

Packet Switched vs. Time Multiplexed FPGA Overlay Networks

Nachiket Kapre, Nikil Mehta, Michael deLorimier, Raphael Rubin,
Henry Barnor, Michael J. Wilson, Michael Wrighton, and André DeHon

Dept. of CS, MC 256-80
California Institute of Technology
Pasadena, CA 91125

contact: <nachiket@caltech.edu>

Abstract—Dedicated, spatially configured FPGA interconnect is efficient for applications that require high throughput connections between processing elements (PEs) but with a limited degree of PE interconnectivity (e.g. wiring up gates and datapaths). Applications which virtualize PEs may require a large number of distinct PE-to-PE connections (e.g. using one PE to simulate 100s of operators, each requiring input data from thousands of other operators), but with each connection having low throughput compared with the PE's operating cycle time. In these highly interconnected conditions, dedicating spatial interconnect resources for all possible connections is costly and inefficient. Alternatively, we can time share physical network resources by virtualizing interconnect links, either by statically scheduling the sharing of resources prior to runtime or by dynamically negotiating resources at runtime. We explore the tradeoffs (e.g. area, route latency, route quality) between time-multiplexed and packet-switched networks overlayed on top of commodity FPGAs. We demonstrate modular and scalable networks which operate on a Xilinx XC2V6000-4 at 166MHz. For our applications, time-multiplexed, offline scheduling offers up to a 63% performance increase over online, packet-switched scheduling for equivalent topologies. When applying designs to equivalent area, packet-switching is up to $2\times$ faster for small area designs while time-multiplexing is up to $5\times$ faster for larger area designs. When limited to the capacity of a XC2V6000, if all communication is known, time-multiplexed routing outperforms packet-switching; however when the active set of links drops below 40% of the potential links, packet-switched routing can outperform time-multiplexing.

I. INTRODUCTION

When implementing a computation as a simple circuit at the gate or RTL level, we expect each gate or register output to be conveyed to its successor gate or register input on every cycle. Since every link in the computational graph is active on every cycle, dedicating a physical interconnection link between operators as a configured FPGA wire is efficient.

When a computation is described at a higher level, it is often possible to identify cases where operators do not need to communicate on every link for every cycle [1]. For example, a compressor operator will output compressed data at a lower data rate than it consumes input data. These cases provide us with the opportunity to time share and virtualize interconnect links (e.g. another operator can use the compressor's output link on idle cycles).

At even higher levels of abstraction, we may virtualize compute elements as well [1], [2]. Physical area constraints

may dictate that a 1:1 mapping of operators to physical PEs is infeasible. For N operators mapped onto a single PE, a single operator may only produce data and communicate it to another operator once every N cycles, where N can be very large.

For these sparse communication patterns on large computational graphs, dedicating spatial interconnect links on a configured FPGA network for all possible communications between operators is inefficient. Wires and switches dominate FPGA resources [3], and dedicating spatial interconnect guarantees that these resources will be severely underutilized. In order to handle such problems efficiently, we explore overlaying virtual networks on top of the physical configured FPGA network, allowing us to reuse physical links in time. We consider two types of networks:

1) *Time-Multiplexed*: In this scheme, routes for data packets sent between communicating operators are computed offline prior to runtime. By taking advantage of global route information, time-multiplexed networks promise minimal total communication time between operators and maximal network resource utilization. However, offline computation can be compute intensive and must allocate resources for all possible communications among operators. The communication schedule must be stored in memory to program the behavior of each element in the network on every cycle at a potentially significant area cost.

2) *Packet-Switched*: Opposingly, in this scheme routes for data packets sent between operators are negotiated dynamically at runtime. This promises no additional memory for storing schedules and no offline setup. Routes are only made for operators that are actually communicating. However, switches for these networks are typically more complex (larger, slower) and, since routing decisions are only made locally, routes can be less efficient.

There may be cases where each network is preferred. We use communication workloads of varying density from ConceptNet [2], [4] to help quantify the tradeoffs and understand the situations in which one network switching scheme may be superior to the other. The novel contributions of this work include:

1. Demonstration of efficient and scalable static and dynamic FPGA overlay networks operating at 166MHz.
2. Quantification of the performance difference between of-

fine scheduling and online routing (Section VIII-A).

3. Quantification of performance impacts associated with balancing interconnect and computing resources (Section VIII-B.1).
4. Characterization of area and performance tradeoffs between time-multiplexed and packet-switched networks (Section VIII-B.2).
5. Quantification of the performance difference between time-multiplexing and packet-switching under varying application communication loads (Section VIII-C).

II. BACKGROUND

A. Networks on Chip

Computing systems have long been designed to time share interconnect [5]–[10]. Increasing silicon capacities have made it possible to implement such shared networks on a single chip. In the early days of system-on-a-chip integration, these networks were primarily on-chip buses. However, serial buses do not scale as chip sizes grow. To achieve higher performance, it has become necessary to look beyond buses and investigate scalable, high-performance, low-overhead on-chip networks [11], [12]. FPGAs today seem to be undergoing a similar transition. Commodity FPGAs now support several forms of on-chip buses which could be replaced with low overhead on-chip networks as they scale further.

To demonstrate why forms of interconnect other than buses should be examined, consider how network communication scales with the number of PEs. Figure 1 shows network I/O messages per cycle as a function of PEs on our time-multiplexed network with no bandwidth limitations, given a ConceptNet [2], [4] communication load. Assuming no segmentation, a bus can only handle a single network send and receive per cycle. In Figure 1 we see that as the the number of PEs increases, our our application could benefit from sending and receiving significantly more than one message per cycle. At small numbers of PEs, the bus would only require $1\text{--}2\times$ more cycles to route all messages than an unlimited network, but at large PE counts (> 500) the bus would require over 100 times as many cycles. With fewer PEs a bus may be adequate because the network is lightly loaded, with most cycles dedicated to serialized processing at the PEs. As we increase the number of PEs, each PE communicates more, and we need a network capable of processing multiple messages simultaneously to support the increased load.

B. Communication Patterns

An important choice to make in designing programmable interconnect is the switching style. Table I rounds up the relative strengths and weaknesses of four commonly used communication patterns.

Selection of an appropriate communication pattern should be based on application communication requirements and characteristics. Configured switching and time-multiplexing use static scheduling, requiring that communication be predictable and known ahead of time. Configured switching is inefficient for applications that underutilize physical link throughput.

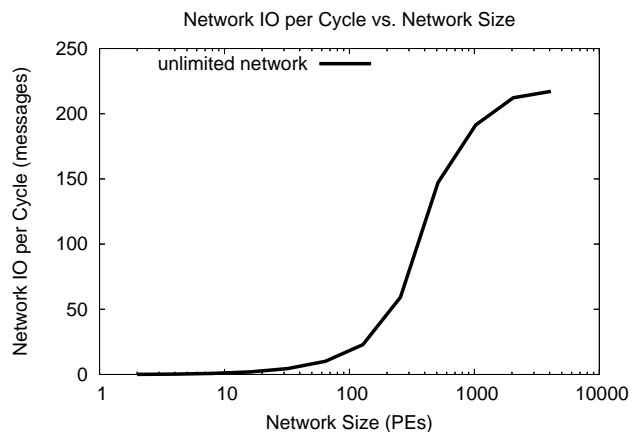


Fig. 1. ConceptNet Network I/O per Cycle vs. Network Size

Circuit-switching is efficient for larger messages on shorter networks, and is not an appropriate solution for the kinds of small message applications on which we focus here.

Hardware requirements are particularly important in comparing packet-switching and time-multiplexing. Packet-switching typically requires larger switchboxes due to buffering and logic required for dynamic decision making. Time multiplexed switchboxes are typically smaller in terms of raw logic, but if the total number of communication cycles is large, switch context memory may require significant area. To illustrate these area and performance tradeoffs, consider the following example. Packet-switched switchboxes may allow us to fit an 8 PE network on a chip. With comparatively smaller switches we could fit a 16 PE time-multiplexed network as long as routing could be completed in 30K cycles. If routing takes more than 30K cycles we can fit only 8 PEs, and at more than 100K cycles we can only fit 4 PEs. Consequently, there are operating ranges where either network may outperform the other.

To select wisely between packet-switching and time-multiplexing, we need to develop analytic area and time models to generalize this example and ground the general trends of Table I into quantitative, empirical tradeoffs.

- For equivalent topologies and communication loads, how much better can offline, global routing solutions be compared to online, local routing solutions?
- How much smaller, faster can a time-multiplexed interconnect scheme support a given level of static traffic?
- For what activity factors is a packet-switched interconnect scheme superior?

C. Packet-Switched Networks

Packet-switched networks have been researched extensively by the networking community since the days of the early telephone switching systems [13]. Consequently, several flavors of packet-switched networks exist today, *e.g.* wormhole, cut-through, store-forward. Even the NoC (Network on Chip) community (Section II-A) has focused on building on-chip packet-switched networks. Surveys in on-chip packet-switched

TABLE I
ROLE OF VARIOUS COMMUNICATION PATTERNS

Characteristics	Configured	TM	PS	Circuit
Know communication needs	early (compile time)		late (runtime)	
Communication predictability	high		low	
Communication throughput compared to physical link throughput	>	<	<	<
PE-to-PE latency compared to packet length	n/a	n/a	>	<
Channel Utilization	high		low	
Switch Logic Area	low	low	high	high
Switch Memory Area	low	high	modest	low
Relative Latency	lowest	low	highest	moderate

communication networks can be found in [10], [14], [15]. These NoC efforts have mostly targeted ASICs and only recently have FPGAs become large enough to merit such networks.

Nonetheless a handful of FPGA-based overlay NoCs have appeared in recent years. We summarize a representative sample in Table II. We improve and expand upon these prior efforts in several ways:

- Most of the designs reported in Table II are two dimensional bidirectional mesh topologies. These research efforts typically dismiss other topologies for asymptotic scaling concerns. Only Dimetalk allows arbitrary topologies in addition to meshes. We built and analyzed several topologies, including meshes, but only present data for Butterfly Fat Trees (BFTs) as we observe that they yield the best results. Quantifying the tradeoffs between different topologies is beyond the scope of this paper.
- Several efforts use synthetic data for understanding the tradeoffs in building their NoCs (*e.g.*, [16], [17]). Marescaux *et al.* [18] report application performance data as a proof-of-concept for their packet-switched NoC, but do not offer analysis on the impacts of network design. We use real applications and realistic PE architectures to generate the traffic workloads for our network exploration.
- Most of these networks run at speeds between 25 and 50 MHz which is far below peak FPGA speeds. Dimetalk routers can potentially deliver 100 MHz performance. We deliver a high-performance FPGA implementation where our entire network runs at 166 MHz on comparable or older FPGA technology.

D. Time-Multiplexed Networks

Offline scheduling of communication between processing elements has been studied extensively, particularly in the distributed parallel processing community. Several of these studies examined time-multiplexing of communication networks at the processor level by using FSMs to moderate communication between multiple chips [19], [20] or between PEs integrated onto a single chip [21]. Other projects used time-multiplexing at the logic-level for implementing circuits [22]–[25]. In these designs the time-multiplexed support is built into the native FPGA or simulation-engine architecture.

The Virtual Wires project [26] examined time-multiplexing of resources not built into a native FPGA or ASIC, but overlaid on top of an FPGA. Virtual Wires attempts to overcome pin limitations by time sharing each physical wire among logical wires and pipelining these connections at a high frequency. To route and schedule physical wires, they developed a greedy spatial and temporal scheduler [27], demonstrating a significant increase in usable I/O bandwidth. Instead of time-multiplexing just chip I/O communication, our work attempts to time-multiplex communication over an entire network overlaid on FPGA resources. We use a similar space-time greedy router to schedule our network.

Some prior work has begun to explore the tradeoff between static scheduling and dynamic routing. The RAW microprocessor [28] utilized a static scheduler to compile streams of computations across PEs, but also supported dynamic hardware to allow for unresolvable events such as memory stalls and interrupts. The designers discovered that a combination of static scheduling techniques and software routines handled dynamic events more efficiently than dedicated hardware. The aSoC architecture [29] utilized a statically scheduled reconfigurable communications processor to network together PEs and compared it to dynamic routing models. They found that static scheduling moderately increased performance, but at an unspecified increase in area due to context memory. aSoC supported only 16 PEs and a light traffic load where PE processing time generally dominated communication time. Rather than simply examining one system size or configuration, this paper attempts to broadly explore the design space tradeoff between static scheduling and dynamic routing, and quantifies when either time-multiplexing or packet-switching are preferred under various applications conditions. Our exploration ranges from 4–4096 PEs and includes traffic loads which can often dominate node processing times.

III. TOPOLOGY

A. Performance Analysis

Network topology can have a large impact on the performance of a NoC. We identify several quantitative network characteristics which bound the number of cycles required for communication.

1) *PE Input Serialization*: In our design, we assume that the PE is capable of processing one external input message

TABLE II

SUMMARY OF PRIOR WORK IN FPGA NETWORK-ON-A-CHIP DESIGNS

NoC	Freq (MHz)	Size	Chip	Switch	
				Params.	Area
IMEC [18]	40	3 × 3 (8.3)	V2 Pro 40	16-bit 2 in 2 out	446 slices 1 BRAM
Hermes [16]	25	2 × 2 (25)	V2 1000	8-bit 5 in 5 out	316 slices
LiPaR [17]	33	3 × 3 (30)	V2 Pro 30	8-bit 5 in 5 out	437 slices
Dime-talk [30]	100	–	V2 -4	32-bit 4 in 4 out 2 simult. routes	450 slices 1 BRAM

Figures in parenthesis indicate percentage of area in interconnect; V2=Virtex-2

or self message per cycle. A bound on cycle count based on input messages can be computed as follows:

$$T_{input} = N_{input} + N_{self} \quad (1)$$

2) *PE Output Serialization*: There is a similar bound on outgoing messages:

$$T_{output} = N_{output} + N_{self} \quad (2)$$

3) *Network Bisection*: The bisection width of our network limits the maximum number of messages that can cross from one side to the other on a given cycle. If the number of message bits is greater than the number of physical wires crossing the bisection, then communication must be serialized across the bisection.

$$T_{cut} = \left\lceil \frac{N_{message.bits}}{N_{topcut}} \right\rceil \quad (3)$$

The top-level bisection might not be the largest serial bottleneck in the network. Hence, we need to recursively bisect the network and identify the most limiting of cuts (T_{cut_i}).

$$T_{bw} = \max_i (T_{cut_i}) \quad (4)$$

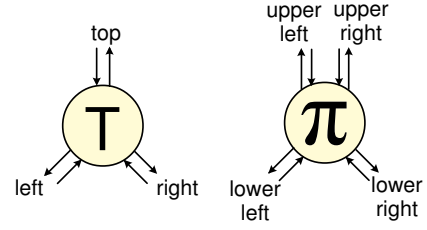
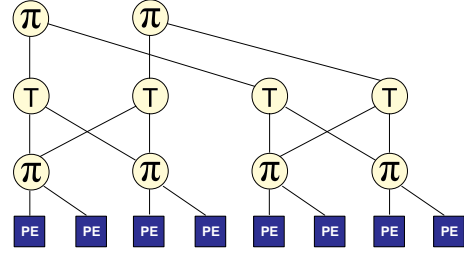
4) *Network Latency*: If the network is sufficiently large, several cycles may be required to traverse the network from one end to the other.

$$T_{latency} = \max_i (route_i) \quad (5)$$

We force sequentialization between the input and output message handling in our PEs. Thus, the lower bound on performance of a topology is follows:

$$T_{cycles} = \max(T_{input} + T_{output}, T_{bw}, T_{latency}) \quad (6)$$

The parameters discussed here can help us analyze and understand the performance of the topologies we consider. These topologies also have widely different area requirements. Thus, different topologies may be more efficient at different chip areas. For example, a binary tree would be best if

Fig. 2. Conceptual Diagram of T and π SwitchesFig. 3. Conceptual Diagram of $p=0.5$ BFT

the number of PEs in the system were small, but it may become bandwidth limited for large PE counts due to the increased number of messages sent into and out of the network (Figure 1).

B. Butterfly Fat Trees

As noted in Section II, most of the FPGA NoC work has focused on building on-chip meshes. Meshes are preferred due to their simple STEP-AND-REPEAT layout, asymptotic latency properties, and the prolific research that has been conducted in devising fair, deadlock-free routing algorithms. However, in our analysis, we observed that Butterfly Fat Trees (BFTs) [9], [31] achieve at least as high performance as the mesh at equivalent chip sizes.

We choose to build BFTs of arity-2 with different values of the Rent parameter p . Increasing the Rent parameter of the BFT increases its bisection bandwidth ($N_{topcut} = c \times n^p$), thereby allowing more bisection traffic per cycle. Larger values of p also require more switches, which influences the number of PEs that can be packed on a chip. We build these trees as a composition of T and π switchboxes, shown in Figure 2 [32], [33]. Each level in the tree uses either T or π switchboxes as shown in Figure 3. We build both packet-switched and time-multiplexed versions of the BFT for our study. We build the BFT switches (Figures 5–6) using basic split and merge primitives (Figure 4). We layout the network on a Virtex-2 6000 hierarchically and pipeline the upper stages of the tree based on layout feedback to retain high performance even when stages cannot be placed adjacent to one another.

IV. PACKET SWITCHED

The packet-switched split and merge primitives (Figure 4) interface with each other using data-presence and back-pressure based flow control. To deal with potential network blocking, we design our primitives with input queues.

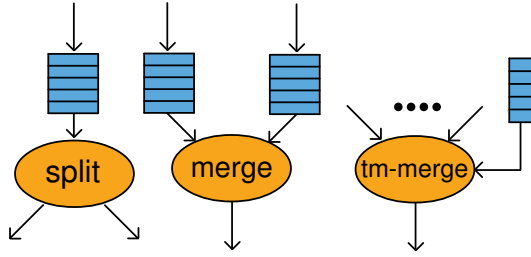


Fig. 4. Conceptual Diagram of Split, Merge and TM-Merge Primitives

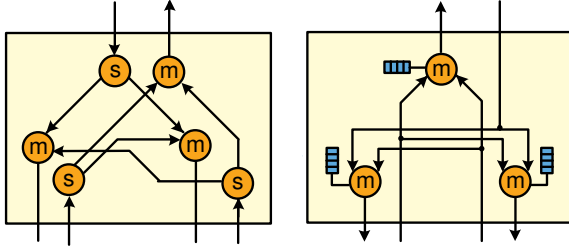


Fig. 5. Conceptual Diagram of BFT T Switch (PS & TM)

The split primitive has one input, two outputs and a routing function which selects one of the two outputs for the incoming packet. The split primitive computes the routing decision in a single cycle based on the destination address of the packet. This routing function determines the path packets will take on the topology. We use an adaptive routing algorithm for packets climbing up the BFT. A single address bit is used to make a routing decision at each switchbox during descent.

The merge primitive has two inputs and one output. Packets arriving on the two inputs must compete for a single output. A simple scheme of arbitration would be to select from the two input ports in a round robin fashion. We use a more adaptive scheme that selects a packet based on FIFO occupancies of the input queues.

Area and latency figures for packet-switched primitives are shown in Table III, assuming a 32-bit datapath (16-bits data, 16-bits destination address) and a buffer depth of 16. We implement buffers in the input queues using SRL16s [34] that allow a compact implementation of storage. The queue controller that we implement around the buffers forms a large portion of the area of the split and merge (Table III). The

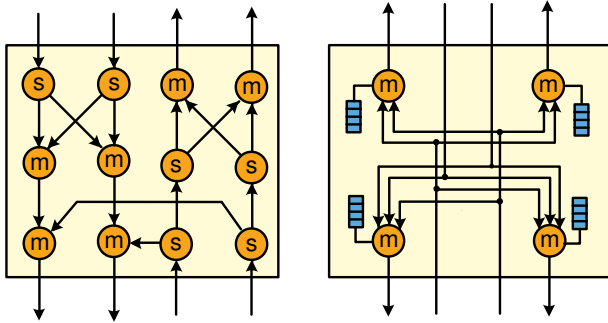


Fig. 6. Conceptual Diagram of BFT π Switch (PS & TM)

TABLE III
AREA, LATENCY AND SPEED OF 32-BIT PACKET-SWITCHED SWITCHING PRIMITIVES WITH 16-ENTRY BUFFERS

Primitive	Area (Slices)		Latency (Cycles)	Speed (MHz)	
	Queue				
	Ctrl	Buffer	Total		
Split	30	33	79	2	218
Merge	60	66	165	2	200

TABLE IV
AREA, LATENCY AND SPEED OF 16-BIT, 1024-DEEP TIME-MULTIPLEXED SWITCHING PRIMITIVES

Primitive	Area (Slices)		Latency (Cycles)	Speed (MHz)
	Context	Total		
Merge2	32	41	1	219
Merge3	64	80	1	204

entire packet-switched network can run at 166MHz limited by the speed of the ConceptNet PE.

V. TIME MULTIPLEXED

The time-multiplexed network statically schedules all communication between PEs over the physical network. It is composed solely from the time-multiplexed merge primitive (Figure 4). Each merge primitive contains context memory which stores the routing decisions that have been computed offline. For a given cycle, data in the context memory indicates which merge input is routed on that cycle. Since primitives need to store routing decisions for every cycle, context memory can dominate the area of the merge primitive for large cycle counts. The following equations describe the area in slices required for time-multiplexed merge primitives as a function of datapath and context depth:

$$Area_{Merge2} = \frac{ContextDepth}{32} + \frac{DataWidth}{2} \quad (7)$$

$$Area_{Merge3} = 2 \times \frac{ContextDepth}{32} + DataWidth \quad (8)$$

Table IV shows the area and latency of 16-bit time-multiplexed primitives for 1024 cycles of context. Context memories are implemented with SRL16s for compact storage. The entire time-multiplexed network is capable of running at 166 MHz.

The static scheduling of the network is performed prior to runtime by a spatial and temporal router. The router is capable of computing schedules using two different algorithms: greedy (similar to that of [27]) and Pathfinder [35]. The greedy router simply allocates paths for messages on the shortest available path in space and earliest possible slot in time, avoiding congested resources. Pseudocode for the greedy algorithm is shown in Figure 7. The Pathfinder router allows routes to be allocated on congested resources and attempts to negotiate congestion through rerouting. Resources which are congested or have a history of congestion become more costly to use and therefore over time uncongested paths are found. Both routers

```

GREEDYROUTE
foreach route
  // start search with source's wires
  foreach wire in source.outputWires
    if (!wire.occupied)
      priorityQueue.push(wire)
  // spread on wires until touch node that is sink
  while network element node != sink
    // wires closer to sink sorted first
    // wires earlier in time sorted next
    cur_wire ← priorityQueue.pop()
    node ← cur_wire.sink
  foreach wire in node.outputWires
    if (!wire.occupied)
      priorityQueue.push(wire)

```

Fig. 7. Greedy Algorithm for Space-Time Routing

utilize a fast A* algorithm (e.g. [36]) to find shortest paths. Additionally, the router is capable of bounding the quality of the computed schedules by computing lower bounds on total cycle time due to physical latency, bandwidth, and PE IO limitations (Section III-A). Our greedy router empirically performs within 15% of all lower bounds (representative examples can be seen in Figures 14–15). Due to the speed and efficiency of the greedy router we avoided using Pathfinder to route communication.

VI. APPLICATION

We map ConceptNet: Spreading Activation Step with diverse communication requirements onto our networks:

ConceptNet [4] is common-sense reasoning knowledge base represented as a graph. Nodes represent nouns or noun-verb pairs (e.g. “cookie”, “eat cookie”) and edges represent semantic relationships (e.g. “used for”, “is a”). Applications of ConceptNet include topic-jisting, analogy-making, text summarization, and semantic prediction.

A key step in the ConceptNet algorithm is spreading activation. In spreading activation, an initial set of graph nodes (corresponding to the set of keywords for a particular query) are activated. More details on our FPGA implementation of spreading activation in ConceptNet are provided in [2]. A single step of spreading activation involves sending an activation potential from each node to its neighbors along its edges, weighted by the edge coefficient. The percentage of active edges (activity factor) over the whole graph depends on the initial query and what step of the spreading activation is being performed. As the time-multiplexed network must compute schedules allowing for all possible communication taking place, we performed our time-multiplexed experiments at 100% activity. The packet-switched experiments were run at 100% activity and for a range of activity factors between 1%–100% which correspond to consecutive steps in selected queries.

Nodes in our ConceptNet graph are limited to 128 edges of fanout or fanin in order to bound the node size and serial processing time. As we can only process a single edge per

cycle in a PE (Eq. 1, 2), the node with the maximum number of fanout or fanin edges imposes a limit on overall system performance.

VII. METHODOLOGY

To evaluate the performance of our networks, we constructed a Java-based infrastructure to simulate the packet-switched network with cycle accuracy and to compute schedules for the time-multiplexed network. We mapped applications to our networks using a partitioner and placer based on MLPart v5.2.14 which is part of the UMPack [37] package. While we ensure that single chip logic and interconnect resources are sufficient to map our applications, we assume that application graphs can be mapped to the available on-chip BRAMs. We also assume that the required instruction memory for the time-multiplexed PEs can fit in BRAMs. This allows us to select a message order which optimizes communication time for static scheduling.

ConceptNet consists of four distinct sets of predicates, each of differing breadth and accuracy. The experiments described here were performed on the smallest, highest quality subset consisting of 14,000 nodes and 27,000 edges. This minimal subset exercises the core algorithm while containing simulation run times.

We use our Java infrastructure to generate a structural VHDL netlist for a given network configuration. We wrote VHDL for the ConceptNet PEs, implementing the 3 phase algorithm of the GraphStep system architecture [2]. We implement multipliers required in the PE using pipelined LUT-level logic instead of the on-chip 18x18 multipliers for speed. Since we decompose our switchboxes into split and merge primitives, we can pipeline and optimize at the level of these individual primitives for high performance (see Tables III and IV). FIFO logic within the split and merge blocks is also internally pipelined. We synthesize, place, and route this entire VHDL design using the Synplicity Compiler v8.0 and the Xilinx ISE v8.1i to obtain hardware operating frequency and slice count results. Based on post place-and-route timing analysis, any long wires that constrain performance are further pipelined.

VIII. RESULTS

We present three quantitative comparisons to help characterize the tradeoffs between packet-switching and time-multiplexing. First, we compare the performance difference between offline and online scheduling by routing identical topologies for identical 100% activity communication loads. Second, we consider the impact of area and compare the performance difference between packet-switching and time-multiplexing for identical areas. To introduce this comparison we examine the balance of compute and interconnect in order to determine the correct topologies to compare across area. Third, for given area capacities we examine performance while varying the activity factor of our communication loads.

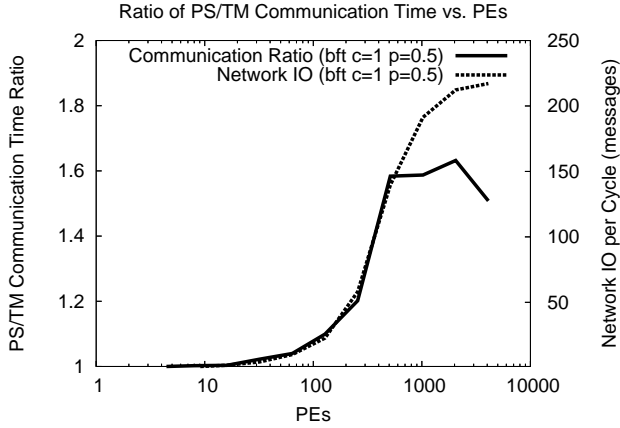


Fig. 8. Ratio of Time-Multiplexed/Packet-Switched Communication Time for Identical Topologies

A. Identical Topologies

To characterize the inherent performance difference between offline and online routing, we routed 100% activity communication loads on equivalent topologies, measuring the total number of communication cycles required to route. We collected data for BFTs over a range of channel widths ($c = 1, 2$) and Rent parameters ($p = 0, 0.5, 0.67, 1$). The ratio of packet-switched to time-multiplexed communication time as a function of PEs for a representative BFT with $c = 1$ and $p = 0.5$ is shown in Figure 8. Network I/O per cycle (as in Figure 1) is also shown on the same graph. At low numbers of PEs (<100) we see that offline and online routing produce equivalent cycle counts. Small numbers of PEs induce a light communication load (1-20 messages per cycle) and little offline/online differentiation. As the number of PEs increase (>100) the communication load increases (100s of messages per cycle), and we begin to see offline routing outperform online routing. Offline routing is able to take advantage of global information to make optimal routing decisions on larger networks, while online routing is limited to making local decisions which affect the overall quality of route. As a result, online routing requires up to 63% more cycles than offline routing to route larger networks.

B. Identical Area Constraints

To fully characterize the performance difference between our packet-switched and time-multiplexed networks we must also consider the area they consume. Before comparing area-time curves for time-multiplexing and packet-switching, we first examine the curves separately to determine the correct BFT configurations for comparison.

1) *PE vs. Interconnect Tradeoff*: Choosing the best network configuration for a given area for either time-multiplexing or packet-switching relies on balancing compute and interconnect. Trading compute (PEs) for interconnect (larger c or p) may actually increase performance for a given area. Consider the following example. Given a XC2V6000 with an area of 32K slices, we fit 32 packet-switched PEs in a BFT $c = 1$,

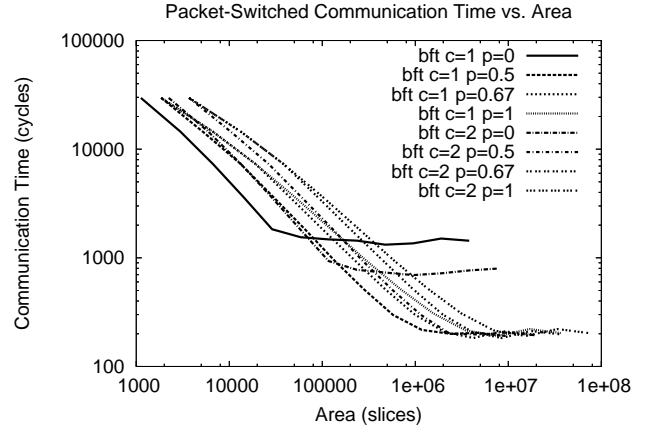


Fig. 9. Packet-Switched Communication Cycles vs. Area

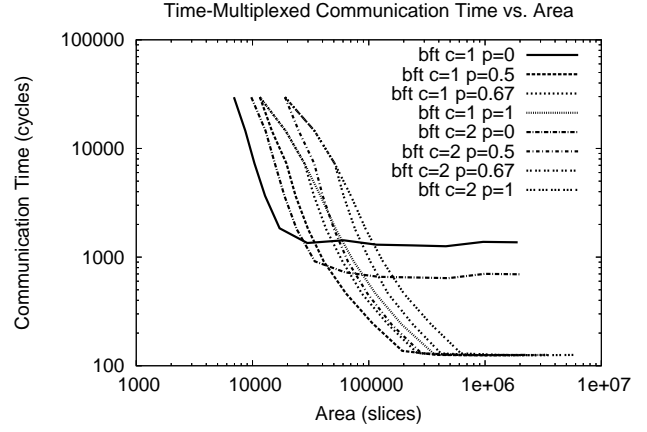


Fig. 10. Time-Multiplexed Communication Cycles vs. Area

$p = 0$ configuration. The smaller number of switches at $c = 1$, $p = 0$ allows us to pack more PEs into the chip area, providing the optimal performance in that area (1800 cycles, as shown in Figure 9). In a XC4VLX200 with 100K slices we can fit 64 PEs, but since the number of messages scales quickly as we increase PEs (Figure 1), a $c = 1$, $p = 0$ configuration does not provide enough bandwidth to maintain high performance. Although a $c = 2$, $p = 0$ network can fit only 32 PEs, the extra bandwidth actually increases performance (1000 cycles vs. 1500 cycles, as shown in Figure 9).

Figures 9–10 illustrate this tradeoff more generally, plotting communication time at 100% activity as a function of area over several BFT configurations for both networks. We see characteristic network performance bounds (Section III) begin to emerge for each topology. As network sizes increase for both packet-switching and time-multiplexing, we see that BFTs with low Rent parameters become bisection limited (Eq. 3), while BFTs with higher Rent parameters are preferred. At low areas we are serialization bottlenecked, and therefore topologies with minimal switch area that allow us to pack the most PEs require fewer cycles to route. At larger areas those topologies become bandwidth bottlenecked, and topologies

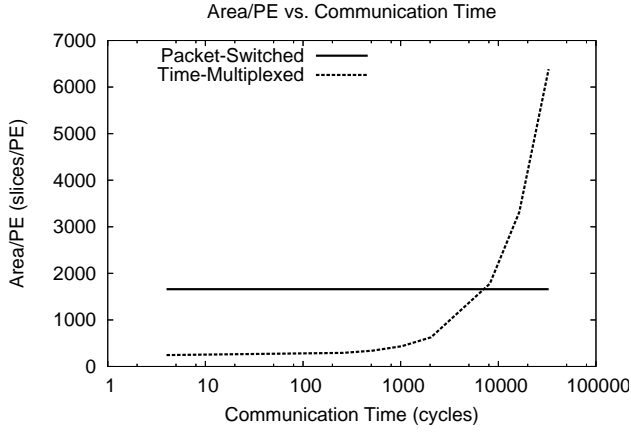


Fig. 11. Normalized Area vs. Communication Cycles

with more bisection bandwidth and hence more switch area are preferred.

One important performance characteristic to note is that beyond 200K slices, time-multiplexed performance remains constant at around 100 cycles. As ConceptNet nodes are limited to 128 edges of fanout or fanin, at large areas and PE counts total communication time is limited by PEs processing the edges of high fanout or fanin nodes (Eq. 1, 2). Communication time could be further reduced by fanout or fanin reduction and node decomposition.

2) *Area-Time Tradeoff*: For equivalent topologies at the same PE count, packet-switched and time-multiplexed networks may use significantly different amounts of area due to differences in switch sizes and the amount time-multiplexed context memory needed. Figure 11 shows normalized area as a function of communication cycles for a BFT with $c = 1$ $p = 0.5$. At low cycle counts time-multiplexing requires a factor of 5 less area/PE than packet-switching due to smaller switches and negligible context memory area. As cycle counts increase, however, context memory area increases and time-multiplexing becomes costly. Beyond 8K cycles the BFT time-multiplexed implementation occupies more area/PE than its packet-switched counterpart.

To fairly compare this area-time tradeoff between time-multiplexing and packet-switching, we examine the best BFT configurations from Figures 9–10 over all area points in our comparison. The composite area-time curves for packet-switched and time-multiplexed are shown in Figure 12. At smaller areas (<12 K slices) time-multiplexing requires more cycles to route than packet-switching. Smaller areas fit fewer PEs, resulting in higher cycle counts. As area increases, however, time-multiplexing can fit more PEs, decreasing serialization and reducing cycle counts.

To quantify this tradeoff, the ratio of packet-switched to time-multiplexed communication time as a function of area over these optimal topology points is shown in Figure 13. We see that at smaller areas (1K–12K slices) time-multiplexing is inefficient and requires around $2\times$ as many cycles to route as packet-switching. At larger areas (12K–200K slices) packet-

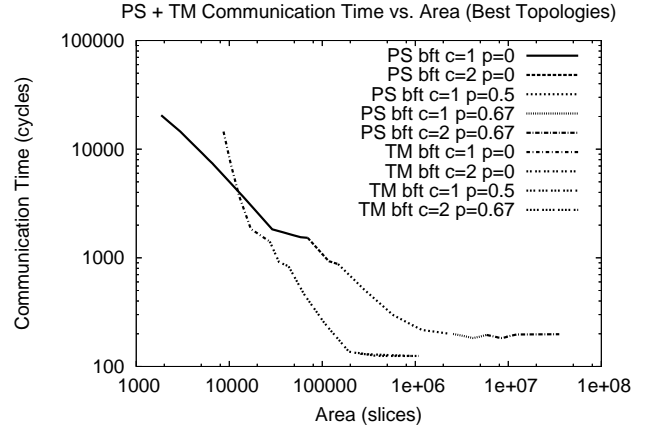


Fig. 12. Communication Cycles vs. Area for Best Topologies

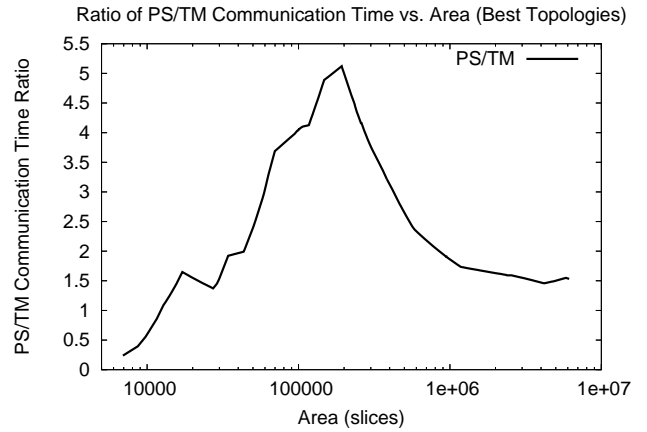


Fig. 13. Ratio of Time-Multiplexed/Packet-Switched Communication Time for Identical Area

switching requires $1\text{--}5\times$ as many cycles to route as time-multiplexing. However, at the largest area values (200K–600M slices), time-multiplexing’s $5\times$ advantage begins to shrink down to around $2\times$. This is due to time-multiplexed performance being limited by ConceptNet’s 128 edge fanout or fanin limit. During this interval the packet-switched network is able to close the performance gap.

C. Activity Factors

Thus far we have compared packet-switching and time-multiplexing assuming 100% communication loads. For activity factors less than 100% time-multiplexing must still route all possible communication, but packet-switching only needs to route those edges that are active. At some activity factor less than 100% packet-switching should be able to outperform time-multiplexing when both are given the same amount of area. Figures 14–15 plot communication time vs. activity for the best time-multiplexed and packet-switched topologies at areas corresponding to the capacities of a XC2V6000 and a XC4VLX200. At 32K slices packet-switching outperforms time-multiplexing for activity factors under 40%. As area increases and time-multiplexing becomes more area efficient;

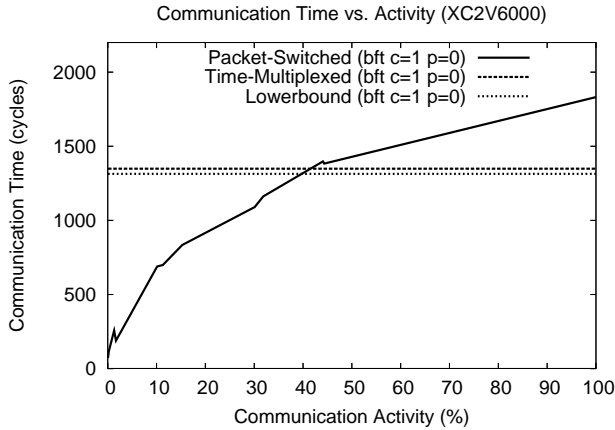


Fig. 14. Communication Cycles vs. Activity on XC2V6000 (32K slices)

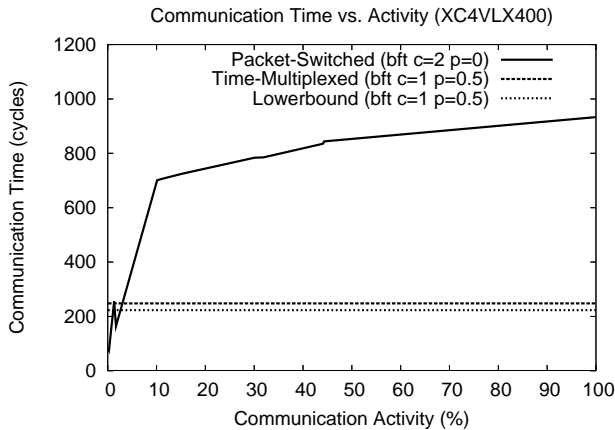


Fig. 15. Communication Cycles vs. Activity on XC4VLX200 (100K slices)

at 100K slices packet-switching’s advantage is limited to under 5% activity.

IX. CONCLUSIONS

We demonstrate scalable, high performance implementations of packet-switched and time-multiplexed FPGA overlay networks operating at 166MHz. To aid designers in choosing the appropriate communication pattern between time-multiplexing and packet-switching, we characterize the tradeoffs associated with these networks and quantify the application conditions under which each is preferred. For our set of applications, offline scheduling offers up to a 63% performance increase over online scheduling for equivalent topologies. When applying designs to equivalent area, packet-switching is up to $2\times$ faster for small areas while time-multiplexing is up to $5\times$ faster for larger areas. For activity factors less than 40% or 5%, packet switching offers better performance at 32K slices and 100K slices respectively.

X. FUTURE WORK

Communication patterns and densities can vary greatly between different applications. Exploration of more applications would help us to better characterize the tradeoffs between

packet-switching and time-multiplexing for any given general application. We are particularly interested in mapping larger communication graphs with smaller fanout limitations in order to increase network communication and test the capabilities of our networks.

Time-multiplexing currently stores context memory as un-encoded data, and significant area savings could be achieved through compression. The efficiency of packet-switching could also be increased through improved routing algorithms and hardware designs. Finally, our network design space exploration has thus far been limited to single chip networks. We hope to extend this work to multiple-chip networks, examining similar area-time tradeoffs.

ACKNOWLEDGMENTS

This work was supported in part by DARPA under grant FA8750-05-C-0011, the NSF CAREER program under grant CCR-0133102, and the Microelectronics Advanced Research Consortium (MARCO) as part of the efforts of the Gigascale Systems Research Center (GSRC).

REFERENCES

- [1] E. Caspi, M. Chu, R. Huang, N. Weaver, J. Yeh, J. Wawrzynnek, and A. DeHon, “Stream computations organized for reconfigurable execution (SCORE): Extended abstract,” in *Proceedings of the International Conference on Field-Programmable Logic and Applications*, ser. LNCS. Springer-Verlag, August 28–30 2000, pp. 605–614.
- [2] M. deLorimier, N. Kapre, N. Mehta, D. Rizzo, I. Eslick, R. Rubin, T. E. Uribe, T. F. Knight, Jr., and A. DeHon, “Graphstep: A system architecture for sparse-graph algorithms,” in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2006.
- [3] A. DeHon, “Balancing Interconnect and Computation in a Reconfigurable Computing Array (or, why you don’t really want 100% LUT utilization),” in *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, February 1999, pp. 69–78.
- [4] H. Liu and P. Singh, “ConceptNet – A Practical Commonsense Reasoning Tool-Kit,” *BT Technical Journal*, vol. 22, no. 4, p. 211, October 2004.
- [5] C. L. Seitz, “The cosmic cube,” *Communications of the ACM*, pp. 22–33, January 1985.
- [6] V. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic*. New York, NY: Academic Press, 1965.
- [7] H. Siegel, *Interconnection Networks for Large-Scale Parallel Processing*. Lexington, MA: Lexington Books, 1985.
- [8] C. P. Kruskal and M. Snir, “The performance of multistage interconnection networks for multiprocessors,” *IEEE Transactions on Computers*, vol. C-32, no. 12, pp. 1091–1098, Dec. 1983.
- [9] C. E. Leiserson, “Fat-trees: Universal networks for hardware efficient supercomputing,” *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, Oct. 1985.
- [10] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*. San Francisco: Morgan Kaufmann Publishers, 2003.
- [11] W. J. Dally and B. Towles, “Route packets, not wires: On-chip interconnection networks,” in *Design Automation Conference*, 2001, pp. 684–689.
- [12] L. Benini and G. D. Micheli, “Networks on chips: A new soc paradigm,” *IEEE Computer*, vol. 35, no. 1, pp. 70–78, 2002.
- [13] R. J. Chapuis, *100 Years of Telephone Switching: Electronics, Computers and Telephone Switching, 1960-1985*. IOS Press, 2003.
- [14] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufman, 2004.
- [15] P. P. Pande, C. Grecu, A. Ivanov, R. Saleh, and G. D. Micheli, “Design, synthesis and test of networks on chips,” *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 404–413, 2005.

- [16] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "Hermes: an infrastructure for low area overhead packet-switching networks on chip," *INTEGRATION, The VLSI Journal*, vol. 38, no. 1, pp. 69–93, 2004.
- [17] B. Sethuraman, P. Bhattacharya, J. Khan, and R. Vemuri, "Lipar: A lightweight parallel router for fpga based networks on chip," in *Proceedings of the Great Lakes Symposium on VLSI*, 2005.
- [18] T. Marescaux, V. Nollet, J.-Y. Mignolet, A. B. W. Moffat, P. Avasare, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins, "Run-time support for heterogeneous multitasking on reconfigurable socs," *INTEGRATION, The VLSI Journal*, vol. 38, no. 1, pp. 107–130, 2004.
- [19] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P. Tseng, J. Sutton, J. Urbanski, and J. Webb, "iwarp: an integrated solution to high-speed parallel computing," in *Proceedings of Supercomputing*, November 1988, pp. 330–339.
- [20] D. Shoemaker, F. Honore, C. Metcalf, and S. Ward, "Numesh: An architecture optimized for scheduled communication," *Journal of Supercomputing*, vol. 10, no. 3, pp. 285–302, 1996.
- [21] J. Oliver, R. Rao, P. Sultana, J. Crandall, E. Czernikowski, L. W. J. IV, V. Akella, and F. T. Chong, "Synchrosalar: A multiple clock domain, power-aware, tile-based embedded processor," in *Proceedings of the International Symposium on Computer Architecture*, 2004.
- [22] M. Denneau, "The yorktown simulation engine," in *19th Design Automation Conference*. IEEE, 1982, pp. 55–59.
- [23] N. B. Bhat, K. Chaudhary, and E. S. Kuh, "Performance-oriented fully routable dynamic architecture for a field programmable logic device," University of California, Berkeley, UCB/ERL M93/42, June 1993.
- [24] D. Jones and D. Lewis, "A time-multiplexed fpga architecture for logic emulation," in *Proceedings of the IEEE Custom Integrated Circuits Conference*. IEEE, May 1995, pp. 495–498.
- [25] A. DeHon, "Reconfigurable Architectures for General-Purpose Computing," MIT Artificial Intelligence Laboratory, 545 Technology Sq., Cambridge, MA 02139, AI Technical Report 1586, October 1996. [Online]. Available: http://www.cs.caltech.edu/~andre/abstracts/dehon_phd.html
- [26] J. Babb, R. Tessier, and A. Agarwal, "Virtual wires: Overcoming pin limitations in fpga-based logic emulators," in *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, April 1993, pp. 142–151.
- [27] C. Selvidge, A. Agarwal, M. Dahl, and J. Babb, "Tiers: Topology independent pipelined routing and scheduling for virtualwiretm compilation," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, 1995, pp. 25–31.
- [28] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal, "Baring it all to software: Raw machines," *IEEE Micro*, vol. 30, no. 9, pp. 86–93, September 1997.
- [29] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier, "An Architecture and Compiler for Scalable On-Chip Communication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 7, p. 711, 726 2004.
- [30] "Simplifying communication across dsp networks," Programmable World, 2003, <<http://www.mactivity.com/xilinx/pw2003/workshops/presos/wsa3.nallatech.pdf>> .
- [31] R. I. Greenberg and C. E. Leiserson, *Randomness in Computation*, ser. Advances in Computing Research. JAI Press, 1988, vol. 5, ch. Randomized Routing on Fat-Trees, earlier version MIT/LCS/TM-307.
- [32] W. Tsu, K. Macy, A. Joshi, R. Huang, N. Walker, T. Tung, O. Rowhani, V. George, J. Wawrzynek, and A. DeHon, "HSRA: High-Speed, Hierarchical Synchronous Reconfigurable Array," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, February 1999, pp. 125–134.
- [33] A. DeHon, R. Huang, and J. Wawrzynek, "Hardware-Assisted Fast Routing," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2002, pp. 205–215.
- [34] *The Programmable Logic Data Book-CD*, Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124, 2005.
- [35] L. McMurchie and C. Ebling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays*. ACM, February 1995, pp. 111–117.
- [36] R. Tessier, "Negotiated A* Routing for FPGAs," in *Proceedings of the 5th Canadian Workshop on Field Programmable Devices*, June 1998.
- [37] A. Caldwell, A. Kahng, and I. Markov, "Improved Algorithms for Hypergraph Bipartitioning," in *Proceedings of the Asia and South Pacific Design Automation Conference*, January 2000, pp. 661–666.