Throttling Control for Bufferless Routing in On-Chip Networks

Throttling Control for Bufferless Routing in On-Chip Networks

Yicheng Guan ‡, CISSE AHMADOU DIT ADI‡, Takefumi Miyoshi‡, Michihiro Koibuchi†, Hidetsugu Irie‡ and Tsutomu Yoshinaga‡
‡ The University of Electro-Communications,1-5-1, Chofugaoka, Chofu-shi, Tokyo, Japan 182-8585
Email:{guan, ahmadou, miyoshi, yosinaga}@comp.is.uec.ac.jp
† National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, Japan 101-8430
Email: koibuchi@nii.ac.jp

***Abstract*—As the number of core integration on a single die grows, buffers consume significant energy, and occupy chip area. A bufferless deflection routing that eliminates router's input-port buffers can considerably help saving energy and chip area while providing similar performance of existing buffered routing, especially for low-to-medium network loads. However when congestion increases, the bufferless frequently causes flits deflections, and misrouting leading to a degradation of network performance. In this paper, we propose IRT(Injection Rate Throttling), a local throttling mechanism that reduces deflection and misrouting for high-load bufferless networks. IRT provides injection rate control independently for each network node, allowing to reduce network congestion. Our simulation results based on a cycle-accurate simulator show that using IRT, IRT reduces average transmission latency by 8.65% compared to traditional bufferless routing.**

I. Introduction

In recent years, significant work has examined novel designs for on-chip networks different from traditional buffered virtual channel (VC) routers in order to gain performance. In particular, recent works proposed a bufferless routing [1–3], in which the input port buffers are eliminated and router port contention is mitigated by temporal deflections. This novel architecture yields significant network power saving with minimal performance loss under low network workloads. However, generally speaking, bufferless routing performs significantly worse than a traditional buffered routing with high network workloads; it is clear that our key challenge is performance issue under the network-intensive applications.

A straight-forward approach to mitigate this problem is to take advantage of buffered routing only when network is highly loaded. S. Jafri et al. proposed in [4] an adaptive flow control (AFC) which adaptively switches routing mode. AFC combines a bufferless (backpressureless) router, and buffered (backpressure) router , and dynamically switches between the two modes depending on network load. AFC can gain the energy efficiency of bufferless routing at low network load, and the performance of buffered routing at high network load. When running at high load, AFC will switch to buffered mode, increasing network capacity and requiring power consumption similar to conventional buffered networks. Although AFC can enjoy the power and energy benefits of bufferless routing at low network load, as it incorporates both bufferless and buffered routing side-by-side, it imposes an area penalty because buffers remain present. Another approach is to reduce the deflection rate of bufferless routing under high network workloads.

Injection throttling has been proposed for buffered network in [5] to prevent network saturation at high loads using global network congestion information (buffers occupancy) and adaptively decide injection or not of packets. Throttling based on application has been proposed also for bufferless using global information in [6, 7] to improve the performance of bufferless routing under high network workloads.

In this paper, we propose a new injection rate throttling mechanism (IRT) for bufferless routing. IRT imposes the injection limitation based on only the congestion at local node, that allows control of packet injection individually needless to global network congestion information, meaning injection controller in each node can locally throttle new packet injection operations. As new packet injection may impose different effect on the global network congestion, they are treated differently depending on local node's load. In one hand nodes that tend to further increase network congestion are not allowed to inject a new packet until a certain reduction in node activity (number of flit traversing the local router). On the other hand, no delay is added to inject a new packet when the local network load is low (only few flits transverse the local router in each cycle). IRT is expected to bring the performance of bufferless networks closer to buffered networks while keeping the benefits (area, power consumption) of bufferless network. The contribution and our finding of this work is as follows:

- We propose a simple and local self-throttling algorithm for bufferless networks. Each router adopts dynamically different injection rate for data communication according to the node's impact on the global network congestion status.
- We estimate the network performance by evaluating the number of misrouting in communications. We simulate bufferless networks w/wo injection throttling under intensive benchmarks using a cycle accurate many-core processor simulator (SimMc) [8].
- We show how bufferless routing performance can be

improved under high network workloads. By using IRT, we are able to outperform conventional bufferless networks. Results show considerably gain in energy saving due to the reduction of deflection and misrouting in communications.

The rest of this paper is organized as follows: In Section II we summarize the necessary background and related works. Section III presents our novel IRT scheme for bufferless networks. Section IV describes our reassembly buffer solution for bufferless networks. In section V, we discuss our evaluation methodology. Section VI presents our evaluation and simulation results. And finally, we conclude this paper in section VII.

II. Background and Related Works

Bufferless deflection routing was first proposed as "hot-potato" routing for off-chip networks [9]. Recently, it has been proposed also for on-chip networks mainly for two reasons: It reduces network hardware cost (no input port buffers), and its design simplicity. Several prototype many core systems point toward this trend. In MIT RAW, interconnect consumes $\sim$40% of overall system power [10]; in the Intel Terascale chip, 30% [11]. Buffers consume a significant portion of this power. In [12] baseline buffurless deflection routing (BLESS) reduced network energy by 40% by eliminating the input port buffers. Bufferless routers require only some pipeline registers, a crossbar, and an arbitration logic allowing a significant energy and area saving compared to buferred routers.

In BLESS, two bufferless routing schemes have been proposed as FLIT-BLESS and WORM-BLESS. In FLIT-BLESS, each flit of a packet contains header information and can travel independently in the network. At each router, incoming flits contend for output ports. When two flits contend for the same output port, BLESS avoids the need for buffering by misrouting one of the flit (the newest) to another free output port. The flits continue through the network until they arrive to the destination. Because FLIT-BLESS requires routing information for each flit, it imposes an important overhead to the packet. To reduce packet overhead WORM-BLESS has been proposed. In WORM-BLESS, routing information is added to each worm that may consists of several flits. In this paper, we adopt WORM-BLESS routing in our simulations.

In bufferless routing, flits cannot be buffered at input ports. Routers always forward incoming flits to an available output port. To guarantee livelock freedom, BLESS uses an Oldest-First priority rule. Flits arbitration is based on a timestamp. The oldest flit is prioritized for a productive output port[1]. Hence no livelock can occur because once a flit is the oldest flit in the network, it cannot be deflected anymore and makes forward progress until it arrives to the destination. We also use Oldest-First policy in our simulation. BLESS allows injection of a new flit whenever at least one input port is free. It guarantees that all flits entering a router can leave it because there are as many output ports as input ports.

[1]an output port towards the destination.

Flits or truncated worms from a packet may arrive in out-of-order at destination in bufferless deflection routing; therefore, it is necessary to reconstruct the original packet before ejection. The buffers allocated for this process are called reassembly buffers. In [3], the authors propose a combination of flow control (Retransmit-Once) and cache memory protocol to provide a network level packet reassembly. The cache protocol's cache miss buffers are used to handle packets reassembly. In our case, as in BLESS [12], we assume an infinite receiver buffers without flow control. We set the receiver buffer unit size to be equal to a worm-buffer. A worm-buffer is set free when a reassembly is completed. In the locally busy areas, our throttling mechanism delay further new packet injection. This can reduce the network deflection/misrouting rate, thus avoiding excessive out-of-order delivery and the required large reassembly buffer space.

By eliminating input port buffers, bufferless on-chip networks help saving significant network power with a minimal performance loss under low-to-medium network loads. However, when network load rises above a certain threshold, deflection/misrouting in the network increases, leading to higher transmission latency. To tackle this issue, injection rate throttling mechanism has been proposed. In [6] the authors used a $centrally-coordinated$ application-layer information in order to determine whom to throttle. An intensity ranking of application's IPF (instruction per fetch) is used to control congestion periodically (every 100, 000 cycles). Based on starvation rate in the network and IPF metric the appropriate applications are throttled. The injection throttling mechanism used in [7] also is based on application network intensity. By grouping applications into clusters based on their network intensity a throttling range (varying from 0 to 100%) is applied to each cluster. In opposition to a global injection rate throttling used in these mechanisms, we proposed a local throttling algorithm. We use a local injection rate control based on individual node's (router's) load to determine node to throttle. A local injection rate throttling rather than a global one has two advantages:

- No need for global control logic which requires communication among nodes to determine when to throttle new packet injection.
- By locally detects the network load, and locally controls the injection rate, congestion are detected earlier and solved before further degradation.

III. Injection Rate Throttling: IRT

Bufferless network performs worse than traditional buffered network under intensive network workloads. Because in bufferless network, a new packet can be injected into the network whenever at least one input port is free, congestion easily occurs resulting to an increase of packets deflection/misrouting. In this section, we describe our proposed injection rate throttling (IRT), a local throttling mechanism that delay new packet injection at high network load allowing better performance compare to conventional bufferless networks. With the throttling mechanism, we can control local packet

injection rate at individual router injection port according to the router's load[2]. IRT is self-throttling, that means injection controller existing in each node locally throttles injection operations.

A. Throttling Mechanism

At high network load, injection rate throttling (IRT) works by delaying new packet injection for nodes which routers are highly loaded with packets on the input ports. Before a router injects a new packet, it observes first the load of the local input ports. If the router is highly loaded[3] the injection of new packet is delayed allowing a local control of network injection rate.

In this work we propose a local injection rate throttling based on node and set our router load threshold to two, three input ports occupied by incoming flits, respectively then, new packet injection is delayed when the local load is higher than the threshold.

The throttling is used to control new packet injection in order to reduce network load when the network is congested. This reduces interference leading to less deflection of flits in the network. Throttling a particular node's injection rate at high network workload can lead to an overall gain in network performance. In a bufferless network, injection rate directly contributes to network utilization. High utilization in a bufferless network causes a high deflection rate, which distributes flits throughout the network and degrades its overall performance. By delaying new packet injection of a node in a congested area, the throttling mechanism reduces the deflection probability of other input packets of the node. As delaying injection of a node in a high loaded area will reduce deflection more than that of a node in low loaded area, our IRT mechanism uses the following principles:

- **Never delay Low-Loaded Nodes new packet injection:** Low-loaded nodes, or nodes that are within a low loaded area of the network, contribute little to the neighboring network load. Thus, delaying new packet injection for such nodes will not usually benefit system performance. Hence, IRT never delay injection of new packets for low-loaded nodes.
- **Always delay High-Loaded Nodes new packet injection:** High-loaded nodes in the other hand contributes a lot to the congestion of surrounded nodes. In other words, many flits are incoming to this node occupying the local node input ports with incoming flits. In such case, local injection certainly increase the network congestion. Hence, IRT always delay the injection of new flits for those nodes.

In summary, IRT delay injection of new flits for high-loaded nodes reducing further congestion of the network. This reduces the probability of flits deflection hence may improve the overall network performance.

[2]number of packet traversing the router at a given time.

[3]more than 2, 3 input ports (threshold), are occupied with incoming flits at a given time.

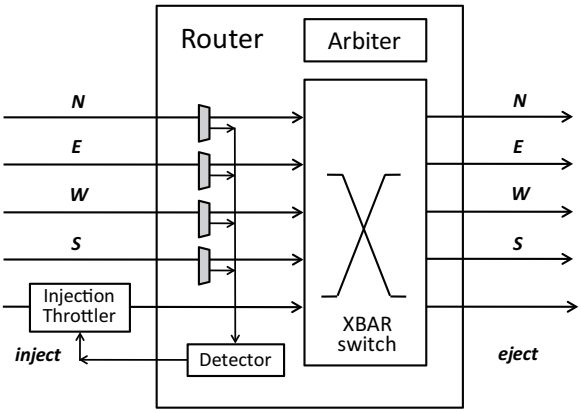


Fig. 1. IRT router architecture

B. Throttling Architecture

Our IRT router architecture is shown in Figure 1. Two components are added to a conventional wormhole router [13] : an input flit detector and an injection throttler at the inter-node communication controller (INCC) between the node's memory and the router.

- **Input flit detectors:** Detects incoming flits from the four input ports, in a case of 2D mesh, north (N), east (E), west (W), and south (S) at each given time, and propagates the result (number of incoming flits) to the injection throttlers.
- **Injection throttlers:** Estimates the local router load (number of incoming flits) and decides whenever new packet injection is to be delayed or not.

C. Throttling Algorithm

1) Measuring Local Network Intensity: To measure local network load, we use input port utilization, which is the number of input ports occupied by incoming flits in the router at a given time. Injection rate of each is locally managed. This rate is dynamically controlled by the local router's load. When the node is lightly loaded, no throttling is necessary. As load increases, the throttling mechanism becomes as aggressive as necessary to reduce congestion.

2) Throttling Procedure: The injection rate throttling is performed as follows: Before the local injection port injects a new packet:

- First step: The input flit detector checks the four input ports and informs the injection throttle of the number of incoming flits in the same cycle.
- Second step: Injection throttler defines the router's load intensity (high or low) according to a threshold, and delay further new packet injection for highly-loaded router.
- When the local router load decreases below the threshold new packet injection is allowed.

Algorithm 1 shows the throttling algorithm. At each router cycle, the input ports detector check the input ports occupancy. If the number of incoming flits is equal or larger than a

Algorithm 1 IRT Algorithm

At each router cycle :
Input Detector: Counts *incoming flits* from input ports (NEWS) and informs *Injection Throttler*.
if injection of new packet **then :**
  if *incoming flits* $\geq X$ **then** $\{X$ = IRT threshold$\}$
    delay sending flit
  else { *incoming flits* count $\prec$ Threshold }
    inject flit
  end if
end if

threshold (X in the algorithm) new packet injection will be delayed. Otherwise packet is injected without any delay.
In conventional bufferless networks, a node can inject new packet into the network with only one free input-port of its local router, hence deflection of packets easily occurs at high network load. Using local injection rate throttling, we can improve the performance of conventional bufferless network by limiting injection of packet using a certain threshold (2, 3 incoming flits to the router at the same time). At high load,IRT reduces further degradation of congested area by delaying new packet injection, hence less deflection/misrouting occurs.

IV. Reassembly Buffers at Receivers

Bufferless networks must provide some buffer space at the receivers side for packet reassembly (sorting). Because of packet truncation in worms (groups of flits) and their deflection during transmission, worms may arrive in out-of-order to their destination. Thus a buffer space is required for packet reassembly. Management of reassembly buffer is still yet an issue in existing bufferless On-chip networks. Let's assume a simple reassembly buffer allocation algorithm that allocate buffer to flits when they arrive at the destination node, and release buffers when an entire packet is reassembled. Bufferless routing has no flow control, there is no credits flow feedback to source nodes for injection control as in traditional buffered network. A full reassembly buffer condition is not transmitted to source nodes. When several packets from different source nodes are sent to the same destination node, the reassembly buffer is allocated to the first arrived flit. Once all buffers are occupied, flits from other source nodes must wait in the network until the partially reassembled packets are released. However the remaining flits of the partially received packet are deflected in the network giving priority for older flits to be buffered, hence deadlock occurs. To avoid deadlock, a large space of buffers is required. As in BLESS [12] we assume an infinite size of buffers. Although in a real systems buffers are finite, we considered a worst-case buffer size to allow each destination node to reassemble packet from all sending nodes at the same time.

A. Reassembly Buffer Architecture

Figure 2 shows the reassembly buffers architecture. At the receiver-side we propose a worm-buffer size as unit of packet recovery. The reassembly buffers are hence constituted of multiple worm-buffers. One worm-buffer contains 10 flits space[4]. There are two detectors, one for the buffers inputs, and one for their outputs.

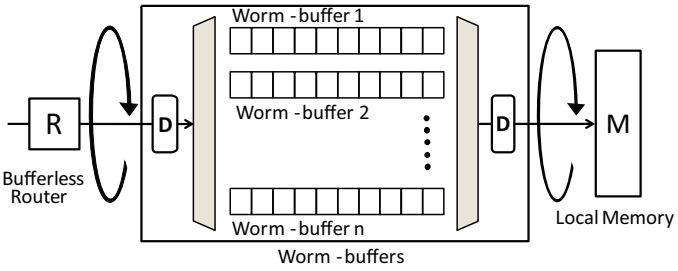


Fig. 2. Reassembly Buffer Architecture

- The input detector is used to distinguish packets coming from different source nodes. It also allow reordering of worms from the same source node. The input detector assigns a buffer space for the incoming flits. An empty worm-buffer is allocated to newly incoming flits of a packet.
- The output port detector checks if a packet is fully recovered (full worm-buffer) and transfers the packet to the local memory.

B. Reassembly Buffer Algorithm

To be able to reorder flits from out-of-order delivery, in our algorithm we add two additional tags to each worm of flits. One for distinguishing worms from different source nodes (source node tag) and one for identifying worms order (worm order tag) from the same packet. Our reassembly algorithm has three steps:

1) Input Buffer: When a worm arrives at the destination node, the input detector checks it source node tag. If it is the first worm of a packet, a free worm-buffer is allocated for buffering the entire packet. This worm-buffer is associated with the corresponding source node tag. The following worms of the packet are ordered in the same worm-buffer according to their worm order tag. The Algorithm 2 summarizes the input-side of the reassembly algorithm.

2) Worm flits reordering: We use a simple sort algorithm that order worm of flits simply according to their tags. Worm of flits from the same packet ID are buffered in the same worm-buffer, and ordered according to the flit-ID.

3) Output Buffer: Output port detector checks if any worm-buffer is full, and assigns an output port to the full worm-buffer. The algorithm is detailed in Algorithm 3.

C. Advantages

Reassembly buffers are essential for bufferless routing networks. As packet flits may reach the destination node in out-of-order due to deflection, a buffer space is required to be able to reassemble the original packet. Although our proposal does

[4]Original simulator's packet size.

Algorithm 2 Input-side Reassembly Buffer Algorithm

At destination:
if first $worm$ arrival of a packet **then**
Find $empty$ worm-buffer
Input $worm$ in worm-buffer-i, at position j { Where i is associated to the source node tag, and j the worm order tag}
$number$ ++ { For computing occupied buffer space}
else{not packet's first $worm$ arrival}
Find worm-buffer-id corresponding to *source tag*
Input $worm$ in worm-buffer-id at position *worm order tag*
$number$ ++
end if

Algorithm 3 Output-side Reassembly Buffer Algorithm

Output port detector checks for full worm-buffer:
if any $full$ worm-buffer **then**
for($number$!=0) **do :**
Output buffer
$number$ - -
if $number$ = = 0 **then**
Change worm-buffer state to $empty$
end if
end if
else{ no $full$ worm-buffer }
break

not eliminate the usage of worst-case reassembly buffer size, we can significantly reduce buffers utilization.

- We reduce the reassembly buffers size to a minimal required to avoid deadlock. Using the injection rate throttling we can reduce the deflection rate in the network hence the number of out-of-order packet delivery.
- Allocation of worm-buffers according to source tag improves the utilization of reassembly buffers, and accelerates flits reassembly.

V. Experimental Methodology

A. Simulation environment

In our simulations, we use a cycle accurate many-core processor architecture simulator SimMc [8]. We model 32 nodes(4x6), 64 nodes(8x8) and 128 nodes(8x16) 2D-mesh networks. Table I summarizes the detailed simulation configurations. Using bufferless, each network node contains a router, an inter-node communication controller (ICNC), a core, and a node's memory. The Figure 3 shows a block diagram of a single computational node.

B. NAS Parallel Benchmark

The NAS Parallel Benchmarks (NPB)[14] are a small set of programs which are designed to help evaluate the performance of parallel supercomputers. In this paper, simulation results for four kernels (MG, CG, FT, and IS)[14] are shown in the following section. We propose throttling policy of injection rate to improve network performance under heavy traffic load. Hence, we choose the kernels which are good test for communication performance.

1) Multigrid (MG) Benchmark: MG is a simplified multigrid kernel, which solves a 3-D Poisson PDE. This code is a good test of both short and long distance for highly structured communication.

2) Conjugate Gradient (CG) Benchmark: In this benchmark, a conjugate gradient method is used for computing an approximation of the smallest eigenvalue of a large, sparse, symmetric positive definite matrix. CG benchmark tests irregular long-distance communication and employs sparse matrix-vector multiplication.

3) 3-D FFT PDE (FT) Benchmark: In this benchmark, a 3-D partial differential equation is solved by using FFTs. This kernel performs the essence of many spectral methods. It is a good test of long-distance communication performance.

4) Integer Sort (IS) Benchmark: A sorting operation that is important in particle method codes. This application is similar to particle-in-cell applications of physics, wherein particles are assigned to cells and may drift out. The sorting operation is used to reassign particles to the appropriate cells. This benchmark tests both integer computation speed and communication performance.

TABLE I
Simulation Parameters

Parameter	Setting
Network Topology	2D - Mesh
Routing Technique	Wormhole Bufferless Routing
Virtual Channel	None
Routing method	X-Y dimension ordered routing.
Router	Single cycle router
Benchmark	NAS parallel benchmarks
Injection Policy	Throttling
Input flits ≤ 2	Throttle when above 2 flits
Input flits ≤ 3	Throttle when above 3 flits

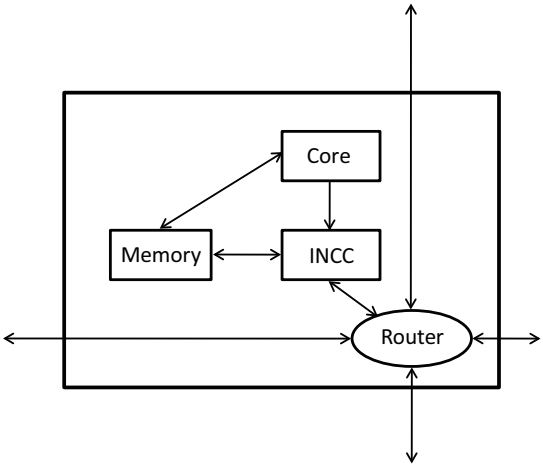


Fig. 3. A Block Diagram of Computational Node [8]

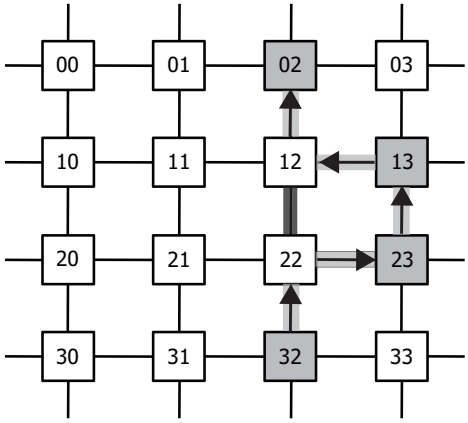


Fig. 4. An Example of Misrouting

C. Misrouting

As we introduced in section II, input port buffers are eliminated in bufferless routing. BLESS always assigns a flit to an output port even though this port does not provide minimal path in a case of deflection. The flit does not always take the shortest path to the destination(using X-Y dimension ordering routing). In other words, flits are deflected or "misrouted" [13] by the router to a non-productive output port when the best request output port is not available.

Figure 4 illustrates an example of misrouting. In this case, flit destined for 02 is misrouted to 23 at node 22. Hence, the misrouting spends 2 extra hops(calculation method as can be seen in equation 1). In our experiments, we compute total number of misrouting hops in the entire network for all source-destination communication pairs. The results of misrouting are shown in the following section.

$$Total_{misrouting} = \sum_{i=0}^{N} (HOPS_{Actual\_i} - HOPS_{Regular\_i}). \quad (1)$$

Where N is the total number of worms, $HOPS_{Actual}$ is the total number of hops taken in our simulation for a worm, and $HOPS_{Regular}$ is the hop counts if no deflection occurs in the worm's communication (shortest path).

VI. Experimental Evaluation

A. Performance

We evaluate the simulation latency of BLESS (baseline bufferless routing) and IRT(injection rate throttling) with different injection threshold: " input flits ≥ 2" means when incoming flits are more than 2 flits (count by input port detector), the injection will be delayed. " input flits ≥ 3", delays injection when the number of incoming flits is more than 3 flits at the same time. The injection rate of " input flits ≥ 3" becomes higher than the case of " input flits ≥ 2".

We evaluate the simulation cycles of NAS parallel benchmarks with 32, 64, 128 cores and compare with the baseline BLESS and the baseline BLESS using our injection throttling mechanism.

1) Multigrid (MG) Benchmark: The benchmark is a simplified multigrid kernel, which solves a 3-D Poisson PDE. MG is a good test of both short and long distance highly structured communication. At 32, 64, and 128 cores, throttling algorithm reduces simulation cycles by an average of 3.6%as shown in Figure 5(a).

2) Conjugate Gradient (CG) Benchmark: The benchmark is a simplified multigrid kernel, which solves a 3-D Poisson PDE. MG is a good test of both short and long distance highly structured communication. At 32, 64, and 128 cores, throttling algorithm reduces simulation cycles by an average of 14.3% as shown in Figure 5(b).

3) 3-D FFT PDE (FT) Benchmark: Figure 6(a) shows our evaluation results for FT benchmark. In this benchmark a 3-D partial differential equation is solved using FFTs. It is a good test of long-distance communication performance for 32, 64, and 128 cores, throttling algorithm reduces simulation cycles by an average of 8.6%.

4) Integer Sort (IS) Benchmark: The Figure 6(b) shows simulation cycles of IS (Integer Sort) benchmark with 32, 64, and 128 cores, it tests a sorting operation which is important in particle method codes, both in integer computation speed and in communication performance. Throttling algorithm reduces simulation cycles by an average of 8.1%.

B. Misrouting Reduction

Figure 7(a) shows total number of misrouting hops for NAS parallel benchmarks with 32 nodes. These results show the total number of misrouting hops. Both " input flits ≥ 2" and " input flits ≥ 3" of IRT demonstrate less misrouting hops count than the baseline bufferless routing, due to the injection rate throttling. Thus, in all cases, IRT designs take less misrouting hops than traditional BLESS. In MG benchmark, total number of misrouted hops is reduced by 10.2%, IRT has 22.2% less misrouting hops for CG benchmark compare to BLESS. In FT and IS benchmarks, the misrouted hops are reduced by 17.3% and 28.7%, respectively.

C. Energy Consumption

In an on-chip network, the average energy consumption to translate one single flit from the source to the destination can be expressed with the following formula [15] :

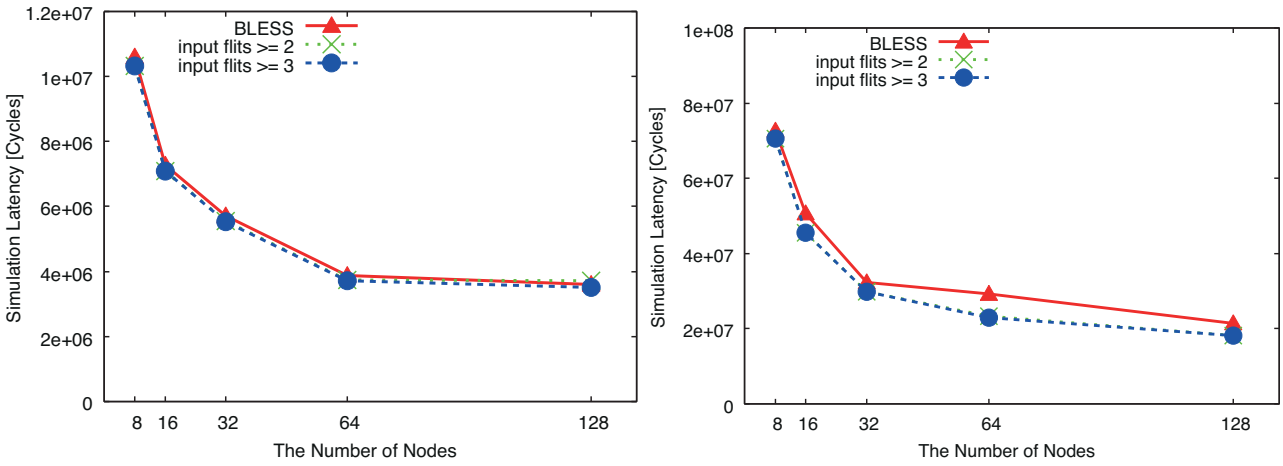
$$E_{flit} = wH_{ave}(E_{sw} + E_{link}) \quad (2)$$

Where w is the flit width, $Have$ is the average hop count, E_{sw} is the average energy consumed in switching per bit data through the router, E_{link} is the average energy per bit for link traversal.

We compare the energy consumed by each flit transmitted from source to destination of our throttling control mechanism and the baseline bufferless using the equations (3), and (4).

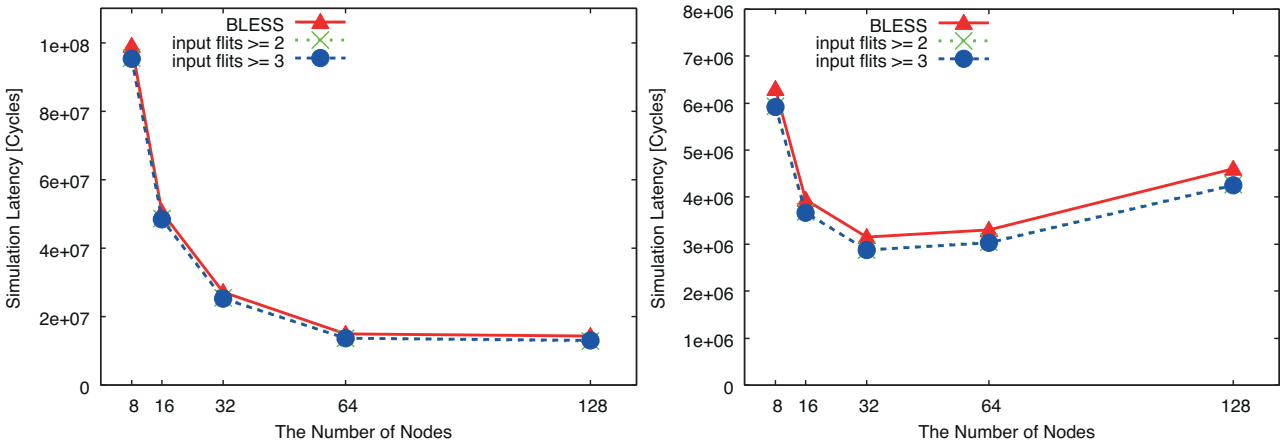
$$E_{flit(IRT)} = w_{IRT} H_{ave(IRT)} \quad (3)$$

$$E_{flit(BLESS)} = w_{BLESS} H_{ave(BLESS)} \quad (4)$$



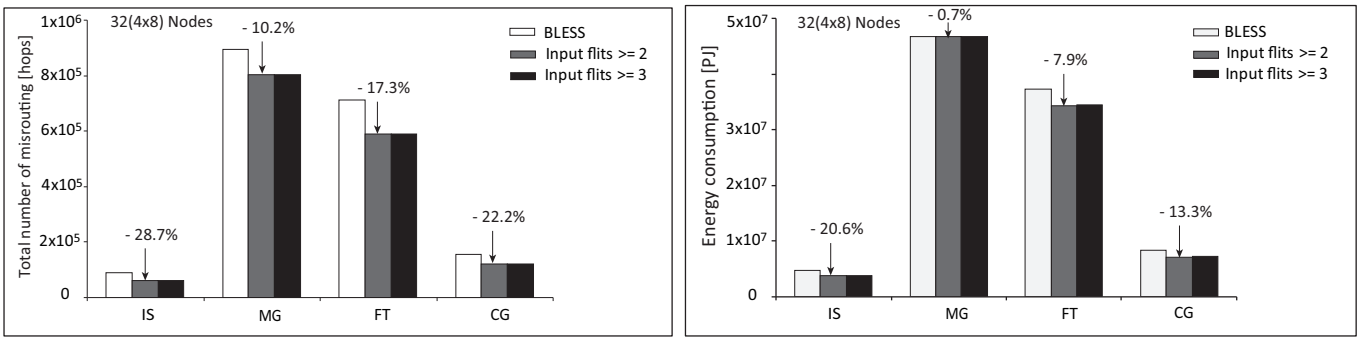
(a) MG (b) CG

Fig. 5. Simulation cycles evaluation for MG and CG benchmarks.



(a) FT (b) IS

Fig. 6. Simulation cycles evaluation for FT and IS benchmarks.



(a) Total Number of Misrouting. Hops (b) Energy consumption

Fig. 7. Number of Mirouting and Energy consumption evaluations.

We assume the same energy consumption for switching E_{sw} and link traversal E_{link} for both baseline and IRT bufferless networks. E_{link} is 0.46, estimated link traversal energy in [16]. We set 1-hop distance to $2.33\ mm$ assuming a typical semiglobal interconnect in the 45nm CMOS technology [16]. For W, IRT adds the additional overhead bits in each flit. Because we must reassemble flits at the destination nodes. Figure 7(b) shows the transfer energy as a portion of system energy for NPB in a 32 nodes(4x8) 2D-mesh network. Results show a significant reduction in energy consumption using IRT as it considerably reduces the number of deflected flits(misrouting count) as shown in Figure 7(a).

VII. Conclusion

To the best of our knowledge, IRT is the first work that applies the local injection rate throttling to bufferless network in order to improve its performance under heavy network load. Other work studies tackle the bufferless-buffered performance gap at high-loads using various alternatives. IRT is a local throttling mechanism that improves performance of bufferless routing under high network workloads. IRT delays injection when nodes are in local highly congested areas. Using a cycle accurate simulator, results show that IRT improves the average performance by 8.65% compared to a baseline bufferless for (4x8), (8x8), (8x16) NoCs under four kernels of NPB. IRT retains the benefits of simplicity in router design for reducing power and chip area. Therefore, a bufferless NoC with IRT is a compelling option to improves performance of bufferless routing under high-intensity workloads.

Acknowledgment

This research is supported in part by NII Joint research and Grants-in-Aid for Scientific Research of Japan Society for Promotion of Science (JSPS), No.22500042.

References

[1] M. Hayenga, N. Jerger, and M. Lipasti, "Scarab: A single cycle adaptive routing and bufferless network," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, dec. 2009, pp. 244 –254.

[2] C. G. Requena, M. E. Go'mez, P. Lo'pez, and J. Duato, "Reducing packet dropping in a bufferless noc," in *Euro-Par 2008 - Parallel Processing, 14th International Euro-Par Conference, Las Palmas de Gran Canaria, Spain, August 26-29, 2008, Proceedings*, ser. Lecture Notes in Computer Science, E. Luque, T. Margalef, and D. Benitez, Eds., vol. 5168. Springer, 2008, pp. 899–909.

[3] C. Fallin, C. Craik, and O. Mutlu, "Chipper: A low-complexity bufferless deflection router," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, feb. 2011, pp. 144 –155.

[4] S. Jafri, Y.-J. Hong, M. Thottethodi, and T. Vijaykumar, "Adaptive flow control for robust performance and energy," in *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, dec. 2010, pp. 433 –444.

[5] M. Thottethodi, A. Lebeck, and S. Mukherjee, "Self-tuned congestion control for multiprocessor networks," in *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, 2001, pp. 107 –118.

[6] G. Nychis, C. Fallin, T. Moscibroda, S. Seshan, and O. Mutlu, "Congestion control for scalability in bufferless on-chip networks," Microsoft Research , Carnegie Mellon University, SAFARI Technical Report 2011-003, July 2011.

[7] R. Ausavarungnirun, K. K.-W. Chang, C. Fallin, and O. Mutlu, "Adaptive cluster throttling: Improving high-load performance in bufferless on-chip networks," Computer Architecture Lab (CALCM) Carnegie Mellon University, SAFARI Technical Report 2011-006, September 2011.

[8] K. Uehara, S. Sato, T. Miyoshi, and K. Kise, "A study of an infrastructure for research and development of many-core processors," in *Parallel and Distributed Computing, Applications and Technologies, 2009 International Conference on*, dec. 2009, pp. 414 –419.

[9] P. Baran, "On distributed communications networks," *Communications Systems, IEEE Transactions on*, vol. 12, no. 1, pp. 1 –9, march 1964.

[10] H. Wang, L.-S. Peh, and S. Malik, "Power-driven design of router microarchitectures in on-chip networks," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, dec. 2003, pp. 105 – 116.

[11] T. Mattson, R. Van der Wijngaart, and M. Frumkin, "Programming the intel 80-core network-on-a-chip terascale processor," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, nov. 2008, pp. 1 –11.

[12] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in *Proceedings of the 36th annual international symposium on Computer architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 196–207. [Online]. Available: http://doi.acm.org/10.1145/1555754.1555781

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

[14] S. Saini and D. H. Bailey, "Nas parallel benchmark (version 1.0," Report NAS 96-18, 1996.

[15] H. Wang, L.-S. Peh, and S. Malik, "A technology-aware and energy-oriented topology exploration for on-chip networks," in *Design, Automation and Test in Europe, 2005. Proceedings*, march 2005, pp. 1238 – 1243 Vol. 2.

[16] A. Shacham, K. Bergman, and L. Carloni, "The case for low-power photonic networks on chip," in *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, june 2007, pp. 132 –135.