IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 29 November 2022, accepted 2 January 2023, date of publication 5 January 2023, date of current version 11 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3234933

RESEARCH ARTICLE

NCDE: In-Network Caching for Directory Entries to Expedite Data Access in Tiled-Chip Multiprocessors

JAE EUN SHIM[1], (Student Member, IEEE), MINGU KANG[2], (Student Member, IEEE), AND TAE HEE HAN[3], (Senior Member, IEEE)
[1]Department of Artificial Intelligence, Sungkyunkwan University, Suwon 16419, South Korea
[2]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, South Korea
[3]Department of Semiconductor Systems Engineering, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Tae Hee Han (than@skku.edu)

This work was supported in part by the National Research and Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (MSIT) under Grant 2020M3H2A1076786; in part by the Ministry of Trade, Industry and Energy (MOTIE) under Grant 20011074; in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) funded by the Korean Government (MSIT) through the Artificial Intelligence Graduate School Support Program, Sungkyunkwan University, under Grant 2019-0-00421; and in part by the MOTIE and KEIT (Development of System Level Design and Verification for in Storage Processing Architecture Based on Phase Change Memory) under Grant 20010560.

ABSTRACT The processing of data-intensive applications, followed by an unprecedented amount of data traffic, drives explosive accesses to the memory subsystem. The overloaded memory subsystem experiences increased data access latency. To expedite data access, a network caching technique that leverages network-on-chip (NoC) virtual channels (VCs) as an expanded memory subsystem has emerged. Previous network caching studies focused on utilizing VCs on the NoC's local input port as a victim cache to reduce local data access latency. In contrast to previous studies, we explore the opportunity of mitigating problems associated with shared data access via *in-network caching for directory entries* (*NCDE*), which can utilize every input port's VCs to hold directory entries. NCDE exploits VCs as the victim and prefetch buffers of the directory entries, each reducing directory eviction-induced invalidations and simplifying the cache-to-cache (C2C) data transfer. The effectiveness of NCDE was evaluated using a gem5 full-system simulator, and the results show that the average memory access time (AMAT) and workload execution time were reduced by 7.69% and 5.82%, respectively. As a cost for accelerating the data access latency, implementing NCDE incurs a negligible router area overhead of 1.56%.

INDEX TERMS Directory-based cache coherence protocol, directory caching, network-on-chip, sparse directory, virtual channel (VC) utilization.

I. INTRODUCTION

Many-core architecture has been highlighted as a vital component of high-performance computing (HPC) platforms owing to its high parallel processing capability [1]. A real-world example of a many-core architecture is the tiled-chip-multiprocessor (TCMP), which integrates multiple tiles comprised of processor cores and caches via network-on-chip (NoC) [1], [2], [3], [4]. TCMP elevates parallel processing capability by investing a large portion of the area budget to increase the core count. Increasing the cache size is considered the single viable solution to cope with the increased capacity miss rates in data-intensive applications [5]. In the worst-case scenario, almost half of the cache lines are shared as a result of using a large amount of data across a massive number of cores. Frequent access to shared cache lines can increase the cache data transfer rate by up to 27.40%, leading to TCMP suffering from increased data access latency [6].

Various techniques that exploit NoC as a supplementary buffer for the memory subsystem, referred to as network

The associate editor coordinating the review of this manuscript and approving it for publication was Songwen Pei.

TABLE 1. Summary of the previous studies addressing DE eviction.

Category	Work	Key idea	Pros	Cons
Directory size optimization	Fang, Lei, *et al.* [7]	Reduce the number of DE or the size of DE	Only modifies directory architecture	Imprecise coherence information
	SCD [8]			
	SPACE [9]			
Exploiting memory hierarchy	NUDA [10]	Leverages additional memory space for DE	Reduce directory area overhead	Increase memory access complexity
	Shukla, *et al.* [11]			
	ZEV [12]			
OS-supported cache coherence	Fensch, *et al.* [13]	Retrofit the function of directory to OS	Fully prevent DE eviction-induced invalidation	Requires modification of OS and application software
	Shim, *et al.* [14]			
	Davari, *et al.* [15]			
	SARC [16]			

caching, have been developed to accelerate data access [17], [18], [19], [20], [21]. Several researchers have investigated the possibility of exploiting virtual channels (VCs) of the local input port as a sort of victim cache based on the observation that VCs are underutilized in typical operating scenarios of NoC [20], [21]. TCMP leverages parallelism through multithreading, resulting in frequent access to shared data [22]. Utilizing the local input port VC as a victim cache is advantageous when data are re-referenced within the same core; however, it exposes a weakness in shared data access.

Focusing on the unexplored opportunities associated with shared data access in TCMP, we devised an *in-network caching for directory entries* (*NCDE*) that holds directory entries (DEs) in VCs. By utilizing VCs as victim buffer and prefetch buffer for DEs, respectively, thus reduces DE eviction-induced invalidations and simplifies the cache-to-cache (C2C) data transfer. To the best of our knowledge, NCDE is the first network caching method that employs VCs as the victim and prefetch buffers for DEs. NCDE accelerates shared data access and maximizes NoC resource utilization by participating VCs at all input ports for network caching.

The eviction of a DE leads to invalidating the associated private cache line because the coherence of the cache line can no longer be guaranteed. As a result, DE eviction-induced invalidations increase the miss rate of private caches and consequently hamper the overall system performance of TCMP. The performance-criticality of DE eviction was investigated by Chaudhuri [12] through running multithreaded workloads with varying directory sizes. The results indicate that by reducing the number of DEs by 25%, the execution time for the multithreaded workloads can be prolonged by up to 10% due to the increased DE evictions, which causes coherence cache miss. The primary causes of the performance degradation are an increased private cache miss rate and the associated heavier network traffic. The fact that DE eviction-induced invalidation is ignorant of memory locality causes a substantial increase in data access latency. When the CPU accesses the invalidated data, it experiences an expensive miss penalty while retrieving up-to-date data from the shared last-level cache (LLC) or, in the worst-case scenario, from main memory, which also exacerbates network congestion. NCDE helps to alleviate the challenges posed by DE evictions with minimal hardware modifications and area overhead.

To sum up, NCDE exploits VCs as a victim and prefetch buffer for DEs to reduce DE eviction-induced invalidations and simplify the cache-to-cache (C2C) data transfer. Through NCDE, shared data access, which frequently occurs in TCMP, can be expedited while maximizing NoC resource utilization by engaging every input port's VCs for network caching. The remainder of this paper is organized as follows. Section II discusses and contrasts NCDE against related work, and Section III addresses the necessary background of directory-based cache coherency in TCMP associated with NCDE. Section IV illustrates the implementation of NCDE from the perspective of NoC router architecture and operation algorithms. Section V presents the simulation results and analysis under various conditions. Finally, concluding remarks are drawn in Section VI.

II. RELATED WORK

There have been several research efforts, referred to as network caching, that projected NoC in the context of supporting memory subsystems to facilitate faster data access [17], [18], [19], [20], [21]. The work of Mizrahi et al. [17] is notable for conducting one of the earliest attempts at exploiting NoC as a supplementary medium for the memory subsystem. Mizrahi et al. advocated that NoC can be included in the memory hierarchy by placing a cache-like buffer on the NoC routers.

Eisley et al. [18] proposed a network caching technique to simplify the C2C data transfer. Implementing virtual tree links throughout NoC removes directory lookup from the direct route to the valid data. However, the virtual tree-based method requires significant changes to the NoC design, such as adding a buffer to implement the function of the directory within the router. Augustine et al. [19] focused on the congestion caused by burst requests to a particular last level cache (LLC) slice. The NoC router features a separate router buffer cache (RBC) for storing LLC lines to eliminate an LLC queuing delay. Additionally, the similarity of the page granularity access was leveraged to select the LLC line to be stored in the RBC. However, this approach did not fully exploit the performance improvement opportunity because it addressed only the LLC queue delay, which was accompanied by an area overhead caused by the RBC.

Although previous network caching methods differ in their specific areas of focus, they generally require additional buffer space in the router. LSR-FD, which exploits VCs as victim buffers for a private cache, is an area-efficient network caching [20]. Evicted private cache lines are stored in the local input port VC to respond to local accesses with a reduced private cache miss penalty. However, LSR-FD was intended to accelerate local data access, thereby exhibiting a shortcoming when executing multithreaded applications. In addition, because the evicted cache lines were selected to be placed in the VC, a scenario in which the VC depth is less than the cache line size was not considered.

NCDE utilizes VCs as an extended buffer space for DE and can resolve frequent shared data access in the multithreading operation of the TCMP. NCDE deploys VCs as a victim and prefetch buffers for DE to reduce directory eviction-induced invalidation and facilitate C2C transfer. In addition, by using VCs residing in all input ports, the NoC resource utilization is increased. Furthermore, because the packet-type DE is smaller than the cache line size, the requirement of the VC depth to apply the method can be relieved (Section IV illustrates the numerical analysis regarding the size of the packet-type DE).

An inefficient DE eviction that is unaware of memory locality will invalidate the private cache line more often than is required, which will ultimately result in a in TCMP performance degradation. The earlier studies that attempted to reduce DE eviction-induced invalidation can be categorized into three parts, as presented in Table 1.

The methods of the first category focused on saving the directory area. Because the conflict miss in the directory is the primary reason for the DE eviction, the occurrence of DE eviction-induced invalidation is closely related to the size and organization of the directory. Therefore, relaxing the area requirement for the directory can effectively reduce invalidations caused by DE evictions. The objective of studies in the first category [7], [8], [9] is to reduce the number or size of DEs. However, these methods must tolerate the imprecise representation of coherence information caused by uniting multiple DEs into a single DE or encoding DE data. NCDE can reduce DE evictions while maintaining the accuracy of coherence information by using VCs as the victim buffers for DEs.

Second, several architectural supports from the memory hierarchy were investigated to mitigate DE eviction-induced invalidations, such as expanding the directory with LLC [11], [12] or DRAM [10]. These approaches are beneficial in reducing invalidations owing to DE eviction with a negligible area overhead; however, they complicate the memory access procedure, thus increasing the directory access latency. Meanwhile, NCDE does not complicate the data access mechanism, since it leverages VCs, which are network resources that do not directly affect the memory hierarchy.

The third category maintains cache coherence with the operating system (OS) support. In this context, the use of cache coherence-aware page mapping [13], thread migration [14], and private cache self-invalidation [15], [16] can mitigate the problem of DE eviction-induced invalidation.

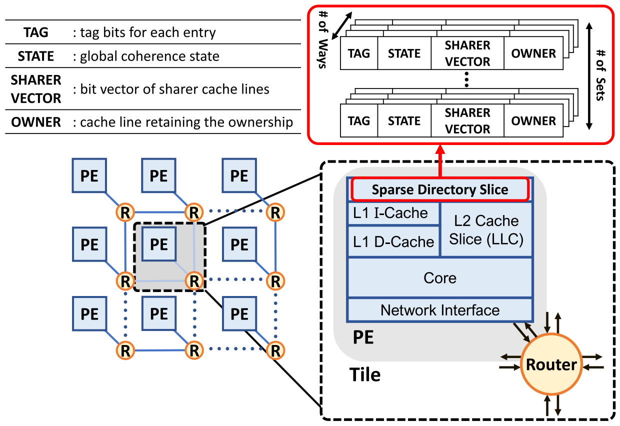

TAG : tag bits for each entry
STATE : global coherence state
SHARER VECTOR : bit vector of sharer cache lines
OWNER : cache line retaining the ownership
# of Ways
# of Sets
TAG
STATE
SHARER VECTOR
OWNER
PE
R
Sparse Directory Slice
L1 I-Cache
L1 D-Cache
L2 Cache Slice (LLC)
Core
Network Interface
Router
Tile

FIGURE 1. Basic TCMP architecture. The tiles of the TCMP architecture consist of a single core with a router. Each core has an L1 private cache and uses the L2 cache as a shared last level cache (LLC).

However, these methods involve the overhead of modification in OS kernel and/or application software. In addition, the lengthy processing time of the software-based solution is also a drawback.

III. DIRECTORY IN TCMP

Fig. 1 shows the configuration of a basic NoC-based TCMP, which is used as a reference architecture to explain the necessary background. As shown in Fig. 1, a TCMP typically comprises tens of tiles in a 2D-mesh topology, each of which includes a router and processing element (PE). For cache coherence maintenance, a directory-based cache coherence protocol is preferred for a TCMP owing to the benefit of scalable traffic volume [23].

A. SPARSE DIRECTORY

Compared with the traditional directory, which strictly allocates a DE for each physical address with the granularity of the cache line, the sparse directory, which is organized as a tagged set-associative structure, is more area-efficient. Consequently, the sparse directory structure is a cost-effective strategy for TCMPs, which cannot devote a significant portion of the area budget to the memory subsystem. The sparse directory is sliced according to the number of tiles within TCMP and then distributed to each tile. Each directory slice is responsible for an equal portion of the available physical address space, similar to shared and distributed LLCs. Each entry consisting of sparse directory slices is configured as depicted in Fig. 1. During sparse directory lookups, TAG is used to ensure that the upper portion of a physical address matches, analogous to its function in normal cache lookups. STATE and SHARER, tracked by the sparse directory entry, specify the global coherent state of the cache line and the tile containing the cache line, respectively. Finally, when STATE is **M** (modified) or **O** (owned), OWNER indicates a private cache with the ownership.

TABLE 2. Notations used in directory-based cache coherence protocol.

Notation	Description
REQ	Tile of the L1 cache that is requesting a cache data
HOME	Tile of the LLC/DIR slice that is responsible for the requesting address
RESP	Tile containing the valid, up-to-date data in L1 cache
GETS	Read request message from REQ
FWD-GETS	Redirected message of GETS from HOME. Destination is RESP
DATA	Response message of FWD GETS from RESP containing valid, up-to-date cache data
UNBLOCK	Message to notify HOME that all transactions are finished

DE eviction necessarily invalidates the associated L1 cache lines. This type of invalidation degrades TCMP performance because it is unaware of memory locality [12], [24]. Configuring the associativity of the sparse directory to match the aggregated associativity of the L1 cache (core count × number of L1 cache associativity) can completely prevent DE eviction; however, it prolongs latency with significant area overhead. Therefore, a certain amount of DE eviction-induced invalidation has to be tolerated as a cost of adopting a sparse directory structure. NCDE utilizes VCs as victim buffers for DE to mitigate this fundamental problem stemming from the area-efficiency of sparse directories. This is an area-efficient solution because VCs, which already exist, are leveraged as an additional buffer for sparse directory expansion.

B. DIRECTORY-BASED COHERENCE MESSAGE TRANSACTION

The coherence message transaction in the directory-based cache coherence protocol is illustrated using an example of the L1 cache read miss being handled through C2C data transfer. This type of coherence message transaction is the target to be simplified by utilizing VCs as a prefetch buffer for DE. Therefore, the basic procedure is discussed with the corresponding notations prior to the detailed explanation of NCDE. Table 2 summarizes the notations used in this study, based on the popular work of Sorin et al. [25].

Fig. 2 shows a coherence message transaction regarding messages and related REQ, HOME, and RESP. REQ in Fig. 2, which experienced an L1 cache read miss, initiates coherence message transactions through the GETS message. The message transaction procedure can be described as follows: ① GETS message is transmitted to HOME with a directory slice to acquire owner information that possesses up-to-date data of the address, ② HOME forwards the received GETS message to RESP, ③ DATA is transmitted from RESP to REQ, and ④ HOME is notified that the request is completed through UNBLOCK.

$$Hit\_time_{C2C} = t_{GETS} + T_{DIR}^{lookup} + t_{FWD-GETS} + T_{L1}^{lookup} + t_{DATA} \qquad (1)$$

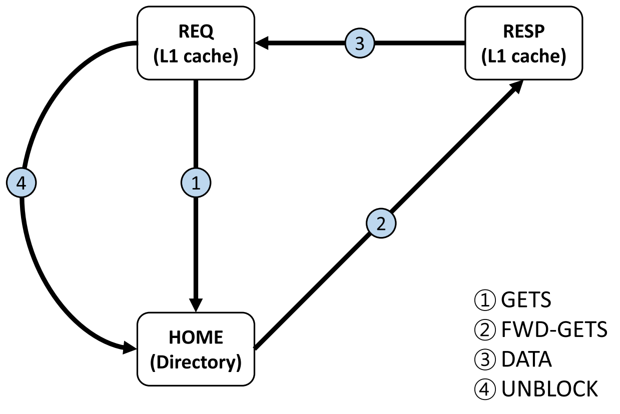


FIGURE 2. Example of coherence message transaction of the C2C data transfer, which handles L1 cache read miss.

The example of C2C data transfer is formulated in equation 1 for the analysis from the perspective of latency. Herein, Hit_time_{C2C} refers to the overall latency required by REQ to receive DATA after generating GETS. Hit_time_{C2C} can be further broken down into five terms, where T_{DIR}^{lookup} and T_{L1}^{lookup} denote the latency consumed by interrogating the directory slice and L1 cache, respectively. Meanwhile, t_{GETS}, $t_{FWD-GETS}$, t_{DATA} denote the message transmission latencies taken by GETS, FWD-GETS, and DATA, respectively. Message transmission latency is the time consumed by the packet of the message while traversing NoC.

The occupancy of terms representing message transmission latency in (1) grows with the core count because the expected value of the hop count increases [26], [27], [28]. Therefore, in a system with a large core count, such as TCMP, the underlying NoC plays an important role in C2C data transfer. As TCMP frequently accesses shared data owing to multithreading, C2C data transfer commonly occurs [29], [30]. Consequently, the impact of reducing such message transmission latency contributes to shortening the shared data access latency in the TCMP.

IV. NCDE

The design goal of NCDE can be summarized as directory-based network caching that focuses on shared data access and resource utilization by leveraging VCs in TCMP architecture. Specifically, NCDE utilizes every input port's VC to reduce the inherent DE eviction-induced invalidation and simplify the C2C data transfer. The implementation efforts to achieve these design goals are categorized into three parts: defining the packet format for directory entry (PDE), PDE-aware VC for effectively exploiting a VC that holds PDE, and operation of NCDE unit to utilize VCs for DEs. Each method for exploiting VCs as victim buffer and prefetch buffer is referred to as victim directory caching (victim DC) and prefetch directory caching (prefetch DC). First, an overview of NCDE is provided, and a detailed illustration of the three implementation efforts is presented.

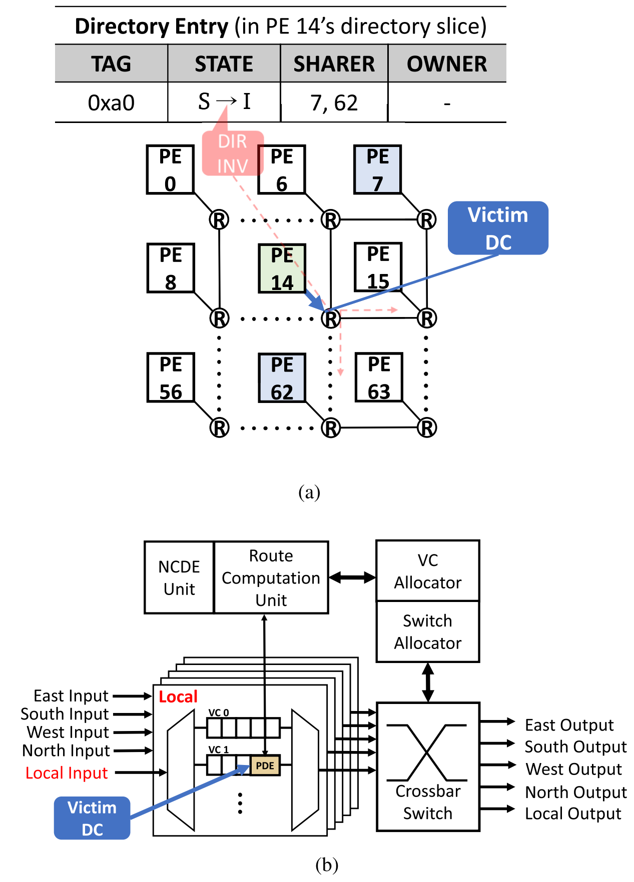


FIGURE 3. Procedure of victim DC in TCMP: (a) basic flow of PDE by victim DC, (b) local input port of router in PE14.

A. OVERVIEW OF NCDE

Two-level cache hierarchy with directory-based MOESI coherence protocol is considered a memory subsystem of the baseline architecture for explaining NCDE design. NoC of the baseline architecture is assumed to adopt 2D mesh topology with wormhole flow control and uses VC. Representative TCMP models [1], [2], [3], [4] were referred to configure a baseline architecture that appropriately reflects the architectural characteristics of TCMP. The baseline architecture uses 128-bit flit as modern NoCs does, in purpose to provide amenable interconnection latency considering data transfer of a cache line granularity (generally 64B) [2], [31], [32].

Fig. 3 depicts the overall procedure of victim DC with the perspective of the flow of PDE and the corresponding router. TCMP's sparse directory must tolerate a certain amount of DE eviction-induced invalidation. Fig. 3a shows the procedure when the DE of tag 0xa0 in PE14 is evicted, which transitions to **I** (invalid) state. Because the corresponding DE was **S** (shared) state with sharer cache lines in PE7 and PE62, these cache lines must be invalidated before the DE to be evicted if the victim DC is not adopted. Victim DC, which holds evicted DEs on the VCs of the local input port, can handle DE eviction without sending an invalidation message to the sharer cache lines. Because the evicted DE is actually exported from the sparse directory slice to the local input port VC, sharer cache lines can be managed. Fig. 3b depicts the corresponding router of PE14 holding the PDE in the VC of the local input port after the DE of tag 0xa0 is evicted.

Fig. 4 depicts the overall prefetch DC procedure from the perspective of the flow of the PDE and corresponding router. A C2C transfer inherently requires a maximum of 3-hop message transactions, and the message transmission latency of each transaction tends to be proportional to the core count [26], [27], [28]. The prefetch DC aims to shorten the message transmission latency for C2C data transfer by simplifying message transactions. To facilitate the message transactions, the request for the shared data of the L1 cache is predicted and the corresponding DE is transmitted to the router of the PE containing the predicted L1 cache.

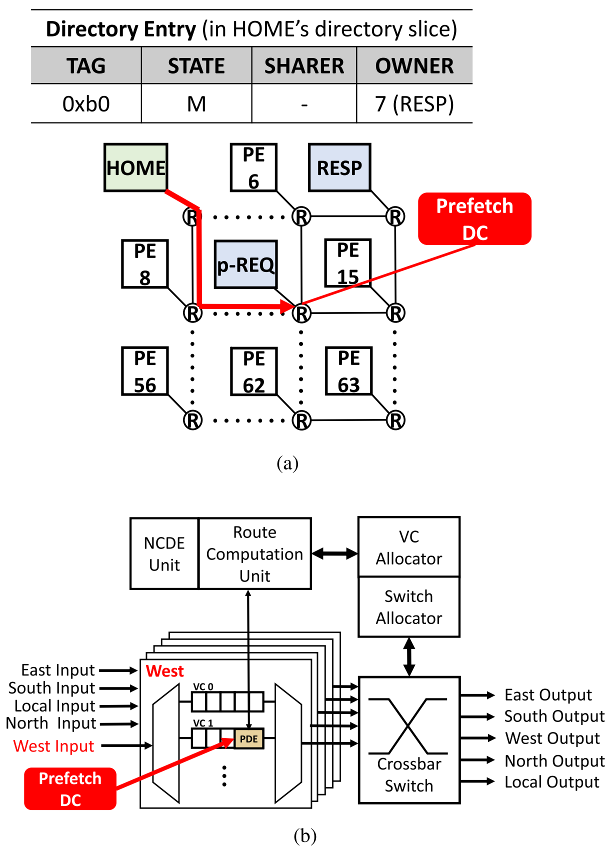
The PDE is stored in the input port VC, which is used for entering the router (the west input port in the example in Fig. 4b). Input ports, except the local port, can be utilized for the prefetch DC, and a total of four input ports are available in the case of the 2D mesh topology. When the expected request occurs in the predicted REQ (p-REQ) in PE14, coherence information originally acquired from HOME is provided at the router in advance. By virtue of the coherence information provisioned from the PDE, a detour for the directory lookup is not required, and the request traverses directly to RESP. Fig. 4a shows an example of a prefetch DC, with the DE of tag 0xb0 being transferred to the predicted REQ (p-REQ) in PE14. Since the DE is in **M** (modified) state, there must be an apparent owner; in this case, the L1 cache in PE7 is the owner. The PDE containing ownership information is transmitted to the router of PE14. Fig. 4b depicts the router of PE14, which stores the prefetched PDE in its west input port VC.

B. PACKETIZED DIRECTORY ENTRY

While implementing the network caching for the DE, we focused on piggybacking on the behavior of the existing router architecture; for this purpose, the directory entry is stored in VC in the form of a packet. In contrast to the cache line, the packetization of DE is handled only by NCDE. Therefore, defining the format of PDE is necessary.

For a PDE head flit, it does not differ from non-PDE packets. The body flit should contain the DE, and the size of the components in DE determines the number of flits for the payload of PDE. DE contains SHARER VECTOR, STATE, OWNER, and TAG. As the head flit already includes the address handled by the message, TAG does not need to be included in the payload of PDE. In directory-based two-level MOESI protocol, STATE is three bits. The number of bits allocated to the remaining DE components depends on the core count and is as follows: core count $= N$, SHARER VECTOR $= N$ bits, and OWNER $= \lceil \log_2 N \rceil$ bits.

Assuming a flit size of 128 bits, the payload of the PDE in systems with a core count of 118 (STATE (3 bits) + SHARER VECTOR (118 bits) + OWNER (7 bits) = 128 bits) or less can fit into a single flit. Message passing [33] will



Directory Entry (in HOME's directory slice)

TAG	STATE	SHARER	OWNER
0xb0	M	-	7 (RESP)

FIGURE 4. Procedure of prefetch DC in TCMP: (a) basic flow of PDE by prefetch DC, (b) west input port of router in p-REQ.

be preferred over directory-based cache coherency protocols when the core count exceeds 118. Therefore, we believe it is reasonable to consider the PDE's size as two flits (including a head flit). Meanwhile, a packet of cache line data requires five flits (including a head flit). Even expanding to the case where the PDE consists of 3 flits, the capable core count is up to 245. Regarding the core count of practical directory-based TCMP, PDE's superiority in size compared with cache lines is reasonable. Although a 128-bit flit is assumed to clarify the comparison of packet sizes, the comparison is valid without loss of generality because the underlying size of the cache line (generally 64B) and the required bit length for each component of the DE remain the same. Compared with a packet containing cache data that requires five flits, including a head flit, the PDE is much smaller. Therefore, the use of the PDE can widen the application range in terms of VC depth.

C. PDE-AWARE VC

Since the PDE can be smaller than the VC depth, empty buffers may be obtained in the VC used for NCDE. To efficiently utilize buffer space in VCs, PDE-aware VC is employed. PDE-aware VC enables the PDE to occupy only a portion of the buffers in VC rather than consuming the entire VC. Hence, a single VC can be simultaneously used for network caching and on-chip communication. In order to leverage the free space, the flits of the non-PDE packet can

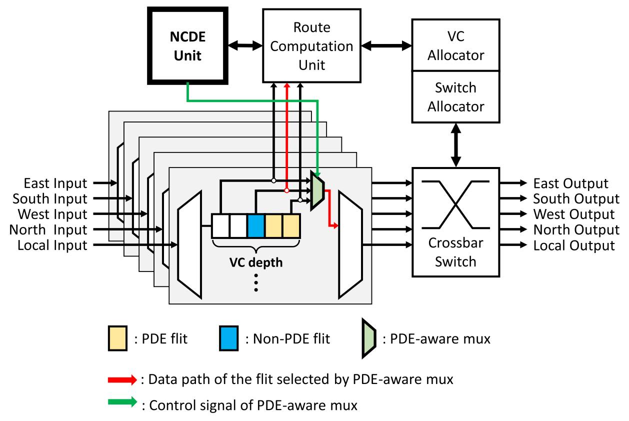


FIGURE 5. Router architecture including PDE-aware VC in each input port.

be accommodated in the VC where the PDE resides. An additional multiplexer, referred to as PDE-aware mux, on each VC is responsible for selecting the flit that will proceed to the next stage in the router pipeline. As indicated in Fig. 5, a PDE-aware VC demands a PDE-aware mux. The 128-bit wide 3 × 1 PDE-aware mux selects the flit to advance to the successive router pipeline stage. When a PDE enters the VC and is recognized by the NCDE unit, the PDE-aware mux's select signal is set to the next buffer in the VC. The NCDE unit generates the select signal of the PDE-aware mux. While working parallel with the route computation unit, the occupancy of each buffer of VC and the related status of the flits is forwarded to the NCDE unit. The number of PDEs residing in the VC is computed using the flit's forwarded information to generate PDE-aware mux's select signals. For example, with a five-flit VC depth and a two-flit size PDE, as shown in Fig. 5. The number of available buffers in VC to pass the flit is three. Because there is only one PDE in Fig. 5, the flit in the third buffer in the VC is routed through the PDE-aware mux. A one-flit size normal packet in the VC's third buffer can proceed with the router pipeline ahead of the PDE.

To guarantee the versatility of a single VC, the maximum number of PDEs per VC, referred to as *MAX_PDE*, is defined in (2). According to (2), the PDE does not occupy the entire VC. For example, 2-flit size PDE with 5-flit VC depth, *MAX_PDE* is 2. Because NCDE cannot operate if PDE size exceeds the VC depth, we exclude this case from (2). However, cache data transfer over the interconnection inevitable in a system with many core counts, such as TCMP, and thus NoC employs an input buffer that is amenable to the cache line size [34]. In general, the PDE size is smaller than the typical cache line size of 64B; therefore, it is not a typical case where the VC depth is smaller than the PDE size.

$$MAX\_PDE = \begin{cases} 1, & \text{if PDE size} == \text{VC depth} \\ \left\lceil \dfrac{\text{VC depth}}{\text{PDE size}} \right\rceil - 1, & \text{if PDE size} < \text{VC depth} \end{cases} \tag{2}$$

VC depth: maximum number flits for VC
PDE size: number of flits required for PDE

TABLE 3. Description of packet types based on NCDE flag bits.

NCDE flag	Description of packet types
Victim PDE	PDE for victim DC
DIR REQ	All coherence request message packets for directory lookup
Victim Discard	PDE discarded from the VC
Victim Hit	Former *Victim PDE* that has been hit
Prefetch PDE	PDE for prefetch DC
Prefetch REQ	GETS message packet
Prefetch Hit	Former *Prefetch PDE* that has been hit
Prefetch FWD	Pseudo FWD-GETS message packet for direct traverse to RESP
NONE	General packets that are unrelated with NCDE

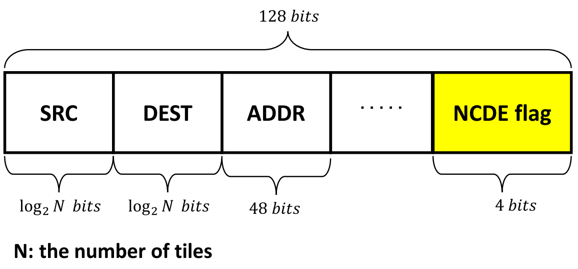


FIGURE 6. Head flit compositions including additional flag bit vector (NCDE flag); other meta data for routing are not descripted since they are unrelated to the working of NCDE.

D. OPERATION OF NCDE UNIT

Within an NoC router, the NCDE unit attached to the route computation unit is responsible for identifying the PDE entering the router, holding it in the VC, and controlling the overall process of victim DC and prefetch DC. Such operations of the NCDE unit are performed based on the NCDE flag of the head flit, as shown in Fig. 6. The route computation unit in each NoC router decodes the head flit to obtain data related to route decisions, such as SRC and DEST. By additionally allowing the extraction of the NCDE flag included in the head flit decoding process, it can be provided to the NCDE unit under the conventional operation of the router. To implement the NCDE unit, extracting the NCDE flag bits adds overhead to the route computation (RC) unit. The RC unit requires an additional hardware resource to transfer the NCDE flag bits and ADDR decoded from the head flit to the NCDE unit. Table 3 lists all types of packets based on the NCDE flag. These notations are used identically to refer to the NCDE flag and corresponding packets.

Similar to non-PDE packets, the PDE passes through the route computation (RC) stage when entering the router pipeline. A two-stage NoC router pipeline (stage 1: RC, stage 2: VA and SA) is considered, in which the NCDE unit works parallel to the RC. The NCDE unit performs PDE identification and store, hit, and discard operations based on the acquired NCDE flag. The store process of the PDE first identifies the NCDE flag as *Victim PDE* or *Prefetch PDE* ("1." in Fig. 7). If the incoming packet is a PDE, the head flit of the PDE is flushed from the router pipeline ("2." in Fig. 7). The PDE remains stalled in the VC by flushing the

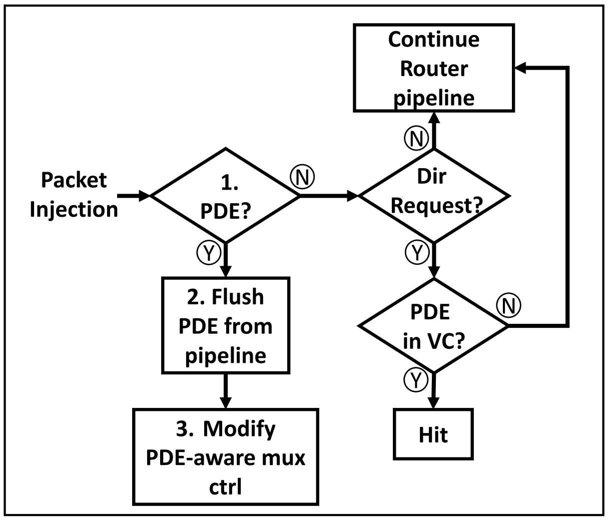


FIGURE 7. Flow chart of the operation of NCDE unit in identifying PDEs and holding them in the VC.

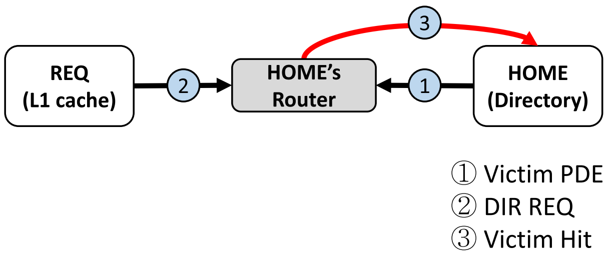


FIGURE 8. Victim DC procedure based on coherence message transactions.

head flit from the router pipeline register. The control signal of PDE-aware mux in that VC is set to pass the subsequent packets ("3." in Fig. 7). For a subsequent incoming packet, the flit of that packet is passed to the router pipeline. The details for the remaining parts of Fig. 7 differ in whether the PDE was generated for performing victim DC or prefetch DC. Therefore, the operation of the NCDE unit for the remaining parts is presented separately.

1) VICTIM DC

The victim DC scheme primarily focuses on decreasing the directory eviction invalidations, thereby reducing the L1 cache miss rate. That is, the victim DC scheme indirectly increases the size of the directory by postponing the eviction of the entries selected for replacement. Fig. 8 shows the process of the victim DC from the perspective of a message transaction. For ease of understanding, the notation of the coherence messages required by NCDE follows the notation of the corresponding packet. For example, "① *Victim PDE*" in this figure refers to the message that induces *Victim PDE* type packets. The notations in Fig. 9 also follows this.

• **Store**

When an eviction of DE occurs due to a conflict miss, and if SHARER VECTOR or OWNER of the DE is not a null vector, it is elected as the target of the victim DC. The entry selected as the target postpones the progress of DE eviction-induced invalidation, and sharer cache lines associated with that DE survives. The target DE from the directory slice is created as a packet with the NCDE flag of *Victim PDE*. Meanwhile, the processor's access toward the sharer cache lines, which has survived DE-eviction invalidation, is performed while oblivious to whether a *Victim PDE* was evicted. These survived cache lines can provide data to the processor, but a directory lookup is required for operations such as write and writeback. Just as the PDE remains in the directory slice, the coherence message for directory lookup is transmitted to the tile to which the *Victim PDE* initially belonged. In order to handle request messages for directory lookup, *Victim PDE* should be stored in the local input port VC of the router connected to the tile to which it originally belonged (① in Fig. 8).

• **Hit**

When a request to the address of *Victim PDE* occurs, including a request from the survived sharer cache lines, *Victim PDE* should return to the sparse directory slice to complete the required coherence action. The return of *Victim PDE* is referred to as hit, and for the NCDE unit to determine whether a hit occurred, all request message packets for directory lookup have NCDE flag set to *DIR REQ*. When *DIR REQ* enters the DEST router, which is attached to the tile of the corresponding directory entry, NCDE unit checks whether *Victim PDE* can hit the request based on the addresses of the packets (② in Fig. 8). If a *Victim PDE* exists whose address matches *DIR REQ*, NCDE unit modifies the NCDE flag of *Victim PDE* to *Victim Hit* and swaps SRC and DEST. Subsequently, the PDE-aware mux is passes *Victim Hit*. Because DEST was altered to the current router, *Victim Hit* traverses to the local output port and, returns to the directory slice (③ in Fig. 8).

• **Discard**

With the assistance of PDE-aware VC, *Victim PDE* is free of occupying the entire buffer space in a certain local input port VC. However, a new *Victim PDE* can be created while all VCs of the local input port store PDEs as much as *MAX_PDE*, and in this circumstance, the oldest existing *Victim PDE* is discarded by prioritizing to the newly created *Victim PDE*. Because all *Victim PDE*s are formerly selected as replacement victims, the *Victim PDE* to be discarded must resume the deferred invalidation to remove all private cache lines of the corresponding address. After changing the NCDE flag to *Victim Discard*, it returns to the tile by swapping of SRC and DEST. However, the invalidation operation is handled by the memory subsystem controller without returning to the directory slice. When the PDE size equals the VC depth, i.e., when a non-PDE packet cannot be handled concurrently with a PDE in a single VC, an additional policy of discard is triggered. In this case, the oldest PDE is removed, and the remaining steps of the discard proceed in the same way as when a new *Victim PDE* enters.

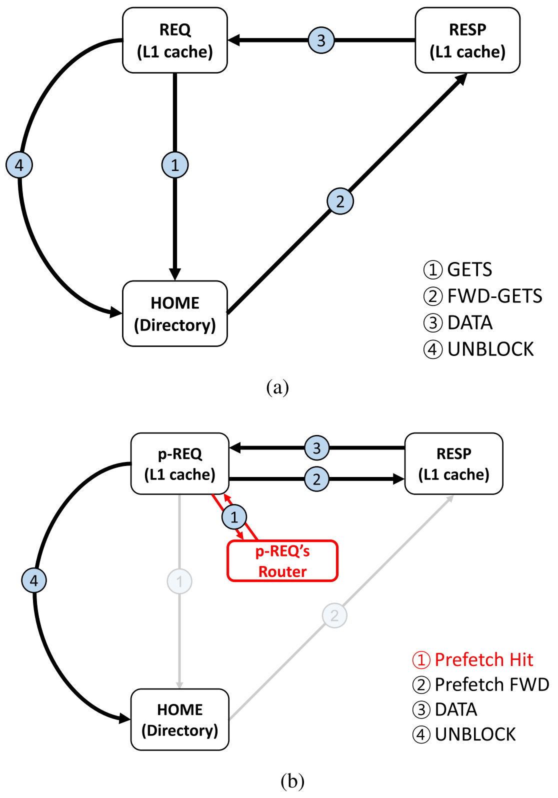


FIGURE 9. Procedure of prefetch DC based on coherence message transactions: (a) existing message transactions for handling GETS with C2C data transfer and, (b) proposed message transactions for handling GETS with C2C data transfer.

2) PREFETCH DC

Prefetch DC primarily aims to shorten the procedure of C2C data transfer, which requires a considerable amount of traversal through NoC because the detour for directory lookup must be included in the route destined toward the valid data [26], [27], [28]. Prefetching a DE to the predicted REQ (p-REQ) tile discards detours corresponding to forward message transactions in the total traverse for fetching up-to-date data. Simplifying C2C data transfer contributes to the shared data access expedition by curtailed L1 cache miss penalty. Fig. 9b shows the process of prefetch DC from the perspective of the message transaction.

In particular, prefetch DC provides coherence information to the invalidated cache lines that stem from the store operation. When a store operation occurs, sharer cache lines are invalidated prior to the store operation to maintain coherence. After all invalidations are completed, the store operation is performed, and the cache line acquires ownership. Meanwhile, with a subsequent load operation from the invalidated cache line, which is referred to as the load after store (LAS) sequence, GETS message for the load operation should first traverse to HOME and then to RESP, as shown in Fig. 9a. By providing coherence information in advance, the detour for directory lookup is excluded from the C2C transfer procedure, as shown in Fig. 9b, which results in a direct traverse to RESP.

• Store

The DE being opted for a target of prefetch DC converts to PDE with DEST set as p-REQ. In the LAS sequence, the tiles that contain the cache line invalidated by the store operation are p-REQ. *Prefetch PDE* is injected following the invalidation message packet, both destined for p-REQ. The process of entering the router of p-REQ and being stalled in the VC is identical to that of *Victim PDE*.

• Hit

The request type that *Prefetch PDE* can respond to is GETS. The coherence message transaction for C2C transfer in which prefetch DC is not applied is shown in Fig. 9a, and Fig. 9b shows the case in which prefetch DC excludes the detour for the directory lookup. GETS message packets' NCDE flag is marked by *Prefetch REQ*. As the first step to arrive at HOME, *Prefetch REQ* enters the router through a local input port. The NCDE unit checks whether a *Prefetch PDE* can process the GETS message based on the address of the packet. When an address match occurs, the corresponding PDE's NCDE flag is modified from *Prefetch PDE* to *Prefetch Hit* and enters to p-REQ. *Prefetch FWD* is generated according to the information of the RESP location contained in the *Prefetch Hit* in the memory subsystem controller of p-REQ. *Prefetch FWD* performs the very same function as FWD-GETS in Fig. 9a, except that SRC is REQ instead of HOME. *Prefetch FWD* induces RESP to send DATA to p-REQ.

• Discard

Similar to the victim DC, if there is no available space in the VC for a new PDE, one of the existing PDEs is fired. However, unlike *Victim PDE*, *Prefetch PDE* does not require any action after the discarding. Therefore, no NCDE flags are needed to indicate it. In the case of PDE size being equal to the VC depth, which requires consideration of conflict between PDE and non-PDE packets, is also identical to the victim DC. As in victim DC, discarding *Prefetch PDE* must consider the case where the PDE size and VC depth are identical. In this case, non-PDE packets cannot be handled simultaneously with PDE in a single VC, and an additional policy takes effect. That is, the oldest PDE is removed when a non-PDE packet is injected, while every VC is blocked with PDEs.

• Prefetch miss

Prefetch PDE can cause a race condition because the antecedent entry apparently resides in the directory slice. A race condition occurs when the ownership of the cache data changes, while *Prefetch PDE* waits for *Prefetch REQ*. Consequently, *Prefetch FWD*, which is conducted according to the operation process of "Hit," is transmitted to the outdated RESP (referred to as former RESP) rather than to the current owner. If cache line is retrieved from the former RESP or the valid cache data cannot be found, several message transactions are added in "Hit." Whether the visited RESP is the former RESP can be determined through the state of the cache line. The cache state of the RESP that operates the store in the LAS sequence is initially **M** (modified) state. Subsequently, the only state that can transition while maintaining ownership is **O** (owned). Therefore, if the cache state of the RESP where the *Prefetch FWD* arrives is not **M** or **O**, the operation of the prefetch DC is failed. Instead of reporting to p-REQ that prefetch DC has failed, the former RESP sends the message directly to HOME to guarantee a certain level of C2C transfer latency, even in the case of failure. In this process, the memory subsystem controller of the former RESP regenerates the GETS message based on the information of the *Prefetch FWD*. After GETS arrives at HOME, the process of fetching up-to-date data from the real RESP is the same as that in Fig. 9a.

TABLE 4. Simulation configuration.

Processor	x86-64 core (1 core per tile, total 16 tiles)
L1 Cache	8KB, 4-way, for each I-cache and D-cache
L2 Cache	128KB per core, 8-way, shared
Sparse directory	Number of DEs: 4096 8-way set associated
Cache line size	64B
Cache coherence	Directory-based MOESI protocol
NoC	4x4 2D mesh X-Y dimension-order routing algorithm Wormhole flow control Flit width: 128 bits VC depth: 5 flits 4 VCs per input port
Benchmarks	
PARSEC	Multi-threaded (16 threads)
blackscholes, bodytrack, canneal, ferret, fluidanimate, freqmine, swaptions, x264	

V. EVALUATION

A. SIMULATION SETUP

NCDE was evaluated using an event-driven gem5 simulator [35]. The gem5 simulator was modified to include the required NCDE functionalities, which can be categorized into the NoC and memory subsystem domains, corresponding to Garnet and Ruby in gem5. To implement NCDE-based router, Garnet has been customized as follows:

- In order to include the NCDE flag bits in the head flit of the packets, the existing "Flit" and "Network Interface" parts of Garnet source code have been adapted.
- We updated Garnet's "Input Unit" and "Virtual Channel" to function as PDE-aware VC.
- The NCDE unit, which is in charge of the PDE flow control, is implemented as a part of Garnet's "Input Unit" connected with the PDE-aware VC and the existing router computation unit.

The following changes are applied to Ruby to implement the sparse directory, DE eviction, and coherence message transactions needed by the victim/prefetch DC:

- We made some adjustments to the MOESI CMP directory protocol that Ruby offered to create a simulation environment that uses the directory-based MOESI cache coherence protocol.
- The MOESI CMP directory protocol's cache and directory controller is augmented with new events, actions,

and transitions to support DE eviction-induced invalidation.

- To implement additional coherence message transactions by victim/prefetch DC, events, actions, and transitions are added to the cache controller and directory controller in the MOESI CMP directory protocol.

Table 4 lists the system configurations used in the evaluation. The simulated architecture is structured as a TCMP composed of 16 identical tiles, each with a single ×86 core, private L1 cache, LLC slice, and sparse directory slice, which are connected by a 4×4 2D mesh NoC. The NoC-based TCMP baseline architecture for the simulations was constructed using publicly available representative models from industry and academia [1], [2], [3], [4]. Other additional details were derived from leading journals and conference proceedings [10], [20], [22] in the field to reflect the characteristics of TCMP appropriately and ensure the fairness of experimental results. TCMPs commonly have a two-level cache hierarchy due to the large core count, limited on-chip area, and associated implementation cost. In the case of the cache coherence protocol, Mittal [22] stated that the directory-based MOESI protocol is advantageous concerning the multithreading of TCMP, which can be confirmed again from the adoption of Xeon Phi [36]. VC depth was set to 5-flit size for a comparison with LSR-FD.

The number of DEs was matched to be identical to the total number of entries in the L1 caches aggregated over all TCMP cores. Meanwhile, the number of ways in the sparse directory are opted to those of the LLC, to mimic an eligible shared memory structure. This sparse directory configuration has been empirically established to perform sufficiently close to an unbounded sparse directory [12]. To be clear, this sparse directory configuration was not chosen to provide an undue advantage to the proposed idea. This is challenging because, with a sufficient number of DEs, the victim DC has less of an opportunity to improve on performance.

For a fair comparison, we established the baseline architecture as in previous studies, including the processor, cache, and most of the NoC configuration. Therefore, the number of DEs and the VC depth are considered the primary factors in NCDE performance. Since victim DC helps prevent DE eviction-related performance drops, its effectiveness is expected to grow as the number of DEs, which affects the occurrence of DE eviction, decreases. In particular, L1 cache miss rate can be lowered thanks to the effect of victim DC attenuating the adverse effects of DE eviction-induced invalidation. In the case of VC depth, it is related to the opportunity for NCDE to be performed. By utilizing VCs as the victim and prefetch buffers for DEs, NCDE expedites data access in NoC-based TCMPs. In this regard, the potential for NCDE performance increases with VC depth. NCDE also has the advantage of supporting a wide range of VC depths because it stores PDEs, which are more compact and adaptable than cache line data. Section V-D supplemented an additional analysis of the performance gain of NCDE with varying the number of DEs and VC depth.

To evaluate and analyze the improvement in memory subsystem performance while data are shared, we used multithreaded workloads from the PARSEC 3.0 benchmarks [6]. Linux kernel 4.19.83 version takes the role of thread scheduling, process management, etc. Running multithreaded workloads represented a typical situation for a TCMP, which leverages thread-level parallelism with large core counts. Although every workload from PARSEC can be multithreaded, the dependence on shared data access varies. Among the workloads shown in Table 4, *canneal*, *ferret*, and *x264* can be classified as communication-intensive workloads that demand a high level of data sharing and exchange, whereas *swaptions* has the lowest demand for communication [6]. For communication-intensive workloads, the portion of shared data access of the overall data access and occurrence of C2C data transfer is higher compared with other workloads. Because NCDE focuses on accelerating shared data access, we can expect NCDE to show superior performance gains in communication-intensive workloads.

B. PERFORMANCE IMPROVEMENT

We considered the following five architectures for evaluation:

- **w/o-NC**: A Vanilla architecture without any network caching method applied.
- **LSR-FD** [20]: An architecture which utilizes local input port's VCs as a victim buffer for evicted L1 cache lines.
- **NCDE-all**: The proposed architecture implementing both victim and prefetch DCs.
- **NCDE-v**: A decoupled architecture from NCDE-all with only victim DC implemented.
- **NCDE-p**: A decoupled architecture from NCDE-all with only prefetch DC implemented.

A TCMP without network caching (w/o-NC) is the basic architecture that does not use any network caching method, and LSR-FD is the counterpart architecture that utilizes local input port's VCs as a victim buffer for evicted L1 cache lines. LSR-FD is the most appropriate comparison target for evaluating NCDE-all, regarding its usage of VC, in addition to being one of the most recent and notable studies. NCDE-all is an architecture that implements all proposed ideas. In contrast, the NCDE-v and NCDE-p architectures also deploy PDE-aware VC and NCDE units like NCDE-all, but their functions are restricted to performing only victim DCs and prefetch DCs, respectively. Consequently, the memory subsystem controllers of NCDE-v and NCDE-p are designed to perform additional coherent message transactions derived from each victim DC and prefetch DC, respectively. With the configuration of the NoC in Table 4, the packet for cache data consists of 5 flits. Therefore, network caching for the L1 cache lines of LSR-FD consumes every buffer in a single VC. Meanwhile, *MAX_PDE* is 2; thus, NCDE-v, NCDE-p, and NCDE-all can utilize a single VC to hold two PDEs. PDE is advantageous in terms of latency from the perspective of efficient memory resource utilization and traffic management

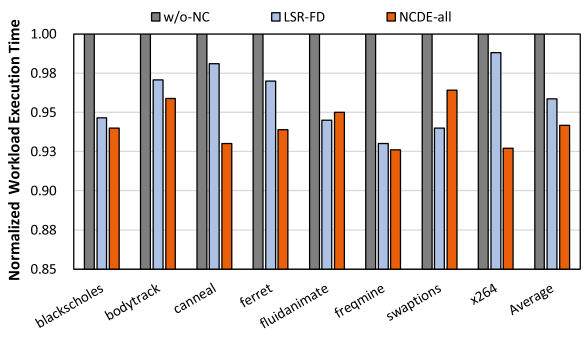


FIGURE 10. Comparison of workload execution time normalized to w/o-NC.

in NoC-based TCMP because it is much smaller than a cache line data. Simulations were performed with various VC depths, including a 5-flit size for a direct comparison with LSR-FD.

1) WORKLOAD EXECUTION TIME

We discuss the overall reduction in workload execution time to observe the effectiveness of NCDE-all. Normalized workload execution is the overall time consumed while running each workload. Fig. 10 depicts every result normalized with respect to w/o-NC, and the Average bar shows the geometric mean of the results for all workloads. The average workload execution time of NCDE-all was reduced by 5.82% compared with w/o-NC, and by 1.77% compared with LSR-FD. With each workload having differing characteristics, variations in their results are observed. Both LSR-FD and NCDE-all show significant improvements in *freqmine* of 7.01% and 7.40%, respectively. The ratio of read/write instructions to total instructions of *freqmine* is approximately 49.5%, which is the largest among the PARSEC benchmark suite [6]. The memory access instruction dominance feature of *freqmine* can be observed as a provision of abundant opportunity to achieve execution time reduction by enhancing the memory subsystem, which is focused on by both LSR-FD and NCDE-all.

Other remarkable results are from communication-intensive workloads, which are expected to show the most significant gap between LSR-FD and NCDE-all. Evidently, NCDE-all is superior to LSR-FD based on the results from *canneal*, *ferret*, and *x264*. Because LSR-FD achieves data access acceleration by replying to L1 cache's re-reference to evicted cache lines, sharing those evicted lines hinders the effectiveness of LSR-FD's network caching. The communication-intensive behavior of these workloads, which accompanies several shared cache lines, violates the possibility of LSR-FD benefiting. Meanwhile, NCDE-all can leverage these characteristics, which can be seen in its reduction of the workload execution time by 5.20%, 3.22%, and 6.19% compared with LSR-FD and 7.00%, 6.12%, and 7.30%

TABLE 5. Comparison of workload execution time with the PARSEC 3.0 benchmark suite between w/o-NC, LSR-FD, and NCDE-all (in *ms* unit).

Simulation architecture	w/o-NC	LSR-FD	NCDE-all
blackscholes	36.96	34.98	34.74
bodytrack	372.62	361.70	357.26
canneal	143.01	140.29	133.00
ferret	1,110.39	1,077.08	1,042.44
fluidanimate	224.20	211.85	212.99
freqmine	1,388.69	1,291.49	1,285.93
swaptions	435.75	409.60	420.06
x264	5,127.78	5,066.25	4,753.71
Average	1,104.93	1,074.15	1,030.02

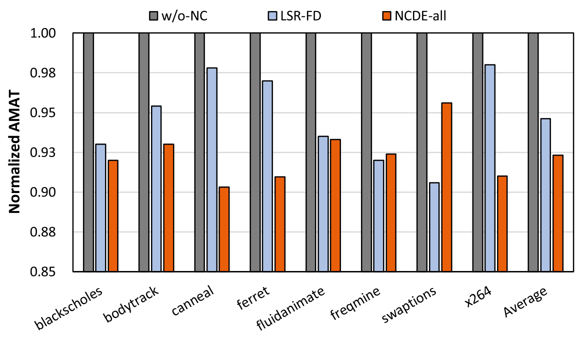


FIGURE 11. Comparison of average memory access time normalized to w/o-NC.

compared with w/o-NC while running *canneal*, *ferret*, and *x264*, respectively. LSR-FD outperforms NCDE-all while running *swaptions*. The cause of this performance gap is the exact opposite of the reason why NCDE-all is superior to LSR-FD with communication-intensive workloads. In addition to the normalized data presented in Fig. 10, Table 5 provides the exact values for the execution time consumed in w/o-NC, LSR-FD, and NCDE-all with selected benchmarks from the PARSEC 3.0 suite. The results are represented in *ms* unit.

The improvement in the workload execution time must have been affected by the reduction in data access latency because both LSR-FD and NCDE-all aim to enhance the performance of the memory subsystem. Therefore, in the following, we analyze and compare NCDE-all with LSR-FD with respect to data access latency.

2) AVERAGE MEMORY ACCESS TIME

The overall data access latency can be represented by the average memory access time (AMAT) metric. It adequately represents the overall performance enhancement of the memory subsystem. NCDE focuses on reducing the L1 cache miss rate and penalty, which are the dominant terms in AMAT, and its impact is demonstrated in Fig. 11. The effectiveness of the congruence of the victim and prefetch DCs is reflected by the reduced AMAT of the NCDE-all architecture. NCDE-all outperforms LSR-FD by 2.45% and w/o-NC by 7.69% in the geometric mean. Among the benchmarks from PARSEC, the results from *canneal*, *ferret*, and

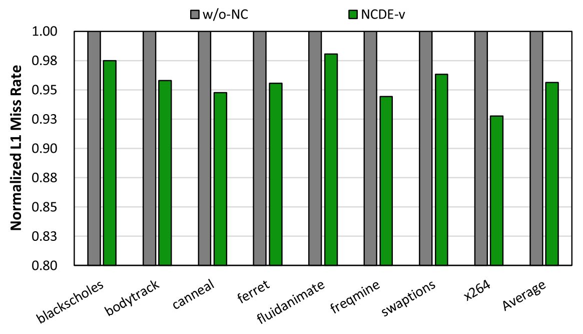


FIGURE 12. Comparison of L1 cache miss rate normalized to w/o-NC.

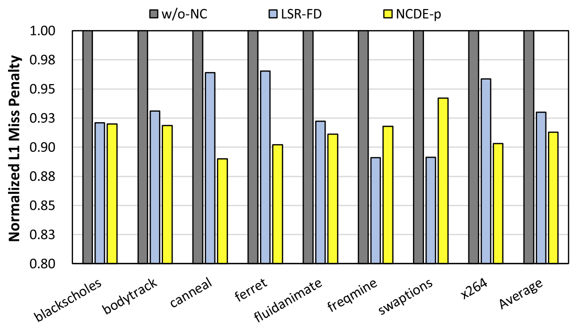


FIGURE 13. Comparison of L1 cache miss penalty normalized to w/o-NC.

x264 show a superior gap between LSR-FD and NCDE-all. The communication-intensive behavior of the workloads diminishes the opportunity for LSR-FD to benefit, whereas NCDE-all can leverage it. To analyze the root cause of the improvement that NCDE-all accomplished, we performed simulations with NCDE-v and NCDE-p architectures, which implemented solely only victim DC and prefetch DC, respectively. Since each method benefits from a reduced L1 miss rate and penalty, the corresponding results are provided.

3) L1 CACHE MISS RATE

The L1 cache miss rate is determined by the average miss rate in each tile's L1 cache, including both the instruction and data caches. Because directory address allocation and request for it have no relationship, averaging over all tiles reflects the L1 miss rate without bias. The L1 cache miss rate directly represents the effectiveness of the victim DC, because the DE eviction-induced invalidations of L1 cache lines are reduced. Fig. 12 depicts the L1 cache miss rate normalized with respect to the w/o-NC architecture. NCDE-v was 4.37% superior to w/o-NC with respect in terms of the geometric mean miss rate. LSR-FD was not evaluated because it targets victim caching, which is an after-treatment for cache misses. Meanwhile, NCDE-v targets victim buffering for DEs, which can contribute to reducing the number of cache misses.

4) L1 CACHE MISS PENALTY

The L1 cache miss penalty was derived from the average of the total L1 cache miss penalties. The L1 cache miss penalty directly represents the effectiveness of the prefetch DC because it curtails detours for acquiring coherence information during C2C data transfers. Fig. 13 depicts the L1 cache miss penalty normalized with respect to the w/o-NC architecture. NCDE-p outperforms LSR-FD by 1.84% and w/o-NC by 8.70% in terms of the geometric mean. For the same reason as AMAT, the results for *canneal*, *ferret*, and *x264* show a more significant performance gain compared with the others. This reflects that our method is effective in mitigating problems stemming from shared data accesses. In Fig. 11 and 13, NCDE-all and NCDE-p show 2.45% and 1.84% superior to LSR-FD in terms of AMAT and L1 cache miss penalty, respectively. The lower performance gain of NCDE-p in comparison to NCDE-all can be attributed to the absence of the victim DC function. As shown in Fig. 12, NCDE-v reduces L1 cache miss rate, which LSR-FD nor NCDE-p cannot affect. Through the evaluation results from NCDE-all, NCDE-v and NCDE-p, we can ensure that victim DC and prefetch DC are complementary.

The evaluation of the L1 miss penalty confirms that shortening the C2C data transfer latency using prefetch DC affects the data access latency. In a NoC-based TCMP, the message transmission latency tends to increase with the number of cores, which can be considered an opportunity for a prefetch DC. Therefore, we conducted an additional experiment to determine the impact of NCDE-all as the number of cores increases. Therefore, we conducted additional experiment to determine the impact of NCDE-all as the number of cores increases.

C. SCALABILITY

To prove the scalability of the NCDE-all, results regarding the performance improvement are provided in a more extensive scale system. There is a core count restriction to apply the shared/distributed directory with 2D mesh based NoC in the gem5 simulator: the number (core count) must be both a power of two and a square. Therefore, the available core counts following 16 are 64 and 256. However, systems with a core count of 256 are generally configured as a cluster, which is not the focus of our work. The core count of 64 is inadequate to fit our former evaluation model due to the limitation on thread spawning. As an alternative, the Ruby random generator provided by gem5 was used for evaluating NCDE-all in a system with a core count of 64. The Ruby random generator raises random memory access with a burst access pattern; therefore, it effectively reflects the case in which the memory subsystem is overloaded. The architecture we configured for a Ruby random generator with 16 and 64 core counts are referred to as 16 tiles and 64 tiles, respectively. The tiles of both architectures consisted of a single core, as in the configuration above. We generated 1,000,000 random memory accesses for 16 tiles and 4,000,000 for 64 tiles.

The size of the L1 cache, LLC slice, and directory slice belonging to a single tile of the 64 tiles architecture is the same as that of the 16 tiles architecture. An identical

TABLE 6. Comparison of workload execution time (in *ms* unit) between w/o-NC and NCDE-all with varying number of DEs.

N_R (total # of DEs / total # of L1 cache lines)	N_R = 1		N_R = 1/2		N_R = 1/4	
Simulation architecture	w/o-NC	NCDE-all	w/o-NC	NCDE-all	w/o-NC	NCDE-all
blackscholes	36.96	34.74	55.35	51.39	60.39	55.24
bodytrack	372.62	357.26	402.32	379.94	457.82	418.65
canneal	143.01	133.00	270.92	251.49	315.55	289.65
ferret	1,110.39	1,042.44	1,180.21	1,097.76	1,340.09	1,230.84
fluidanimate	224.20	212.99	432.98	396.18	681.62	607.87
freqmine	1,388.69	1,285.93	1,519.92	1,387.69	1,838.97	1,667.71
swaptions	435.75	420.06	598.41	556.59	1,052.42	954.87
x264	5,127.78	4,753.71	5,122.94	4,636.26	5,123.42	4,547.03
Average	1,104.93	1,030.02	1,197.88	1,094.66	1,358.78	1,221.48

TABLE 7. Comparison of the occurrence of DE eviction-induced invalidations (in million unit) with the PARSEC 3.0 benchmark suite between w/o-NC and NCDE-all with varying the number of DEs.

N_R (total # of DEs / total # of L1 cache lines)	N_R = 1		N_R = 1/2		N_R = 1/4	
Simulation architecture	w/o-NC	NCDE-all	w/o-NC	NCDE-all	w/o-NC	NCDE-all
blackscholes	0.12	0.08	1.27	1.00	2.11	1.62
bodytrack	1.48	0.91	10.62	8.19	31.65	28.39
canneal	4.24	3.67	13.53	10.59	32.06	28.65
ferret	2.85	1.99	10.96	9.41	51.14	38.35
fluidanimate	4.50	2.37	35.58	28.26	73.26	61.99
freqmine	13.84	8.36	60.08	50.41	152.02	137.63
swaptions	11.85	7.61	61.15	47.51	169.27	145.32
x264	8.08	6.22	12.62	10.72	22.63	20.36
Average	5.87	3.90	25.73	20.76	66.77	57.79

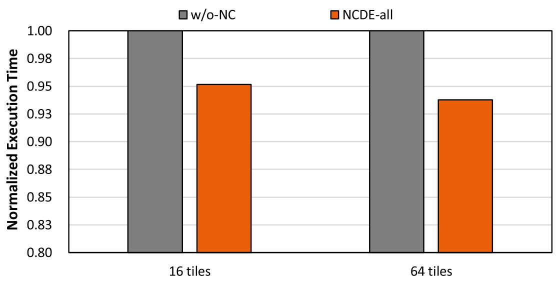


FIGURE 14. Variation of execution time of NCDE-all with 16 and 64 tiles.

configuration, from the perspective of a single tile, maintains the ratio of the hardware budget allocated to the memory subsystem.

The higher the core count, the greater the expected influence of NCDE-all because the performance gain owing to the operation of the prefetch DC is more advanced. Specifically, the larger the core count, the higher the probability that the hop count to go from REQ to HOME increases. We used the execution time to provide the performance gain of the NCDE-all while running memory access operations. Fig. 14 depicts the execution time normalized with respect to the w/o-NC architecture. NCDE-all achieved a reduction in execution time by 4.84% and 6.21% for 16 tiles and 64 tiles, respectively.

D. SENSITIVITY

For a fair comparison with related studies, the processor, cache, and most NoC configurations were set equivalently. Given the focus of the study, the number of DEs and VC depth are therefore considered to be the most influential factors on NCDE-all performance.

1) NUMBER OF DEs

First, an experiment was conducted to assess the data access performance by varying the number of DEs, which affects the occurrence of DE evictions. The number of DEs is denoted by N_R, where N_R is the ratio of the number of DEs to the total number of lines in the L1 cache across all tiles. N_R varies between 1, 1/2, and 1/4, which correspond to 4096, 2048, and 1024 DEs, and other parameters are listed in Table 4. The performance gain achieved through NCDE-all can be evaluated by workload execution time. Fig. 15 presents the normalized average execution time running the selected PARSEC 3.0 benchmark suite (in Table 4) on w/o-NC and NCDE-all with different N_R values of 1, 1/2, and 1/4. NCDE-all showed 5.82%, 7.58%, and 9.27% lower average execution time than w/o-NC when N_R is 1, 1/2, and 1/4, respectively. Table 6 provides the exact values (in *ms* unit) for the execution time results shown in Fig. 15.

The average L1 cache miss rates are depicted in Fig. 16, where it can be seen that the performance gain of NCDE-all increases as N_R decreases. NCDE-all achieves 4.37%, 7.59%, and 9.79% lower L1 cache miss rate with respect to w/o-NC when N_R is 1, 1/2, and 1/4, respectively. Table 7 provides exact values for the number of DE eviction-induced invalidations for the benchmarks with different N_R cases in million unit. Victim DC's design goal is to alleviate the

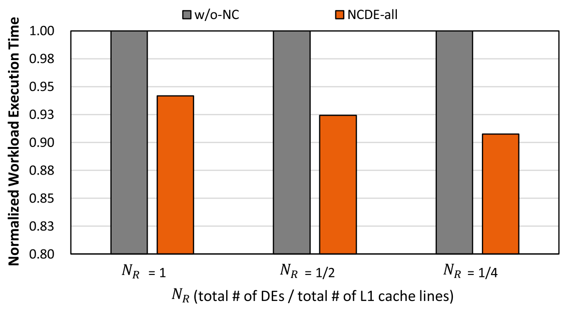


FIGURE 15. Normalized workload execution time with varying number of DEs.

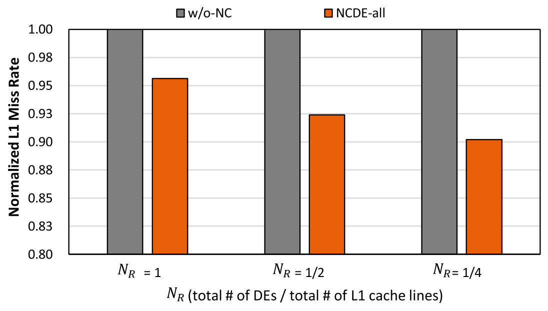


FIGURE 16. Normalized L1 cache miss rates with varying number of DEs.

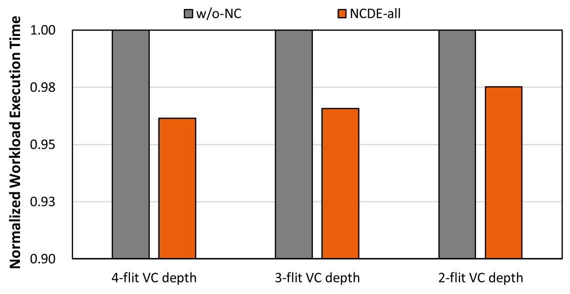


FIGURE 17. Sensitivity of NCDE with VC depth.

problem caused by DE eviction. Therefore, it is expected that higher performance improvement can be achieved when DE eviction is frequent, and this can be confirmed through simulation result analysis of execution time, L1 cache miss rates, and the occurrence of DE eviction-induced invalidation. Table 7 shows that the difference between NCDE-all and w/o-NC architecture increases as N_R shrinks. This disparity indicates that the effectiveness of NCDE-all escalates when DE eviction is frequent. The coinciding tendency of performance gain can also be observed in the results of L1 cache miss rate and workload execution time (in Fig. 15 and 16), which are triggered by the reduction of DE eviction-induced invalidation.

TABLE 8. Area overhead of NCDE-all.

Area (um^2)	w/o-NCDE	NCDE-all
RC / VA / SA / X-bar	210,348	210,965
Input buffer (VC)	205,464	207,071
NCDE unit	-	4,248
Total Area	415,815	422,284

2) VC DEPTH

For all results discussed, 5-flit VC depth was considered, as shown in Table 4. To show that our method has a wide applicable range in terms of VC depth, the impact of varying the VC depth on performance gain is provided. The VC depth condition varies from 2, 3, and 4, and architecture models with each condition are referred to as the 2-flit VC depth, 3-flit VC depth, and 4-flit VC depth, respectively. LSR-FD was not analyzed because applying it to such an NoC is impossible; meanwhile, NCDE-all is applicable. The workloads from Table 4 were used, and the geometric mean of the workload execution times was chosen to present the performance gain of the NCDE-all. Fig. 17 depicts the average workload execution time normalized with respect to w/o-NC. NCDE-all reduced the execution time by 3.86%, 3.43%, and 2.48% with VC depths of 4, 3, and 2 flits, respectively, compared with w/o-NC. In decreasing the VC depth, the opportunity for victim and prefetch DCs reduces, which can be observed from the diminishing results. The result from the 2-flit VC depth is notable, which shows only a 2.48% reduction. Because PDE-aware VC is unavailable for use at a 2-flit VC depth, the PDE occupies the entire buffer of the VC it uses. Therefore, performing victim and prefetch DCs can be considered as reducing the number of VCs, which can slow down inter-tile communication.

E. AREA OVERHEAD ANALYSIS

Three additional bits in the header flit are required to support the operation of the NCDE unit. However, even including metadata for routing, data of the head flit does not exceed 128-bit of flit size. Therefore, NCDE flag bits can be accommodated in head flit without an additional increment in size for the flit or channel. We synthesized router architecture with NCDE-all using SAED 32nm cells [37] and Synopsys Design Compiler [38] to demonstrate area overhead induced by implementing NCDE. The logic synthesis results are represented in Table 8. w/o-NCDE is a conventional router to which NCDE is not applied, and NCDE-all is a router in which NCDE is implemented. The RC unit requires an additional hardware resource to transfer the NCDE flags, and ADDR decoded from the head flit to the NCDE unit, resulting in an extra 617 um^2, or 0.29% area overhead. Each PDE-aware VC requires a 128-bit 3×1 multiplexer and extra wires to select the flit for the successive pipeline stage. As a result, the size of the entire input buffer grows, leading to a 0.78% area overhead compared with the input buffer of the w/o-NCDE. Consequently, implementing PDE-aware VC and NCDE unit add a combinational area overhead on NoC router of about 1.56% compared to the w/o-NC, which is trivial amount compared with the performance gain achieved as shown above.

VI. CONCLUSION

This study proposed a novel network caching method that utilizes VCs as an opportunistic buffer for DEs. NCDE alleviates problems stemming from shared data accesses, which is strongly related to the multithreading of the TCMP architecture. By utilizing VCs as both victim and prefetch buffers for DEs, NCDE explores the reduction of DE eviction-induced invalidations and simplification of C2C data transfers. As the evaluations demonstrate, NCDE achieves a lower data access latency with minimal router area overhead. Simulations with varying numbers of DEs, VC depths, and core counts demonstrate that NCDE has a wide applicability range in terms of VC depth and has the potential to reduce a greater amount of data access latency in TCMPs, which have a limited number of DEs and a large core count.

REFERENCES

[1] 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, Mar./Apr. 2016.

[2] 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 *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 25–36.

[3] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[4] T. Yoshida, "Fujitsu high performance CPU for the post-*k* computer," in *Proc. Hot Chips*, vol. 30, 2018, p. 22.

[5] J. M. Sabarimuthu and T. G. Venkatesh, "Analytical miss rate calculation of L2 cache from the RD profile of L1 cache," *IEEE Trans. Comput.*, vol. 67, no. 1, pp. 9–15, Jan. 2018.

[6] C. Bienia, S. Kumar, and K. Li, "PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors," in *Proc. IEEE Int. Symp. Workload Characterization*, Oct. 2008, pp. 47–56.

[7] V. Mekkat, A. Holey, P.-C. Yew, and A. Zhai, "Building expressive, area-efficient coherence directories," in *Proc. 22nd Int. Conf. Parallel Architectures Compilation Techn.*, Oct. 2013, pp. 299–308.

[8] D. Sanchez and C. Kozyrakis, "SCD: A scalable coherence directory with flexible sharer set encoding," in *Proc. IEEE Int. Symp. High-Perform. Comp Archit.*, Feb. 2012, pp. 1–12.

[9] H. Zhao, A. Shriraman, and S. Dwarkadas, "SPACE: Sharing pattern-based directory coherence for multicore scalability," in *Proc. 19th Int. Conf. Parallel Architectures Compilation Techn. (PACT)*, 2010, pp. 135–146.

[10] W. Shu and N.-F. Tzeng, "NUDA: Non-uniform directory architecture for scalable chip multiprocessors," *IEEE Trans. Comput.*, vol. 67, no. 5, pp. 740–747, May 2018.

[11] S. Shukla and M. Chaudhuri, "Tiny directory: Efficient shared memory in many-core systems with ultra-low-overhead coherence tracking," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 205–216.

[12] M. Chaudhuri, "Zero directory eviction victim: Unbounded coherence directory and core cache isolation," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 277–290.

[13] C. Fensch and M. Cintra, "An OS-based alternative to full hardware coherence on tiled CMPs," in *Proc. IEEE 14th Int. Symp. High Perform. Comput. Archit.*, Feb. 2008, pp. 355–366.

[14] K. S. Shim, M. Lis, O. Khan, and S. Devadas, "The execution migration machine: Directoryless shared-memory architecture," *Computer*, vol. 48, no. 9, pp. 50–59, Sep. 2015.

[15] M. Davari, A. Ros, E. Hagersten, and S. Kaxiras, "An efficient, self-contained, on-chip directory: DIR1-SISD," in *Proc. Int. Conf. Parallel Archit. Compilation (PACT)*, Oct. 2015, pp. 317–330.

[16] S. Kaxiras and G. Keramidas, "SARC coherence: Scaling directory cache coherence in performance and power," *IEEE Micro*, vol. 30, no. 5, pp. 54–65, Sep. 2010.

[17] H. E. Mizrahi, J.-L. Baer, E. D. Lazowska, and J. ZahorJan, "Introducing memory into the switch elements of multiprocessor interconnection networks," in *Proc. 16th Annu. Int. Symp. Comput. Archit.*, 1989, pp. 158–166.

[18] N. Eisley, L.-S. Peh, and L. Shang, "In-network cache coherence," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2006, pp. 321–332.

[19] J. Augustine, R. K. J. Jose, and M. Mutyam, "Router buffer caching for managing shared cache blocks in tiled multi-core processors," in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, Oct. 2020, pp. 239–246.

[20] A. Das, A. Kumar, J. Jose, and M. Palesi, "Opportunistic caching in NoC: Exploring ways to reduce miss penalty," *IEEE Trans. Comput.*, vol. 70, no. 6, pp. 892–905, Jun. 2021.

[21] A. Das, A. Kumar, J. Jose, and M. Palesi, "Exploiting on-chip routers to store dirty cache blocks in tiled chip multi-processors," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2020, pp. 147–152.

[22] S. Mittal, "A survey on evaluating and optimizing performance of Intel Xeon phi," *Concurrency Comput., Pract. Exp.*, vol. 32, no. 19, Oct. 2020, Art. no. e5742.

[23] A. Ros, M. E. Acacio, and J. M. Garcıa, "Cache coherence protocols for many-core CMPs," in *Parallel and Distributed Computing*. 2010, p. 93.

[24] M. Ferdman, P. Lotfi-Kamran, K. Balet, and B. Falsafi, "Cuckoo directory: A scalable directory for many-core systems," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, Feb. 2011, pp. 169–180.

[25] D. J. Sorin, M. D. Hill, and D. A. Wood, "A primer on memory consistency and cache coherence," *Synth. Lectures Comput. Archit.*, vol. 6, no. 3, pp. 1–212, 2011.

[26] R. Iyer, L. N. Bhuyan, and A. Nanda, "Using switch directories to speed up cache-to-cache transfers in CC-NUMA multiprocessors," in *Proc. 14th Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2000, pp. 721–728.

[27] C. Hristea, D. Lenoski, and J. Keen, "Measuring memory hierarchy performance of cache-coherent multiprocessors using micro benchmarks," in *Proc. ACM/IEEE Conf. Supercomput. (CDROM)*, 1997, pp. 1–12.

[28] M. R. Marty and M. D. Hill, "Virtual hierarchies to support server consolidation," in *Proc. 34th Annu. Int. Symp. Comput. Archit.*, 2007, pp. 46–56.

[29] M. E. Acacio, J. González, J. M. García, and J. Duato, "A novel approach to reduce L2 miss latency in shared-memory multiprocessors," in *Proc. 16th Int. Parallel Distrib. Process. Symp.*, 2002, p. 8.

[30] L. A. Barroso, K. Gharachorloo, and E. Bugnion, "Memory system characterization of commercial workloads," in *Proc. 25th Annu. Int. Symp. Comput. Archit.*, 1998, pp. 3–14.

[31] M. Galles, "Spider: A high-speed network interconnect," *IEEE Micro*, vol. 17, no. 1, pp. 34–39, Jan./Feb. 1997.

[32] A. Ejaz, V. Papaefstathiou, and I. Sourdis, "FreewayNoC: A DDR NoC with pipeline bypassing," in *Proc. 12th IEEE/ACM Int. Symp. Netw.-on-Chip (NOCS)*, Oct. 2018, pp. 1–8.

[33] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Comput.*, vol. 22, no. 6, pp. 789–828, 1996.

[34] L. Wang, P. Kumar, K. H. Yum, and E. J. Kim, "APCR: An adaptive physical channel regulator for on-chip interconnects," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, 2012, pp. 87–96.

[35] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[36] R. Rahman, *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*. Springer, 2013.
[37] R. Goldman, K. Bartleson, T. Wood, K. Kranen, C. Cao, V. Melikyan, and G. Markosyan, "Synopsys' open educational design kit: Capabilities, deployment and future," in *Proc. IEEE Int. Conf. Microelectron. Syst. Educ.*, Jul. 2009, pp. 20–24.
[38] P. Kurup and T. Abbasi, *Logic Synthesis Using Synopsys*. Springer, 2012.

JAE EUN SHIM (Student Member, IEEE) received the B.S. degree in electronic and electrical engineering from Sungkyunkwan University, Suwon, South Korea, in 2021, where he is currently pursuing the M.S. degree in artificial intelligence. His current research interests include machine learning and computer architecture.

MINGU KANG (Student Member, IEEE) received the B.S. degree in electronic and electrical engineering from Soonchunhyang University, Asan, South Korea, in 2018, where he is currently pursuing the M.S. and Ph.D. degrees in electrical and computer engineering. His research interests include machine learning and computer architecture.

TAE HEE HAN (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 1992, 1994, and 1999, respectively. From 1999 to 2006, he was with the Telecom Research and Development Center, Samsung Electronics, where he developed 3G wireless, mobile TV, and mobile WiMax handset chipsets. Since March 2008, he has been with Sungkyunkwan University, Suwon, South Korea, as a Professor. From 2011 to 2013, he was worked as a full-time Advisor on system ICs for the Korean Government. His current research interests include SoC/chiplet architectures for AI, advanced memory architecture, network-on-chip, and system-level design methodologies.