2011 19th Annual IEEE Symposium on High Performance Interconnects

Express Virtual Channels with Taps (EVC-T): A Flow Control Technique for Network-on-Chip (NoC) in Manycore Systems

Chao Chen, Jie Meng, Ayse K. Coskun and Ajay Joshi
Electrical and Computer Engineering Department, Boston University, 8 Saint Mary's Street, Boston, MA
{chen9810, jiemeng, acoskun, joshi}@bu.edu

***Abstract*—Manycore systems require energy-efficient on-chip networks that provide high throughput and low latency. The performance of these on-chip networks affects cache access latency and, consequently, system performance. This paper proposes solutions to address the performance limitations related to the use of snoop-based cache coherence protocol on switched network-on-chip (NoC). We propose a new network flow control technique, Express Virtual Channel with Taps (EVC-T), for transmitting both broadcast packets and data packets efficiently. In addition, we propose a low-latency broadcast packet notification tree network that maintains the order of broadcast packets on an unordered NoC. We evaluate our technique using both synthetic traffic and parallel benchmark suites through detailed system simulation. EVC-T reduces the average network latency by 24% with a negligible change in power for synthetic benchmarks. For NAS parallel applications, EVC-T increases the instructions per cycle (IPC) by 9% on average with minimal increase in power. Our technique reduces the energy-delay product (EDP) by 13% on average across all benchmarks.**

***Keywords*-Express virtual channel with taps; Manycore systems; NoC; Simulation; Snoop based cache coherence.**

I. Introduction

Future manycore systems are expected to have hundreds of cores integrated on a single chip. These cores communicate with each other through a network-on-chip (NoC). NoC traffic includes transfer of cache lines and associated control information among various levels of caches and memory controllers. The wide distribution of NoC latency, however, affects the performance of the applications running on the manycore system. Similarly, power dissipated on the NoC is an important concern given the power-limited nature of current and future manycore systems.

The general trend for NoC architectures is towards designing low-radix high-diameter network topologies (e.g., mesh) that have short router-to-router channels [1], [2]. These topologies are easier to design from the hardware perspective. However, mapping an application to these topologies is difficult due to the large variance in packet latencies. High-radix low-diameter topologies such as crossbar are more amenable to application mapping due to low network diameters, but are difficult to design from the hardware perspective because of the long wires.

Another issue with NoC-based manycore systems is maintaining cache coherency across multiple caches. Previously, snoop-based and directory-based cache coherency protocols have been investigated for manycore systems [3]. In this paper, we focus on the snoop-based protocol. Snoop-based cache architecture uses a broadcasting mechanism for cache coherency, and is commonly used for bus-based topology in systems with a small number of cores (e.g., fewer than 10 cores). This protocol, however, does not scale well for NoC-based manycore systems due to the packet latency distribution.

To harness the true potential of manycore systems we need to develop low-cost, high-performance and energy-efficient NoC architectures. This paper makes two contributions towards achieving this goal. First, we propose a new broadcasting mechanism for snoop-based cache coherency protocol for manycore systems. Each core uses a dedicated notification tree to rapidly inform all other cores of an incoming broadcast message. In this way, other cores can make early decisions to wait for the packet or to proceed with execution in presence of simultaneously transmitted packets over the shared network. The proposed broadcast mechanism has higher performance in comparison to conventional snoop-based broadcast mechanisms for low network traffic. For high network traffic, the performance of the proposed approach is similar to the conventional approaches.

We also propose a novel network flow-control mechanism: Express Virtual Channels with Taps (EVC-T). Our flow-control mechanism, when mapped to a physical concentrated mesh (cmesh) network, results in a logical topology with high radix and low diameter. As a result, this NoC is easy to design from the hardware perspective and easy to program. The logical topology is similar to multidrop express channel (MECS) [4]. However, unlike MECS, the proposed logical topology supports both data and broadcast packets, and does not use physically separated express channels. Our specific contributions are as follows:

- To maintain the sequential consistency for snoop-based cache coherence, we propose using a contention-free notification tree per core as a supporting network for broadcasting. These notification trees ensure each core knows when to expect a broadcast packet and the exact timestamp when a broadcast packet was generated. This information guarantees that on average a core has to wait for less than a cycle before it can make the decision about processing a broadcast packet and all the broadcast packets are processed in the correct order.
- To achieve low network latency and save energy, we propose EVC-T flow control mechanism to transmit both broadcast and data packets in a snoop-based cache co-

herency protocol. For broadcasting, our EVC-T technique allows intermediate routers to receive and store broadcast packets while forwarding them to downstream routers simultaneously. For NAS parallel benchmarks [5], EVC-T reduces the average packet latency (data and broadcast) by 24% and improves the system energy efficiency, reducing the energy delay product (EDP) by 13% on average.

- To evaluate the impact of our proposed EVC-T flow control mechanism and broadcasting technique, we integrate a NoC simulator, BookSim [6], with an architecture-level performance simulator M5 [7]. The primary motivation is that the operations of the NoC and the rest of the system have strong correlations with each other, and hence adopting an integrated evaluation process is necessary.

The rest of the paper starts with a discussion of the related work. Section III provides the details of our target system. Section IV explains the use of notification trees as supporting networks for cache coherence. Section V describes the EVC-T flow control technique. Section VI evaluates our techniques using synthetic traffic and the NAS parallel benchmarks, and Section VII concludes the paper.

II. Related Work

A number of techniques have been proposed for maintaining cache coherence in manycore systems with NoCs. Ordered broadcast trees and ring topologies address the cache coherence problem on packet switch NoCs by creating ordering points [8], [9]. Although the ordering points method in these techniques is convenient and straightforward, the technique increases packet latency. A similar approach has been proposed in [10], where a ring cache coherence protocol is used for ordering. In this case, in addition to the snoop request broadcast, the requester also initiates a response message that collects responses from all nodes as it travels around the ring. A global ordering of networks has been proposed using isotach-like networks in [11]–[13]. To maintain the orders of broadcast packets, some approaches use snoop ordering [14]. This method avoids using the ordering points. However, the received broadcast packets have to wait for other packets with lower snoop orders, which results in large waiting time overhead. These techniques affects system performance.

Today's commercial systems such as Tilera [2] and Intel Single-Chip Cloud (SCC) [1] use low-radix high-diameter networks such as mesh due to their ease of hardware design. Several high-radix and low-diameter network topologies, such as flattened butterfly, clos, and MECS [4], [15], [16] have been proposed for lower network latency distribution. These network topologies provide low network latency distributions and high network throughput by connecting distant routers with physical express channels. However, the hardware and energy overheads of physical express channels make it difficult to justify their use for current and future power-limited systems. To improve the performance of low-radix and high-diameter network topologies, express virtual channels (EVC) and a corresponding flow control technique have been proposed in [17]. This technique enables the intermediate routers to forward received packets immediately using buffering, arbitration, and crossbar switching. However, EVC is not efficient for transmitting broadcast packets due to the multiple transmissions of one broadcast packet on the same physical channel.

We propose a broadcasting technique with notification trees as the supporting networks for cache coherence. Our broadcasting technique allows caches to process received broadcast packets with much shorter waiting time and limited hardware overhead. To support effective transmission of broadcast-packets, we extend the traditional EVC technique to EVC-T that has multiple taps along the EVC. This approach transmits both broadcast packets and data packets with reduced traffic overhead and network latencies.

III. Target System

We choose a 64-core processor as our target system that is manufactured using 22 nm technology process as a representative node for future manycore chips [18]. Each core on the processor supports 2-way issue out-of-order execution, and has two integer ALU, one integer multiplication unit, one floating-point ALU, and one floating-point multiplication unit. The core architecture is configured based on the cores used in Intel's 48-core SCC [1]. The micro-architectural parameters are listed in Table I. The cores operate at 1 GHz frequency and have a supply voltage of 0.9 V, while the on-chip network operates at 2 GHz.

Each core has 16 KB private L1 instruction cache and 16 KB private L1 data cache. We use a shared memory programming model and explore a distributed L2 cache architecture. The manycore system uses the snoop-based MESI protocol for maintaining the cache coherence.

Table I: Micro-architectual parameters of the target system.

Micro-architecture Configuration	
Core Frequency	1.0 GHz
Branch Predictor	Tournament predictor
Issue	2-way Out-of-order
Reorder Buffer	128 entries
Functional Units	2 IntAlu, 1 IntMult 1 FPALU, 1 FPMult
Physical Regs	128 Int, 128 FP
Instruction Queue	64 entries
L1 ICache	16 KB @ 2 ns
L1 DCache	16 KB @ 2 ns
L2 Cache	4-way set-associative, 64 B block Distributed 16 x 1 MB @ 6 ns
NoC Frequency	2.0 GHz

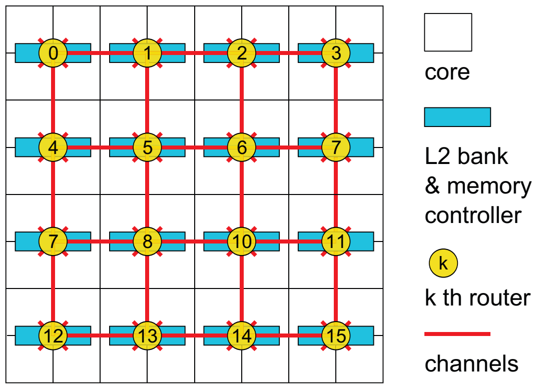

Figure 1: Physical layout of our 64-core target system. 16 L2 cache banks (1 MB each) are uniformly distributed across the chip. Each L2 bank has one memory controller physically located next to it. Each router uses a concentration of 4 cores and one L2 cache bank. It has 13 input and 13 output ports (4 for inter-router interconnect, 8 for L1 I-cache and D-caches of 4 cores, and 1 for 1 L2 cache bank).

Figure 1 shows the physical layout of our 64-core target system. There are 64 cores, 16 L2 cache banks (1 MB each), 16 memory controllers that are uniformly distributed across the chip. Four cores and one L2 bank share one router and communicate with other cores and L2 banks through the on-chip network. Each memory controller is associated with one L2 bank and there is a dedicated channel between them, which is not shown in Figure 1. Each router has four-cycle zero-load latency for the four pipelined routing stages: route computation, virtual channel allocation, switch allocation, and switch traversal [19]. After energy optimization by repeater insertion, each 5 mm channel between two neighboring routers has single-cycle latency.

IV. Notification Trees for Broadcasting Order

For network topologies such as cmesh, clos or crossbar, where the network enables parallel accesses, multiple sources can insert packets into the network at the same time. The latency of each packet varies based on the traffic workload as well as the physical location of its source L1 cache. Therefore, a destination L1 cache can potentially receive broadcast packets in a different order than the original order in which the broadcast packets were generated. Hence, a destination L1 cache needs to wait for all broadcast packets that are on the fly before processing the received broadcast packet. The worst-case waiting time can be determined using empirical methods. However, depending on the size of the manycore system, the waiting period could be considerably long, which has a negative impact on the system performance. In our 64-core target system with a cmesh network, the cores at the corners have to wait for more than 34 cycles assuming the shared cmesh network has zero-load latency. As the network traffic becomes high, the waiting time increases because of network contention. When an L1 cache knows in advance how many cycles to wait for the broadcast packets that are already on-the-fly, we can avoid the wasted cycles at the cache.

We propose using notification trees along with the existing shared network to maintain the sequential consistency of broadcast packets that are transmitted on unordered interconnects. Each L1 cache in a core has a dedicated notification tree, connecting it to all other L1 caches. Figure 2(a) and 2(b) show the notification trees from the L1 caches in core 0 and core 18, respectively. Each notification tree is pipelined and each pipeline segment uses energy-optimized repeater-inserted single-bit wire.

The notification tree for an L1 cache sends a notification pulse to all other destination L1 caches whenever it has a read/write cache miss, and a new broadcast packet requesting the missing cache line is generated. The actual broadcast packets are transmitted over the shared network. As the notification trees are contention free, the notification pulses reach destination L1 caches within fixed latencies. Thus, each destination L1 cache can determine the exact waiting time before other received broadcast packet get processed. The waiting time of any broadcast packet is calculated by subtracting the broadcast packet transmission time from the notification transmission time for the farthest network source. The maximum waiting time in a destination L1 cache depends on its physical location relative to other L1 caches.

Figure 2(c) shows an example for two broadcast packets: packet A and packet B from the L1 caches in core 0 and core 63, respectively. Packet A reaches the L1 cache of core 1 after traveling through one router, resulting in a zero-load latency of 4 cycles. Packet B reaches the L1 cache in core 1 after traveling through 7 routers and 6 channels, resulting in a zero-load latency of 34 cycles. On the other hand, the notification pulses for both packets reach their destinations much faster. The notification pulse for packet A reaches the L1 cache in core 1 right after packet A is generated and the notification pulse for packet B reaches the L1 cache in core 1 in 6 cycles after traveling through 6 wire segments on the notification tree.

Table II: Timing analysis of the proposed notification tree architecture for managing broadcasting for snoop-based cache coherency in our 64-core target system. Notification arrival time corresponds to the latency of a notification pulse through the dedicated notification tree. Packet arrival time corresponds to the zero-load latency of a broadcast packet traveling through the shared cmesh network.

Packet ID	Source Core	Send Time	Dest. Core	Notification Arrival Time	Packet Arrival Time
A	0	T	1	T	T + 4
B	63	T - 1	1	T + 5	T + 33

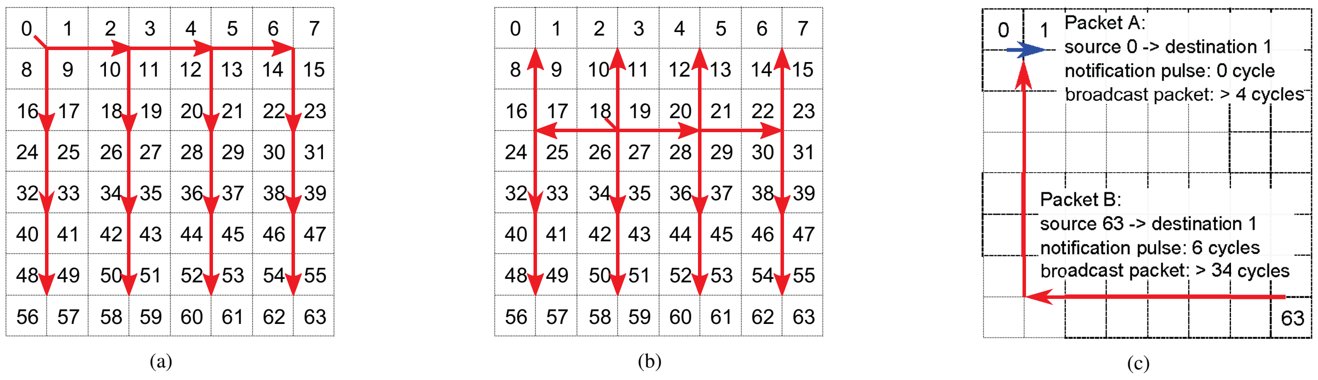
(a)
(b)
(c)
Packet A:
source 0 -> destination 1
notification pulse: 0 cycle
broadcast packet: > 4 cycles
Packet B:
source 63 -> destination 1
notification pulse: 6 cycles
broadcast packet: > 34 cycles

Figure 2: (a) Notification tree from core 0. (b) Notification tree from core 18. (c) Example of broadcast packets from core 0 and core 63. They are broadcast to all receivers across the chip, but only one receiver is shown in the figure.

In our analysis, packet A is generated at time 'T', while packet B is generated at time 'T-1'. The notification pulses for packet A and B reach the L1 cache in core 1 at time 'T' and 'T+5', respectively. To maintain cache coherency, packet B has to be processed before packet A. After the L1 cache for core 1 receives the broadcast packet A at 'T+4', it monitors the notification trees for the remaining L1 caches for an additional cycle. The 1-cycle waiting time is calculated based on the fact that the core 63 is farthest away from core 1, and it takes 6 cycles for a notification pulse to be transmitted from the L1 cache in core 63 to the L1 cache in core 1. As the L1 cache in core 1 receives the notification for packet B at 'T+5', it can formulate the correct order for processing the packets. On average, the L1 cache in a core has to monitor the notification trees for less than a cycle before it can decide on the broadcast packet processing order. The exact time when a broadcast packet is processed depends on the network latency of the broadcast packet. Table II shows the timing analysis of these two broadcast packets at their sources (core 0 and core 63) and at the destination (core 1) as an example.

The overall hardware overhead for the proposed broadcast technique includes 64 notification trees, buffers in each core to store incoming notifications, and combinational logic in each core to decide on the processing order. The area overhead of 64 notification trees is 31% with respect to the wiring area of the existing shared network and the power overhead is 0.27 W. The area and power overheads for buffers and combinational logic are negligible. The total hardware overhead can be reduced by 4× using a shared architecture, where a group of 4 cores shares a notification tree.

V. EVC-T Flow Control Mechanism

In this section, we propose a novel flow control mechanism that, when used with a low-radix high-diameter physical network, provides a high-radix low-diameter logical topology. This reduces the network latency for broadcast and data packets, and therefore improves the manycore system performance. We introduce the traditional EVC flow control mechanism as the background at first, and then describe our proposed EVC-T flow control mechanism and its application in our target system.

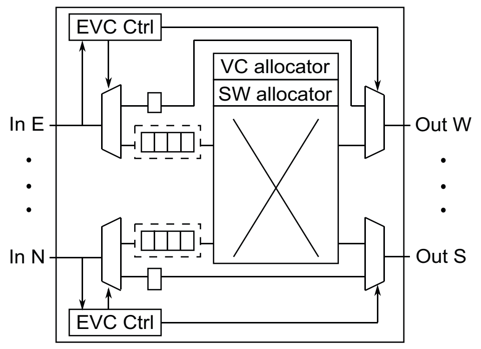

Figure 3: The router architecture for EVC [17]. A packet received at an intermediate router of an EVC is immediately forwarded to the subsequent physical channel of the same EVC. A packet received at a destination router of an EVC is buffered and switched to channels in another direction.

A. Express Virtual Channels (EVC)

The EVC flow control mechanism proposed in [17] enables packets to entirely bypass routers. Figure 3 shows the router architecture supporting EVCs. Each router receives four types of packets – a packet generated by a core connected to that router, a packet that will bypass the router, a packet that will change direction (X $\rightarrow$ Y) in the router, and a packet that has one of the attached core as its destination. The EVC controllers at the input ports differentiate between the packets that will be buffered by the router (change direction, get ejected to, or are injected from an attached core) and those that will bypass the router. The packets that bypass the router get priority to access the downstream inter-router physical channel among all packets.

At low network traffic, when using EVCs, a data packet

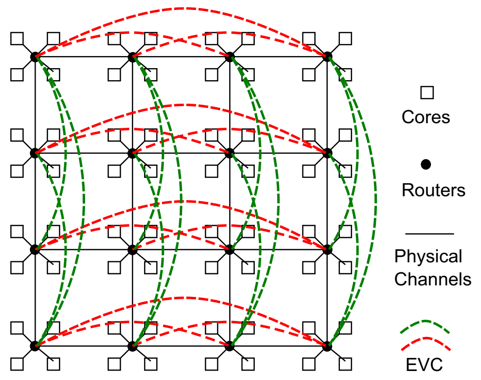

Figure 4: The C-EVC network: physical cmesh layout with EVC. EVCs use the existing physical channels to establish express paths between each pair of two distant routers in both X and Y dimensions. The resulting logical topology is effectively a flattered butterfly.

effectively passes through a series of inter-router channels as it bypasses all intermediate routers. As a result, it has lower latency compared to the equivalent networks with no EVCs. On the other hand, at high network traffic, the latency and saturation throughput are comparable to equivalent networks with no EVCs. Figure 4 shows an example network, where the network has the same physical layout as cmesh topology; i.e, it connects the neighboring routers with short physical channels. By connecting distant routers with EVCs, this network approaches the low zero-load latency of physical flattened butterfly topology while maintaining the low energy cost benefits of the traditional cmesh topology.

The limitation of EVC is that they cannot transmit broadcast packets as efficiently as data packets. On a traditional cmesh network, broadcast packets are transmitted through multiple hops and at each hop each broadcast packet is duplicated and transmitted in both X and dimensions. Here, a broadcast packet is transmitted only once through any physical channel. However, if the EVC flow control mechanism is used, a broadcast packet is transmitted through multiple EVCs from one source router to several destination routers, resulting in multiple transmissions through one physical channel shared by those EVCs. This will increase the network congestion, which increases the packet latencies and negatively affects the manycore system performance. We propose an upgraded EVC-T flow control mechanism that transmits both broadcast and data packets with low latency and power consumption on a single shared network.

B. *Express Virtual Channels with Taps (EVC-T)*

The EVC-T maintains the key feature of EVC: reducing packet latency through router bypassing. The difference is that EVC-T establishes an express virtual path from one source router to multiple receiver routers. For data transmission, only one target receiver router buffers the data packet, while other routers are bypassed. For broadcasting,

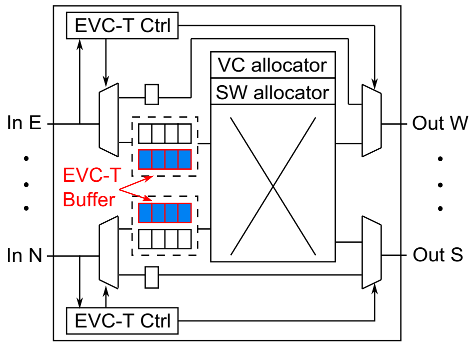

Figure 5: The router architecture supporting EVC-T. The received packets can be buffered in the router and forwarded to the subsequent channel simultaneously.

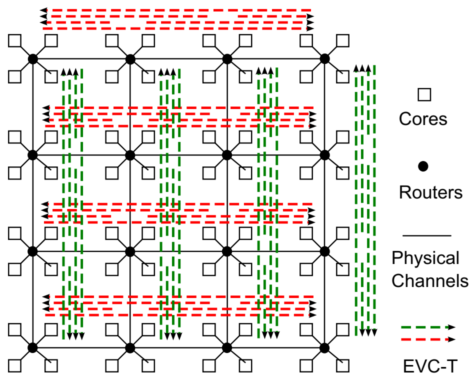

Figure 6: The C-EVC-T network: the physical cmesh layout with EVC-T. Single EVC-T connects one source router to all receiver routers in each direction (east, west, north, or south). The resulting logical topology is similar to MCES.

all receiver routers simultaneously buffer and forward the broadcast packet. The source router uses one EVC-T to transmit a broadcast packet to all receiver routers in each direction (east, west, north, or south). This ensures that a broadcast packet is transmitted only once through any physical channel.

Figure 5 shows the router architecture supporting EVC-T. Each EVC-T has input buffers at all receiver routers, including the intermediate routers and the destination router. Thus, an incoming broadcast packet can be simultaneously forwarded to the subsequent channel in the current direction, while buffered and switched to channels in other directions. The EVC-T controllers make decisions for buffering and/or forwarding the incoming packets. Similar to EVC, the packets that are bypassing the router will have priority access to the subsequent physical channel in comparison to other packets that are switching directions in the router.

Figure 6 shows an example network using EVC-T. Here we use the same router ID as labeled in Figure 1. On the top of physical cmesh layout, each router is connected to multiple receiver routers through a single EVC-T in

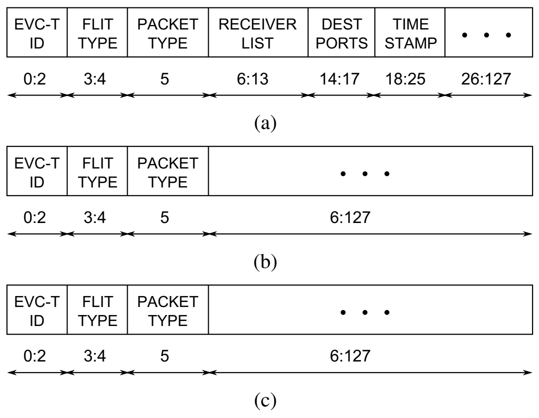

Figure 7: Flit organization of an EVC-T packet (a) Head flit (b) Body flit (c) Tail flit

.

each direction (east, west, north, or south). For example, a broadcast packet injected at router 0 is transmitted to router 1, 2, and 3 through a single EVC-T to the east. Each of these three routers buffer and transmit the broadcast packet to three more routers through its EVC-T to the south. The broadcast packet reaches all network destinations within two network hops – one in each dimension. For another example, a data packet from router 0 to router 14 is transmitted through the same EVC-T to the east used by the above broadcast packet. Router 2 buffers and transmits the data packet to router 14 through its EVC-T to the south.

Unlike the data packets, the broadcast packets usually request the EVC-Ts for multiple output ports at routers. For example, a broadcast packet injected at router 0 would request an EVC-T to the south and an EVC-T to the east at the same time. The routing delay and energy costs will be reduced if the VC allocator grants both EVC-Ts and the crossbar transmits the broadcast packet to both output ports simultaneously. We design our VC allocator based on the ISLIP routing scheduling algorithm [20]. We change the grant phase of ISLIP algorithm to grant one EVC-T for each requested output port. The SW allocator controls the switch crossbar and grants the switching paths between crossbar input ports and output ports. We change the traditional SW allocator to turn on several switches along a crossbar input wire, which enables one-to-many transmission through these crossbar switches. With our upgraded VC router, the broadcast packets have the same zero-load routing delay as the data packets. In the shared network, broadcast and data packets have the same priorities; i.e, a broadcast packet has to wait if another packet is using the requested output port.

C. *Flit orgnization for EVC-T*

Figure 7 shows the flit organization of an EVC-T packet. The EVC-T controller uses this information to decide between buffering and/or forwarding a packet. Because there are 6 EVC-Ts in each row or column, each EVC-T uses a unique 3-bit identification number (EVC-T ID). The EVC-T controller uses this identification number to store the packet in the appropriate input buffer of a router as each router maintains a dedicated input buffer for each incoming EVC-T. The 2-bit FLIT_TYPE identifies the type of the received flit: header flit (10), body flit (00), or tail flit (01). The EVC-T controllers make buffering and/or forwarding decisions based on the data in the header flit and keep the decisions for the following body flits and tail flits. The PACKET_TYPE identifies whether the received packet is a broadcast packet (1) or a data packet (0). The 8-bit RECEIVER_LIST (4-bits for each dimension) indicates the routers along the EVC-T path where the packet has to be buffered. For example, for a broadcast packet, RECEIVER_LIST has a value of 1111_1111. For a data packet, there is only one receiver router and two bits (one for each dimension) will be set in the RECEIVER_LIST field. For example, a data packet from router 0 to router 14 uses 0010_0001 to notify router 2 and router 14 to buffer the received data packet. The 4-bit DEST_PORTS indicates the ejection ports at the destination routers for data packets. In our target system, 9 ejection ports are used because 4 cores (4 L1 ICache caches and 4 L1 DCache) and 1 L2 bank share one router. For broadcast packets, the DEST_PORTS is not used because the receiver routers must forward broadcast packets to all ejection ports. The 10-bit TIME_STAMP is the packet generation time used for our proposed cache coherence technique in Section IV. The 10-bit length is designed to cover the time interval between notification arrival time and broadcast packet arrival time, which is variable due to the potential contentions in the shared network. When broadcast packets reach their destinations, they are matched with corresponding notification pulses by comparing the TIME_STAMP with the stored notification pulse generation time. The remaining bits in the packet are used for data.

D. *Credit channels for EVC-T*

For avoiding buffer overflow, a source router maintains the buffer status of all receiver routers of connected EVC-Ts. Each receiver router has a dedicated credit channel to the source router. For the C-EVC-T network in Figure 6, each pair of routers in the same row or column is connected with a pair of credit channels. The overhead of these credit channels are minimal because they are 1 to 2-bit wide. A source router grants an EVC-T after making sure that all target receiver routers have available credits. For example, a broadcast packet injected at router 0 is granted with an EVC-T to the east only when router 1, 2, 3 have available credits, and a data packet (from router 0 to router 14) is granted with the above EVC-T only when router 2 has at least one available credit. The source router deducts a credit for each target receiver router after sending every flit and increments a credit after receiving a new credit through a dedicated credit channel.

VI. Evaluation

In this section, we evaluate our proposed EVC-T technique using both synthetic traffic running on BookSim simulator, and NAS parallel benchmarks running on an integrated M5-BookSim full-system simulator. We compare four networks as shown in Table III. The first three networks have the same cmesh physical layout but different logical topologies. VC type is the flow control techniques used to build various logical topologies. VC count is the number of VC, EVC, or EVC-T channels used between one source router and any of its destination routers. The C-VC network uses three VCs to connect neighboring routers and has a cmesh logical topology. The C-EVC network uses a dedicated EVC to connect each pair of routers in both X and Y dimensions, as shown in Figure 4, and has a flattened butterfly (flatfly) logical topology. The C-EVC-T uses a single EVC-T to connect one router to multiple routers in each direction (east, west, north, or south), as shown in Figure 6, resulting in a logical topology similar to the MECS network proposed in [4]. The MECS network has a logical and physical MECS topology that uses a one-to-many communication model enabling a high degree of connectivity in a bandwidth-efficient manner. However, the logical MECS topology in the C-EVC-T network supports more efficient broadcast packet transmission and has less hardware overhead than the physical MECS network.

All four networks support both broadcast and data packets. The data packets in the C-EVC network and C-EVC-T reach the destination routers through virtual express paths, while the data packets in C-VC reach the destination routers through multiple hops. The broadcast packets in C-EVC-T reach multiple destination routers through a single virtual express path, while the broadcast packets in C-VC and C-EVC network reach destination routers through multiple hops. The C-EVC-T is the only network that uses virtual express paths to transmit both broadcast and data packets.

A. Evaluation Platform

We use BookSim network-on-chip simulator to evaluate synthetic network traffic patterns. To evaluate the impact of the proposed EVC-T technique on the whole manycore system, we integrate BookSim into M5 full-system simulator. M5 is an event-based manycore simulator that uses Alpha instruction set architecture (ISA), while BookSim is a cycle-precise network simulator. BookSim is integrated as a sub-module of M5, and we add a network interface for handling the communication between the two simulators. The packets generated by M5 are converted to the BookSim format and injected into the network instantiated by BookSim. The injected packets are preserved in a flying packet list in the network interface for tracking. M5 schedules events for checking the network output ports at every cycle if there are outstanding packets in the flying packet list. When completed network packets are detected by M5 events, they are converted back to the M5 format by referencing the flying packet list and are processed at the destination caches.

Table III: Network architecture details.

Network	Physical Layout	Logical Topology	VC Type	VC Count
C-VC	cmesh	cmesh	VC	3
C-EVC	cmesh	flatfly	EVC	1
C-EVC-T	cmesh	MECS	EVC-T	1
MECS	MECS	MECS	N.A	N.A.

Using our integrated M5-BookSim full-system simulator, we run 8 benchmarks from the NAS parallel benchmark suite (`mg, ep, is, cg, lu, sp, ua,` and `ft`) with class B problem set. We fast-forward 2 billion instructions to warm up the system for avoiding cold-start effects and to reach the parallel execution phase of these applications. We execute 1 billion instructions after the fast-forward phase using the detailed out-of-order CPUs in M5 for all benchmarks to quantify their performance. The performance statistics are also used as inputs for our power model.

We use application IPC, defined in Equation (1) [21], as the metric to evaluate the performance of the benchmarks. This metric considers the variations of the execution time among different threads. It accumulates all the instructions executed across all threads and then divides the total instruction count by the number of cycles for the longest thread, as the longest thread determines the application finish time.

$$IPC_{app} = \frac{\sum_{i=1}^{num\_core} Committed\_instructions_{core[i]}}{\max_{1\leq i\leq num\_core} Number\_of\_cycles_{core[i]}} \quad (1)$$

To estimate power for the cores in our target system, we utilize McPAT 0.7 [22] that computes the power costs based on the performance statistics collected by M5. We calibrate the McPAT outputs to match the published core power values of the Intel SCC [1] using the scaling method introduced in [21]. The L2 cache power is computed using CACTI 5.3 [23]. Since the current version of CACTI does not support 22 nm process technology, we calculate L2 cache power in 32 nm technology and calibrate it using the same scaling method used for calculating core power.

The power for the network is estimated using detailed circuit modeling. For the 64-core target system, we use energy-optimized repeater-inserted wires for inter-router channels. The power dissipated in the SRAM array and crossbar of the router is calculated using the methodology described in [24] and [25], respectively. We design the network channels and routers using PTM for 22 nm technology [18]. The static power consumed by the network depends on the physical layout and the dynamic power is determined by the flow control mechanism and network traffic workloads.

We use EDP defined in equation (2) as a metric to evaluate system energy efficiency. $System\_power$ includes core power, cache power and NoC power. The running time

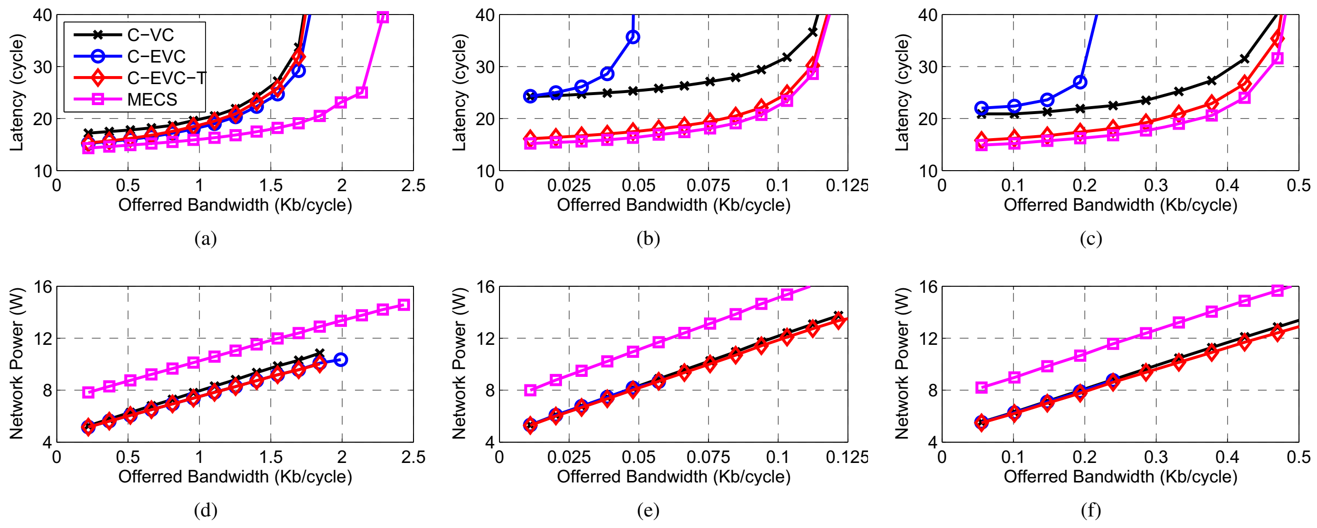

Figure 8: Network latency and network power vs. offered bandwidth for a 64-core system with uniform random traffic. (a) and (d) Data traffic. (b) and (e) Broadcast traffic. (c) and (f) Mixed traffic of both packet types.

is calculated by dividing the maximum number of execution cycles among all the cores by the system frequency.

$$\begin{aligned} EDP &= System\_power \cdot Application\_running\_time^2 \quad (2)\\ &= System\_power \cdot \left(\frac{Longest\_thread\_execution\_cycles}{System\_frequency}\right)^2 \end{aligned}$$

B. Synthetic benchmarks

For synthetic network traffic, we assume uniform random (UR) traffic pattern, which is widely used for NoC evaluations. We consider UR selection of sources and destinations for data packets and UR selection of sources for broadcast packets. We assume the same number of broadcast and data packets are injected into the network. However, for real-world network traffic, a higher number of broadcast packets may exist in comparison to the number of data packets due to operations such as write back and global status update. For example, an L1 cache requests exclusive access of a cache line by sending a broadcast packet to all other L1 caches. The receivers remove the local cache lines without sending response packets. Such packets are ignored in the synthetic traffic simulation but are included in the M5-BookSim full-system simulation.

Figure 8 compares the latencies and power costs of the four networks listed in Table III for data traffic, broadcast traffic, and mixed traffic of broadcast and data packets. For data traffic, in Figure 8(a) and Figure 8(d), the C-EVC and C-EVC-T networks have lower latencies than the C-VC network because some data packets are bypassed at several intermediate routers. The C-EVC-T and MECS network have comparable low latencies when the network traffic is not high. The MECS network has higher saturation throughput than the C-EVC-T network but the physical express channels in the MECS network consumes lots of fixed power, which is inefficient for low network traffic. The C-VC, C-EVC and E-EVC-T networks have comparable power costs because they are using the same numbers of physical channels, router buffers and crossbar sizes. The slight power difference is due to their respective router control logic. When the network traffic increases, the C-EVC and C-EVC-T networks have lower power costs than the C-VC network because the dynamic power is not consumed in buffering and crossbar switching at the bypassed routers in the C-EVC and C-EVC-T networks.

For broadcast traffic, in Figure 8(b), the C-VC and C-EVC network have higher latencies than the C-EVC-T network because of the multiple hop transmission of broadcast packets in these two networks. Here, we assume that broadcast packets complete network transmission after reaching all destinations. In a traditional MECS channel, a data packet has one target receiver router and only the target receiver router drops the passing packet. For comparison of broadcast traffic in the C-EVC-T and MECS networks, we modify the packet format and router architecture of the MECS network to support broadcasting. A broadcast packet in MECS channel has multiple target receivers, and multiple target routers drop the passing broadcast packet. Figure 8(e) shows that the EVC-T and MECS networks have comparable latencies and saturation throughput for broadcast traffic. The C-EVC-T and MECS networks get saturated by high broadcast traffic because of increased contentions inside the routers. The higher bisection bandwidth of the MECS network does not increase its saturation throughput for broadcast traffic.

The mixed traffic of broadcast and data packets provides a realistic comparison of the four networks. In Figure 8(c),

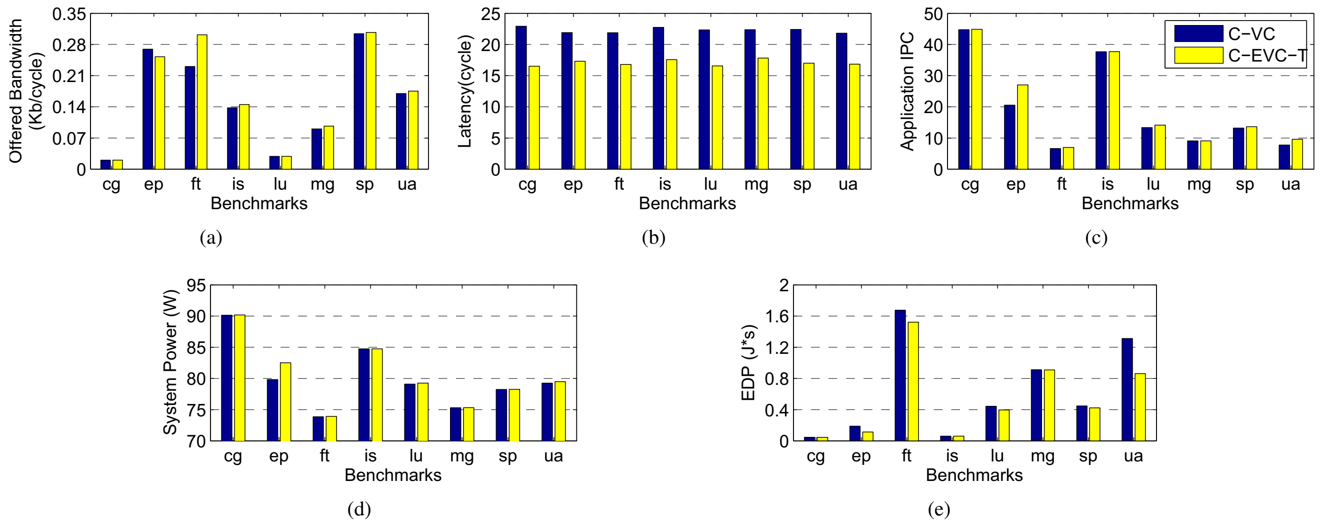

Figure 9: Comparison of C-VC and C-EVC-T networks using NAS benchmarks running on the 64-core target system. (a) Offered bandwidth. (b) Average network latency. (c) Application IPC. (d) System power. (e) EDP. The detailed system configuration is given in Table I. BookSim was integrated into M5 to enable a detailed cycle-accurate comparison.

the latency is the average value for all broadcast and data packets. The C-EVC-T shows better performance than the C-VC and C-EVC networks because it is the only network that uses virtual express paths for both broadcast and data packets. The saturation throughput of the C-EVC-T network is similar to the C-VC network. In Figure 8(f), the C-EVC-T network has the similar power costs to the C-VC network. In summary, the MECS network provides the best performance–low network latency and high saturation throughput because of its physical express channels and high bisection bandwidth. The C-VC network provides the best energy efficiency–low power cost, due to its short channels and simple control logic. The proposed C-EVC-T network can achieve the similar high performance to the MECS network while maintaining the similar low energy costs to the C-VC network.

C. NAS benchmarks

We next evaluate our EVC-T technique by running the NAS parallel benchmark suite on our M5-BookSim simulator. In the full-system simulation, we compare the C-EVC-T and C-VC networks to demonstrate that EVC-T can help improve the system IPC and reduce the EDP of parallel applications. Figure 9 shows the full-system simulation results for NAS parallel benchmarks. Figure 9(a) shows the offered bandwidth for each benchmark. For our distributed L2 cache architecture, the offered bandwidth is calculated as the number of transmitted packet per cycle multiplied by the packet size. We assume that one data packet consists of four 128-bit flits and one broadcast packet consists of single 128-bit flit. The offered bandwidth shows that NAS benchmarks have various network demands, but none of them reaches the saturation region of the C-VC and C-EVC-T networks (see Figure 8(c)). For the offered bandwidth, the C-EVC-T network has lower average latency as shown in Figure 9(b). On average, the C-EVC-T network reduces the latency by 24%. If the C-EVC-T network runs benchmarks that have higher offered bandwidth, then the performance and power will be comparable to C-VC, which still outperforms than C-EVC.

Figure 9(c) shows the application IPC for each benchmark. The ‘ep’ and ‘ua’ benchmarks show an application IPC improvement of 31% and 24%, respectively, after using EVC-T. This improvement is a result of the reduced execution time of the longest thread (see Equation 1). The ‘ep’ benchmark is one of the highly parallel benchmarks in NAS benchmark suite and has the greatest improvement on application IPC. The application IPC of other benchmarks improve by up to 6%. Some benchmarks, such as ‘cg’ and ‘is’, have very high application IPCs, however, their low offered bandwidth indicates that they are relatively less dependent on network performance; i.e., they are not memory bound. Figure 9(d) shows the system power costs for each benchmark, which includes the power of networks and cores and caches. The ‘ep’ benchmark shows 3% power increase because of the improved IPC, while other benchmarks show less than 1% power increase because of the relatively non-uniform workload distribution. Figure 9(e) shows the EDP for each benchmark. The ‘ep’ and ‘ua’ benchmarks show 40% and 34% EDP reduction after using EVC-T. The average EDP reduction by EVC-T is 13%. The system simulation results show that our proposed EVC-T improves the system energy efficiency as well as the system performance.

VII. Conclusion

In this paper, we have proposed a new flow control technique: express virtual channels with taps (EVC-T) for NoC architectures in manycore systems. When used with cmesh physical layout, EVC-T helps create a logical topology with low diameter. This provides a NoC architecture which is easy to design from the hardware perspective and easy to program from the software perspective. In addition, we have also proposed a contention-free tree architecture that supports broadcasting on unordered on-chip interconnects for snoop-based cache coherency protocols. The notification trees enable a core to wait for less than one cycle after broadcast packet is received on average to make decisions on the correct processing order of broadcast packets.

We have evaluated the EVC-T flow control technique and the new broadcast mechanism for snoop-based cache coherency protocols using BookSim network simulator that is integrated into M5 full-system simulator. We have explored both synthetic traffic and parallel benchmarks from the NAS suite. The synthetic benchmark analysis shows the potential of our proposed techniques where the average packet (data and broadcast) latency is reduced by 24%, while consuming the same amount of power as a conventional cmesh network. For NAS parallel benchmarks, our techniques increase the application IPC by 9% on average with negligible changes in power. The system energy efficiency (quantified by EDP) is increased by 13% on average.

References

[1] J. Howard *et al.*, "A 48-core ia-32 message-passing processor with dvfs in 45nm cmos," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, 2010, pp. 108 –109.

[2] S. Bell *et al.*, "Tile64 - processor: A 64-core soc with mesh interconnect," in *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, 2008, pp. 88 –598.

[3] D. Culler, J. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, 1st ed. Morgan Kaufmann, 1998, the Morgan Kaufmann Series in Computer Architecture and Design.

[4] B. Grot, J. Hestness, S. Keckler, and O. Mutlu, "Express cube topologies for on-chip interconnects," in *Proc. IEEE 15th Int. Symp. High Performance Computer Architecture HPCA 2009*, 2009, pp. 163–174.

[5] D. Bailey *et al.*, "The NAS parallel benchmarks," Tech. Rep. RNR-94-007, 1994.

[6] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[7] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt, "The m5 simulator: Modeling networked systems," *Micro, IEEE*, vol. 26, no. 4, pp. 52 –60, 2006.

[8] A. Charlesworth, "The sun fireplane interconnect," *IEEE Micro*, vol. 22, no. 1, pp. 36–45, 2002.

[9] M. R. Marty and M. D. Hill, "Coherence ordering for ring-based chip multiprocessors," in *Proc. MICRO-39 Microarchitecture 39th Annual IEEE/ACM Int. Symp*, 2006, pp. 309–320.

[10] K. Strauss, X. Shen, and J. Torrellas, "Uncorq: Unconstrained snoop request delivery in embedded-ring multiprocessors," in *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, 2007, pp. 327 –342.

[11] J. Reynolds, P.F., C. Williams, and J. Wagner, R.R., "Isotach networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 8, no. 4, pp. 337 –348, 1997.

[12] E. Bilir *et al.*, "Multicast snooping: a new coherence method using a multicast address network," in *Computer Architecture, 1999. Proceedings of the 26th International Symposium on*, 1999, pp. 294 –304.

[13] C. Williams, P. F. Reynolds, and B. R. de Supinski, "Delta coherence protocols," *IEEE Concurrency*, vol. 8, pp. 23–29, 2000.

[14] N. Agarwal, L.-S. Peh, and N. K. Jha, "In-network snoop ordering (inso): Snoopy coherence on unordered interconnects," in *Proc. IEEE 15th Int. Symp. High Performance Computer Architecture HPCA 2009*, 2009, pp. 67–78.

[15] J. Kim, J. Balfour, and W. J. Dally, "Flattened butterfly topology for on-chip networks," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO-40)*, 2007, pp. 172–182.

[16] A. Joshi, B. Kim, and V. Stojanovic, "Designing energy-efficient low-diameter on-chip networks with equalized interconnects," in *Proc. 17th IEEE Symp. High Performance Interconnects HOTI 2009*, 2009, pp. 3–12.

[17] A. Kumar, L.-S. Peh, P. Kundu, and N. Jha, "Toward ideal on-chip communication using express virtual channels," *IEEE Micro*, vol. 28, no. 1, pp. 80–90, 2008.

[18] K. Kuhn, M. Liu, and H. Kennel, "Technology options for 22nm and beyond," in *Junction Technology (IWJT), 2010 International Workshop on*, 2010, pp. 1 –6.

[19] T. Krishna, A. Kumar, L.-S. Peh, J. Postman, P. Chiang, and M. Erez, "Express virtual channels with capacitively driven global links," *IEEE Micro*, vol. 29, no. 4, pp. 48–61, 2009.

[20] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Netw.*, vol. 7, no. 2, pp. 188–201, 1999.

[21] J. Meng, C. Chen, A. K. Coskun, and A. Joshi, "Run-time energy management of manycore systems through reconfigurable interconnects," in *Proceedings of ACM Great Lakes Symposium on VLSI*, ser. GLSVLSI '11, 2011.

[22] S. Li *et al.*, "McPAT: An integrated power, area, and timing modeling framework for multicore and many-core architectures," in *MICRO-42*, 2009, pp. 469 –480.

[23] S. Thoziyoor *et al.*, "CACTI 5.1," HP Laboratories, Palo Alto, Tech. Rep., 2008.

[24] X. Liang, K. Turgay, and D. Brooks, "Architectural power models for sram and cam structures based on hybrid analytical/empirical techniques," in *International Conference on Computer Aided Design (ICCAD)*, 2007, pp. 824 –830.

[25] H. Wang, L.-S. Peh, and S. Malik, "Power-driven design of router microarchitectures in on-chip networks," in *IEEE/ACM International Symposium on Microarchitecture (MICR0-36)*, 2003, pp. 105 – 116.