Implementation and Evaluation of a Dynamically Routed Processor Operand Network

Paul Gratz*, Karthikeyan Sankaralingam†, Heather Hanson*, Premkishore Shivakumar†
Robert McDonald†, Stephen W. Keckler†, and Doug Burger†
* Department of Electrical and Computer Engineering, The University of Texas at Austin
† Department of Computer Sciences, The University of Texas at Austin
cartel@cs.utexas.edu

***Abstract*— Microarchitecturally integrated on-chip networks, or micronets, are candidates to replace busses for processor component interconnect in future processor designs. For micronets, tight coupling between processor microarchitecture and network architecture is one of the keys to improving processor performance. This paper presents the design, implementation and evaluation of the TRIPS operand network (OPN). The TRIPS OPN is a 5x5, dynamically routed, 2D mesh micronet that is integrated into the TRIPS microprocessor core. The TRIPS OPN is used for operand passing, register file I/O, and primary memory system I/O. We discuss in detail the OPN design, including the unique features that arise from its integration with the processor core, such as its connection to the execution unit's wakeup pipeline and its in flight mis-speculated traffic removal. We then evaluate the performance of the network under synthetic and realistic loads. Finally, we assess the processor performance implications of OPN design decisions with respect to the end-to-end latency of OPN packets and the OPN's bandwidth.**

I. Introduction

As process technologies continue to descend into the deep sub-micron range, wire delay and design complexity become limiting factors for current microprocessor designs [1]. In current processor microarchitectures, data and control are conveyed on specialized busses and other ad-hoc interconnect. Some processor designs incorporate extra pipeline stages to accommodate the wire delay global wires require [2]. Wire routing and electrical design of these specialized busses increases in complexity with smaller process technologies. Micronets, or microarchitecturally integrated on-chip networks, provide a solution to this design challenge by offering an alternative to bus-based interconnects that are scalable and have reasonable design complexity.

On-chip networks enjoy a scaling advantage relative to busses since network wire lengths between adjacent routers can be kept short and unidirectional. On-chip networks also enable better pipelining of data between nodes and greater aggregate bandwidth than busses. Finally, design complexity is bounded since a router is designed once and replicated for use wherever needed.

Micronets, a type of on-chip network, are integrated with the microarchitecture of their host processors to improve system performance. For example a micronet may be tightly integrated with a processor's pipeline to reduce packet generation latency, by taking advantage of information available before the full data payload has been computed. Micronets can also be used to implement, in a distributed fashion, higher level protocols that are centralized in more traditional architectures, such as instruction commit.

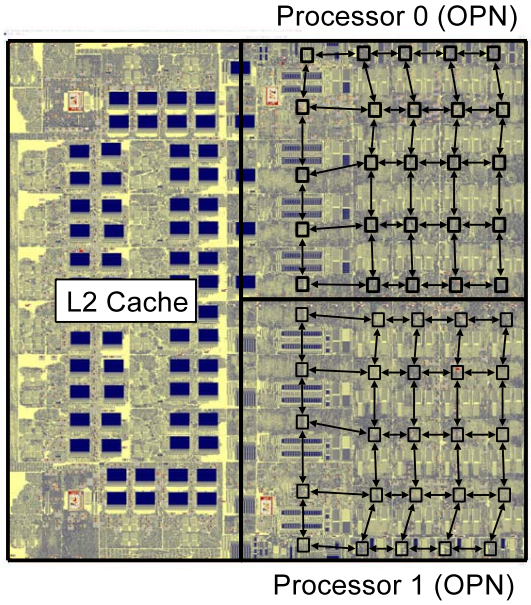

Fig. 1. TRIPS chip plot with operand networks highlighted.

In this paper we discuss and evaluate the design and implementation of one such micronet, the TRIPS prototype processor's operand network (OPN). Figure 1 shows a plot of the TRIPS prototype processor chip, which was fabricated in a 130nm ASIC process. On the right side of the figure are the two processors, each with its own separate operand network, as indicated by the superimposed diagram. Each processor's OPN is a 5x5 dynamically routed 2D mesh network with 140-bit links. The OPN connects a total of 25 distributed execution, register file, and data cache tiles. Each tile is replicated and interacts only with neighboring tiles via the OPN and other control networks. The OPN subsumes the role of several traditional microprocessor interconnect busses, including the operand bypass network, register file read and write interface, and the L1 memory system bus.

We describe our experience in implementing the OPN

in a 130nm process technology as a part of the TRIPS prototype processor and include a discussion of timing and area costs of the network. We also explore the latency and bandwidth of the OPN under statistical loads common in the networking literature as well as realistic loads extracted from TRIPS program execution. Finally, we examine trade-offs in OPN implementation, including the sensitivity of processor performance to network latency and bandwidth.

The remainder of this paper is organized as follows. Section II describes related work in micronets. Section III introduces the TRIPS processor microarchitecture. Section IV describes the design and implementation of the TRIPS OPN, highlighting where the OPN is different from typical on-chip networks. Section V presents an evaluation of the network's performance under different loads and explores the sensitivity of processor performance to OPN latency and bandwidth. Section VI concludes and discusses future work in micronets.

II. Related Work

The first operand bypass network was introduced with the IBM System 360/model 91 to avoid delaying the sequential execution of dependent instructions [3]. This bypass network employed a simple broadcast bus (common data bus) that linked each ALU output to each ALU input. Thus an instruction can receive an operand directly from a preceding instruction's output without the delay of passing it through the register file.

The development of deeply pipelined superscalar architectures drove increases in the complexity and latency of bypass networks. Deeper pipelines increased the number of stages in which an instruction could produce a result or consume an operand. Wider pipelines increase the bypass bus network complexity quadratically with the number of ALUs because of the full connectivity between ALU outputs and ALU inputs. This N^2 scaling rate is not viable beyond a small number of ALUs. The Multiscalar processor architecture used partitioning of the ALUs and register file into separate components connected by a ring to reduce operand delivery complexity [4]. Similarly, the Alpha 21264 architecture divided its four integer ALUs into two clusters to reduce the complexity of its bypass network [5]. Operands produced within one cluster are available for use in the same cluster in the next cycle, but they must pay a single cycle penalty to be used in the other cluster.

Other machines have sought to reduce communication latency between processors through cross-processor register-to-register communication. The M-Machine employed an on-chip cluster switch to connect the register bypass networks for three processors; an instruction writing to a remote register injects its result into the switch, which delivers the data to a waiting instruction on a remote processor [6]. The MIT RAW processor took this strategy further, by using a 4x4 mesh network to interconnect its processor tiles between execution units [7]. The integration of the RAW network into the local bypass network of each execution unit reduced the latency of operand passing between units to three cycles. One interesting feature of RAW is that network routing arbitration and ordering are statically determined. While this strategy simplifies the routers, a compiler or programmer must generate a routing program that executes concurrently with the application program. In addition to the statically routed network, RAW also implemented dynamically routed networks for load/store traffic. The TRIPS OPN is also integrated directly with the execution unit. However, to allow for out-of-order instruction execution and uncertain memory delays, the OPN routers are dynamic. We also employed additional routing optimizations to reduce the per-hop latency to one cycle.

The Monsoon processor was a dataflow architecture that used a custom switched interconnection network to provide similar capabilities as the TRIPS OPN [8]. The WaveScalar processor has a similar philosophy and execution model as TRIPS, but uses a hierarchy of interconnection networks to pass operands between processing elements [9]. Operands are broadcast within the eight processing elements making up one domain. Operands pass through a crossbar switch to travel between the four domains that make up a cluster. Operands traveling to another cluster traverse a 2D mesh network similar to the TRIPS OPN.

Taylor, et al. [10] and Sankaralingam, et al. [11] both present useful taxonomies of operand networks. Taylor categorizes operand networks based on whether the assignment, transport, and ordering are each either static or dynamic. In their terms, the TRIPS OPN has statically assigned operations that are dynamically transported and ordered. Sankaralingam categorizes operand networks based on network organization (point-to-point vs. broadcast), network architecture (single hop vs. multi-hop), and router control (static vs. dynamic). By this taxonomy, the TRIPS OPN is a point-to-point, multi-hop network with dynamic router control.

Pinkston and Shin [12] use data from the 2003 International Technology Roadmap for Semiconductors (ITRS-2003) [13] to demonstrate how trends in semiconductor technology are leading to partitioned microsystem architectures. They provide a taxonomy that categorizes microsystem architectures based on how they are partitioned, with the insight that the trend toward partitioned architectures is driving the adoption of on-chip networks. By this taxonomy TRIPS processor core is physically partitioned in a compiler-visible form.

Dally and Towles [14] proposed a 2D torus network as a replacement for general on-chip interconnect, but not specifically for operand networks. They claim that on-chip network modularity would shorten the design time and reduce the wire routing complexity. Our experience bears this out, as the design, implementation, timing optimization, and verification of the TRIPS OPN were all straightforward. On-chip routed networks have also been proposed for use in SoCs (system-on-a-chip) such as in CLICHE [15], in which a 2D mesh network is proposed to interconnect a heterogeneous array of IP blocks.

III. TRIPS Processor Overview

TRIPS is a distributed processor consisting of multiple tiles connected via multiple micronets. Figure 2 shows a tile-level diagram of the processor with its OPN links. The

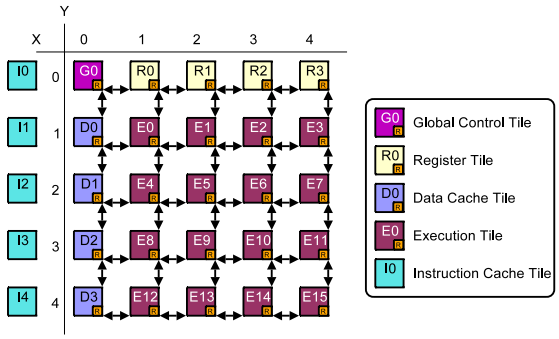

Fig. 2. Block diagram of the TRIPS processor core with tiles and OPN network connections.

processor contains five types of tiles: execution tiles (ET) which contain ALUs and reservation stations, register tiles (RT) which each contain a fraction of the processor register file, data tiles (DT) which each contain a fraction of the level-1 data cache, instruction tiles (IT) which each contain a fraction of the level-1 instruction cache, and a global control tile (GT) which orchestrates instruction fetch, execution and commit. In addition, the processor contains several control networks for implementing protocols such as instruction fetch, completion, and commit in a distributed fashion. Tiles communicate directly only with their nearest neighbors to keep wires short and mitigate the effect of wire delay.

ISA and execution model: TRIPS is an explicit datagraph execution (EDGE) architecture, an instruction set architecture with two key features: (1) the hardware fetches, executes, and commits blocks of instructions, rather than individual instructions, in an atomic fashion; and (2) within a block, instructions send their results directly to other instructions waiting to execute, rather than communicating through a common register file [16]. The compiler is responsible for constructing blocks, which can contain up to 128 instructions. Since basic blocks typically contain only a handful of instructions, the TRIPS compiler uses techniques such as predication, loop unrolling, and function inlining to create large hyperblocks. After hyperblock formation, a scheduler maps the block onto the fixed array of 16 execution units, with up to 8 instructions per ET. The scheduler is aware of the topology of the ETs and attempts to minimize the distance between dependent instructions along the program's critical path. The scheduler determines where an instruction will execute and encodes this in the program binary, but the hardware executes instructions in dataflow order based on when an individual instruction's operands arrive.

Block execution: Processing a TRIPS block requires four phases: fetch, execute, complete, and commit. To fetch a block, the GT transmits a fetch request to each of the ITs using the TRIPS global dispatch network (GDN). Each IT then retrieves a portion of the block (32 instructions) from its instruction cache bank and delivers them to pre-allocated reservation stations in the ETs and RTs. An instruction waits in its reservation station until all of its operands have arrived before it can execute. Block execution is instigated by special register read instructions that fetch block inputs from the RTs and deliver them to waiting instructions via the OPN. Instructions within the block then execute in dataflow order. Load and store instructions compute their addresses in the ETs, which are then transmitted to one of the DTs to access the data cache. Addresses are interleaved across the DTs on cache-line boundaries (64 bytes). Register outputs are transmitted back to the RTs where they wait in write queues before updating the architecturally persistent register file banks.

When all of the RTs and DTs have received all of the register writes and stores for the block, they communicate this to the GT via the global control network (GCN). When the GT receives completion notification from all DTs and RTs, the block is complete. If the block has not caused any exceptions, the GT signals to the DTs and RTs that the block can commit. The DTs then update the cache with the store values from the store buffers and the RTs update the register file banks with the contents of the write queues. When all of the state of the block has committed, a new block may be mapped into its place for execution. The TRIPS processor allows up to 8 blocks in-flight and executing simultaneously, with 1 being non-speculative and 7 being speculative. Complete details of the TRIPS microarchitecture and can be found in [17].

During block execution, the TRIPS operand network (OPN) has the responsibility for delivering operands among the tiles. The TRIPS instruction formats contain target fields indicating to which consumer instructions a producer sends its values. At runtime, the hardware resolves those targets into coordinates to be used for network routing. An operand passed from producer to consumer on the same ET can be bypassed directly without delay, but operands passed between instructions on different tiles must traverse a portion of the OPN. The TRIPS execution model is inherently dynamic and data driven, meaning that operand arrival drives instruction execution, even if operands are delayed by unexpected or unknown memory latencies. Because of the data driven nature of execution and because multiple blocks execute simultaneously, the OPN must dynamically route the operand across the processor.

IV. OPN Design and Implementation

The operand network (OPN) is designed to deliver operands among the TRIPS processor tiles with minimum latency. While tight integration of the network into the processor core reduces the network interface latency, two primary aspects of the TRIPS processor architecture simplify the router design and reduce routing latency. First, because of the block execution model, reservation stations for all operand network packets are pre-allocated, guaranteeing that all OPN messages can be consumed at the targets. Second, all OPN messages are of fixed length, one flit broken into header and payload phits.

A. OPN Design Details

The OPN is a 5x5 2D routed mesh network as shown in Figure 2. Flow control is on/off based, meaning that the receiver tells the transmitter when there is enough buffer space available to send another flit. Packets are routed through the

Control phit		Data phit	
Field	bits	Field	bits
Valid	1	Valid	1
Type (LD/ST/etc.)	4	Type (normal/null/exception)	2
Block ID	3	Data operation (access width)	3
Dest. node	6	Data payload	64
Dest. instruction	5	LD/ST Address	40
Source node	6		
Source instruction	5		

TABLE I
BREAKDOWN OF BITS FOR OPN CONTROL AND DATA PHITS.

network in Y-X dimension-order with one cycle taken per hop. A packet arriving at a router is buffered in an input FIFO prior to being launched onward towards its destination. Due to dimension-order routing and the guarantee of consumption of messages, the OPN is deadlock free without requiring virtual channels. The absence of virtual channels reduces arbitration delay and speeds routing.

Each operand network message consists of a control phit and a data phit. The control phit is 30 bits and encodes OPN source and destination node coordinates, along with identifiers to indicate which instruction to select and wakeup in the target ET. The data phit is 110 bits, with room for a 64-bit data operand, a 40-bit address for store operations, and 6 bits for status flags. Table I shows a breakdown of all of the bits in the data and control phits.

The data phit always trails the control phit by one cycle in the network. The OPN supports different physical wires for the control and data phit so one can think of each OPN message consisting of one flit split into a 30-bit control phit and a 110-bit data phit. Because of the distinct control and data wires, two OPN messages with the same source and destination can proceed through the network separated by a single cycle. The data phit of the first message and the control phit of the second are on the wires between the same two routers at the same time. Upon arrival at the destination tile, the data phit may bypass the input FIFO and be used directly, depending on operation readiness. This arrangement is similar to flit-reservation flow control, although here the control phit contains some payload information and does not race ahead of the data phit [18]. In all, the OPN has a peak injection bandwidth of 175 GB/sec when all nodes are injecting packets every cycle at its designed frequency of 400MHz. The network's bisection bandwidth is 70 GB/sec measured horizontally or vertically across the middle of the OPN.

Figure 3 shows a high-level block diagram of the OPN router. The OPN router has five inputs and five outputs, one for each ordinal direction (N, S, E and W) and one for the local tile's input and output. The ordinal directions inputs each have two four entry deep FIFOs, one 30 bits wide for control phits and one 110 bits wide for data phits. The local input has no FIFO buffer. The control and data phits of the OPN

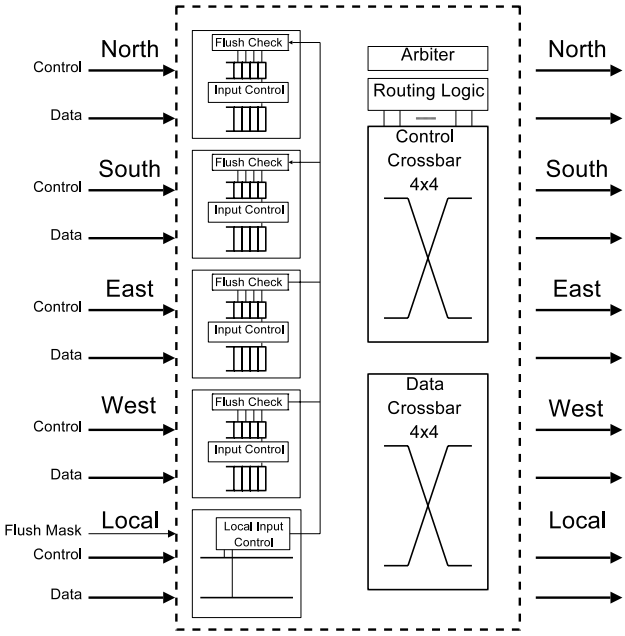

Fig. 3. OPN router microarchitecture.

packet have separate 4x4 crossbars. All arbitration and routing is done on the control phit, in round-robin fashion among all incoming directions. The data phit follows one cycle behind the control phit in lock step, using the arbitration decision from its control phit.

B. OPN/Processor Integration

ET/OPN datapath: Figure 4 shows the operand network datapath between the ALUs in two adjacent ETs. The instruction selection logic and the output latch of the ALU are both connected directly to the OPN's local input port, while the instruction wakeup logic and bypass network are both connected to the OPN's local output. The steps below describe the use of the OPN to bypass data between the ALUs.

- Cycle 0: Instruction wakeup/select on ET 0
  - ET0 selects a ready instruction and sends it to the ALU.
  - ET0 recognizes that the instruction target is on ET1 and creates the control phit.
- Cycle 1: Instruction execution on ET0
  - ET0 executes the instruction on the ALU.
  - ET0 delivers the control phit to router FIFO of ET1.
- Cycle 2: Instruction wakeup/select on ET1
  - ET0 delivers the data phit to ET1, bypassing the FIFO and depositing the data in a pipeline latch.
  - ET1 wakes up and selects the instruction depending on the data from ET0.
- Cycle 3: Instruction execution ET1
  - ET1 selects the data bypassed from the network and executes the instruction.

The early wakeup, implemented by delivering the control phit in advance of the data phit, overlaps instruction pipeline

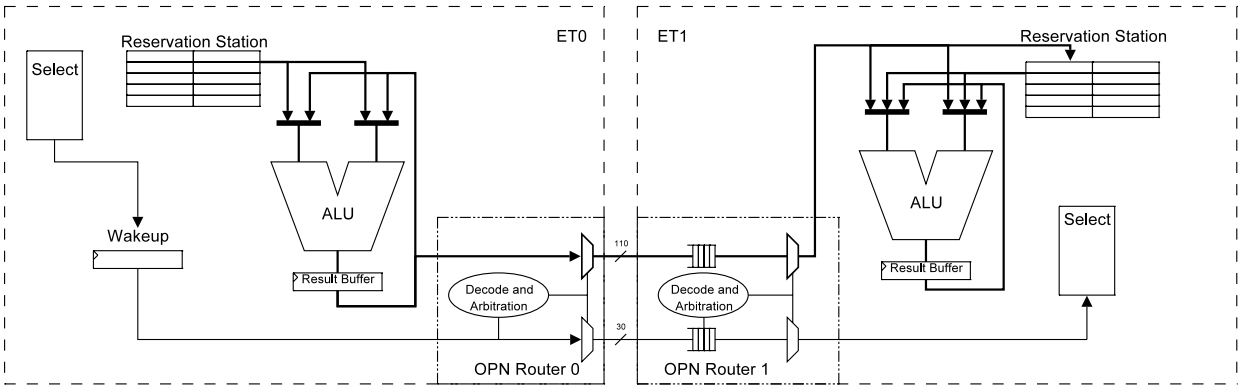

Fig. 4. Operand datapath between two neighboring ETs.

control with operand data delivery. This optimization reduces the remote bypass time by a cycle (to one cycle) and improves performance by approximately 11% relative to a design where the wakeup occurs when the data arrives. In addition, the separation of the control and data phits onto separate networks with shared arbitration and routing eliminates arbitration for the data phit and reduces network contention relative to a network that sends the header and payload on the same wires in successive cycles. This optimization is inexpensive in an on-chip network due to the high wire density.

The OPN employs round-robin arbitration among all of the inputs, including the local input. If the network is under load and chooses not to accept the control phit, the launching node captures the control phit and later the data phit in a local output buffer. The ET will stall if the instruction selected for execution needs the OPN and the ET output buffer is already full. However, an instruction that needs only to deliver its result to another instruction on the same ET does not stall due to OPN input contention. While OPN contention can delay instruction execution on the critical path of program execution, the scheduler is effective at placing instructions to mitigate the distance that operands must travel and the contention they encounter.

Selective OPN message invalidation: Because the TRIPS execution model uses both instruction predication and branch prediction, some of the operand messages are actually speculative. On a branch misprediction or a block commit, the processor must flush all in-flight state for the block, including state in any of the OPN's routers. The protocol must selectively flush only those messages in the routers that belong to the flushed block. The GT starts the flush process by multicasting a flush message to all of the processor tiles using the global control network (GCN). This message starts at the GT and propagates across the GCN within 10 cycles. The GCN message contains a block mask indicating which blocks are to be flushed. Tiles that receive the GCN flush packet instruct their routers to invalidate from their FIFOs any OPN messages with block-identifiers matching the flushed block mask. As the invalidated packets reach the head of the associated FIFOs they are removed. While we chose to implement the FIFOs using shift registers to simplify invalidation, the protocol could also be implemented for circular buffer FIFOs. A collapsing FIFO that immediately eliminates flushed messages could further improve network utilization, but we found that the performance improvement did not outweigh the increased design complexity. In practice, very few messages are actually flushed.

Component	% Router Area	% E-Tile Area
Router input FIFOs	74.6%	7.9%
Router crossbar	20.3%	2.1%
Router arbiter logic	5.1%	0.5%
Total for single router	–	10.6%

TABLE II

AREA OCCUPIED BY THE COMPONENTS OF AN OPN ROUTER.

C. Area and Timing

The TRIPS processor is manufactured using a 130nm IBM ASIC technology and returned from the foundry in September 2006. Each OPN router occupies approximately $0.25mm^2$, which is similar in size to a 64-bit integer multiplier. Table II shows a breakdown of the area consumed by the components of an OPN router. The router FIFOs dominate the area in part because of the width and depth of the FIFOs. Each router includes a total of 2.2 kilobits of storage, implemented using standard cell flip-flops rather than generated memory or register arrays. Utilizing shift FIFOs added some area overhead due to extra multiplexors. We considered using the library generated SRAMs instead of flip-flops, but the area overhead turned out to be greater given the small size of each FIFO.

A single OPN router takes up approximately 10% of the ET's area and 14% of a processor core. While this area is significant, the alternative of a broadcast bypass network across all 25 tiles would consume considerable area and is not feasible. We could have reduced router area by approximately

Component	Latency	% Path
Control Phit Path		
Read from instruction buffer	290ps	13%
Control phit generation	620ps	27%
ET0 router arbitration	420ps	19%
ET0 OPN output mux	90ps	4%
ET1 OPN FIFO muxing and setup time	710ps	31%
Latch setup + clock skew	200ps	9%
Total	2.26ns	–
Data Phit Path		
Read from output latch	110ps	7%
Data phit generation	520ps	32%
ET0 OPN output mux	130ps	8%
ET1 router muxing/bypass	300ps	19%
ET1 operand buffer muxing/setup	360ps	22%
Latch setup + clock skew	200ps	12%
Total	1.62ns	–

TABLE III

CRITICAL PATH TIMING FOR OPN CONTROL AND DATA PHIT.

1/3 by sharing the FIFO entries and wires for the control and data phits. However, the improved OPN bandwidth and overall processor performance justifies the additional area.

We performed static timing analysis on the TRIPS design using Synopsys Primetime to identify and evaluate critical paths. Table III shows the delay for the different elements of the OPN control and data critical paths, matching the datapath of Figure 4. We report delays using a nominal process corner, which we obtained by scaling our worst-case process corner delays by a factor of 2/3. A significant fraction of the clock cycle time is devoted to overheads such as flip-flop read and setup times as well as clock uncertainty (skew and jitter). A custom design would likely be able to drive these overheads down. On the logic path, the control phit is much more constrained than the data phit due to router arbitration delay. We were a little surprised by the delay associated with creating the control phit, which involves decoding and encoding. This path could be improved by performing the decoding and encoding in a previous cycle and storing the control phit with the instruction before execution. We found that wire delay was small in our 130nm process given the relatively short transmission distances. Balancing router delay and wire delay may be more challenging in future process technologies.

D. Design Optimizations

We considered a number of OPN enhancements but chose not to implement them in the prototype to simplify the design. One instance where performance can be improved is when an instruction must deliver its result to multiple consumers. The TRIPS ISA allows an instruction to specify up to 4 consumers, and in the current implementation, the same value is injected in the network once for each consumer. Multicast in the network would automatically replicate a single message in the routers at optimal bifurcation points. This capability would reduce overall network contention and latency while increasing ET execution bandwidth, as ETs would spend less time blocking for message injection. Another optimization would give network priority to those OPN messages identified to be on the program's critical path. We have also considered improving network bandwidth by replicating the operand network by replicating the routers and wires. We examine this optimization further in Section V-E. Finally, the area and delay of our design was affected by the characteristics of the underlying ASIC library. While the trade-offs may be somewhat different with a full-custom design, our results are relevant because not all on-chip networked systems will be implemented using full-custom silicon. Our results indicate that such ASIC designs would benefit from new ASIC cells, such as small but dense memory arrays and FIFOs.

V. OPN EVALUATION

In this section, we evaluate the behavior of the OPN on statistical and realistic network workloads, using our operand network simulator to model the OPN hardware. We characterize the operand network message workload and show that injection is not distributed evenly across the nodes, due to the TRIPS execution model and scheduler optimizations. Finally, we examine the sensitivity of program performance and operand network latency to OPN bandwidth and latency parameters.

A. Methodology

The OPN simulator is a custom network simulator configured with the operand network design parameters. It can inject messages using different traffic patterns, including random and bit-reversal, with variable injection rates. It can also accept a network trace file that specifies source nodes, destination nodes, and injection timestamps. We obtained realistic workload traces from an abstract TRIPS processor performance estimator (*tsim-cyc*), which runs compiled TRIPS programs. This simulator models TRIPS block execution at a high level, but employs a simple analytical performance model without accurate OPN contention estimation. Nonetheless, this simulator matches performance of the logic design of TRIPS to within 25%. The high simulation speed of tsim-cyc allows us to obtain traces for long running programs. However, the message injection times only approximate those that will be seen in hardware. For more detailed analysis, we also used our low-level simulator (*tsim-proc*) which accurately models all aspects of a TRIPS processor core, including network contention. This simulator has been validated for accuracy against the TRIPS RTL and hardware. Unfortunately, the speed of this simulator prevents analysis of large programs.

Our realistic workloads include programs from the EEMBC [19] and SPEC2000 [20] benchmark suites. The 30 EEMBC benchmarks are small enough to run to completion on both tsim-cyc and tsim-proc. The 19 SPEC CPU2000 benchmarks were run with the Minne-SPEC [21] reduced input set, but were still too long-running for tsim-proc. The SPEC benchmarks were run to completion (50 million cycles for the shortest benchmark), or for 300 million cycles after program

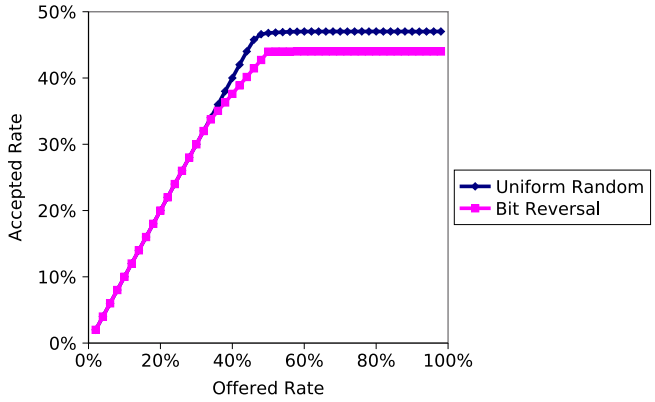

Fig. 5. Offered vs accepted rate for random and bit-reversal traffic.

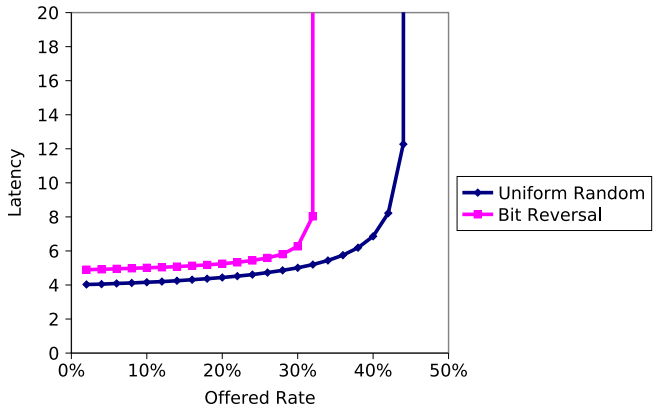

Fig. 6. Offered rate vs average latency for random and bit-reversal traffic.

warmup. The traces include 2–70 million operand messages, depending on the benchmark.

B. Synthetic Statistical Loads

Interconnection networks are typically evaluated by examining their performance on stochastically generated loads. Two common loads are bit-reversal and uniform random traffic. In bit-reversal, each node exchanges packets with a node on the opposite side of the network. The random traffic model randomly chooses source and destination pairs from among all the TRIPS core tiles. Both traffic models inject packets at a uniform random distributed rate. Figure 5 shows the offered vs. accepted rate for both of these types of traffic. The offered rate is the rate at which packets are generated, while the accepted rate is the throughput of the network. In these diagrams, the offered and accepted rates are shown as a percentage of the peak injection bandwidth. The accepted rate tracks the offered rate for bit-reversal traffic up to 33%, from there the accepted rate continues to increase, finally leveling off at 44%. For random traffic the accepted bandwidth tracks the offered bandwidth up to approximately 46% before leveling off to a maximum of 47%. These are typical curves for this type of 2D mesh network.

Figure 6 shows the average measured packet latency in

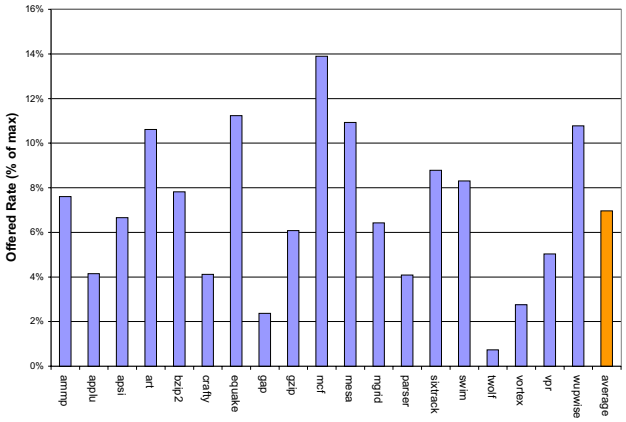

Fig. 7. Average offered rates in SPEC CPU2000 benchmark traces.

cycles for increasing offered rate. The average latency for bit-reversal traffic gradually increases from around 5 to 8 cycles for offered rates of 1% to 32%. The latency then increases exponentially as the network becomes saturated. Similarly for random traffic the latency increases from about 4 to 7 cycles for offered rates from 1% to 40% before increasing dramatically. This diagram shows that 32% and 40% are the saturation offered rates for bit-reversal and random traffic respectively.

C. OPN Traffic Trace Analysis

Our earlier work examining the OCN [22] showed that real benchmark generated traffic in on-chip networks would not be modeled well by traditional synthetic loads. We perform a similar analysis for the OPN using network traces generated from tsim-cyc and characterize the network workload.

Variation in application offered rate: Figure 7 shows the average offered rate for various SPEC CPU2000 benchmark traces generated from tsim-cyc. For each application's trace, we derived the offered rate by dividing the number of total messages by the product of the cycle count and the number of injecting nodes (25 for the 5x5 network). While the offered rates vary widely, from under 1% for `twolf` to almost 14% for `mcf`, the average offered rate is well below the saturation threshold range of 30-40% for bit-reversal and uniform random. The magnitude of the offered rates correlate to the degree of ILP that the TRIPS compiler has exposed to the processor. A benchmark with more exposed ILP will have more operations occurring simultaneously and will therefore generate more operands each cycle than a benchmark that has long dependency chains and lower ILP.

Average packet hop distance: Figure 8 shows the average number of hops, or router traversals, from source to destination for OPN packets from various SPEC CPU2000 benchmark traces. These values are generated by averaging the Manhattan distance from source to destination for each packet in the trace of each benchmark. Because the OPN is a single cycle per hop network, this distance also represents a best case routing delay for each packet in the absence of any network contention.

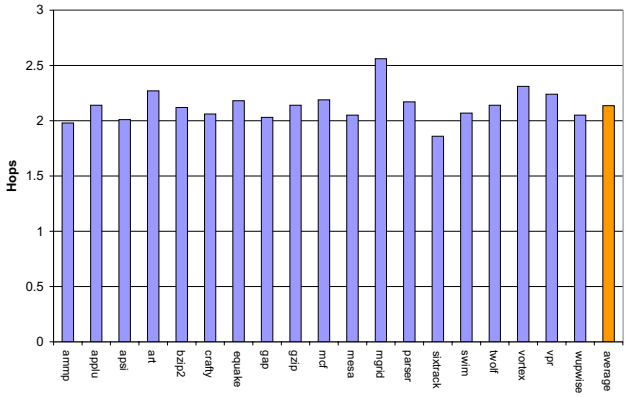

Fig. 8. Average number of hops from source to destination for various SPEC CPU2000 benchmarks.

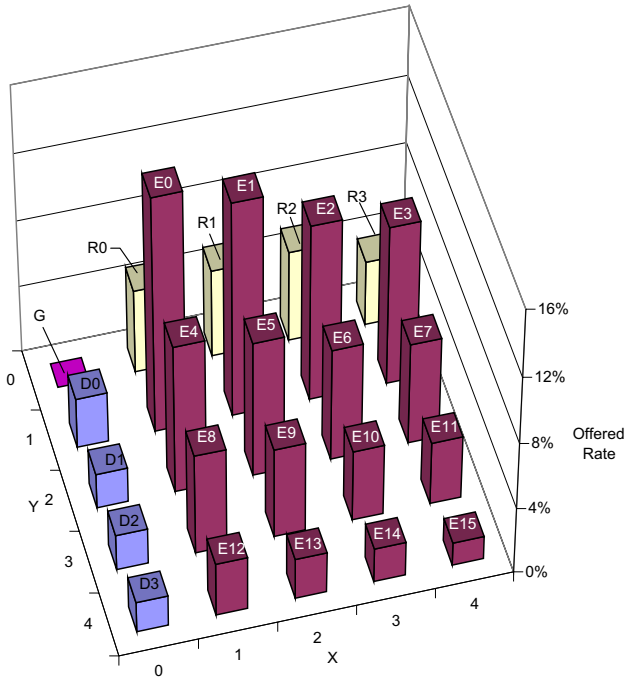

Fig. 9. Offered rates for SPEC CPU2000 benchmarks broken down by sources.

The figure shows that average hop distance for all benchmarks is 2.13. There is little variance from one benchmark to the next. In TRIPS, instructions are statically mapped on particular nodes; the TRIPS compiler tries to map instructions as close as possible to the source of their operands, be that the register file or data cache for register reads or memory loads, or other E tiles producing their operands directly. Offsetting this, the compiler must ensure that the instructions are evenly distributed across the execution resources to ensure minimal resource contention.

Variation in offered rate by source: While the overall average offered rate of the OPN is low, that metric does not accurately capture hot spots in the network. Figure 9 shows the average offered rate for each individual OPN node as a percentage of the peak offered rate of one message per cycle,

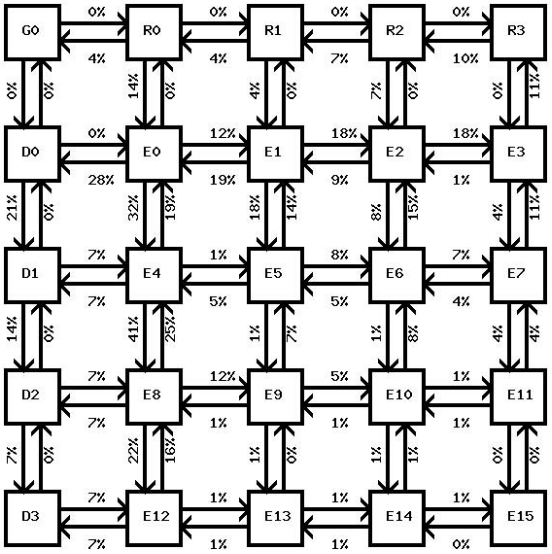

Fig. 10. Link utilization for `mesa` SPEC CPU2000 benchmark.

averaged across all SPEC CPU2000 benchmarks. The X-Y plane of the graph matches the layout of the 5x5 operand network and the different shades highlight the different tile types. The per-tile offered rates vary widely, from a low of 2.6% for E15 in the lower right to a high of 16.7% for E0 at the upper left. The disposition of offered rates reflects the TRIPS compiler's instruction placement optimizations that attempt to minimize operand routing distance. Thus, instructions are preferentially placed near the register file and data cache tiles to reduce block input and output latency. Our analysis shows that even though average offered rate is low, applications can easily create network injection hot spots that may approach local saturation, producing higher than expected transmission latencies. The compiler schedules instructions to more evenly distribute the network traffic; however such optimizations must be balanced against the effect of increasing the average source to destination hop count.

Variation in link utilization: Hot spots also form when many messages must pass through the same link. Figure 10 labels each OPN link with the link occupancy percentage for the `mesa` SPEC CPU2000 benchmark. We choose to show the data for one benchmark instead of averaging across all of them because of the variance across the benchmarks. The southbound link between E4 and E8 has a high utilization of 41% and many other links are in the 15%–20% range. High link utilizations will have a disproportionately large effect on latency because of congestion and limits the performance improvement available through virtual channel flow control. Our experience shows that other benchmarks place a maximum load of only 5-10% on any link.

Traffic burstiness: In addition to load variability across applications and network nodes, offered traffic can vary over

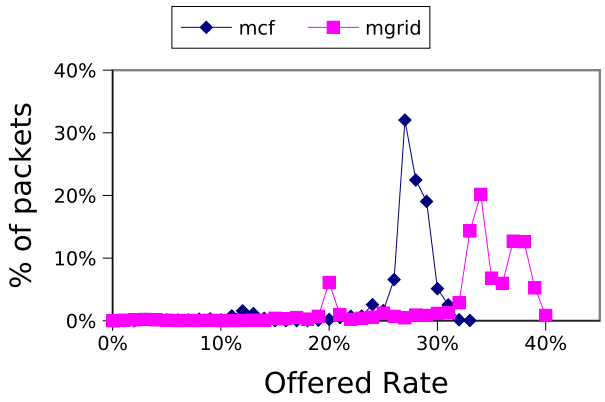

Fig. 11. Distribution of offered rates measured as a percentage of packets injected at a given offered rate.

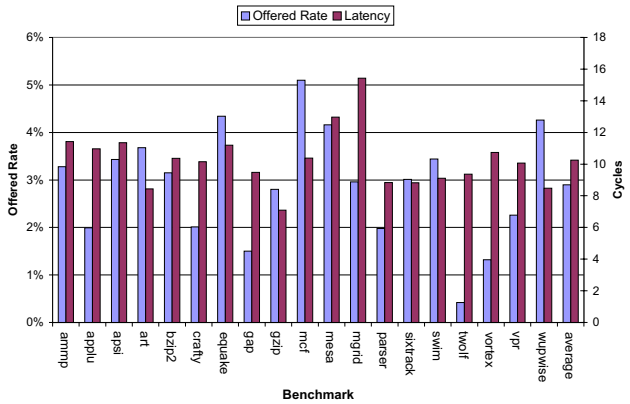

Fig. 12. Offered rates and latencies for SPEC CPU2000 benchmarks from the OPN network simulator.

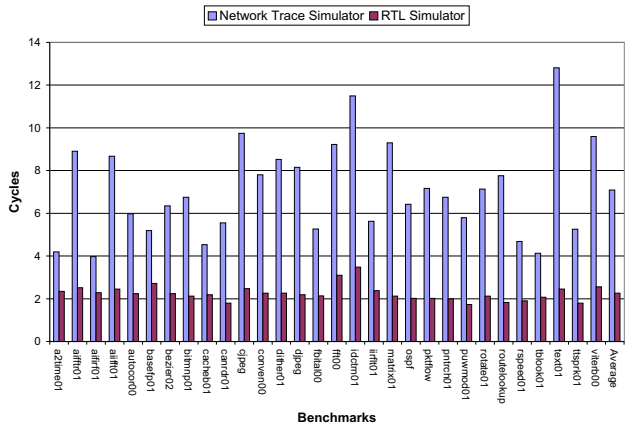

Fig. 13. EEMBC benchmark latency from the high-level tsim-cyc traces versus the detailed processor model tsim-proc.

time. TRIPS naturally has traffic bursts because a block begins execution through the injection of many register values from the top of the network. To measure this burstiness, we examined the trace at 1000 cycle intervals, counted the number of messages in each interval, and computed the offered rate for the interval. Figure 11 shows a histogram of the offered rates for two SPEC CPU2000 benchmarks, `mcf` and `mgrid`. The X-axis shows the histogram buckets at 1% intervals, while the Y-axis shows the fraction of all messages that fall into each bucket. The figure shows that `mcf` has a relatively stable offered rate centering around 29% for most packets, meaning that the network is evenly loaded over time. Conversely, `mgrid` shows more diversity in its offered rates with significant numbers of packets clustered around 20%, 34% and 38%. Based on these results we conclude that the traffic of `mgrid` has more bursts than that of `mcf`, and likely has spikes in latency for critical operands traversing the network. Such bursts may motivate lightweight network designs that tolerate and spread traffic in response to varying loads.

D. Network Simulator-based Analysis

To examine how the network performs under load, we applied the traces to the OPN trace-driven simulator. The inherent weakness of trace-driven network simulation is the lack of a feedback loop between the network simulation and trace generation. In the real TRIPS processor, network congestion will throttle instruction execution, in turn throttling the offered rate. To bound this difference, for each message we tracked the instruction block to which it belongs and ensures that messages from only eight consecutive blocks are considered for injection at any one time. These eight correspond to the one non-speculative and seven speculative blocks that can execute simultaneously. This approach represents a reasonable compromise that keeps the processor and network simulators separate. The lack of intra-block throttling places some excess stress on the network, giving additional insight on the load if the network were ideal and non-contented.

Offered rate and latency: Figure 12 shows the average offered rate and latency for the SPEC CPU2000 on the OPN simulator. Compared to the results in Figure 7, the offered rates are significantly lower because of the block-level throttling. The benchmarks that had the highest offered rates show offered rates that are reduced by as much as two thirds.

The right bar for each benchmark shows the average message latency for each benchmark. In general, benchmarks with higher offered rates show higher average latencies, but certain benchmarks show the reverse. For example, `mcf` has the highest offered rate at 5.1% while it has a fairly average latency of about 10 cycles. Conversely `mgrid` has a fairly average offered rate of around 3% but the highest average latency at 15.5 cycles. This dichotomy can be attributed to the burstiness in the traffic and high utilization of particularly hot links. The OPN simulator shows that the average latencies are high, ranging from 6 to 15 cycles. Although throttling will prevent actual OPN latencies from reaching these levels, the measured latencies highlight where OPN network performance improvement has a direct affect on processor performance.

Network throttling: To examine the impact of throttling on latency, we used the cycle-level simulator tsim-proc. Because tsim-proc is approximately 300 times slower than the tsim-

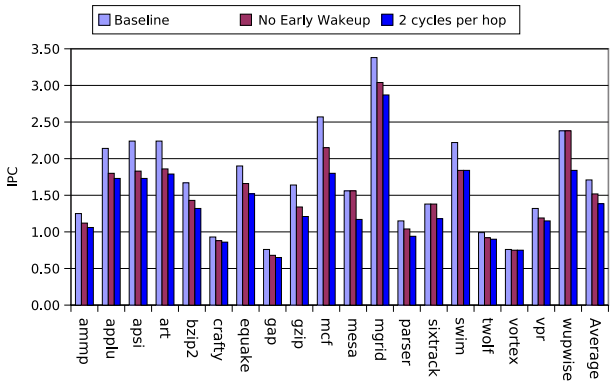

Fig. 14. Comparison of baseline OPN versus an OPN without the early wakeup and an OPN that consumes two cycles per hop.

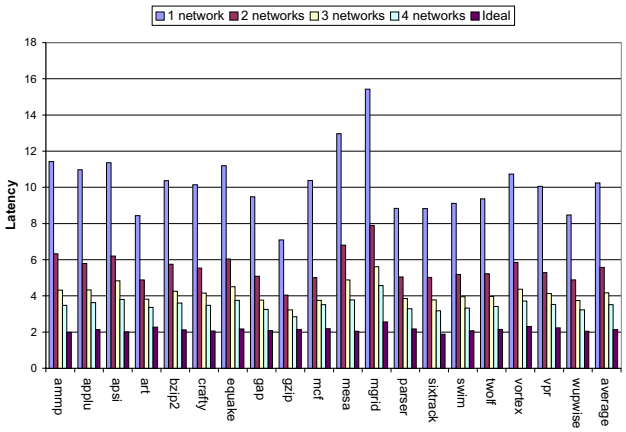

Fig. 15. Average packet latency for SPEC CPU2000 benchmarks with 1, 2, 3 and 4 OPN networks.

cyc simulator we used to generate traces, we chose the shorter EEMBC 2.0 suite of embedded system benchmarks. Figure 13 shows the average latencies of OPN packets as measured in both tsim-proc and the OPN network trace driven simulator. While the network simulator shows an average latency of 7 cycles, tsim-proc shows only 2.25 cycle, again due to throttling from instruction dependences in the program. This can be a little deceiving because throttling manifests as stalls in the execution tiles (ETs) rather than in the network. Thus for network research, trace-based simulation still provides good insight into network behavior, but one must take care when analyzing system performance based on network performance.

E. Operand Network Sensitivity Studies

Packet End-To-End Latency: The TRIPS prototype is designed to support one-cycle communication latency between adjacent ETs. Speculative injection of the operand message header, early wakeup of the consumer, and bypassing directly from the network input limit the latency of operand network transmission. Each additional hop in the network costs only one cycle. To examine the sensitivity of performance to latency, we simulated two alternate designs. The first emulates an architecture that does not have early wakeup and thus requires one additional cycle for every operand transmission. The second emulates a two-cycle-per-hop network to model slower routers and wires. Figure 14 compares the IPC (instructions-per-clock) of the TRIPS processor core for the different design points. Without early wakeup, processor performance drops by about 11%; a two-cycle per hop network decreases IPC by 20%. Thus performance of TRIPS is quite sensitive to OPN latency.

Bandwidth: A simple way to improve the performance of a network is to increase its bandwidth. Typically one would increase the bit-width of the network's interfaces to decrease the number of flits per message, network occupancy, and message injection and extraction latency. Because the OPN already has single-flit packets, increasing the link-width will not affect network occupancy or processor performance. Another way to improve the performance of a network is to decrease the network diameter by using higher-radix routers and a more highly interconnected topology. This approach is not a good fit for the OPN for two reasons. First, increasing the radix of the routers increases the logical complexity of the routers, possibly to the point of becoming the TRIPS core's critical timing path. Second, as shown in figure 8 the average hop distance for packets on the OPN is just over 2, so increasing the network's order would not decrease the end-to-end latency of a large fraction of the injected messages. Slowing down the clock or pipelining the routers in order to achieve timing would mitigate any gains.

As an alternative, we investigated replicating the network links and routers as a means to increase the effective bandwidth of the network and reduce contention. We simulate a simple scheme in which nodes inject packets into each network in a round-robin fashion. If a network is blocked due to congestion, the injecting node skips it until the congestion is alleviated. Figure 15 shows the average packet latency from the OPN trace simulator for the SPEC CPU2000 benchmarks with 1 (the current OPN configuration), 2, 3 and 4 networks interconnecting the nodes of the OPN. The expected latency without contention is shown as "Ideal". The biggest improvement in latency occurs between 1 and 2 networks, almost halving the average latency. However, replication comes at a cost of doubling the area consumed by the network.

VI. Conclusions and Future Work

In this paper, we presented the design, implementation, and evaluation of the TRIPS OPN. The TRIPS OPN is a micronet that interconnects the functional units within the TRIPS processor core. The OPN replaces an operand bypass bus and primary memory system interconnect in a technology scalable manner. The tight integration between the OPN and the processor core elements enables fast operand bypassing across distributed ALUs, providing opportunity for greater instruction-level concurrency. Our implementation and fabrication shows that such a network is feasible in terms of area and delay, and that the network design provides good performance for the traffic provided by real applications.

We used synthetic benchmarks along with static traces generated from SPEC CPU2000 traffic to evaluate the performance of the OPN micronet under different loads. We found that the offered traffic varied widely across multiple applications and across different processor tiles; stochastic workloads are not representative of such real workloads. Our experiments confirm the expectation that distributed processor performance is quite sensitive to network latency, as just one additional cycle per hop results in a 20% drop in performance. Increasing the link width in bits does not help this network since the messages already consist of only one flit. Replicating the network to improve bandwidth and reduce latency is promising as increasing the wire count in on-chip networks is not prohibitively expensive. However, network router area (particularly router buffers) is not insignificant and these costs must be balanced with network performance benefits.

We expect that fine-grained networks will increase in importance, initially as memory oriented networks for chip multiprocessors and SoCs, but ultimately in support of finer grained communication and synchronization. Further research is needed to re-examine standard multichip interconnection network architectures with respect to the constraints and opportunities of on-chip networks. In addition to network latency and area, we expect network power, efficiency, and quality of service to be critical. We also expect micronets to provide new opportunities in other aspects of distributed system and processor design. As an example, we are currently examining how micronet flow control can help reduce area overheads of distributed memory ordering hardware.

REFERENCES

[1] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock Rate Versus IPC: The End of the Road for Conventional Microarchitectures," in *27th International Symposium on Computer Architecture (ISCA)*, 2000, pp. 248–259.

[2] E. Sprangle and D. Carmean, "Increasing Processor Performance by Implementing Deeper Pipelines," in *30th International Symposium on Computer Architecture (ISCA)*, 2002, pp. 25–34.

[3] R. Tomasulo, "An Efficient Algorithm for Exploring Multiple Arithmetic Units," *IBM Journal of Research and Development*, vol. 11, no. 1, pp. 25–33, Jan. 1967.

[4] S. E. Breach, T. N. Vijaykumar, and G. S. Sohi, "The Anatomy of the Register File in a Multiscalar Processor," in *27th ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 1994, pp. 181–190.

[5] R. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24–36, 1999.

[6] S. W. Keckler, W. J. Dally, D. Maskit, N. P. Carter, A. Chang, and W. S. Lee, "Exploiting Fine-grain Thread Level Parallelism on the MIT Multi-ALU Processor," in *25th International Symposium on Computer Architecture (ISCA)*, 1998, pp. 306–317.

[7] 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 Computer*, vol. 30, no. 9, pp. 86–93, September 1997.

[8] C. F. Joerg and G. A. Boughton, "The Monsoon Interconnection Network," in *IEEE International Conference on Computer Design (ICCD)*, 1991, pp. 156–159.

[9] S. Swanson, A. Putnam, M. Mercaldi, K. Michelson, A. Petersen, A. Schwerin, M. Oskin, and S. Eggers, "Area-Performance Trade-offs in Tiled Dataflow Architectures," in *33rd International Symposium on Computer Architecture (ISCA)*, 2006, pp. 314–326.

[10] M. B. Taylor, W. Lee, S. P. Amarasinghe, and A. Agarwal, "Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architecture," in *9th International Symposium on High-Performance Computer Architecture (HPCA)*, 2003, pp. 341–353.

[11] K. Sankaralingam, V. A. Singh, S. W. Keckler, and D. Burger, "Routed Inter-ALU Networks for ILP Scalability and Performance," in *IEEE International Conference on Computer Design (ICCD)*, 2003, pp. 170–177.

[12] T. M. Pinkston and J. Shin, "Trends toward on-chip networked microsystems," *Int. J. High Performance Computing and Networking*, vol. 3, no. 1, pp. 3–18, 2005.

[13] "International technology roadmap for semiconductors (ITRS), 2003 edition." [Online]. Available: http://public.itrs.net/Files/2003ITRS/Home2003.htm

[14] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *38th Design Automation Conference (DAC)*, 2001, pp. 684–689.

[15] S. Kumar, A. Jantsch, M. Millberg, J. Öberg, J.-P. Soininen, M. Forsell, K. Tiensyrjä, and A. Hemani, "A Network on Chip Architecture and Design Methodology," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2002, pp. 117–124.

[16] D. Burger, S. W. Keckler, K. S. McKinley, M. Dahlin, L. K. John, C. Lin, C. R. Moore, J. Burrill, R. G. McDonald, W. Yoder, and the TRIPS Team, "Scaling to the End of Silicon with EDGE Architectures," *IEEE Computer*, vol. 37, no. 7, pp. 44–55, July 2004.

[17] K. Sankaralingam, R. Nagarajan, P. Gratz, R. Desikan, D. Gulati, H. Hanson, C. Kim, H. Liu, N. Ranganathan, S. Sethumadhavan, S. Sharif, P. Shivakumar, W. Yoder, R. McDonald, S. Keckler, and D. Burger, "The Distributed Microarchitecture of the TRIPS Prototype Processor," in *39th ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2006.

[18] L.-S. Peh and W. J. Dally, "Flit-Reservation Flow Control," in *6th International Symposium on High-Performance Computer Architecture (HPCA)*, 2000, pp. 73–84.

[19] A. R. Weiss, "The Standardization of Embedded Benchmarking: Pitfalls and Opportunities," in *IEEE International Conference on Computer Design (ICCD)*, 1999, pp. 492–498.

[20] J. L. Henning, "SPEC CPU2000: Measuring CPU Performance in the New Millennium," *IEEE Computer*, vol. 33, no. 7, pp. 28–35, 2000.

[21] A. J. KleinOsowski and D. J. Lilja, "MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research," *Computer Architecture Letters*, vol. 1, 2002.

[22] P. Gratz, C. Kim, R. McDonald, S. W. Keckler, and D. Burger, "Implementation and Evaluation of On-Chip Network Architectures," in *IEEE International Conference on Computer Design (ICCD)*, 2006.