Reducing Off-Chip Miss Penalty by Exploiting Underutilised On-Chip Router Buffers

Abhijit Das, Abhishek Kumar and John Jose
Dept. of Computer Science and Engineering, Indian Institute of Technology Guwahati, India
{abhijit.das, abhishek18a, johnjose}@iitg.ac.in

***Abstract*—The era of data driven applications expose limited on-chip caching in modern Tiled Chip Multi-Processors (TCMPs). Some applications suffer from frequent last level cache (LLC) miss and travel off-chip to fetch data and instructions. Off-chip miss penalty is very expensive as it severely hampers application execution time. Modern Network-on-Chip (NoC) based TCMPs employ input buffered routers for scalable communication bandwidth. In this work, we exploit underutilised buffers of NoC routers to store recently evicted LLC blocks. While these blocks are locally stored, future data requests for such blocks are directly replied from the NoC router. Local reply from routers avoid off-chip travel and significantly reduces LLC miss penalty. To make sure that such storage of evicted LLC blocks does not create NoC congestion, we incorporate block forwarding and dropping using dynamic router buffer contention updates. We experimentally validate that our proposed optimisations significantly reduces LLC miss penalty and improves overall system performance. We achieve a maximum system speedup of up to 13% and an average system speedup of 7%.**

***Index Terms*—Miss Penalty, Last Level Cache (LLC), Cache Coherence, Network-on-Chip (NoC), Virtual Channel (VC)**

I. Introduction

In the era of data driven applications, the demand for information processing is increasing exponentially. The increasing demand is driving a parallel increase in the number of processing cores in Tiled Chip Multi-Processors (TCMPs). It is indeed visible in the industry for example with Intel Xeon Phi Processors featuring 64-72 cores [1]. With ever increasing cores in modern TCMPs, Network-on-Chip (NoC) communication usually plays a very significant role in data access latency. Nevertheless, for data access requests that travel off-chip memory banks, the role of NoC is limited. Existing literature argues that NoC resource utilisation is very low, with an average injection rate of around 5% [2][3][4]. Exploiting underutilised NoC resources to improve data access latency for off-chip accesses is an interesting area to explore.

Due to the usage of relatively simpler Out-of-Order (OoO) processing cores, modern TCMPs including Intel Xeon Phi Processors [5], use only two levels of on-chip caching. In these processors, L2 serves as the last level cache (LLC) and communicates with the off-chip memory banks for data transfer. On-chip caches are usually small due to their associated cost and hence going to the off-chip memory banks for fetching data is inevitable. With data driven applications, going off-chip for data is even more common. Current NoC based TCMPs like Intel Xeon Phi Processors [5] have private, write-back L1 caches and a shared distributed, write-back

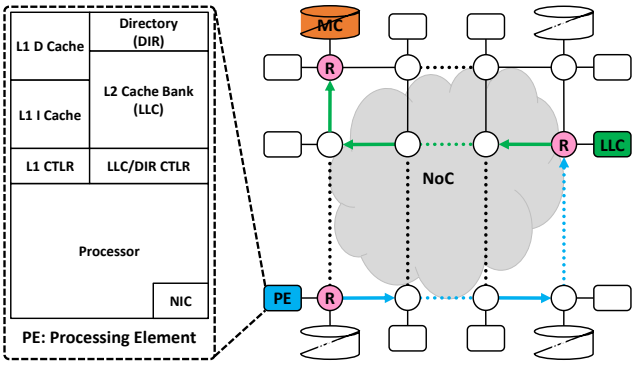
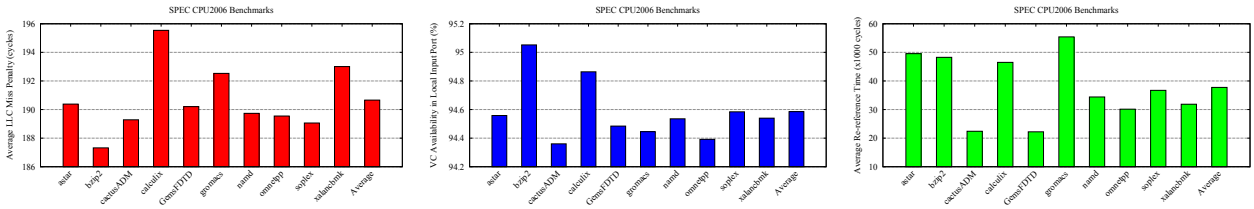


Figure 1: Conceptual view of an NoC based TCMP

LLC with directory. When an L1 cache miss occurs, the data request reaches the corresponding LLC bank. When an LLC miss occurs, the data request reaches the corresponding off-chip memory bank through memory controller (MC) as shown in Figure 1. The requested data is fetched from the off-chip memory and forwarded to the corresponding LLC bank and then to L1 cache to resume execution. The entire communication is packet based and is done through the underlying NoC. The time required to replace an existing cache block in LLC with an incoming cache block is called LLC miss penalty. It is also known as *off-chip miss penalty* as the incoming block is fetched from off-chip memory. Since LLC is small in size, it fills up quickly and all subsequent LLC miss always evicts a valid cache block. As per the coherence directory information, the LLC/Directory controller (LLC/DIR CTLR) decides whether to discard the block (clean) or send it to off-chip memory for write-back (dirty). However for applications with good temporal locality of reference, a recently evicted LLC block, if requested again needs to be re-fetched from the off-chip memory. The LLC miss penalty is generally in order of hundreds of clock cycles. This severely hampers application execution time and degrades overall system performance.

Modern NoC based TCMPs employ input buffered routers for scalable on-chip bandwidth [6][7]. In-transit packets on their way to destination are stored in the buffers of intermediate routers to take part in routing and arbitration decisions. However, an experimental analysis on real application-based workloads show that the average buffer utilisation of NoC routers is very low except during peak network congestion (Section II-B). In this work, we attempt to exploit empty buffers of NoC routers in a way that increases their utilisation

2576-6996/20/$31.00 ©2020 IEEE
DOI 10.1109/ICCD50377.2020.00049



(a) LLC miss penalty (b) VC availability in local input port (c) Re-reference time of evicted LLC blocks

Figure 2: Key observations and motivation for the proposed architecture

and positively impacts system performance. When evicted, dirty blocks of an LLC bank reach the local router to travel over the NoC to reach off-chip memory for write back; we propose to delay their travel. We temporarily disable arbitration of such evicted blocks to delay their travel and keep them stored in local router buffers. We also propose to store some of the evicted, clean blocks in local routers, which are otherwise discarded. Since the buffer utilisation is low, these blocks can be kept stored in local routers without inducing any NoC congestion. When a recently evicted LLC block is re-referenced, we propose to arrange a quick reply with the stored block from the local router. These optimisations help to avoid the off-chip miss penalty and improve overall system performance. We make the following major contributions:

1) **Local Store:** We propose an NoC architecture that identifies evicted, dirty LLC blocks in local router buffers and disables their arbitration. We also make some evicted, clean LLC blocks reach the local router and get stored in available buffers. Both clean and dirty LLC blocks are stored in local routers for as long as possible.
2) **Local Reply:** We propose an optimisation such that during the time an evicted LLC block is locally stored, a re-reference request for the same block can be locally replied from the router. With local replies, we avoid off-chip miss penalty and improve system performance.
3) **Block Forward and Drop:** We forward the locally stored dirty LLC blocks for write-back towards their destination using two approaches based on router buffer contention. We discard the stored clean LLC blocks as they need not be forwarded for write-back. This way, we make sure that locally storing evicted LLC blocks does not create injection suppression for others.

II. Background and Motivation

A. LLC Miss Penalty

Latest NoC based TCMPs like Intel Xeon Phi Processor (2016) [5], Princeton Piton Processor (2015) [8], MIT Scorpio Processor (2014) [9] and others use two levels of on-chip caching. Data driven, memory-intensive applications expose this limited on-chip caching and result in frequent LLC misses. The time required to replace an existing cache block in LLC with an incoming block is called LLC miss penalty. In NoC based TCMPs, LLC miss penalty can be given as:

$$Miss\ Penalty_{LLC} = T_{on-chip} + T_{off-chip} \quad (1)$$

where

$$T_{on-chip} = t^{Request}_{LLC/DIR\ CTLR-MC} + t^{Reply}_{MC-LLC/DIR\ CTLR} \quad (1a)$$

$$T_{off-chip} = t^{Request}_{MC-MEM} + t^{Reply}_{MEM-MC} \quad (1b)$$

$$T_{on-chip} << T_{off-chip} \quad (1c)$$

where t^i_j is the time taken by message i to travel distance j, for example $t^{Request}_{LLC/DIR\ CTLR-MC}$ is the time taken by an off-chip data request to travel from LLC/Directory controller (LLC/DIR CTLR) to the memory controller (MC) (Figure 1).

Figure 2a shows the average LLC miss penalties for different SPEC CPU2006 benchmarks. The average LLC miss penalty for the presented benchmarks is around 190 cycles which is very expensive. As given in equation (1c), LLC miss penalty is dominated by the off-chip data transfer time between MC and memory (MEM). Off-chip data transfer time is usually in hundreds of cycles, whereas on-chip data transfer time is only in tens of cycles. Hence, the role of NoC based on-chip data transfer in LLC miss penalty is minimum. In another observation from Figure 2a, the LLC miss penalty is very similar for all the presented benchmarks. It is attributed to the fact that after acquiring the shared data bus, off-chip transfer time is almost fixed. Since off-chip data transfer time dominates in LLC miss penalty, it remains similar across benchmarks. On-chip congestion and delay in NoC routers owing to application behaviour have negligible impact.

Observation 1:

LLC miss penalty is expensive and application oblivious with limited role of NoC and its resources.

B. VC Availability

NoC based systems use routers to communicate the transfer of packets from their source to destination. NoC routers have three design alternatives: buffered, minimally buffered and bufferless; each with different pros and cons. Modern TCMPs employ input buffered NoC routers for scalable on-chip bandwidth [6][7]. Packets coming through different input ports (north, east, south, west and local) gets stored in the available buffers of virtual channels (VCs) and take part in routing and arbitration decisions. VC availability in NoC based TCMPs can be represented as:

$$VC\ Availability_n = \frac{Cycles\ when\ n\ VCs\ are\ Free}{Total\ Execution\ Cycles} \quad (2)$$

Figure 2b shows the VC availability in local input port of NoC routers for different SPEC CPU2006 benchmarks. As the average injection rate of these multiprogrammed benchmarks is only around 5%, except during peak NoC congestion, at least one VC is always free ($\approx$ 95%). The observation in Figure 2b is in sync with the conclusions in the literature about low buffer utilisation with real applications [2][3][4].

Observation 2:

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

C. Motivation

The concept of caching is governed by the principle of locality of reference; temporal and spatial. Applications with good temporal locality of reference may request for recently evicted cache blocks in LLC. The duration from the eviction of an LLC block to the request of the same block in future is called re-reference time. Figure 2c shows the average re-reference time for different SPEC CPU2006 benchmarks. For example, in a 64-core NoC based TCMP running a memory-intensive benchmark *GemsFDTD*, an evicted LLC block is re-referenced within an average time of 22213 cycles. Across all benchmarks, on average, within an interval of around 37000 cycles, an evicted LLC block is re-referenced. For all the observations presented in Figure 2a, 2b and 2c, we run 64 copies of the same benchmark in a 64-core NoC based TCMP.

Evicted, clean LLC blocks are discarded whereas dirty LLC blocks are sent over the NoC to the off-chip memory bank for write-back. To reach their destination for write-back, evicted, dirty LLC blocks enter the local router through the local input port as packets. We call them LLC write-back packets. They get stored in the available VC buffers to take part in routing and arbitration decisions. In this work, we propose to delay the travel of LLC write-back packets and keep them stored in the local NoC router buffers for as long as possible. During the time an LLC write-back packet is locally stored, a re-reference request for the same LLC block can be locally replied. We also propose to bring some of the evicted, clean LLC blocks to the local router and keep them stored in buffers to improve our chances of local reply. As shown in Figure 2b, there are sufficient free VCs available in the local input port of NoC routers. So, keeping the evicted LLC blocks (clean and dirty) locally stored will not create injection suppression for other packets. As given in equation (1), LLC miss penalty is dominated by the off-chip transfer time and the role of NoC is limited. With direct reply of LLC blocks from the local router, we propose to avoid off-chip transfer time. Our proposed optimisations have the potential to reduce LLC miss penalty, thereby increasing overall system performance.

Proposed Solution:

Store evicted LLC blocks in underutilised NoC router buffers (VCs) and upon re-reference, generate local replies from the routers to reduce LLC miss penalty.

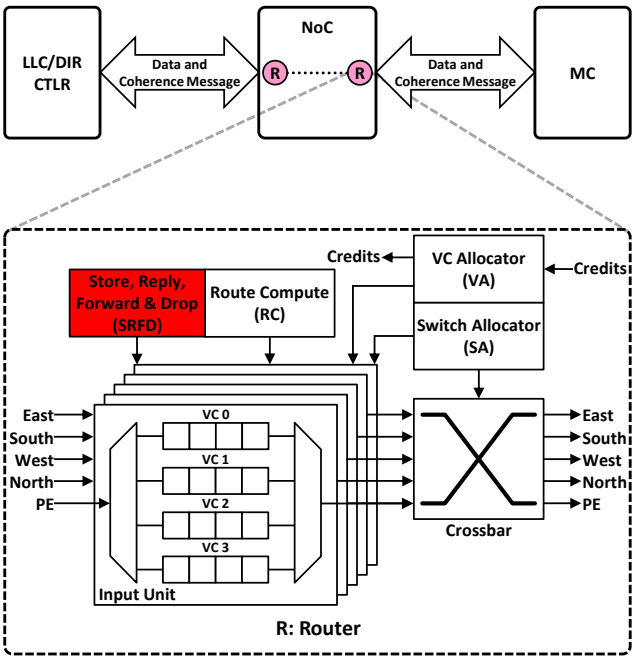


Figure 3: Illustration of the proposed router microarchitecture

Table 1: MOESI Directory Protocol - Stable states of LLC

State	Description
I	Invalid everywhere; LLC and all the private L1 caches
S	Valid, but not exclusive, not owned and not dirty
ILS	Invalid in LLC, but locally held in S state by one or more L1 caches
E	Valid, exclusive, owned but not dirty
ILE	Invalid in LLC, but locally held in E state by one of the L1 caches
O	Valid, owned and potentially dirty but not exclusive
ILO	Invalid in LLC, but locally held in O state by one of the L1 caches
M	Valid, exclusive, owned and potentially dirty.
ILM	Invalid in LLC, but locally held in M state by one of the L1 caches

III. Proposed Architecture

In this section, we explain the working of our proposed architecture. We consider MOESI directory protocol with non-inclusive LLC, and the stable states of LLC are presented in Table 1. Directory is distributed and co-located with the corresponding LLC bank (refer Figure 1). LLC/Directory controllers (LLC/DIR CTLRs) are responsible for maintaining on-chip coherence. Memory controllers (MCs) acts as an interface to off-chip memory banks and maintains off-chip coherence (when applicable). A simple, conceptual illustration of the communication between an LLC/DIR CTLR and an MC with our proposed router microarchitecture is given in Figure 3. The memory hierarchy dictates that L1 cache controller (L1 CTLR) communicates with LLC/DIR CTLR which communicates

Table 2: LLC/DIR Controller - Block replacement. REPLACEMENT: block replacement, ACK-PUT: acknowledgement from MC for PUT request, PUTE/PUTO/PUTM: write-back request for E/O/M state, DATA: evicted block, INV: invalidation, REQ: L1 requester, DIR: directory, MEM: memory

		REPLACEMENT	ACK-PUT
1	S	Remove REQ from Sharers, Store Block State to DIR, Deallocate Block / ILS	
2	E	Send PUTE to MEM / EI^A	
3	EI^A	Stall	Clear Owner, Deallocate Block / I
4	EI^A	Stall	**Add a Store Flag as SET**, **Add a Dirty Flag as RESET**, **Send DATA to MEM**, Clear Owner, Deallocate Block / I
5	O	Send PUTO to MEM / OI^A	
6	OI^A	Stall	Send DATA to MEM, Send INV to All Sharers, Clear Sharers, Clear Owner, Deallocate Block / I
7	OI^A	Stall	**Add a Store Flag as SET**, **Add a Dirty Flag as SET**, Send DATA to MEM, Send INV to All Sharers, Clear Sharers, Clear Owner, Deallocate Block / I
8	M	Send PUTM to MEM / MI^A	
9	MI^A	Stall	Send DATA to MEM, Clear Owner, Deallocate Block / I
10	MI^A	Stall	**Add a Store Flag as SET**, **Add a Dirty Flag as SET**, Send DATA to MEM, Clear Owner, Deallocate Block / I
11	I, ILS, ILE, ILO, ILM: Explanation of transitions is out of scope		

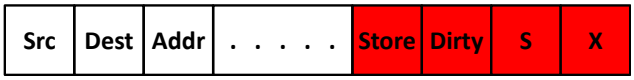


Figure 4: Modified message/packet header

with MC for data and coherence. An L1 CTLR can not bypass the LLC/DIR CTLR to communicate with MC directly. We use the table-based method popularised by Nagarajan et al. [10] to explain the working of MOESI directory protocol. Explaining the protocol for all the possible states and events are beyond the scope of this paper. Hence, we limit the explanation to only the states and events that are related to our proposed work.

A. Local Store: Keeping Evicted LLC Blocks in Router Buffers

On a replacement, a valid cache block is evicted from an LLC bank. Table 2 presents the transitions and actions performed at an LLC/DIR CTLR on a block replacement. A blank entry in the table indicates that the corresponding transition is either impossible or irrelevant for our discussion. A shaded row i is the modified and proposed version of row $i-1$ to implement our optimisations. For example, in Table 2, row 4 is the modified and proposed version of row 3. Within an entry in a shaded row, only the bold actions are additionally performed along with the existing actions to implement the optimisations. A transient state XY^Z denotes that a block is transitioning from a stable state X to another stable state Y and the transition is waiting for an event Z.

With MOESI directory protocol, a valid cache block evicted from an LLC bank can be either clean (S/E) or dirty (O/M). Clean blocks are usually dropped, and dirty blocks are forwarded towards off-chip memory bank for write-back. To evict a dirty block, LLC/DIR CTLR initiates a write-back request (PUTO/PUTM) to send over the NoC towards off-chip memory bank as given in rows 5 and 8 of Table 2. After receiving such a request, MC replies with an acknowledgement (ACK-PUT) to receive the evicted block for write-back. When an ACK-PUT message reaches the LLC/DIR CTLR, it evicts the dirty block and sends the block for write-back to the off-chip memory as a DATA message (rows 6 and 9 of Table 2). Our first optimisation targets these write-back DATA messages on their way to the destination. These messages are marked with a 1-bit flag *Store* in their header for identification, as shown in Figure 4. Related modifications in the LLC/DIR CTLR are given in rows 7 and 10 of Table 2. All the messages travel as packets through the underlying NoC to reach their destination. They enter NoC through the local router which is connected to their tile, get stored in the available buffers of VCs and take part in routing and arbitration decisions. An additional **Store, Reply, Forward & Drop (SRFD)** unit (refer Figure 3) in the local router checks the *Store* flag of all the buffered packets. We consider two-stage NoC routers (1: RC, 2: VA and SA) where SRFD unit works in parallel with the Route Compute (RC) unit. So, while SRFD unit checks the *Store* flag of a packet, RC unit calculates the route for the same packet in parallel. If the *Store* flag is found SET for a packet, SRFD unit disables its VC and switch arbitration (VA and SA). This packet is an evicted, dirty LLC block (DATA message of PUTO/PUTM) which remains stored in the local router unless forwarded for write-back (explained in Section III-C). Both LLC/DIR CTLR and MC are unaware of the proposed optimisation of local store. For LLC/DIR CTLR, the DATA message is on its way or already reached the off-chip memory for write-back. On the other side, since MC sent an acknowledgement (ACK-PUT) to receive the evicted LLC block for write-back, it expects the block (DATA message) and believes that the DATA message is on its way.

Evicted, clean LLC blocks are usually not sent for write-back as the off-chip memory has the same version of such blocks. If a valid cache block evicted from an LLC bank is clean, it is either in S or E state (refer Table 1). To evict a block which is in S state, LLC/DIR CTLR silently updates the directory and deallocates the LLC block without communicating with the MC (row 1 of Table 2). Since the LLC is non-inclusive, evicting a block from an LLC bank does not invalidate other sharers of the block. The corresponding state

of the block in MC is not required to change (already in S), and hence no intimation from LLC/DIR CTLR is necessary. However, to evict a clean block which is in E state, LLC/DIR CTLR needs to communicate with the MC. Even though the block is clean, it is a single copy without any sharers. So, deallocating the LLC block requires the state to be updated to I in both, directory and the MC. The LLC/DIR CTLR initiates a write-back request (PUTE) as given in row 2 of Table 2. Upon receiving a PUTE message, the MC updates the corresponding state of the block to I and sends an ACK-PUT. When LLC/DIR CTLR receives an ACK-PUT message, it updates the directory and deallocates the LLC block (row 3 of Table 2). Since the evicted block is clean, LLC/DIR controller does not send the block for write-back, and the MC never expects it. Our next optimisation proposes to store these evicted, clean LLC blocks which are in E state, in the local router buffers. We initiate a special write-back DATA message of PUTE, set its *Store* flag in the header and send it towards off-chip memory (row 4 of Table 2). SRFD unit in the local router identifies the message based on the *Store* flag and keep it stored there for as long as possible. All the DATA messages of PUTE, PUTO and PUTM are also marked with an additional 1-bit flag *Dirty* to facilitate local reply (explained in Section III-B). The detailed working of the additional SRFD unit is presented in Algorithm 1.

Algorithm 1: Working of SRFD Unit

Input : VC information, modified packet header
Output: Local store or reply, block forward or drop

Notations:
n: Number of virtual channels (VC)
P_i^j: Packet in VC_i where $j \in \{new, stored\}$

if $P_i^{new}[Store] == SET$ **then**
  /* Local store of DATA message of PUT(E/O/M) [III-A] */
  Disable VA and SA for P_i^{new}
else if $P_i^{new}[S] == SET$ **then**
  /* Local reply to GETS message [III-B] */
  for $\forall VC_k$ *where* $VC_k \neq NULL$ **do**
    if $P_k^{stored}[Addr] == P_i^{new}[Addr]$ **then**
      $P_k^{stored}[Src] = P_i^{new}[Dest]$
      $P_k^{stored}[Dest] = P_i^{new}[Src]$
      Deallocate VC_i to drop P_i^{new}
      Enable VA and SA for P_k^{stored}
else if $P_i^{new}[X] == SET$ **then**
  /* Local reply to GETX message [III-B] */
  for $\forall VC_k$ *where* $VC_k \neq NULL$ **do**
    if $P_k^{stored}[Addr] == P_i^{new}[Addr]$ **then**
      if $P_k^{stored}[Dirty] == SET$ **then**
        $P_k^{stored}[Src] = P_i^{new}[Dest]$
        $P_k^{stored}[Dest] = P_i^{new}[Src]$
        Deallocate VC_i to drop P_i^{new}
        Enable VA and SA for P_k^{stored}
      else
        Deallocate VC_k to drop P_k^{stored}

/* Forward/Drop of DATA message of PUT(E/O/M) [III-C] */
for $\forall VC_k$, *if* $VC_k \neq NULL$ **do**
  /* Defensive vacate */
  if $\exists VC_k$ *where* $P_k^{stored}[Store] == SET$ **then**
    if $P_k^{stored}[Dirty] == SET$ **then**
      Enable VA and SA for P_k^{stored}
    else
      Deallocate VC_k to drop P_k^{stored}
  /* Aggressive vacate */
  if $\forall VC_k$, $P_k^{stored}[Store] == SET$ **then**
    if $P_k^{stored}[Dirty] == SET$ **then**
      Enable VA and SA for P_k^{stored}
    else
      Deallocate VC_k to drop P_k^{stored}

B. Local Reply: Responding to Block Requests from Routers

On a cache miss, the corresponding LLC bank requests the block from the off-chip memory. Table 3 presents the transitions and actions performed at an LLC/DIR CTLR on a block miss. Based on the type of miss, LLC/DIR CTLR issues either a read request (GETS) or a write request (GETX) to send over the NoC towards off-chip memory bank as given in rows 1, 7 and 11 of Table 3. After receiving such a data request, MC fetches the requested block from off-chip memory and forwards it to the LLC/DIR CTLR. As per the request, LLC/DIR CTLR may receive the block either in shared (DATA) or exclusive (EX-DATA) state. Our optimisation identifies GETS/GETX messages when they reach the local router to travel over the NoC to their destination. We mark GETS by a 1-bit flag *S* and GETX by a 1-bit flag *X* in the message header as shown in Figure 4. Related modifications in the LLC/DIR CTLR is given in rows 2, 8 and 12 of Table 3.

For a new packet entering the local router with *S* flag SET (GETS message), SRFD unit compares its requested address with the addresses of non-empty VCs. One of the non-empty VCs may have the requested block stored as a packet when it was evicted from the LLC in the recent past (DATA message of PUTE/PUTO/PUTM). If a match is found, we can generate a local reply to the GETS message with a stored DATA message. SRFD unit swaps the source and destination of the matched DATA message (stored packet) with the GETS message (request packet) and drop the request packet, as given in Algorithm 1. The new destination of the stored packet is the same LLC bank from where it was evicted. The stored packet is now enabled for VC and switch arbitration, which were disabled to keep it stored in the local router. Since the destination LLC bank is connected to the very same router, the stored packet gets ejected out of the router through the local output port. Avoiding off-chip travel to fetch the requested block significantly reduces LLC miss penalty. In essence, we satisfy a GETS message with a matching DATA message of PUTE/PUTO/PUTM stored in the local router.

For a new packet entering the local router with *X* flag SET (GETX message), SRFD unit performs the same steps but with an additional check. When a match is found for the requested block in the local router, we have to make sure that the matched block (stored packet) is in exclusive state. SRFD unit checks the *Dirty* flag of the stored packet to identify if it is a DATA message of PUTO/PUTM and not PUTE (explained in Section III-D). If the *Dirty* flag is SET, a local reply with the matched DATA message can be generated from the router. If a match is found in the local router but the *Dirty* flag is not SET (DATA message of PUTE), then the GETX message needs to travel off-chip to fetch the requested block. We drop

Table 3: LLC/DIR Controller - Block miss. GETS: read request from L1, GETX: write request from L1, DATA: shared block from off-chip memory, EX-DATA: exclusive block from off-chip memory, CAN-PUT: cancel PUT request for coherence

		GETS	**GETX**	**DATA**	**EX-DATA**
1	I	Send GETS to MEM / IS^D	Send GETX to MEM / IM^D		
2	I	**Add an S Flag as SET**, Send GETS to MEM / IS^D	**Add an X Flag as SET**, Send GETX to MEM / IM^D		
3	IS^D	Stall	Stall	Send DATA to REQ, Add REQ to Sharers / ILS	Send EX-DATA to REQ, Clear Sharers, Make REQ the Owner / ILE
4	IS^D	Stall	Stall	**If (Store == SET)** { **Send CAN-PUT to MEM** } Send DATA to REQ, Add REQ to Sharers / ILS	**If (Store == SET)** { **Send CAN-PUT to MEM** } Send EX-DATA to REQ, Clear Sharers, Make REQ the Owner / **ILM**
5	IM^D	Stall	Stall		Send EX-DATA to REQ, Clear Sharers, Make REQ the Owner / ILM
6	IM^D	Stall	Stall		**If (Store == SET)** { **Send CAN-PUT to MEM** } Send EX-DATA to REQ, Clear Sharers, Make REQ the Owner / ILM
7	S	Send Data to REQ, Add REQ to Sharers	Send GETX to MEM / SM^D		
8	S	Send Data to REQ, Add REQ to Sharers	**Add an X Flag as SET**, Send GETX to MEM / SM^D		
9	SM^D	Stall	Stall		Send EX-DATA to REQ, Send INV to Sharers except REQ, Clear Sharers, Make REQ the Owner / ILM
10	SM^D	Stall	Stall		**If (Store == SET)** { **Send CAN-PUT to MEM** } Send EX-DATA to REQ, Send INV to Sharers except REQ, Clear Sharers, Make REQ the Owner / ILM
11	ILS	Forward GETS to Owner, Add REQ to Sharers	Send GETX to MEM / SM^D		
12	ILS	Forward GETS to Owner, Add REQ to Sharers	**Add an X Flag as SET**, Send GETX to MEM / SM^D		
13	E, ILE, O, ILO, M, ILM: Explanation of transitions is out of scope				

the matched DATA message of PUTE before forwarding the GETX message towards memory to avoid creating a stale copy of the block. SRFD unit takes care of this operation as given in lines 24-25 of Algorithm 1. We can satisfy a GETX message with a matching DATA message of PUTO/PUTM only and not of PUTE as it may violate cache coherence.

C. Block Forward and Drop: Releasing Stored Blocks

The evicted LLC blocks can not be kept stored in local routers forever as they might create injection suppression during on-chip congestion. Our optimisation is about exploiting underutilised router buffers and not create buffer (VC) unavailability. If a new packet can not be injected into the local router due to VC unavailability, we immediately take a suitable action to vacate one of the VCs where an evicted LLC block is stored. Based on the local input port buffer contention, we implement two ways of vacating a VC occupied by the evicted LLC blocks; defensive and aggressive. When all the VCs are full, defensive approach dictates that if any one of the VCs contains a stored LLC block (DATA message of PUTE/PUTO/PUTM), a VC needs to be vacated. However, with aggressive approach, only when all the VCs are full with stored LLC blocks, a VC is vacated by the SRFD unit (refer Algorithm 1). When multiple VCs have stored LLC blocks, we vacate the oldest of them. After we identify the VC to be vacated, the *Dirty* flag helps us to decide whether to forward

or drop the LLC block stored in that VC. If the VC has a dirty LLC block (DATA message of PUTO/PUTM), it needs to be forwarded towards off-chip memory for write-back. We enable the VC and switch arbitration for the stored block which were disabled when we kept it stored in the VC buffer. If the VC has a clean LLC block (DATA message of PUTE), it is silently dropped as off-chip memory has the same version of the block. Neither the MC sent any acknowledgement to receive the block, nor the LLC/DIR CTLR sent the block for write-back. DATA message of PUTE is a special message generated for our optimisation; to give more scope for local reply.

D. Maintaining Cache Coherence

When an evicted, dirty LLC block is sent for write-back, the LLC/DIR CTLR invalidates all its sharers (if exists) and clear the directory entry (refer rows 6 and 9 of Table 2). Thus, when we keep such blocks (DATA messages of PUTO and PUTM) stored in the local router, no one else has copy of the blocks. MC is expecting these blocks for write-back and believes that they are on the way. As per the memory hierarchy, a new request for any of these blocks reaches the same LLC/DIR CTLR from where they were evicted. LLC/DIR CTLR issues a new request (GETS/GETX) for the block towards the same MC. When the request reaches the local router and a local reply is generated, the stored block is sent back to the same LLC bank for where it was evicted. While generating local replies with the stored blocks (DATA messages of PUTO and PUTM), we need to preserve their states as they are dirty. So, irrespective of the type of request (GETS or GETX), a local reply is always generated in M state. This makes sure that the block can not be discarded and will be eventually sent for write-back. When LLC/DIR CTLR receives a block in the form of a DATA or EX-DATA message, we perform an additional check of the *Store* flag (rows 4, 6 and 10 of Table 3). If the *Store* flag is found SET, LLC/DIR CTLR learns that the block has come from the local router (local reply). MC was expecting the block for write-back and now we have generated a local reply with the block. So, we make LLC/DIR CTLR send a CAN-PUT message to the MC. With the CAN-PUT message, MC knows that the evicted block will not reach for write-back and updates the state to M to maintain coherence.

When a clean block which is in E state is evicted, we make LLC/DIR CTLR generate a special DATA message towards off-chip memory (row 4 of Table 2). Neither the LLC/DIR CTLR sent the message for write-back, nor the MC expects it. The purpose of this message is to keep the evicted, clean LLC block stored in the local router and improve the scope of local reply. Since the block is clean and not required to be sent for write-back, a local reply with such a stored block (DATA message of PUTE) is always generated in S state. Hence, a DATA message of PUTE can generate local replies to only GETS messages and not GETX messages. Upon receiving the local reply, LLC/DIR CTLR issues a CAN-PUT message to the MC. Since the MC was not expecting anything, when a CAN-PUT arrives it understands that a local reply is generated in S state and updates its record to maintain coherence.

Table 4: 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
Memory Bank	4; one located at each corner
Cache Coherence	MOESI distributed directory
NoC	8×8 2D mesh, 4 VCs/port, 128-bit flit channel
Routing	2-stage routers, X-Y dimension-order routing
Packets	1-flit for control packets, 5-flit for data packets
Benchmarks	SPEC CPU2006 (Multiprogrammed)

Table 5: Application scheduling for various workload mixes

Mix	Benchmark Instances			
M1	GemsFDTD(64)			
M2	gromacs(64)			
M3	astar(16)	cactusADM(16)	omnetpp(16)	soplex(16)
M4	astar(16)	calculix(16)	GemsFDTD(16)	soplex(16)
M5	bzip2(16)	calculix(16)	gromacs(16)	xalancbmk(16)
M6	cactusADM(16)	calculix(16)	GemsFDTD(16)	omnetpp(16)
M7	cactusADM(16)	namd(16)	omnetpp(16)	xalancbmk(16)

IV. Experimental Analysis

We consider the following architectures for evaluation:

- **Baseline:** Without any optimisation.
- **DB-Defensive:** Store evicted, dirty LLC blocks in local router and use defensive block forward.
- **DB-Aggressive:** Store evicted, dirty LLC blocks in local router but use aggressive block forward.
- **CB+DB-Aggressive:** Store both clean and dirty LLC blocks in local router and use aggressive block forward.

A. Simulation Framework and Workloads

We model the baseline and proposed architectures on event-driven gem5 simulator [11]. Our system configuration is similar to Intel Xeon Phi Processor 7235 [12] with shared and distributed L2 cache (LLC). Due to certain limitations in gem5, we could not exactly model the cache configuration of Intel Xeon Phi Processor 7235. Our cache configuration is not chosen to give undue advantage to the proposed optimisations. Rather, it challenges the optimisations with a hit rate of around 95% for all the benchmarks we used. Our system configuration is presented in Table 4 for reference. We modify GARNET [13] module in gem5 to implement our modified router microarchitecture. We modify MOESI_CMP_directory protocol in Ruby to implement and maintain cache coherence.

To evaluate the performance, we consider SPEC CPU2006 multiprogrammed benchmarks to mimic a modern NoC based TCMP running multiple applications in parallel. We create two different types of workloads with varying misses per kilo instructions (MPKIs) and re-reference interval (refer Figure 2c) to run on all the 64 cores of the system, as given in Table 5. First type runs 64 copies of the same benchmark on all the 64 cores (1×64: 64). The second type runs a random combination of 4 different benchmarks with 16 copies each (4×16: 64). By separately profiling each benchmark, we choose a smaller representative window of instructions

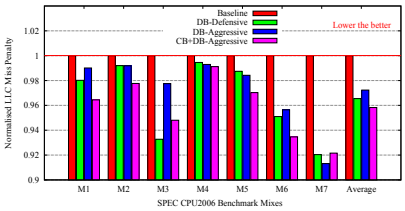


Figure 5: LLC Miss Penalty

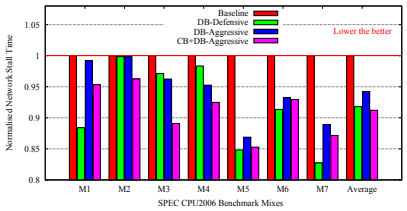


Figure 6: Network Stall Time

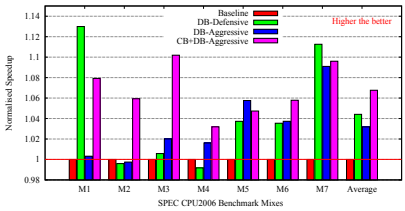


Figure 7: Speedup

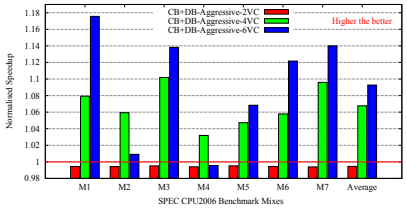


Figure 8: Variation of number of VCs

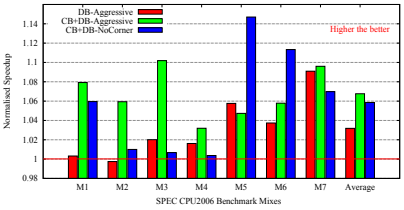


Figure 9: No local store in corner routers

to have a tractable simulation time. We evaluate the baseline and proposed architectures for performance, storage, area and power. Our results are normalised with respect to the baseline.

B. Performance Evaluation

LLC Miss Penalty: It is defined as the number of cycles required to replace an existing cache block in LLC with an incoming block. LLC miss penalty directly reflects the effectiveness of the proposed local reply optimisation. Figure 5 shows the normalised LLC miss penalty with respect to the baseline architecture. With local replies, the proposed architectures reduce LLC miss penalty for all the workload mixes. In general, *CB+DB-Aggressive* architecture performs better, as with more locally stored blocks (both clean and dirty), it has more scope of local replies. However, for mixes M3 and M7, *CB+DB-Aggressive* perform relatively less compared to other proposed architectures. Mixes M3 and M7 contains high MPKI valued benchmarks like *astar*, *soplex* and *xalancbmk* which frequently inject packets into the network. As a result, the evicted LLC blocks are not able to stay stored in the local routers for long. A maximum reduction of 9% in LLC miss penalty is achieved by one of our proposed architectures.

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 LLC blocks in local routers impact the NoC communication. Figure 6 shows the normalised network stall time with respect to the baseline architecture. As expected, across all the workloads, our proposed architectures significantly reduce network stall time. With local reply from the NoC routers, we avoid both on-chip and off-chip travel time as given in equation (1). Avoiding on-chip travel time indirectly translates into reduced network stall time. However, occupying VC buffers for long may affect NoC communication during peak network congestion. To avoid such a scenario, we proposed a defensive and an aggressive block forward/drop approach. *DB-Defensive* performs significantly better than *DB-Aggressive* and *CB+DB-Aggressive* architectures for mixes M1 and M7. *GemsFDTD* having the highest MPKI value among the presented benchmarks is run on all the 64 cores in M1. Since M1 frequently injects network packets from all the 64 cores, the routers are flooded with packets. *DB-Aggressive* and *CB+DB-Aggressive* architectures vacate a VC for new packet injection only when all the VCs are occupied by evicted LLC blocks. As a result, new packets suffer an injection delay that increases the network stall time.

Speedup: We use total instructions per cycle (IPC) of the system to compare speedup between baseline and the proposed architectures. Figure 7 shows the normalised speedup with respect to the baseline architecture. From the improvements in LLC miss penalty and network stall time, the increase in system speedup with the proposed architectures is intuitive. We achieve a maximum system speedup of 13% and an average system speedup of 7% for the presented workloads. In general, *CB+DB-Aggressive* performs well across all the workloads with a few exceptions. Mix M1 performs the best with *DB-Defensive* architecture among all the workloads. M1 running *GemsFDTD* benchmark has a very small re-reference interval of only around 22000 cycles (refer Figure 2c). However, frequent network packet injection and the delay due to aggressive vacate approach in the other two architectures, *DB-Defensive* performs better. Mix M2 runs a low MPKI valued benchmark *gromacs*, that has the largest re-reference interval. Since *CB+DB-Aggressive* stores maximum evicted LLC blocks for the longest duration, infrequent LLC block re-references by M2 are also locally replied. The other two architectures suffer performance degradation, as mix M2 being a light workload can not reap the benefits of local replies. One of the main reasons for mix M7 to perform well across all the architectures is the composition of benchmarks with low re-reference interval. *cactusADM* has one of the lowest (quickest) re-reference interval among all the presented benchmarks.

C. Sensitivity Analysis on Design Parameters

As our optimisation is based on storage of evicted LLC blocks in buffers of VCs, we also explore the impact of number of VCs per port on system performance. For all the results discussed so far, we have considered number of VCs as 4 (as presented in Table 4). However, in Figure 9, we show that

varying number of VCs will have different implications on the system performance. Based on the available number of VCs in a system, our optimisations achieve appropriate improvement.

While understanding the VC availability (refer Figure 2b), we find that corner routers have fewer free VCs compared to others. The unavailability of free VCs can be attributed to the presence of MCs in the corners servicing continuous requests. Since our optimisation is about exploiting free VCs, we do not benefit much in the corner routers. To explore further, we design a heterogeneous architecture called *CB+DB-NoCorner* where the corner routers do not participate in local store and reply. Figure 8 shows the comparison of system speedup for *CB+DB-NoCorner* with other architectures.

D. Storage, Area and Power Evaluation

We use 4 additional bits (*Store*, *Dirty*, *S* and *X*) in the packet header (refer Figure 4) to facilitate the working of SRFD unit. However, a typical packet header is much smaller than the channel bandwidth; hence we can accommodate the additional 4 bits in the header without any storage overhead. We use DSENT [14], integrated with gem5 to evaluate the area and power of 8×8 2D mesh NoC in our proposed architectures. In DSENT, we use 22nm processor technology at 1GHz operating frequency. The addition of SRFD unit in the routers incur a negligible area overhead of 1.69% and a leakage power overhead of 1.92% compared to the baseline routers. However, due to improvement in overall system performance, we achieve 4.21% reduction in dynamic power compared to the baseline routers. Since SRFD unit works in parallel to the RC unit (refer Section III-A), it is not in the critical path of execution.

V. Related Work

One of the first works by Mizrahi et al. [15] attempted to change the abstraction of NoC from communication to storage. They proposed to migrate the data cache entirely in the NoC routers. Going forward in the same line, Eisley et al. [16] decoupled data and coherence and proposed to keep only the coherence directories in NoC. Yanamandra et al. [17] combined the goodness of both and proposed to keep frequently used cache blocks along with the coherence directories. There is another work by Wang et al. [18] which is also focused on in-network cache and coherence. However, almost all the proposed optimisations require additional storage in the NoC routers. Recently, Jindal et al. [19] proposed to reuse a design-for-debug (DFD) storage hardware as extended VCs in routers.

Existing literature argues that NoC router buffers are under-utilised. For example, SPLASH-2 benchmarks have an average router utilisation of less than 20% [20]. The era of power aware NoC designs explored bufferless and minimally buffered routers [21][22]. Now, the trend is more towards application and computation aware NoC designs [23][24]. Irrespective of the promising design alternatives, input buffered NoC routers are employed in modern TCMPs for scalable on-chip bandwidth [6]. In a new and promising attempt, Das et al. [25] proposed to exploit idle buffers of NoC routers to store dirty blocks evicted from private L1 caches. Whereas, our work can be thought of as an implementation of victim caching [26] for LLC; using underutilised VCs to reduce off-chip miss penalty.

VI. Conclusion

In this work, we proposed a set of NoC based TCMP architectures to reduce LLC miss penalty. We used the underutilised buffers of NoC routers to keep evicted LLC blocks stored. Future references to these evicted blocks, now stored in local routers are directly replied from the routers. Local reply avoids off-chip travel to fetch a block and significantly reduces LLC miss penalty. We also proposed two approaches to forward/drop the locally stored LLC blocks in due time and avoid injection suppression. Since the number of evicted LLC blocks are way more than what can be accommodated in local routers, we explore nearby storage in the future work.

References

[1] (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

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

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

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

[5] A. Sodani *et al.*, "Knights Landing: Second-Generation Intel Xeon Phi Product," in *IEEE Micro*, 2016.

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

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

[8] J. Balkind *et al.*, "OpenPiton: An Open Source Many-Core Research Framework," in *ASPLOS*, 2016.

[9] B. K. Daya *et al.*, "SCORPIO: A 36-Core Research Chip Demonstrating Snoopy Coherence on a Scalable Mesh NoC with In-Network Ordering," in *ISCA*, 2014.

[10] V. Nagarajan *et al.*, *A Primer on Memory Consistency and Cache Coherence*. Morgan & Claypool, 2020.

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

[12] (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

[13] N. Agarwal *et al.*, "GARNET: A Detailed On-Chip Network Model inside a Full-System Simulator," in *ISPASS*, 2009.

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

[15] H. E. Mizrahi *et al.*, "Introducing Memory Into The Switch Elements Of Multiprocessor Interconnection Networks," in *ISCA*, 1989.

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

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

[18] J. Wang *et al.*, "Network Caching for CMPs," in *IPCCC*, 2009.

[19] N. Jindal *et al.*, "Enhancing Network-on-Chip Performance by Reusing Trace Buffers," *IEEE TCAD*, 2020.

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

[21] T. Moscibroda and O. Mutlu, "A Case for Bufferless Routing in On-Chip Networks," in *ISCA*, 2009.

[22] C. Fallin *et al.*, "MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect," in *NOCS*, 2012.

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

[24] K. Sangaiah *et al.*, "SnackNoC: Processing in the Communication Layer," in *HPCA*, 2020.

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

[26] N. P. Jouppi, "Improving Direct-Mapped Cache Perf. by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," in *ISCA*, 1990.