IET Computers & Digital Techniques

Special Issue: Energy-efficient Computing for Embedded and IoT Devices

ECAP: energy-efficient caching for prefetch blocks in tiled chip multiprocessors

ISSN 1751-8601
Received on 31st January 2019
Accepted on 10th April 2019
E-First on 8th August 2019
doi: 10.1049/iet-cdt.2019.0035
www.ietdl.org

Dipika Deb[1] ✉, *John Jose*[1], *Maurizio Palesi*[2]

[1]*MARS Research Laboratory, Department of Computer Science and Engineering, Indian Institute of Technology, Guwahati, India*
[2]*Department of Electrical, Electronic and Computer Engineering, University of Catania, Catania, Italy*
✉ *E-mail: d.dipika@iitg.ac.in*

Abstract: With the increase in processing cores performance have increased, but energy consumption and memory access latency have become a crucial factor in determining system performance. In tiled chip multiprocessor, tiles are interconnected using a network and different application runs in different tiles. Non-uniform load distribution of applications results in varying L1 cache usage pattern. Application with larger memory footprint uses most of its L1 cache. Prefetching on top of such application may cause cache pollution by evicting useful demand blocks from the cache. This generates further cache misses which increases the network traffic. Therefore, an inefficient prefetch block placement strategy may result in generating more traffic that may increase congestion and power consumption in the network. This also dampens the packet movement rate which increases miss penalty at the cores thereby affecting Average Memory Access Time (AMAT). The authors propose an energy-efficient caching strategy for prefetch blocks, ECAP. It uses the less used cache set of nearby tiles running light applications as virtual cache memories for the tiles running high applications to place the prefetch blocks. ECAP reduces AMAT, router and link power in NoC by 23.54%, 14.42%, and 27%, respectively as compared to the conventional prefetch placement technique.

1 Introduction

With the advancement toward green computing, it is important to design multicore architectures that can provide efficient power savings and enhanced system performance with minimal hardware overhead. Advancements of complementary metal–oxide–semiconductor technology enabled an increase of computational units to be placed in the same chip area known as chip multiprocessor (CMP). Although transistor scaling has enabled an increase in computational units but the power consumed by the other on-chip units is no more negligible. Hence, the design choice has now been shifting its focus on developing energy-aware architectures satisfying the needs of modern data-intensive applications.

For processors such as Intel Xeon-Phi [1] and Tilera TILEPro [2] architectures, the processors are organised as tiles and each tile consists of an out-of-order superscalar processor (core), a private L1 cache and a slice of shared L2 cache. Such architectures are also called as tiled CMP (TCMP) [3]. The L2 cache is inclusive, and it is equally divided among all the tiles. Each such slice of L2 cache is called a bank. The tiles are arranged in a mesh topology and interconnected by an underlying network on chip (NoC) [4, 5] as shown in Fig. 1. A conventional NoC consists of routers and bi-directional links that communicate cache block request (cache misses) and cache block reply (data) as packets. The packets are further divided into multiple flits with each flit size equal to the inter-router link bandwidth. All cache miss request and replies are carried to the destination tile as per the static non uniform cache access (SNUCA) bank mapping policy [6].

As shown in Fig. 2, a different application in TCMP runs at different cores with individual data requirements. The cache block demand and cache access pattern of the application varies across the cores. Such a distributed system provides high-throughput computing [3]. However, in these systems, data communication and on-chip interconnect have become a costly affair in terms of energy consumption [7, 8]. Therefore in TCMPs, a high-throughput and energy-efficient communication backbone is a must to achieve a better system performance.

All tile-to-tile communication in TCMP travels through the NoC as either cache miss request, cache block reply, or coherence packets. Miss penalty of a cache miss can be estimated in traditional CMPs having monolithic L2 cache. However, in TCMPs, miss penalty consists of memory access latency of L2 cache bank and the round trip network latency between the respective tiles. The round trip network latency in such a scenario depends on the network congestion. Hence in TCMP, the network contributes in miss penalties occurring during cache misses which in turn regulates the average memory access time (AMAT).

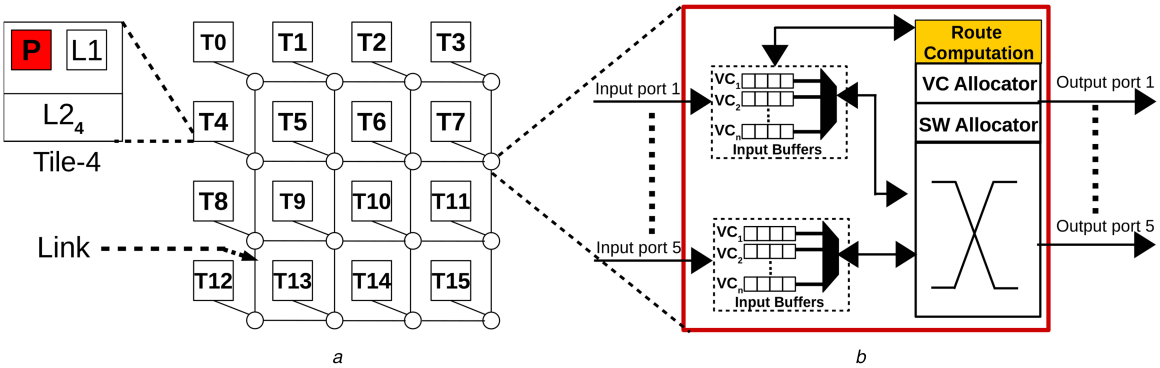


Fig. 1 *Tiles are arranged in a mesh topology and interconnected by an underlying NoC*
(a) 4 × 4 TCMP, ***(b)*** Router in NoC

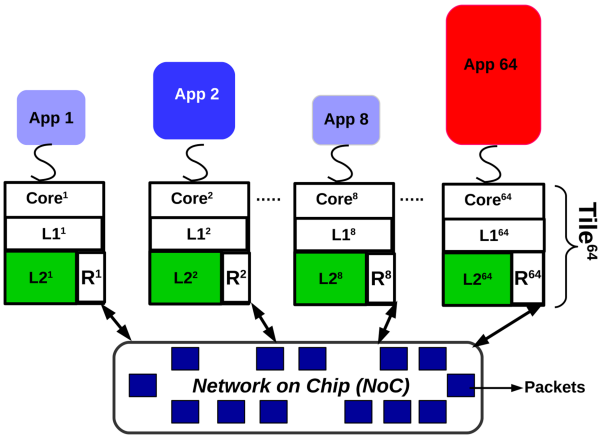


Fig. 2 *Different applications running at different cores in TCMP*

Reduction of AMAT is imperative in achieving high throughput. This is possible if the underlying NoC provides a low-latency communication backbone for packet movement. Since the NoC characteristic changes dynamically based on the network traffic status, prefetching comes handy in this regard.

Prefetching is a technique that helps in hiding the long memory access latency and the underlying network latency by speculatively bringing cache blocks for future use [9–16]. A prefetch engine monitors the run-time activities of cache access pattern by the core. On observing a pattern, the prefetch engine fetches the desired block into the cache for later use. Since in TCMP, the L2 cache is distributed among all the tiles, the cache access pattern is also distributed across them. Therefore, in TCMP, the prefetch engine is trained using the L1 cache access patterns which are local to each tile.

In TCMP, an enormous amount of energy and power is consumed by the underlying NoC, after the cores. NoC consumes around 28% of each tile power in Intel-Teraflop chip [17] and 36% in MIT-RAW chip [18]. NoC being the bottleneck in inter-tile communication, the energy and power consumed by NoC has become a major concern. The power consumption in NoC can be classified into two categories: static power and dynamic power. Static power is mostly consumed by buffers [virtual channels (VCs)] in the router as shown in Fig. 1*b*. On the other hand, dynamic power is consumed due to switching activity per cycle which occurs during transmitting a flit from an upstream router to a downstream router. Hence, reducing the number of packets will result in reducing network congestion as well as reduction in the static and dynamic power consumption in NoC.

In a prefetch-enabled cache, there are two types of blocks: demand and prefetch blocks. Demand blocks are fetched when the processor requests for them and prefetch blocks are fetched ahead of its use. In conventional prefetching, the prefetch block is placed in the local cache (L1 or L2) of the requesting core. In this paper, we call such prefetch block placement as source core placement (SCP). In SCP, placing a prefetch block may result in evicting a useful demand block from the cache location where it is placed. Thus, the speculative nature of a prefetcher can pollute the cache which triggers frequent cache block replacements [10]. In TCMP, cache replacement increases the network packets which in turn increase power consumption in NoC. There are many notable techniques that focuses on reducing power consumption in NoC such as dynamic voltage frequency scaling [19], bufferless NoC [8, 20, 21], and packet compression [22]. Thus, taking this into account, we propose a prefetch block placement technique which can prefetch aggressively as in L2 cache and achieves a faster cache access time as that of an L1 cache.

This paper proposes to increase the L1 cache size virtually at run time. Increase in cache size will result in placing more prefetch blocks which in turn will reduce cache misses. Reduction in cache misses will also reduce network packets. Hence, we build an energy-efficient caching strategy for prefetch blocks (ECAP) in TCMPs by reducing the number of network packets and AMAT during cache misses. ECAP uses the fact that applications running in each tile have different cache miss rates. For application experiencing heavy misses, it may happen that the L1 cache size is not sufficient to hold the working set size. Prefetching in such cases may harm the application performance by causing severe cache pollution. To reduce cache pollution, ECAP uses an intelligent prefetch block placement strategy at the L1 caches of neighbouring tiles that has less used cache sets. This helps in virtually increasing the L1 cache size without harming the useful demand blocks of that application. ECAP also reduces the number of network transactions by reducing cache pollution and also providing the extra storage options for the prefetch blocks.

We evaluate ECAP using out-of-order simulations with gem5 [23], BookSim [24] simulator, and multi-programmed workloads from SPEC CPU 2006 benchmark suite [25]. We compare ECAP with the baseline prefetch block placement technique (see Section 4 for more details). The result shows that ECAP is capable of having power savings in the network routers and links by around 14.42 and 27%, respectively, over the other techniques in a 64-core system. Moreover, the AMAT in ECAP is reduced to 23.56% with respect to the baseline prefetch block placement technique. It incurs only a small hardware overhead per tile with a power savings in the NoC for tile-to-tile communication.

Additional experimental result confirms that ECAP performs fairly well for a wide variety of system parameters as described in Section 4. Therefore, by virtually increasing cache size with minimal hardware overhead can result in achieving enhanced system performance with reduced energy and power consumption. This results in addressing the following questions:

- How does ECAP search less used space in adjacent tiles?
- How is a prefetch block located when it is placed in adjacent tiles?
- How does ECAP help in achieving reduced network transactions?
- What are the hardware overheads involved in ECAP?

This paper addresses the above queries in the subsequent sections. Section 2 provides the motivation behind this work. Section 3 describes the proposed technique, ECAP followed by a detailed experimental analysis in Section 4. Section 5 analyses the sensitivity of different parameters that determines the best performance of ECAP. Section 6 contains the hardware aspect of our proposed technique. Section 7 presents the related work done in this regard and finally this paper is concluded in Section 8.

2 Motivation

Some important terms used in this paper are defined as follows:

Target set: The target set of a block in a cache indicates the set index in the cache where the block is mapped.
Home cache: In TCMP, when a core c associated with a tile T_c requests for a block b, then the private L1 cache of T_c is considered as the home cache for the block b.
Remote cache: In TCMP, for a block b, the neighbouring (one-hop distant neighbours) L1 caches of its home cache are considered as remote cache.

We use next-line prefetcher as the baseline prefetching technique. This prefetcher on a cache miss for a block b initiates a prefetch request for the next sequential block $(b+1)$, if it is not already present in the cache. If the target set of cache block b is I, then next-line prefetcher fetches the block $(b+1)$ and places it in $I+1$.

We model a 64-core TCMP in gem5 [23] running real workloads from the SPEC CPU 2006 benchmark suite [25]. The workloads are generated by choosing different benchmarks with various cache footprints. Fig. 3 shows the tilewise distribution of L1 cache misses in a 64-core TCMP enabled with next-line prefetcher. We can observe that the number of L1 cache misses vary significantly across the cores. This is due to the diversity in cache footprints of applications running on these cores.

We categorise the applications into high and low based on the number of L1 cache misses. For high applications, the working set

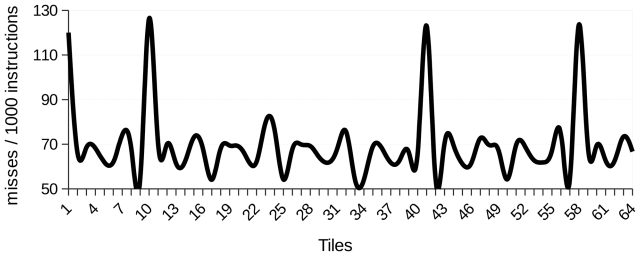


Fig. 3 *Distribution of L1 cache misses per tile*

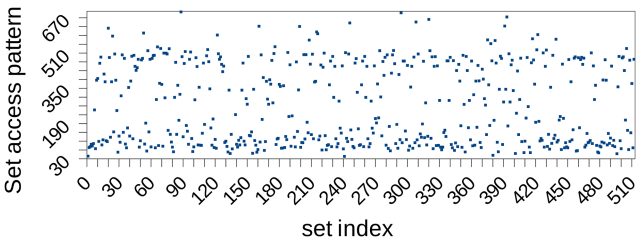


Fig. 4 *Cache set access behaviour during execution of workloads from SPEC CPU 2006 benchmark suite*

size is much larger than the available L1 cache size. Such applications suffer from cache space constraints resulting in an increase in the cache miss count. Prefetching for such applications may exacerbate the problem of cache pollution which may increase the number of cache block replacements. Each cache block replacement contributes to a network transaction, leading to increase in network traffic. Moreover, the evicted cache blocks may be subsequently re-requested by the cores. This can further lead generation of additional cache block request and reply packets. Light applications on the other hand may under-utilise the available L1 cache space. This may occur in two cases: (i) L1 cache space is sufficient to hold the working set of the application. (ii) Application exhibits more temporal locality than spatial locality. For such applications, the available L1 cache space may remain under-utilised resulting in cache space wastage.

Fig. 4 shows the frequency of set access pattern in L1 cache averaged across all the tiles in a 64-core TCMP enabled with next-line prefetcher. These statistics are collected for a time window of 10 K cycles after performing sufficient fast forwarding to eliminate cold misses. We can observe that the cache access pattern is non-uniformly distributed across the cache sets. Some of the sets are heavily used while some of them are lightly used. Thus, we can conclude that some applications under-utilise their cache space. Few cache sets of such cores can be utilised by the neighbouring cores running high applications for which either the cache is of insufficient size or has higher miss rates. Hence, a high application can utilise the lightly used cache sets of other cores to temporarily accommodate its prefetch blocks.

This motivated us to propose ECAP that increases the cache size of high applications as per its need. Furthermore, it reduces cache miss and cache pollution for high applications. It has an incremental effect in reducing the number of network packets, thereby reducing network congestion and power consumption in NoC. Reducing network congestion has a direct impact on reducing network latency which is responsible in reducing the AMAT of cache misses in the cores. Thus, the throughput of the core increases resulting in improved system throughput.

3 Energy-efficient caching of prefetch blocks

ECAP is a placement strategy for prefetch cache blocks in a TCMP system. When an L1 cache miss occurs at a core, a request packet is sent from the source tile to the L2 cache bank, where the block is mapped. The L2 cache bank sends the demand block as well as the next sequential block (prefetch block) as reply packets to the source tile. On reaching the source tile, the demand block is placed on the target set of its home cache. Our proposed technique ECAP finds the most suitable location to place the prefetch block.

Considering the L1 cache access time, the first suitable location to place a prefetch block is the target set in its home cache. However, if the target set in the home cache is heavily used, then placing a prefetch block there may result in evicting a useful demand block. thereby causing cache pollution. Therefore, ECAP then tries to search for a lightly used target set in the remote cache of its neighbouring tiles. Section 3.1 explains this searching procedure in detail. The prefetch block is stored in the suitable remote cache on finding a lightly used target set. Some metadata are stored in the home cache of the prefetch block which gives the information of prefetch cache blocks placed in the remote caches. A detailed description of the metadata storage is explained in Section 3.2. If no suitable target set is found in a remote cache, ECAP uses a special buffer called as prefetch buffer pool (PBP) in each local tile to accommodate few such prefetch blocks.

Fig. 5*a* shows a 4×4 TCMP, where tile 6 is running a high application. The illustration of ECAP technique is described as follows. ECAP initially explores the possibility of placing a prefetch block brought during a cache miss by tile 6 at its home cache itself. If ECAP could not place the prefetch block in the target set of its home cache (due to heavy usage of the existing cache blocks already present in the target set), then it explores the possibility of finding a suitable remote cache. ECAP searches for a lightly used target set in the remote caches, i.e. tile 2, 5, 7, and 10. If none of the target set is lightly used, ECAP uses the PBP of local tile, i.e. tile 6 to place the prefetch block.

For prefetch block placement, the remote caches are limited to one-hop distance only from the home cache. This is to reduce the traffic in NoC. It also reduces the cache access time when there is a hit in the remote cache. On a hit in remote cache, the prefetch block is placed in its home cache and the corresponding metadata is updated accordingly. The block searching mechanism in ECAP is explained in detail at Section 3.3. Throughout this paper, we use the term ‘adjacent’ and ‘neighbour’ interchangeably to indicate tiles that are at one-hop distance away from the source tile.

SCP and ECAP are not prefetch engines. A prefetch engine speculates the cache access pattern of an application. However, SCP and ECAP only use caching strategy to place prefetch blocks in a cache (remote or home). Hence, both SCP and ECAP can be implemented on top of any prefetchers such as stream and correlating prefetchers [14, 26]. If ECAP is implemented, an L1 cache block can be classified into one among the four categories:

- *Invalid blocks (I):* These are unused blocks.
- *Demand blocks (D):* These blocks contain words that are fetched on a cache miss by the local core.
- *Local prefetch blocks (P):* These blocks contain words that are prefetched on a cache miss by the local core.
- *Satellite blocks (S):* These blocks contain words that are prefetched on a cache miss by one of the adjacent core.

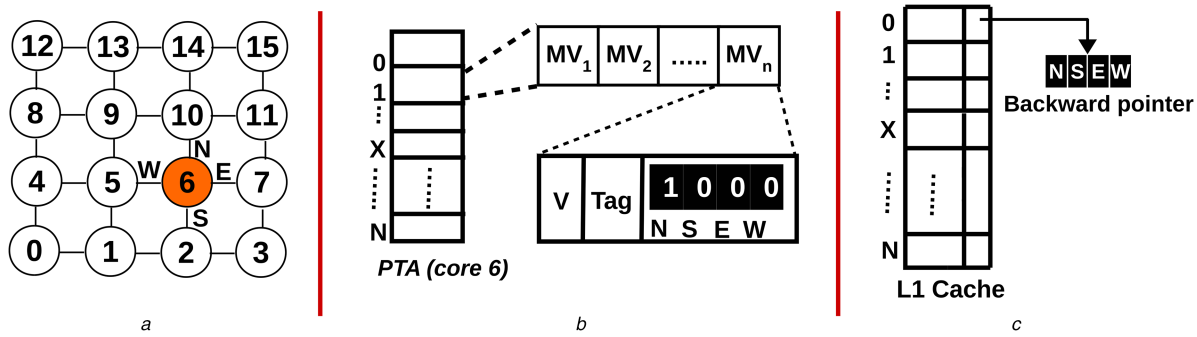


Fig. 5 *Metadata storage for ECAP*
(a) 4 × 4 TCMP, ***(b)*** Forward pointer, ***(c)*** Backward pointer

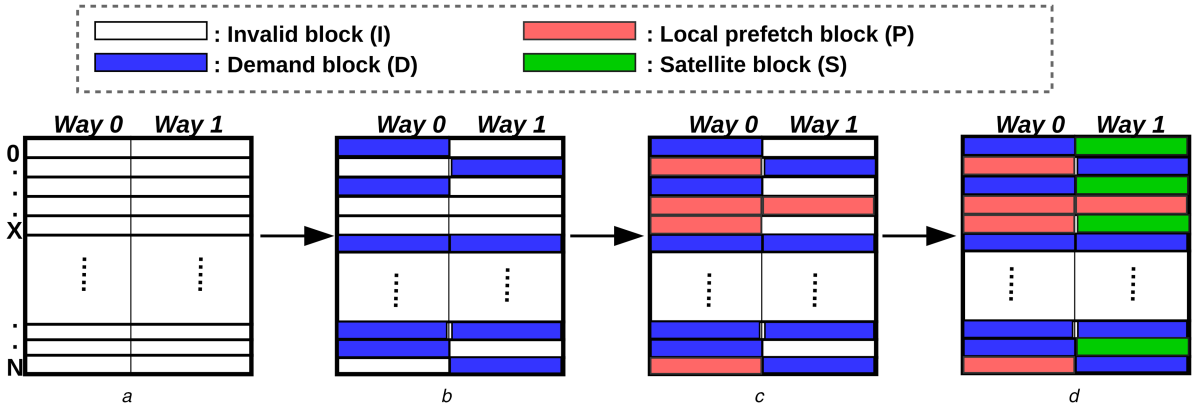


Fig. 6 *Contents of cache block in different scenarios*
(a) Cache empty, ***(b)*** No prefetching, ***(c)*** SCP, ***(d)*** ECAP

3.1 Searching for a target set in adjacent tiles

Once the prefetch request is sent to the L2 cache tile, searching for a suitable target set in remote caches is done in parallel, if the target set of the home cache is heavily used. ECAP adopts a simple neighbour searching scheme for prefetch block placement at remote caches. The source core searches for a suitable target set in the remote caches. This is done by sending probe packets to one-hop neighbours of the source core.

For prefetch block placement at remote caches, ECAP uses two policies: (a) prefetch block placement policy and (b) cache block replacement policy. Prefetch block placement policy deals with the selection of a target set in the remote cache. It uses the frequency of set access pattern of applications running in the remote caches and classifies them as either heavy or light. We define a cache set as heavy or light by counting the frequency of accesses per set within an interval of 1024 cycles. We use a 4 bit saturating counter per set with a threshold value of 12 for categorising sets into heavy and light. The cache replacement policy proposed in ECAP is *confidence-aware replacement policy* (CARP) which assigns different confidence values to the blocks present in the cache. Section 3.5 explains CARP in detail.

If one such target set is identified, the corresponding remote cache acknowledges with a positive response to the home cache. The remote cache also books a cache way in the target set to avoid race conditions. The satellite block on reaching at the home cache is forwarded to the corresponding target set in the remote cache.

3.2 Metadata management for satellite block mapping

For maintaining the records of satellite blocks and its subsequent management, ECAP uses *forward pointer* and *backward pointer* for each set of the L1 cache. The forward pointer is used for identifying the remote cache, where the block is currently residing and the backward pointer is used to identify the owner of a cache block (local cache block or satellite block).

3.2.1 Forward pointer: ECAP uses an additional map per tile called as prefetch tag array (PTA). The number of PTA entries is equal to the number of L1 cache set. It contains the details of all prefetch blocks placed at the remote caches. The target set of the prefetch block in the remote cache is same as that of the home cache. Similarly, the PTA is also indexed by the same target set. The metadata of the prefetch block is stored in PTA as shown in Fig. 5*b*. Each PTA index contains a set of mapping vectors (MVs) for prefetch blocks. Since PTA index and target set of remote cache is same, we need not store the set number in an MV.

A MV contains a valid bit (V), tag bits for a prefetch block, and four flag bits for each direction (N, S, E, W). In a mesh topology, a tile is surrounded by four adjacent tiles. The direction flag corresponds to one of the four neighbours. Since MV is used to forward a cache miss request to a remote cache, we call the MV as forward pointer. In a valid MV, exactly one of the direction flags will be set. Experimentally, we have found that in ECAP, each PTA index can store at most four prefetch blocks ($n = 4$) for optimised system performance.

3.2.2 Backward pointer: As shown in Fig. 6*d*, an ECAP enabled cache may contain four types of blocks in its home cache, i.e. invalid block, demand block, local prefetch block, and satellite block. To indicate whether the block stored in an L1 cache belongs to itself or to a neighbour (satellite), four bit flags are used. Each flag bit corresponds to one of the four neighbours in a mesh topology. For satellite blocks, exactly one of the flag bits is set. On the other hand, for demand blocks and local prefetch blocks, none of the flag bit is set. Since the flag points back to the owner of the block, these flags are also called as backward pointers as shown in Fig. 5*c*.

3.3 Block searching and identification

In a conventional cache, whenever a core generates a request for a word, the block containing the word is searched in its home cache. For a TCMP enabled with ECAP, the block is searched in its home cache, PTA of the local tile, and also PBP of the local tile in parallel. Fig. 7 depicts the block searching procedure in ECAP.

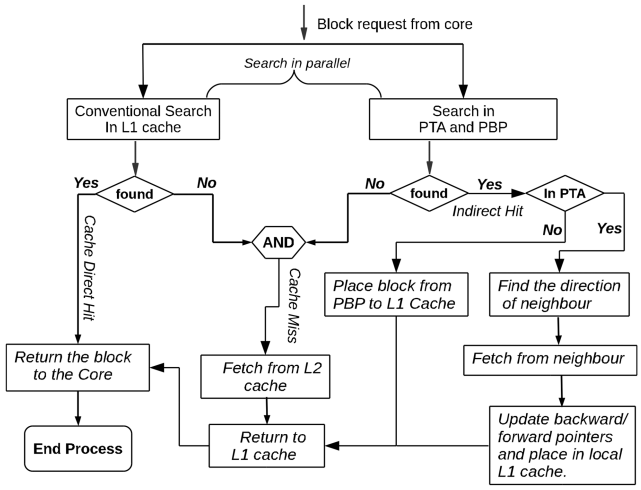


Fig. 7 *Flowchart for block searching*

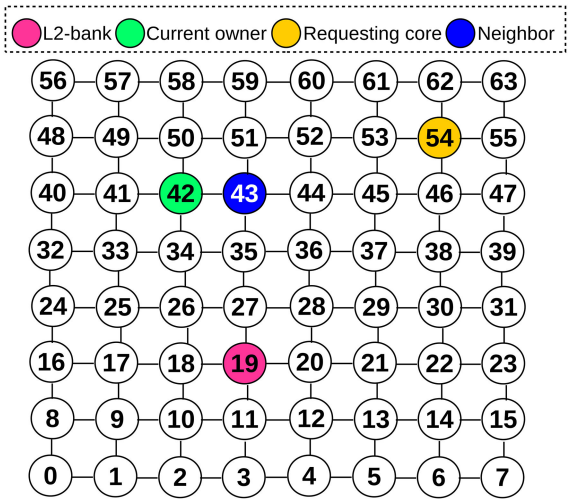


Fig. 8 *Example of coherence protocol for* 8 × 8 *2D mesh in ECAP*

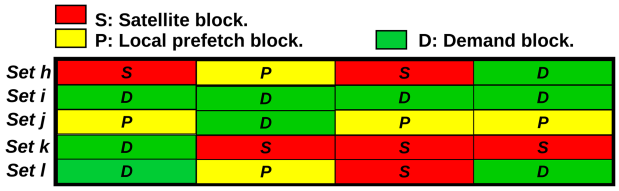


Fig. 9 *Snapshot of ECAP enabled cache*

If the block is a hit in its home cache, it is called as 'direct hit'. On a direct hit, the word is returned to the core immediately. If the block is found either in PTA or PBP, it is called as an 'indirect hit'. If the indirect hit is from PTA, this indicates that the block is placed in a remote cache. The forward pointer in the corresponding MV of the PTA index will identify the remote cache. Request packets are sent from the source tile to the remote cache tile to bring the cache block in its home cache. On placing the block in home cache, the backward/forward pointers are updated in the remote cache and the local PTA, respectively. Similarly on a hit in PBP, the block is removed from PBP and placed in the target set of its home cache. Once the block is placed in the home cache after an indirect hit, it is considered as a demand block.

3.4 Cache coherence management

Multicore processors can have the same data cached in the private caches of different cores. We use MESI CMP protocol for maintaining data coherence across multiple cores. Neighbour placement of prefetch blocks is hidden from the underlying coherence protocol. The actual placement of prefetch blocks is known to the L1 cache only but not to its L2 cache bank. Implementation of ECAP requires minor modification in the cache coherence protocol as described in the following example.

For example, Fig. 8 shows that core 42 sends a prefetch request for a block, whose L2 bank is in tile 19 as per the SNUCA block mapping policy. Consider that the prefetch block is stored as a satellite block in tile 43 with backward pointer set to tile 42. Hence, the tag address of the block is stored in the PTA of tile 42. The coherence protocol running in the system updates the block status as E (exclusive) and owner as 42 (not tile 43). On receiving any request for the same block (read/write (RD/WR) from a different tile 54, the L2 bank sends a request to the curren)t owner of the block, i.e. tile 42 as per the record. Tile 42 searches for the block using block searching procedure as described above. The block search procedure results into a hit in its PTA. From the MV, the location of the block is determined, i.e. tile 43. Hence, tile 42 sends a request to tile 43 in order to forward the corresponding block from tile 43 to the requesting core in tile 54. On successfully forwarding the block, tile 54 acknowledges back to the L2 bank (tile 19) and the status of the block is changed from E to S (shared) with an addition of another owner as 54. The remote cache has no RD or WR access permission on its satellite blocks. This avoids the case of data inconsistency that may arise when the block resides in a remote cache.

3.5 Confidence-aware replacement policy

The simplest block replacement policy used in traditional CMPs is least recently used (LRU). LRU works within each cache set to identify a victim block during cache replacement [27–29]. It exploits the reuse factor of the cache block. A newly inserted block is placed in most recently used (MRU) position and for every reuse of the block, it is promoted to MRU. Otherwise, the block eventually reaches the LRU position and gets evicted from the cache.

However, the inherent problem of LRU is that a block once promoted to MRU position requires a longer time period to become LRU [27, 28], if the access to the cache set is very less. Hence, the demand blocks in such cache sets become dead blocks [30, 31]. A cache block is called dead if it is never accessed again before its eviction. Eviction of such blocks from the set gets delayed resulting in poor usage of critical resources such as cache space. Some interesting dead block prediction mechanisms are already been proposed [30, 31]. Most of the existing proposals are for the last-level caches. We understand this limitation of conventional LRU policy and propose a new CARP that is more suited for ECAP enabled caches.

To discuss this issue in detail, we have taken an example as shown in Fig. 9. This figure shows the structure of a four-way set-associative L1 cache filled with three possible types of blocks. Multiple colours are used to show the different types of blocks present in each cache set:

- *Category I:* Lightly used sets (*set i*) containing all demand blocks.
- *Category II:* Lightly used sets (*set j*) containing a combination of demand blocks and local prefetch blocks.
- *Category III:* Lightly used sets (*sets h, k, l*) containing a combination of demand blocks, local prefetch blocks, and satellite blocks.
- *Category IV:* Heavily used sets. These sets can also be divided into two sub-categories based on the presence of local prefetch and demand blocks. Since ECAP does not use such sets for satellite blocks, the *N* is not relevant for this discussion.

For the placement of a satellite block, ECAP may choose a lightly used set of any category, i.e. I, II, or III as the target set (in a remote cache). The category shown in this figure is for a particular instance and the set behaviour may change dynamically. Such a behaviour of cache set is discussed in Section 3.6.

During classification of sets into heavy or low, ECAP uses the core behaviour of local tile only (access by neighbouring tiles are not considered). Since the cache set is lightly used by the local core, demand blocks residing in such a cache set may experience

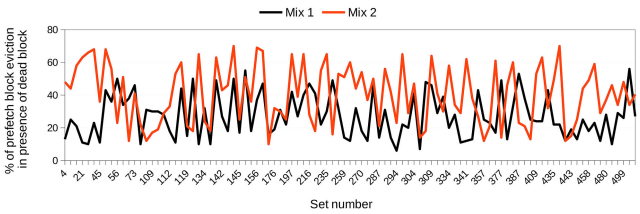


Fig. 10 *Percentage of prefetch blocks evicted (due to replacement policy) from a set in the presence of dead block in the same set*

Table 1 Confidence values used in CARP for a lightly used set

Block types	Confidence value
demand block	3
local prefetch block	4
satellite block	4

minimal access. In addition to these if LRU replacement policy is used, such cache blocks will require a longer time in demoting to LRU position (due to low access by the local core) before finally evicting from the cache. Hence, the chances of a demand block becoming a dead block is more for a lightly used set. Since prefetch blocks (both local prefetch blocks and satellite blocks) are speculatively brought into the cache for future use, such blocks are expected to experience a hit after a longer time period than a demand block. Hence, LRU replacement policy will result in demoting the prefetch blocks faster than a demand block. For categories II and III, such a dead block may result in evicting the prefetch blocks early. Therefore, LRU replacement policy may result in decreasing the effectiveness of prefetch block placement strategy used in ECAP.

Dead blocks in category IV can also create similar issues and may remove important blocks from the set. Since this work uses only the lightly used set of remote caches, CARP gives equal priority to every local block that is placed in the heavily used sets. The policy gives benefit for sets belonging to categories I, II, and III. For category IV, the policy may not improve the performance but at least gives the same performance as that of LRU.

Fig. 10 shows the percentage of prefetch blocks evicted from a cache set in the presence of dead blocks. The percentage is calculated as (total prefetch blocks evicted in the presence of dead blocks/total prefetch blocks evicted)×100. The experiment is performed in an in-house stand-alone cache memory simulator written in C++. To detect dead blocks with 100% accuracy, the programme takes the traces as input and simulates the cache evictions. Without trace, the detection of a dead block with 100% accuracy is not possible as the future cache access pattern is unknown. From this figure, we can observe that in most of the cache sets, the prefetch blocks are unnecessarily evicted in the presence of dead blocks within the same set. Our proposed CARP handles these issues effectively. This is done by assigning different confidence values to cache blocks pertaining to different categories of cache sets (heavy/light).

A replacement algorithm has three policies: insertion, promotion, and eviction. Insertion means where to place the block initially. Promotion policy deals with how to handle the block when there is a hit in the block. Replacement/eviction policy selects the victim block to be replaced from the cache.

3.5.1 Insertion, promotion, and demotion policies of CARP: CARP uses a 3 bit saturating counter per block, also known as the *confidence value* of the block. Confidence value is used as a weight to define how important the cache block is for the local core's performance. Lower the confidence value, lower is the reuse frequency of the cache block. During the event of cache block replacement in a target set in remote cache, the cache block with lowest confidence value is chosen as the victim.

When a block is newly inserted to a cache set, based on the type of the block, the confidence value is initialised with either three or four. The confidence values are decided experimentally. Whenever the cache block experiences a hit, its confidence value is incremented. At the end of every 32 cycles also called as *window*, the confidence value of all the blocks are decremented by one. By this process, the confidence value of dead blocks will eventually become lower than the blocks that are frequently accessed. However, the confidence value of those blocks that are accessed frequently keeps on increasing. This results in clear separation of cache blocks based on its access pattern within each set. For a heavily used set, since the set is accessed frequently, equal confidence value is given to each cache blocks that are mapped to it. Thus, the confidence value of each block are initialised with four irrespective of its type. This is similar to LRU, where each block is initially placed to MRU position. For a lightly used set, the confidence value is initialised as mentioned in Table 1.

The main motive of attaching a higher value four to the satellite block is to retain such blocks for a longer period of time. This is because replacing a satellite block incurs more cost in terms of network packets. Moreover, when the cache set is lightly used, the usage of such set by the local core is also less. Therefore, more priority is given to the satellite blocks so that it is kept for a longer time period. This helps in effectively utilising the less used sets in L1 caches of low applications.

3.5.2 Replacement policy of CARP: During a cache block replacement, a block with least confidence value is considered as the victim block. If more than one such block is present, then CARP selects a random block. Before a victim block is evicted, the L1 cache controller checks the direction flags in its backward pointer. In case any of the direction flag is set (the victim is a satellite block), the cache controller informs the owner of the satellite block by sending an invalidation message. In the meantime, the evicted block is stored in the victim buffer until it hears from the owner. Meanwhile, the tile can store the incoming block in its target set.

The owner of the prefetch block on receiving the invalid request invalidates the forward pointer in its PTA. It then acknowledges back to the remote cache. On receiving the acknowledgment from the block owner, the evicted block which is temporarily stored in the victim buffer for ensuring consistency is removed. With this the replacement procedure is completed.

3.6 Behavioural changes of cache sets

In TCMP, the access pattern of cache set changes with time [32]. Some cache sets which are lightly used previously may change its pattern to heavy and vice versa. This is because the load on the cache set changes as the application makes progress. Hence, the heavy sets that are not used to place the satellite blocks in one phase may be used for prefetch block placement in another phase. A cache replacement policy should also handle the dynamic nature of cache sets.

CARP handles this issue very efficiently. When a heavily used set changes into lightly used, new incoming satellite blocks are given higher priority by attaching a higher confidence value (4). For the existing blocks in the set, the blocks are evicted based on the least confidence value as explained in Section 3.5.2. On the other hand, when a lightly used set changes to heavily used, no satellite blocks are further mapped to such sets. Hence, blocks further mapping to such sets are local to the core and are initialised with the same confidence value as mentioned in the previous section. In addition to these, the confidence values of the previously mapped satellite blocks to such lightly sets are gradually decremented after every window. Therefore, CARP dynamically avoids the insertion of any new satellite blocks into a newly converted heavily used set.

3.7 Near vicinity prefetcher

The idea of prefetch block placement in neighbouring tiles is taken from our previous work, near vicinity prefetcher (NVP) [33]. On top of the naive NVP design, ECAP uses additional features such as the replacement policy CARP and suitable neighbour selection which significantly improves its performance. NVP is focused on

increasing the performance of prefetch-enabled caches running multiple applications in TCMP. It uses LRU replacement policy in the caches.

On the other hand, ECAP specially focuses on providing the best choices on satellite block placement and dead block removal. ECAP proposes a replacement policy CARP to remove such blocks from the lightly used set of remote caches, thereby enabling efficient utilisation of cache space in remote caches. This also helps in reducing the network packets by providing more time for a satellite block to reside in the caches. Hence, ECAP also contributes to reduce the power and energy saving aspects of prefetch block placement in remote caches. This is supported by additional results produced using extensive experimental analysis as described in the subsequent sections.

4 Experimental analysis and workload characterisation

Experimental setup: We evaluate the effectiveness of our proposed technique, ECAP using gem5 [23], a cycle-accurate full system simulator. BookSim 2.0 [24] is closely integrated with gem5 to model the NoC. We also use CACTI 6.0 [34] to evaluate the power and area analyses of ECAP, and Orion 2.0 [35] to calculate the network power. The experiments are conducted using workloads generated from SPEC CPU 2006 benchmark suite [25].

The system is modelled with out-of-order x86 superscalar processor, two levels of memory hierarchy, and uses directory-based MESI CMP coherence protocol. The L2 cache is shared among all the tiles and the L1 caches (L1I cache and L1D cache) are private to each core. The access latency of L1 cache is 2 cycles while that of L2 cache is 12 cycles. We assume that the access latencies of PBP and PTA are same as that of L1 cache. The main memory latency is modelled as 100 cycles. We use SCP as the baseline prefetch block placement technique with the conventional next-line prefetcher for data prefetching. The simulation parameters are summarised in Table 2.

We have also implemented NVP with LRU replacement policy (NVP-LRU), NVP with pseudo LRU (PLRU) replacement policy (NVP-PLRU), and our proposed technique ECAP with CARP replacement policy. We analyse the performance of ECAP with different replacement policies in terms of cache misses (in misses per kilo instructions (MPKI)), network packets, packet latency in the network, AMAT, percentage of direct and indirect hits, and packet distribution per hop. We also analyse the sensitivities of various design parameters to evaluate the performance of ECAP in Section 5.

Workload description: The workloads used to evaluate ECAP are created from real SPEC CPU 2006 benchmark suite. As mentioned in Table 3, 12 benchmarks from SPEC 2006 suite are used as single core applications. The benchmarks are classified into three categories based on their MPKI values as low (MPKI < 5), medium (25 > MPKI < 5), and high (MPKI > 25) as described in Table 3. The MPKIs are calculated after fast forwarding for 10L cycles in order to avoid cold misses. Using the benchmarks, 64-core multi-programmed workloads are generated, where in each core one of the SPEC CPU 2006 benchmark application (high, low, or medium) is mapped.

We have generated five workload mixes (B1–B5) each consisting of four (P, Q, R, S) benchmarks chosen from high, low, or medium-MPKI benchmark categories. H(Bi), M(Bi), and L(Bi) indicate that the workload Bi ($1 \le i \le 5$) is generated from high, low, or medium-MPKI benchmarks, respectively. The workload B1 is generated for 64-core TCMP by mapping benchmark P to first 16 cores (0–15), Q to next 16 cores (16–31), R to the next set of 16 cores (32–47), and S to the remaining set of 16 cores (48–63). B2 is generated by dividing 64-cores into four equal clusters, where each cluster has 16 cores. Within each cluster, benchmark P runs on core 0–3, Q runs on core 4–7, R runs on core 8–11, and S runs on core 12–15. The pattern is repeated across all the four clusters. Similarly for B3, 64 cores are divided into 8 clusters, where each cluster has 8 cores. With the pattern repeated within each cluster, P runs on first two cores, Q runs on next two cores, and so on. In a similar fashion B4 and B5 workloads are also generated.

Table 2 Simulation parameters

gem5 and BookSim configuration	
processor	64, x86 cores with out-of-order execution
processor frequency	3 GHz
L1 cache/core	64 kB, four-way associative, 32 B block
L2 cache/core	1 MB, 16-way associative, 64 B block
L1 cache access latency	two cycles
L2 cache access latency	12 cycles
prefetcher	next-line prefetcher
prefetch block placement	SCP (baseline), ECAP (proposed)
NoC topology	8×8 2D mesh with *XY* routeing
packet size	one-flit request and four-flit reply

4.1 Effect on L1 cache miss

As detailed above, we examine the impacts of ECAP and NVP with different replacement policies on L1 cache misses using real workloads from SPEC CPU 2006 benchmarks mixes. Fig. 11 shows the normalised reduction in L1 cache miss of various workloads generated from the three benchmark categories (high/low/medium). From this figure, we can observe that on an average the L1 cache misses for NVP-LRU, NVP-PLRU, and ECAP reduce by 8.05, 14.28, and 23.42% as compared with SCP.

L1 cache miss reduction in ECAP is due to many reasons. ECAP increases the cache space of high applications dynamically

Table 3 Details of SPEC CPU 2006 workload constituents; $X(Y)^n$: benchmark X runs in Y cores with repetition of n times

high MPKI (H)	hmmer, leslie3d, lbm, mcf
medium MPKI (M)	bwaves, gcc, bzip2, gamess
low MPKI (L)	calculix, gromacs, h264ref, gobmk
B1 (Mix1)	$P(16), Q(16), R(16), S(16)$
B2 (Mix2)	$(P(4), Q(4), R(4), S(4))^4$
B3 (Mix3)	$(P(2), Q(2), R(2), S(2))^8$
B4 (Mix4)	$(P(4), Q(4))^4, (R(4), S(4))^4$
B5 (Mix5)	$(P, Q, R, S, S, R, Q, P)^8$

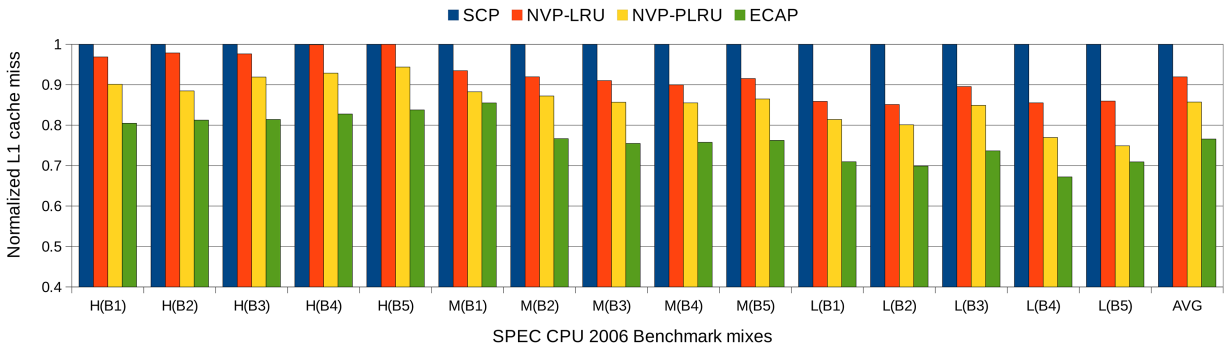


Fig. 11 *Reduction of L1 cache misses for different prefetch block placement techniques*

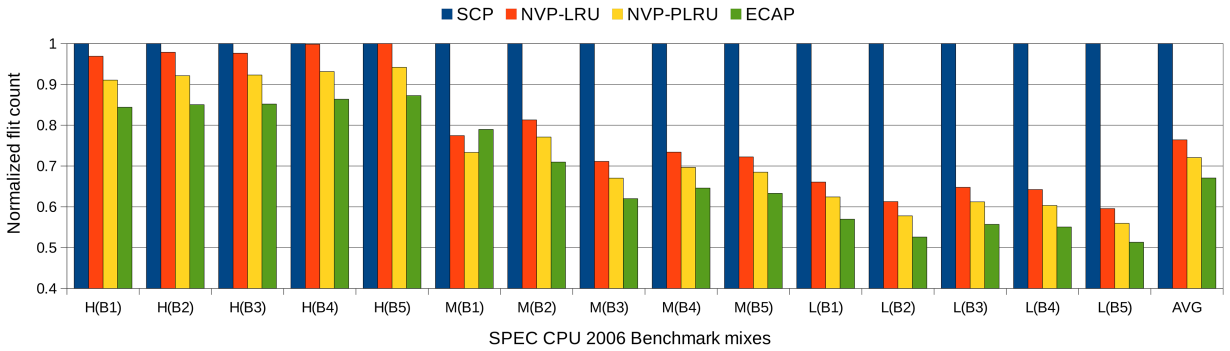


Fig. 12 *Reduction of flits in NoC for different prefetch block placement techniques*

at run time. This increase in cache space allows placing more prefetch blocks in remote caches and in PBP. Applications that have good spatial locality benefits from this placement. ECAP also reduces cache pollution. A useful demand block that is already cached can reside for a longer time period. The confidence value associated with a demand block increases when it experiences re-referencing during its residency in the cache. Therefore, the cache sets where such blocks are mapped turns out to be a heavily used set. ECAP targets only lightly used set for prefetch block placement in its home cache and also in the remote caches. Thus, ECAP avoids evicting useful demand blocks from the cache, thereby decreasing the L1 cache miss count.

From Fig. 11, we can observe that for high-MPKI workloads, the performance improvement of NVP-LRU is negligible as compared with that of SCP. The load on local L1 cache is more when high-MPKI applications are running on the cores. As a result of this, the home caches as well as the remote caches are almost utilised by the applications running on the respective cores. This makes it hard for NVP to place prefetch blocks in remote caches. In addition to this, LRU replacement policy cannot recognise dead blocks within a cache set. Hence, NVP-LRU cannot utilise the cache space efficiently.

However, NVP-PLRU and ECAP perform better than NVP-LRU. The relative reduction of L1 cache misses in both the techniques is due to the fact that NVP-PLRU provides a sub-optimal choice, whereas ECAP produces an optimal choice in identifying victim blocks during cache replacement. PLRU policy distinguishes only the most recently accessed cache block within a set. Apart from the most recently accessed block, PLRU selects a victim randomly among other blocks within the cache set. This results in providing a sub-optimal solution to find a victim block during prefetch block placement in the home cache and remote caches. Hence, NVP-PLRU may also evict a demand block that has been accessed by the core within few cycles.

CARP on the other hand, provides an optimal solution by replacing a less used cache block or a dead block within a cache set. If a cache block is not accessed periodically within a window, the confidence value of the block reduces. The eviction policy of CARP chooses a victim block that has the least confidence value. Hence, ECAP evicts the least used cache block from the cache. This results in fewer cache misses as compared with SCP, NVP-LRU, and NVP-PLRU. This also improves the efficiency of ECAP in placing prefetch blocks in home cache and remote caches. In this figure, we can clearly observe a considerable reduction of L1 cache miss in ECAP as compared with NVP-PLRU and NVP-LRU in all the workloads.

4.2 Effect on network traffic

Fig. 12 shows the normalised reduction of flits (basic flow control unit in NoC) in 64-core TCMP organised as 8×8 2D mesh network running SPEC CPU 2006 benchmark mixes. Reduction in number of flits reduces traffic in the network. From this figure, we can observe that there is an average reduction of flits by 23.58, 27.93, and 32.93% in NVP-LRU, NVP-PLRU, and ECAP, respectively, as compared with SCP.

As described in Section 4.1, ECAP reduces the number of L1 cache misses in the system by efficiently utilising the less used cache space as well as removing dead blocks from remote caches. Efficient utilisation of cache space increases the cache size virtually which in turn reduces cache misses. Reduction in cache misses results in generating less network packets. Therefore, the amount of network traffic decreases in ECAP.

In NVP, the reduction of network packets for high-MPKI workload is less than that of medium- and low-MPKI workloads. This is evident from the fact that in high-MPKI workloads, the number of lightly used set is less. Therefore, most of the prefetch blocks are placed in the PBP of local tiles. Hence, high-MPKI applications are not NVP friendly. However, for all types of applications, ECAP reduces network packets by identifying dead blocks from the cache irrespective of the set type (light/heavy). So removal of an inactive demand block results in placing useful demand blocks in the cache. Hence, the cache space is efficiently utilised resulting in fewer cache misses which further reduce network packets.

4.3 Effect on average packet latency

Reduction in network packet reduces the traffic in NoC. This results in faster tile-to-tile communication. Hence, the average packet latency in the network reduces. This can be clearly observed in Fig. 13, where NVP-LRU, NVP-PLRU, and ECAP reduce packet latencies by 16.2, 20.05, and 25.34%, respectively. For high-MPKI applications, H(B1), H(B2), and H(B3) perform better than H(B4) and H(B5). Owing to the cache usage pattern for such applications, the performance of ECAP is nominal.

For medium- and low-MPKI applications, the variations in cache set usage lead to increased performance of ECAP. From the average packet latency figure, we can observe that ECAP performs better than NVP-LRU and NVP-PLRU. This is because in lightly used sets, the demand blocks residing in such sets become inactive. Satellite blocks residing in such cache sets may experience a delayed access due to its speculative nature. Therefore, the chances of evicting such block is more than an inactive demand block. CARP gives more chances for a satellite block to stay in a remote cache by assigning them higher confidence values. This effectively helps in utilising the less used cache sets in remote caches, thereby increasing the cache size for high applications.

4.4 Effect on AMAT

In TCMP, the AMAT of an L1 cache depends on the hit time of L1 cache, miss rate of L1 cache, miss penalty of L1 cache, and also on the underlying network condition. Since each cache miss is carried as packets in the network, the network congestion plays a major role in determining the AMAT. As shown in Fig. 14, NVP-LRU, NVP-PLRU, and ECAP reduces to 12.9, 17.86, and 23.56% of AMAT as compared with SCP. On an average reduction of 32.93% network packets in ECAP results in reducing the packet latency by 25.34%. This has a direct impact on reducing AMAT during cache misses occurring in the core.

For high applications, the reduction of packets in the network is less as compared with SCP. Therefore, the reduction of AMAT is also less for such applications. On the other hand, for medium- and low-MPKI applications, NVP-LRU, NVP-PLRU, and ECAP perform better in reducing the network packets. Therefore, the

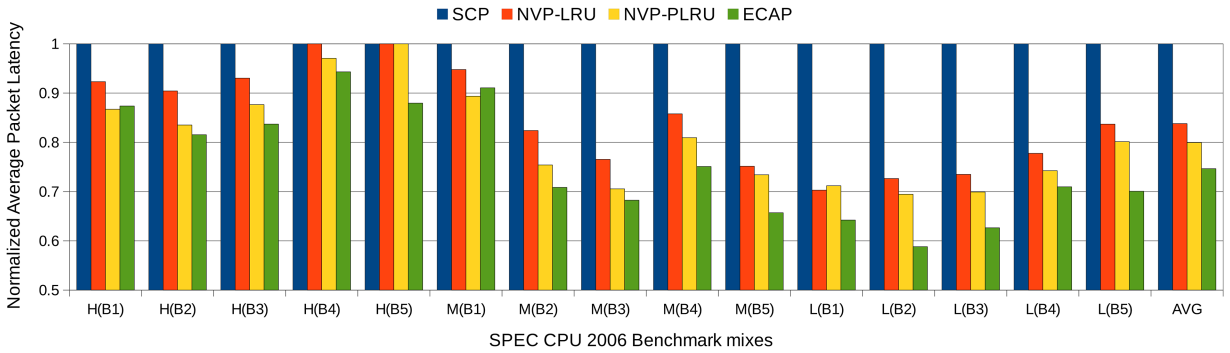


Fig. 13 *Reduction of packet latency in the network for different prefetch block placement techniques*

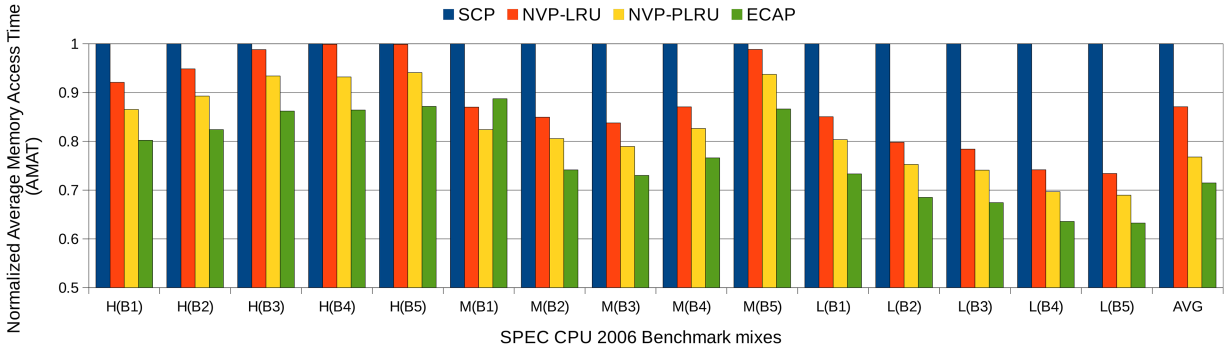


Fig. 14 *Reduction of AMAT for different prefetch block placement techniques*

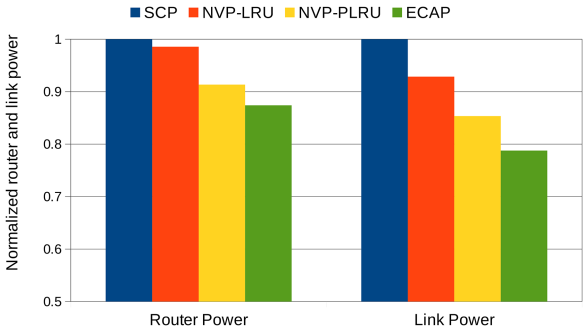


Fig. 15 *Power consumption across routers and links in NoC*

AMAT for medium and low applications shows significant reduction as compared with high-MPKI benchmarks. However, ECAP helps in achieving best performance than NVP-LRU and NVP-PLRU as shown in this figure.

4.5 Effect on dynamic power consumption across routers and links in NoC

Orion 2.0 tool [36] is used to evaluate the dynamic power consumption of routers and links in NoC. The proposed techniques are built using 32 nm process technology and the operating frequency of the NoC is 1 GHz. The tiles in NoC are arranged as 8 × 8 2D mesh topology. For our experimental purpose, the routers are input queued with five input/output ports. The number of VCs used per input port is four. Power is consumed when a flit resides in the VCs and also during flit traversal from one router to another through NoC links.

Fig. 15 shows the dynamic power consumption across routers and links in NoC. From this figure, we can observe that for various techniques, i.e. SCP, NVP-LRU, NVP-PLRU, and ECAP, ECAP achieves the best reduction in power consumption across routers and links. ECAP on an average reduces the router power and link power by 14.42 and 27%, respectively, as compared with SCP. This is because the reduction of cache pollution reduces the number of cache misses. Thus, the number of flits in the NoC reduces. Hence, the routers and links in the underlying NoC have to carry less number of traffic across the tiles. This resulted in reducing the dynamic power consumption at the routers and links in NoC.

4.6 Percentage of hits in different categories

Fig. 16 shows the percentages of direct and indirect hits in ECAP for different types of benchmark mixes. In this figure, we have shown the hit percentages for ECAP only because CARP replacement policy helps in achieving better performance than NVP-LRU or NVP-LRU. Indirect hits are achieved either from PTA or PBP. Hit in PTA means that the prefetch block is placed in remote caches. Such hits are mentioned as PTA hits in this figure. More PTA hits indicate that more number of prefetch blocks are placed in remote caches. Therefore, applications with more PTA hits are better ECAP friendly than those with less PTA hits.

Clearly, from this figure, we can observe that for most of the high applications, the number of PBP hits is more than that of PTA hits. From 18.8% of indirect hits, around 7.9% of the hits are from PTA and rest 10.9% of the hits are from PBP. As mentioned earlier, for high-MPKI application, the number of lightly used set is substantially less. Therefore, experimentally it is found that most of the prefetch blocks are accommodated in the PBP of its local tile as shown in this figure. For medium-MPKI applications, 21.74% of hits are indirect hits. Out of the total indirect hits, most of the hits are from PTA, i.e. 15.04%. In low-MPKI applications, a total of 30% hits are indirect hits and maximum hits are PTA hits which constitute around 21.26% of total indirect hits.

Therefore, it is clear that ECAP combined with CARP replacement policy performs better for benchmark mixes that has a combination of high, medium, and low MPKI or some specific patterns of high-MPKI workloads such as H(B1).

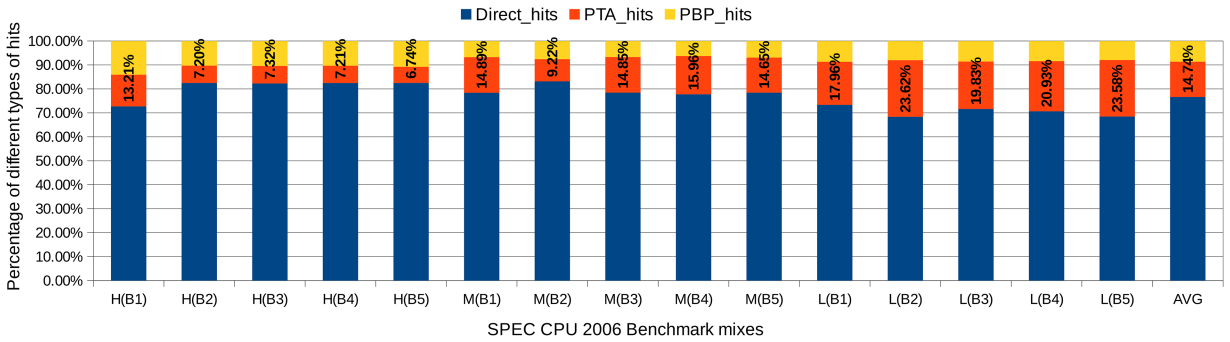


Fig. 16 *Percentage distribution of direct and indirect hits in ECAP*

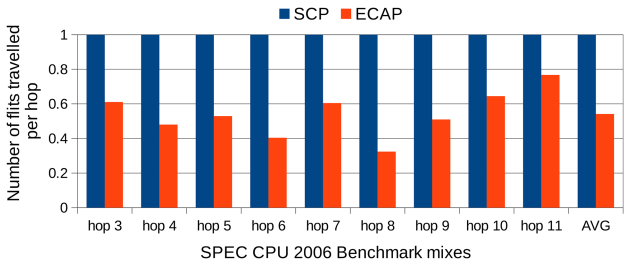


Fig. 17 *Normalised reduction of long-distance communication during cache miss in NoC*

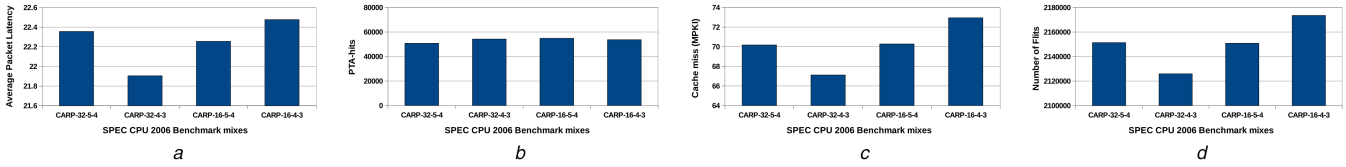


Fig. 18 *PBP = 8, change in window size and confidence value in CARP replacement policy*
(a) Network packet latency, ***(b)*** PTA hits, ***(c)*** Cache miss, ***(d)*** Number of flits

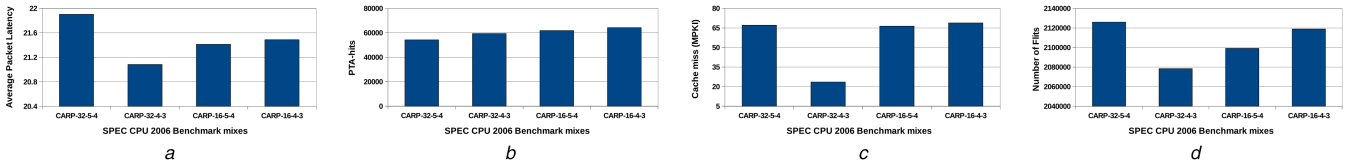


Fig. 19 *PBP = 16, change in window size and confidence value in CARP replacement policy*
(a) Network packet latency, ***(b)*** PTA hits, ***(c)*** Cache miss, ***(d)*** Number of flits

4.7 Reduction of long-distance network packets in ECAP

Fig. 17 shows the normalised reduction of long-distance packets occurring during cache misses in ECAP as compared with the baseline prefetch block placement technique SCP. In this figure, we call those packets as long distance that requires equal or more than three hops to reach its destination. From this figure, we can observe that on an average ECAP reduces around 49.2% of long-distance packets in the network during cache miss.

In ECAP, the prefetch blocks are placed at remote caches that are at one-hop distance away from the requesting core. Hence, such prefetch blocks are fetched from shorter distances, thus avoiding long-distance communication during cache misses. As a result of this, ECAP hides the number of long-distance communication required to travel for a cache miss packet in the network by placing the prefetch blocks in a remote cache. This also justifies the fact that due to placement of prefetch blocks in remote caches, the existing demand blocks from the cache are not evicted. Thus, in a prefetch-enabled cache, ECAP reduces cache pollution caused by prefetch blocks.

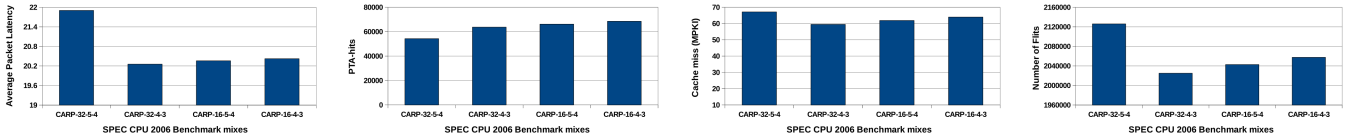
5 Sensitivity analysis

We hereafter concentrate solely on CARP and examine its sensitivity to various parameters. CARP uses various parameters such as window size (W) and different confidence values for satellite block and local block (prefetch and demand blocks) in each L1 caches. We change different parameters and plot the results as network packet latency, cache misses, number of PTA hits, and number of flits. For ECAP, the PBP size is also one of the parameters that determines the performance of ECAP. Combining the parameters, i.e. PBP size with window and confidence values plays a significant role in determining the performance of ECAP.

This section describes the variation among the parameters and tries to come up with an optimal design parameter for better system performance. For each PBP size, we change different parameters of CARP and we name each variations as CARP_W_S_D, where W is the window size, S is the confidence values for satellite block and local prefetch block, and D is the confidence value for local demand block. Figs. 18–20 show variations in different parameters and its impact on system performance.

From these figures, we observe that for different PBP size, the least value of average packet latency in Figs. 18*a*, 19*a*, and 20*a* is around 21.8, 21, and <21, respectively. This behaviour is found in CARP_32_4_3 (PBP size = 32). It is self-explanatory from the fact that larger the PBP size, more number of prefetch blocks can be placed in the PBP of local tile. Hence, the average packet latency in the network reduces. Among all the variation of CARP parameter, CARP_32_4_3 performs better than CARP_32_5_4, CARP_16_5_4, and CARP_16_4_3.

In Figs. 18*b*, 19*b* and 20*b*, we can observe that the number of PTA hits is almost similar for all the variations of CARP parameters in larger window sizes, i.e. $W = 32$ (CARP_32_5_4 and CARP_32_4_3). For $W = 16$, the PTA hits are less as compared



a *b* *c* *d*

Fig. 20 *PBP = 32, change in window size and confidence value in CARP replacement policy*
(a) Network packet latency, ***(b)*** PTA hits, ***(c)*** Cache miss, ***(d)*** Number of flits

Table 4 Additional hardware cost per tile in ECAP

Cache size per tile	1024 kB (L2) + 64 kB (L1) = 1088 kB		
Component	Bits/entry	Number of entries	Bytes per tile
backward pointer	4	512 (sets)×4 (ways)	1024
PTA	22	512 (sets)×4 (MV)	5632
PBP	274	16	548
access counter/set	4	512 (sets)	256
confidence value/block	3	512 (sets)×4 (ways)	768
	additional storage in ECAP per tile		**≃9 kB**
	hardware cost per tile, %		**0.83%**

Table 5 Area and power consumption analysis in ECAP

Component	L1 cache	L2 cache	PTA	PBP
area, mm^2	0.1374	1.92	0.0074	0.00027
power, nJ	0.0304	0.2872	0.0017	0.00006
	total area (in SCP): (L1 + L2)			2.0574 mm^2
	total area (in ECAP)			2.066 mm^2
	total power (in SCP): (L1 + L2)			0.3176 nJ
	total power (in ECAP)			0.3194 nJ
	increase in area per tile, %			0.42%
	increase in power per tile, %			0.57%

with that of $W = 32$. Figs. 18*c*, 19*c*, and 20*c* show the number of L1 cache misses that occurred in different combinations of W and confidence values. Among the three figures, in Fig. 19*c*, CARP_32_4_3 experiences less misses as compared with CARP_32_5_4, CARP_16_5_4, and CARP_16_4_3. This scenario is also similar for Figs. 18*c* and 20*c*.

This behaviour can be well-explained because if the window size is small, then CARP reduces the confidence value of blocks faster. As a result of this, the blocks are demoted faster. This may result into evicting a cache block from the lightly used set much prior than experiencing a hit. Experimentally, it is found that CARP_32_4_3 is the optimal combination of PBP size, window size, and confidence value as compared with other combinations. This is also evident from Figs. 18*d*, 19*d*, and 20*d* which show that the reduction of flits is maximum for $W = 32$ and confidence value for satellite block and local prefetch block as four and that of demand block as three.

Therefore, we use the combination of PBP size = 16, and CARP parameters as window of 32 cycles and confidence values as mentioned in Table 1 throughout our experiments.

6 Hardware analysis

The extra hardware used in ECAP is a PTA, PBP, backward pointer, access counter per set (to check whether the set is lightly used or heavily used), and confidence value per block for CARP replacement policy. The detailed hardware analysis of ECAP in terms of the additional storage as well as energy and power consumption is shown in Tables 4 and 5. The main memory used in our experiment is of 4 GB. Hence, 32 bit address is used. On the basis of the conventional address split-up, 17 bits are reserved as tag bits per block and 9 bits are reserved for set index bits.

Experimentally, it has been found that ECAP performs well when each PTA index holds four MVs. Since each MV has 1 valid bit, 17 bits of tag, and 4 bits of direction flag, the total bits required for each MV is 22 bits. The backward pointer requires four bits of direction flag per cache block. As mentioned in Section 5, the optimal value of PBP size is 16 which requires around 5632 B. ECAP also stores a 4 bit value per set to check the access count within 1024 cycles.

When CARP is implemented in conjunction with ECAP, then an additional hardware overhead of 3 bits per block is used for the confidence value that is attached with each block. If LRU replacement policy is used, then the number of bits required for a four-way set-associative cache is 2 bits per block. Hence, the additional overhead required in ECAP with CARP replacement policy is $512 \times 4 \times 3$ bits$\simeq$768 B. This has a marginal impact on the hardware overhead in ECAP.

Table 4 shows that the additional hardware cost per tile for ECAP design increases by only 0.83% as compared with SCP. If 64 bit address is used, then ECAP has an additional hardware overhead of 1.47% per tile because of the increase in tag bits per MV. From Table 5, we can observe that the area and power consumption of ECAP increases minimally by 0.42 and 0.57% per tile as compared with SCP.

7 Related work

Existing works in prefetching focused on reducing the cache pollution, timeliness etc. of prefetchers [12, 15, 16, 37, 38]. However, these works are compatible with either a non-banked cache [15] or a banked cache with centralised access [12]. Prefetching in a distributed banked cache such as TCMP is still an active research area and very less works have been done in this regard. Since this paper focuses on distributed banked cache architecture, we summarise related work for such cache structures only.

Jorge *et al.* [11] proposed an adaptive controller ABS to prefetch blocks in the last-level cache. The controller controls the number of misses generated from each L2 cache bank when the prefetch aggressiveness varies. Since increase in miss count reduces system performance, ABS dynamically assigns different

aggressiveness value to each L2 bank. It uses a hill-climbing approach to control the aggressiveness of prefetchers running at each L2 bank separately.

Cireno *et al.* [13] proposed a prefetch system that uses a server and client for each core. On encountering a cache miss, the client feeds this information to a finite state machine. The finite state machine uses time-series prediction to predict the time when the core will generate a next request. The server prefetches block based on the prediction to increase the timeliness of the prefetcher.

Aziz *et al.* [39] proposed a balanced prefetch controller to prefetch blocks in L1 cache. The controller reduces miss penalty by controlling the aggressiveness of prefetchers in TCMPs. The author uses a cache sniffer and a directory sniffer at each tile. The directory sniffer samples out misses to predict the next cache request. The cache sniffer estimates the delay involved in fetching a block from lower level of memory to L1 cache. Both together controls the aggressiveness of prefetchers by reducing the network delays. Hence, the miss penalties in a core also reduce. This paper also emphasises on reducing miss penalty by placing prefetch blocks closer to the source core.

8 Conclusion

In this paper, we propose an energy-efficient prefetch block placement technique to reduce the negative impact of prefetching such as cache pollution which in turn increases the number of network packets in TCMP. For high-MPKI applications, ECAP leverages the less used cache space of remote caches that are running low applications. Hence, prefetch blocks of high application are placed in the remote caches. This results in virtually increasing the cache space of high applications, thereby reducing cache pollution. Experimentally, we found that ECAP achieves an overall reduction of 23.42% of L1 cache misses as compared with SCP. This results in reduction of around 32.93% of network packets. As a result of this, the packet latency of cache miss packets reduces by 25.34% which further reduce the AMAT by 23.56%, router and link power by 14.42 and 27%, respectively.

9 Acknowledgment

This research was supported in part by the Department of Science and Technology (DST), Government of India vide project Grant no. ECR/2016/000212.

10 References

[1] Sodani, A., Gramunt, R., Corbal, J., *et al.*: 'Knights landing: second-generation Intel Xeon Phi product', *IEEE Micro*, 2016, **36**, (2), pp. 34–46

[2] Bell, S., Edwards, B., Amann, J., *et al.*: 'TILE64 – processor: a 64-core SoC with mesh interconnect'. 2008 IEEE Int. Solid-State Circuits Conf. – Digest of Technical Papers, San Francisco, CA, USA, 2008, pp. 88–598

[3] Balasubramonian, R., Jouppi, N.P., Muralimanohar, N.: '*Multi-core cache hierarchies*' (Morgan and Claypool Publishers, San Rafael, CA, USA, 2011)

[4] Dally, W.J., Towles, B.: 'Route packets, not wires: on-chip interconnection networks'. Proc. 38th Design Automation Conf., Las Vegas, NV, USA, 2001, pp. 684–689

[5] Bjerregaard, T., Mahadevan, S.: 'A survey of research and practices of network-on-chip', *ACM Comput. Surv.*, 2006, **38**, (1)

[6] Kim, C., Burger, D., Keckler, S.W.: 'An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches', *SIGARCH Comput. Archit. News*, 2002, **30**, (5), pp. 211–222

[7] Jose, J., Nayak, B., Kumar, K., *et al.*: 'DeBAR: deflection based adaptive router with minimal buffering'. Proc. Conf. Design, Automation and Test in Europe, Grenoble, France, 2013, pp. 1583–1588

[8] Fallin, C., Nazario, G., Yu, X., *et al.*: 'MinBD: minimally buffered deflection routing for energy-efficient interconnect'. IEEE/ACM Sixth Int. Symp. Networks-on-Chip, Copenhagen, Denmark, May 2012

[9] Lee, J., Kim, H., Vuduc, R.: 'When prefetching works, when it doesn't, and why', *ACM Trans. Archit. Code Optim.*, 2012, **9**, (1), pp. 2:1–2:29

[10] Mittal, S.: 'A survey of recent prefetching techniques for processor caches', *ACM Comput. Surv.*, 2016, **49**, (2), pp. 35:1–35:35

[11] Jorge, A., Rubén, G., Pablo, I., *et al.*: 'ABS: a low-cost adaptive controller for prefetching in a banked shared last-level cache', *ACM Trans. Archit. Code Optim.*, 2012, **8**, (4), pp. 19:1–19:20

[12] Ebrahimi, E., Mutlu, O., Lee, C.J., *et al.*: 'Coordinated control of multiple prefetchers in multi-core systems'. Proc. 42nd Annual IEEE/ACM Int. Symp. Microarchitecture, New York, NY, USA, 2009, pp. 316–326

[13] Cireno, M., Aziz, A., Barros, E.: 'Temporized data prefetching algorithm for NoC-based multiprocessor systems'. 27th Int. Conf. Application-specific Systems, Architectures and Processors, London, UK, 2016, pp. 235–236

[14] Lai, A.-C., Fide, C., Falsafi, B.: 'Dead-block prediction amp; dead-block correlating prefetchers'. Proc. 28th Annual Int. Symp. Computer Architecture, Gothenburg, Sweden, 2001, pp. 144–154

[15] Srinath, S., Mutlu, O., Kim, H., *et al.*: 'Feedback directed prefetching: improving the performance and bandwidth-efficiency of hardware prefetchers'. 2007 IEEE 13th Int. Symp. High Performance Computer Architecture, Scottsdale, AZ, USA, February 2007, pp. 63–74

[16] Mehta, S., Fang, Z., Zhai, A., *et al.*: 'Multi-stage coordinated prefetching for present-day processors'. Proc. 28th ACM Int. Conf. Supercomputing, Munich, Germany, 2014, pp. 73–82

[17] Vangal, S.R., Howard, J., Ruhl, G., *et al.*: 'An 80-tile sub-100 W TeraFLOPS processor in 65 nm CMOS', *IEEE J. Solid-State Circuits*, 2008, **43**, (1), pp. 29–41

[18] Wang, H., Peh, L.-S., Malik, S.: 'Power-driven design of router microarchitectures in on-chip networks'. Proc. 36th Annual IEEE/ACM Int. Symp. Microarchitecture, series MICRO 36, San Diego, CA, USA, 2003

[19] Shang, L., Peh, L.-S., Jha, N.K.: 'Dynamic voltage scaling with links for power optimization of interconnection networks'. Ninth Int. Symp. High-Performance Computer Architecture 2003 HPCA-9 2003 Proc., Anaheim, CA, USA, February 2003, pp. 91–102

[20] Fallin, C., Craik, C., Mutlu, O.: 'CHIPPER: a low-complexity bufferless deflection router'. 2011 IEEE 17th Int. Symp. High Performance Computer Architecture, San Antonio, TX, USA, 2011, pp. 144–155

[21] Moscibroda, T., Mutlu, O.: 'A case for bufferless routing in on-chip networks'. Proc. 36th Annual Int. Symp. Computer Architecture, series ISCA '09, Austin, TX, USA, 2009, pp. 196–207. Accessed June 2009, available at http://doi.acm.org/10.1145/1555754.1555781

[22] Mittal, S., Vetter, J.S.: 'A survey of architectural approaches for data compression in cache and main memory systems', *IEEE Trans. Parallel Distrib. Syst.*, 2016, **27**, (5), pp. 1524–1536

[23] Binkert, N., Beckmann, B., Black, G., *et al.*: 'The gem5 simulator', *SIGARCH Comput. Archit. News*, 2011, **39**, (2), pp. 1–7

[24] Jiang, N., Becker, D.U., Micheologiannakis, G., *et al.*: 'A detailed and flexible cycle-accurate network-on-chip simulator'. Proc. Performance Analysis of Systems and Software, Austin, TX, USA, 2013, pp. 86–96

[25] Henning, J.L.: 'SPEC CPU2006 benchmark descriptions', *ACM SIGARCH Comput. Archit. News*, 2006, **34**, (4), pp. 1–17

[26] Zhang, C., McKee, S.A.: 'Hardware-only stream prefetching and dynamic access ordering'. Proc. 14th Int. Conf. Supercomputing, Santa Fe, NM, USA, 2000, pp. 167–175

[27] Seshadri, V., Yedkar, S., Xin, H., *et al.*: 'Mitigating prefetcher-caused pollution using informed caching policies for prefetched blocks', *ACM Trans. Archit. Code Optim.*, 2015, **11**, (4), pp. 51:1–51:22

[28] Seshadri, V., Mutlu, O., Kozuch, M.A., *et al.*: 'The evicted-address filter: a unified mechanism to address both cache pollution and thrashing'. Proc. 21st Int. Conf. Parallel Architectures and Compilation Techniques, series PACT '12, Minneapolis, MN, USA, 2012, pp. 355–366

[29] Beckmann, N., Sanchez, D.: 'Modeling cache performance beyond LRU'. 2016 IEEE Int. Symp. High Performance Computer Architecture (HPCA), Barcelona, Spain, March 2016, pp. 225–236

[30] Khan, S.M., Tian, Y., Jimenez, D.A.: 'Sampling dead block prediction for last-level caches'. 2010 43rd Annual IEEE/ACM Int. Symp. Microarchitecture, Atlanta, GA, USA, December 2010, pp. 175–186

[31] Faldu, P., Grot, B.: 'Leeway: addressing variability in dead-block prediction for last-level caches'. 2017 26th Int. Conf. Parallel Architectures and Compilation Techniques (PACT), Portland, OR, USA, September 2017, pp. 180–193

[32] Huh, J., Kim, C., Shafi, H., *et al.*: 'A NUCA substrate for flexible CMP cache sharing'. Proc. 19th Annual Int. Conf. Supercomputing, series ICS '05, Cambridge, MA, USA, 2005, pp. 31–40

[33] Deb, D., Jose, J., Palesi, M.: 'Performance enhancement of caches in TCMPs using near vicinity prefetcher'. Proc. 2019 32nd Int. Conf. VLSI Design and 2019 18th Int. Conf. Embedded Systems (VLSID), New Delhi, India, 2019

[34] Muralimanohar, N., Balasubramonian, R., Jouppi, N.: 'Cacti 6.0: a tool to model large caches', HP Laboratories, Chicago, IL, USA, 01 2009

[35] Kahng, A.B., Li, B., Peh, L., *et al.*: 'ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration'. Design, Automation Test in Europe Conf. Exhibition, Nice, France, April 2009, pp. 423–428

[36] Batten, C., Joshi, A., Orcutt, J., *et al.*: 'Building manycore processor-to-DRAM networks with monolithic silicon photonics'. Proc. 16th IEEE Symp. High Performance Interconnects, Stanford, CA, USA, August 2008, pp. 21–30

[37] Yedlapalli, P., Kotra, J., Kultursay, E., *et al.*: 'Meeting midway: improving CMP performance with memory-side prefetching'. Proc. 22nd Int. Conf. Parallel Architectures and Compilation Techniques, Edinburgh, UK, September 2013, pp. 289–298

[38] Jouppi, N.P.: 'Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers'. Proc. 17th Annual Int. Symp. Computer Architecture, Seattle, WA, USA, 1990, pp. 364–373

[39] Aziz, A., Cireno, M., Barros, E., *et al.*: 'Balanced prefetching aggressiveness controller for NoC-based multiprocessor'. 27th Symp. Integrated Circuits and Systems Design (SBCCI), Aracaju, Brazil, September 2014, pp. 1–7