TransCom: Transforming Stream Communication for Load Balance and Efficiency in Networks-on-Chip

Ahmed H. Abdel-Gawad* and Mithuna Thottethodi
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
{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). 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 (TransCom) to achieve higher throughput via better network load balance and more efficient network utilization 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 these transformations for stream communication.) Fission splits communication streams in to multiple streams that may be routed over independent network paths to achieve better network load balance. 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 is solved at compile time. Evaluations with a suite of StreamIT benchmarks show that TransCom achieves significant performance improvement (nearly 60% on average) over prior application-specific (non-transformed) routing techniques.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; D.3.4 [**Programming Languages**]: Processors—*Compilers*

General Terms

Design

*On leave from Cairo University, Egypt.

1. INTRODUCTION

Recent work has examined using application-specific knowledge of communication patterns to optimize network routing [9, 15] (for throughput/performance) and/or design (for simpler hardware) [7]. 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.

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 [8] 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 [8] 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 [9]. 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 [9]) to ensure acyclic channel dependence graphs (CDG) or using 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 [5]. 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 *integer linear programming* (ILP) problem 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 using TransCom without free-routing, thus trading off performance for correctness. Note, TransCom, even without the performance boost of free-routing is significantly better than BSOR. 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.

The combination of all three methods yielded significant reductions in channel load (as found by the solver) for all of the 39 configurations. Significant performance improvements (60%, on average) were also measured experimentally by simulation.

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

- 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 cast both the above optimizations as integer linear programming (ILP) optimization problems, which enables the use of commercial solvers to automatically apply our techniques. For the most part, the ILP problems can be solved in seconds/minutes with only two applications seeing any benefits beyond 10-minutes of solver effort.

[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.

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 TransCom 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 8 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 [6] 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 [10].

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.* [9]. 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. TransCom 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 TransCom offers superior performance for network-bound applications. (Because of the dual-

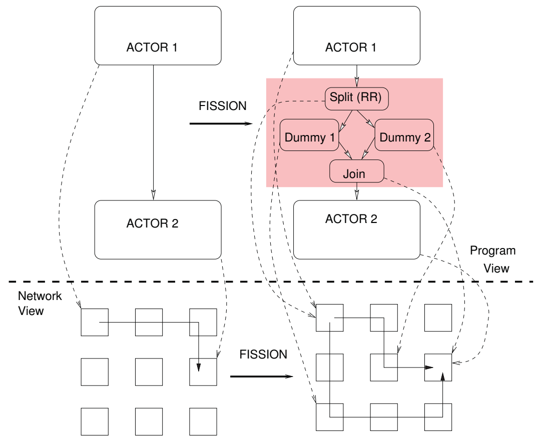


Figure 1: Program view and network view of fission

ity principle, our results imply that TransCom 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 [11, 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 (MCL) – 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.

3. TRANSCOM

This section describes the two transformations – fission and fusion – in TransCom. 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 transforms the program to have multiple flow-splits per flow. As shown in Figure 1, fission may 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.

Note, because joiners emit items in the correct (pre-fission) stream order (a property inherited from basic StreamIT), there are no correctness issues with fission. 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 MCL.

We simplify the first choice by picking an "adequately large" degree of fission based on two observations (a precise definition follows). First, the MCL 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 (i.e., worse) 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, in Section 5.3, we show that the degree of fission indeed exhibits this property and shows no improvement beyond 4-way fission.

The above simplification which fixes the degree of fission enables us to cast the determination of the remaining two choices as an ILP formulation. Effectively, the two remaining challenges (i.e., determining the fraction of traffic in each split and the path taken by each split) are resolved by the ILP solver in a manner that minimizes maximum channel load. To determine the fraction of traffic in each split (the second detail), we associate a variable for the the load on each split of the flow (henceforth referred to as *flow-split*) rather than with the entire flow (as done in BSOR). To determine the path taken by each split, our ILP formulation uses constraints on the flow split variables such that all *legal* paths for each split satisfy the constraints, and uses the objective function to minimize maximum channel load. A *legal* path as one that follows a simple path (i.e., no branching, no cycles) from source to destination.

Our definition of a flow is similar to that used in BSOR, which is the entire stream of unicast communication.

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 amount $q_{i,j}$, $i = 1, \ldots, m$ and $j = 1, \ldots, k$, represents the load carried by j^{th} split of the i^{th} flow. Further, we also define per-channel flow-split variables $f_{i,j}(u, v)$, $i = 1, \ldots, m$ and $j = 1, \ldots, k$, which represents the load carried by j^{th} split of the i^{th} flow on the edge $(u, v) \in E$. The per-channel, 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. Consequently, $f_{i,j}(u, v) = b_{i,j}(u, v) \times q_{i,j}$.*

The constraints and objectives of our ILP formulation are shown in Table 1. The constraints and objectives are numbered for convenient discussion. The objective consists of two sub-objectives[2]. Primarily we wish to minimize the maximum channel load, which is Z as shown in **(O1)**. Secondarily, we wish to minimize the number of occupied channels, which is O as shown in **(O2)**. 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.

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

Table 1: ILP formulation for fission

	Objective Function (Minimization)
O1	$Z = \max_{(u,v)\in E} \sum_{i=1}^{m} \sum_{j=1}^{k} f_{i,j}(u,v)$
O2	$O = \sum_{(u,v)\in E} \sum_{i=1}^{m} \sum_{j=1}^{k} b_{i,j}(u,v)$
	Constraints
C1	$\forall i,\ \forall j,\quad \sum_{(s_i,v)\in E} f_{i,j}(s_i,v) = q_{i,j}$ $\forall i,\ \forall j,\quad \sum_{(u,d_i)\in E} f_{i,j}(u,d_i) = q_{i,j}$ $\forall i \quad \sum_{j=1}^{k} q_{i,j} = l_i$
C2	$\forall i,\ \forall j,\ \forall v \neq s_i,\ d_i$ $\sum_{(u,v)\in E} f_{i,j}(u,v) = \sum_{(v,u)\in E} f_{i,j}(v,u)$
C3	$\forall i,\ \forall j,\ \forall (u,v)\in E \quad f_{i,j}(u,v) \leq l_i \cdot b_{i,j}(u,v)$ $\forall i,\ \forall j,\ \forall u \quad \sum_{(u,v)\in E} b_{i,j}(u,v) \leq 1$
C4	$\forall i,\ \forall j,\ \forall (u,s_i)\in E \quad f_{i,j}(u,s_i) = 0$ $\forall i,\ \forall j,\ \forall (d_i,v)\in E \quad f_{i,j}(d_i,v) = 0$

The constraints at a high level must avoid illegal paths. To that end, we 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) $q_{i,j}$ is equal to the original (unfissed) channel flow, which is l_i **(C1)**. This constraint, in conjunction with the second constraint of **(C3)** also ensures that at most one link from/to the source has the same load as $q_{i,j}$ with other $f_{i,j}(u,v)$ values being zero. Second, we ensure that hop-by-hop flow conservation is enforced at intermediate routers, wherein the sum of incoming flows is equal to the sum of outgoing flows **(C2)**. 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.

The formal ILP problem includes additional constraints that are important for correctness in the following corner cases: (1) avoiding negative flow values first constraint of **(C3)** by ensuring that no flow-split exceeds the unfissed flow, (2) avoiding branching flows and ensuring simple paths (for each flow-split, there can be only one outgoing binary flow-split occupancy variable $b_{i,j}(u,v)$ from any router as defined by the second constraint of **(C3)**), and (3) ensuring source-to-destination connectivity by avoiding meaningless solutions in which the source and destination have self-loops without a contiguous path between them **(C4)**. To remain focused on the intuition, we retain some seemingly non-linear operations (e.g., maximum, and logical ORing) in our formulation. Mapping from such operators to the exact ILP formulation may be trivially achieved by using widely-known ILP tricks [2].

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 TransCom. For multicast flows, TransCom'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).

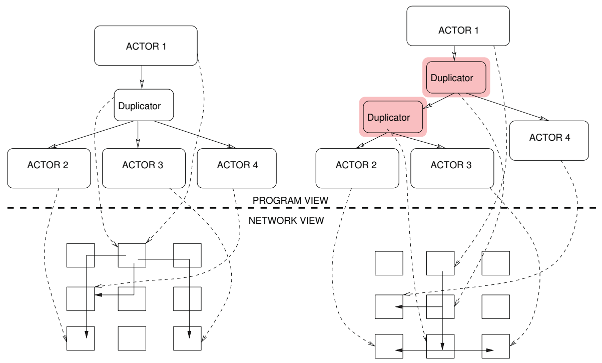


Figure 2: Program view and network view of fusion

Table 2: ILP formulation for fusion+fission

	Constraints
C1	$\forall i,\ \forall j,\ \forall v \neq s_i,\ v \notin D_i$ $\sum_{(u,v)\in E} f_{i,j}(u,v) \leq \sum_{(v,u)\in E} f_{i,j}(v,u)$
C2	$\forall i,\ \forall j,\ \forall (u,v)\in E,\ u \neq s_i,\ u \notin D_i$ $f_{i,j}(u,v) \leq \sum_{(w,u)\in E} f_{i,j}(w,u)$ $b_{i,j}(u,v) \leq \sum_{(w,u)\in E} b_{i,j}(w,u)$
C3	$\forall i,\ \forall j,\ \forall (u,v)\in E \quad f_{i,j}(u,v) \leq l_i \cdot b_{i,j}(u,v)$ $\forall i,\ \forall j,\ \forall v \neq s_i \quad \sum_{(u,v)\in E} b_{i,j}(u,v) \leq 1$
C4	$\forall i,\ \forall j,\ \forall (u,s_i)\in E \quad f_{i,j}(u,s_i) = 0$
C5	$\forall i \quad \sum_{(s_i,v)\in E} \sum_{j=1}^{k} f_{i,j}(s_i,v) \geq l_i$ $\forall i \quad \sum_{j=1}^{k} q_{i,j} = l_i$
C6	$\forall i,\ \forall j,\ \forall v \in D_i \qquad \sum_{(u,v)\in E} f_{i,j}(u,v) = q_{ij}$
C7	$\forall i,\ \forall j,\ \forall (u,v)\in E,\ u \neq s_i$ $f_{i,j}(u,v) = \begin{cases} q_{i,j}, & b_{i,j}(u,v) = 1 \\ 0, & b_{i,j}(u,v) = 0 \end{cases}$
C8	$\forall i,\ \forall j,\ \forall (s_i,v)\in E \quad p_{i,j}(s_i,v) = b_{i,j}(s_i,v)$ $\forall i,\ \forall j,\ \forall (u,v)\in E,\ u \neq s_i$ $p_{i,j}(u,v) = \begin{cases} \sum_{(w,u)\in E} p_{i,j}(w,u) + 1, & b_{i,j}(u,v) = 1 \\ 0, & b_{i,j}(u,v) = 0 \end{cases}$
C9	$\forall i,\ \forall j,\ \forall t$ $\sum_{(u,D_{i,t})\in E} p_{i,j}(u,D_{i,t}) \geq \text{MIN\_HOPS}(s_i, D_{i,t})$

3.2.1 *ILP formulation*

The ILP formulation for fission+fusion (Table 2) uses identical (and hence omitted from the table) minimization objective functions as the fission ILP. However, there the following key changes (relative to the fission-only transformation) in the constraints.

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. All the other definitions carry over with similar change in meaning. For example, $q_{i,j}$ refers to the flow on the j^{th} split of the i^{th} multicast tree, and so on. As a consequence of this change in definition, the constraints have to be changed to allow legal *trees* as opposed to legal paths.

Distribution trees alter the constraints in several ways. First, they fundamentally require that flows be able to branch at routers. This has several implications. (1) Because multicast data is replicated at such branching routers, flow-conservation does not hold. Instead, the flow can potentially increase at such branching routers, which is expressed in constraint **(C1)** by constraining the outgoing flows at router v to exceed the incoming flows at the same node. (2) To ensure legal trees in which paths can only emerge from a prior path, we add a constraint that for each flow-split, there can not be any outgoing flow-split variable ($f_{i,j(u,v)}$ or $b_{i,j}(u,v)$) from any router (except the source and destination routers) unless there is an incoming binary flow-split (occupancy) variable to this router. (3) We similarly relax the no-branching rule that there can be at most one outgoing $b_{i,j}(u,v)$ enabled at a given router u. Instead, we constrain that at most one incoming $b_{i,j}(u,v)$ is enabled at any router v **(C3)**, consistent with a tree structure. (4) While the source may not have any incoming traffic, a node in the destination set of a distribution tree may have outgoing traffic if it is a branching node of the multicast tree. Thus, we retain the constraint eliminating incoming traffic at the source while removing the constraint on outgoing traffic at the destination **(C4)**. (5) Constraints **(C5)** and **(C6)** are variants of the **(C1)** constraint from fission (from Table 1) modified to handle branching. Constraint **(C5)** ensures that the total outgoing load at the source equals or exceeds the original stream load (because of branching at the source, the outgoing load may exceed the original load). Similarly, **(C6)** ensures that each destination node in the multicast set receives the full stream load. (6) Constraint **(C7)** explictly imposes the relationship between the per-split traffic ($q_{i,j}$), and the per-channel-per-split variables ($b_{i,j}(u,v)$ and $f_{i,j}(u,v)$) that we had defined earlier in Definition 1. Though the equivalent relationship was also enforced in fission, the absence of branching and flow conservation made it possible to implicitly guarantee the relationship without explicit constraints. (7) Finally, 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 for each spur of the tree. We use additional per-split, per-channel variables ($p_{i,j}(u,v)$) and constraints to ensure that the distance of a channel along the path from source to destination monotonically increases with each hop **(C8)**. Cycle formation is prevented because there is no feasible way to assign a monotonically increasing set of values to a cycle. (For example, $a > b, b > c, c > a$ is an infeasible set of constraints.)

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).

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 TransCom 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.

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. 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).

The details of the ILP formulation are shown in Table 3. 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. Our basic constraint **(C1)** ensures that any given per-channel flow is assigned to a single VC. We use the same cycle-avoidance formulation that was used earlier in the fusion+fission formulation (with the per-channel, per flow-split p variable). 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. Any VC assignment that satisfies the constraints is a deadlock-free assignment. 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 acyclic CDGs as in BSOR).

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.

Table 3: ILP formulation for VC assignment

	Objective Function
O1	$R_1 = 2\sum_{(u,v)\in IE} \max_{vc} \sum_{t\in T(u,v)} Turn_{u,v,t,vc}$ $- \sum_{(u,v)\in IE} \min_{vc} \sum_{t\in T(u,v)} Turn_{u,v,t,vc}$ $\forall (u,v) \in IE,\ \forall t \in T(u,v)\ \forall vc$ $Turn_{u,v,t,vc} = \text{OR}_{(i,j)\in CFR(u,v,t)} FV_{i,j,u,v,vc}$
O2	$R_2 = -\sum_{(u,v)\in IE} \sum_{t\in T(u,v)} \sum_{vc=1}^{N\_VCS} Turn_{u,v,t,vc}$
	Constraints
C1	$\forall i,\ \forall j\ \forall (u,v) \in IE \quad \sum_{vc=1}^{N\_VCS} FV_{i,j,u,v,vc} = 1$

To be specific, we include two minimization objectives. The primary objective corresponds to the VOQ-like goal which is to minimize the maximum number of unique turns per VC at each edge(*max* summation in **(O1)**) as well as the difference between the maximum and minimum number of turns which balances the load across VCs (the difference between the *max* summation and the *min* summations). We also wish to maximize the number of occupied VCs with turns in order to maximize VC utilization **(O2)**. This objective is negated because we wish to cast all objectives as minimization objectives.

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.

3.4 ILP Solver results

We analyze TransCom's benefits for 13 StreamIT benchmarks, each for three different network sizes. We use 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. Because ILP is known to be NP-hard, and because unbounded compile time is not an option, we limit the ILP solver to 10 minutes. (Extending the ILP solver effort to 6 hours did not result in any improvement of the MCL except for AC8x8 and CC8x8. For these two benchmarks, limiting the time of the solution to 10 minutes resulted in an MCL that is larger than the minimum MCL with extended time by around 6% for each of them, but the simulated throughput was within 1% of each other.) We argue that 10 minutes compile time overhead is an acceptable tradeoff given that they are incurred once as an overhead while the performance benefits recur on each run. For the development phase, this optimization can be avoided till the release phase as with the case of all 'heavy-weight' compiler optimizations.

Table 4 summarizes the results from our ILP solver and compares the MCL achieved by TransCom with that of BSOR. The MCL 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 TransCom 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 (i.e., no fission). Effectively, it captures the benefits of removing BSOR's routing restrictions (and instead using VC-based deadlock avoidance).

The following observations may be made from the results. First, we observe that all benchmarks can benefit from TransCom. 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 4) 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 TransCom column of Table 4). Second, we observe that the removal of routing restrictions improves (reduces) the MCL for four configurations, even in the absence of fission and fusion.

3.5 Hardware support for TransCom

TransCom requires similar router hardware as used in prior application specific router proposals [9], which includes the programmable routing tables. In addition, TransCom also requires multicast support similar to that in [8] (with one key difference as explained below). Because the basic router we use is similar to prior proposals [9, 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 [12].

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 [8] (in which 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 (say) 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 queuing 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

Table 4: Benchmarks and their characteristics
(MCL: Maximum channel load, **bold** indicates improvement)

Code	Benchmark	Config.	MCL			
			BSOR CDG	Fission	Trans Com	Fission1
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**	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

at most one multicast packet per router (which can be enforced via any traditional fair arbiter such as round-robin priority arbiters [3]) to forward multiple requests. 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 round-robin arbitration 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 TransCom's. Unlike TransCom, 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 [9]). 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 TransCom 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.

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 [13] (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 four injection/ejection ports to match our maximum degree of fission (as we show later in Section 5.3). 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 TransCom 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 TransCom 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 TransCom bar for such applications; TransCom performance is equivalent to fission-only performance for such benchmarks. We reuse the unicast ("U") and multicast ("M") annotations from Table 4 to mark the unicast and multicast benchmark configurations.

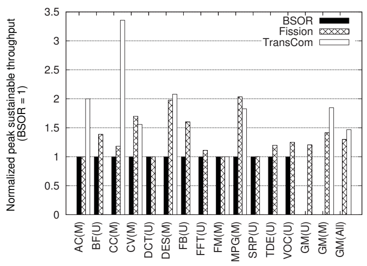
The overall mean speedup for the 4×4, 6×6, and 8×8configurations were 46.8%, 79.3%, and 55.3%, respectively. The geometric mean performance improvement across all configurations and all benchmarks is 59.9%. Further the results also show that without fusion, the mean performance degrades to 30.8% (across all benchmarks and all network sizes).

There are some interesting cases where TransCom underperforms BSOR. For example, DES in the 8×8 configuration sees a performance degradation from the full TransCom 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 TransCom 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.

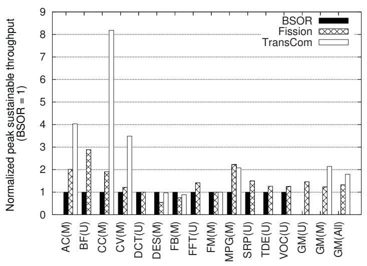
5.2 Isolating the Contributions of the Techniques

TransCom 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 all 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 TransCom.

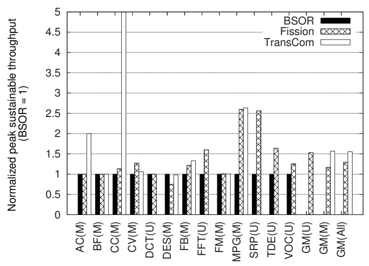
We make three observations from Figure 4. First, we observe that the three techniques improve performance as they are combined. Full TransCom achieves nearly 60% average speedup which is significantly higher than the speedup of any subset of techniques. As shown in Figure 4, taking away even one component from TransCom halves the average performance improvement. Second, we observe that without free routing, fission's improvement degrades from 30.8% to 5.2%. Free-routing boosts the performance of fission+fusion (which is effectively TransCom without free-routing) from 25.6% faster than BSOR to 59.9% faster than BSOR. This observation supports our arguments from Section 1 on why routing restrictions can severely limit the performance benefits of fission. More importantly, this result shows that TransCom



(a) 4 × 4 network



(b) 6 × 6 network



(c) 8 × 8 network

Figure 3: Application Performance Improvement

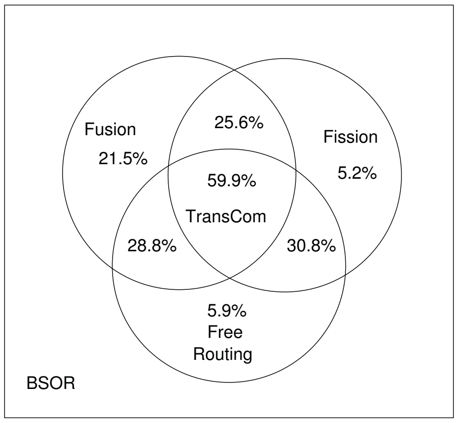


Figure 4: Isolating the impact of individual techniques (Performance improvement over BSOR, geometric mean across all network sizes and all benchmarks)

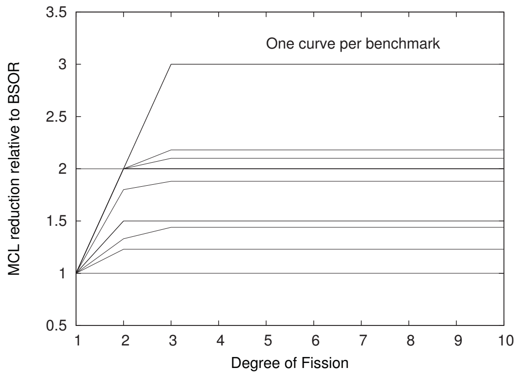


Figure 5: Impact of degree of fission on performance (4x4 network, with free routing)

has value even in the absence of free routing since it can achieve 25.6% better throughput than BSOR. 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.

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 routing for each benchmark (individual curves) for the 4×4 network size. We omit identifying labels because they add clutter without adding value to the point we wish to make. 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).

One may think that the limit on the degree of fission is an artifact of our topology (2D-mesh) or the number of local ports. While it is true that such hardware limits will correspondingly limit the benefit of fissing at a given node, it does not, in general, imply that the hardware limits translate to an equivalent limit on degree of fission. For example, one may imagine a case with eight-way fission where two fissed streams each occupy the same outgoing link at a source node. However, they may branch out to different paths beyond the first hop. In such cases, if the bottleneck link is not an outgoing link of the source node, it is possible to reduce MCL. We leave an analysis of the impact of network configuration/topology on MCL for future work.

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 4.

5.4 Overheads

At a high level, there are two costs that TransCom imposes. First, there is the cost of the additional software splits/joins at the program level. However, these are extremely lightweight software sections that effectively do one memory copy from the one stream to another; and are unlikely to contribute significantly to the overheads. Because such costs are expected to be minimal, we focus on the second aspect that affects cost; the potentially longer (non-minimal) paths that packets may traverse in TransCom which may incur additional energy costs.

5.4.1 Impact on energy/power

On the energy front, the overall effect of TransCom can be broken into two components which correspond to fusion and fission. On the one hand, fusion reduces energy compared to BSOR because its multicast communication eliminates the redundant flit transmissons caused by multiple unicast communicaton streams. On the other hand, fission may cause an increase in energy consumption relative to BSOR because it may route fissed streams over longer paths, efffectively increasing the average hops-per-flit for a given stream. Our analysis of TransCom's routes shows the following key results.

- Fission alone increases the average hops per flit by 44% on average (averaged across all 3 network configurations and all 13 benchmark) compared to BSOR.
- Fusion-alone decreases hops-per-flit by 9% one average compared to BSOR.
- TransCom, which combines the effect of both techniques, results in a 39% increase in average hops per flit.

To quantify the impact of increased link activity on dynamic energy, we use the LUNA energy/power model [4]. LUNA is a high-level link-utilization based power model that enables high-speed energy modeling with a slight dilution in accuracy compared to detailed energy models such as Orion [19]. LUNA has been validated to be within 6% of Orion [4]. While LUNA can be used to estimate *absolute* dynamic power, LUNA's approach enables us to model *relative* network power (i.e., TransCom's network power relative to BSOR's network power) directly in terms of link utilization in a technology independent way. LUNA enables such simplification because the energy associated with one flit hop (which is the aggregate energy of a buffer-read, a crossbar traversal, a link traversal and a buffer write) is the same for BSOR and TransCom routers as long as they use the same technology and router configuration. Although TransCom and BSOR routers differ in two ways, neither difference affects energy analysis. First, TransCom's arbiters are different (as described in Section 3.5). However, arbiter energy is known to be a small fraction of router energy [19, 4]. Second, TransCom's handling of multicast complicates energy computation because a single input-port buffer read can cause two or three crossbar/link traversals and buffer-writes. In such cases, we conservatively overstate TransCom's energy to include as many buffer-reads as crossbar/link traversals. As a result, energy becomes directly proportional to link activity (flit hops).

Note, if we increased performance without *any* energy overheads, it would effectively result in an increase in power. Thus, even if TransCom had zero energy overhead, the 60% increase in performance would result in 60% higher power as a baseline. In practice, the non-zero energy overheads further increase our power overheads by 26% beyond the baseline. In other words, the power consumption of TransCom is 2.01X ($= 1.26 \times 1.6$) the power consumption of BSOR. Comparing TransCom and BSOR using the *(throughput)*^{3}*/(power)* (T^3/P) measure, which is invariant under dynamic voltage and frequency scaling (DVFS), as a metric of power-efficiency, we observe that TransCom achieves superior T^3/P by a factor of 2 compared to BSOR (because $(1.6)^3/2.01 \approx 2$). Further, even the above overhead is limited to network energy/power only, which is a fraction of overall system energy/power; thus implyng that the overall increase in system energy/power is likely to be further reduced.

5.4.2 Impact on latency

The key performance metric for stream processing is throughput (analogous to instruction throughput or IPC). The latency of processing a single item in a stream (analogous to the latency of single instruction) is not important.

However, for completeness, we include a brief analysis of the impact of TransCom on stream-processing latency. TransCom affects latency in two different ways. On the one hand, latency increases because of longer stream routes compared to BSOR (recall the increase in average hops per flit). On the other hand, because of increased communication concurrency (due to fission) and decreased contention (due to both fission and fusion), the latency decreases. We tracked the latency of packets both on an unloaded network and at the steady peak sustainable load. In the former case, the latency of TransCom improved by 23% which indicates that the advantages of communication concurrency overshadows the disadvantage of longer paths. Even in the latter case, where TransCom operates with a higher network load than BSOR, TransCom achieves 10% lower latency than BSOR due to its higher concurrency and throughput.

6. RELATED WORK

The closest related work to our work is BSOR/BSORM [9] 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 [7]. 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 [8]. 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 TransCom which targets communication efficiency and load balance. At a high level, there is an analogy between what StreamIT compiler does for computation and what TransCom 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. DISCUSSION AND FUTURE WORK

In this section, we briefly discuss applying TransCom's underlying principles to a broader domain by relaxing one of the constraints we assumed in earlier sections. Specifically, we assumed that the placement of actors was compiler-driven, and thus, static for the duration of the run. If run-time systems can change placement, the use of our techniques will be governed by how frequently such changes occur. If changes are rare (relative to recompilation time which is seconds/minutes) then compile-time Transcom may still be applicable. If the changes are frequent enough to rule out compile-time routing, then all compile-time routing techniques (including BSOR) will fail. Effectively, frequent unpredictable placement changes convert application-specific communication to resemble dynamic traffic. However, the basic fission and fusion techniques may be generalized to be implemented at run-time by using heuristics rather than ILP solvers. For example, the fission and fusion may be used to extend run-time application-specific routing techniques such as tentacle routing [15]. We leave such extensions of TransCom for future work.

8. CONCLUSION

Recent work has recognized that sophisticated compile-time analysis may be used to optimize communication for application-specific communication [9]. This paper aims to expand the capability of such compile-time communication optimization. Our TransCom 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 (including TransCom without free-routing) if cycles are not avoidable in the VC assignment stage. While this diminishes the performance benefits of TransCom, there are still significant performance benefits compared to BSOR.

All the above transformations and optimizations can be expressed as integer linear programming (ILP) optimization problems. In practice, solvers can converge on solutions in seconds/minutes (typical case) capturing almost all of the benefit within 10 minutes (bounded worst case) of solver effort. The analytical results generated by the solvers show significant opportunity in all the benchmarks/size combinations we evaluated. Evaluation by simulation with 13 StreamIT benchmarks over three different system sizes reveals that TransCom achieves a mean throughput improvement of 60%.

9. ACKNOWLEDGMENTS

The authors thank the anonymous reviewers and the shepherd for their comments and suggestions. This work was supported in part by the National Science Foundation (CCF-0644183, CCF-0541385 and CCF-0702612). Any opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of the National Science Foundation.

10. 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] J. Bisschop. Aimms, optimization modeling. paragon decision technology, 2006.

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

[4] N. Eisley and L.-S. Peh. High-level power analysis for on-chip networks. In *Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*, CASES '04, pages 104–115, 2004.

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

[6] M. I. Gordon, W. Thies, M. Karczmarek, J. Lin, A. S. Meli, A. A. Lamb, C. Leger, J. Wong, H. Hoffmann, D. Maze, and S. 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.

[7] W. H. Ho and T. M. Pinkston. A design methodology for efficient application-specific on-chip interconnects. *IEEE Trans. Parallel Distrib. Syst.*, 17(2):174–190, 2006.

[8] N. E. Jerger, L.-S. Peh, and M. Lipasti. Virtual circuit tree multicasting: A case for on-chip hardware multicast support. In *ISCA '08: Proceedings of the 35th Annual Intl. Symposium on Computer Architecture*, pages 229–240, 2008.

[9] M. A. Kinsy, M. H. Cho, T. Wen, E. Suh, M. van Dijk, and S. Devadas. Application-aware deadlock-free oblivious routing. In *ISCA '09: Proc. of the 36th annual international symposium on Computer architecture*, pages 208–219, 2009.

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

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

[12] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The alpha 21364 network architecture. In *HOTI '01: Proc. of the The Ninth Symposium on High Performance Interconnects (HOTI '01)*, page 113, 2001.

[13] L.-S. Peh and W. 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.

[14] F. A. Samman, T. Hollstein, and M. 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] D. Seo and M. 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] K. S. 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] M. B. Taylor et al. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2):25–35, 2002.

[18] W. Thies, M. Karczmarek, and S. Amarasinghe. Streamit: A language for streaming applications. In *Intl. Conference on Compiler Construction*, Apr. 2002.

[19] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, MICRO 35, pages 294–305, 2002.