A Case for Bufferless Routing in On-Chip Networks

Thomas Moscibroda
Microsoft Research
moscitho@microsoft.com

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu

ABSTRACT

Buffers in on-chip networks consume significant energy, occupy chip area, and increase design complexity. In this paper, we make a case for a new approach to designing on-chip interconnection networks that eliminates the need for buffers for routing or flow control. We describe new algorithms for routing without using buffers in router input/output ports. We analyze the advantages and disadvantages of bufferless routing and discuss how router latency can be reduced by taking advantage of the fact that input/output buffers do not exist. Our evaluations show that routing without buffers significantly reduces the energy consumption of the on-chip cache/processor-to-cache network, while providing similar performance to that of existing buffered routing algorithms at low network utilization (i.e., on most real applications). We conclude that bufferless routing can be an attractive and energy-efficient design option for on-chip cache/processor-to-cache networks where network utilization is low.

Categories and Subject Descriptors: C.1.2 [Computer Systems Organization]: Multiprocessors–Interconnection architectures; C.1.4 [Parallel Architectures]: Distributed architectures.

General Terms: Design, Algorithms, Performance.

Keywords: On-chip networks, multi-core, routing, memory systems.

1. INTRODUCTION

Interconnection networks are commonly used to connect different computing components [12]. With the arrival of chip multiprocessor systems, on-chip interconnection networks have started to form the backbone of communication between cores and cores and memory within a microprocessor chip [53, 42, 24, 6]. Several network-on-chip (NoC) prototypes show that NoCs consume a substantial portion of system power: ∼30% in the Intel 80-core Terascale chip [24], and ∼40% in the MIT RAW chip [50]. As power/energy consumption has already become a limiting constraint in the design of high-performance processors [20] and future on-chip networks in many-core processors are estimated to consume hundreds of watts of power [6], simple energy- and area-efficient interconnection network designs are especially desirable.

Previous on-chip interconnection network designs commonly assumed that each router in the network needs to contain buffers to buffer the packets (or flits) transmitted within the network. Indeed, buffering within each router improves the bandwidth efficiency in the network because buffering reduces the number of dropped or "misrouted" packets [12], i.e. packets that are sent to a less desirable destination port. On the other hand, buffering has several disadvantages. First, buffers consume significant energy/power: dynamic energy when read/written and static energy even when they are not occupied. Second, having buffers increases the complexity of the network design because logic needs to be implemented to place packets into and out of buffers. Third, buffers can consume significant chip area: even with a small number (16) of total buffer entries per node where each entry can store 64 bytes of data, a network with 64 nodes requires 64KB of buffer storage. In fact, in the TRIPS prototype chip, input buffers of the routers were shown to occupy 75% of the total on-chip network area [22]. Energy consumption and hardware storage cost of buffers will increase as future many-core chips will contain more network nodes.

In this paper, we propose to eliminate buffers in the design of on-chip cache-to-cache and cache-to-memory networks to improve both energy- and area-efficiency, as well as reduce network design complexity and router latency. The basic idea of "bufferless routing" is to *always route a packet (or a flit) to an output port* regardless of whether or not that output port results in the lowest distance to the destination of the packet. In other words, packets are deflected or "misrouted" [12] by the router to a different output port if an output port that reduces the distance to the destination node is not available. Bufferless routing has also been called "hot-potato" routing in network theory [2], alluding to the scenario that the router immediately needs to pass the potato (i.e. the packet) on to some other router as the potato is too hot to keep (i.e. buffer).

We propose and evaluate a set of simple and practical bufferless routing algorithms (BLESS), and compare them against baseline buffered algorithms in terms of on-chip network energy consumption and latency. We find that BLESS routing can yield substantial reductions in network energy consumption, while incurring little extra latency (versus buffered algorithms) if the average injected traffic into the network is low, i.e. below the network saturation point. We find that in most application scenarios, due to low cache miss rates, the average injected traffic into the cache-to-cache and cache-to-memory networks is very low, making BLESS a potentially attractive mechanism for on-chip networks.

Contributions: This work makes the following contributions:

- We propose a variety of simple and effective routing algorithms for bufferless routing. We show how bufferless routing can be effectively combined with wormhole routing.
- We comprehensively evaluate the network energy consumption, performance, latency, and area requirements of bufferless routing using real applications simulated on a self-throttling chip multiprocessor (CMP) on-chip-network, as well as using synthetic workloads. We show that bufferless routing can result in average network energy reduction of ∼40% without significantly impacting application performance, while reducing network buffer area requirements by ∼60%.
- We show how eliminating buffers can enable reductions in router latency. The reduced router latency can enable bufferless routing algorithms to outperform baseline buffered algorithms.

2. WHY COULD IT WORK?

At first thought, eliminating buffers in on-chip interconnection networks might appear audacious, since it will result in an increase in average packet latencies and a decrease in achievable network throughput compared to buffered routing schemes. Moreover, there could be other issues such as livelocks. Nonetheless, the approach could be suitable for on-chip networks, as we describe below. Intuitively, bufferless deflection routing works well when network utilization is

ISCA'09, June 20–24, 2009, Austin, Texas, USA.
Copyright 2009 ACM 978-1-60558-526-0/09/06 ...$5.00.

low. A packet is deflected only if a collision occurs in a router, i.e., if multiple packets arrive at a router at the same time, and if not all of these packets can be sent in a productive direction.[1] If only few packets are in the network simultaneously, the number of collisions is low. Hence, most packets progress quickly and can be routed to their destination without being frequently deflected.

For larger traffic volumes, the fundamental effect of removing buffers is a *reduction of the total available bandwidth* in the network. In a buffered network, a packet waits idle in some buffer until it can be routed in a productive direction and therefore does not unnecessarily consume link bandwidth while it is buffered. In contrast, in a bufferless network all packets *always* consume link bandwidth because, in effect, links act as "buffers" for the packets. Therefore, beyond a certain packet injection rate into the network, bufferless routing algorithms will fail, while good buffered routing algorithms can still perform well. More precisely, the network saturation throughput Θ_{BLESS} of bufferless routing is less than the saturation throughput of buffered routing Θ_B.

The critical questions that determine the potential usefulness of bufferless routing in on-chip interconnection networks are therefore 1) how much energy reduction can be achieved by eliminating buffers, 2) how large is the gap between Θ_{BLESS} and Θ_B, and how well does bufferless routing perform at injection rates below Θ_{BLESS}, 3) are there any realistic situations in which an interconnection network is operated at a traffic injection rate below Θ_{BLESS}, and 4) are there benefits to eliminating buffers, such as simplicity of design or ability to reduce router latency?

Our results in this paper show that the answers to the first three questions are promising for bufferless routing in on-chip networks. Many on-chip interconnection networks are observed to be operating at relatively low packet injection rates [27, 25], which are significantly below their peak throughput, an observation we confirm in this paper using realistic applications and a self-throttling CMP network design in which processors stop injecting into the network once their internal buffers (e.g. MSHRs [29]) become full. For instance, the L1 miss rate is typically below 10%, which, in a chip multiprocessor with a distributed shared L2 cache, results in very low packet injection rates for the network connecting L1 caches and L2 cache banks [9]. Hence, bufferless routing, which performs well at low packet injection rates can be a promising approach for on-chip networks that primarily operate at low utilization.

Finally, this paper also provides promising answers to the fourth question by analyzing the tradeoffs, advantages, and disadvantages involved in bufferless routing and utilizing the lack of buffers to simplify router design and reduce router latency in a simple way.

3. ON-CHIP BUFFERLESS ROUTING

3.1 Overview

The basic principle of bufferless routing in on-chip interconnect networks is simple: Since in bufferless routing routers cannot store packets in transit, all packets that arrive at a router must immediately be forwarded to an adjacent router. We first present a simple version of our algorithm (FLIT-BLESS) in which routing is flit-switched, i.e., every flit is routed through the network individually. We then propose WORM-BLESS, an optimization to FLIT-BLESS which combines bufferless routing with ideas from wormhole routing. Finally, in §3.4, we show how BLESS can seamlessly be used *with* buffers, if desired. We discuss advantages and disadvantages of BLESS in §5.

3.2 Basic Algorithm: Flit-Level Routing

Overview: In flit-level bufferless routing (FLIT-BLESS or simply FLIT), each flit of a packet is routed independently of every other flit through the network, and different flits from the same packet may take different paths.[2] When there is contention between multiple flits that are destined for a particular direction, only one of these flits is actually sent to the corresponding output port. In such a case, traditional routing algorithms would temporarily store such flits in a buffer within the router; and credits would flow between neighboring routers in order to prevent buffer overflows. In contrast, BLESS does not have buffers and therefore sends these flits to other, potentially undesirable output ports. In other words, flits that cannot be sent to a productive direction are "deflected." The basic idea of BLESS is that deflected packets will eventually reach their destinations, and—as we show in our evaluation—that the total extra latency due to the detours resulting from deflections is not too high.

Network Topology: BLESS routing is feasible on every network topology that satisfies the following two constraints: Every router 1) has at least the same number of output ports as the number of its input ports, and 2) is reachable from every other router. Many important topologies such as Mesh, Torus, Hypercubes, or Trees satisfy these criteria. However, bufferless routing cannot easily be applied to networks with directed links, such as the Butterfly network, as a deflected packet may no longer be able to reach its destination [12]. In the sequel, we specifically assume a Mesh topology.

Injection Policy: A processor can safely inject a flit into its router when at least one incoming link (from other routers) is free. Unless all input ports are busy, there must be at least one free output port, to which a newly injected flit can be sent. That is, all incoming flits can be routed to some direction. Observe that this allows for entirely *local flow and admission control*; every processor can locally decide whether or not it can inject a flit in a given cycle. Thus, in contrast to existing routing algorithms, BLESS does not require a credit-based system to avoid overflows.

Arbitration Policy: The algorithm's arbitration policy has to decide which incoming flit is routed to which output port. The arbitration policy of BLESS is governed by two components, a *ranking component* and an *port-selection component*. In combination, these two components determine the arbitration of flits, but the components are orthogonal in the sense that they can be changed independently.

BLESS' arbitration policy is *rank-based*. In every cycle, the router ranks all incoming flits using the *flit-ranking component*. For each flit, the *port-prioritization component* ranks the available output ports in order of their "desirability" for this flit. The router then considers the flits one by one in the order of their rank (highest rank first) and assigns to each flit the output port with highest priority that has not yet been assigned to any higher-ranked flits.

Example: *Consider two flits A and B contending in a router. Let the flit-ranking component rank A higher than B, and let the port-priorities for A and B be (N, E) and (N, W), respectively. BLESS will assign A to North, and B to West.*

We now discuss flit-ranking and port-prioritization components of FLIT-BLESS in detail.

FLIT-BLESS: Flit-Ranking: We implement and evaluate five different flit-ranking schemes in BLESS (see Table 1, Row 2). Different policies have different advantages and disadvantages. However, as we show in the evaluation (see §7.6) the simple Oldest-First (OF) policy, which ensures there is a total age order among flits and prioritizes older flits, performs the best over a wide range of traces, both in terms of average/maximum latency and deflection-count/energy-efficiency. OF has another crucial advantage over other policies in that it is guaranteed to avoid livelocks (see below). For these two reasons, we select OF as our primary ranking policy. Implementing OF such that there is a total age order of flits is non-trivial. However, there are previously-researched techniques (e.g. timestamping [34]) to make OF implementable.

[1] A productive direction is a direction (or output port) that brings the packet closer to its destination [45]. A non-productive direction brings the packet further away from its destination.

[2] Technically, what we call a flit in FLIT-BLESS is actually a "packet" as it is a unit of routing, but we refer to it as "flit" to keep consistency with WORM-BLESS, which is explained later.

Flit Ranking Rules	Comments
Rule 1) MSF: Must-Schedule First	(see Section 3.4)
Rule 2) Evaluated flit-ranking rules:	
— i) OF: Oldest First	Older flits before younger flits.
— ii) CF: Closest First	Flits closer to their destination before flits whose remaining distance is larger.
— iii) DEFs: Most Deflections First	Flits that have been deflected more before flits deflected fewer times.
— iv) RR: Round Robin	Flits from different input ports are ranked in round robin fashion.
— v) Mix: Mixed Policy	In odd (even) cycles, use OF (RR).

Table 1: Different flit-ranking schemes. The highest priority MSF rule applies only to BLESS with buffers (see Section 3.4).

FLIT-BLESS: Port-Prioritization: Table 2(left) shows the port-prioritization rules of FLIT-BLESS. For a given flit, the router picks the output port that conforms to the highest priority according to the figure. In the case of FLIT-BLESS, this simply means that productive directions are preferred over non-productive directions. If there are two possible ports with same priority, the router picks the x-direction over y-direction as in dimension-order routing. During this process, only *free* output ports are considered, i.e., ports not already assigned to higher-ranked flits in the same cycle.

Deadlocks: Guaranteeing the absence of deadlocks and livelocks is of critical importance to any routing algorithm. The use of deflection routing and ensuring that the number of output ports in a router is greater than or equal to the number of input ports ensure no deadlocks can occur. Protocol (request-reply) deadlocks are avoided by ensuring that replies can always reach their destinations: a reply inherits the age of its corresponding request and OF ranking ensures the oldest flit will always be delivered to its destination.

Livelocks: The combination of OF ranking and port-prioritization ensures that no livelocks can occur. In OF, the oldest flit is highest-ranked and hence, it can always be assigned a productive direction. By induction, this guarantees that no livelocks can occur because once a flit is the oldest flit in the network, it cannot be deflected anymore and is guaranteed to make forward progress until it reaches its destination. Eventually, a flit will become the oldest flit in the network, after which it cannot be deflected any more.

Implementation: In FLIT-BLESS, every packet is routed individually through the network. Therefore, every flit needs to contain routing information, or in other words, every flit needs to be a *head-flit*. BLESS adds additional wires between routers to transport this information. Our evaluations show that the energy consumed by these additional wires is small, compared to the energy savings due to not having buffers in the routers. As in the baseline algorithms, a destination BLESS router buffers received flits in a receiver-side buffer until all flits of a packet have arrived upon which the packet is delivered to the receiving processor.

3.3 BLESS Wormhole Routing

Compared to wormhole routing [11] in traditional, buffered baseline routing algorithms, flit-level switching in FLIT-BLESS has three potential disadvantages. First, it is *not energy-optimal* because every flit needs to be a head-flit, i.e., the additional header-wires need to be activated for every flit instead of only the first flit of a packet as in baseline algorithms. Second, it can have a negative impact on packet latency. The reason is that since every flit may take a different route through the network, it is statistically more likely that one flit is delayed or takes a detour, thereby delaying the entire packet. Third, for the same reason, flit-level switching tends to increase the receiver-side buffering requirement as different flits may take different paths.

To mitigate the above disadvantages, we propose WORM-BLESS, which is an optimization to FLIT-BLESS that combines BLESS with ideas from wormhole routing. Ideally, only the first flit of each packet should contain header information (head-flit) and all subsequent flits should simply follow the preceding flit. In case a head-flit is deflected, the entire packet (=worm) would follow this head-flit and would be deflected. With buffers, this pure form of wormhole routing can be enabled using credit-based flow-control and is very efficient. Without buffers, however, pure wormhole routing is impossible because the injection policy becomes unclear, and livelocks can occur.

- *Injection Problem:* In FLIT-BLESS, a processor can inject a packet whenever at least one input port is free. If all flits of a packet have to be routed in succession, this is no longer possible. If an input port is free and a processor starts injecting a worm, it could happen that in a subsequent cycle (while this packet injection is still going on), flits arrive on all input ports. In this case, the router would be unable to send out all incoming flits, which in the absence of buffers, must not happen. Say, in a 2D-Mesh, a core starts injecting an 8-flit worm into the router at cycle T (because not all of the input ports were busy). At cycle $T + 3$, new worms arrive at all four input ports to be forwarded to other routers. Now, at cycle $T + 3$, there are 5 worms (flits) that need to be routed to 4 output ports, which is not possible.
- *Livelock Problem:* If entire worms can be deflected, livelock can occur because arbitration is performed only for head-flits. When a worm arrives at a router, even if it is the oldest in the network, it might not get a chance to arbitrate with other worms in other input ports because other worms might already be transmitting their body flits. In such a case, the oldest worm is deflected instead of being prioritized. This can happen to a worm in all routers and as a result the worm might never reach its destination; livelock ensues.

In WORM-BLESS, we solve these problems using *worm truncation*. As in baseline wormhole routing, each router maintains an allocation of worms to output ports. Once the head-flit of a packet is routed to a specific output port, this port is allocated to this worm until the tail-flit of the worm passes the output port (see [12] for a detailed description of wormhole routing). This allocation is accomplished by keeping a small table in the router that contains information on which output port is allocated to which worm.

WORM-BLESS: Injection Policy: The injection policy remains the same. A worm can be injected whenever in a cycle, not all 4 input ports are busy. In case new worms arrive on all 4 input ports while the source is injecting a worm, the injected worm is truncated. The second part of the worm can be injected as soon as one input port becomes free. When a worm is truncated, the first flit of the truncated worm's second part becomes a new head-flit.

WORM-BLESS: Flit-Ranking: Flit-ranking remains unchanged compared to FLIT-BLESS (see Table 1).

WORM-BLESS: Port-Prioritization: In WORM-BLESS, port-prioritization is critical to avoid livelocks. In order to maintain the key invariant that the oldest flit in the network *always* makes progress towards its destination, we allow worms to be *truncated* in certain cases. Table 2 (middle) shows the port-prioritization rules. The rules distinguish between head-flits and non-head flits. For a head-flit, a new output port must be allocated. For this purpose, productive ports that are not currently allocated to any existing worms have highest priority (Rule 1). If no such port exists, the next highest priority are free ports that are productive, but are currently allocated to an existing worm (Rule 2). In this case, the port is re-allocated to the new worm. Effectively, this means that the existing worm (to which that port had been allocated before) will be *truncated*. Notice that only a higher-ranked head-flit can truncate an existing worm. If a worm is *truncated*, the first flit of the worm's second part (which has not yet left the router) becomes a new head-flit; it will be allocated a new output port once it is this flit's turn to be assigned. Both parts of the worm are then independently routed to the destination. In case all productive directions are already occupied, a head-flit is assigned a non-productive port, i.e., it is deflected (Rules 3 and 4). Finally, if the flit is not a head-flit, then by definition, this flit's worm is already allocated to a specific output port, to which this flit is sent (Rule 5).[3]

[3] Note that the allocated output port of a non-head-flit can never be blocked

FLIT-BLESS:	**WORM-BLESS:**	**WORM-BLESS with buffers:**
	if flit is head-flit **then**	**if** flit is head-flit **then**
1: assign flit to productive port	1: assign flit to unallocated, productive port	1: assign flit to unallocated, productive port
		if flit is mustSchedule **then**
2: assign flit to non-productive port	2: assign flit to allocated, productive port	2: assign flit to allocated, productive port
→ **packet is deflected**	→ **another worm is truncated**	→ **another worm is truncated**
	3: assign flit to unallocated, non-productive port	3: assign flit to unallocated, non-productive port
	→ **packet is deflected**	→ **packet is deflected**
	4: assign flit to allocated, non-productive port	4: assign flit to allocated, non-productive port
	→ **another worm is truncated**	→ **another worm is truncated**
	→ **packet is deflected**	→ **packet is deflected**
	else	**else**
	5: assign the flit to port that is allocated to its worm	5: buffer the flit → **packet is buffered**
	end if	**end if**
		else
		6: assign the flit to port that is allocated to its worm
		end if

Table 2: BLESS port-prioritization policy for bufferless FLIT-BLESS (left), bufferless WORM-BLESS (middle), and WORM-BLESS with buffers (right). For a given flit, the router picks the output port that conforms to the highest priority according to the above table. Only free output ports are considered, i.e., output ports that have not previously been assigned to higher-ranked flits in the same cycle. If two free output ports have the same priority, the router picks the x-direction ahead of the y-direction.

The key insight is that the combination of OF routing and port-prioritization with truncation ensures that the oldest flit is *always* routed to a productive direction. Therefore, the *absence of livelocks* is guaranteed, because the oldest flit always makes progress towards its destination. In our evaluation section, we show that because of its reduction in header-flits, WORM-BLESS does indeed save energy compared to FLIT-BLESS in many cases.

Additional Information Needed in the Router: To implement truncation, the router needs to be able to create head flits out of body flits at the time of truncation. For this, the router stores the packet header information from the original head-flit of a worm in the table that also maps output ports to allocated worms. Note that, the stored header information is transmitted using the additional wires dedicated for header transmission, as described in FLIT-BLESS.

3.4 BLESS with Buffers

Completely bufferless routing as described so far is the extreme end of a continuum. In this section, we show how BLESS can also be used at other points in this continuum to achieve different tradeoffs. Specifically, although our evaluations, e.g. §7.2), show that not using any buffers has little impact on performance of real applications, adding buffers to BLESS can nonetheless have benefits: It increases throughput and decreases latency for high injection rates (§7.5).

Hence, if good performance at high injection rates is desired, buffers can easily be integrated into BLESS. The basic principle of deflection routing remains the same: no non-local (e.g., credit-based) flow-control is required, arbitration is determined purely locally based on ranking and port-prioritization. The key benefit of buffers in BLESS is to reduce the *probability of misrouting*, i.e. to increase the likelihood that a worm/flit is routed to a productive output port. If a productive port does not exist for a buffered flit, the flit remains in the buffer until 1) the buffer is full in which case the flit must be scheduled or 2) a productive port becomes available.

Compared to bufferless WORM-BLESS, the following changes apply. The router maintains one bit, mustSchedule, per input port. When the buffer associated to this input port becomes full, the bit is set to true. It means that the oldest flit in this buffer is now mustSchedule, i.e., it *must* be sent out in the next cycle in order to make room for a new incoming flit that might arrive on that input port. If a router did not send out all mustSchedule flits each cycle, it could happen that it suddenly has to send out more flits than it has output ports.

WORM-BLESS with Buffers: Injection Policy: Remains the same as in WORM-BLESS.

WORM-BLESS with Buffers: Flit-Ranking: The basic ranking policy is adjusted in one critical way. Flits that are mustSchedule are ranked higher than flits that are not mustSchedule. This is important for preventing livelocks. Among mustSchedule flits and among flits that not mustSchedule, the ranking is determined by OF.

WORM-BLESS with Buffers: Port-Prioritization: As shown in Table 2 (right), for mustSchedule head-flits and non head-flits, port prioritization rules remain the same (including possible truncations). However, for head-flits that are not mustSchedule prioritization is different: A flit is sent out (and the corresponding output port allocated) only if it can be sent to an unallocated (by another worm), productive direction. Otherwise, the flit is buffered (Rule 5). Hence, only mustSchedule head-flits can truncate worms.

Livelocks: The absence of livelock is guaranteed. In any cycle, at most $InputPorts$ input ports (including the injection port) can be mustSchedule. Hence, all mustSchedule flits can be sent out. Furthermore, the oldest flit in the network is always guaranteed to make progress to the destination: If it is mustSchedule, it can always be sent to a productive direction (possibly by truncating an existing worm with lower-ranked flits, see Table 2 (right)). If the oldest flit is not mustSchedule, it is not deflected and waits for a productive output port to free up.

Finally, note that BLESS with buffers can also be used in combination with multiple *virtual channels* [10].

4. REDUCTION OF ROUTER LATENCY

Bufferless routing eliminates input buffers and virtual channels from the design of the on-chip network. The elimination of input buffers and virtual channels not only reduces design complexity in the router, but can also enable reduction in router latency.

Baseline router pipeline: Figure 1(a) shows the baseline router pipeline of a state-of-the-art virtual channel router, as adapted from Dally and Towles [12]. The baseline router we assume in this work is similar to that used in [30] employs several features to reduce router latency for each flit, including speculation [44, 39]. The pipeline consists of three stages: 1) buffer write (BW) and for head flits route computation (RC), 2) virtual channel allocation (VA) and switch allocation (SA) done speculatively in parallel, and 3) switch traversal (ST), followed by link traversal (LT). We refer the reader to Dally and Towles [12] for more detailed descriptions of this baseline router pipeline. Note that in our evaluations we assume an aggressive 2-cycle latency for the baseline router pipeline as with careful design, the pipeline can be modified to employ double speculation so that switch traversal can be performed along with VA and SA.

BLESS router pipeline: In the baseline router, VA/SA stage is needed to allocate virtual channels (VA) and to arbitrate between virtual channels for physical output ports (SA). Since all virtual chan-

(already assigned to another flit). If a higher-ranked flit is assigned to an output port allocated to a different worm, that worm is truncated and the first flit of the worm's second part becomes a head-flit. Hence, once it is this flit's turn to be assigned, the head-flit port-prioritization rules apply to that flit.

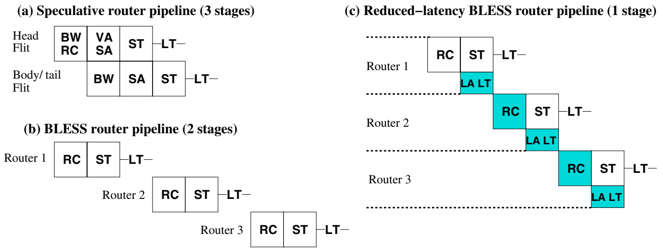

Figure 1: Baseline and BLESS router pipelines

nels are eliminated in our design, this stage can be eliminated. In addition, BW can be eliminated since there are no buffers, but this likely does not reduce latency because RC still needs to be performed in the first stage (RC stage implements the bufferless routing and arbitration algorithms described in the previous section). As a result, the baseline BLESS router consists of 2 stages as shown in Figure 1(b). In our main evaluations, we assume the BLESS router latency is 2 cycles.

Reduced-latency BLESS router pipeline: The latency of the BLESS router can be further optimized by employing lookahead techniques [17, 12] as shown in Figure 1(c). The basic idea is to have enough information about a flit to travel to its destination one cycle ahead of the flit itself on a separate, narrow link. The link traversal of the lookahead (LA LT) is accomplished in parallel with the switch traversal (ST) of the flit. As a result, while the flit itself traverses the link, the next router performs route computation (RC) for the flit, using the lookahead information. The result of the route computation is stored in the pipeline latches feeding the ST stage. When the flit itself arrives in the next cycle, using this information it directly enters the ST stage. Hence, using lookahead-based routing, it takes a flit only one stage (ST) to pass through the router pipeline. Figure 1(c) depicts the lookahead pipeline stages as shaded in dark. With this optimization, a BLESS router can have single-cycle latency, which we assume in some of our evaluations.

This lookahead-based latency optimization for the BLESS router is *not speculative*. Since, the bufferless router guarantees that a flit will always routed after it arrives, lookahead routing performed in the previous cycle never fails. In contrast, in a buffered router it is possible that lookahead routing can fail, in which case actions should be taken to detect the failure and recover [30].

We would like to note that eliminating virtual channels comes with a disadvantage: different classes of traffic can no longer be accommodated easily in our bufferless routing technique because they cannot be assigned different virtual channels. This is a shortcoming of our existing design, which we intend to address in the future. Instead of using virtual channels, our mechanism needs to use the routing algorithm to distinguish between different classes of traffic. For example, each packet can be augmented with its priority. Bufferless routing algorithms can be designed to prioritize such packets in their routing decisions: algorithms can be designed to minimize the deflections incurred by traffic classes that have high priority. In essence, to incorporate different treatment of different traffic classes, the routing logic that makes deflection decisions needs to be made aware of the class of each packet. Designing such algorithms, while a very interesting research direction, is beyond the scope of this paper.

5. ADVANTAGES AND DISADVANTAGES

In this section, we provide a qualitative discussion of the advantages and disadvantages of BLESS compared to traditional buffered schemes. Quantitative comparisons follow in Section 7.

5.1 Advantages

No Buffers: Clearly, this is the key advantage of our approach because it helps reduce both complexity and, as we show in our evaluation, energy consumption.

Purely Local and Simple Flow Control: Any buffered routing scheme inherently requires some kind of communication-based flow control mechanism or rate limitation in order to prevent the buffers in the routers from overflowing. Flow control is simpler in bufferless routing. A node safely injects a new packet into the network when at least one incoming link from another node is free, which can be detected locally without any need for communication between routers.

Simplicity and Router Latency Reduction: Buffered routing algorithms employ virtual channels [10] to improve buffer performance and flow control mechanisms (e.g. credit based flow control) to control buffer management in virtual channels. Since bufferless routing eliminates buffers, there are no virtual channels and there is no need to manage buffer allocation/deallocation. This reduces complexity in the router and can enable router latency reductions (see Section 4).

Area saving: As we show in Section 7.7, removing buffers from routers can result in significant area savings.

Absence of Deadlocks: Deflection-based bufferless routing is free of deadlock. Since the number of input and output ports are the same, every packet that enters a router is guaranteed to leave it.

Absence of Livelock: One of the potential challenges in bufferless routing is livelocks that could arise if a packet continuously gets deflected. As shown in Section 3, the combination of oldest-first flit-ranking and port-prioritization is guaranteed to prevent livelocks in FLIT-BLESS, WORM-BLESS with and without buffers. In all cases, the proof is based on the invariant that the oldest flit in the network always makes forward progress.

Adaptivity: BLESS has the ability to be adaptive "on demand" to a certain degree. When there is no congestion, BLESS routes packets along shortest paths. In congested areas, however, the packets will be deflected away from local hotspots, which allows different links to be utilized and packets to be routed around congested areas. As such, BLESS automatically provides a form of adaptivity that buffered routing schemes must achieve using more sophisticated and potentially complex means.

For the same reason, BLESS can cope well with temporary bursty traffic. To a certain degree, the network itself (i.e., its links and routers) acts as a temporary buffer. In buffered routing, if a traffic burst occurs and many packets are sent to a router R, the buffers in routers close to R will gradually fill up. In BLESS, the packets are continuously deflected in the extended neighborhood of R, until the burst completes and they can gradually reach R.

5.2 Disadvantages

Increased Latency and Reduced Bandwidth: The key downside of bufferless routing is that it can increase average packet latency because deflected flits will take a longer path to the destination than necessary. Also, bufferless routing effectively reduces the available network bandwidth as all in-network packets always consume link resources. Hence, saturation is reached at lower injection rates compared to buffered routing. However, our evaluations show that for the kinds of low and moderate injection rates commonly seen in on-chip networks, the performance of BLESS is close to that of buffered routing. For such application domains, the advantages of BLESS can thus outweigh its disadvantages.

Increased Buffering at the Receiver: Since bufferless routing deflects individual flits, flits of a packet can arrive out-of-order and at significantly different points in time at the receiver (i.e. destination node). This likely increases the number of flits/packets that need to be buffered at the receiver side compared to baseline wormhole routing.[4] In addition, in-order delivery of packets requires buffering of packets that arrive out of order, both in bufferless and buffered routing. The increased receiver-side buffering requirements of BLESS

[4] Note that wormhole routing still requires buffers at the receiver because multiple virtual channels can interleave the delivery of their flits into the receiver and the receiver needs to buffer flits for different packets coming from each virtual channel.

Network Configuration 1 (sparse)	4x4 network, 8 cores, 8 L2 cache banks, each node is either a core or an L2 cache bank
Network Configuration 2 (dense)	4x4 network, 16 cores, 16 L2 cache banks, each node is a core and an L2 cache bank
Network Configuration 3 (sparse)	8x8 network, 16 cores, 64 L2 cache banks, each node is an L2 cache bank and may be a core
Processor pipeline	2 GHz processor, 128-entry instruction window (64-entry issue queue, 64-entry store queue), 12-stage pipeline
Fetch/Exec/Commit width	3 instructions per cycle in each core; only 1 can be a memory operation
L1 Caches	64K-byte per-core, 4-way set associative, 64-byte block size, 2-cycle latency, 32 MSHRs
L2 Caches	total 16MB, 16-way set associative, 64-byte block size, 12-cycle latency, 16 MSHRs per bank
DRAM controllers (on-chip)	4 channels; FR-FCFS; 128-entry req. buffer and 64-entry write data buffer each, reads prioritized over writes, XOR mapping [15]
DRAM chip parameters	Micron DDR2-800 [37], t_{CL}=15ns, t_{RCD}=15ns, t_{RP}=15ns, $BL/2$=10ns; 8 banks, 2K-byte row-buffer per bank

Table 3: Baseline CMP and memory system configuration for application simulations

and the additional logic to reorder flits reduces the energy reductions obtained by eliminating input buffers in routers. However, our evaluations show that the energy reduction due to eliminated router buffers outweighs the energy increase due to increased receiver-side buffers.

Header Transmission with Each Flit: FLIT-BLESS requires that header information be transmitted with each flit because each flit of a packet can follow a path that is different from another. This introduces additional overhead. In our design, we increase the width of the links to accommodate the additional header bits, which increases energy consumption of links. However, our energy consumption evaluation shows that energy reduction due to buffer elimination outweighs this additional source of energy consumption.

6. EXPERIMENTAL METHODOLOGY

We evaluate the performance and network energy-efficiency of bufferless routing techniques using a cycle-accurate interconnection network simulator. The different versions of BLESS (both flit-level and worm-based) routing is compared to three different baseline routing algorithms in terms of average/maximum packet delivery latency, saturation throughput, buffering requirements at the receiver, and network energy consumption: dimension-order routing (DO), an aggressive implementation of minimal adaptive routing (MIN-AD) [3], and a DO version of the ROMM algorithm (ROMM) [41].

6.1 Interconnection Network Model

The modeled network configuration is a two-dimensional mesh topology of varying size.[5] Each router has 5 input ports and 5 output ports, including the injection ports. Router latency is 2 cycles; link latency is 1 cycle. In our standard configuration, we assume that each link is 128-bit wide and each data packet consists of 4 flits, each of which is assumed to have 128 bits. Address packets are one flit long. All packets are fixed length. We model the overhead due to head-flits required by our mechanism. In the standard configuration, we consider routers with 4 virtual channels [10] per physical input port for the baseline algorithms. Each virtual channel is 4 flits deep.

6.2 Request Generation

We use a combination of real applications and synthetic traces to compare BLESS with the baseline algorithms. The main evaluation comparisons are conducted using SPEC benchmarks and desktop applications. Synthetic traces are primarily used for various sensitivity analyses, as well as for comparing the different BLESS algorithms among each other, with different buffer sizes.

Synthetic Traces: In the synthetic traces, each of the 8×8 routers is associated with a processor and the destination address of a packet is determined by the statistical process of the traffic pattern. We use four different traffic patterns: uniform random (UR), transpose (TR), mesh-tornado (TOR), and bit complement (BC) (see [45]). Each simulation experiment is run for 100,000 packet injections per processor.

Applications: We use multiprogrammed mixes of the SPEC CPU2006 benchmarks[6] and Windows Desktop applications (matlab, xml parser) for evaluation. Each benchmark was compiled using gcc 4.1.2 with -O3 optimizations and run for 150 million instructions chosen from a representative execution phase [43]. Due to shortcomings in our infrastructure (which cannot accurately model parallel applications), we do not simulate parallel applications, but we are investigating this in current work.

Application Simulation Methodology: To model real applications, we connect our cycle-accurate interconnection network simulator with a cycle-accurate x86 simulator. The functional frontend of the simulator is based on Pin [32] and iDNA [4]. We model the on-chip network and the memory system in detail, faithfully capturing bandwidth limitations, contention, and enforcing bank/port/channel/bus conflicts. Table 3 shows the major processor and memory system parameters. We model a static non-uniform cache architecture (S-NUCA) [26] where lower order bits in the address of the cache line determines which bank the cache line resides in. We model all traffic due to instruction and load/store data requests and replies. Note that our simulation infrastructure cycle-accurately models stalls in the network and limited buffering capacity of network packets within the processor (due to limited instruction window size, load-store queue size, and MSHRs [29]) and network interface. Therefore, the system is *self-throttling* as real systems are: if the buffers of a processor are full, the processor cannot inject more packets into the network until the stall conditions are eliminated. However, note that the processors we model are aggressive, which penalizes BLESS compared to buffered algorithms. Each core can support 32 outstanding L1 misses within the 128-entry instruction window, which is significantly larger than in existing processors (e.g., Pentium-4, can support only 8 outstanding L1 misses [5]).

Network Configurations: We repeat all our evaluations in 3 network configurations: 1) 4x4 network with 8 processors, 2) 4x4 network with 16 processors, and 3) 8x8 network with 16 processors, as shown in Table 3.

6.3 Network Energy Model

To evaluate energy consumption in the network, we use the energy model provided by the Orion simulator [51], assuming 70nm technology and 2GHz router at 1.0 V_{dd}. Link length of adjacent nodes is assumed to be 2.5mm.

We accurately model the energy consumption of additional hardware required by BLESS. In particular, we model the energy consumed by 1) additional buffers needed on the receiver side, 2) increased link width to transmit header information (we conservatively assume three extra bytes), and 3) the logic to reorder flits of individual packets in the receiver. We explicitly partition the network energy consumption into buffer energy, router energy and link energy in all our results (including their dynamic as well as static components). Buffer energy includes both the input buffers of routers and the receiver-side buffers needed to reorder packets for in-order delivery (or re-assembly in flit-level BLESS). Note that even the baseline DO and MIN-AD routing algorithms require receiver side buffering 1) because flits from the virtual channels arrive in an interleaved order and 2) to support in-order packet delivery. Router energy includes routing and arbitration energy, the latter of which we found to be negligible. Each of these can further be divided into static and dynamic components. In the case of buffer energy, dynamic buffer energy is consumed whenever a flit is written to or read from a buffer. Sta-

[5] We choose the 2-D Mesh for our investigations because this topology is simple and has been implemented in the on-chip networks of several large-scale chip multi-processor prototypes [53, 42, 24].

[6] 410.bwaves, 416.gamess, and 434.zeusmp are not included because we were not able to collect representative traces for them.

tic energy is dissipated because the many transistors constituting the buffers are imperfect, leaking current even when they do not switch.

6.4 Application Evaluation Metrics

Our main performance metrics for the application evaluations are energy consumption (taken over the entire execution of all applications) and system performance. Since we use multiprogrammed workloads, we evaluate the performance of different algorithm using two commonly used multiprogram performance metrics [13]. We measure system throughput using *Weighted-Speedup* [49, 13]:

$$WeightedSpeedup = \sum_i IPC_i^{shared}/IPC_i^{alone},$$

where IPC_i^{alone} and IPC_i^{shared} are the IPC of application i when running alone (without any interconnection network contention from any other application) and when running with the other applications, respectively. We also measure thread turnaround time using *Hmean-Speedup* [13], which balances fairness and throughput [33], but since measured hmean- and weighted-speedup results are qualitatively very similar in all our measurements, we mainly present weighted speedups for brevity. Finally, we measure fairness using the *unfairness index* proposed in [40, 16]. This is the ratio between the maximum and the minimum slowdown among all threads sharing the network. That is, $Slowdown_i = CPI_i^{shared}/CPI_i^{alone}$ and the *unfairness index* is defined as $\max_i Slowdown_i / \min_j Slowdown_j$.

7. EXPERIMENTAL EVALUATION

We start with a discussion of several case studies with varying application mixes in §7.1. In §7.2, we present aggregate results over all applications. §7.3–§7.6 present our results using synthetic traces. Unless otherwise noted, our results for BLESS assume a router latency of 2 cycles, i.e., *without* the optimization discussed in §4. In some cases, we write FLIT-1 and FLIT-2 (WORM-1 and WORM-2) to denote flit-based (or worm-based) BLESS with a router latency of 1 and 2, respectively. All network energy consumption results are normalized to that of the baseline DO routing algorithm, unless otherwise specified.

7.1 Application Evaluation – Case Studies

We distinguish between homogeneous and heterogeneous case studies. In the homogeneous case, all applications running on the different nodes are the same, whereas they are different in the heterogeneous case. In many of our evaluations, L2 cache banks are perfect (i.e., bank miss latency is zero). We do this to evaluate BLESS under the worst possible conditions for BLESS: perfect L2 caches increase the pressure of the applications on the on-chip network. We also present results with non-perfect L2 banks which show that BLESS performs even better in this realistic scenario.

7.1.1 Homogeneous Case Studies

We classify applications into different categories (heavy, medium, light), depending on how network-intensive they are, i.e., how much stress they put on the interconnection network. The stress an application puts on the network is proportional to its L1 cache miss rate since each L1 miss needs to access the network to be satisfied from an L2 bank. We pick one application from each group: matlab (heavy, L1 MPKI (Misses per 1000 retired instructions): 78.4), milc (medium, L1 MPKI: 32.4) and h264ref (light, L1 MPKI: 4.7) respectively. Matlab is the most network-intensive application in our suite. For each case, weighted speedup and normalized energy consumption are shown in Figs. 2 and 3, respectively. Each case study is conducted using the three network configurations shown in Table 3.

Performance: In all case studies and in all network configurations, the performance degradation due to not using buffers is relatively small. Several observations are in order:

- In the two sparser network configurations (Config 1 & 3: 4x4 with 8 processors and 8x8 with 16 processors), the maximum reduction in weighted speedup between the best baseline (typically MIN-AD) with buffers and the worst BLESS (WORM-2) is 0.5% (in the case of Matlab on Config 1). In all other case studies, the performance loss of BLESS compared to the best buffered algorithm is even smaller.
- In the dense network configuration (Config 2: 4x4 with 16 apps), the reduction in weighted (and similarly hmean) speed-up between MIN-AD (again, the best baseline algorithm) and the worst BLESS with router latency 2 (WORM-2) is 16.5% (Matlab), 4.6% (milc) and 0.1% (h264ref). Thus, the worst-case performance degradation with BLESS occurs in most network-intensive benchmarks with dense network configurations.
- If router latency of BLESS is reduced to 1 cycle (see Section 4), BLESS sometimes significantly outperforms the best baseline algorithm (by ∼10%). This is especially prevalent in the larger 8x8 network where router latency is more important for performance, as shown in results for matlab and milc. In smaller networks, the performance improvement due to reduced router latency is smaller. In Config 1, for instance, the performance of FLIT-1 exceeds FLIT-2 by 0.8% (Matlab), 0.9% (milc) and 0.02% (h264ref).
- FLIT-BLESS tends to achieve slightly better performance than WORM-BLESS. This is primarily due to the more efficient allocation of flits to output ports: the fraction of productively-routed flits is higher in FLIT-BLESS. In WORM-BLESS, if a header-flit is deflected, all (or many) of this packet's flits are deflected, whereas in FLIT-BLESS, later flits may be routed in productive directions thus reaching the destination more quickly.

It follows that using BLESS does not result in significant performance degradation compared to the best previous baseline, especially if the applications' memory intensity is not very high (milc, h264ref) or if the network configuration is sparse in terms of number of cores (Config 1 and 3). Performance degradation due to lack of buffering becomes noticeable only for very intense applications run on a dense network where every node is a core. Even then, weighted speedup reduces by at most 17.1% (leslie3d) over the best baseline.

The reason for BLESS's high performance is twofold. First, the injection rates of real applications are not high: in our workloads, on average an L1 miss packet is injected into the network 11.6 times every 1000 instructions. Second, and related to the first point, real systems are self-throttling. If a long-latency memory request is not satisfied, its core will block sooner or later, thereby not issuing any further requests, which reduces the network's injection rate.[7]

Network Energy: Fig. 3 shows the normalized energy consumption of the different algorithms. For all three benchmarks and for all network configurations, BLESS significantly reduces energy consumption. While link and router energy is slightly higher in BLESS due to deflections, buffer energy is reduced by almost an order of magnitude, since only the receive port needs buffers. The energy consumed by increased receiver-side buffering is significantly less than the energy consumed by eliminated router-based input buffering. Energy savings of WORM-2 over the best baseline in network configuration 1 are 34.0% (Matlab), 39.8% (milc) and 45.0% (h264ref). Results for Config 3 are similar. In h264ref, WORM-2 is more energy-efficient than FLIT-2 (45.0% vs. 38.3% energy reduction). This is because WORM reduces the fraction of head-flits compared to FLIT, where every flit needs to contain header information.

The energy savings of BLESS are smaller in the most network-intensive case study (Matlab on Config 2). The reason is twofold: 1) because traffic intensity is relatively high in this case, many flits are deflected to other routers, leading to an increase in both link and router energy, 2) the performance degradation caused by BLESS re-

[7] Notice that we observe this self-throttling behavior even though we assume a reasonably large, 128-entry instruction window size per core.

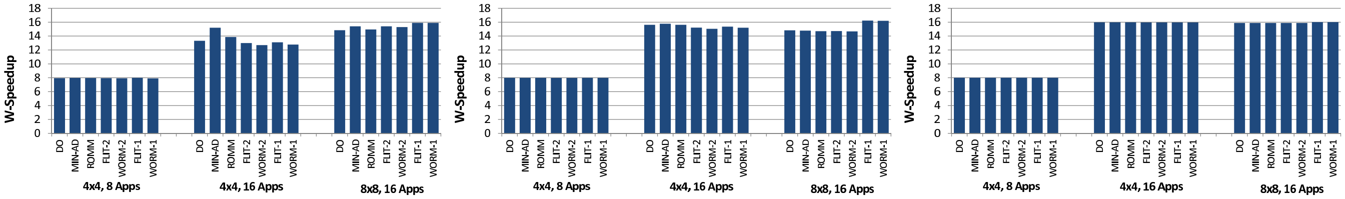

Figure 2: Homogeneous case studies–Weighted speedup: All applications are *matlab* (left), *milc* (middle), and *h264ref* (right).

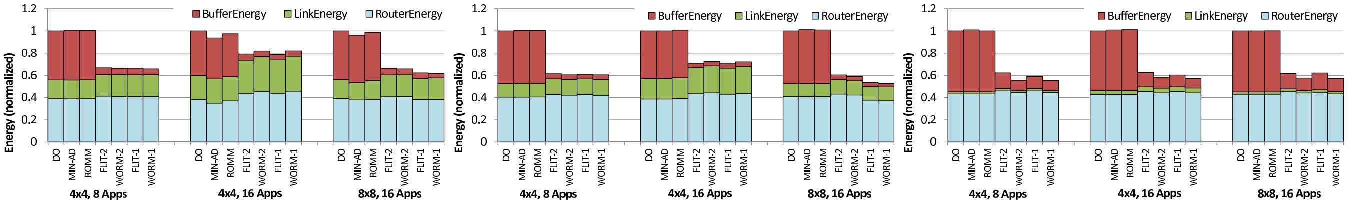

Figure 3: Homogeneous case studies–Network energy: All applications are *matlab* (left), *milc* (middle), and *h264ref* (right).

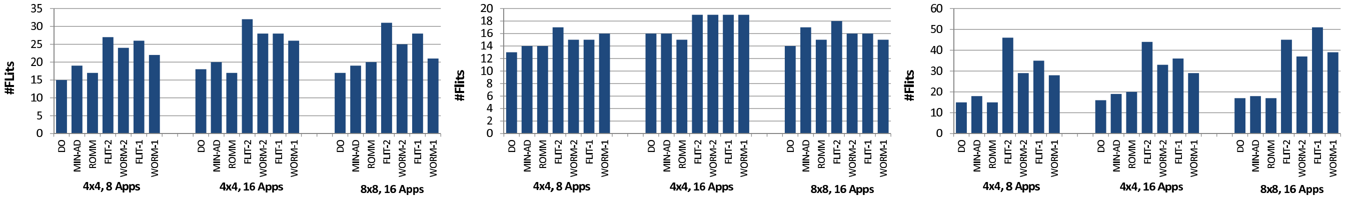

Figure 4: Homogeneous case studies–Maximum buffering requirement (in flits) at the receiver: *matlab* (left), *milc* (middle), and *h264ref* (right).

sults in increased static buffer energy. However, BLESS still reduces total NoC energy by 15.7% over the best baseline, MIN-AD.

Buffering Requirements at Receiver: BLESS increases buffering requirement at the receiver compared to the baseline because, due to deflections, 1) each flit of a packet is more likely to arrive at different points in time than the baseline wormhole routing (especially in flit-level BLESS) and 2) the variance in delivery times of packets increases, which increases buffering requirements to support in-order delivery. Fig. 4 shows the *maximum buffering requirement in flits at any receiver* for the three case studies. Notice that even though the buffering requirement at the receiver is larger in BLESS compared to the baseline algorithms, the energy results in Fig. 3 show that in terms of overall network energy consumption, this increase is overcome by the reduction in energy in the routers themselves. Further, note that in virtually all cases (e.g. matlab on Config 3), the receiver-side buffering requirement of WORM is less than FLIT. This is expected since WORM routes most packets (81% on average) as complete worms, reducing receiver-side buffering requirement. Finally, h264ref's buffering requirement is higher than others due to the processor-to-cache communication patterns that cause high congestion, increasing the variance of packet delivery times using BLESS.

Impact of Memory Latency: The results shown in Figs. 2–4 assume a memory configuration with perfect caches, i.e., every memory request to a cache in one of the routers is assumed to hit. As mentioned above, we use this model since it puts maximum stress on the interconnection network (and is therefore maximally challenging for BLESS). With realistic, non-perfect caches, the performance degradation of BLESS compared to the baseline algorithms is smaller. This is because the occurrence of cache misses adds DRAM latency to the application's execution time, which in turn reduces the stress put on the on-chip cache-to-cache network. Fig. 5 shows the most-intensive case study (Matlab) with realistic caches. Compared to perfect caches, Matlab's performance degradation caused by BLESS significantly reduces (1.5% instead of 16.5% weighted speedup reduction in the dense network configuration), whereas its network energy reduction further increases (33.7% vs 15.7%). We conclude that with realistic caches BLESS performs very similarly to the best baseline algorithm with buffers, while reducing energy consumption significantly.

7.1.2 Heterogeneous Case Studies

Fig. 6 shows an application mix consisting of the most network-intensive applications in our suite. We also evaluated two application mixes, each of which consists of a mix of intensive and non-intensive applications. Fig. 7 shows the results for the more network-intensive of these mixes.

- Similarly to the homogeneous case studies, there is only one experiment in which BLESS causes a performance degradation of more than 3% compared to the baseline. It is the case with all memory-intensive applications running on the dense network configuration 2. The reduction in weighted speedup in this case is 16.3% compared to MIN-AD routing. Reducing router latency with BLESS largely improves performance in the larger network.
- BLESS reduces network energy considerably in all cases. In Config 1 & 3, energy reduction is between 20% and 39% in all workloads. In the dense Config 2, the reduction is between 10% (Mix 1) and 31% (Mix 2).
- Of particular interest in mixes with applications of different characteristics is the question of fairness, i.e., whether all applications experience an equal slowdown due to the contention in the network. Figs. 6 and 7 (right) show that using BLESS does not significantly increase unfairness compared to baseline. In most cases, the UnfairnessIndex achieved by BLESS and baseline are within a few percent. The only exception is Mix 1 run on the dense configuration: in this case, the increase in unfairness due to BLESS is about 18% due to a large number of deflections experienced by the less network-intensive applications.

We conclude that BLESS is effective at handling mixes of different applications efficiently and fairly.

7.2 Application Evaluation – Aggregate Results

Fig. 8 shows performance and normalized network energy of all 29 applications in the network configuration 1 and with perfect caches (which puts additional stress on the network). The rightmost bars show the geometric mean across all applications. It can be seen that BLESS decreases average (worst-case) performance by only 0.5% (3.2%), whereas it reduces average (worst-case) network energy consumption by 39.4% (28.1%).

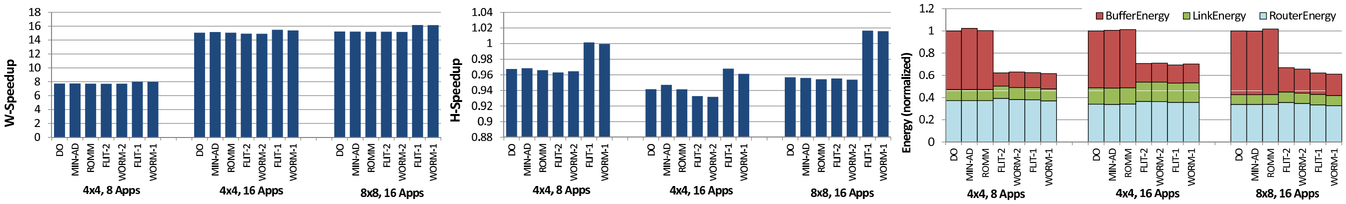

Figure 5: Homogeneous case study 1 (Matlab), with realistic, non-perfect caches that reduce the stress on the network: weighted speedup (left), hmean speedup (middle), and network energy (right).

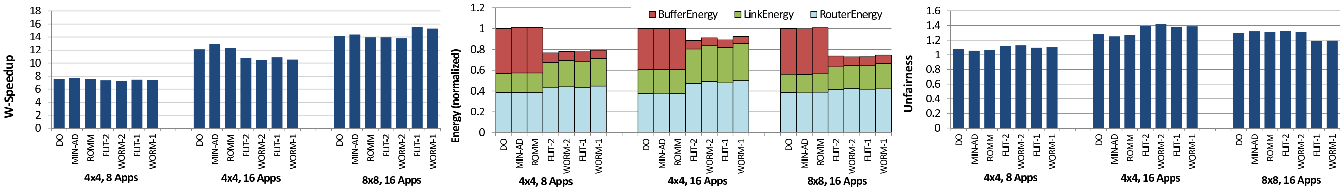

Figure 6: Heterogeneous Mix 1–matlab, mcf, libquantum, leslie3d: weighted speedup (left), network energy (middle), and unfairness (right).

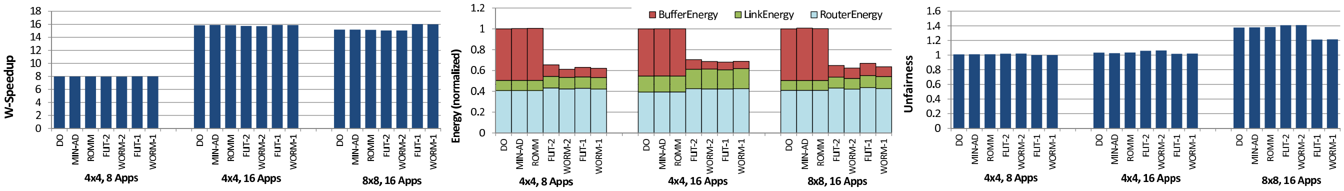

Figure 7: Heterogeneous Mix 2–libquantum, h264ref, astar, omnetpp: Weighted speedup (left), network energy (middle), and unfairness (right) .

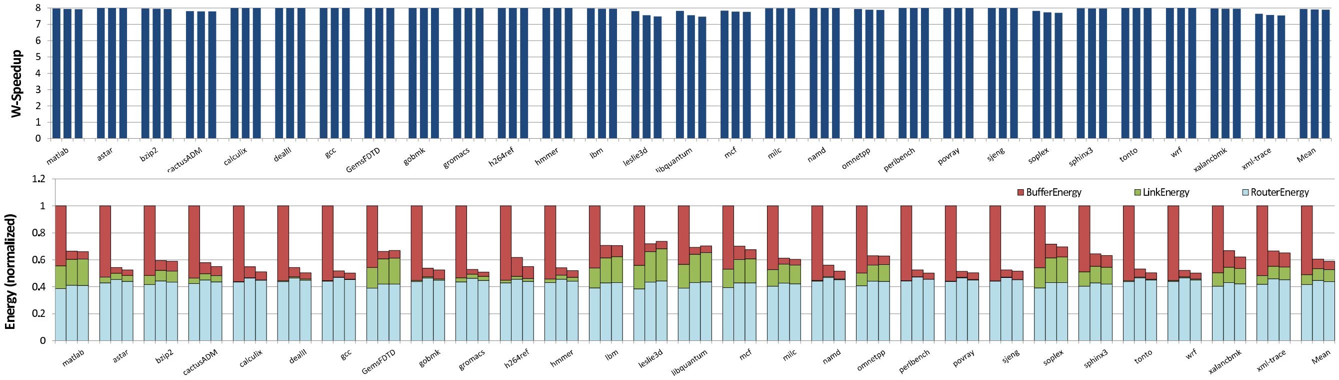

Figure 8: Performance and network energy of all applications in network configuration 1. Router latency for BLESS is 2. Weighted speedup (top) and energy (bottom). Bars from left to right for each application correspond to Best Baseline Algorithm (generally MIN-AD), FLIT-2, and WORM-2.

Table 4 summarizes the average (across all 29 applications) and worst-case performance and energy impact of BLESS on the sparse network Config 1 and the dense network Config 2. For example, with realistic L2 caches in Config 2, the average (worst-case) performance degradation becomes very small, 0.65% (1.53%), whereas average (worst-case) energy reduction increases to 42.5% (33.7%). We conclude that averaged over a very wide range of applications, BLESS achieves significant energy savings at negligible loss in performance.

Network Config 1	**Perfect L2**		**Realistic L2**	
	Average	Worst-Case	Average	Worst-Case
Δ Network Energy	-39.4%	-28.1%	-46.4%	-41.0%
Δ System Performance	-0.5%	-3.2%	-0.15%	-0.55%

Network Config 2	**Perfect L2**		**Realistic L2**	
	Average	Worst-Case	Average	Worst-Case
Δ Network Energy	-32.8%	-14.0%	-42.5%	-33.7%
Δ System Performance	-3.57%	-17.1%	-0.65%	-1.53%

Table 4: Average and worst-case (in all cases either leslie3d or Matlab) network energy and system performance of FLIT-BLESS vs. best previous buffered routing algorithm (generally MIN-AD): Config 1 (top) and 2 (bottom). Perfect caches (left) and realistic, non-perfect caches (right). Router latency is 2 cycles.

7.3 Synthetic Traces – Traffic Patterns

As discussed, a key characteristic of real CMP systems running real applications is that they are self-throttling, preventing the injection rates into the interconnection network from becoming very high over an extended period of time. Synthetic traces do not have such a self-throttling mechanism and it is therefore possible to study the behavior of routing algorithms at injection rates much higher than typically experienced in real networks. Hence, synthetic traces are used to investigate the *saturation points* of different routing algorithms.

Fig.9 shows average (top row) and maximum (bottom row) packet latency of BLESS algorithms compared to the best buffered baseline. Several comments are in order. First, at low packet injection rates, bufferless routing provides similar average packet latencies as the baseline algorithms. For example, for the UR traffic pattern, bufferless routing increases the average packet latency by less than 10% even with a large injection rate of 0.3. For maximum packet latencies, we observe a similar behavior, albeit for slightly lower injection rates. On the other hand, baseline routing algorithms with buffers can clearly withstand higher injection rates than BLESS, i.e., buffered networks achieve higher saturation throughput. This is because, at very high network utilization, bufferless routing wastes significant network bandwidth by causing too many deflections. Specifically,

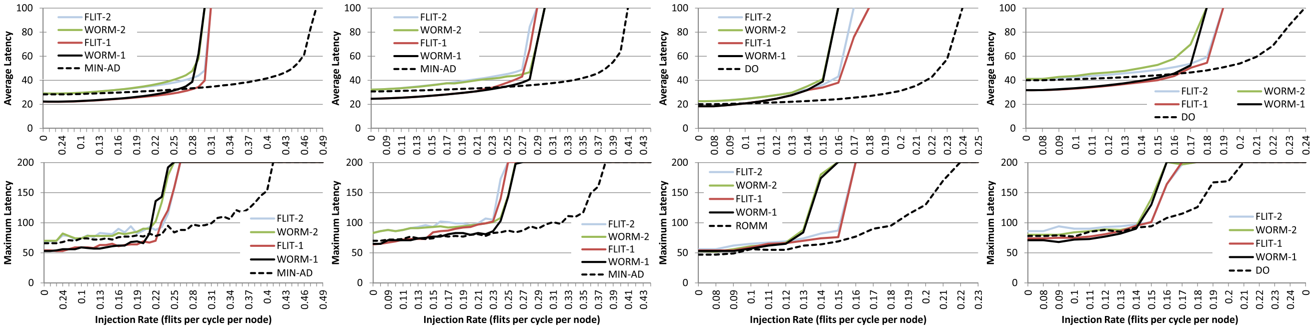

Figure 9: Average latency (top) and maximum latency (bottom) as a function of injection rate. From left to right: a) UR, b) TR, c) TOR, d) BC. Dashed lines are the best buffered baseline.

compared to the best baseline algorithm, the maximum sustainable injection rate of the best BLESS algorithm is smaller by 35% (UR), 26% (TR), 29% (TOR), and 20% (BC). For traffic patterns with significant contention (BC, TR), BLESS has relatively closer saturation throughput to the best baseline than for the purely random, well-load-balanced traffic pattern (UR). The reason is the inherent adaptivity of BLESS: flits get deflected from congested areas, which can help in circumventing hotspots. The results further show that:

1. For low injection rates, BLESS with 1-cycle router latency achieves better latencies than buffered baselines.

2. For TR traffic, for instance, BLESS has smaller saturation throughput than MIN-AD, but has much higher saturation throughput than the non-adaptive dimension-order routing. This is because BLESS provides adaptivity without requiring any explicit information about congestion. BLESS performs in-between the non-adaptive DO and the adaptive algorithms because it allows packets to avoid congested paths by deflecting them toward other parts of the network.

Hence, BLESS routing achieves good performance and can reduce congestion in on-chip networks if network utilization is not too high.

7.4 Synthetic Traces – Baseline Configurations

So far, we have studied the baseline algorithms in their standard configuration. In Figure 10 (top), we investigate the performance and network energy consumption of DO routing with varying buffer sizes. The left figure shows how the maximum sustainable injection rate drops significantly as the size of buffers in each router is reduced. With a single two-flit deep virtual channel, only an injection rate of 0.1 flits/cycle is sustainable, which is significantly smaller than the 0.3 flits/cycle sustainable with BLESS (shown in Fig. 9).

Fig. 10 (top-right) shows energy normalized to that of BLESS. Energy consumption of the baseline algorithm can be reduced somewhat by reducing buffer sizes compared to our standard configuration. However, notice that even if we pick the energy-optimal buffer configuration for the baseline, for each injection rate, energy savings would be significantly lower compared to BLESS. For example, using the buffered DO routing scheme with one 2-flit deep VC per input port consumes 10% more energy than BLESS while providing *significantly* worse latency and saturation throughput (compare DO in Fig. 10 (top-left) and WORM-2 in Fig. 9 (top-left)).

7.5 Synthetic Traces – BLESS with Buffers

As described, it is possible to use BLESS with buffers, too. Figure 10(bottom-left) shows that while the saturation throughput of BLESS does increase with increasing buffer size, the marginal improvement is rather small. While pure bufferless BLESS sustains a maximum injection rate of 0.3, BLESS with a single 2-flit-deep (4-flit-deep) virtual channel achieves 0.33 (0.35), respectively. Beyond this, performance increases no further.

Fig. 10(bottom-right) shows that the energy optimal buffer size of BLESS changes with the injection rate. The figure also provides insight into the behavior of BLESS by showing the breakdown of normalized network energy for different injection rates. At low injection rates, BLESS eliminates the energy consumed by buffers without significantly increasing the energy consumption in the links and the crossbar (including arbiter and routing energy). As the injection rate increases, BLESS causes an increase in link and crossbar energy consumption compared to the baseline because congestion in the network causes more deflections to happen and more routers and links to be utilized by the deflected packets.

7.6 Synthetic Traces – BLESS Alternatives

So far, we have shown results only for Oldest-First BLESS (FLIT-OF and WORM-OF). For the sake of completeness, Fig. 11 shows that both FLIT-OF and WORM-OF consistently achieve the lowest average latency compared to other arbitration policies discussed in §3. This advantage of oldest-first is particularly pronounced for maximum latency. In both flit-level and worm-level BLESS, the maximum latency achieved by oldest-first is significantly lower than for the other schemes. At injection rate 0.24, for instance, WORM-OF's maximum latency is less than half of the maximum latency achieved by any other scheme. The figure also shows that OF achieves the lowest average number of deflections per packet, which is important since a low deflection rate implies higher energy efficiency. We conclude that OF arbitration policy outperforms alternative policies.

7.7 Effect on Buffer and Link Area

We express the area occupied by buffers in terms of flits using the following first-order model:

$$BufferArea = (IBPR + RSBPR) \cdot NumberOfRouters,$$

where $IBPR$ and $RSBPR$ are the number of input buffers and receiver side buffers per router, respectively. BLESS eliminates input buffers per router while increasing receiver-side buffers. Based on our simulations on Config 1 with all applications, we found that BLESS increases the receiver-side maximum buffering requirement over the buffered DO algorithm from 16 to 38 flits. However, BLESS eliminates all input buffers (note that pipeline latches stay the same between DO and BLESS). Consequently, we found that BLESS reduces the buffer area requirements by 60.4% compared to our DO baseline. As buffers were shown to occupy 75% of the total on-chip network area [22] in the TRIPS chip, this simple analysis suggests that large area savings are possible with BLESS. BLESS does increase the area occupied by links in order to accommodate header information. Even with a conservative estimate of 3-byte increase in our 16-byte links, the link area increases by 18.75% to a first order. However, since link area is not a significant fraction of the NoC area [22], we expect BLESS to provide significant area savings.

8. RELATED WORK

To our knowledge, this is the first work that proposes a wide variety of routing algorithms for bufferless routing in on-chip networks and thoroughly evaluates the energy, application performance, latency, throughput, and area characteristics of bufferless routing in on-

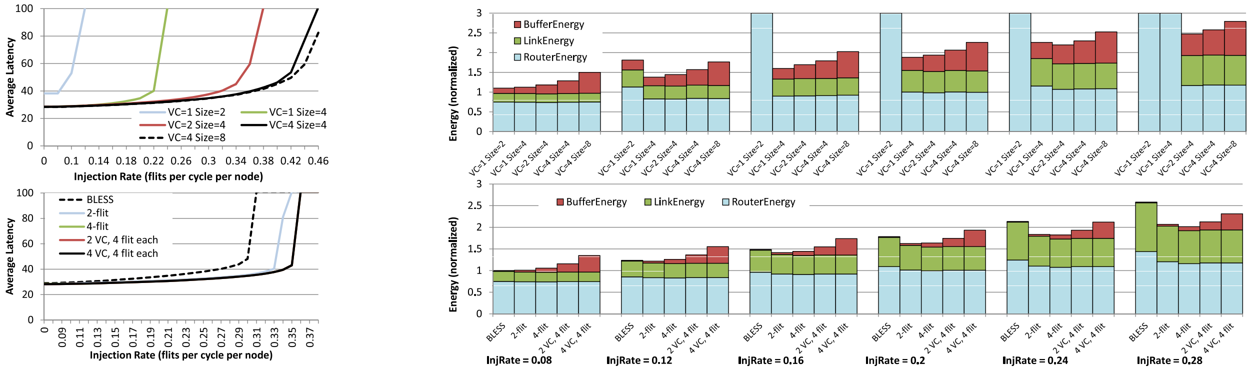

Figure 10: Impact of buffer size on performance and energy consumption of baseline (top row) and BLESS routing (bottom row) for UR traffic: Average latency (left) and energy (right). For ease of comparison, all energy results are normalized to bufferless BLESS at injection rate 0.08 (shown in leftmost bars in the right figure of bottom row).

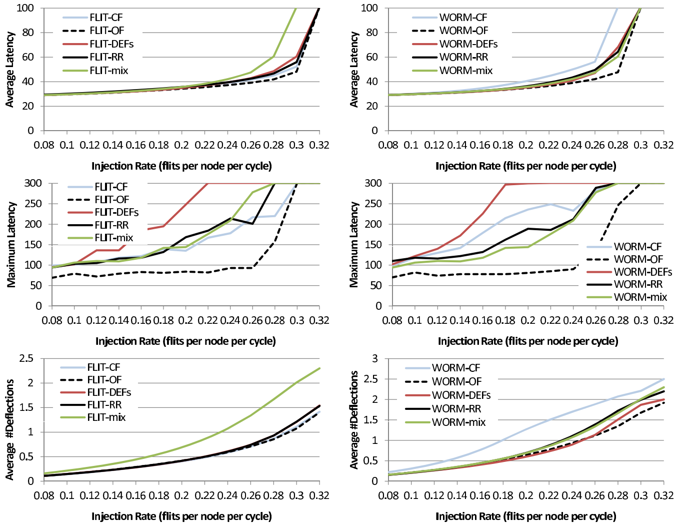

Figure 11: Comparison of different BLESS algorithms. Flit-based (left column) and worm-based algorithms (right column). In each case, from top to bottom: average latency, maximum latency, and average number of deflections per packet.

chip networks. This work is also the first to 1) describe how to take advantage of eliminated buffers in NoCs to reduce router latency and to simplify NoC design, 2) provide an application-level evaluation of bufferless NoCs, and 3) combine deflection routing with wormhole routing algorithms. This section briefly surveys related work.

Hot-potato and deflection routing in large scale systems: Hot-potato routing was first proposed by Baran [2] for distributed communication networks. Several massively parallel machines, such as the HEP [46, 47], the Tera [1], and the Connection Machine [23] have used deflection routing to connect different chips. These techniques are not disclosed in detail and, to our knowledge, have not been publicly evaluated in terms of energy consumption or performance. Some of these deflection routing algorithms do not eliminate buffers [48]. Moreover, their application was to large scale off-chip networks with large path diversity and long link latencies rather than on-chip networks with short link latencies. The Chaos router [28] uses a form of deflection routing when a node is congested, however it still buffers packets in the router. Our main contributions beyond these works are: 1) we propose specific, new algorithms for bufferless routing in on-chip networks, 2) we provide thorough, application-level energy and performance evaluations of bufferless on-chip routing, which were not available previously, 3) we show how in absence of router input buffers the router and network design can be optimized for reduced latency and simplicity.

Theoretical studies of hot-potato routing: In the theory community, there has been work studying algorithms and models for hot-potato routing, e.g. [14]. Most of these algorithms are *static*, i.e., all packets are assumed to be injected at time zero, and the analysis examines the time needed to deliver the packets. The dynamic analysis in [8] does not investigate average performance or energy efficiency.

Deflection routing in on-chip networks: Several recent studies [38, 31, 19, 18] examined the use of bufferless deflection routing in on-chip networks. [19, 18] require packet dropping when congestion arises, a complexity that is not present in our techniques. These previous studies mainly consist of evaluation of deflection routing and packet dropping algorithms on the performance of synthetic workloads. As such, they do not evaluate 1) the energy consumption, routing latency, and area benefits of deflection routing, 2) effect on real applications. In contrast, our work extends the state-of-the-art in NoC deflection routing the following ways: 1) it provides new routing algorithms to combine wormhole routing with deflection routing, 2) it examines energy, performance, and router latency benefits of bufferless routing, 3) it provides extensive evaluations with real applications and a realistic system simulator that models the self-throttling nature of CMP NoCs. The techniques proposed to reduce deflections or enable dropping in [31, 19] are orthogonal to our design: BLESS can be combined with them to further improve performance, at the expense of extra energy consumption.

Deflection routing in optical networks: Recently, deflection routing has found popular use in optical transmission networks [52, 7]. Deflection routing reduces the need for optical buffering, which is expensive. Optical networks have significantly different energy and performance characteristics than the electrical networks we examine.

Packet dropping: Flow control techniques have been proposed to reduce buffer sizes by dropping packets/flits if not enough buffer slots are available [12, 19, 18]. These techniques require retransmission of packets/flits by the source node. As such, they are likely to be more complex than bufferless routing. However, the concept of packet/flit dropping can be used in a bufferless network to reduce congestion when too many packets are being deflected.

Reducing buffering requirements in on-chip networks: Techniques have been proposed to reduce buffering in routers by pre-configuring routes [36]. These techniques usually add extra complexity to the router pipeline whereas bufferless routing reduces the complexity in the router pipeline. Circuit switching [12] can also remove/reduce buffering requirements but it comes at the cost of connection setup/teardown overheads that are not present in our proposal. Recently-proposed elastic-buffer flow control mechanism [35] uses the pipeline flip flops in router channels for packet storage, effectively using channels as distributed FIFO queues. As such, this mechanism eliminates input virtual channels and reduces router complexity. However, since it does not employ deflection routing, it requires multiple physical channels to avoid deadlock. The benefits of both BLESS and elastic-buffer flow control can be increased by combining the algorithms proposed in this paper with elastic buffering.

Adaptive Routing: Bufferless routing provides some of the benefits of adaptive routing by naturally routing packets around congested areas in the network. We compare BLESS in terms of energy and performance with two adaptive routing algorithms [3, 41] and show that they perform similarly on real applications.

9. CONCLUSIONS & FUTURE WORK

We make the case that bufferless routing could be used beneficially in on-chip interconnection networks. We show that, by getting rid of router input/output buffers, significant energy reductions can be achieved at modest performance loss compared to buffered routing algorithms, as long as the volume of injected traffic is not extremely high, which is the case with most real applications. Bufferless routing can also enable lower router latency, which can result in increased performance. We believe that bufferless routing algorithms, which also simplify network and router design by eliminating complex buffer management/allocation techniques, could thus be the method of choice for interconnection networks that are known to run at below-peak throughput most of the time. Our bufferless network design lacks many functionalities that have been developed for buffered networks, including support for starvation freedom/avoidance, QoS and different traffic service classes, fault tolerance in the presence of faulty links/routers, congestion awareness [21], and energy management. Our future work will focus on incorporating such support into the design of bufferless routing algorithms.

ACKNOWLEDGEMENTS

We thank Xuehai Qian for his help in bringing up the network energy modeling infrastructure. We thank Reetuparna Das, Eiman Ebrahimi, Boris Grot, Jim Larus, Kevin Lepak, Burton Smith, and the anonymous reviewers for their comments on earlier drafts of this paper.

REFERENCES

[1] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The Tera computer system. In *ICS*, 1990.

[2] P. Baran. On distributed communications networks. *IEEE Trans. on Communications*, Mar. 1964.

[3] P. E. Berman, L. Gravano, G. D. Pifarre, and J. L. C. Sanz. Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks. *IEEE TPDS*, 12(5), 1994.

[4] S. Bhansali, W.-K. Chen, S. D. Jong, A. Edwards, R. Murray, M. Drinic, D. Mihocka, and J. Chau. Framework for instruction-level tracing and analysis of programs. In *VEE*, 2006.

[5] D. Boggs et al. The microarchitecture of the Intel Pentium 4 processor on 90nm technology. *Intel Technology Journal*, 8(1), Feb. 2004.

[6] S. Borkar. Thousand core chips: A technology perspective. In *DAC*, 2007.

[7] S. Bregni and A. Pattavina. Performance evaluation of deflection routing in optical ip packet-switched networks. *Cluster Computing*, 7, 2004.

[8] C. Busch, M. Herlihy, and R. Wattenhofer. Routing without flow control. In *SPAA*, 2001.

[9] S. Cho and L. Jin. Managing distributed, shared L2 caches through OS-level page allocation. In *MICRO*, 2006.

[10] W. J. Dally. Virtual-channel flow control. In *ISCA-17*, 1990.

[11] W. J. Dally and C. L. Seitz. The torus routing chip. *Distributed Computing*, 1:187–196, 1986.

[12] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[13] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 28(3):42–53, 2008.

[14] U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In *STOC*, 1992.

[15] J. M. Frailong, W. Jalby, and J. Lenfant. XOR-Schemes: A flexible data organization in parallel memories. In *ICPP*, 1985.

[16] R. Gabor, S. Weiss, and A. Mendelson. Fairness and throughput in switch on event multithreading. In *MICRO-39*, 2006.

[17] M. Galles. Spider: A high-speed network interconnect. *IEEE Micro*, 17(1):34–39, 2008.

[18] C. Gomez, M. E. Gomez, P. Lopez, and J. Duato. A bufferless switching technique for NoCs. In *Wina*, 2008.

[19] C. Gomez, M. E. Gomez, P. Lopez, and J. Duato. Reducing packet dropping in a bufferless NoC. In *Euro-Par*, 2008.

[20] M. K. Gowan, L. Biro, and D. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. In *DAC*, 1998.

[21] P. Gratz, B. Grot, and S. W. Keckler. Regional congestion awareness for load balance in networks-on-chip. In *HPCA-14*, 2008.

[22] P. Gratz, C. Kim, R. McDonald, S. W. Keckler, and D. Burger. Implementation and evaluation of on-chip network architectures. In *ICCD*, 2006.

[23] W. D. Hillis. *The Connection Machine*. MIT Press, 1989.

[24] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-ghz mesh interconnect for a teraflops processor. *IEEE Micro*, 27(5), 2007.

[25] N. D. E. Jerger, L.-S. Peh, and M. H. Lipasti. Circuit-switched coherence. In *NOCS*, 2008.

[26] C. Kim, D. Burger, and S. Keckler. An adaptive, non-uniform cache structure for wire-dominated on-chip caches. In *ASPLOS-X*, 2002.

[27] J. Kim, J. D. Balfour, and W. J. Dally. Flattened butterfly topology for on-chip networks. In *MICRO*, 2007.

[28] S. Konstantinidou and L. Snyder. Chaos router: architecture and performance. In *ISCA*, 1991.

[29] D. Kroft. Lockup-free instruction fetch/prefetch cache organization. In *ISCA-8*, 1981.

[30] A. Kumar, L.-S. Peh, and N. K. Jha. Token flow control. In *MICRO-41*, 2008.

[31] Z. Lu, M. Zhong, and A. Jantsch. Evaluation of on-chip networks using deflection routing. In *GLSVLSI*, 2006.

[32] C.-K. Luk et al. Pin: Building customized program analysis tools with dynamic instrumentation. In *PLDI*, 2005.

[33] K. Luo, J. Gummaraju, and M. Franklin. Balancing throughput and fairness in SMT processors. In *ISPASS*, 2001.

[34] M. M. K. Martin et al. Timestamp snooping: An approach for extending smps. In *ASPLOS-IX*, 2000.

[35] G. Michelogiannakis, J. Balfour, and W. J. Dally. Elastic-buffer flow control for on-chip networks. In *HPCA-15*, 2009.

[36] G. Michelogiannakis, D. Pnevmatikatos, and M. Katevenis. Approaching ideal NoC latency with pre-configured routes. In *NOCS*, 2007.

[37] Micron. *1Gb DDR2 SDRAM Component: MT47H128M8HQ-25*, May 2007. http://download.micron.com/pdf/datasheets/dram/ddr2/1GbDDR2.pdf.

[38] M. Millberg, R. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *DATE*, 2004.

[39] R. Mullins, A. West, and S. Moore. Low-latency virtual-channel routers for on-chip networks. In *ISCA-31*, 2004.

[40] O. Mutlu and T. Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *MICRO-40*, 2007.

[41] T. Nesson and S. L. Johnsson. ROMM: Routing on mesh and torus networks. In *SPAA*, 1995.

[42] J. D. Owens, W. J. Dally, R. Ho, D. N. Jayashima, S. W. Keckler, and L.-S. Peh. Research challenges for on-chip interconnection networks. *IEEE Micro*, 27(5), 2007.

[43] H. Patil et al. Pinpointing representative portions of large Intel Itanium programs with dynamic instrumentation. In *MICRO-37*, 2004.

[44] L.-S. Peh and W. J. Dally. A delay model and speculative architecture for pipelined routers. In *HPCA-7*, 2001.

[45] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles. GOAL: A load-balanced adaptive routing algorithm for torus networks. In *ISCA*, 2003.

[46] B. J. Smith. A pipelined shared resource MIMD computer. In *ICPP*, 1978.

[47] B. J. Smith. Architecture and applications of the HEP multiprocessor computer system. In *Proc. of SPIE*, 1981.

[48] B. J. Smith, Apr. 2008. Personal communication.

[49] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous mutlithreading processor. In *ASPLOS-IX*, 2000.

[50] M. B. Taylor et al. Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for ILP and streams. In *ISCA-31*, 2004.

[51] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *MICRO*, 2002.

[52] X. Wang, A. Morikawa, and T. Aoyama. Burst optical deflection routing protocol for wavelength routing WDM networks. In *SPIE/IEEE Opticom*, 2004.

[53] D. Wentzlaff et al. On-chip interconnection architecture of the Tile processor. *IEEE Micro*, 27(5), 2007.