Designing Data-Aware Network-on-Chip for Performance

Abhijit Das[1 2] and John Jose[1]
[1]*Indian Institute of Technology Guwahati, Assam, India*
[2]*Univ Rennes, Inria, Lannion, France*
abhijit.a.das@inria.fr

***Abstract*—Network-on-Chip (NoC) is a widely adopted communication infrastructure in Tiled Chip Multi-Processors (TCMPs) due to its high transfer bandwidth, scalability and reliability. As the number of cores continues to increase in modern TCMPs, NoC plays a pivotal role in determining the performance. In fact, it is reported that NoC is responsible for 60% to 75% of the miss latency experienced by the applications run on TCMPs. Nevertheless, most of the existing memory access techniques and optimisations are NoC oblivious, which limits their performance.**

This work advocates for considering NoC and memory hierarchy together when designing techniques and proposing optimisations for TCMPs. It shows that designing data-aware NoC with the help of memory hierarchy improves resource utilisation, reduces memory access latency and improves system performance. Specifically, the architectures proposed in this work can improve overall system performance by up to 19% with negligible storage, area and power overhead. While TCMPs continue to scale with the help of electrical, wireless and photonic NoCs, the proposed work can influence future design decisions.

***Index Terms*—Network-on-Chip (NoC), Tiled Chip Multi-Processor (TCMP), Network Stall Time, Cache Miss Penalty.**

I. Introduction

Multiple factors, including the continuance of Moore's Law [1], end of Dennard's Scaling [2] and limits of Instruction-Level Parallelism (ILP) [3] have led to the paradigm shift from uni-core to multi-core systems. 2015 International Technology Roadmap for Semiconductors (ITRS) report predicts that the increasing demand for information processing will drive a 30-fold increase in the number of processing cores by 2030 [4]. It is indeed visible in the industry with Intel Xeon Phi [5], AMD EPYC [6] and Ampere Altra [7] processors featuring up to 128 cores in their Tiled Chip Multi-Processors (TCMPs). As the number of cores continues to increase in TCMPs, on-chip communication plays a pivotal role in determining the performance. Global wires, shared buses, and monolithic crossbars have failed to provide the necessary communication infrastructure. As a result, a shared Network-on-Chip (NoC) based communication infrastructure is employed in most of the modern TCMPs [8]. Figure 1 shows the conceptual view of a standard NoC based TCMP.

NoC is a multi-hop packet-based communication infrastructure that connects cores through routers. NoC is preferred over others due to its high transfer bandwidth, scalability, and reliability. Applications running on different cores use it to access on-chip cache memories and off-chip main memory. NoC

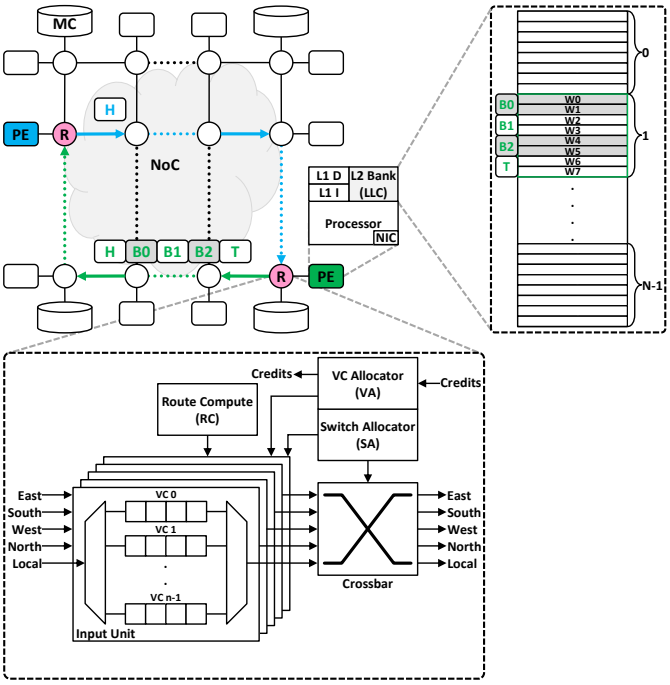


Figure 1: Conceptual view of an NoC based TCMP.

plays a significant role in determining the performance as it is responsible for 60% to 75% of the miss latency experienced by these applications [9]. Nevertheless, most of the existing memory access techniques and optimisations just focus on the memory hierarchy and treat the underlying NoC like a black box. The main reason behind this practice is the assumption that the on-chip communication latency is fixed (defined) and hardly changes over time. On the contrary, packets in NoC experience indefinite delays at each hop due to routing and arbitration decisions. Hence, the communication latency in NoC is undefined and varies according to the available traffic. As NoC plays a vital role in memory access latency, ignoring it may severely impact the overall performance of TCMPs.

Existing literature suggests that considering NoC and memory hierarchy together is the way forward in TCMP design [9]. Existing literature also considers efficient utilisation of NoC resources as a fundamental challenge in TCMP design, as currently, it is only about 20% [10]. This work accepts both; the suggestion as well as the challenge. There are innovative solutions available in the literature about using

NoC for caching and coherence [11][12][13][14][15]. Data and application-aware solutions are also available for efficient utilisation of NoC resources [16][17][18][19][20]. However, most of these solutions focus on independently optimising the NoC without bothering about its interaction with the memory hierarchy, thus leading to sub-optimal performance. This work attempts to establish a dynamic cooperation between NoC and the memory hierarchy for improved performance. By gathering information about the data travelling from one level of memory to another, we exploit underutilised NoC resources to design techniques and propose optimisations that reduce memory access latency and improve overall system performance[1].

II. Data-Aware Network-on-Chip

Existing techniques usually target individual shared resources, like NoC is optimised for packet latency, Last Level Cache (LLC) is optimised for hit rate, and Memory Controller (MC) is optimised for bank access latency, etc. These optimised metrics may not be directly related to the performance experienced by the applications. For example, service latency of packets might be hidden due to Memory-Level Parallelism (MLP) [22], and hence packet latency might not be indicative of the network-related stall time at the core (processor)[23]. Similarly, LLC is distributed across multiple cores as banks and requested data can be anywhere, from the nearest to the farthest bank. Hence, just the hit rate alone might not be indicative of the LLC access time at the processor [24]. Moreover, optimising individual shared resources for minimum service guarantees often leads to over-provision. Studies on Warehouse-Scale Computers (WSCs) have reported average resource utilisation between 10% and 50% only [25][26].

To design techniques and propose optimisations that can directly impact the overall system performance, shared resources must be considered together. So, rather than ignoring the NoC, it should be considered alongside the memory hierarchy. All the shared resources must also understand the characteristics and importance of the applications to improve their utilisation. However, labelling a particular application as more important than others at all times can be misleading, as for the same application, data requested at different times will have different importance to the processor. This work goes one level deeper, and instead of being application-aware, it focuses on designing data-aware NoC to improve the performance. While the characteristics and importance of the data are relatively well understood at the memory hierarchy, i.e. on-chip cache memories and off-chip main memory, less is known at the NoC. The complex and chaotic interaction between the cores in a TCMP makes it difficult for the NoC to understand about the data (packet) on its own. Additional challenges include varying packet injection rates, queuing delay, MLP, spatial location of the LLC banks and MCs, etc. Hence, the NoC must interact with the memory hierarchy to understand about the data travelling from one level of memory to another.

[1]This paper summarises the Ph.D. dissertation of author A. Das [21].

Table 1: System configuration (*varies in IV, V and VI)

Processor	64 OoO x86 cores, 1.3GHz
L1 Cache	32KB, 8-way, private, split
L2 Cache (LLC)	512KB×64 cores, 16-way, shared
Memory Bank	4; one located at each corner
Coherence	MESI/MOESI distributed directory
NoC	8×8 2D-Mesh topology, 128-bit channel 3 Virtual Networks (VNs) 3 Virtual Channels (VCs) per VN 1-flit depth control VC, 4-flit depth data VC
Routing	2-stage routers (1.54ns), XY-DOR algorithm VC based wormhole packet-switching
Packets	1-flit for control packet, 5-flit for data packet
Word/Flit/Block	64-bit/128-bit/64B; 2-words/flit, 8-words/block
Benchmarks	SPEC CPU2006 (multi-programmed) PARSEC 3.0 and SPLASH-2x (multi-threaded)

This work advocates for considering NoC and memory hierarchy together when designing techniques and proposing optimisations for TCMPs. It shows that designing data-aware NoC with the help of memory hierarchy improves resource utilisation, reduces memory access latency and improves system performance in TCMPs. To this end, our work makes the following contributions toward data-aware NoC design:

- Critical Packet Prioritisation [27]
- Critical Word Prioritisation [28], and
- Opportunistic Caching [29][30][31][32]

III. Evaluation Methodology

A. Simulation Infrastructure

All the architectures proposed in this work are modelled on event-driven gem5 simulator [33], and the system configuration is given in Table 1. It is similar to Intel Xeon Phi Processor 7235 [34] with shared and distributed L2 cache (LLC). The additional hardware-related area and power overheads are obtained using ORION 2.0 [35] at 65nm, and DSENT [36] and McPAT [37] at 22nm processor technology, all at 1GHz operating frequency. Some of the additional hardware are also implemented in structural RTL Verilog HDL and synthesised in Synopsis Design Compiler using a TSMC 65nm standard cell library at 1GHz operating frequency. For a relative comparison, most of the results are normalised with respect to a standard baseline architecture (without any optimisation).

B. Performance Metrics

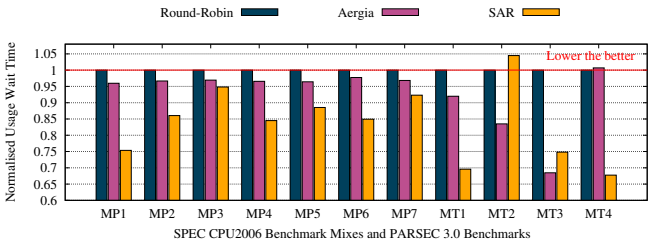
Though multiple metrics are considered to evaluate and analyse the performance, the most important of them are:

- **Network Stall Time (NST):** It is defined as the number of cycles a processor stalls waiting for a network packet.
- **L1 Cache Miss Penalty (MP_{L1}):** It is defined as the number of cycles required to replace an existing block in the L1 cache with the requested, incoming block.

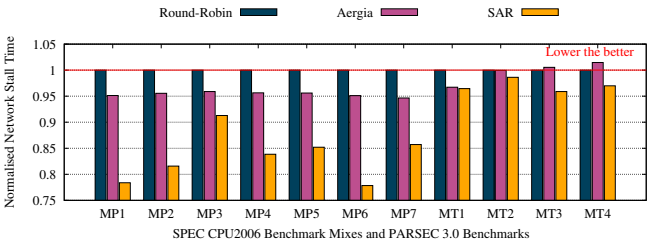
$$MP_{L1} = t^{L1-LLC}_{Request} + T^{LLC}_{Access} + t^{LLC-L1}_{Reply} \quad (1)$$

where

$$T^{LLC}_{Access} = \begin{cases} T^{LLC}_{Hit} & \text{if LLC Hit} \\ T^{LLC}_{Miss} + MP_{LLC} & \text{if LLC Miss} \end{cases} \quad (2)$$



(a) Usage wait time



(b) Network stall time

Figure 2: Performance of critical packet prioritisation

Here, T_j^i is the time taken by module i to complete a task j whereas, t_k^{i-j} is the time taken by message k to travel from module i to module j through the underlying NoC.

- **System Speedup (S):** For multi-programmed workloads:

$$S = \frac{IPC_{Proposed}}{IPC_{Baseline}} \tag{3}$$

where $IPC_{Baseline}$ and $IPC_{Proposed}$ are the total Instructions Per Cycle (IPC) of the baseline and proposed architectures, respectively. For multi-threaded workloads:

$$S = \frac{ExecTime_{Baseline}}{ExecTime_{Proposed}} \tag{4}$$

where $ExecTime_{Baseline}$ and $ExecTime_{Proposed}$ are the total execution time of the baseline and proposed architectures, respectively. We prefer execution time as multi-threaded workloads have synchronisation primitives like locks and barriers, which brings variation in IPC.

IV. Critical Packet Prioritisation

A. Introduction

Being a shared infrastructure, multiple cores can compete for an NoC resource at the same time. For example, multiple packets can request the same output port of a particular router. In that case, the employed arbitration policy uses some criteria to select a winner from the competitors. However, different packets travelling through the underlying NoC can have a very different impact on their application's performance. For example, some packets may be more critical and can not be delayed as they stall execution in the processor, whereas others may tolerate delay as their latency is hidden by the outstanding latency of their predecessors. We propose techniques and optimisations to make the routing and arbitration policy aware of the criticality of packets for better decision making.

B. Motivation

We consider a metric from the literature called *slack* that represents the relative importance (criticality) of packets [17]. Specifically, the slack of a packet is the number of cycles that packet can be delayed in the NoC without impacting the execution of the processor. Lower slack packets are more critical than higher slack packets, whereas no-slack packets are most critical. An existing work prioritises lower slack packets over higher slack packets but does not deal with the no-slack packets [17]. We observe that due to low packet injection rate

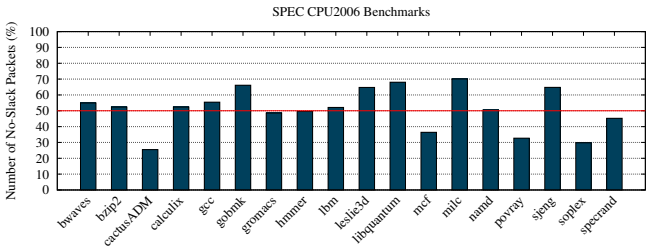


Figure 3: No-slack packets travelling through NoC routers

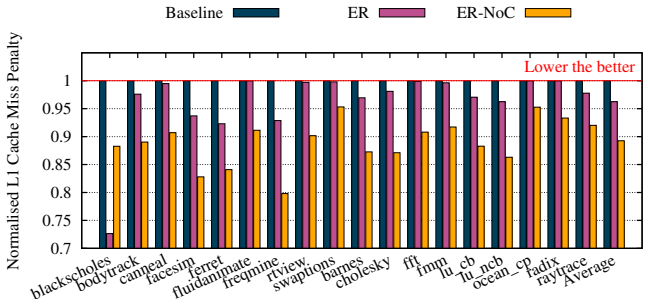
in modern applications, no-slack packets dominate the NoC routers. As shown in Figure 3, more than 50% no-slack packets can be found in most of the SPEC CPU2006 benchmarks. Since no-slack packets are the most critical ones, their delay in routers will severely hamper the application's performance.

C. Proposed Technique

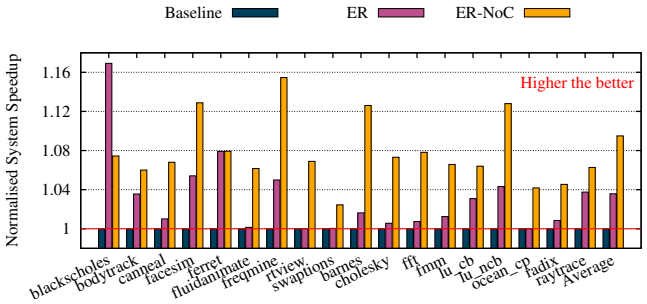
We obtain the slack of travelling packets at run-time by interacting with the on-chip cache controllers and use this slack as a priority during routing and arbitration decisions. We propose multiple techniques and optimisations to prioritise lower slack packets over their higher counterparts and find alternate minimal path for no-slack packets [27]. We adopt a look-ahead routing to facilitate the re-routing of no-slack packets through the new path. Our proposed prioritisation based architecture called Slack-Aware Re-routing (SAR) makes sure that no-slack packets are not delayed in the NoC and reach their destination at the earliest to resume execution. Compared to the state-of-the-art Aergia architecture [17], SAR is capable of significantly improving the overall system performance.

D. Performance Analysis

Usage wait time is the number of cycles a reply packet waits from its arrival at the destination until being used (consumed) by the processor. Usage wait time shows how early or late reply packets reach their destination than necessary. Figure 2a shows the normalised usage wait time with respect to the baseline architecture (Round-Robin). The proposed SAR achieves a significant reduction between 5% to 25% in usage wait time. Figure 2b shows the normalised network stall time for the presented workload mixes with respect to the Round-Robin. Our prioritisation policy in SAR significantly reduces network stall time by up to 22% over Round-Robin and 18% over Aergia. While we have a negligible area and leakage



(a) L1 cache miss penalty



(b) System speedup

Figure 4: Performance of critical word prioritisation

power overhead of 1.70% and 2.10%, respectively, dynamic power reduces by 7.50% due to performance improvement.

V. Critical Word Prioritisation

A. Introduction

A processor usually requests for a single word called *critical word* from the memory hierarchy [38]. When it encounters an L1 cache miss on the critical word, a data transfer from the next level of memory is triggered. The smallest unit of data transfer between different levels of memory is in the unit of blocks containing multiple words. So, even though the processor requests a single word, an entire data block (containing the critical word) is brought from memory as a packet through the NoC. Transfer bandwidth in NoC is limited to the channel width called *flit* (much smaller than a packet, refer Table 1). Hence, a data block (packet) is divided into multiple flits and sent to the requester, as shown in green in Figure 1. The critical word can be in any of the incoming flits with indefinite delay along the way. We propose techniques and optimisations to make the routing and arbitration policy aware of the critical words for better decision making.

B. Motivation

Figure 5 shows the average position of critical word in the corresponding blocks requested from the L2 cache (LLC) for PARSEC 3.0 and SPLASH-2x benchmarks. There is a very interesting trend: *the first word is the critical word for most of the requested blocks*. The trend is unusual but not unreasonable, as the existing literature has proof of critical word regularity [39][40]. Literature also states that it is reasonable to expect that data in a given region may be accessed in similar order on multiple occasions. Hence, the first incoming flit carries the critical word most of the time since it carries the first two words (refer Table 1) of the requested data block. If the flit carrying the critical word can be prioritised to minimise the routing and arbitration delay and reach its destination, the (stalled) processor can resume its execution at the earliest.

C. Proposed Technique

Two of the most popular memory access optimisations to reduce miss penalty of the critical word are *Early Restart (ER)* and *Critical Word First (CWF)* [38]. However, they are NoC

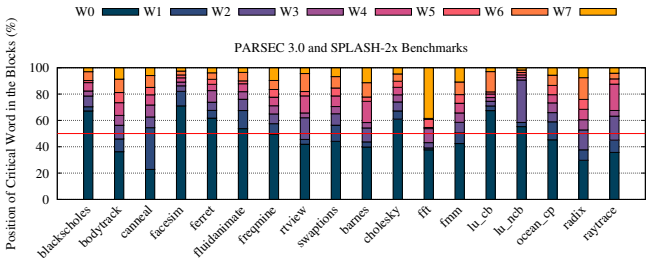
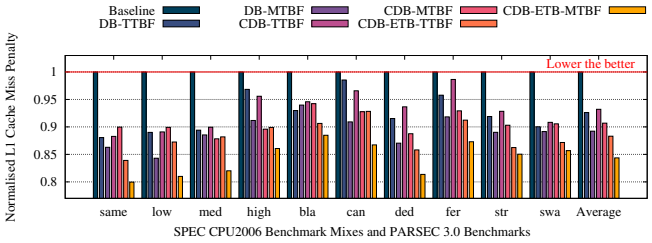


Figure 5: Position of critical word in the requested blocks

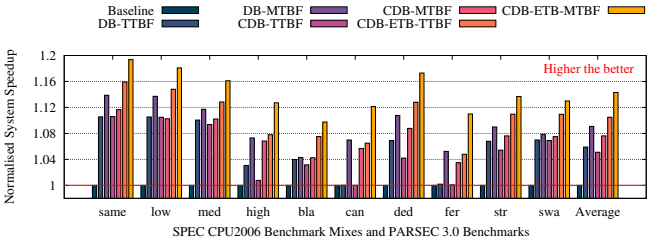
oblivious and, when employed in NoC based TCMPs, neglect the concept of flit based discrete transfer and router delay. As a result, the effectiveness of ER and CWF-like optimisations is less in modern TCMPs. As shown in Figure 5, since a majority of the benchmarks have their critical word in the first flit itself, CWF might not be even necessary. We propose an NoC-aware ER optimisation where the flits carrying the critical words are prioritised to reach their destination as soon as possible [28]. The proposed technique involves dynamic cooperation between L1 cache controllers, NoC and the LLC controller to propagate the information about the critical word.

D. Performance Analysis

For the baseline architecture (no-optimisation), L1 cache miss penalty is defined in Equation (1). Whereas for ER and ER-NoC, it is defined as the number of cycles required to receive the critical word while the requested block is brought into the L1 cache. Figure 4a shows the normalised L1 cache miss penalty with respect to the baseline. L1 cache miss penalty directly reflects the effectiveness of the ER-NoC over others. The proposed prioritisation in ER-NoC significantly reduces the L1 cache miss penalty by 11%. Figure 4b shows the normalised system speedup, and as expected, the reduction in L1 cache miss penalty translates into the improvement of overall system performance. ER-NoC architecture achieves a maximum and an average system speedup of 15% and 9%, respectively. While we have a negligible area and leakage power overhead of 1.34% and 3.60%, respectively, dynamic power reduces by 5.85% due to performance improvement.



(a) L1 cache miss penalty



(b) System speedup

Figure 6: Performance of opportunistic caching

VI. Opportunistic Caching

A. Introduction

Due to limited on-chip caching, applications with large memory footprint encounter frequent data misses. Such applications suffer from recurring miss penalty when they re-reference recently evicted cache blocks. On the other side, due to the very low packet injection rate of only around 5% in modern applications [41][42][43], and post-silicon debug and validation, a lot of NoC resources remain underutilised or even worse, unused. We propose techniques and optimisations to exploit these available resources and accommodate evicted cache blocks in the routers to service future references quickly.

B. Exploiting Underutilised Router Buffers

1) Motivation: To maintain the worst-case performance bandwidth and scalability, NoC routers are provisioned with input port buffers [44][45], which are further divided into Virtual Channels (VCs) for deadlock-free routing and better utilisation [46]. As shown in Figure 1, packets entering through different input ports (east, south, west, north, and local) get temporarily stored in the available VCs and take part in routing and arbitration decisions. However, due to the low packet injection rate, VCs are underutilised and remain free most of the time. VC availability in local input port of routers is shown in Figure 7 for different SPEC CPU2006 benchmarks. Since these benchmarks have an average packet injection rate of only around 5%, at least one VC always remain free ($\approx$ 95%).

2) Proposed Technique: We propose to store evicted cache blocks in underutilised VCs without hampering the usual NoC transfer [29][30]. Such a block can either be *clean* or *dirty*, where the former is discarded while the latter is sent to the next level of memory for write-back. We store the evicted, dirty cache blocks in the local input port VCs on their way for the write-back. When they are re-referenced, we arrange a local reply with the matching stored block from the routers. Local reply completely avoids the NoC and off-chip travel and significantly reduces miss penalty, and improves performance. We also bring some evicted, clean cache blocks to the routers (which are otherwise discarded) and store them in the local input port VCs to increase our chances of a local reply.

C. Exploiting Unused Trace Buffers

1) Motivation: When we bring evicted, clean cache blocks to be stored in the routers to increase our chances of local reply, they compete against the evicted, dirty cache blocks for a place. Input port VCs are underutilised, but they are also limited in numbers. Thus the competition between clean and dirty defeats the purpose of accommodating more evicted cache blocks. Due to the design complexity of NoC based TCMPs, post-silicon debug is practised to validate a design before going into production. An important phase of the debug involves validating the on-chip interaction between cores. To aid the process, Design-for-Debug (DfD) hardware are embedded across various modules and cores in a TCMP [47]. *Trace Buffers* are such a DfD hardware embedded in routers. However, when a design goes into production, most of the DfD hardware become non-functional and are left unused.

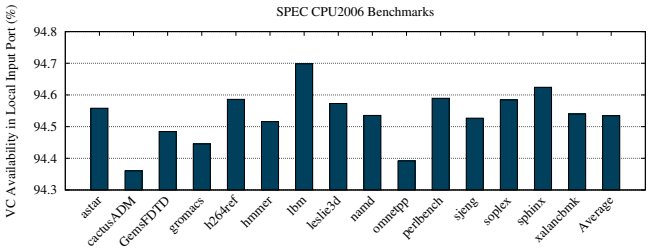


Figure 7: VC availability in local input port of NoC routers

2) Proposed Technique: Similar to the input port VCs, the trace buffers are already present in the routers; hence they do not contribute to on-chip area overhead. We re-purpose them to store evicted, clean cache blocks [31][32]. With the dirty blocks stored in input port VCs and the clean blocks stored in embedded trace buffers, we are able to accommodate more evicted cache blocks, which increases our chances of local reply. We also propose two techniques to forward stored, dirty cache blocks for write-back towards their respective destination and a technique to drop stored, clean cache blocks from the trace buffers. To preserve the state of evicted blocks and maintain coherence, we propose an additional coherence message. Our proposed technique is able to significantly reduce miss penalty and improve overall system performance.

D. Performance Analysis

L1 cache miss penalty directly reflects the effectiveness of the proposed local store and reply optimisation. Figure 6a shows the normalised L1 cache miss penalty with respect to the baseline architecture (no-optimisation). With local replies, the proposed architectures reduce the L1 cache miss penalty for all the simulated workload mixes. A maximum reduction of 21% and an average reduction of 16% is achieved. Figure 6b

shows the normalised system speedup with respect to the baseline, where the improvement is intuitive due to reduced miss penalty. We achieve a maximum and average system speedup of 19% and 14%, respectively. Intelligent usage of trace buffers in the proposed architectures significantly improves overall system performance with frequent local replies from the routers. While we have a negligible area and leakage power overhead of 2.58% and 3.94%, respectively, dynamic power reduces by 6.12% due to performance improvement.

VII. Future Research Directions

The future is moving towards domain-specific, heterogeneous architectures, and NoC will be required to smoothly integrate specific Intellectual Property (IP) cores from diverse domains. One way of looking into improving our packet prioritisation technique will be to get the criticality metric from the IP cores, as they might have specific QoS requirements. Hence, a hardware-software co-design may be explored. Another possible way to extend our technique is by considering different routing algorithms for different VCs within the same input port. This will reduce the conflict between packets.

The number of processing cores is massively increasing in Domain-Specific Architectures (DSAs), and they have frequent memory interactions. If processing cores only need the critical word to continue their execution, NoC channel bandwidth can be divided among all the competitors to transfer multiple critical words to different cores rather than sending the entire block. This could be one possible way of extending our critical word prioritisation technique for DSAs. Another possible extension could be in the line of NoC traffic compression.

Our work on opportunistic caching can be extended to store pre-fetched blocks to avoid cache pollution. It may also be explored for Non-Volatile Memories (NVMs) to reduce expensive writes and increase their lifetime. As we advocate for the design of data-aware NoC, emerging technologies like wireless and photonic NoC (WiNoC and ONoC) provide even better scope. One such example is the design of coherence protocols by taking advantage of the WiNoC. Neuromorphic architectures like Spiking Neural Network (SNN) use NoC for scalability. They exhibit higher memory interactions, where designing data-aware NoC might become a game-changer.

References

[1] G. E. Moore, "Cramming More Components onto Integrated Circuits," *Electronics*, 1965.

[2] R. H. Dennard *et al.*, "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions," *IEEE JSSC*, 1974.

[3] D. W. Wall, "Limits of Instruction-Level Parallelism," in *ASPLOS*, 1991.

[4] (2015) International Technology Roadmap for Semiconductors (ITRS). [Online]. Available: https://tinyurl.com/2015-itrs-report

[5] (2017) Intel Xeon Phi Processors. [Online]. Available: https://tinyurl.com/intel-xeon-phi-processors

[6] (2021) AMD EPYC Processors. [Online]. Available: https://tinyurl.com/amd-epyc-processors

[7] (2021) Ampere Altra Processors. [Online]. Available: https://tinyurl.com/ampere-altra-processors

[8] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Inteconnection Networks," in *DAC*, 2001.

[9] D. Sanchez *et al.*, "An Analysis of On-Chip Interconnection Networks for Large-Scale Chip Multiprocessors," *ACM TACO*, 2010.

[10] H. Farrokhbakht *et al.*, "SMART: A Scalable Mapping and Routing Technique for Power-Gating in NoC Routers," in *NOCS*, 2017.

[11] H. E. Mizrahi *et al.*, "Introducing Memory into the Switch Elements of Multiprocessor Interconnection Networks," in *ISCA*, 1989.

[12] N. Eisley *et al.*, "In-Network Cache Coherence," in *MICRO*, 2006.

[13] A. Yanamandra *et al.*, "In-Network Caching for Chip Multiprocessors," in *HiPEAC*, 2009.

[14] L. Huang, "Leveraging On-Chip Networks for Efficient Prediction on Multicore Coherence," in *DATE*, 2014.

[15] Y. Yao and Z. Lu, "iNPG: Accelerating Critical Section Access with In-Network Packet Generation for NoC Based Many-Cores," in *HPCA*, 2018.

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

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

[18] R. Das *et al.*, "Application-to-Core Mapping Policies to Reduce Memory System Interference in Multi-Core Systems," in *HPCA*, 2013.

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

[20] Z. Li *et al.*, "The Runahead Network-on-Chip," in *HPCA*, 2016.

[21] A. Das, "Designing Data-Aware Network-on-Chip for Performance," Ph.D. dissertation, Indian Institute of Technology (IIT) Guwahati, 2021.

[22] A. Glew, "MLP yes! ILP no!" in *ASPLOS WACI*, 1998.

[23] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," in *MICRO*, 2007.

[24] C. Kim *et al.*, "An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches," in *ASPLOS*, 2002.

[25] D. Lo *et al.*, "Towards Energy Proportionality for Large-Scale Latency-Critical Workloads," in *ISCA*, 2014.

[26] L. A. Barroso *et al.*, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 2nd ed. Morgan & Claypool Publishers, 2013.

[27] A. Das *et al.*, "Critical Packet Prioritisation by Slack-Aware Re-routing in On-Chip Networks," in *NOCS*, 2018.

[28] A. Das *et al.*, "Data Criticality in Multi-Threaded Applications: An Insight for Many-Core Systems," *IEEE TVLSI*, 2021.

[29] A. Das *et al.*, "Reducing Off-Chip Miss Penalty by Exploiting Under-utilised On-Chip Router Buffers," in *ICCD*, 2020.

[30] A. Das *et al.*, "Revising NoC in Future Multi-Core based Consumer Electronics for Performance," *IEEE CEM*, 2021.

[31] A. Das *et al.*, "Exploiting On-Chip Routers to Store Dirty Cache Blocks in Tiled Chip Multi-Processors," in *ISVLSI*, 2020.

[32] A. Das *et al.*, "Opportunistic Caching in NoC: Exploring Ways to Reduce Miss Penalty," *IEEE TC*, 2021.

[33] N. Binkert *et al.*, "The gem5 Simulator," *ACM SIGARH CAN*, 2011.

[34] (2017) Intel Xeon Phi Processor 7235. [Online]. Available: https://tinyurl.com/intel-7235-processor

[35] 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.

[36] C. Sun *et al.*, "DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic NoC Modeling," in *NOCS*, 2012.

[37] S. Li *et al.*, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Arch." in *MICRO*, 2009.

[38] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. Elsevier, 2017.

[39] E. J. Gieske, "Critical Words Cache Memory," Ph.D. dissertation, University of Cincinnati (UC), 2008.

[40] N. Chatterjee *et al.*, "Leveraging Heterogeneity in DRAM Main Memories to Accelerate Critical Word Access," in *MICRO*, 2012.

[41] N. Barrow-Williams *et al.*, "A Communication Characterisation of Splash-2 and Parsec," in *IISWC*, 2009.

[42] P. Gratz and S. W. Keckler, "Realistic Workload Characterization and Analysis for Networks-on-Chip Design," in *CMP-MSI*, 2010.

[43] R. Hesse *et al.*, "Fine-Grained Bandwidth Adaptivity in Networks-on-Chip using Bidirectional Channels," in *NOCS*, 2012.

[44] G. Michelogiannakis *et al.*, "Evaluating Bufferless Flow Control for On-Chip Networks," in *NOCS*, 2010.

[45] B. K. Daya *et al.*, "Quest for High-Perf Bufferless NoCs with Single-Cycle Express Paths and Self-Learning Throttling," in *DAC*, 2016.

[46] W. Dally and C. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE TC*, 1987.

[47] B. Vermeulen and S. K. Goel, "Design for Debug: Catching Design Errors in Digital Chips," *IEEE D&T*, 2002.