Critical Packet Prioritisation by Slack-Aware Re-routing in On-Chip Networks

Abhijit Das*, Sarath Babu†, John Jose*, Sangeetha Jose†and Maurizio Palesi‡
* Dept. of Computer Science and Engineering, Indian Institute of Technology Guwahati, India
† Dept. of Information Technology, Government Engineering College, Idukki, India
‡ Dept. of Electrical, Electronic and Computer Engineering, University of Catania, Italy
{abhijit.das, johnjose}@iitg.ac.in, {saratbabu0410, sangeethajosem}@gmail.com, maurizio.palesi@dieei.unict.it

***Abstract*—Packet based Network-on-Chip (NoC) connect tens to hundreds of components in a multi-core system. The routing and arbitration policies employed in traditional NoCs treat all application packets equally. However, some packets are critical as they stall application execution whereas others are not. We differentiate packets based on a metric called slack that captures a packet's criticality. We observe that majority of NoC packets generated by standard application based benchmarks do not have slack and hence are critical. Prioritising these critical packets during routing and arbitration will reduce application stall and improve performance. We study the diversity and interference of packets to propose a policy that prioritises critical packets in NoC. This paper presents a slack-aware re-routing (SAR) technique that prioritises lower slack packets over higher slack packets and explores alternate minimal path when two no-slack packets compete for same output port. Experimental evaluation on a 64-core Tiled Chip Multi-Processor (TCMP) with 8×8 2D mesh NoC using both multiprogrammed and multithreaded workloads show that our proposed policy reduces application stall time by upto 22% over traditional round-robin policy and 18% over state-of-the-art slack-aware policy.**

***Index Terms*—Quality-of-Service (QoS), slack estimation, adaptive routing, input selection, stall time reduction**

I. Introduction

After the paradigm shift towards multi-core systems, limitation in global wires, shared buses and monolithic crossbars are exposed. Packet based NoCs now connect tens to hundreds of components in TCMP based multi-core systems. NoCs are scalable and reliable with predictable and well controlled communication properties [1]. The most fundamental challenges in the design of general purpose TCMPs include devising efficient resource sharing and scheduling policies. Behaviour and interference of applications for fundamental shared resources like NoC [2][3][4], last level cache (LLC) [5][6][7] and memory bandwidth [8][9][10] are explored in different capacities. NoC trivially becomes the most critical shared resource as it is the communication backbone for the entire system. Even other shared resources including LLC and memory bandwidth are dependent on NoC directly or indirectly. We explore the impact of NoC because it has various hidden and indirect but significant performance defining factors. Important factors include queueing delay, memory level parallelism (MLP), irregular traffic patterns and unpredictable application interferences. These network-level factors can have a significant impact on the application-level performance.

A TCMP generally consists of processing elements organised as tiles. Each processing element houses a simple processor, a private L1 cache and a slice of shared distributed L2 cache. Typically, each L1 cache miss triggers an NoC request packet and corresponding reply packet. In an NoC, packets of different applications mainly interact with one another in the routers. The arbitration policy in these routers decide which application's packet is to be prioritised over others when they request the same output port. Traditional arbitration includes round-robin and age-based policies which are application oblivious, i.e. they treat all application packets equally. However, applications can be heterogeneous in nature with different QoS requirements and hence each of these packets will have different impact on the application-level performance. One of the main reasons for this differential impact is the presence of MLP. Servicing multiple memory requests in parallel reduces the application stall time and criticality of each of these requests to the application depends on MLP to a large extend.

Consider the following example: Assume that an application issues two network requests (cache misses), one after another, first to a distant tile in the network, and second to a closer tile. The application can continue execution only after the reply of these requests are received. The first request packet travels far and hence take more time to return, whereas the second request packet travels less and come back before the first packet. Even after the second reply packet arrives, the application continues to stall because the first reply packet is expected. Clearly, the second packet is less critical and can be delayed for multiple cycles without adding any stall to the application's execution. This is because the latency of second packet is hidden under the first packet, which takes more time. Thus, the delay tolerance of each packet can be different with respect to its impact on the application's performance.

We study the diversity and interference of packets to design packet-aware NoCs for general purpose TCMPs. We differ-

This research is supported in part by Department of Science and Technology (DST), Government of India vide project grant ECR/2016/000212.

978-

978-1-5386-4893-3/18/$31.00 ©2018 IEEE

entiate packets based on a metric called slack which is a measure of packet's criticality. Slack of a packet is defined as the number of cycles the packet can be delayed in the network without affecting application execution [3]. Therefore, packets with available slack (slack-1) are non-critical compared to the packets with no available slack (slack-0). Increasing the latency of slack-0 packets stalls application execution.

We propose an NoC architecture that prioritises critical packets in the network by a slack-aware re-routing (SAR) technique. Our SAR routers prioritise lower slack packets over higher slack packets like in Aergia [3]. But when two slack-0 packets have a port conflict, we re-route one of them through an alternate minimal path towards destination. Experimental analysis show that our policy effectively improves application-level performance compared to the existing policies. Our main contributions of this paper can be summarised as follows:

- We estimate slack of cache miss requests at runtime based on MLP of predecessor misses and incorporate this slack value on NoC packets as a priority.
- We adopt a look-ahead routing to facilitate re-routing of slack-0 packets through alternate minimal paths.
- We modify baseline routers to prioritise lower slack packets during routing and arbitration and re-route slack-0 packets when the desired output port is unavailable.
- We qualitatively and quantitatively compare our proposal to traditional round robin and state-of-the-art Aergia [3] policies to assess the performance.

II. Motivation

Modern TCMPs employ different MLP based methods like out-of-order execution, runahead execution etc. to reduce the penalty of load misses. These methods basically issue parallel memory requests with an intention to overlap future load misses with current load misses. If the application's behaviour shows MLP in NoC, the latencies of outstanding packets overlap and introduce slack cycles.

A. *Exploiting Slack and its Diversity*

If the NoC routers are aware of the available slack, they can take routing and arbitration decisions by prioritising lower slack packets. We identify few cases where exploiting slack information of packets can reduce application stalls.

Case 1: Interference between Different Slack Packets: Consider a 64-core TCMP as given in Figure 1. Two applications, one in Core-A (tile 57) and other in Core-B (tile 46) run simultaneously. Core-A encounters two load misses and generates two packets (A0 and A1). The first packet A0 is sent to tile 7 and is not preceded by any outstanding packet, hence it has a latency of 13 hops and a slack of 0 hops. In the next cycle, the second packet A1 is sent to tile 40 with a latency of 3 hops. Since packet A1 is preceded (and thus overlapped) by the 13-hops packet A0, it has a slack of minimum 10 hops (13 - 3 hops). Similarly, for Core-B the first packet B0 has a latency of 7 hops and a slack of 0 hops while the second packet B1 has a latency of 3 hops and a slack of 4 hops.

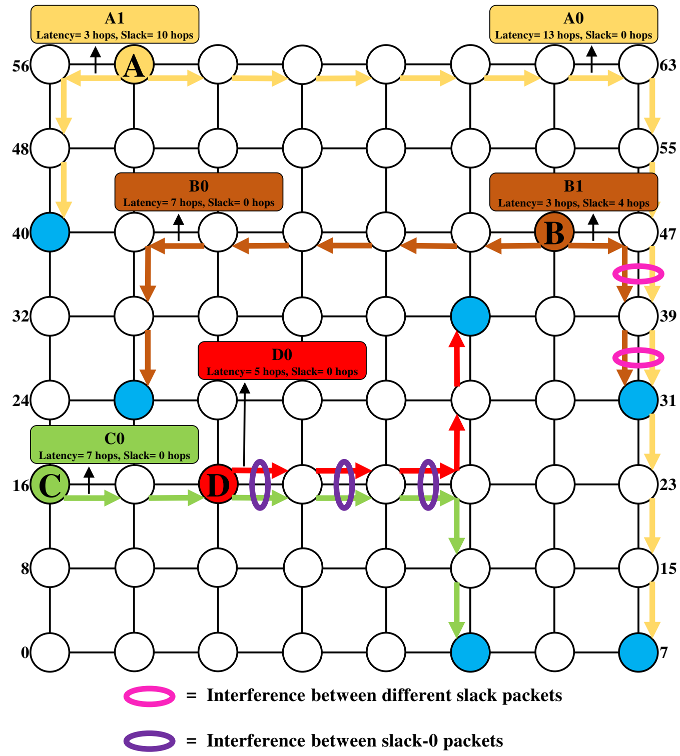

Figure 1: Illustrative example of slack

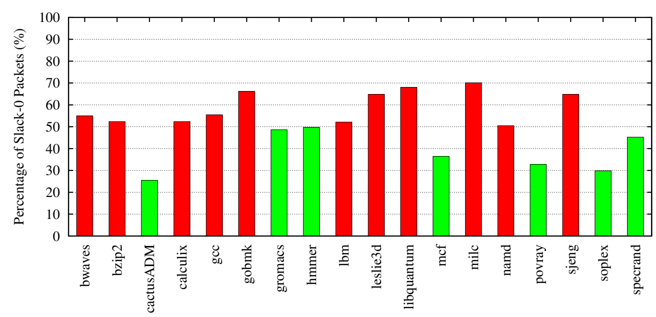

Figure 2: Slack-0 packets in SPEC CPU2006 benchmarks

Packets A0 and B1 interfere at 2 points (tiles 47 and 39) as shown in Figure 1. A traditional application oblivious slack-unaware routing and arbitration policy that prioritises B1 over A0 degrades the application-level performance as A0 is more critical than B1. In contrast if the NoC is slack-aware, it will prioritise packet A0 over B1 and reduce the stall time of Core-A without actually increasing the stall time of Core-B. Workload characteristics in Aergia shows that there exists sufficient diversity in slack of packets across various benchmarks [3]. This observation led to the slack-aware routing techniques in NoC [11][12].

Case 2: Interference between Slack-0 Packets: Consider another two applications in Core-C (tile 16) and Core-D (tile 18) which also run simultaneously with Core-A and Core-B on the same 64-core TCMP as shown in Figure 1. Core-C generates a packet C0 with a latency of 7 hops and a slack

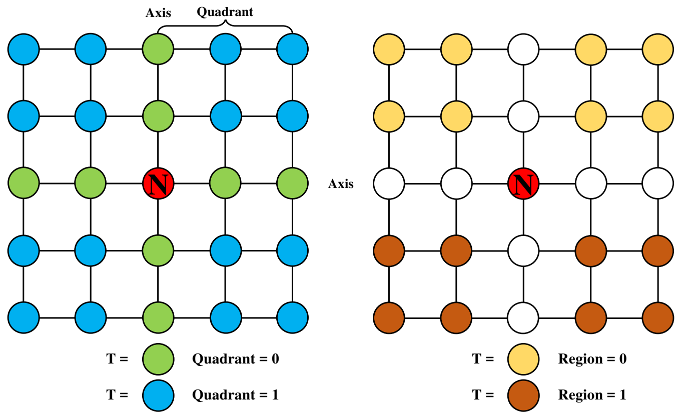

Figure 3: *Quadrant* and *Region*

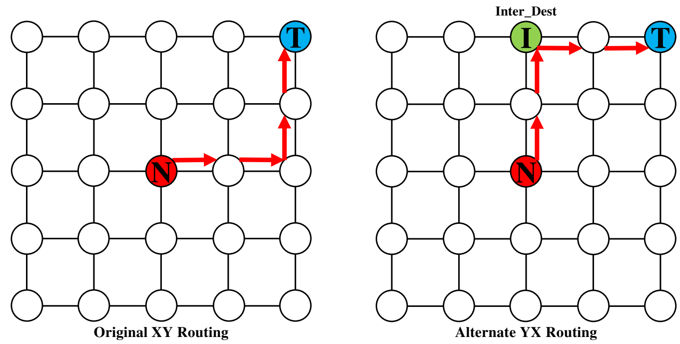

Figure 4: Alternate minimal path re-routing

of 0 hops. While Core-D generates a packet D0 that has a latency of 5 hops and a slack of 0 hops.

Packets C0 and D0 also interfere at 3 points (tiles 18, 19 and 20) as shown in Figure 1. Since both C0 and D0 are equally most critical, delaying either of them degrades the application-level performance. In this case, as per Aergia [3] only one of the packets will get productive port and other will be delayed at least for 1 cycle. There are cases where slack-0 packets are delayed upto 8 cycles due to this port conflict. In contrast if the NoC is slack-aware and can forward one of the slack-0 packets through an alternate minimal path, both C0 and D0 can progress in parallel. This reduces the stall time of both Core-C and Core-D.

Figure 2 presents the percentage of slack-0 packets in a representative set of 18 SPEC CPU2006 benchmarks. This set is a mix of heterogeneous applications with different network related characteristics. X-axis lists all the benchmarks and Y-axis shows the percentage of slack-0 packets in them. A trend is clearly visible. Most of them have more than 50% slack-0 packets. Therefore, an NoC architecture with simple slack-aware routing policy [3] will not guarantee performance. We observe from a 64-core workload mix running on an 8x8 2D NoC that there are upto 34% cases where two slack-0 packets have port conflicts in the NoC routers. We address this issue with a novel re-routing technique.

III. SAR Architecture

Our proposed SAR routers perform online estimation of slack and alternate minimal path for routing and arbitration.

A. Slack Estimation

We estimate slack with respect to outstanding network transactions (L1 miss requests). We define slack of a packet as the difference between the maximum expected latency of its predecessor (i.e. any outstanding packet that was injected before this packet) and its own expected latency with proper adjustments on injection time. This latency is based on the minimum distance to be traversed in the network by a packet.

Literature has other indirect metrics like L2 cache access status (hit or miss), number of miss predecessors (predecessors of a packet that are L2 cache miss) etc. which also correlates with slack and criticality of packets. However, such estimations become computation (hit/miss predictor) and storage expensive (miss predecessor's list). We intuitively assume all L2 cache access are hits and avoid the off-chip slack computation as it is irregular and cannot be quantised accurately.

We modify the structure of L1 miss status handling registers (MSHRs) to include predecessor related information. Before a cache miss request packet is injected into the network, slack is computed using this information from MSHRs. The slack is then quantised as a 1-bit value (*Slack*) and stored in the packet header. All the slack-0 packets are quantised as 0 and all higher slack packets are quantised as 1. This *Slack* bit is used to enable priority based routing and arbitration.

B. Minimal Path Estimation

Slack based priority policy is used to make sure that a slack-0 packet is not delayed in any router. But when two slack-0 packets compete in a router for a single output port one has to be delayed. Rather than delaying a slack-0 packet, we explore the possibility of assigning another productive port to one of the slack-0 packets by re-routing. Re-routing is a technique where a packet is forwarded to an intermediate router within the minimal quadrant of current router and destination router. This makes sure both the conflicting slack-0 packets get a productive port. SAR routers use some additional metrics to estimate alternate minimal path for packet forwarding.

When a packet with destination router T leaves a router S to N (N is neighbour of S), two 1-bit metrics; *Quadrant* and *Region* is computed and quantised in its header. *Quadrant* and *Region* bits indicate the relative position of T with respect to N. If N and T are on an axis of the N, i.e. on the same row or same column then the *Quadrant* bit is set to 0 else 1 as shown in Figure 3. If *Quadrant=1* then *Region=0* indicates T is in upper region of N and *Region=1* indicates T is in lower region of N. If *Quadrant=0* then *Region* bit is irrelevant. This *Quadrant* and *Region* bit update happens on each router before the packet moves to its crossbar stage. *Region* helps to identify an alternate minimal path towards destination if the desired output port is not available at N. A packet with *Quadrant* bit set to 1 can be re-routed at N.

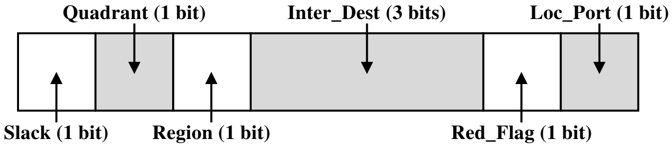

Figure 5: SAR priority vector

Slack	0	Slack-0 packets
	1	All other packets
Quadrant	0	Destination is on axis (X/Y) of N
	1	Destination is on quadrant
Region	0	Destination is on upper region
	1	Destination is on lower region
Inter_Dest	000 - 111	Last router in Y direction if YX routing is used at N
Red_Flag	0	Packet is not re-routed
	1	Packet is re-routed
Loc_Port	0	Nothing
	1	Packet is sent to local port at *Inter_Dest*

Table 1: SAR priority vector description

Another metric called *Inter_Dest* stores the address to be used for re-routing using alternate minimal path. It is the last router in Y direction from N if YX routing is used to reach destination of the packet. Figure 4 shows *Inter_Dest* (router I) with an example and verifies its position on the minimal path towards destination. For our evaluation of an 8×8 2D mesh, a 3-bit *Inter_Dest* along with a 1-bit *Red_Flag* is used. Only the column number (3-bits) is stored, as *Inter_Dest* (router I) is on the same row as that of actual destination (router T). If the *Red_Flag* bit is 0, *Inter_Dest* is invalid.

Another 1-bit metric *Loc_Port* is used for deadlock prevention (will be discussed when deadlock is addressed). An 8-bit SAR priority vector (as shown in Figure 5) that incorporates all the above discussed metrics is added on the head flit of each packet. Table 1 describes the fields of SAR priority vector.

C. Router Microarchitecture Modification

Architectural block diagram of SAR router microarchitecture is presented in Figure 6. Like a generic 2D mesh baseline router, SAR router also has 5 input and 5 output ports/channels; one from each direction (east, west, north and south) and one from the local tile (through network interface). North and south input ports have additional demultiplexers (D1 and D2) to redirect packets to local input port. Multiplexers (M1/M2/.../Mn) in local input port send the re-routed packets to the appropriate VCs. SAR routers use XY routing and wormhole switching where only the head flit participates in routing and arbitration. We use round-robin policy in baseline routers and slack-aware re-routing policy in SAR routers for performance comparison.

The Routing Computer (RC), VC Arbiter (VA) and Switch Arbiter (SA) units are same as that of baseline routers. Two additional units Packet Pre-processor (PP) and Look Ahead Re-router (LR) facilitates the technique of SAR.

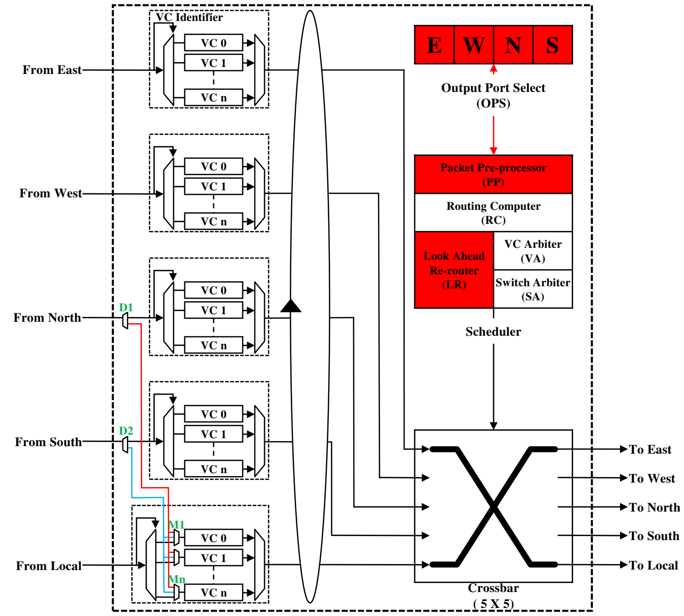

Figure 6: SAR router microarchitecture

Packet Pre-processor Unit (PP): This unit is an addition to the baseline routers and works in parallel across all input ports. The fields in SAR priority vector is used by this unit for initiating re-routing operations for every incoming head flits. This unit works in conjunction with 4-bit *Output Port Select (OPS)* structure to identify port conflicts of slack-0 packets. PP unit identifies all slack-0 packets and direct them towards productive output ports by enabling re-routing if required and possible. The working of PP unit is presented in Algorithm 1.

Algorithm 1: Working of Packet Pre-processor unit (PP)

Input : 8-bit SAR priority vector,
Output Port Select (*OPS*), destination (*T*)
Output: Identification of minimal output port

```
Loc_Port = 0
if Slack == 0 && Quadrant == 1 then
    if desired output port (E/W) not marked on OPS then
        Mark East (E) or West (W) output port on OPS
    else if Region == 0 && N not marked on OPS then
        Mark North (N) output port on OPS
        Swap column bits of T with Inter_Dest
        Red_Flag = 1
    else if Region == 1 && S not marked on OPS then
        Mark South (S) output port on OPS
        Swap column bits of T with Inter_Dest
        Red_Flag = 1
    else
        break
else
    break
```

Look Ahead Re-router Unit (LR): This is another additional unit in SAR routers. LR unit uses the next router information from RC unit and calculates alternate minimal path related metrics in advance only to be used by the PP unit of the next router. Algorithm 2 describes the working of LR unit in proposed SAR routers.

Algorithm 2: Working of Look Ahead Re-router unit (LR)

Input : 8-bit SAR priority vector,
next router (N), destination (T)
Output: *Quadrant*, *Region* and *Inter_Dest*

if *Slack* == *0* **then**
  if T == N && *Red_Flag* == *1* **then**
    Replace column bits of T with *Inter_Dest*
    Red_Flag = 0
    Loc_Port = 1
  else if T != N **then**
    row_diff = row of T - row of N
    col_diff = column of T - column of N
    if *row_diff* == *0* || *col_diff* == *0* **then**
      Quadrant = 0
    else
      Quadrant = 1
      if *row_diff* > *0* **then**
        Region = 0
      else
        Region = 1
      Temp_Dest = N + row_diff * network_radix
      Inter_Dest = column bits of Temp_Dest
  else
    break
else
  break

LR unit works in parallel with VA and SA units since the metrics it calculates are used only by the next router. Since LA unit is not in the critical path of router pipeline it incurs no additional delay.

In our SAR routers, each virtual channel has an extra priority field which stores the 1-bit *Slack* value of the head flit when it reserve the channel. This field is used by the body flits for priority based arbitration. An illustrative example of packet re-routing from router D is given in Figure 7. For packet C0, the dashed line through routers D, P, and T indicate the original path and the solid line through routers D, I and T indicate the re-routed path.

D. Comparison and Design Challenges

We compare the effectiveness of our technique with Aergia [3] that estimates slack in packets and prioritises lower slack packets over higher slack packet during VC and switch arbitration. Aergia also uses batching to prevent higher slack packets from starvation.

All the same slack packets within a batch are treated as equal in Aergia and prioritised at random. But we have seen in Figure 2 that almost across all benchmarks slack-0 packets dominate. Hence, Aergia suffers from performance degradation when one slack-0 packet is prioritised over other.

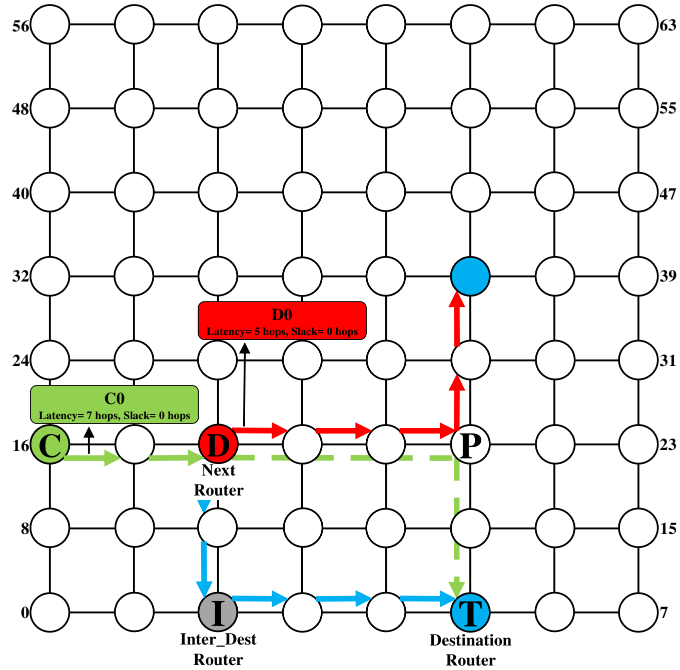

Figure 7: Illustrative example of slack-aware re-routing

In contrast, our SAR policy works similar to Aergia to prioritise lower slack packets but also re-routes slack-0 packets in alternate minimal path when required. We do not use batching to prevent starvation as our 1-bit slack based priority does not add any significant unfairness to the proposed architecture. Our proposal can be used as a complimentary policy with any other packet prioritisation technique.

Starvation: When we evaluate our proposed SAR architecture we observe that our 1-bit slack based priority does not add any significant unfairness to the system when compared to traditional round-robin policy. Thus, we do not use any additional metric for starvation prevention. Our proposed policy can always be extended with techniques like batching [2][3].

Livelock: In SAR routers, a packet always travels on a minimal path towards destination whether or not it is re-routed. Lower slack packets are prioritised over higher slack packets and slack-0 packets are re-routed through alternate minimal path; but forward progress is always ensured. Hence, the proposed SAR architecture is livelock free.

Deadlock: Our SAR routers use XY routing where packets are first routed in X direction followed by Y direction. However, when a packet is re-routed through an alternate minimal path, it takes an early Y-direction as shown in Figure 7 (rather than routers D, P, and T, packet C0 takes routers D, I, and T). After reaching *Inter_Dest* (router I), if the packet attempts to take X-direction again, then it violates XY routing which may lead to deadlock. To prevent this situation, a 1-bit *Loc_Port* is used in SAR priority vector. When *Loc_Port* is set to 1, the packet after reaching router I is sent to the local input port VC. The demultiplexers (D1 and D2) placed in north and south input ports will extract the packet and add it to local port

Processor	64 OoO x86 cores
L1 cache	32KB, 4-way, 64B lines, private
L2 cache	512KB×64 cores, 16-way, 64B lines, shared SNUCA
NoC	8×8 2D mesh, 4 VCs/port, 128-bit flit channel
Packets	1-flit request, 5-flit reply
Benchmarks	SPEC CPU2006 (multiprogrammed), PARSEC (multithreaded)

Table 2: Simulation configuration

#	Benchmark	Slack-0%	MPKI	#	Benchmark	Slack-0%	MPKI
1	cactusADM	25.47	Low	8	gobmk	66.15	High
2	soplex	29.82	Low	9	libquantum	68.03	Low
3	povray	32.73	Low	10	milc	70.12	Low
4	specrand	45.25	Low	11	blackscholes	44.59	Low
5	namd	50.49	Low	12	ferret	46.72	High
6	lbm	52.14	High	13	streamcluster	48.28	Low
7	bzip2	52.36	High	14	x264	48.65	High

Table 3: Benchmark characteristics

VC via multiplexers (M1/M2/.../Mn). From this local input port, the packet can take X-direction towards destination like a newly injected packet. Thus, even though we use both XY and YX routing by incorporating local port VC, we prevent deadlock in the proposed SAR architecture.

IV. Experimental Setup

In this section, we describe the experimental framework, the metrics and application workloads used for performance evaluation and the trade-offs in the choice of design parameters.

A. Simulation Setup

We implement the proposed SAR architecture on cycle-accurate, trace-driven BookSim [13] simulator. The memory traces are generated by event-driven gem5 simulator with Ruby [14]. We extend the Ruby memory model on gem5 and modify the structure of L1 MSHR to include latency based slack calculation. Every newly injected packet refers these modified MSHR entries to get predecessor related information. Modified router microarchitecture is modelled in BookSim to enable slack-aware re-routing policy for priority based routing and arbitration. BookSim driven by gem5 traces forms the simulation framework for our performance evaluation. Table 2 provides the configuration details of our simulation including processor, cache and NoC parameters.

B. Evaluation Metrics

We evaluate the existing and proposed policies using different performance metrics. We define *network stall time* (NST) as the number of cycles an application stalls waiting for a network packet. We assume all L2 access are hits as we want to identify the effects of NoC alone. We define *usage wait time* (UWT) as the number of cycles a reply packet waits from arrival at the source tile until being used by an application. UWT shows how early or late reply packets arrive at the source tile than necessary. We also define a metric called *reply difference time* (RDT) as the number of cycles between the arrival of first and last flits of the reply packet at the source tile. RDT shows how long an application may stall due to the delayed arrival of remaining flits of a packet after the head flit has arrived. Ideally, we need lower NST, UST and RDT for better application-level performance.

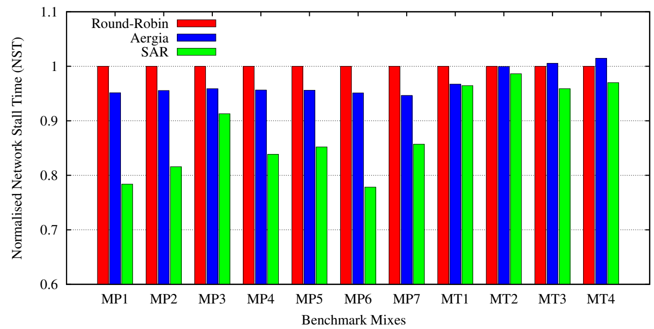

Figure 8: Effect on network stall time (NST)

C. Application Categories and Characteristics

We use both multiprogrammed (MP1–MP7) and multithreaded (MT1–MT4) application workloads for performance evaluation. For multiprogrammed workloads, we use SPEC CPU2006 benchmarks where each core runs a separate application. For multithreaded workloads, we use PARSEC benchmarks where each core runs a separate process/thread but of a single application. In total we study 14 different benchmarks, 10 multiprogrammed and 4 multithreaded.

To evaluate our proposal, we create different workloads of varying network characteristics with SPEC CPU2006 and PARSEC benchmarks. We estimate percentage of slack-0 packets to identify the criticality of benchmarks. We also calculate misses per kilo instructions (MPKIs) to estimate the network load contributed by the respective benchmarks. The characteristics of benchmarks are presented in Table 3. The workload formation is presented in Table 4 with description. For example, workload MP1 consists of 32 instances of high MPKI benchmarks (16 cores run *bzip2* and another 16 cores run *lbm*) and 32 instances of high slack-0% benchmarks (16 cores run *milc* and another 16 cores run *libquantum*).

V. Performance Evaluation

We compare SAR to baseline round robin and state-of-the-art Aergia policies based on NST, UWT and RDT for both multiprogrammed and multithreaded application workloads. The plotted result for each workload is averaged over 8 different spatially scheduled combinations. We also present router critical path, area and power overheads of SAR routers.

A. Effect on NST

Figure 8 shows the normalised NSTs of workloads with respect to the baseline round-robin policy. Round-robin delay packets during port conflicts irrespective of their load and criticality. This is because local round-robin policy is application-oblivious. SAR reduces stall time for all workloads. Significant reduction in stall time can be seen for workload mixes of

Workload	Representative Benchmark Combinations				Workload Characteristics
MP1	bzip2(16)	lbm(16)	milc(16)	libquantum(16)	32 high-MPKI with 32 high-slack-0% benchmarks
MP2	bzip2(16)	lbm(16)	cactusADM(16)	soplex(16)	32 high-MPKI with 32 low-slack-0% benchmarks
MP3	specrand(16)	namd(16)	milc(16)	libquantum(16)	32 low-MPKI with 32 high-slack-0% benchmarks
MP4	specrand(16)	namd(16)	cactusADM(16)	soplex(16)	32 low-MPKI with 32 low-slack-0% benchmarks
MP5	milc(16)	libquantum(16)	cactusADM(16)	soplex(16)	32 high-slack-0% with 32 low-slack-0% benchmarks
MP6	milc(16)	libquantum(16)	gobmk(16)	povray(16)	48 high-slack-0% with 16 low-slack-0% benchmarks
MP7	cactusADM(16)	soplex(16)	povray(16)	gobmk(16)	16 high-slack-0% with 48 low-slack-0% benchmarks
MT1	blackscholes(64)				64 threads of the benchmark; 1 per core
MT2	ferret(64)				64 threads of the benchmark; 1 per core
MT3	streamcluster(64)				64 threads of the benchmark; 1 per core
MT4	x264(64)				64 threads of the benchmark; 1 per core

Table 4: Core-wise application scheduling for various workload mixes

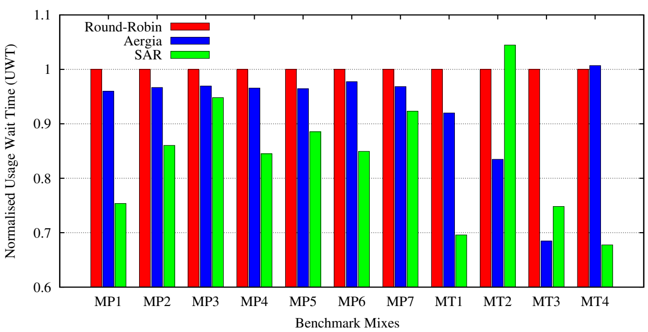

Figure 9: Effect on usage wait time (UWT)

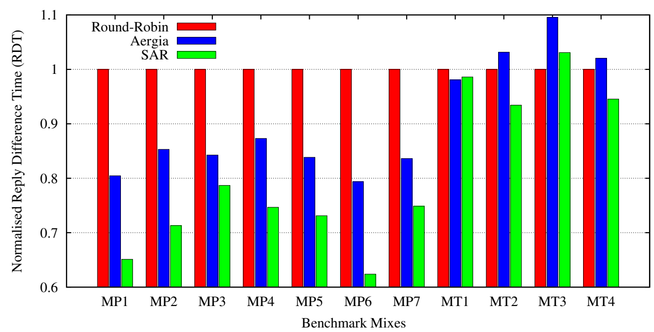

Figure 10: Effect on reply difference time (RDT)

high load and high slack critical benchmarks. We observe highest NST reduction (22% over round-robin and 18% over Aergia) for workload MP6 as it consists of 75% of slack-0 rich benchmarks (refer Table 4). Similarly, for MP1 also we see a very good NST reduction as it is a mix of high load and high slack critical benchmarks. In MP3 we achieve only 5% reduction over Aergia because it has fewer port contentions due to low rate of packet injection (low MPKI benchmarks).

For multithreaded workloads (MT1–MT4), due to the inherent DNUCA based assignment of L2 address space, majority of L1 cache misses travel less to corresponding L2 cores. This results in either no slack or very little slack for cache misses. Hence there are very less opportunities to apply slack-aware re-routing leading to marginal NST reduction with our technique. Even Aergia gets little improvement on 3 out of 4 multithreaded workloads.

B. Effect on UWT

Figure 9 shows the normalised UWTs of workloads with respect to the baseline round-robin policy. A reply packet has UWT if it reaches the source tile earlier than needed. While the packet has reached, at least one of its predecessors is still in the network. This might happen by penalising peer packets during port conflict at intermediate routers. Another possibility of having a UWT is that the packet may have very high slack which is not fully used during port conflicts. By our prioritisation technique a slack-1 packet will never delay a slack-0 packet. We observe that SAR reduces UWT significantly. It varies from 5% reduction over round-robin in MP3 to 25% reduction over round-robin in MP1.

For multithreaded workloads, SAR perform much better than Aergia which uses multi-bit slack value with batching. Results show that even with single-bit slack value without batching we can avoid starvation. In our case any slack above 0 whether it is between 1 and 5 (low slack) or above 5 (high slack), all are represented as slack-1 packets. In MT2 we find that low slack packets are over-penalised leading to higher UWT than round-robin and Aergia. In MT1, MT3 and MT4 all slack-0 and slack-1 packets are received just in time.

C. Effect on RDT

Since we consider 128-bit flit channel and 64B cache lines (blocks), every 1-flit cache miss request packet generates a 5-flit reply packet (1 head flit, 4 body flits) that bring the cache block from L2 tile to the L1 tile. Application can resume execution only if all the four body flits of reply packet reach the source tile. Our technique facilitates forwarding body flits of reply packets as soon as possible. Hence RDT is a very important metric that contribute to performance evaluation. Figure 10 shows the normalised RDTs of workloads with respect to the baseline round-robin policy. In our prioritisation policy, reply flits get forwarded without any interleaving. Due to which the body and tail flits reach the source tile without much delay. Aergia too have good RDT reduction on an average. SAR achieves an average RDT reduction of 14% over Aergia in multiprogrammed workloads.

For multithreaded workloads since there is not much slack diversity; all the packets are more or less equal and hence the reply packets are received just in time. Round-robin performs better than Aergia because there is no slack diversity. The unnecessary level of slack based priority and negative effects of batching is the reason for this behaviour.

D. Effect on Router Critical Path, Area and Power

We implement SAR router microarchitecture in Verilog and synthesize using Synopsys Design Compiler with 65nm cell library to obtain timing characteristics. We assume 65nm technology for an NoC operating at 1GHz frequency with an inter-router link delay of 1 cycle. We use the traditional 2-cycle pipelined router with first cycle for PP and RC units (refer Figure 6). Even though PP is in the critical path, the combined combinational delay of PP and RC is 7% lower than the combined combinational delay of VA and SA units that constitute the second cycle stage. Our LR unit works in parallel to VA and SA units. The experimental observation that VA and SA stage determines the pipeline latency is already established [15]. Hence, SAR routers can be operated with the same pipeline frequency. Our additional units incur a router area overhead of 1.7% and static energy consumption of 2.1%.

We compute the dynamic power dissipation estimates of SAR using Orion 2.0 [16]. Dynamic power consumption of NoC using SAR is 7.5% lower than using Aergia routers due to effective re-routing and reduction in latency of slack-0 packets. This compensates for the minor area and hardware overhead.

VI. Related Work

Criticality: Available literature has proposals that target criticality of data and instructions [17][18][19][20]. Cache miss criticality is explored with MLP based proposals [21][22]. Memory scheduling is explored with bank level parallelism [23][24]. Slack based criticality is studied for both performance and power optimisation [3][11][12]. But the impact of slack-0 packets are not observed before.

Prioritisation: Other than traditional round-robin and age-based prioritisation, literature also has QoS [25][26][27] and application-aware [2][4][28] prioritisation policies. There are prioritisation proposals based on latency-sensitivity of NoC packets [29][30]. While most of the available proposals aimed for guaranteed service or fairness, our aim is to reduce application stall time and improve system performance. Furthermore, available proposals assign static priority to improve real-time performance. In contrast, our proposal computes dynamic priority for routing and arbitration.

VII. Conclusion

By understanding the diversity and interference of packets we propose a policy that prioritises critical packets in NoC. We present SAR, a slack-aware re-routing technique that prioritises lower slack packets over higher slack packets and re-routes slack-0 packets through alternate minimal path towards destination. Experimental analysis show that our policy improves system performance over existing policies for both multiprogrammed and multithreaded workloads. The performance gain is achieved with only a negligible area and static power overhead. We believe SAR routers can be good design alternative for TCMPs that run time critical applications.

References

[1] W. J. Dally and B. P. Towels, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[2] R. Das *et al.*, "Application-Aware Prioritization Mechanisms for On-Chip Networks," in *MICRO*, 2009.

[3] R. Das *et al.*, "Aergia: Exploiting Packet Latency Slack in On-Chip Networks," in *ISCA*, 2010.

[4] N. C. Nachiappan *et al.*, "Application-aware Prefetch Prioritization in On-chip Networks," in *PACT*, 2012.

[5] L. R. Hsu *et al.*, "Communist, Utilitarian, and Capitalist Cache Policies on CMPs: Caches as a Shared Resource," in *PACT*, 2006.

[6] Y. Li *et al.*, "Heterogeneous-Aware Cache Partitioning," *Parallel Computing*, 2014.

[7] S. W. Keckler, "Rethinking Caches for Throughput Processors: Technical Perspective," *Communications of the ACM*, 2014.

[8] Y. Kim *et al.*, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," in *HPCA*, 2010.

[9] T. Pimpalkhute and S. Pasricha, "NoC Scheduling for Improved Application-Aware and Memory-Aware Transfers in Multi-Core Systems," in *VLSID*, 2014.

[10] L. Subramanian *et al.*, "BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling," *IEEE TPDS*, 2016.

[11] J. Zhan *et al.*, "Optimizing the NoC Slack Through VFS in Hard Real-Time Embedded Systems," *IEEE TCAD*, 2014.

[12] B. Sudev *et al.*, "Network-on-Chip Packet Prioritisation based on Instantaneous Slack Awareness," in *INDIN*, 2015.

[13] N. Jiang *et al.*, "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator," in *ISPASS*, 2013.

[14] N. Binkert *et al.*, "The gem5 Simulator," *SIGARCH CAN*, 2011.

[15] C. Nicopoulos *et al.*, "On the Effects of Process Variation in Network-on-Chip Architectures," *IEEE TDSC*, 2010.

[16] A. B. Kahng *et al.*, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration," in *DATE*, 2009.

[17] B. Fields *et al.*, "Slack: Maximizing Performance Under Technological Constraints," in *ISCA*, 2002.

[18] S. Subramaniam *et al.*, "Criticality-Based Optimizations for Efficient Load Processing," in *HPCA*, 2009.

[19] S. Ghose *et al.*, "Improving Memory Scheduling via Processor-Side Load Criticality Information," in *ISCA*, 2013.

[20] J. S. Miguel and N. E. Jerger, "Data Criticality in Network-On-Chip Design," in *NOCS*, 2015.

[21] M. K. Qureshi *et al.*, "A Case for MLP-Aware Cache Replacement," in *ISCA*, 2006.

[22] M. Moreto *et al.*, "Dynamic Cache Partitioning Based on the MLP of Cache Misses," *Springer THiPEAC III*, 2011.

[23] Y. Kim *et al.*, "A Case for Exploiting Subarray-level Parallelism (SALP) in DRAM," in *ISCA*, 2012.

[24] X. Tang *et al.*, "Improving Bank-Level Parallelism for Irregular Applications," in *MICRO*, 2016.

[25] J. W. Lee *et al.*, "Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks," in *ISCA*, 2008.

[26] B. Li *et al.*, "Dynamic QoS Management for Chip Multiprocessors," *ACM TACO*, 2012.

[27] B. Li *et al.*, "Dirigent: Enforcing QoS for Latency-Critical Tasks on Shared Multicore Systems," in *ASPLOS*, 2016.

[28] T. Pimpalkhute and S. Pasricha, "An Application-Aware Heterogeneous Prioritization Framework for NoC Based Chip Multiprocessors," in *ISQED*, 2014.

[29] E. Bolotin *et al.*, "The Power of Priority: NoC Based Distributed Cache Coherency," in *NOCS*, 2007.

[30] W. Dai *et al.*, "A Priority-Aware NoC to Reduce Squashes in Thread Level Speculation for Chip Multiprocessors," in *ISPA*, 2011.