Opportunistic Caching in NoC: Exploring Ways to Reduce Miss Penalty

Abhijit Das, *Student Member, IEEE,* Abhishek Kumar, John Jose, *Member, IEEE,* and Maurizio Palesi, *Senior Member, IEEE*

Abstract—Due to limited on-chip caching, data-driven applications with large memory footprint encounter frequent cache misses. Such applications suffer from recurring miss penalty when they re-reference recently evicted cache blocks. To meet the worst-case performance requirements, Network-on-Chip (NoC) routers are provisioned with input port buffers. However, recent studies reveal that these buffers remain underutilised except during network congestion. Trace buffers are Design-for-Debug (DfD) hardware employed in NoC routers for post-silicon debug and validation. Nevertheless, they become non-functional once a design goes into production and remain in the routers left unused. In this work, we exploit the underutilised NoC router buffers and the unused trace buffers to store recently evicted cache blocks. While these blocks are stored in the buffers, future re-reference to these blocks can be replied from the NoC router. Such an opportunistic caching of evicted blocks in NoC routers significantly reduce the miss penalty. Experimental analysis shows that the proposed architectures can achieve up to 21% (16% on average) reduction in miss penalty and 19% (14% on average) improvement in overall system performance. 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 the improvement in performance.

Index Terms—Network-on-Chip (NoC), Miss Penalty, Cache Coherence, Virtual Channel (VC), Embedded Trace Buffer (ETB).

1 Introduction

In the era of data-driven applications, the demand for information processing is increasing exponentially. 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 [1]. It is indeed visible in the industry, for example with Intel Xeon Phi Processors featuring 64-72 cores in their Tiled Chip Multi-Processors (TCMPs) [2]. With the increasing cores in TCMPs, scalable Network-on-Chip (NoC) communication plays a very significant role in data access latency. However, for standard applications, the average packet injection rate is only around 5% in NoC based TCMPs [3][4][5]. Low packet injection directly translates into poor utilisation of available NoC resources. While TCMPs continue to scale, proposing policies to improve NoC resource utilisation is a necessary step forward. Improving NoC resource utilisation can reduce data access latency and positively impact overall system performance [6].

Due to the increasing core counts, limited on-chip area and associated cost, most of the modern TCMPs have only two levels of on-chip caching [7][8][9]. They usually have private, write-back L1 caches and a shared and distributed, write-back L2 cache. When an L1 cache miss occurs, the requested cache block is fetched from the corresponding L2 cache bank. A cache miss in L2 requires the block to be fetched from off-chip memory. The entire communication is packet based and is done through the underlying NoC. The time required to replace an existing block in L1 with an incoming, requested cache block is called miss penalty. Since L1 caches are small and frequently accessed, an L1 cache miss almost always evicts a valid cache block. Based on whether an evicted, valid block is clean or dirty (modified), it is either discarded or sent to the L2 cache bank for write-back. However, due to temporal and/or spatial locality, if a recently evicted block from L1 cache is re-referenced, it needs to be fetched again. The re-referenced block is fetched from the corresponding L2 cache bank. Since L2 is distributed, the corresponding L2 cache bank can be located anywhere, in the nearest, to the farthest core. In the worst case, when the re-referenced block is not present in the L2 cache bank (L2 miss), it is fetched from off-chip memory. Increasing the L1 cache size may delay the block eviction and avoid cache miss penalty up to an extent. However, it is not feasible, as increasing the cache size may impact its hit time and affect instruction pipeline. Increasing the cache size is also not feasible due to the on-chip area and associated cost constraints. In any way, experiencing cache miss penalties on re-reference of recently evicted blocks is inevitable. Frequent cache miss penalties severely hamper application execution time and degrade overall system performance.

Packet based NoC use routers to establish on-chip communication between the available cores in a TCMP. Modern NoC based TCMPs employ input buffered routers for scalable on-chip bandwidth [10][11]. Packets on their way from source to destination are temporarily stored in the input port buffers of NoC routers. Stored packets take part in routing and arbitration and get forwarded towards destination as soon as they get the desired output port. However, due

- *A. Das and J. Jose are with the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Assam 781039, India.*
  E-mail: {abhijit.das, johnjose}@iitg.ac.in
- *A. Kumar is with Oracle Inc., Bengaluru 560029, India.*
  E-mail: abhishek18a@iitg.ac.in
- *M. Palesi is with the Department of Electrical, Electronics and Computer Engineering, University of Catania, Catania 95124, Italy.*
  E-mail: maurizio.palesi@dieei.unict.it

Manuscript received August 00, 0000; revised August 00, 0000.

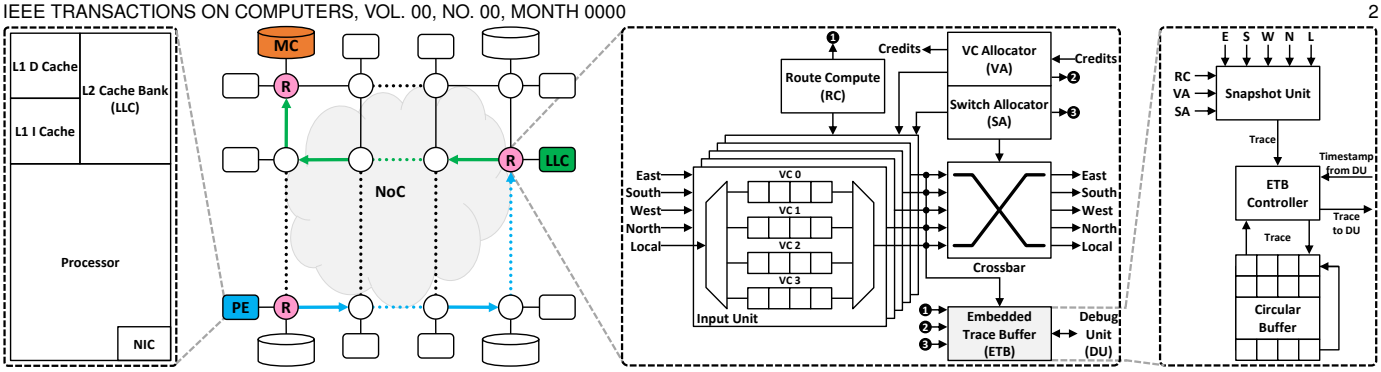


Figure 1: Conceptual view of an NoC based TCMP

to the low packet injection rate, input port buffers in NoC routers are underutilised. Experimental analysis with standard applications shows that buffer utilisation of routers is very low, except during peak NoC congestion (Section 2.2).

Due to the design complexity of NoC based TCMPs, post-silicon debug is usually practised to validate a proposed design before going into the production. To aid post-silicon debug and validation, Design-for-Debug (DfD) hardware are embedded across various cores and modules of a TCMP [12]. An important phase of the debug involves validating the on-chip interaction between different cores. *Trace buffers* are DfD hardware embedded in NoC routers to record their state for post-silicon debug and validation. However, when a TCMP design goes into production, most of the DfD hardware become non-functional. Since the usage of DfD hardware, including trace buffers, is sporadic and rare after the production, most of them are left unused.

In this work, we exploit the underutilised storage space available in NoC routers to store evicted L1 cache blocks. If an evicted L1 cache block is dirty, it is sent to the corresponding L2 cache bank for write-back. Such blocks enter the local router [1] as packets, gets stored in local input port buffers, and take part in routing and arbitration to reach their destination for write-back. We propose to disable the arbitration of such blocks and keep them stored in the local router buffers for as long as possible. Since buffer utilisation is low, evicted, dirty L1 cache blocks can be kept stored in the local routers without inducing any NoC congestion. If an evicted L1 cache block is clean, it is discarded and not sent for write-back as the corresponding L2 cache bank has the same copy of the block. We propose to send such blocks to the local router and keep them stored in the unused embedded trace buffer. Now, when a recently evicted L1 cache block is re-referenced, we propose to arrange a quick reply with the stored block from the local router. It is possible as the recently evicted cache block might be present (stored) either in the local input port buffer or the embedded trace buffer. These optimisations can significantly reduce the L1 cache miss penalty and improve overall system performance. In this work, we make the following major contributions:

- **Reply with Stored Dirty Blocks:** We identify evicted, dirty L1 cache blocks when they enter the local routers to travel for write-back towards their destination L2 cache bank. We propose to disable arbitration of such blocks to keep them stored in local input port buffers. Future re-reference to the stored blocks are replied by the local router to reduce miss penalty.
- **Reply with Stored Clean Blocks:** To increase the chances of local reply, we propose to keep the evicted, clean L1 cache blocks in the unused, embedded trace buffer. Local router can reply to the future re-references from local input port buffers as well as trace buffer which reduces miss penalty even further.
- **Forward/Drop of Stored Blocks:** We propose two techniques to forward dirty L1 cache blocks stored in the local input port buffers. A time-triggered technique based on a certain time threshold, and a message-triggered technique based on a request from L2. We also propose a technique to drop clean L1 cache blocks stored in trace buffer and inform L2.
- **Maintain Cache Coherence:** To preserve the states of evicted, L1 cache blocks, we propose a new coherence message. Since L2 cache is shared, we make sure that the proposed optimisations of local store and reply do not violate the cache coherence.

1. Local router connects a tile (core) to the underlying NoC.

2 Background

Conceptual view of an NoC based TCMP is shown in Figure 1 for reference as we explain the necessary background.

2.1 L1 Cache Miss Penalty

Since L1 caches are small, a cache miss is likely to occur against a requested block. On an L1 cache miss, the requested block is fetched from the next level of memory (L2 cache). In NoC based TCMPs, the L2 cache is usually divided into multiple banks and distributed across all the cores, as shown in Figure 1. Hence, the requested block needs to be fetched from the corresponding L2 cache bank, which can be anywhere, in the nearest, to the farthest core. If the requested block is not present in the L2 cache bank (L2 miss), it is fetched from the next level of memory. Latest NoC based TCMPs like Intel Xeon Phi Processor (2016) [7], Princeton Piton Processor (2015) [8], MIT Scorpio Processor (2014) [9] and others, use only two levels of on-chip caching. In these systems, L2 serves as the last level cache (LLC), and a cache miss in L2 requires the block to be fetched from off-chip memory. Hence, in the worst case, L1 cache miss on a requested block requires the block to be fetched from the off-chip memory, which is very time expensive. The time required to replace an existing block in L1 with an incoming

cache block is called L1 cache miss penalty. In NoC based TCMPs, L1 cache miss penalty (MP_{L1}) can be given as:

$$MP_{L1} = t_{L1-LLC}^{Request} + T_{LLC}^{Access} + t_{LLC-L1}^{Reply} \quad (1)$$

where

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

here, T_i^j is the time taken by module i to complete a task j whereas, t_{i-j}^k is the time taken by message k to travel from module i to module j through the underlying NoC. For example, T_{LLC}^{Access} is the time taken by an LLC bank to access a cache block whereas, $t_{L1-LLC}^{Request}$ is the time taken by a cache miss request to travel from L1 cache to the LLC bank.

As given in equation (1), L1 cache miss penalty is dominated by on-chip transfer time (t_{i-j}^k). In the worst case of an LLC miss, the dominance of transfer time is even more since the miss request and reply travels to and from the memory controller (MC). The corresponding MC can be located in any of the corner routers (refer Figure 1). During on-chip congestion, the transfer time can get longer due to unknown router delay along the way. Hence, the underlying NoC plays an important role in L1 cache miss penalty.

2.2 VC Availability

NoC based systems employ routers, which have three design alternatives: input buffered, minimally buffered and bufferless; each with different pros and cons. To meet the worst case performance bandwidth, modern TCMPs prefer input buffered NoC routers [10][11]. Input buffers are further divided into virtual channels (VCs) for deadlock-free routing and better utilisation [13]. 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. VC availability in NoC based TCMPs can be given as:

$$VC\ Availability_n = \frac{Cycles\ when\ n\ VCs\ are\ Free}{Total\ Execution\ Cycles} \quad (3)$$

Figure 2 shows the VC availability in local input port of NoC routers for a set of standard multi-programmed benchmarks (SPEC CPU2006 [14]). As the average injection rate of these benchmarks is only around 5%, except during peak NoC congestion, at least one VC is always free ($\approx$ 95%). A similar observation is expected for standard multi-threaded benchmarks (PARSEC 3.0 [15]) where the average injection rate is even lower. The observation in Figure 2 is in sync with the conclusions in the available literature [3][4][5].

> NoC based TCMPs use input buffered routers for worst case bandwidth, but buffers (VCs) remain underutilised.

2.3 Embedded Trace Buffer

Pre-silicon validation is a standard practice in the process of any hardware system design. It involves theory-based

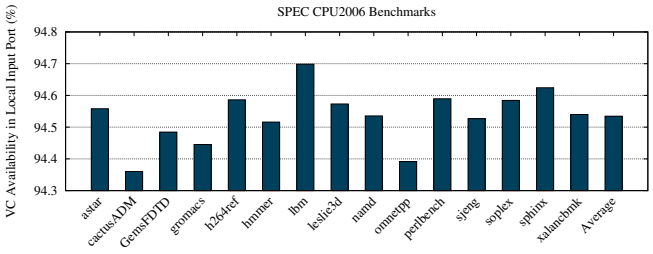


Figure 2: VC availability in local input port

formal verification of the design for functional correctness and simulation-based verification of the RTL description [16]. However, increasing core counts and the need for an efficient and scalable communication increase the design complexity of NoC based TCMPs. For such systems, theory-based formal verification suffers from state space explosion problem. Furthermore, simulation-based verification is very slow. Hence, exhaustively exploring the entire design space of a TCMP is not feasible in a time-bound pre-silicon validation. Thus, post-silicon debug and validation is necessary.

Post-silicon debug and validation begins when the first few silicon prototypes of the proposed design are available. Longer tests are run on actual hardware (prototype) in native speed for thorough validation. Hence, post-silicon validation can expose functional bugs that might have been missed during pre-silicon validation. Key to an effective post-silicon debug and validation lies in the observability and controllability of internal signals when the tests are run. To facilitate debug and validation, Design-for-Debug (DfD) hardware are embedded across various cores and modules of a TCMP [12]. DfD hardware can trace internal signals, dump contents of registers and memory, patch microcode and firmware, create user-defined triggers and interrupts, etc. An important phase of debugging NoC based TCMPs is to validate the on-chip interaction between different cores. *Trace buffers* are DfD hardware embedded in NoC routers to record their state for post-silicon debug and validation. Trace buffer and embedded trace buffer (ETB) are used interchangeably throughout the text, thus should not be confused with. Trace buffers periodically take snapshot of the NoC router and stores it as a compressed trace in a circular memory storage, as shown in Figure 1. Size of trace buffers typically varies between 2KB - 8KB in different modules and can roughly store 10K - 30K lines of compressed trace. After successful debug and validation, the silicon prototype goes for mass production. Thereafter, most of the DfD hardware including trace buffers become non-functional. Since the usage of DfD hardware is sporadic and rare after the production, most of them are left unused. Even though the DfD hardware are power-gated, their area footprint remains in the routers (chip) without any benefit.

> Trace buffers facilitate post-silicon debug and validation of NoC routers, but they are left unused after production.

3 Motivation

Since L1 caches are small and frequently accessed, an L1 cache miss almost always evicts a valid block. However,

due to temporal locality, a recently evicted L1 cache block may be re-referenced. Since a cache block contains multiple words, even for spatial locality, a recently evicted L1 cache block may be re-referenced. The duration from the eviction of an L1 cache block to the request of the same block in future, by the same core is called re-reference time. Re-reference time (RT) in NoC based TCMPs can be given as:

$$RT_i^j = |Request(B_i^j)|_{T_y} - |Eviction(B_i^j)|_{T_x} \quad (4)$$

where at time T_x, cache block i was evicted from core j and in the future at time T_y, block i is requested again by the same core j. Figure 3 shows the average re-reference time for different SPEC CPU2006 [14] and PARSEC 3.0 [15] benchmarks. For example, in a 64-core NoC based TCMP running a multi-programmed benchmark *astar*, an evicted L1 cache block is re-referenced within an average time of 6532 cycles. Across all benchmarks, on average, within a small interval of around 12000 cycles, an evicted L1 cache block is re-referenced. This interval indirectly indicates how frequently an NoC based TCMP suffers from L1 cache miss penalty due to unfortunate block evictions.

In this work, we explore ways to reduce L1 cache miss penalty in NoC based TCMPs. Evicted, clean L1 cache blocks are discarded, whereas dirty L1 blocks are sent over the NoC to the corresponding L2 cache bank for write-back. To reach their destination for write-back, evicted, dirty L1 cache blocks enter the local router through the local input port as packets. They temporarily get stored in the available VCs and take part in routing and arbitration decisions to get the desired output port. In this work, we propose to disable the arbitration of such evicted, dirty L1 cache blocks while they are stored in the local VCs. Without taking part in the arbitration, these evicted, dirty blocks can not get the desired output port and leave the local router. From the observation in Figure 2, we know that local input VCs remain underutilised (free) most of the time. Hence, we can keep the evicted, dirty L1 cache blocks stored in the local router for as long as possible without creating injection suppression for other packets. During the time an evicted, dirty L1 cache block is locally stored, a re-reference request for the same block by the same core can be locally replied. We propose to generate direct reply from the local router if a requested block is present in the local VCs. From equation (1) and Section 2.1, we know that L1 cache miss penalty involves on-chip travel and may also suffer from NoC communication delay. Local reply to L1 cache miss requests from the NoC routers can avoid the on-chip travel altogether and get significant reduction in miss penalty.

Unlike dirty blocks, clean blocks are discarded after eviction from L1 cache since a write-back is not necessary. However, the number of clean blocks evicted from L1 cache is much more than the number of dirty blocks. From the observation in Figure 3, we are aware that an evicted L1 cache block (clean or dirty) is re-referenced within a small interval of around 12000 cycles. Hence, to improve the chances of local reply, we propose to bring evicted, clean L1 cache blocks to the local routers and keep them stored in local VCs. But, the underutilised local VCs are already employed to store evicted, dirty L1 cache blocks. Making the evicted, clean and dirty blocks compete against each other for a place in the local VCs kill the purpose of local

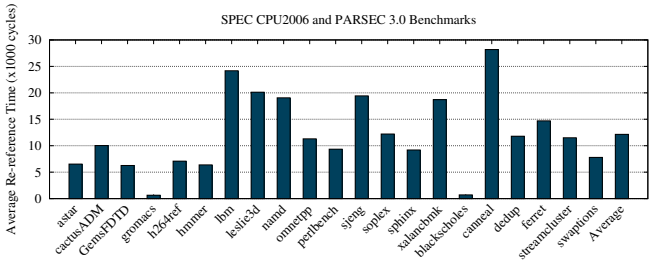


Figure 3: Re-reference time of evicted L1 cache blocks

store. From the conclusion in Section 2.3, we know that a DfD storage infrastructure called trace buffer, embedded in NoC routers is left unused. In this work, we re-purpose the unused trace buffer in NoC routers to store evicted, clean L1 cache blocks; which are normally discarded. Until a block is stored in the trace buffer, a re-reference to the same block by the same core can be serviced from the local router. With the evicted, dirty blocks stored in local VCs and the evicted, clean blocks now stored in the trace buffer, our chances of local reply increase many-fold. Our proposed optimisations to generate immediate local reply from NoC routers can significantly reduce L1 cache miss penalty, thereby improving overall system performance.

Proposed Solution:

Store evicted L1 cache blocks in underutilised NoC router buffers (VCs) and unused trace buffers (ETBs). Upon re-reference, generate direct reply from the local routers.

4 Opportunistic Caching in NoC

A conceptual view of the router microarchitecture that implements our proposed optimisations is given in Figure 4. We consider two-level on-chip caching with MOESI distributed directory coherence protocol. Keeping Figure 4 and MOESI protocol as reference, we explain the working of our proposed architecture in the following sub-sections.

In MOESI distributed directory coherence based cache organisation, a cache block can be in **Modified (M):** Possibly different from memory and only copy, **Owned (O):** Possibly different from memory and possibly shared, **Exclusive (E):** Same as memory and only copy, **Shared (S):** Same as memory/owner and possibly shared, or **Invalid (I):** Invalid copy state. Exclusive (E) state can be considered as a subset of Owned (O) state. Our discussion includes the following coherence messages from the protocol. GETS/GETX: Read/Write request, DATA-GETS/DATA-GETX: Shared/Exclusive data, PUTS/PUTO/PUTM: Write-back request for Shared/Owned/Modified data, ACK-PUTS/ACK-PUTO/ACK-PUTM: Acknowledgement for PUTS/PUTO/PUTM write-back, DATA-PUTS/DATA-PUTO/DATA-PUTM: Shared/Owned/Modified data for write-back, UNBLOCK: Intimation for DATA-PUTS drop.

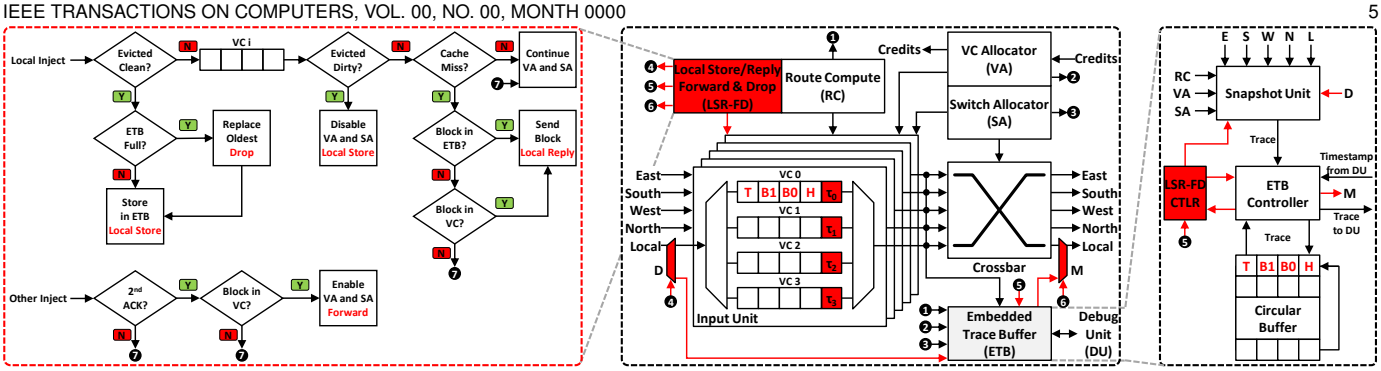
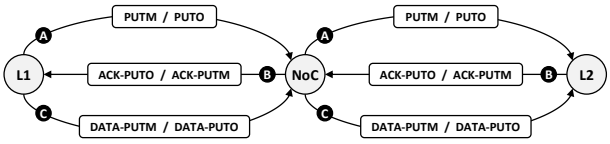


Figure 4: Conceptual view of the proposed router microarchitecture. All the additional units and links are shown in red. Evicted L1 cache blocks enter the NoC as packets and get divided into multiple smaller units called flits (H, B0, B1, T). Based on whether a block is clean or dirty, the corresponding flits get stored in either the trace buffer or the local VCs.

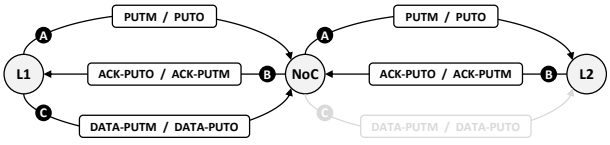
Figure 5: Modified message/packet header

4.1 Block Store in Router Buffers

A valid block evicted from L1 cache can be either clean (shared) or dirty (owned/modified). Clean blocks are discarded and dirty blocks are sent for write-back. For evicted, dirty blocks, a PUTO/PUTM write-back request is initiated from L1 cache controller (L1 CTLR) to the corresponding L2 cache bank. As shown in Figure 6a, such requests travel through the underlying NoC and reach their destination (Ⓐ). After receiving a request, the corresponding L2 cache bank controller (L2 CTLR) replies with an acknowledgement (ACK-PUTO/ACK-PUTM) to receive the evicted, dirty block for write-back (Ⓑ). As soon as L1 CTLR receives an acknowledgement, the evicted block is sent towards the L2 cache bank as a DATA-PUTO/DATA-PUTM message(Ⓒ). All the data and control messages enter the local NoC router as packets, gets stored in the available VCs and take part in routing and arbitration decisions to reach their destination. For our optimisations, all the evicted L1 cache blocks (both clean and dirty) are marked with a 1-bit flag (*Evicted*) in their packet header for identification, as shown in Figure 5.



(a) Existing flow



(b) Proposed flow

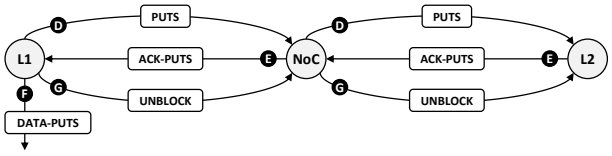
Figure 6: Eviction of a dirty L1 cache block

Our first optimisation targets DATA-PUTO/DATA-PUTM write-back data messages on their way to the destination. When any new packet enters the local router and gets buffered in the VC for routing and arbitration, *Evicted* flag is checked by the additional **Local Store/Reply, Forward & Drop (LSR-FD)** unit as shown in Figure 4. If the *Evicted* flag is SET, we know that the corresponding packet is actually an evicted, dirty cache block (DATA-PUTO/DATA-PUTM), which is on its way to the L2 cache bank for write-back. Even though the *Evicted* flag is set for both clean and dirty blocks, the identified block in the router can not be clean as they are dropped after eviction. We consider two-stage NoC routers (stage-1: RC, stage-2: VA and SA) where LSR-FD unit works in stage-1 in parallel with the Route Compute (RC) unit. While a check is performed by the LSR-FD unit to identify an evicted block (packet), route for the packet is also computed in parallel by the RC unit. If the *Evicted* flag is found SET for a packet (DATA-PUTO/DATA-PUTM) in stage-1, LSR-FD unit disables stage-2, i.e. VC and switch arbitration for the packet. Without arbitration, such packets can not leave the local router, as shown in Figure 6b. This way, we keep all the evicted, dirty L1 cache blocks stored in the input port VCs of local router for as long as possible (explained in Section 4.4). Since VCs are under-utilised due to low packet injection rate, keeping the evicted, dirty L1 cache blocks stored in local routers do not usually create any injection suppression. Both L1 and L2 caches are unaware of the proposed optimisation. For L1 CTLR, the evicted, dirty block is on its way or already reached the corresponding L2 cache bank for write-back. On the other side, since L2 CTLR already sent an acknowledgement to receive the block, it believes that the block is on its way.

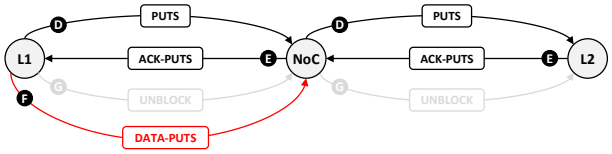
4.2 Block Store in Trace Buffers

If an evicted L1 cache block is clean (shared), it is simply discarded since a write-back is not necessary. As shown in Figure 7a, L1 CTLR initiates a PUTS write-back request towards the corresponding L2 cache bank (Ⓓ). Upon receiving the request, L2 CTLR sends an acknowledgement (ACK-PUTS) to the L1 cache (Ⓔ). The acknowledgement from L2 CTLR serves as the permission to discard the evicted, clean block (DATA-PUTS) in the L1 cache. Accordingly, L1 CTLR drops the evicted block (Ⓕ) and intimate the L2 cache bank with an UNBLOCK message (Ⓖ). After receiving the UNBLOCK message, L2 CTLR removes the L1 cache entry from the corresponding sharer list of that block.

Our second optimisation targets clean, L1 cache blocks that are discarded after eviction (DATA-PUTS). Since clean blocks are more frequently evicted, we propose to keep them



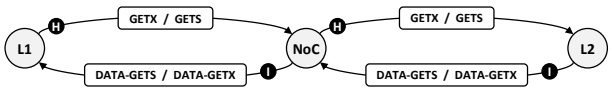
(a) Existing flow



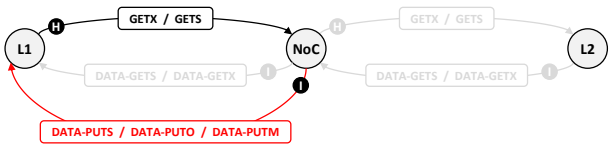
(b) Proposed flow

Figure 7: Eviction of a clean L1 cache block

stored to increase our chances of local reply (explained in Section 4.3). As shown in Figure 7b, instead of dropping DATA-PUTS, we redirect the message towards NoC (F). We also prohibit the transfer of UNBLOCK message towards the L2 cache bank. Since an acknowledgement is already sent, L2 CTLR believes that the corresponding block is discarded, and the UNBLOCK message is on the way. Now, the challenge is to accommodate the evicted, clean L1 cache blocks in local routers, which are normally discarded. Though we advocate that modern NoC based TCMPs use input buffered routers and buffers (VCs) are underutilised, but VCs are limited. With the first optimisation in Section 4.1, underutilised VCs are already employed to store evicted, dirty L1 cache blocks when they enter the local router to travel for write-back. Making the clean blocks compete with dirty blocks for the limited VCs available in local input port defeats the propose of accommodating more blocks. Thus, we consider storing the evicted, clean L1 cache blocks in the unused embedded trace buffer (ETB) of local routers. To facilitate the optimisation, all the evicted, clean L1 cache blocks are marked with a 1-bit flag (*Clean*) in their packet header as shown in Figure 5. When a new packet enters the local router, a 1:2 DEMUX (D) checks the *Clean* flag and if found SET, routes the packet towards the ETB (refer Figure 4). These packets are actually evicted, clean L1 cache blocks sent to NoC by our optimisation to be locally stored. We have re-purposed the ETB with the help of LSR-FD unit to accommodate such incoming packets. The detailed working of LSR-FD unit is presented in Algorithm 1. All the evicted, clean L1 cache blocks are now stored in the local routers to facilitate local reply when a possibility appears.



(a) Existing flow



(b) Proposed flow

Figure 8: L1 cache miss on a requested block

Algorithm 1: Working of LSR-FD Unit

Input: Status of embedded trace buffer and local virtual channels, modified packet header
Output: Local store or reply, block forward and drop
Notations:
m: Number of trace buffer entries (TB)
n: Number of virtual channels (VC)
τ_i: Time threshold of $VC_i \mid 0 \leq i < n$
P^{new}_{local}: Packet entered through local input port
$P^{stored}_{TB_i}$: Packet stored in $TB_i \mid 0 \leq i < m$
$P^{j}_{VC_i}$: Packet in $VC_i \mid 0 \leq i < n, j \in \{new, stored\}$

if $P^{new}_{local}[Clean] == SET$ **then**
  /* Local store of DATA-PUTS [4.2] */
  Enqueue TB_i to store P^{new}_{local}
  Increment i for next store
else
  if $P^{new}_{VC_i}[Evicted] == SET$ **then**
    /* Local store of DATA-PUT(O/M) [4.1] */
    $\tau_i = 64 \vee 128 \vee \ \ldots \ \vee 1024$
    Disable VA and SA for $P^{new}_{VC_i}$
  else if $P^{new}_{VC_i}[Miss] == SET$ **then**
    /* Local reply of GET(S/X) [4.3] */
    for $\forall TB_j \mid TB_j \neq NULL$ **do**
      if $P^{stored}_{TB_j}[Addr] == P^{new}_{VC_i}[Addr]$ **then**
        Dequeue TB_j to send $P^{stored}_{TB_j}$
        Deallocate VC_i to drop $P^{new}_{VC_i}$
    for $\forall VC_k \mid \tau_k > 0$ **do**
      if $P^{stored}_{VC_k}[Addr] == P^{new}_{VC_i}[Addr]$ **then**
        $\tau_k = 0$
        $P^{stored}_{VC_k}[Src] = P^{new}_{VC_i}[Dest]$
        $P^{stored}_{VC_k}[Dest] = P^{new}_{VC_i}[Src]$
        Enable VA and SA for $P^{stored}_{VC_k}$
        Deallocate VC_i to drop $P^{new}_{VC_i}$

/* *Defensive Vacate* of DATA-PUT(O/M) [4.4] */
for $\forall VC_i \mid VC_i \neq NULL$ **do**
  if $\exists VC_i \mid P^{stored}_{VC_i}[Evicted] == SET$ **then**
    $\tau_i = 0$
    $P^{stored}_{VC_i}[Evicted] = RESET$
    Enable VA and SA for $P^{stored}_{VC_i}$
/* TT-BF of DATA-PUT(O/M) [4.4.1] */
for $\forall VC_i \mid \tau_i > 0$ **do**
  $\tau_i = \tau_i - 1$
  if $\tau_i == 0$ **then**
    $P^{stored}_{VC_i}[Evicted] = RESET$
    Enable VA and SA for $P^{stored}_{VC_i}$
/* MT-BF of DATA-PUT(O/M) [4.4.2] */
if $P^{new}_{VC_i}[Forward] == SET$ **then**
  for $\forall VC_j \mid \tau_j > 0$ **do**
    if $P^{stored}_{VC_j}[Addr] == P^{new}_{VC_i}[Addr]$ **then**
      $\tau_j = 0$
      $P^{stored}_{VC_j}[Evicted] = RESET$
      Enable VA and SA for $P^{stored}_{VC_j}$
      Deallocate VC_i to drop $P^{new}_{VC_i}$

4.3 Block Reply from Routers

On a cache miss, the L1 CTLR issues a GETS/GETX request to the corresponding L2 cache bank (H), as shown in Figure 8a. Based on the received request, L2 CTLR replies with the block either in shared (DATA-GETS) or exclusive (DATA-GETX) state (I). Since the L2 cache is distributed, based on the location of the corresponding L2 cache bank and the underlying NoC congestion, reply takes an indefinite time to reach L1 cache. In the worst case of an L2 cache

miss, the reply message can take even longer time.

Our next optimisation identifies GETS/GETX messages and attempts local reply with the stored DATA-PUTS/DATA-PUTO/DATA-PUTM messages from NoC routers (❶), as shown in Figure 8b. All the data request messages (GETS and GETX) are marked with a 1-bit flag (*Miss*) in their packet header, as shown in Figure 5. When a new packet enters the local router with its *Miss* flag SET, LSR-FD unit attempts to generate a local reply if possible. These packets are actually GETS/GETX request messages carrying the address of a requested cache block. LSR-FD unit compares the requested address with the addresses of the stored cache blocks in trace buffer. One of the entries may have the requested block since the stored blocks are evicted from the same L1 cache in the recent past. If a match is found, we can generate a local reply to the cache miss request with a stored DATA-PUTS message (❶), as shown in Figure 8b. The matched block (packet) is forwarded from the trace buffer to the local output port (refer Figure 4). A 2:1 MUX (M) checks the *Clean* flag and if found SET, knows that the packet has come from the trace buffer. Such packets are given priority to take the local output port for destination.

If the requested address is not found in the trace buffer, LSR-FD unit compares it with the addresses of all the stored blocks in the non-empty VCs. A match is possible since the stored blocks in local input port VCs are recently evicted, dirty L1 cache blocks. If a match is found, we can generate a local reply to the cache miss request with a stored DATA-PUTO/DATA-PUTM message (❶), as shown in Figure 8b. LSR-FD unit swaps the source and destination of the matched block (packet) with the request packet (GETS/GETX) and drop the GETS/GETX packet as given in Algorithm 1. The new destination of the matched block (packet) is the same L1 cache from where it was evicted. The stored packet is now enabled for VC and switch arbitration, which were disabled earlier to keep it stored in the local router. Since the destination (L1 cache) is connected to the very same local router, such packets get ejected through the local output port. Avoiding on-chip travel (also off-chip travel in case of an L2 cache miss) to fetch a requested block (DATA-GETS/DATA-GETX) significantly reduces L1 cache miss penalty. In realisation, the proposed optimisations satisfy a GETS/GETX request with a matching DATA-PUTS/DATA-PUTO/DATA-PUTM message stored in the local router (in VCs or in embedded trace buffer (ETB)).

4.4 Block Forward and Drop from Routers

As we store evicted L1 cache blocks in the underutilised VCs and trace buffer of the local NoC routers, we face two key challenges. First, at a time during NoC congestion (when the packet injection rate is high), keeping the VCs occupied with stored blocks may create VC unavailability for incoming packets. Second, an evicted block that is now stored in the local router may be requested by others in the L2 cache bank resulting in the delay of their execution. Since our work is all about opportunistic caching, we take all the necessary steps to make sure that the proposed local store and reply does not hamper the usual NoC communication.

If a new packet can not be injected into the local router due to VC unavailability, we employ a *Defensive Vacate* approach to identify one of the VCs to be vacated where an evicted block is stored. When all the VCs are full, *Defensive Vacate* dictates that if any one of the VCs contains a stored block, that VC needs to be vacated. When multiple VCs have stored blocks, the oldest of them is vacated. *Defensive Vacate* is given in lines 29-34 of Algorithm 1. The identified VC contains an evicted, dirty L1 cache block since clean blocks are stored in the trace buffer. To vacate the identified VC, we must forward the stored, dirty block towards its destination for write-back. LSR-FD unit simply enables the VC and switch arbitration for the stored block, which were disabled when we kept it stored in the VC. This action ensures that the corresponding VC will be free in subsequent cycles and hence make room for new injection. NoC congestion can create scenarios like hotspots and Head-of-Line (HoL) blocking. In such cases, vacating all the VCs instead of just one, having stored blocks may revive the network. However, run-time detection of scenarios like HoL blocking is difficult [17]. Nevertheless, *Defensive Vacate* can be modified accordingly when such a direction is explored.

Even in the absence of injection pressure, the status of a stored block (both clean and dirty) may be expected in the L2 cache bank by other requesters to continue their execution. Since L2 cache bank controller is expecting a reply (UNBLOCK/DATA-PUTO/DATA-PUTM), it makes all the requester wait for the status. Trace buffer accommodates evicted, clean L1 blocks in a small circular queue (refer Figure 4) and hence such blocks do not stay stored for very long. When the trace buffer is full, the oldest clean block is replaced by an incoming block. When the oldest clean block is replaced (dropped), an UNBLOCK message is sent to the corresponding L2 cache bank for necessary action (explained in Section 4.5). To make sure that the wait for evicted, dirty blocks in the corresponding L2 cache bank is not too long, we propose the following two techniques.

4.4.1 Time-Triggered Block Forward (TT-BF)

An evicted, dirty L1 cache block is stored in the local router until a certain time threshold which is decided based on the re-reference time of evicted blocks (refer Figure 3). We add a threshold counter (τ_i) correspond to each VC of the local input port, as shown in Figure 4. When a counter reaches the threshold, the stored block (packet) in the corresponding VC is enabled for VC and switch arbitration. This action triggers forwarding of the stored, dirty block towards its destination for write-back. Till the time a block is locally stored, access requests for the block by others is delayed by a time equal to the threshold in the corresponding L2 cache bank.

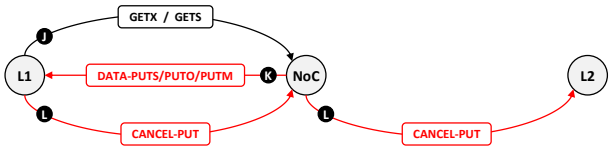
4.4.2 Message-Triggered Block Forward (MT-BF)

An evicted, dirty L1 cache block is stored in the local router until it is requested by someone in the corresponding L2 cache bank. In such a case, we make L2 CTLR resend an acknowledgement (ACK-PUTO/ACK-PUTM) for the requested block. Such acknowledgements are marked with a 1-bit flag (*Forward*) in their packet header, as shown in Figure 5. An acknowledgement was already sent, and the L2 CTLR is now waiting for the block for write-back. We send the second acknowledgement to inform that the block is requested by someone. When the second acknowledgement arrives at the destination router (local router of the stored

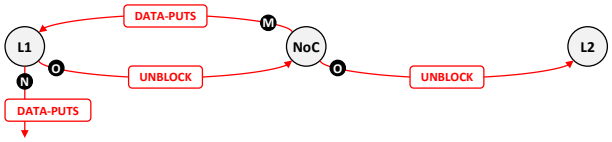
block), the corresponding stored block is enabled for VC and switch arbitration. LSR-FD unit takes the decision when it finds the *Forward* flag SET for an incoming packet, as given in lines 41-48 of Algorithm 1. Then, the second acknowledgement reaches L1 CTLR, where it is simply ignored.

4.5 Cache Coherence

As the shared L2 cache is involved in the proposed optimisations, we have to make sure that cache coherency is maintained throughout. After sending an acknowledgement (ACK-PUTO/ACK-PUTM), the L2 CTLR waits for the corresponding DATA-PUTO/DATA-PUTM message to initiate write-back. Since the evicted, dirty L1 cache block is coming for write-back, it must be the only copy in the entire system. Therefore, as long as evicted, dirty L1 cache blocks are stored in local routers, no special action is needed to maintain coherence in the system. A write-back is not necessary for evicted, clean L1 cache blocks. Hence, after sending an acknowledgement (ACK-PUTS), the L2 CTLR waits for an UNBLOCK message to remove the corresponding entry from the sharer list. While the L2 CTLR waits for the incoming UNBLOCK message, new requests for the corresponding clean block may be serviced. A new request for shared access (GETS) to the block is granted, while a request for exclusive access (GETX) is put on wait. Though multiple copies of the block may exist in the system, all of them are clean, and hence the cache block is coherent. Thus, no special action is needed for coherence when an evicted, clean L1 cache block is kept stored in the local router.



(a) During a local reply



(b) During a block drop

Figure 9: Messages to maintain cache coherence

When we attempt a local reply of GETS/GETX message with the stored DATA-PUTS/DATA-PUTO/DATA-PUTM message, we need to maintain coherence. As shown in Figure 9a, when a local reply reaches L1 CTLR (K), a special coherence message CANCEL-PUT is initiated towards the L2 cache bank (L). With that CANCEL-PUT message, L2 CTLR learns that the corresponding UNBLOCK/DATA-PUTO/DATA-PUTM message will not come. Hence, L2 CTLR rolls back the state of the corresponding block as if the eviction never happened. This way, we preserve the state of an evicted L1 cache block to maintain coherence. However, there can be a scenario where a write request (GETX) that requires a block with exclusive access is locally replied by a stored block that has shared access (DATA-PUTS). In such a scenario, we make sure that the L1 CTLR take permission from the L2 CTLR before granting the write request.

Table 1: Simulation configuration

Processor	64 OoO x86 cores
L1 cache	16KB, 4-way, 64B blocks, private, split
L2 cache	128KB×64 cores, 8-way, 64B blocks, shared
MC/Directory	4; one located at each corner
Cache Coherence	MOESI distributed directory
NoC	8×8 2D mesh, 128-bit flit channel 3 Virtual Networks (VNs), VN0, VN1, VN2 2/4/6 Virtual Channels (VCs)/VN 1-flit depth control VC, 5-flit depth data VC
Routing	2-stage routers, X-Y dimension-order routing
Packets	1-flit control packets, 5-flit data packets
Trace Buffer (ETB)	2KB/router
Benchmarks	SPEC CPU2006 (multi-programmed)
	PARSEC 3.0 (multi-threaded)

When a VC is to be vacated, the corresponding dirty block stored in that VC is forwarded for write-back, hence no coherence violation. When an entry in the trace buffer needs to be deleted, the stored clean block is dropped; but the corresponding L2 cache bank needs to be intimated to maintain coherence. Hence, to drop a clean block stored in the trace buffer, we send the block (DATA-PUTS) back to the same L1 cache from where it was evicted (M), as shown in Figure 9b. After receiving the DATA-PUTS message, L1 CTLR drops the block (N) and generates an UNBLOCK message for the L2 cache bank (O). With the arrival of the UNBLOCK message, L2 CTLR removes the sharer and completes the process of clean (shared) block eviction.

5 Performance Analysis

We consider the following architectures for evaluation:

- **Baseline:** Without any optimisation.
- **DB-TTBF:** Store evicted, dirty L1 cache blocks in local router VCs and use TTBF.
- **DB-MTBF:** Store evicted, dirty L1 cache blocks in local router VCs and use MTBF.
- **CDB-TTBF:** Store evicted, clean as well as dirty L1 cache blocks in local router VCs. Use TTBF for dirty blocks and drop clean blocks.
- **CDB-MTBF:** Store evicted, clean as well as dirty L1 cache blocks in local router VCs. Use MTBF for dirty blocks and drop clean blocks.
- **CDB-ETB-TTBF:** Store evicted, clean L1 cache blocks in ETB and dirty blocks in local router VCs. Use TTBF for dirty blocks and drop clean blocks.
- **CDB-ETB-MTBF:** Store evicted, clean L1 cache blocks in ETB and dirty blocks in local router VCs. Use MTBF for dirty blocks and drop clean blocks.

5.1 Simulation Framework and Workloads

The baseline and proposed architectures are modelled on event-driven gem5 simulator [18]. Our system configuration is similar to Intel Xeon Phi Processor 7235 [19] with shared and distributed L2 cache (LLC). Due to certain limitations in gem5, we could not model the exact cache configuration

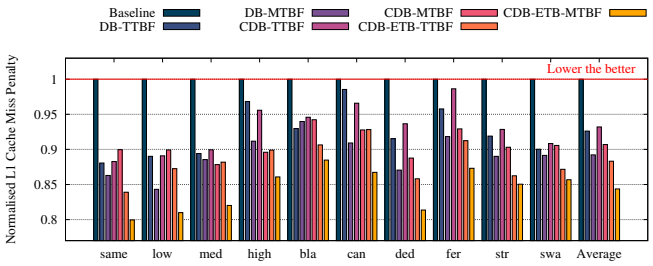


Figure 10: L1 cache miss penalty

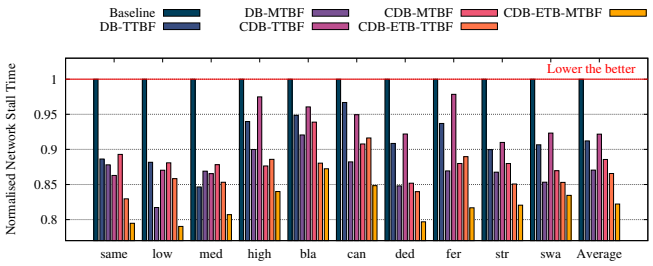


Figure 11: Network stall time

Table 2: Workload mixes

Mix	Benchmarks	Copies
same	astar, cactusADM, GemsFDTD, gromacs, h264ref, hmmer, lbm, leslie3d, namd, omnetpp, perlbench, sjeng, soplex, sphinx, xalancbmk	1×64: 64
low	gromacs, GemsFDTD, hmmer, astar, h264ref	4×16: 64
med	sphinx, perlbench, cactusADM, omnetpp, soplex	
high	xalancbmk, namd, sjeng, leslie3d, lbm	
bla	blackscholes; runs with 64 threads	1×64: 64
can	canneal; runs with 64 threads	
ded	dedup; runs with 64 threads	
fer	ferret; runs with 64 threads	
str	streamcluster; runs with 64 threads	
swa	swaptions; runs with 64 threads	

of Intel Xeon Phi Processor 7235. However, our cache configuration is not chosen to give undue advantage to the proposed optimisations. Rather, it challenges the optimisations with an L1 cache hit rate of around 90-95% for all the benchmarks we evaluate. Our system configuration is presented in Table 1 for reference. We modify GARNET [20] module in gem5 to implement the proposed router microarchitecture. We modify MOESI_CMP_directory protocol in Ruby inside gem5 to implement and maintain cache coherence.

To evaluate and analyse the performance, we consider multi-programmed as well as multi-threaded applications. For multi-programmed workloads, we consider SPEC CPU2006 benchmarks [14] to mimic a modern NoC based TCMP running multiple applications in parallel. We create different workload mixes based on the re-reference time of the benchmarks (refer Figure 3), as given in Table 2. *Same* is a homogeneous workload mix that runs 64 copies of the same benchmark on all the 64 cores (1×64: 64). *low*, *med* and *high* workload mixes are created by grouping benchmarks with low, medium and high re-reference times, respectively. These mixes run a random combination of 4 different benchmarks from their groups with 16 copies each (4×16: 64). By separately profiling each benchmark, we choose a smaller representative window of instructions to have a tractable simulation time. We create a total of 45 workload mixes (15 homogeneous, and 10 each for *low*, *med* and *high*) to extensively evaluate the proposed architectures.

For multi-threaded workloads, we consider PARSEC 3.0 benchmarks [15] to mimic a modern NoC based TCMP running multiple threads of a single application. We identify a mix of 6 computation-intensive, communication-intensive and memory-intensive benchmarks, as given in Table 2. *dedup* has huge working sets (computation-intensive) whereas the working set for *streamcluster* can be varied. *blackscholes* has negligible communication whereas *ferret* is very communication-intensive. *canneal* has the most demanding memory behaviour (memory-intensive) and so on. These benchmarks are run individually as a 64-thread workload on all the 64 cores of the TCMP (1 thread/core). We consider *sim-medium* input set of PARSEC 3.0 and evaluate the performance on region-of-interest. Altogether, we have 10 workloads to evaluate and analyse the performance, 4 multi-programmed benchmark mix and 6 multi-threaded benchmarks. For a relative comparison, all the results are normalised with respect to the baseline architecture.

5.2 Result Analysis and Discussion

L1 Cache Miss Penalty: It is defined as the number of cycles required to replace an existing cache block in L1 with an incoming block. L1 cache miss penalty directly reflects the effectiveness of the proposed local store and reply optimisation. Figure 10 shows the normalised L1 cache miss penalty with respect to the baseline architecture. With local replies, the proposed architectures reduce the L1 cache miss penalty for all the simulated workload mixes. In general, *CDB-ETB-TTBF* and *CDB-ETB-MTBF* architectures perform better compared to others. With more blocks (both clean and dirty) locally stored in more space (ETB and VCs), *CDB-ETB-TTBF* and *CDB-ETB-MTBF* has more scope for local reply (hits) on re-reference. *CDB-TTBF* and *CDB-MTBF* architectures also store both clean and dirty blocks in local NoC routers, but the storage space is limited to VCs. As a consequence, blocks are frequently moved in and out of the VCs, which reduces the chance of local hits. A maximum reduction of 21% and an average reduction of 16% in L1 cache miss penalty is achieved by our proposed architectures.

Among the multi-programmed workloads, *same* and *low* are outperforming *med* and *high* mixes as they have benchmarks with low re-reference time of evicted L1 cache blocks (refer Figure 3). Whereas the miss penalty reduction in multi-threaded workloads is relatively less when compared with the multi-programmed counterparts. It is due to the frequent sharing of data among the participating threads of a particular workload. Keeping evicted L1 cache blocks stored in local routers for long increases the miss penalty of other threads waiting in the corresponding L2 cache bank. In general, TTBF architectures (*DB-TTBF*, *CDB-TTBF* and *CDB-ETB-TTBF*) perform poorly as they keep evicted, dirty blocks stored for a certain time threshold (τ) even if there are other requesters waiting in the L2 cache bank. On the other hand, MTBF architectures (*DB-MTBF*, *CDB-MTBF* and *CDB-ETB-MTBF*) can forward stored blocks as and when a

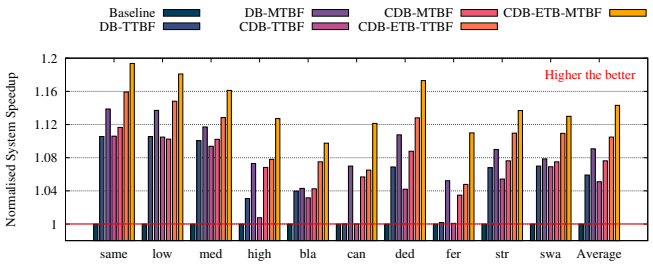


Figure 12: System speedup

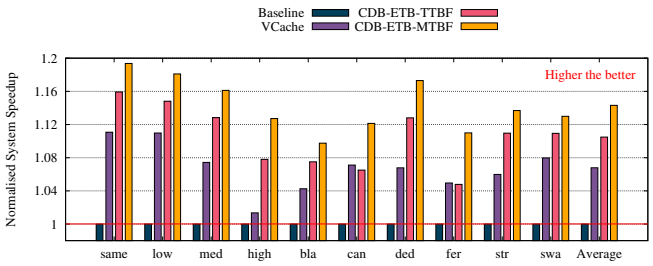


Figure 13: Comparison with *VCache* [21] architecture

request is received from the L2 cache bank. For example, *can* and *fer* workloads suffer the most while running in TTBF architectures as they have the most demanding memory and communication behaviour, respectively.

Network Stall Time: It is defined as the number of cycles the processor stalls waiting for a network packet. Network stall time helps us to understand how storing evicted L1 cache blocks in local routers impact the NoC communication latency. We prefer network stall time over *Packet Latency/Network Latency* as the former is a more appropriate metric to evaluate network-related slowdown in NoC based TCMPs [22]. Figure 11 shows the normalised network stall time with respect to the baseline architecture. As expected, across all the simulated workloads, our proposed architectures significantly reduce network stall time. With local reply from the routers, we avoid both on-chip travel and NoC communication delay as given in equation (1). Saving on-chip travel time indirectly translates into reduced network stall time. A maximum of 21% and an average of 18% reduction in network stall time is achieved by our proposed architectures. In TTBF architectures, a stored dirty block is forwarded for write-back only after the time threshold (τ) expires. However, with MTBF architectures, a stored dirty block is forwarded either after the time threshold (τ) expires or even earlier if a request from the corresponding L2 cache bank is received. As a result, all the MTBF architectures experience less network stall time in general.

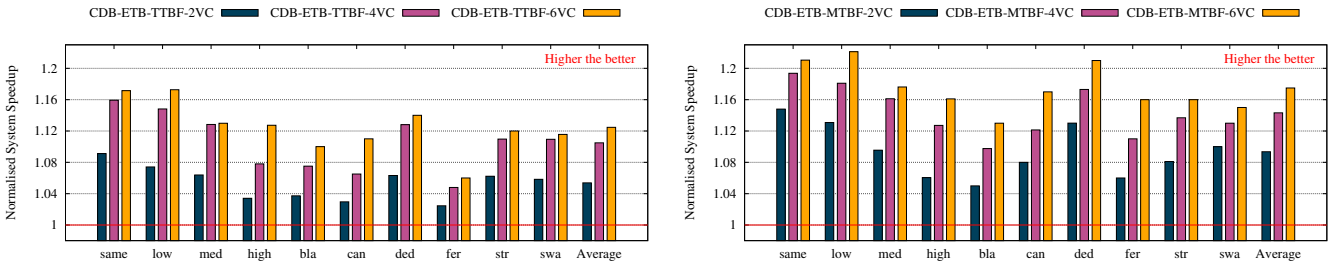
Usually, network stall time reduction is relatively less for communication-intensive workloads when compared to others. Since these workloads frequently inject packets in the network, evicted blocks can not be stored in the routers for long. As a consequence, their chances of local reply reduce. Additionally, frequent store and forward of evicted blocks increases the network latency for others. For example, the network stall time reduction in *high* is minimum when compared with other multi-programmed workloads. This is because *high* workload contains *leslie3d* and *lbm* benchmarks which have very high packet injection rate. Similarly, being the most communication-intensive multi-threaded workload, *fer* experience less reduction in network stall time. Interestingly, *bla* experiences the lowest reduction in network stall time even though it has a negligible communication pattern. This is due to the fact that *bla* is not able to get the benefit of local store and reply. Even though the average re-reference time of *bla* is one of the lowest (refer Figure 3), the number of re-references are low. So, evicted blocks just stay in the local router for some time and then get forwarded or dropped. To mitigate the negative effect of occupying VCs, Dynamically Allocated Multiple Queue (DAMQ) buffering schemes can be explored [23].

System Speedup: We use Instructions Per Cycle (IPC) to compare system speedup between baseline and the proposed architectures for multi-programmed workloads (SPEC CPU2006). Whereas, for multi-threaded workloads (PARSEC 3.0), we use execution time to compare system speedup. We prefer execution time for multi-threaded workloads as they have synchronisation primitives like locks and barriers, which brings variation in IPC. Figure 12 shows the normalised system speedup with respect to the baseline architecture. From the improvements in L1 cache miss penalty and network stall time, the increase in system speedup with the proposed architectures is intuitive. We achieve a maximum system speedup of 19% and an average system speedup of 14% for the presented workloads. Usage of trace buffers in *CDB-ETB-TTBF* and *CDB-ETB-MTBF* architectures significantly improves overall system performance with frequent local replies from the NoC routers.

5.3 Qualitative Comparison with An Existing Work

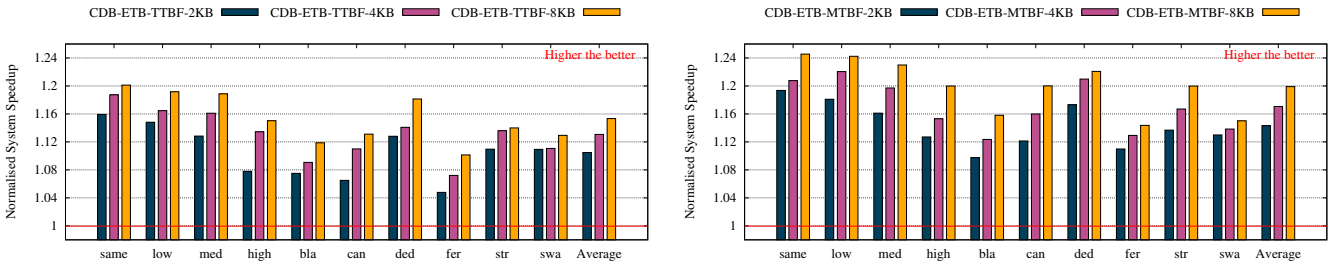
Jindal et al. [21][24] proposed to reuse trace buffer embedded in the processor as a victim cache [25] to improve system performance. The key idea is to re-purpose the trace buffer (ETB) as a set-associative cache called *VCache* to hold recently evicted blocks of L1 data cache. VCache indirectly increases the size of L1 data cache as they are mutually exclusive. A block requested by the processor is simultaneously searched in both the L1 data cache and VCache. If the requested block is not found in L1 data cache but the VCache, it is brought into the L1 data cache by swapping out another block into the VCache. The authors learn that in simultaneous multithreading (SMT), competing threads may flush cache blocks of each-other from the VCache resulting in poor performance. Thus, they propose two techniques to partition the VCache among the participating threads, which promotes cooperation. These two techniques attempt to increase VCache utilisation and improve performance. The concept of VCache and our proposed TTBF/MTBF architectures are complementary in nature and can be implemented together. However, there are a few important differences between VCache and the proposed TTBF/MTBF:

- VCache does not differentiate between clean and dirty cache blocks and flushes them immediately with incoming blocks. Whereas, TTBF/MTBF segregates clean and dirty blocks in such a way that dirty blocks are cached in VCs until a certain time threshold. This optimisation delays/avoids expensive writes to the L2 cache bank. Hence, TTBF/MTBF



(a) (b)

Figure 14: Impact of number of VCs



(a) (b)

Figure 15: Impact of trace buffer size

indirectly uses VCs like a write buffer [26] and ETB like a victim cache to improve system performance.

- VCache does not talk about coherence. Cache blocks stored in VCache can be either in a shared or exclusive state. Frequent data sharing among different cores (threads) is more prevalent in multi-threaded applications. VCache is evaluated only for multi-programmed applications (SPEC CPU2006), and the discussion about shared memory and the associated coherence is not included. Whereas, TTBF/MTBF provides a detailed discussion about how coherence is maintained during local store, reply, forward and drop of evicted L1 cache blocks. TTBF/MTBF also adds a new coherence message to make sure that the states of evicted L1 cache blocks are preserved.
- VCache is set-associative where multiple evicted L1 data cache blocks will map into the same set. This may result in a frequent flush of stored blocks from VCache, which hampers performance. On the other hand, TTBF/MTBF uses VCs and ETB on NoC routers like a fully-associative cache. Hence, there are less conflicts and blocks can be retained longer, which improves chances of local reply from routers.
- VCache works as an extension of L1 data cache, and hence it only stores evicted data cache blocks. Whereas TTBF/MTBF stores all the evicted blocks, both of data and instruction L1 caches. Additionally, since TTBF/MTBF use VCs as well as ETB of NoC routers to store evicted L1 cache blocks, they can accommodate more evicted blocks simultaneously.

VCache is originally proposed for LEON3 Processor [27] which can be realised for up to 16 CPU cores. To make a fair comparison with TTBF/MTBF architectures, we faithfully model a 64-core equivalent VCache architecture. Figure 13 shows the overall system performance comparison of VCache and the proposed TTBF/MTBF architectures.

CDB-ETB-TTBF and *CDB-ETB-MTBF* performs better than VCache in almost all the simulated workloads. Simultaneously accommodating more evicted L1 cache blocks, fewer conflicts during block store, and longer retention of stored blocks are the key factors. However, VCache performs better than *CDB-ETB-TTBF* for *can* and *fer* workloads. With the most demanding memory and communication behaviour, *can* and *fer* suffers in *CDB-ETB-TTBF* that keeps evicted, dirty L1 cache blocks stored for a certain time threshold (τ) even if other requesters are waiting in the corresponding L2 cache bank. When compared to VCache, an average of 4% and 8% improvement in system speedup is seen for *CDB-ETB-TTBF* and *CDB-ETB-MTBF*, respectively.

6 Sensitivity and Overhead Analysis

6.1 Impact of Number of VCs

Our first optimisation requires evicted, dirty L1 cache blocks to be stored in VCs of the local input port. Hence, we explore the impact of the number of VCs/VN in the proposed architectures. For all the results discussed so far, we have considered 4 VCs/VN (as presented in Table 1). However, Figure 14a and 14b shows how varying the number of VCs/VN impact the overall system performance. We have given the results of only *CDB-ETB-TTBF* and *CDB-ETB-MTBF* architectures as they have the best performance in their respective groups (TTBF and MTBF groups). It is almost trivial that increasing the number of VCs/VN will improve system performance. However, an interesting observation is the performance of architectures with only 2 input port VCs/VN (*CDB-ETB-TTBF-2VC* and *CDB-ETB-MTBF-2VC*). The main reason for a performance gain despite having only 2 VCs/VN in the local input port is the presence of ETB. A good number of re-references are locally replied with the stored clean blocks from ETB that contributes to the improvement in overall system performance.

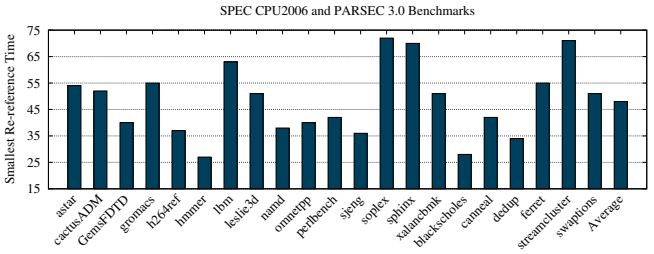


Figure 16: Smallest re-reference time

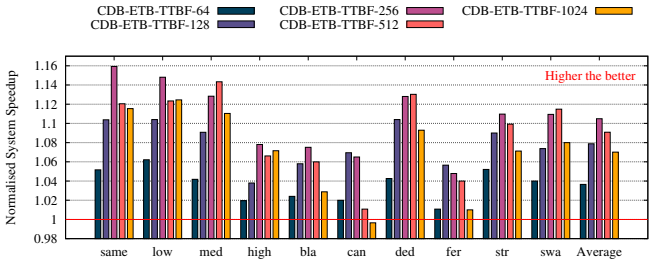


Figure 17: Impact of time threshold

6.2 Impact of Trace Buffer Size

Our second optimisation requires evicted, clean L1 cache blocks to be stored in ETB of NoC routers. Hence, we explore the impact of trace buffer size in the proposed architectures. For all the results discussed so far, we have considered trace buffer size as 2KB (usually the minimum size). However, Figure 15a and 15b shows how varying the size of trace buffer impact the overall system performance. With our optimisation, trace buffer in NoC routers can be viewed as a cache that holds recently evicted, clean L1 cache blocks. As we increase the size of the trace buffer, we get appropriate improvement in the system performance.

6.3 Impact of Time Threshold

To make sure that the evicted, dirty L1 cache blocks are not stored for too long and penalise others, we proposed TTBF that uses a time threshold (τ). For all the results discussed so far with TTBF architectures (*DB-TTBF*, *CDB-TTBF* and *CDB-ETB-TTBF*), we have considered τ as 256 cycles. Intuitively, the optimal value of τ should be 12150 cycles, equal to the average re-reference time of evicted L1 cache blocks (refer Figure 3). However, keeping a block stored for such a long duration delays execution of others who are expecting the block in the corresponding L2 cache bank. This scenario is more prevalent in multi-threaded workloads, where a lot of data sharing happens among the participating cores. Hence, we perform an empirical study to find the optimal value of τ. We use an incremental approach and begin from the smallest re-reference time of evicted L1 cache blocks. Figure 16 shows that the smallest re-reference time of all the benchmarks we have considered is under 80 with an average of 48 cycles. So, we begin with the value of τ as 64 cycles and incrementally change it to 128, 256, 512 cycles and more. Figure 17 shows how varying the value of τ impact overall system performance. Based on the observation in Figure 17, we considered τ as 256 for our evaluation.

However, for all the results with MTBF architectures (*DB-MTBF*, *CDB-MTBF* and *CDB-ETB-MTBF*), we kept τ as 16384 cycles; smallest power of 2 which is large enough for the average re-reference time (12150 cycles). This is not optimal rather an intuitive time threshold to increase our chances of local reply. Now, an evicted, dirty L1 cache block is forwarded towards destination either after getting a second acknowledgement for write-back (refer Section 4.4.2) or after 16384 cycles, whichever is earlier. Thus, MTBF architectures optimise performance by triggering a block forward based on a message as well as a time threshold.

6.4 Storage, Area and Power Overhead

We use 4 additional bits (*Evicted*, *Clean*, *Miss* and *Forward*) in the packet header (refer Figure 5) to facilitate the working of LSR-FD unit. Our NoC uses 128-bit flit channel (refer Table 1) and a typical packet header (head flit) is much smaller ($\approx$ 64 bits). So, we can accommodate the additional 4 bits in the head flit without any storage overhead.

Since LSR-FD unit works in parallel to the RC unit (refer Section 4.1), it is not in the critical path of execution. In Algorithm 1, lines 7-28, 29-34, 35-40 and 41-48 can execute in parallel to complete the working of LSR-FD unit in time to avoid the critical path. However, the addition of LSR-FD unit in NoC routers contributes to the area and power overhead. As we have 4 VCs/VN and the local input port requires to have a time threshold counter (τ_i) for each VC, we add binary down counters[2]. Among the 3 VNs (refer Table 1), VN2 carries evicted cache blocks. Hence, we add only four 8-bit binary down counters (1 counter/VC for VN2) in TTBF architectures to count from 255 down to 0. Whereas, 14-bit binary down counters are added in MTBF architectures to count from 16383 down to 0. As a result, the addition of these counters also contributes to the area and power overhead. The MUX-DEMUX pair of M and D (refer Figure 4) and the connecting links also contribute to a negligible area and power overhead. Embedded trace buffer (ETB) was always present in NoC routers in power-gated mode. So, ETB does not contribute to the area but only to the power overhead. We use McPAT [28] at 22nm processor technology and feed the configuration and output files of gem5 [18] to get the area, leakage and dynamic power overheads. We present the percentage increase/decrease in overhead for *CDB-ETB-TTBF* and *CDB-ETB-MTBF* architectures compared to the baseline in Table 3. While we get negligible area and leakage power overhead due to the additional circuits, dynamic power is reduced due to the significant improvements in overall system performance.

During post-silicon debug and validation, ETB is typically used for functional test. ETB requires to monitor internal signals in real time for functional bugs. Hence, ETB operates at full system clock frequency, and there is no additional delay while using it in our optimisation. However, even though the usage of ETB post production is very rare, but it is possible. In such a scenario, during the time the ETB is used for debug and validation, *CDB-ETB-TTBF* and *CDB-ETB-MTBF* architectures will behave like *CDB-TTBF* and *CDB-MTBF* architectures, respectively.

2. An N-bit binary down counter counts from $2^N - 1$ to 0.

Table 3: Area and power overhead compared to the baseline

Overhead	CDB-ETB-TTBF	CDB-ETB-MTBF
Area	↑ 2.23%	↑ 2.58%
Leakage Power	↑ 3.71%	↑ 3.94%
Dynamic Power	↓ 5.06%	↓ 6.12%

7 Related Works

Existing literature explored different possibilities for efficient utilisation of NoC resources. Since our work is about using NoC as a storage, our discussion is limited to the related works where NoC is projected in that light. One of the first works that attempted to change the abstraction of NoC from communication to storage is by Mizrahi et al. [29]. They advocated that NoC can be included in the memory hierarchy by placing a small cache in the routers. Going forward in the same line, Eisley et al. [30] decoupled cache from coherence and proposed to keep only the coherence directories in NoC. Using the stored directories, they designed a coherence protocol within the NoC routers. Yanamandra et al. [31] combined the advantages of both and proposed to keep frequently used cache blocks along with the coherence directories in NoC routers. There are other significant works that are focused towards NoC aware cache and coherence implementations [32][33][34]. However, almost all the proposed works employed additional storage in NoC routers.

A new direction has gained popularity where the unused DfD hardware that were added for post-silicon debug and validation are viewed as a storage. For example, Jindal et al. [21][24] have re-purposed DfD hardware for improving the cache performance by using them as a victim cache. DfD hardware are also used to store critical information for runtime verification and system security [35][36]. Specifically, in the context of NoC, embedded trace buffers (ETBs) in routers are used as extended VCs to improve communication [37][38]. However, extending VCs might not be beneficial in input buffered NoC routers, where existing VCs are already underutilised [3][4][5] (refer Section 2.2).

Sanchez et al. [6] provided a key insight that NoC is responsible for 60-75% of the miss latency in TCMPs. They argued that as NoC based TCMPs continue to scale, considering NoC and memory hierarchy together is the way forward. In a promising new attempt, Das et al. [39][40] recently proposed to exploit underutilised VCs of local NoC routers to store some evicted cache blocks. They attempted to reply future references to such blocks from the local routers and reduce miss penalty. A similar work on NoC based consumer electronics is also available [41]. However, none of these works considered all the evicted L1 cache blocks. In this work, we extend [39] to store evicted, dirty L1 cache blocks in VCs and clean L1 cache blocks in ETB of local NoC routers. Future references to recently evicted L1 cache blocks are replied either from the VCs or the ETB.

8 Conclusion

In this work, we explored opportunities to store recently evicted L1 cache blocks in NoC to reduce cache miss penalty. We identified underutilised input port buffers and unused embedded trace buffers as potential storage space in NoC routers. We proposed multiple architectures to store recently evicted cache blocks in NoC routers and facilitate direct reply when such blocks are re-referenced. We also proposed two techniques to forward stored, dirty cache blocks for write-back and a technique to drop stored, clean cache blocks. To preserve the state of evicted cache blocks and maintain coherence, we also propose an additional coherence message. We experimentally validated that the proposed architectures have the potential to reduce cache miss penalty and improve overall system performance. Since the proposed optimisations are on NoC, they can be easily integrated into any existing optimisation in the memory.

References

[1] (2015) International Technology Roadmap for Semiconductors (ITRS). [Online]. Available: https://www.semiconductors.org/resources/2015-international-technology-roadmap-for-semiconductors-itrs/

[2] (2017) Intel Xeon Phi 72x5 Processor Family. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/series/132784/intel-xeon-phi-72x5-processor-family.html

[3] N. Barrow-Williams, C. Fensch, and S. Moore, "A communication characterisation of splash-2 and parsec," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2009.

[4] P. Gratz and S. W. Keckler, "Realistic workload characterization and analysis for networks-on-chip design," in *The 4th workshop on chip multiprocessor memory systems and interconnects (CMP-MSI)*, 2010, pp. 1–10.

[5] R. Hesse, J. Nicholls, and N. E. Jerger, "Fine-grained bandwidth adaptivity in networks-on-chip using bidirectional channels," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*. IEEE, 2012, pp. 132–141.

[6] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis, "An analysis of on-chip interconnection networks for large-scale chip multiprocessors," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 7, no. 1, pp. 1–28, 2010.

[7] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y.-C. Liu, "Knights landing: Second-generation intel xeon phi product," *Ieee micro*, vol. 36, no. 2, pp. 34–46, 2016.

[8] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang *et al.*, "Openpiton: An open source manycore research framework," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016, pp. 217–232.

[9] B. K. Daya, C.-H. O. Chen, S. Subramanian, W.-C. Kwon, S. Park, T. Krishna, J. Holt, A. P. Chandrakasan, and L.-S. Peh, "Scorpio: a 36-core research chip demonstrating snoopy coherence on a scalable mesh noc with in-network ordering," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 2014, pp. 25–36.

[10] B. K. Daya, L.-S. Peh, and A. P. Chandrakasan, "Quest for high-performance bufferless nocs with single-cycle express paths and self-learning throttling," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.

[11] G. Michelogiannakis, D. Sanchez, W. J. Dally, and C. Kozyrakis, "Evaluating bufferless flow control for on-chip networks," in *2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*. IEEE, 2010, pp. 9–16.

[12] B. Vermeulen, "Design-for-debug to address next-generation soc debug concerns," in *2007 IEEE International Test Conference*. IEEE, 2007, pp. 1–1.

[13] W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. 36, no. 5, pp. 547–553, 1987.

[14] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[15] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.

[16] R. Abdel-Khalek and V. Bertacco, "Post-silicon platform for the functional diagnosis and debug of networks-on-chip," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 3s, pp. 1–25, 2014.

[17] J. Duato, I. Johnson, J. Flich, F. Naven, P. Garcia, and T. Nachiondo, "A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks," in *11th International Symposium on High-Performance Computer Architecture*. IEEE, 2005, pp. 108–119.

[18] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[19] (2017) Intel Xeon Phi Processor 7235. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/128694/intel-xeon-phi-processor-7235-16gb-1-3-ghz-64-core.html

[20] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in *2009 IEEE international symposium on performance analysis of systems and software*. IEEE, 2009, pp. 33–42.

[21] N. Jindal, P. R. Panda, and S. R. Sarangi, "Reusing trace buffers as victim caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 9, pp. 1699–1712, 2018.

[22] O. Mutlu and T. Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 2007, pp. 146–160.

[23] J. Liu and J. G. Delgado-Frias, "Damq self-compacting buffer schemes for systems with network-on-chip." in *CDES*, 2005.

[24] N. Jindal, P. R. Panda, and S. R. Sarangi, "Reusing trace buffers to enhance cache performance," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017.

[25] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," *ACM SIGARCH Computer Architecture News*, vol. 18, no. 2SI, pp. 364–373, 1990.

[26] M. A. Mills Jr and L. M. Crudele, "Write buffer," Feb. 14 1989, uS Patent 4,805,098.

[27] (2017) LEON3 Processor. [Online]. Available: https://www.gaisler.com/index.php/products/processors/leon3

[28] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 469–480.

[29] H. E. Mizrahi, J.-L. Baer, E. D. Lazowska, and J. Zahorjan, "Introducing memory into the switch elements of multiprocessor interconnection networks," in *Proceedings of the 16th annual international symposium on Computer architecture*, 1989, pp. 158–166.

[30] N. Eisley, L.-S. Peh, and L. Shang, "In-network cache coherence," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE, 2006, pp. 321–332.

[31] A. Yanamandra, M. J. Irwin, V. Narayanan, M. Kandemir, and S. H. K. Narayanan, "In-network caching for chip multiprocessors," in *International Conference on High-Performance Embedded Architectures and Compilers*. Springer, 2009, pp. 373–388.

[32] C. Fensch, N. Barrow-Williams, R. D. Mullins, and S. Moore, "Designing a physical locality aware coherence protocol for chip-multiprocessors," *IEEE Transactions on Computers*, vol. 62, no. 5, pp. 914–928, 2012.

[33] L. Huang, "Leveraging on-chip networks for efficient prediction on multicore coherence," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–4.

[34] W. Shu and N.-F. Tzeng, "Nuda: Non-uniform directory architecture for scalable chip multiprocessors," *IEEE Transactions on Computers*, vol. 67, no. 5, pp. 740–747, 2017.

[35] A. Basak, S. Bhunia, and S. Ray, "Exploiting design-for-debug for flexible soc security architecture," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.

[36] N. Jindal, S. Chandran, P. R. Panda, S. Prasad, A. Mitra, K. Singhal, S. Gupta, and S. Tuli, "Dhoom: reusing design-for-debug hardware for online monitoring," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[37] S. S. Rout, M. Badri, and S. Deb, "Reutilization of trace buffers for performance enhancement of noc based mpsocs," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 97–102.

[38] N. Jindal, S. Gupta, D. P. Ravipati, P. R. Panda, and S. R. Sarangi, "Enhancing network-on-chip performance by reusing trace buffers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 922–935, 2019.

[39] A. Das, A. Kumar, J. Jose, and M. Palesi, "Exploiting on-chip routers to store dirty cache blocks in tiled chip multi-processors," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2020, pp. 147–152.

[40] A. Das, A. Kumar, and J. Jose, "Reducing off-chip miss penalty by exploiting underutilised on-chip router buffers," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 2020.

[41] A. Das, A. Kumar, J. Jose, and M. Palesi, "Revising noc in future multi-core based consumer electronics for performance," *IEEE Consumer Electronics Magazine*, 2021.

Abhijit Das is a PhD Scholar in the Department of Computer Science and Engineering at Indian Institute of Technology (IIT) Guwahati, India. His current research interests include performance and security aware on-chip networks and memory hierarchy in many-core systems. One of his recent works was a best paper candidate at ISVLSI 2020. He is a student member of ACM SIGARCH, ACM SIGMICRO, IEEE TCAA and IEEE TCuARCH communities.

Abhishek Kumar is a Member of Technical Staff at Oracle Inc., Bengaluru, India. He completed M.Tech. in Computer Science and Engineering from Indian Institute of Technology (IIT) Guwahati, India in 2020 where he secured the first position in the university. His M.Tech. thesis won the best thesis award in the university. He primarily works in the broad area of advance computer architecture. One of his recent works was a best paper candidate at ISVLSI 2020.

John Jose is an Assistant Professor in the Department of Computer Science and Engineering at Indian Institute of Technology (IIT) Guwahati, India. He completed PhD in Computer Science and Engineering from Indian Institute of Technology (IIT) Madras, India in 2014. His research interest is on computer architecture with a special focus to performance optimisation related to on-chip memory and communication aspects of multi/many-core processors. He is a member of ACM and IEEE.

Maurizio Palesi is an Associate Professor in Computer Engineering at University of Catania, Italy. His research activity is focused in the area of embedded systems with particular emphasis on single-chip implementations based on the network-on-chip design paradigm. He has served as Guest Editor of 20 special issues in top-level journals. He has served as General Chair and TPC Co-Chair in several international conferences and workshops. He serves as Associate Editor in 12 international journals. He has been recipient of the best paper award at the DATE 2011 and the HiPEAC paper award 2014. He is member of the HiPEAC and IEEE Senior Member.