Load Balance, Efficiency, and Fault-tolerance in Networks-on-Chip

Ahmed Hamdy Abdel-Gawad Mithuna Thottethodi
Purdue University
{aabdelga,mithuna}@purdue.edu

Abstract

Recent work has examined using application-specific knowledge of streaming communication to optimize network routing (for throughput/performance) and/or design (for simpler hardware). However, previous techniques have assumed that the communication streams are directly mapped to networks-on-chip. In contrast, this paper explores the use of communication transformations (TxComm) to achieve (1) higher throughput via better network load balance, (2) more efficient network utilization, and (3) better fault-tolerance, while retaining the communication semantics of the original streaming application. Specifically, we propose two transformations: stream fission and stream fusion. (While fission and fusion transformations have been applied to computation in streaming programs, we are the first to propose fission and fusion transformations for stream communication.) Stream fission splits streams of communication to multiple streams that may be routed over independent network paths to achieve better network load balance. Stream fusion targets multicast communication and fuses multiple streams to effectively capture the well-known benefits of tree-based multicast, which include more efficient link utilization. Both techniques can be integrated in an integer linear program formulation that executes at compile time. Another key component of TxComm is the use of *free routing* which serves two key purposes. First, it boosts the performance of fission and fusion. Second, it enables application-specific fault-tolerance. Evaluations with a suite of StreamIT benchmarks show that TxComm achieves significant performance improvement over prior application-specific (non-transformed) routing techniques. On the fault-tolerance front, TxComm achieves similar performance as a fault-free base case even when as many as 10% of links are faulty.

1. Introduction

Recent work has examined using application-specific knowledge of communication patterns to optimize network routing [8, 15] (for throughput/performance) and/or design (for simpler hardware) [6]. Common to all the above techniques is the observation that failing to exploit the known, application-specific traffic patterns in routing effectively imposes an opportunity cost. This paper expands the communication optimizations that are possible for such application-specific patterns by observing that there are ways to transform the patterns such that the transformed patterns are (a) equivalent in terms of communication; thus preserving original application semantics, and (b) more efficient (i.e., they reduce network channel usage) and better load-balanced (i.e., they distribute channel load in a better way) than untransformed communication. Further, we demonstrate that the transformations support simple extensions that enable application-specific fault-tolerance in the NoC.

Specifically, we examine two transformations. The first transformation is *fission* wherein a flow is split into multiple flows. Intuitively, fission can lead to better load balance when used on bottleneck flows. Our second transformation is *fusion* wherein multiple flows that are being used for multicast communication are fused to form a multicast distribution tree. There have been hardware techniques to create limited forms of communication fusion [7] wherein multicast circuits are opportunistically set-up/torn-down at runtime. Similarly, all forms of routing that use multiple packet paths (e.g., adaptive routing, and some randomized oblivious routing algorithms) mimic the effect of fission. However, in the context of application-specific routing, such run-time attempts at fission and fusion are either too limited (e.g., VCTM [7] can only set up multicast trees that are limited to dimension-ordered routes) or are inferior to compile-time routing which can exploit the known communication pattern [8]. We are the first to automatically apply the fission and fusion transformations at compile time to optimize communication, resulting in significant performance improvement beyond prior application specific routing techniques.

One key challenge in effectively using our transformations is the interaction of our transformations with deadlock-handling mechanisms in networks. Prior work has addressed the possibility of deadlock by either enforcing routing restrictions (e.g., BSOR [8]) to ensure acyclic channel dependence graphs (CDG) or used minimal routing in combination with virtual channels (VCs). Such routing restrictions used for deadlock-freedom can seriously limit the performance improvement offered by our transformations.

For example, consider a BSOR implementation which uses the "West-first" Turn-model based deadlock avoidance [4]. In such an implementation, all westward traversals must be performed first (as the name indicates) because it disallows turns to the west. While it is an elegant way to prevent deadlocks, consider its impact on fission. If the bottleneck flow has to traverse the westward path first, fission will not help reduce the load on the westward links because all the fissed flows must continue to be routed on the same westward links. Effectively, the purpose of fission is defeated.

We address the above problem by using *free routing* – routing without any restrictions that are typically enforced to avoid deadlocks. Free-routing unlocks the full potential of our transformations. However, naively using free-routing may cause deadlocks because unrestricted compile-time routing has the same potential for deadlocks as unrestricted (fully-adaptive) hardware-based routing. To resolve this dilemma, we observe that free routing will result in one of two outcomes. In the best case, there may be no cycles in the CDG, in which case deadlock freedom is guaranteed. However, there may be cycles, in which case we attempt to break the cycles by using virtual channels. Unlike in the case of minimal routing, where it has been proved that two VCs are adequate to break deadlock cycles [16], there is no such property in unrestricted (potentially non-minimal) routing. Consequently, we formulate the deadlock-free VC assignment problem as an ILP formulation which takes the number of VCs as one of its inputs and answers the question "*Can we avoid deadlocks using the given number of VCs?*".[1] If the VCs we have are adequate, again, deadlock-freedom is guaranteed. If both the above techniques fail (i.e., if there are cycles that cannot be eliminated using the available VCs), then we can always fall back on one of the two provably deadlock-free techniques (BSOR routed over an acyclic CDG, or BSOR-minimal with a deadlock-free, 2-VC design). When evaluated over 13 benchmarks and 3 network sizes (4×4, 6×6, and 8×8), we found that in a majority of the cases, there were no cyclic dependences even with free routing. Further, even the few cases that did have cyclic dependences became cycle-free with no more than four VCs. Unlike fission and fusion, free routing is not a program transformation. However, free routing boosts the performance improvements of fission and fusion by eliminating routing restrictions.

Another key advantage of free-routing is that it enables fault-tolerant application-specific routing. Routing algorithms that rely on a given topology-dependent deadlock-free routing algorithm (e.g., BSOR relies on the Turn model for mesh networks) cannot operate when there are permanent link or node failures because the remaining nodes/links may form an irregular topology for which the Turn Model does not guarantee deadlock-freedom and/or connectivity. In contrast, because free routing may achieve deadlock-free routing even when certain links are faulty, Tx-Comm can tolerate failures. While there exist other general-purpose fault-tolerant routing algorithms such as $up*/down*$ routing [13], they fail to exploit the application-specific communication patterns. The tradeoffs between application-specific fault-tolerance and general-purpose fault-tolerance are the same as in the fault-free case: on the one hand, the application-specific approach achieves higher performance; on the other hand, application-specific approach is constrained to be a compile time approach which requires stopping, recompiling and restarting the application.

The combination of all three methods yielded significant reductions in channel load (as found by the solver) for all of the 39 configurations. We observe that

[1] Note, we are not interested in the question "*How many VCs do we need to avoid deadlocks?,*" because our model assumes that applications are being compiled for fixed network hardware. Consequently we do not have the luxury of having as many VCs as we need.

for the most part, the problems can be solved in seconds/minutes (with a small number of cases requiring hours). Significant performance improvements (55%, on average) were also measured experimentally by simulation.

In summary, the main contributions of this paper are threefold.

- We optimize the network performance of application-specific communication beyond what has been done previously by transforming the communication patterns. Specifically, we automate the application of two communication optimizations – fission and fusion – for the StreamIT programming model.
- We propose best-effort *free-routing* in conjunction with a static VC assignment to achieve deadlock-freedom. Although the technique may have to fall back on routing restrictions in the worst-case, free-routing is effective in practice. It eliminates routing restrictions for all 39 application-specific communication patterns we examine.
- We demonstrate that *free-routing* enables compiler-assisted, fault-tolerance wherein faulty network links can be tolerated by avoiding those links while optimizing for performance over the remaining links. This ability is absent in prior application-specific routing techniques such as BSOR because they rely on an underlying deadlock-free channel dependency graph over which all routing is performed. With *free-routing* TxComm can tolerate up to 10% link failures while achieving performance comparable to fault-free BSOR.

We cast all the above techniques as integer linear programming (ILP) optimization problems, which enables the use of commercial solvers to automatically apply our techniques.

The rest of the paper is organized as follows. Section 2 provides a brief background on StreamIT, which is the stream programming model we use because StreamIT programs have fixed, application-specific communication patterns. Section 3 describes TxComm with details on how the transformations and the VC assignment for free-routing are formulated as ILP optimization problems. Section 4 describes our evaluation methodology. Section 5 discusses experimental results. Related work is described in Section 6. Finally, Section 7 concludes this paper.

2. Background

Streaming StreamIT applications are based on the framework of decomposing applications into a graphs of communicating "actors" or "filters" [18]. The nodes of the graph represent the computation while the edges represent the communication. The programming model relies on defining the computation as a filter. The StreamIT compiler [5] then fuses/fisses the filters based on the specified number of processors with the goal of maintaining the load balance among the nodes of the graph. The compiler also performs the placement step to associate an actor with a processor by using simulated annealing to optimize the communication cost imposed by the routing function [9].

Application-specific network routing Routing is the act of determining the output port a packet must be forwarded on at each router as packets traverse networks. In general-purpose routing in networks-on-chip (NoCs), a fixed routing algorithm based on the destination of each packet is hardwired in logic. In contrast, application-specific routing can exploit the fixed nature of communication patterns and use *programmable* routing to achieve better performance, as shown in BSOR by Kinsy *et al.* [8]. Because StreamIT applications also have a fixed communication pattern, similar application-specific routing may be used. Application-specific routing occurs in three distinct phases. At compile time, the compiler determines the routes for the application-specific flows. At application load time, the network "preloads" the network routes for various flows (as computed by the compiler) in to its programmable routing tables. At runtime, the packets corresponding to various flows simply use the preloaded per-flow routing table information for normal operation. Note the programmability of the routing table refers to one-time programming before the beginning of the application. The routing tables do not change during the execution of the application. TxComm modifies the compile-time route computation by using our fission and fusion transformations.

Our goals and our domain of interest Our focus is to improve the communication performance of StreamIT based applications. Because of this focus, we run into three direct implications. First, our improved network performance can lead to improved performance for network-bound applications. As a dual of such performance improvement, we can also imagine that our technique offers more opportunity for power savings in computation-bound applications. That is because computation-bound application can scale-down the voltage/frequency of networks, thus saving power without affecting performance. A better performing network can be frequency/voltage scaled more aggressively. We focus on the performance aspect of the power-performance duality and show that TxComm offers superior performance for network-bound applications. (Because of the duality principle, our results imply that TxComm can save power for computation-bound applications.)

Second, we assume the hardware-pipelined version of streaming (where spatially spread out actors are used to expose pipeline parallelism) as originally suggested for the RAW machine [17]. For traditional multicores, software-pipelining based orchestration has been suggested for traditional memory-hierarchy based, as well as scratch pad memory based multicores [10, 14]. However, we remain focused on the hardware-pipelined version because software pipelining can be memory-inefficient.

Third, the key metric to optimize in network-bound applications is the maximum channel load – which refers to the network load on the bottleneck link in the system. This bottleneck link directly controls network throughput and hence overall application throughput.

While the above three goals focus on improved performance for fault-free operation, we also examine the throughput that TxComm achieves in the presence of faulty links[2].

3. TxComm

This section describes the two transformations – fission and fusion – in TxComm. For each transformation, we present a high-level description as well as the specific ILP formulation to realize the transformation.

3.1 Flow Fission

Intuitively, flow fission improves throughput by fissing/splitting a unicast flow such that the channel load of the bottleneck link is reduced. Unlike randomized load balancing techniques for packet-based systems where the packets of a flow are routed in randomized fashion, our approach simply transforms the program to have

[2] In general, faulty nodes can be handled by changing the placement of actors. We focus on the challenge of application-specific routing in the presence of faulty links.

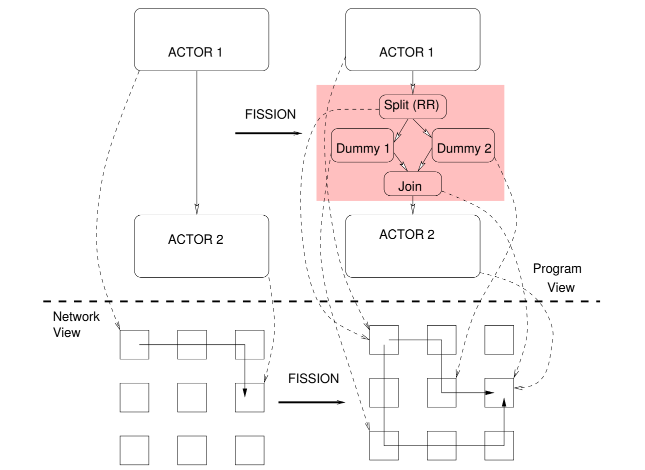


Figure 1. Program view and network view of fission

multiple flow-splits per flow. As shown in Figure 1, the process of fission can be viewed from a stream program point of view or from a network point of view. From a program point of view (upper half of Figure 1), fission interposes (a) a splitter, (b) dummy actors whose sole function is to act as a pass-through entity by popping data from upstream flows and pushing them on downstream flows and (c) a joiner. The interposed components are shown in a shaded rectangle. Note that splitting and joining need not be in an even ratio. As far as the network is concerned, the net effect is that the channel load that was previously in one flow (and one network path) is load-balanced across two network paths.

The above description is the logical view of fission. In practice, we formulate the fission transformation problem as an ILP optimization problem as described below.

3.1.1 ILP formulation

The challenge we face is to determine the following three concrete details: (1) the degree of fission for each flow, (2) the fraction of traffic carried on each fissed split of the flow, and (3) the paths each split of the flow traverses to minimize channel load.

We simplify the first choice by picking an "adequately large" degree of fission based on two observations (a precise definition follows). First, the maximum channel load monotonically decreases with increasing degree of fission. If the optimal channel load is achieved at a degree of fission k, increasing the degree of fission cannot result in a higher maximum channel load because the load on the excess flow-splits may be set to zero, effectively mimicking k-way fission. Second, increasing the degree of fission has a diminishing marginal impact on channel load. This is not surprising because the difference between a k-way split and a $k+1$-way split decreases with increasing k. Later, we show that the degree of fission indeed exhibits this property and shows no improvement beyond 4-way fission. Conservatively, we use 10-way fission (i.e., 10 is adequately large).

The above simplification which fixes the degree of fission enables us to cast the determination of the remaining two choices as an ILP formulation. Our definition of a flow is similar to that used in BSOR, which is the entire stream of unicast communication. However, because TxComm uses splittable flows to enable fission, we associate a variable for the load on each split of the flow (henceforth referred to as *flow-split*) rather than with the entire flow, as formally defined below.

DEFINITION 1. *Given a flow graph $G(V,E)$, where V is the set of vertices (routers) and E is the set of edges (channels), and given a set of m flows to route $W = \{W_1, \ldots, W_m\}$ where each flow $W_i = (s_i, d_i, l_i)$ consists of source s_i, destination d_i and channel load l_i, assuming $s_i \neq d_i$, the flow-split variables $f_{i,j}(u,v)$, $i = 1, \ldots, m$ and $j = 1, \ldots, k$, represents the load carried by j^{th} split of the i^{th} flow on the edge $(u,v) \in E$. The flow-split occupancy variables $b_{i,j}(u,v)$ are binary variables indicating whether the corresponding $f_{i,j}(u,v)$ has any flow or not.*

The objective consists of two sub-objectives[3]. Primarily we wish to minimize the maximum channel load. Secondarily, we wish to minimize the number of occupied links. The secondary objective ensures that, among multiple routing choices that yield the same maximum channel load, the choices that occupy the fewest links are chosen.

The constraints ensure that the following three properties hold. First, we ensure end-to-end channel load conservation for each flow; i.e., the sum of the loads on the flow-splits leaving the source (and a similar sum reaching the destination) is equal to the original (unfissed) flow. Second, we ensure that hop-by-hop flow conservation is enforced at intermediate routers. Finally, we ensure that, though non-minimal paths are allowed, only *simple* paths are traversed (i.e., no flow-split ever revisits the same node) by ensuring that there are no cycles in any flow-split's path. Note that the *simple path* constraint is unnecessary for BSOR because it uses routes that result in an acyclic CDG, thus avoiding cyclic-paths.

[3] Though we present them as two subobjectives, because of their priorities, they may be linearly merged into one composite objective function.

The formal ILP problem includes additional constraints that are important for correctness in corner cases (e.g., avoiding negative flow values, avoiding branching flows, ensuring source-to-destination connectivity). While these constraints are described briefly, we omit a detailed discussion of the such constraints due to lack of space. To remain focused on the intuition, we retain some seemingly non-linear operations (e.g., maximum, logical ORing, etc) in our formulation. Mapping from such operators to the exact ILP formulation may be trivially achieved by using widely-known ILP tricks [2].

Objective Function :

- (First priority) Minimizing the Maximum Channel Load (MCL) (Z).
- (Second priority) Minimizing Channel Occupancy (CO)(O) as defined.

$$\begin{aligned} Z &= \max_{(u,v)\in E} \sum_{i=1}^{m} \sum_{j=1}^{k} f_{i,j}(u,v) \\ O &= \sum_{(u,v)\in E} \sum_{i=1}^{m} \sum_{j=1}^{k} b_{i,j}(u,v) \end{aligned}$$

Constraints :

- For each flow, the sum of all outgoing flow-split variables from the source over all splits of each flow is equal to the load of this flow l_i. Also, the sum of all incoming flow-split variables to the destination over all splits of each flow is equal to the load of this flow l_i. (End-to-end flow conservation.)

$$\begin{aligned} \forall i \quad \sum_{(s_i,v)\in E} \sum_{j=1}^{k} f_{i,j}(s_i,v) &= l_i \\ \forall i \quad \sum_{(u,d_i)\in E} \sum_{j=1}^{k} f_{i,j}(u,d_i) &= l_i \end{aligned}$$

- For each flow-split, the sum of incoming flow variables to each router is equal to the sum of outgoing flow-split variables for the same router. (Hop-by-hop flow conservation.)

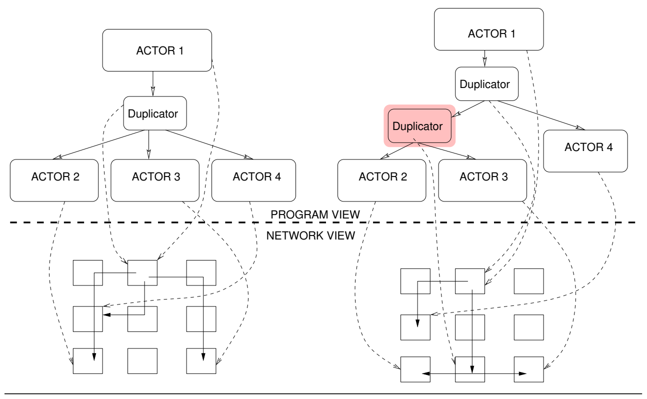


Figure 2. Program view and network view of fusion

$$\forall i,\ \forall j,\ \forall v \neq s_i,\ d_i$$
$$\textstyle\sum_{(u,v)\in E} f_{i,j}(u,v) = \sum_{(v,u)\in E} f_{i,j}(v,u)$$

- For each flow-split, any flow-split variable $f_{i,j}(u,v)$ is upper bounded by the flow load l_i and is enabled/disabled by its corresponding binary flow-split occupancy variable $b_{i,j}(u,v)$. Further, for each flow-split, there can be only one outgoing binary flow-split occupancy variable $b_{i,j}(u,v)$ from any router. (No branching flows to prevent recursive fission.)

$$\begin{array}{rcl} \forall i,\ \forall j,\ \forall (u,v) \in E \quad f_{i,j}(u,v) & \leq & l_i \cdot b_{i,j}(u,v) \\ \forall i,\ \forall j,\ \forall u \quad \displaystyle\sum_{(u,v)\in E} b_{i,j}(u,v) & \leq & 1 \end{array}$$

- For each flow-split, there should be no incoming link to any source, and no outgoing link from any destination. This constraint prevents a meaningless solution in which flow paths create self-loops at the source and destination which satisfies the other constraints but does not effectively route from source to destination.

$$\begin{array}{rcl} \forall i,\ \forall j,\ \forall (u,s_i) \in E \quad f_{i,j}(u,s_i) & = & 0 \\ \forall i,\ \forall j,\ \forall (d_i,v) \in E \quad f_{i,j}(d_i,v) & = & 0 \end{array}$$

3.2 Flow fusion

Figure 2 illustrates how flow-fusion may be viewed as a program transformation. A single multicasting splitter replicates flows to all the multicast recipients (as shown on the left). Fusion effectively interposes additional duplicating splitters creating a tree of splitters. The tree of splitters can then be mapped to network nodes to form a distribution tree (as shown in the bottom half of Figure 2).

We further augment the fusion transformation by integrating fission and fusion in TxComm. For multicast flows, TxComm's integrated fission+fusion allows multi-tree distribution wherein fusion creates individual trees and fission fisses trees into multiple trees. Consequently, even though we speak of fission and fusion as the two primitive transformations, in practical terms, the transformations are actually fission (for unicast traffic) and fission+fusion (for multicast traffic).

3.2.1 ILP formulation

The ILP formulation for fission+fusion has the following key changes (relative to the fission-only transformation).

First, in terms of basic definitions, we define multicast communication as a single flow with a single source (s_i) and a set of destination nodes (D_i) as opposed to a collection of flows, each with a single destination, as originally defined in Definition 1. This change reduces the number of flows in comparison to a unicast-only approach.

Second, distribution trees fundamentally require that flows be able to branch at routers. Further, because multicast data is replicated at such branching routers, flow-conservation does not hold. Instead, the flow increases at such branching routers by the branching factor. Our formulation supports branching by allowing up to three-way branching, which is specific to our two-dimensional topology[4].

Third, in the fission-only formulation, the same constraint that prevents branching of flow-splits also prevents non-simple paths. Because branching cannot be avoided in distribution trees, we develop an alternate mechanism to ensure simple paths. We use additional variables ($p_{i,j}(u,v)$) and constraints to ensure that a flow's distance-from-source monotonically increases with each hop. Cycle formation is prevented because cycles violate monotonicity.

Objective Function :

We use the same objective function as the fission ILP formulation.

Constraints :

[4] We limit branching to three ports because a packet may not branch backwards on the same port it arrived from without violating the *simple path* rule.)

- For each flow-split, the sum of incoming flow-split loads to any router (except the source and destination routers) is less than or equal to the sum of outgoing flow-split loads from this router. This constraint enforces a one-way bound on the load coming-in/going-out at a router to indicate that it can increase due to branching, but it cannot decrease.

$$\forall i,\ \forall j,\ \forall v \neq s_i,\ v \notin D_i$$
$$\sum_{(u,v)\in E} f_{i,j}(u,v) \quad \leq \quad \sum_{(v,u)\in E} f_{i,j}(v,u)$$

- For each flow-split, there can not be any outgoing binary flow-split variable $\sum_{(v,u)\in E} b_{i,j}(v,u)$ from any router (except the source and destination routers) unless there is an incoming binary flow-split variable $b_{i,j}(u,v)$ to this router , and if so, the number of the outgoing binary flow-split variable can not increase than 3. This constraint limits outgoing branching to 3-way branching (which is possible only when there is an incoming flow).

$$\forall i,\ \forall j,\ \forall v \neq s_i,\ v \notin D_i$$
$$\sum_{(v,u)\in E} b_{i,j}(v,u) \quad \leq \quad 3 \sum_{(u,v)\in E} b_{i,j}(u,v)$$

- For each flow, the sum of all outgoing flow-split loads from the source over all splits of each flow is equal to the load of this flow l_i. Also, the sum of all incoming flow-split loads to the destination over all splits of each flow is equal to the load of this flow l_i. (End-to-end flow conservation, generalized for multicast.)

$$\forall i \qquad \sum_{(s_i,v)\in E} \sum_{j=1}^{k} f_{i,j}(s_i,v) \quad = \quad l_i$$
$$\forall i,\ \forall t \qquad \sum_{(u,D_{i,t})\in E} \sum_{j=1}^{k} f_{i,j}(u,D_{i,t}) \quad = \quad l_i$$

- For each flow-split, when the outgoing binary flow-split occupancy variable $b_{i,j}(u,v)$ is enabled, the outgoing flow-split variable $f_{i,j}(u,v)$ from any router (except the source router) is equal to the only one incoming flow-split variables from this router that is carrying a flow $\max_{(w,u)\in E} f_{i,j}(w,u)$. Otherwise, if the the outgoing binary flow-split occupancy variable $b_{i,j}(u,v)$ is disabled, the outgoing flow-split $f_{i,j}(u,v)$ variable is equal to zero. (This constraint effectively ties the enable disable behavior with ho-by-hop flow control. At most one of the incoming flows is non-zero. So the *max* provides a way to say that the outgoing flow must match the solitary non-zero incoming flow.)

$$\forall i,\ \forall j,\ \forall (u,v)\in E,\ u \neq s_i$$
$$f_{i,j}(u,v) = \begin{cases} \max_{(w,u)\in E} f_{i,j}(w,u), & b_{i,j}(u,v) = 1 \\ 0, & b_{i,j}(u,v) = 0 \end{cases}$$

- For each flow-split, any flow-split variable $f_{i,j}(u,v)$ is upper bounded by the flow load l_i and enabled/disabled by its corresponding binary flow-split occupancy variable $b_{i,j}(u,v)$. For each flow-split, there can be only one incoming binary flow-split occupancy variable $b_{i,j}(u,v)$ to any router (except the source router).

$$\forall i,\ \forall j,\ \forall (u,v)\in E \quad f_{i,j}(u,v) \quad \leq \quad l_i \cdot b_{i,j}(u,v)$$
$$\forall i,\ \forall j,\ \forall v \neq s_i \quad \sum_{(u,v)\in E} b_{i,j}(u,v) \quad \leq \quad 1$$

- For each pair of destination routers in each flow-split, the sum of incoming flow-split variables of each destination should be equal.

$$\forall i,\ \forall j,\ \forall t_1,\ t_2,\ t_1 \neq t_2$$
$$\sum_{(u,D_{i,t_1})\in E} f_{i,j}(u,D_{i,t_1}) \quad = \quad \sum_{(u,D_{i,t_2})\in E} f_{i,j}(u,D_{i,t_2})$$

- For each flow-split, there should be no incoming link to any source.

$$\forall i,\ \forall j,\ \forall (u,s_i)\in E \quad f_{i,j}(u,s_i) \quad = \quad 0$$

- For each source router in each flow-split, each outgoing distance-from-source variable $p_{i,j}(s_i,v)$ from this router is equal to the binary flow-split occupancy variable $b_{i,j}(s_i,v)$ corresponding to it. For the other routers, each outgoing distance-from-source variable $p_{i,j}(s_i,v)$ from this router is one more than the maximum incoming distance-from-source variable to this router $\max_{(w,u)\in E} p_{i,j}(w,u)$ when the corresponding binary flow-split occupancy variable $b_{i,j}(u,v)$ is enabled, otherwise, it is equal to zero.

$$\forall i,\ \forall j,\ \forall (s_i,v)\in E \quad p_{i,j}(s_i,v) = b_{i,j}(s_i,v)$$
$$\forall i,\ \forall j,\ \forall (u,v)\in E,\ u \neq s_i$$
$$p_{i,j}(u,v) = \begin{cases} \max_{(w,u)\in E} p_{i,j}(w,u) + 1, & b_{i,j}(u,v) = 1 \\ 0, & b_{i,j}(u,v) = 0 \end{cases}$$

- For each designation router in each flow-split, any incoming distance-from-source variable $p_{i,j}(u, D_{i,t})$ to this router that is greater than zero is lower bounded by the minimum number of hops (the per source-destination pair constant MIN_HOPS) from the source to this destination.

$$\forall i,\ \forall j,\ \forall t \qquad \sum_{(u,D_{i,t})\in E} p_{i,j}(u, D_{i,t}) \geq \text{MIN\_HOPS}(s_i, D_{i,t})$$

3.3 Support for Free routing

Recall that the routing of flows was unrestricted in the above two formulations. Consequently, the routes obtained as solutions to the above formulations may be prone to deadlocks. Rather than limiting the physical channels that flows may route over, we attempt to break deadlock-cycles (if any) by using virtual channels (VCs). While the above approach is indeed analogous to the way VC-based deadlock-avoidance differs from turn-prevention based deadlock prevention, there is one key difference. Classical deadlock-avoidance uses different routing functions for adaptive and deadlock-free VCs. For example, while the adaptive channels may support unrestricted routing, the deadlock-free VCs may support dimension-ordered routing (or some other deadlock-free routing). In contrast, for our problem, we wish to use *only* the optimal routing as determined by our optimization problem. Deviating from the selected routes for any flow removes any expectation of improved performance because the reduction in channel load critically depends on using the optimal routes. Because of the above dilemma, we resort to a best-effort technique to eliminate deadlocks using the given number of VCs in the router hardware. Our technique uses static, compile-time, VC assignment to achieve two goals: the first goal is to achieve correctness (deadlock freedom) and the second goal is to maximize performance (minimize head-of-line blocking).

First, we attempt to assign VCs to each flow-split at each hop while eliminating any cyclic dependence among VCs with the goal of eliminating deadlock. While there are no guarantees that such a cycle-free VC assignment is possible, all our benchmarks were routable in a deadlock-free manner using 4VCs/PC. In the general case, if such cycle-free VC assignment is not possible with the given number of VCs, there may be no alternative to routing restrictions (i.e., using BSOR as a backup). The cycle prevention mechanism used in the ILP formulation is very similar to the cycle-prevention used previously for simple paths and, as such, details are omitted.

Second, we assign VCs to mitigate head-of-line blocking by preferring to bundle together flow-splits that take the same turns (or rather minimizing the number of turns assigned to a given VC). The definition of head-of-line blocking is when a packet has a free physical channel it wants to traverse but is hindered by another packet at the head-of-the-line that is waiting for another unrelated physical channel. By attempting to ensure that packets only wait behind other packets that are taking the same turn, we directly target head-of-line blocking. Effectively, such VC assignment mimics virtual output queuing (VOQ) [3] in the best case, but is again done on a best-effort basis.

Previous static VC assignment (in the context of BSOR) has focused solely on performance because deadlocks are not an issue for BSOR [16]. Further, they have attempted to improve performance by balancing two criteria: the need for isolating flows from each other and the need to balance VC-load. In contrast, our approach tries to minimize HOL-blocking by using VOQ-like VC assignment. Even though, packets of multiple flows are not isolated (i.e., they may wait behind one another) in our technique, there is no penalty as long as all the packets are headed to the same output port because they will be serialized over the physical links, anyway.

Finally, we note that VC assignment is distinct from VC allocation. VC assignment statically chooses which VCs a flow may use at a given hop. A packet belonging to a flow may only request the VCs assigned to it. VC allocation, on the other hand, is the actual process of allocating such requested VCs and is done in hardware at run-time. This distinction is necessary because multiple packets of multiple flows may be assigned to the same VC, but at any given time, a VC may be allocated to only one packet. Effectively, VC allocation manages the time-multiplexing of multiple packets that are assigned to the same VC.

At a high level, our ILP formulation assigns VCs at each hop to avoid cycles which is enforced via constraints. To maximize performance, we set the objective function to (a) minimize the number of turns at each VC and (b) achieve balanced utilization over all VCs. Both the above objectives use simple counting techniques to count the number of different turns at each VC.

Our ILP formulation uses the following definitions. We refer to the number of available VCs as N_VCS. We refer to the set of possible turns after traversing edge (u, v) as $T(u, v)$. The union of injection ports at each router (I) and the set of channels E is IE. Finally, the set of flow-splits that traverse the edge $(u, v) \in E$ and then take a turn $t \in T(u, v)$ is called the contributory flow-split set and represented as $CFR(u, v, t)$. CFR can be easily constructed from the known routes obtained from the solution of the TxComm transformations. We use a one-hot encoded binary vector of flow-VC occupancy free variables $FV_{i,j,u,v}$ of length equal to N_VCS to indicate the VC assigned to the j^{th} split of the i^{th} flow going through the edge (u, v). The one-hot encoding indicates the VC occupied by the flow-split.

Objective Function :

- (First priority) Minimize a) the maximum number of unique turns per VC at each edge (the *max* summation), b) and the difference between the maximum and the minimum number of turns (the difference between the *max* summation and the *min* summations). Minimizing the difference ensures load balance over all VCs. The $Turn$ array holds the details of each VC and the turns assigned to that VC. The $Turn$ element for a single <channel, VC,turn> tuple constructed by logical ORing of the VC-occupancy variables of all the contributory flows (i.e., flows that contribute to the turns).

$$R_1 = 2\textstyle\sum_{(u,v)\in IE} \max_{vc} \sum_{t\in T(u,v)} Turn_{u,v,t,vc}$$
$$- \textstyle\sum_{(u,v)\in IE} \min_{vc} \sum_{t\in T(u,v)} Turn_{u,v,t,vc}$$
$$\forall (u, v) \in IE,\ \forall t \in T(u, v)\ \forall vc$$
$$Turn_{u,v,t,vc} = \text{OR}_{(i,j)\in CFR(u,v,t)} FV_{i,j,u,v,vc}$$

- (Second priority) Maximize the number of occupied VCs with turns in order to maximize VC utilization. It covers corner cases not handled in the first objective.

$$R_2 = -\sum_{(u,v)\in IE} \sum_{t\in T(u,v)} \sum_{vc=1}^{N\_VCS} Turn_{u,v,t,vc}$$

Constraints :

- This constraint specifies that each flow can occupy one VC at each edge, thus enforcing the one-hot encoding.

$$\forall i,\ \forall j\ \forall (u, v) \in IE \qquad \sum_{vc=1}^{N\_VCS} FV_{i,j,u,v,vc} = 1$$

- Cycle prevention constraints similar to those used for fission+fusion using the p variable. Details omitted.

3.4 Tolerating link failures via TxComm

In this section, we consider the application of application specific routing to tolerate permanent faults in links. We do not focus on the detection and identification of faulty-links since that can be realized by using traditional scan-chain based testing. Further, recall that application-specific routing is performed at compile time. If applications can be recompiled (including rerouting) to run on a system with faulty links, we consider the system to be fault-tolerant.

At a high level, our routing strategy must still pursue the same goals as in fault-free TxComm: the minimization of maximum channel load (MCL) while routing the flows from the specified sources to the specified destinations, potentially using fusion and fission to further reduce MCL. Indeed the only additional requirement is to *entirely avoid* all communication over faulty links. The extension of the TxComm ILP formulation to handle the additional constraints is correspondingly minor. To ensure that no stream flows pass through the faulty links, we constrain $b_{i,j}(u, v)$ to be zero for all i and j when the link (u, v) is faulty.

Note that the above solution cannot be trivially grafted to BSOR because BSOR relies on the underlying Turn model to provide deadlock freedom. A single faulty link could violate deadlock freedom in BSOR. In contrast, TxComm attempts best-effort deadlock-freedom using free-routing. Because there is some non-zero probability that free routing may not yield a deadlock free route, we suggest the use of an underlying fault-tolerant routing algorithm such as $up*/down*$ routing [13]. While such an underlying fault-tolerant routing technique is needed for the cases where free-routing fails to discover deadlock-free routes, in our experiments free-routing always succeeded. As such, our results do not include such an underlying fault-tolerant network.

3.5 ILP Solver results

We analyzed TxComm's benefits for 13 StreamIT benchmarks, each for three different network sizes. We used a commercial ILP solver (CPLEX v12.0.1) to solve our optimization problems. On an Intel Core i5 (3.2GHz) processor-based workstation with 4GB memory, the solver run-times were manageable even for the slowest cases (see Table 1). Because the transformations are compile time, incurring such one-time overheads (which are modest, barring outliers) may be acceptable for performance benefits that recur on each run.

Table 2 summarizes the results from our ILP solver and compares the maximum channel load achieved by TxComm with that of BSOR. The maximum channel load is specified in arbitrary relative units. Hence the absolute numbers are not meaningful; only the relative ratios matter. The channel loads highlighted in **bold** indicate improvement (decrease) in channel load. Specifically, the numbers in the fission column are shown in bold only if they achieve lower MCL than BSOR-CDG. Similarly, the numbers in the TxComm column (which includes both fission for unicast and fission+fusion for multicast) are shown in bold only if they achieve lower MCL than by using fission-only. Finally, Fission-1 MCL is the configuration that uses fission with only one split. Effectively, it captures the benefits of removing BSOR's routing restrictions (and instead using VC-based deadlock avoidance) even though no actual flow fission takes place.

The following observations may be made from the results. First, we observe that all benchmarks can benefit from TxComm. Fission-alone benefits 36 of 39 configurations. The multicast-optimization (applicable only to the configurations with multicast communication as indicated by the (M) annotation in Table 2) provides some benefit for the three configurations not covered by fission. Further, it benefits a total of 10 benchmarks. Note that there exist 11 configurations which have multicast, but which do not benefit from fission+fusion (non-bold numbers in the TxComm column of Table 2). Second, we observe that the removal of routing restrictions improves (reduces) the maximum channel load for four configurations, even in the absence of fission and fusion.

3.6 Hardware support for TxComm

TxComm requires similar router hardware as used in prior application specific router proposals [8], which includes the programmable routing tables. In addition, TxComm also requires multicast support similar to that in [7] (with one key difference as explained below). Because the basic router we use is similar to prior proposals [7, 8], and because our claim of novelty is on the compiler technique (not the hardware), we briefly describe the differences from prior art.

Router configuration for fission Fission can fail to improve throughput if the injection/ejection ports – ports through which network traffic enters/leaves the network – serializes all traffic, thus nullifying the advantages of fission. Effectively, the network becomes capable of supporting more throughput than the injection/ejection ports can support.

As such, we use multiple injection/ejection ports. The increased injection/ejection ports were beneficial to BSOR as well. Prior academic research proposals [1] as well as prior industry products have used multiple injection/ejection ports [11].

Support for fusion The tree-based distribution of multicast traffic (employed by fusion) requires multi-transmission support for the case where a flit must remain in the input buffers till copies of it are dispatched to all of the branching ports. Further, true throughput benefits accrue only when a flit on an input port is replicated in a single cycle across multiple output ports of the router's switch. This is a key difference between the multicast router proposed in VCTM [7] where flits are transmitted across multiple output ports in a serial fashion and our router. In a router with no pipeline bubbles between successive flits or packets, the overall throughput does not improve whether a single flit resides in the buffer for three cycles or three different flit occupy the buffer over three cycles. Note that the VCTM's approach may be perfectly valid in their context where injection port queueing delays affect latencies. But for our context, where throughput matters, such serialized transmission offers no throughput benefit.

To that end, we redesign the VC allocation to allow at simultaneous multi-transmission. One major challenge in allowing such multi-transmission is that VC allocation must be achieved in a starvation-free, deadlock-free way. An analogy can be made with the Dining Philosopher's problem when multiple packets (philosophers) attempt to get exclusive access to multiple VCs (silverware). To handle deadlocks, we use one of the simplest solutions possible – exclusive access for multicast packets that are attempting to get multiple VCs. Note that the exclusive access is only for multicast packets; unicast packets continue to operate as before. We allow at most one multicast packet per router (via token capture mechanism) to forward multiple re-

	Fission-only		TxComm	
	Routing	VC Assign.	Routing	VC Assign
50th %ile	3.8s	0.3s	2m 15s	0.6s
90th %ile	1m 30s	0.8s	2m 15s	11.2s
100th %ile	5m 42s	1.5s	171m	13m 20s

Table 1. Solver runtimes (m=minute, s=second)

quests. A packet requesting multiple VCs either gets all its grants or obtains a "lock" on all its grants. Before a VC is freed, the router checks if a lock is held. If so, the VC ownership is transferred to the lock holder. In the absence of such a locking mechanism, it is possible for a multicast packet to be starved by other unicast packets. For switch arbitration, we assume an opportunistic model where packets may request multiple ports when selected at the local arbitration stage. Because crossbar switch paths naturally allow any input to be fed to any output (assuming drivers are sized accordingly), such a design would allow a flit to be replicated on multiple output ports in a single cycle. However, if the grants of the various output ports are staggered, then flit transmission is also staggered across multiple cycles.

Each of the above mechanisms are fairly simple to implement. Intra-router token capture for exclusive access, and tracking a single lock-owner per VC, are both trivial. More importantly, they do not violate the basic allocator operation which operates via purely localized arbitration (e.g., round-robin) at the input and output ports. Because of such localized operation, simultaneous grants of multiple output ports are not guaranteed. However, the locking mechanism guarantees forward progress even under such non-simultaneous grant.

4. Experimental Methodology

Streaming Applications and Network Load Generation We use the StreamIT benchmarks and their compiler. We use the compiler generated actors and layout on three different network sizes (4×4, 6×6, and 8×8). The combination of 13 benchmarks and 3 network sizes gives us a total of 39 configurations. Note that compiling a benchmark for a different network size yields a different actor-stream graph and hence must be treated as a separate communication pattern (i.e., not the scaling up of a given pattern). Further, as stated in Section 2, we assume hardware-pipelined communication. Our hardware pipeline models interlocks to ensure that actors cannot fire when either their operands are unavailable to be "popped" from upstream streams *or* when they are unable to "push" data on downstream streams because buffers of downstream routers are full. We use the actor latencies reported by the compiler but scale the values to examine the peak sustainable application throughput. Such scaling corresponds to faster processors or voltage/frequency-scaled networks as mentioned in Section 2. Note, the application-level throughput metric that may be unique to each application. (e.g., block encryption/decryption rate, video frame processing rate, audio processing rate, radio signal processing rate) and are not comparable across applications. As such, we use normalization to report throughput improvement over BSOR, the prior best application-specific routing technique.

Router Configurations We compare three router configurations. All three configurations use 4VCs/PC. The first configuration uses BSOR routing, which is our main competitor and is used with static VC assignment similar to TxComm's. Unlike TxComm, there is no possibility of deadlocks in BSOR since BSOR achieves deadlock freedom by enforcing routing restrictions (because routes only take turns that are on an acyclic CDG [8]). For completeness, we also compared BSOR-4VC-static to BSOR-4VC-dynamic in which VC allocation is done dynamically at each router. We found that this resulted in poorer performance than BSOR-4VC static. Hence we omit BSOR-4VC-dynamic from the comparison. The second configuration uses fission only (without fusion, but with free-routing), even for applications with multicast communication. This configuration is included to isolate the effects of the two transformations. Finally, we include the full-implementation of TxComm which includes both fission (for unicast) and fission+fusion (for multicast) with free-routing. With 4VCs, we were able to route *all* 39 benchmark configurations in a deadlock-free manner.

Table 2. Benchmarks and their characteristics
(MCL: Maximum channel load, **bold** indicates improvement)

Code	Benchmark	Config.	BSOR-CDG MCL	Fission MCL	TxComm MCL	Fission-1 MCL
AC	Auto Correlation	4×4 (M)	64	64	**10.67**	64
		6×6 (M)	129	**64**	**16**	**64**
		8×8 (M)	64	64	**10.67**	64
BF	Beam Former	4×4 (U)	24	**12.8**	—	24
		6×6 (U)	24	**6.22**	—	24
		8×8 (M)	24	24	**8**	24
CC	Comparison Counter	4×4 (M)	64	**52**	**16**	64
		6×6 (M)	130	**68**	**9.6**	**80**
		8×8 (M)	80	**68**	**8.38**	80
CV	Channel Vocoder	4×4 (M)	801	**267**	267	801
		6×6 (M)	250	**200**	**59.33**	**200**
		8×8 (M)	801	**267**	267	801
DCT	Discrete Cosine Transform	4×4 (U)	256	**128**	—	256
		6×6 (U)	256	**85.33**	—	256
		8×8 (U)	256	**85.33**	—	256
DES	DES Encryption	4×4 (M)	128	**64**	**44.8**	**64**
		6×6 (M)	128	**64**	64	128
		8×8 (M)	128	**64**	64	128
FB	Filterbank	4×4 (U)	128	**64**	—	128
		6×6 (M)	128	**42.66**	42.67	128
		8×8 (M)	128	**42.66**	42.67	128
FFT	FFT	4×4 (U)	512	**354.46**	—	512
		6×6 (U)	512	**279.27**	—	512
		8×8 (U)	512	**256**	—	512
FMR	FM Radio	4×4 (M)	12	**4**	4	12
		6×6 (M)	12	**3**	3	12
		8×8 (M)	12	**3.43**	3.43	12
MPG	MPEG 2	4×4 (M)	834	**397**	**311.77**	834
		6×6 (M)	834	**300**	300	834
		8×8 (M)	834	**238.29**	238.29	834
SRP	Serpent Encryption	4×4 (U)	256	**170.66**	—	256
		6×6 (U)	256	**170.66**	—	256
		8×8 (U)	256	**153.6**	—	256
TDE	Time Delay Equalization	4×4 (U)	1920	**1280**	—	1920
		6×6 (U)	1920	**1080**	—	1920
		8×8 (U)	1920	**840**	—	1920
VOC	Vocoder	4×4 (U)	40	**18.33**	—	40
		6×6 (U)	40	**17.14**	—	40
		8×8 (U)	40	**12.5**	—	40

Simulator Our simulator models a three-stage packet-switched router. The first stage performs two functions: look-ahead routing for the next hop and VC/switch allocation. Recall, VC assignment is predetermined statically under free-routing. However, because there may be multiple flows with the same static VC assignment, the allocation occurs in hardware at run-time. VC/switch allocation is speculatively overlapped [12] (i.e., a switch grant without a VC grant cannot be used). The next two stages are for switch traversal and link traversal respectively. The router architecture models the two hardware changes necessary (multi-transmission support for fusion, and increased injection/ejection ports for fission). Recall that BSOR benefits from the increase in injection/ejection ports as well. All simulations ran till they achieved steady state throughput. This resulted in simulation times ranging from 100,000 cycles to 10,000,000 cycles.

5. Results

5.1 Overall performance improvement

Figure 3 illustrates the performance benefits of TxComm for the three network sizes (4×4 in Figure 3(a), 6×6 in Figure 3(b), 8×8 in Figure 3(c)). The three subgraphs of Figure 3 plot the normalized (peak sustainable) performance (Y-axis) for the fission-only and full TxComm routing methods (bars within a group) for all our benchmarks (groups of bars). In addition, we include a three sets of bars to show the geometric mean improvement in performance over (A) benchmarks which have only unicast communication (i.e., fusion does not help), (B) benchmarks which have multicast communication, and (C) all benchmarks. Because fusion does not benefit configurations without multicast traffic, we do not include the TxComm bar for such applications; TxComm performance is equivalent to fission-only performance for such benchmarks. We reuse the unicast ("U") and multicast ("M") annotations from Table 2 to mark the unicast and multicast benchmark configurations.

The overall mean speedup for the 4×4, 6×6, and 8×8configurations were 1.4X, 1.83X, and 1.44X, respectively. The geometric mean performance improvement across all configurations and all benchmarks is 1.55. Further the results also show that without fusion, the mean performance degrades to 1.33X (across all benchmarks and all network sizes).

There are some interesting cases where TxComm underperforms BSOR. For example, DES in the 8×8 configuration sees a performance degradation from the full TxComm even though its performance improves with the fission-only transformation. Interestingly, DES also improves with the fusion-only transformation (not shown). However, the deadlock-free VC assignment for the combined TxComm configuration results in degraded performance. For such cases, a simple trial-and-error approach can be used to apply subsets of transformations to improve overall performance. However, we do not consider such subsetting in our results.

5.2 Isolating the Contributions of the Techniques

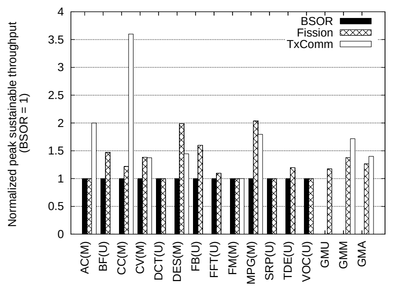
TxComm includes the effects of both fission and fusion, as well as the performance boost from free routing. To isolate the effects of each of these and to understand how they combine/compose, Figure 4 shows the speedup of six of the eight possible combinations of the three techniques over BSOR. Each of fission, fusion and free routing is shown as a circular region. Two-way and three-way intersecting regions correspond to the combination of the corresponding techniques. The central three-way intersection corresponds to TxComm.

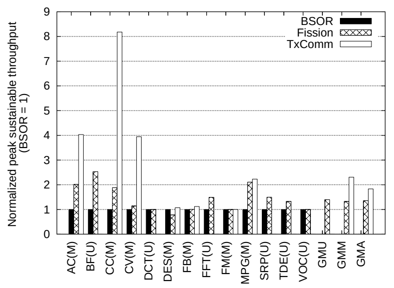
We make three observations from Figure 4. First, we observe that the three techniques improve performance as they are combined. Second, we observe that without free routing, fission's improvement degrades from 1.33X to 1.04X. This observation supports our the arguments from Section 1 on why routing restrictions can severely limit the performance benefits of fission. Third, we also observe that free-routing was not necessarily a good idea for BSOR (i.e., without fission and fusion) because it provides a meager 6% performance boost.

We omitted the two variants of fusion-only and TxComm with deadlock-free routing restrictions because (1) there is strong evidence on the importance of free-routing from the fission experiment, and (2) we had to prioritize gathering more relevant results.

5.3 Impact of Degree of Fission

Recall that our transformation optimizes the MCL assuming a fixed degree of fission. In this section, we vary the degree of freedom and consider its impact on performance. Figure 5 illustrates the variation in MCL (Y-axis, normalized to BSOR) with degree of fission (X-axis) for the fission-only configuration with free


(a) 4×4 network


(b) 6×6 network

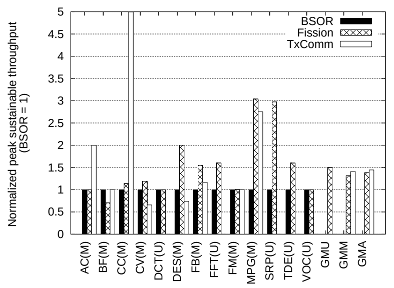

(c) 8×8 network

Figure 3. Performance Improvement

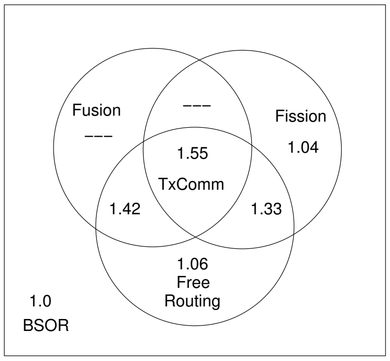


Figure 4. Isolating the impact of individual techniques (Perfomance improvement over BSOR, geometric mean across all network sizes and all benchmarks)

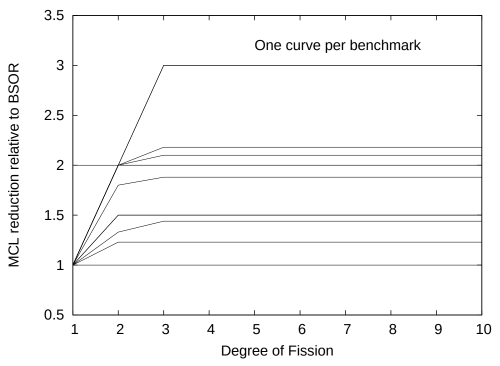


Figure 5. Impact of degree of fission on performance (4x4 network, with free routing)

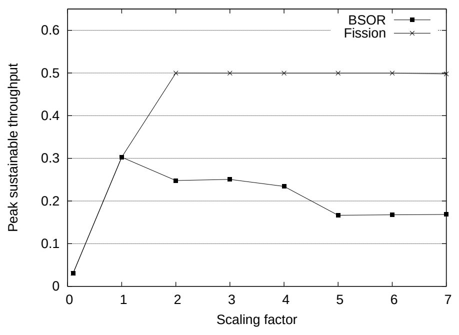
routing for each benchmark (individual curves) for the 4×4 network size. We omit identifying labels omitted because they add clutter without adding value to the point we wish to make. Note, because fission is ineffective without free-routing, we assume free routing in Figure 5. Figure 5 confirms the intuition that most of the benefits of fusion occur at modest fission degrees (≤ 4), across all benchmarks. Fissing flows to a higher degree did not result in any further gains. The lack of benefits beyond 4-way fission is true for 6×6 and 8×8 network sizes as well (not shown).

Note, one curve is better than BSOR even at a fission degree of one. That improvement is because of free-routing in the case of the DES application as shown in Table 2.

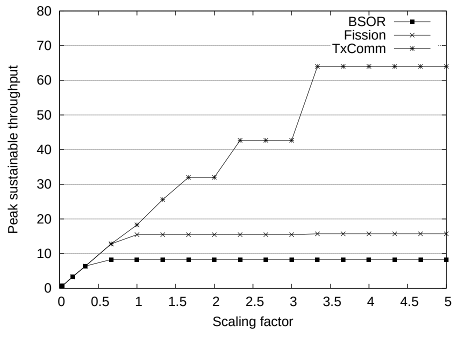
5.4 Computation communication interactions

The previous section focused on the peak sustainable throughput for the network. In this section, we examine the interaction between computation throughput and communication throughput. In a streaming workload, either communication or computation may be the rate-limiting entity. If the computation is slow and is the rate limiting factor, TxComm cannot provide any benefits by improving network communication. To illustrate this issue, Figure 6 plots the achieved throughput (Y-axis) for various values of the computation scaling factor (X-axis) for benchmarks SRP 8×8 (Figure 6(a)) and CC 6×6 (Figure 6(b)). Because SRP does not have multicast, Figure 6(a) includes two curves – one for BSOR and another for fission-only. In contrast, CC, which has multicast communication, has three curves, including the full TxComm with fission+fusion.

There is one common trend in both benchmarks. At low values of computation scaling factor, the computation is the bottleneck and the use of TxComm does not result in any increase in sustainable throughput. On the other hand, as the computation scaling factor increases, the network becomes the bottleneck and TxComm's performance advantage begins to manifest itself. Note, the benchmarks SRP and CC were selected to illustrate the benefits of fission and fission+fusion respectively. As expected, the fission curve corresponding to SRP achieves higher sustainable throughput than



(a) Benchmark SRP, network size 8×8



(b) Benchmark CC, Network size 6×6

Figure 6. Interaction between computation and communication

BSOR. Similarly, in the case of benchmark CC, we observe two distinct jumps in performance. When applying fission, alone there is one significant jump. But the combination of both fission and fusion delivers the full benefits of TxComm.

5.5 Fault-tolerance

Figure 7 plots the normalized bottleneck bandwidth (reciprocal of channel load normalized to fault-free TxComm; higher is better) averaged across all thirteen benchmarks while varying the number of injected faults (X-axis) for each of the three network sizes. The number of injected faults varies from 0% to 10% (with upward rounding) of links in the network. Because the number of links in 4×4, 6×6, and 8×8 networks are 48, 120, and 224 respectively, we inject up to 5, 12, and 23 faults respectively (as shown in the range of the X-axis). Further, we evaluate five independent runs for any given number of faults. Because fault injection is random, the final resulting channel load (and bandwidth) may vary from run to run. To indicate such variation, we plot three curves showing the maximum (best case), minimum (worst case) and mean (average case). Note, the fault-injection is for TxComm only. BSOR is kept fault-free because BSOR may not guarantee deadlock-freedom when certain links are faulty.

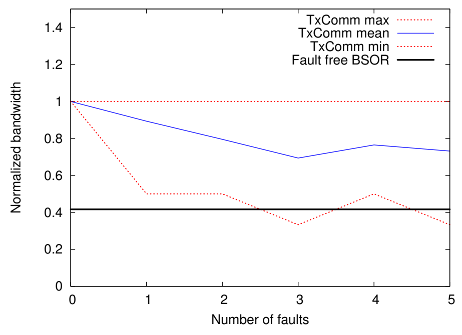
We make the following key observations.

1. Even with 10% of faulty links, the average performance of TxComm is significantly better than that of fault-free BSOR. This highlights the fault-tolerance capability provided by free routing.
2. In the worst case, TxComm is comparable to fault-free BSOR at high fault-rates (approaching 10%) but better at lower fault-rates (less than 5%).
3. Finally, the best case outcomes show that it is possible to get some faults that do not affect bottleneck bandwidth at all. This results in best case performance matching fault-free performance

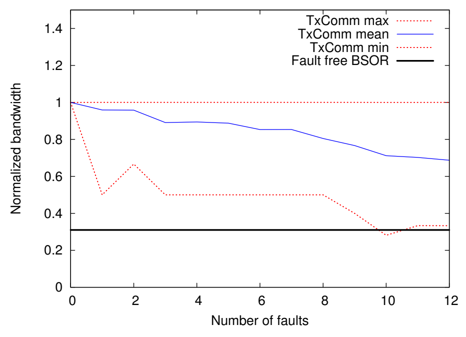
6. Related Work

The closest related work to our work is BSOR/BSORM [8] as already discussed in several previous sections. Seo *et al.* study a run-time route discovery technique to minimize channel loads for application-specific communication [15]. Their routing attempts to discover disjoint paths and does not exploit either fission or fusion. Application-specific communication can be exploited at synthesis time, to produce low-complexity, low-energy NoCs in embedded systems and SoCs [6]. Our focus is on hardware that can serve as a platform for various applications, each of which have very specific communication patterns. Virtual circuit tree multicasting (VCTM) explores tree-based distribution, (which our fusion transformation also exploits), but in the context of multicast coherence traffic [7]. Because of the very significant differences between the nature of coherence traffic and application-specific streaming traffic, the tradeoff between the time needed to setup multicast communication and the sophistication of the routing is reversed. For coherence traffic, setting up the multicast trees quickly is much more important than discovering the optimal distribution trees that reduce the maximum channel load in a globally co-ordinated manner. In contrast, for our domain, we can afford to spend time in the route planning stage to ensure that the routing of all communication is globally co-ordinated to minimize maximum channel load. Note, VCTM does not employ any fission.

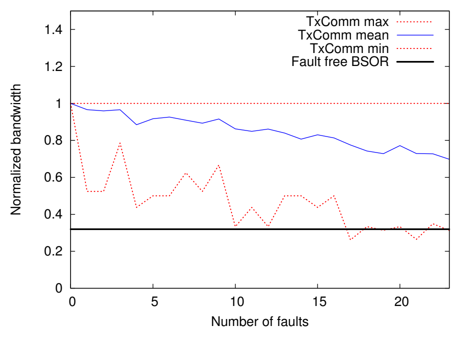
StreamIT's compiler does *actor* fusion and fission for computational efficiency and load balance [18] unlike TxComm which targets communication efficiency



(a) Network size 4×4



(b) Network size 6×6



(c) Network size 8×8

Figure 7. Fault-tolerance: Impact of injected faults

and load balance. At a high level, there is an analogy between what StreamIT compiler does for computation and what TxComm does for communication in that both attempt to improve efficiency and load balance. In fact, our use of similar terms is based precisely on that similarity. However, the insights involved in computation fission/fusion are completely different from communication fission/fusion.

7. Conclusions

Recent work has recognized that sophisticated compile-time analysis may be used to optimize communication for application-specific communication [8]. This paper aims to expand the capability of such compile-time communication optimization. Our TxComm approach has three components. First, we use fission to split streams where such streams are the bandwidth bottlenecks, achieving better load balance on the network links. Second, we use fusion to fuse streams to effectively get the benefits (reduced link-occupancy and contention) of tree-based multicast, which may be further fissed. We are the first to transform the communication of an application to optimize communication performance. Finally, we employ *free routing* – avoiding the use of routing restrictions in conjunction with static VC assignment to avoid deadlocks. Free routing boosts the performance benefits of the two transformations. While such a cycle-free VC assignment is not guaranteed to exist, we found that they do exist for all our benchmarks. Because benchmarks are not adversarially written, this optimization is worth attempting if the expectation is that it will succeed (as seen in our benchmarks). However, to handle the general case, one may fall back on techniques that route over acyclic CDGs if cycles are not avoidable in the VC assignment stage. Another key advantage of free-routing is that it provides fault-tolerant application-specific communication. Without free-routing, a single link fault may eliminate guarantees of deadlock freedom, as in BSOR. With free-routing, deadlock-free routing may still be achieved in the presence of faulty links at the cost of some performance, as in TxComm.

All the above transformations and optimizations can be expressed as integer linear program (ILP) optimization problems. In practice, solvers can converge on solutions in seconds/minutes (common case) and hours (uncommon case). The analytical results generated by the solvers show significant opportunity in all the benchmark/size combinations we evaluated. Similar analytical results with random fault injection shows that TxComm achieves comparable (or better) performance as fault-free BSOR, even when as many as 10% of links are faulty. Evaluation by simulation with 13 StreamIT benchmarks over three different system sizes reveals that, in the fault-free case, TxComm achieves a mean throughput improvement of 55% over BSOR.

References

[1] S. Balakrishnan and D. K. Panda. Impact of multiple consumption channels on wormhole routed k-ary n-cube networks. In *Int'l Parallel Processing Symposium (IPPS '93)*, pages 163–167, 1993.

[2] Johannes Bisschop. Aimms, optimization modeling. paragon decision technology, 2006.

[3] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003.

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

[5] Michael I. Gordon, William Thies, Michal Karczmarek, Jasper Lin, Ali S. Meli, Andrew A. Lamb, Chris Leger, Jeremy Wong, Henry Hoffmann, David Maze, and Saman Amarasinghe. A stream compiler for communication-exposed architectures. In *ASPLOS-X: Proc. of the 10th international conference on Architectural support for programming languages and operating systems*, pages 291–303, 2002.

[6] Wai Hong Ho and Timothy Mark Pinkston. A design methodology for efficient application-specific onchip interconnects. *IEEE Trans. Parallel Distrib. Syst.*, 17(2):174–190, 2006.

[7] Natalie Enright Jerger, Li-Shiuan Peh, and Mikko Lipasti. Virtual circuit tree multicasting: A case for onchip hardware multicast support. In *ISCA '08: Proceedings of the 35th Annual Intl. Symposium on Computer Architecture*, pages 229–240, 2008.

[8] Michel A. Kinsy, Myong Hyon Cho, Tina Wen, Edward Suh, Marten van Dijk, and Srinivas Devadas. Application-aware deadlock-free oblivious routing. In *ISCA '09: Proc. of the 36th annual international symposium on Computer architecture*, pages 208–219, 2009.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[10] Manjunath Kudlur and Scott Mahlke. Orchestrating the execution of stream programs on multicore platforms. *SIGPLAN Not.*, 43(6):114–124, 2008.

[11] Shubhendu S. Mukherjee, Peter Bannon, Steven Lang, Aaron Spink, and David Webb. The alpha 21364 network architecture. In *HOTI '01: Proc. of the The Ninth Symposium on High Performance Interconnects (HOTI '01)*, page 113, 2001.

[12] Li-Shiuan Peh and William J. Dally. A delay model and speculative architecture for pipelined routers. In *HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, page 255, 2001.

[13] Thomas L. Rodeheffer and Michael D. Schroeder. Automatic reconfiguration in autonet. In *Proceedings of the thirteenth ACM symposium on Operating systems principles*, SOSP '91, pages 183–197, 1991.

[14] Faizal A. Samman, Thomas Hollstein, and Manfred Glesner. Multicast parallel pipeline router architecture for network-on-chip. In *DATE '08: Proc. of the conference on Design, automation and test in Europe*, pages 1396–1401, 2008.

[15] Daeho Seo and Mithuna Thottethodi. Disjoint-path routing: Efficient communication for streaming applications. In *IPDPS '09: Proc. of the 2009 IEEE Intl. Symposium on Parallel&Distributed Processing*, pages 1–12, 2009.

[16] Keun Sup Shim et al. Static virtual channel allocation in oblivious routing. In *NOCS '09: Proc. of the 2009 3rd ACM/IEEE Intl. Symposium on Networks-on-Chip*, pages 38–43, 2009.

[17] Michael Bedford Taylor et al. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2):25–35, 2002.

[18] William Thies, Michal Karczmarek, and Saman Amarasinghe. Streamit: A language for streaming applications. In *Intl. Conference on Compiler Construction*, April 2002.