Received November 2, 2019, accepted December 10, 2019, date of publication December 16, 2019, date of current version December 26, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2959840

A Reuse-Degree Based Locality Classifier for Locality-Aware Data Replication

QIANQIAN WU AND ZHENZHOU JI
School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China
Corresponding author: Qianqian Wu (wuqianqian@hit.edu.cn)

This work was supported by the National Natural Science Foundation of China under Project 61472100.

ABSTRACT The last level cache (LLC) in shared configuration is widely used in the tiled chip multiprocessors (CMPs), which reduces the off-chip miss rate but incurs the long on-chip access latency. The state-of-the-art Locality-Aware Data Replication (LADR) scheme provides an effective tradeoff between capacity and latency through an in-hardware structure named locality classifier. However, the best Limited$_3$ locality classifier (Limited$_3$) in LADR equally preserves locality information of 3 cores for all cache lines indiscriminately that is superfluous for some lines reused by less than 3 cores but incomplete for other lines reused by more than 3 cores, which not only wastes the storage space but also limits the performance improvement. In this paper, we propose a novel concept of Reuse-Degree (RD) for each LLC line, since the line is loaded into LLC, to represent the number of cores that have reused the line. Then, we divide cache lines into Not Reused Line (NRL, $RD = 0$), Single Reused Line (SRL, $RD = 1$) and Multiple Reused Line (MRL, $RD >= 2$) based on their RDs and find that a significant fraction of LLC lines are NRLs or SRLs at any time. Based on this observation, we design a Reuse-Degree based Locality Classifier (RD_LC) for LADR. Specifically, RD_LC decouples the locality classifier from the LLC tag array and introduces two kinds of locality information arrays, single locality information array (SLIA) and complete locality information array (CLIA). Besides, RD_LC allocates a locality information entry only for the reused cache lines (SRLs or MRLs) instead of all cache lines, and assigns an SLIA entry to SRLs and a CLIA entry to MRLs. Our proposal avoids a waste of the storage space and also maintains enough locality information for the accuracy of data replication decisions. Experimental results show that our RD_LC for LADR saves 51% of the storage overhead than that of the baseline Limited$_3$ locality classifier with a performance improvement and a network traffic reduction by 7.56% and 3.33 % respectively.

INDEX TERMS Chip multiprocessors (CMPs), last level cache (LLC), data replication, locality classifier, reuse-degree (RD).

I. INTRODUCTION

It is commonly believed that tiled chip multiprocessors (CMPs), which contain a series of identical tiles connected over a switched direct network, are becoming the most scalable and promising architectures for future many-core CMPs [1]–[3]. The last level cache (LLC) in tiled CMPs is equally decomposed into lots of slices physically distributed among all the tiles. Generally, the LLC slices can be organized as either private or shared. In private organization, each LLC slice is private to a processor core on the same tile, which provides lower on-chip access latency as the core can get data from the local LLC slice directly when L1 miss occurs. However, this organization incurs plenty of off-chip memory requests because it is unable to share the aggregated cache resources, and one cache block is permitted to have multiple replicas in LLC, occupying more effective cache capacity. With the continual enlargement of the program working set, shared LLC becomes more attractive than private LLC as it allows sharing cache resource and lowers off-chip miss rate. The main disadvantage of shared LLC organization is that each slice can be visited by all processor cores globally and the response time of cache request relies on the distance between the request core and the home LLC slice possessing the cache block. For future many-core CMPs, due to the long wire delay, the LLC on-chip access latency is likely to become unnegligible, which will results in overall performance degradation seriously.

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

The earliest study on shared LLC organization, namely Victim Replication (VR) [4], maintains shared features and resorts to private features by replicating victim lines of local L1 cache to the local LLC slice. So, VR achieves low off-chip miss rate and reduces on-chip access latency simultaneously. However, VR blindly replicates all victim lines to local LLC slices, resulting in the pollution of the LLC and the increase of off-chip miss rate. This limits the improvement of system performance. In order to overcome the shortcoming of VR, several improved data replication schemes [5]–[7] have been proposed. These schemes can selectively replicate cache lines controlled by some mechanism, but they either do not benefit from the replication of all types of data, or cause too much hardware overhead for guiding accurate data replication.

The Locality-Aware Data Replication (LADR) [7] policy controls data replication according to data locality. LADR introduces an in-hardware run-time Complete Locality Classifier (Complete) to track the locality information of all cores for each cache line in LLC used for guiding the replication decisions. Although LADR performs better than prior other data replication schemes, the hardware overhead caused by Complete is impractical, which prevents LADR from scaling to future many-core CMPs. The improved Limited$_3$ [7] Locality Classifier (Limited$_3$) in LADR that only tracks locality information of 3 cores is presented for alleviating the storage overhead of Complete. Nevertheless, Limited$_3$ couples equivalent hardware with all LLC lines indiscriminately for tracking locality information of 3 cores, which not only wastes storage space but also damages performance. On one hand, for the lines reused by only one core or even have not been reused, it is not necessary to reserve space for them to track locality information of 3 cores. On the other hand, for the lines reused by more than 3 cores, Limited$_3$ cannot maintain complete locality information so that it makes lots of inaccurate data replication decisions and degrades the performance.

In this paper, we define a novel concept of Reuse-Degree (RD) to denote the number of cores that have reused the line since it is loaded into LLC. Motivated by the observation in section IV-A, we propose a Reuse-Degree based locality classifier for LADR (RD_LC). we design RD_LC as an decoupled hybrid locality classifier, which allocates hardware space for LLC lines based on their RDs. Specifically, we decouple locality classifier from the LLC tag array and introduce two kinds of locality information arrays, single locality information array (SLIA) and complete locality information array (CLIA). Meanwhile, we allocate a locality information entry only for the reused cache lines instead of all cache lines, and assign an SLIA entry to the cache lines with a low RD of 1 and a CLIA entry to the cache lines with a high RD over 1. Our proposal avoids a waste of the storage space and also maintains enough locality information for the accuracy of data replication. Full-system simulations of 64-core tiled CMPs show that RD_LC saves the storage overhead, improves the performance and reduces the network traffic than the baseline Limited$_3$. Our proposed RD_LC mechanism is suitable for the real application scenario, that is, the number of cores in tiled CMPs will expand to a greater extent in the future (many-core CMPs), and the on-chip space will be very tight. Complete and Limited$_3$ add excessive storage overhead, which is not suitable for the many-core application scenario. In this scenario, our RD_LC mechanism introduces a new hardware structure with great care, avoiding unnecessary storage overhead and improving the performance.

The rest of this paper is organized as follows. Section II introduces some related work about previously proposed data replication schemes and decoupled structures. Section III gives a description of two kinds of locality classifier in LADR and analyzes the disadvantages of Limited$_3$. Section IV defines a novel concept of Reuse-Degree (RD) and designs a reuse-degree based locality classifier (RD_LC) for LADR. The experimental methodology and results comparing our RD_LC with the baseline Limited$_3$ on the PARSEC [12] and SPLASH-2 [11] benchmark suites are presented in Section V. Finally, the conclusion is summarized in Section VI.

II. RELATED WORK

A. DATA REPLICATION

There is a large quantity of studies addressing the long on-chip access latency problem for performance improvement in CMPs by replicating data to local LLC.

Victim Replication (VR) [4] replicates the L1 victim blocks to local LLC. On one hand, it permits L1 to get data from local LLC, therefore reduces the on-chip access latency. On the other hand, it replicates all L1 victims without control and the replicated local lines occupy abundant LLC space, which increases the local LLC off-chip miss rate. Thus, the blindly replication policy limits the performance improvement of CMPs. Adaptive Selected Replication (ASR) [5] controls data replication by estimating the benefit and cost of it and only allows replicating shared read-only lines. Although ASR relieves the space occupation in VR to some extent, it ignores the access character of a single cache line and cannot benefit from the replication of all types of lines, which results in restricted performance improvement. Nexus [18] adaptively chooses how many local LLC replicas to replicate on the entire chip, depending on the replication degree of the different workloads, but like ASR [5], this scheme only focuses on the replication of read-only cache lines, and it cannot achieve performance gains from the replication of the cache lines with other types. The difference with ASR [5] and Nexus [18] is that our proposed RD_LC focuses on the replication of all types of cache lines, so RD_LC can benefit more from replication than ASR and Nexus. Dynamic Reusability-based Replication (DRR) [6] manages data replication by the reusability of cache lines and only replicates data with high reusability. DRR improves the performance compared to VR and ASR. However, it needs to monitor reuse pattern at cache line granularity and causes too much storage overhead. Locality-Aware Data Replication (LADR) [7] introduces a run-time hardware locality classifier to track the locality information of all cores at the cache line granularity. Nevertheless, the first kind of locality classifier, called Complete, is impractical for the reason that it tracks locality information of all cores for each LLC line, which causes huge storage overhead. The improved Limited$_3$ [7] locality classifier in LADR saves the storage overhead by means of tracking locality information of 3 cores. However, Limited$_3$ still reserves equivalent space for all LLC lines even though some lines do not need locality information tracking, which wastes the hardware space. Moreover, it hurts the performance because the incomplete locality information impacts the accuracy of the replication decisions. Unlike DRR [6] and LADR [7], RD_LC solves the problem of excessive storage overhead by using a decoupled structure. In addition, RD_LC avoids performance loss in Limited$_3$ by tracking the complete locality information for some cache lines. Locality-Aware Data Access Control (LDAC) [17] attempts to further reduce storage overhead by only tracking locality information of 3 cores for a limited number of cache lines on the basis of the Limited$_3$ locality classifier in LADR. LDAC only focuses on the storage overhead problem in Limited$_3$, which not only fails to solve the performance loss problem of Limited$_3$, but also further reduces performance. Compared with LDAC [17], the RD_LC scheme of this paper does not directly reduce the number of cache lines that are tracked, but tracks the different locality information for cache lines based on their reuse-degree (RD), thereby simultaneously solving the problem of excessive storage overhead and performance loss in Limited$_3$. Besides, the CLCE [19] replication scheme is proposed to improve the cache space utilization of the 'in-network caching' in the information-centric network (ICN) architecture. Pyramid [20] is proposed as a decentralized utility- and locality-aware replication method to maximize the average available bandwidth and minimize the average latency in the P2P cloud storage systems. The application scenarios of CLCE [19] and Pyramid [20] are different from our RD_LC. However, the RD_LC strategy is designed to take full advantage of the on-chip LLC space to improve the performance of today's popular tiled CMPs.

B. DECOUPLED STRUCTURES

In addition to the performance improvement, some studies focus on storage space degradation in CMPs using the decoupled structures.

The Reuse Cache (RC) [13] adopts decoupled LLC cache structure to maintain performance using less storage capacity, which decouples the tag and data arrays and only store data for the lines that have been reused. SelectDirectory (SD) [14] applies the decoupled idea to the directory structure for reducing the hardware cost of directory. It decouples the tag array and metadata array in directory and only allocates a metadata entry for the share lines. The decoupled structure allows less data entries than tag entries, which is good for allocating unequal hardware space for different kinds of lines and improving the area efficiency.

Similar to RC [13] and SD [14], we also adopt a decoupled structure in our design. The difference between our proposed RD_LC and the above two decoupled designs is that they apply the decoupled idea to the LLC cache structure and the directory structure respectively, and we use the decoupled structure to design the locality classifier structure. In addition, RD and SD use the decoupled structures to reduce the storage overhead with little performance compromise. The decoupled locality classifier structure proposed in this paper not only reduces the storage overhead but also improves the performance.

C. COMBINATION

In this paper, we analyze the hardware overhead and performance problems resulting from the coupled structure of Limited$_3$ in the LADR [7] data replication scheme and take advantage of the decoupled structures [13], [14] to design a decoupled locality classifier for LADR, which introduces two kinds of locality information arrays and allocates appropriate storage space according to the reuse-degree (RD) of the cache lines. The detailed description of our design will be shown in the following section IV.

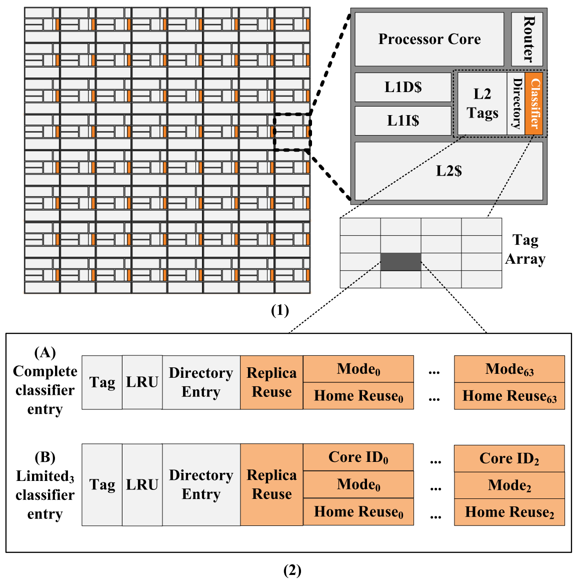

FIGURE 1. The 64-tiled CMP organization of LADR with locality classifier.

III. LOCALITY CLASSIFIER IN LADR

Tiled architectures are suitable for future many-core CMPs because of its high scalability to large core counts. Figure 1(1) shows the tiled CMP organization of LADR with 64 replicated tiles communicating through an on-chip network. Each tile is made up of a processor core, a router, private L1 instruction and data caches, and a slice of logically shared LLC (L2)[1] together with an in-cache directory and a locality classifier. For intelligently replicating data to local LLC slice, LADR introduces an in-cache structure named locality classifier to

[1]In this paper, we assume the tiled CMP has two levels of cache, so the L2 level cache corresponds to LLC.

track the locality information for all cache lines in LLC. The details of the Complete and Limited_3 (Limited_k, $k = 3$) classifier entry are presented respectively in Figure 1(2-A) and Figure 1(2-B).

A. COMPLETE LOCALITY CLASSIFIER

As we can see in Figure 1(2-A), the Complete entry is extended with additional bits including a Replica Reuse counter (RR) along with the Replication Mode bits ($RM_0 \sim RM_{63}$) and Home Reuse counters ($HR_0 \sim HR_{63}$) for all 64 cores on the basis of the directory entry. If the cache line is in home tile (called home line), the HR_i (e.g. HR_0) is a counter to calculate how many times the home line is accessed by core_i and will be incremented on every hit of the request from core_i. Correspondingly, the RM_i (e.g. RM_0) is designed as a identifier to make replication decisions for core_i and will be set true when the value of HR_i reaches the Replication Threshold (RT). On condition that the cache line is a replica in local tile (called local line), the RR is used to count the number of times that the local line is accessed by local L1 and is incremented on every local hit.

B. LIMITED_3 LOCALITY CLASSIFIER

The hardware overhead of Complete is so large that it is impractical for tiled CMPs, therefore, LADR proposes the optimized Limited_k in order to reduce the hardware overhead. But Limited_k hurts the performance due to the inaccuracy of locality information. Limited_3 achieves the best trade-off between performance and hardware overhead. As is shown in Figure 1(2-B), the entry in Limited_3 still maintains a RR, whereas, it only retains 3 groups of RM, HR and Core ID (CID) bits. Limited_3 merely tracks locality information for 3 of the 64 cores with the CID to identify the tracked core, and the replication decisions of the rest cores are made according to the tracked locality information.

C. DISADVANTAGES OF LIMITED_3

We summarize three disadvantages of Limited_3. (1) Adding unified hardware resources without distinguishing home lines and local lines will waste plenty of space. Because, RR is active only when the cache line acts as a local line, and on the other hand, the 3 groups of CID, RM and HR only work for a home line. (2) If the proportion of the home lines with an RD of 1 or 0 is large at any time, it also wastes great space to track locality information of 3 cores for all lines. (3) When a home line has been reused by more than 3 cores, especially by most or all cores, the accuracy of replication decisions and the system performance will be hurt.

IV. REUSE-DEGREE BASED LOCALITY CLASSIFIER FOR DATA REPLICATION

A. THE CHARACTERISTIC OF REUSE-DEGREE

1) THE CONCEPT OF REUSE-DEGREE

We define a novel concept of Reuse-Degree (RD) for each cache line in the shared LLC, which indicates the number of cores that have reused a particular cache line at home location since it is loaded into LLC. In this terminology, an RD of i ($i = 0, 1, 2, \ldots, 64$) means that the number of cores that have reused the cache line is equal to i, and specially, a cache line with an RD of 0 has not been reused by any core. Besides, we divide all cache lines in LLC into three groups based on their RDs: Not Reused Line (NRL), RD = 0; Single Reused Line (SRL), RD = 1; Multiple Reused Line (MRL), RD > 1.

2) REQUIREMENTS FOR REUSE-DEGREE BASED LOCALITY CLASSIFIER

We analyze the behavior of a set of multithread workloads selected in PARSEC [12] running in a 64-core tiled CMP with shared LLC slices. We monitor the RD by adding a counter for each cache line to record the number of cores that have reused the cache line. The simulation details can be seen in Section V-A. Figure 2 demonstrates the distribution of the number of cache lines in LLC with different RDs. As we can see, the cache lines with an RD of 0 (NRL) occupy about 72% on average (If a cache line is a local line, its RD is equal to 0 definitely). And the proportion of the lines with an RD of 1 (SRL) is 20% and the rest part of cache lines have an RD value over 1 (MRL). Therefore, over 92% of cache lines are unnecessary to reserve space for tracking locality information of 3 cores, so a lot of space for hardware will be saved. Because, the cache line with an RD of 0 does not need to track locality information for any core, and if a line's RD is equal to 1, one group of RM, HR, and CID is enough. Moreover, complete locality information is required to track for the home lines whose RDs are much larger than 3 so as to make accurate replication decisions. As a whole, it is necessary to design a Reuse-Degree based locality classifier for LADR. Because RD_LC adds different amount of hardware for cache lines based on their RDs, that not only decreases hardware overhead, but also improves performance compared to Limited_3.

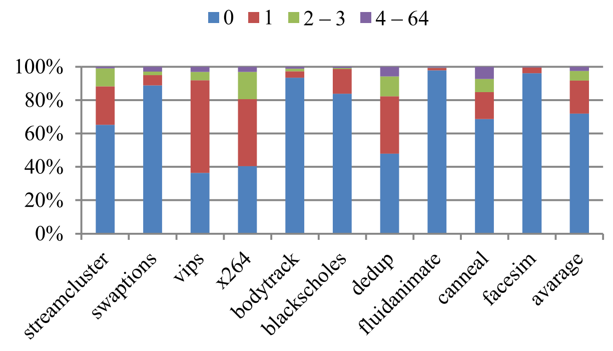

FIGURE 2. Distribution of the number of cache lines with an RD of 0, 1, 2-3, and 4-64.

B. THE OVERVIEW OF REUSE-DEGREE BASED LOCALITY CLASSIFIER

The state-of-the-art LADR policy adopts a run-time hardware locality classifier to track locality information, and replicates cache lines with high reuse to the local LLC based on the locality information. However, the locality classifier in LADR uses a coupled structure, and each tag entry is extended with the same hardware to keep locality information of 3 cores. The coupled and fixed structure between the tag array and locality information array not only wastes space but also damages performance.

According to the observation for the RDs of all LLC cache lines in section IV-A, the majority of cache lines are NRLs or SRLs, and a small portion of lines are MRLs at any time. In this section, we propose a Reuse-Degree Locality Classifier for Locality Aware Data Replication (RD_LC for short), which uses a decoupled and hybrid structure to track locality information. RD_LC decouples the tag array and locality information array, divides the locality information array into two categories: Single locality information array (SLIA) and Complete locality information array (CLIA) and allocates locality classifier entries based on the RDs of cache lines.

Aiming at the three disadvantages in Limited$_3$, RD_LC takes the following three actions. (1) We only need to store replica reuse information for local lines, and locality information for home lines. (2) If the home line is a NRL ($RD = 0$), we will not allocate a locality information entry for the line. In other words, only the lines that have been reused will correspond to locality information entries. If the home line is a SRL ($RD = 1$), we will allocate a single locality information entry in SLIA to track the core's locality information. (3) Otherwise, we will allocate a complete information entry in MLIA for the LLC home lines to track locality information of all cores. For simplicity, we maintain complete and accurate locality information for all the home lines with an RD value over 1, because this kind of lines occupy only a small portion and it will not cause too much hardware cost.

In the following two sections, we will present the details about the organization of RD_LC and how RD_LC works in our data replication scheme.

C. THE ORGANIZATION OF RD_LC

The organization of RD_LC is shown in Figure 3. As a whole in Figure 3(a), we add a slice of RD_LC for each tile in CMPs. The hybrid locality information arrays (SLIA and CLIA) in RD_LC are decoupled from TA, and the entries in SLIA and CLIA are connected with the entries in TA by two points in the opposite direction: Forward Pointer (FP) and Reverse Pointer (RP). Moreover, similar to TA, SLIA and CLIA are organized as set-associative structures. Specifically, the entries of TA, SLIA and CLIA are shown in Figure 3(b). Firstly, we add Valid (V), Single/Complete (S/C) and Replica Reuse/Forward Pointer (RR/FP) for each entry in TA. If the cache line is a local line, V and S/C are invalid, and RR/FP is used to store replica reuse information for it. For a home line, V represents whether the home line is a NRL, S/C is used to distinguish a SRL from a MRL, and RR/FP indicates the way of the locality information entry in SLIA or MLIA related to the home line. Because the entry in SLIA and MLIA can be got by FP rather than tag comparison, no extra latency

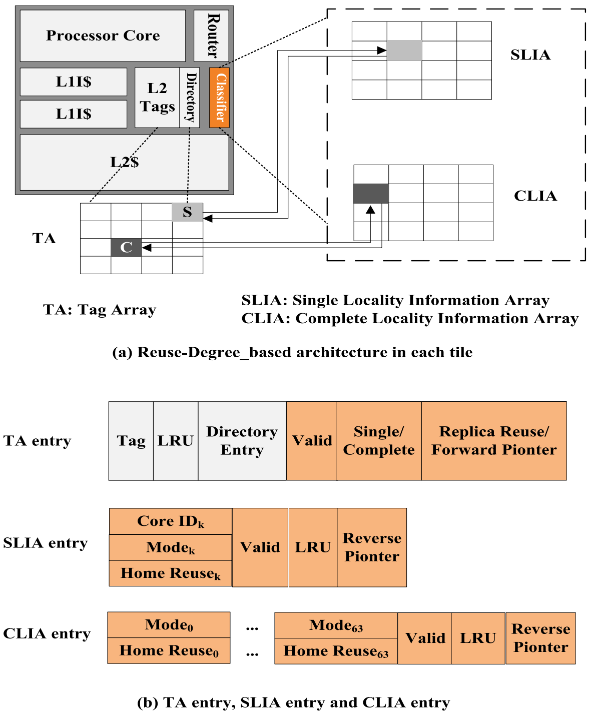

FIGURE 3. The organization of Reuse-Degree based locality classifier.

will be caused. When V is invalid, the home line is a NRL, therefore S/C and RR/FP are also invalid. When V is valid and S/C is invalid, the home line is a SRL, and RR/FP points to the locality information entry in SLIA. Correspondingly, when V and S/C are both valid, the home line is a MRL, and RR/FP points to the locality information entry in MLIA. Secondly, the SLIA entry stores locality information for a related SRL and it contains one group of RM, HR, and CID, Valid (V), LRU bit, and Reverse Pointer (RP). As in Limited$_3$, CID, RM, and HR in RD_LC represent the tracked Core ID, the Replication Mode bit and the Home Reuse Counter respectively. V signs whether the SLIA entry is valid, LRU bit is used for replacement policy, and the way number of the related SRL is indicated by the RP. Thirdly, RD_LC stores complete locality information for a related MRL in a MLIA entry. The group amount of RM and HR in MLIA entry is n, which is equal to the number of cores on chip. The MLIA entry also contains Valid (V), LRU bit, and Reverse Pointer (RP) with the same function as in SLIA entry.

Breaking the one-to-one mapping between the tag array and locality information array will allow fewer locality information entries than the tag entries. And the decoupled and hybrid structure will enable different cache lines have different amount of space for locality information storing. Therefore, RD_LC will save lots of storage space and simultaneously master more accurate locality information. Although the CLIA entry needs large space for complete locality information, its count is small on account of the small proportion of MRL. Thus, CLIA will not offset the spatial advantage of the decoupled and hybrid organization.

In order to demonstrate the area efficiency of our scheme, a comparison among the extra hardware structures and storage overhead required by Limited$_3$ and RD_LC can be found in section V-B(3).

D. THE OPERATION OF RD_LC

1) ACCESS REQUEST

On a request miss in L1 cache, it firstly requests its local LLC. If there is a local line in local LLC, the replicated line is sent to L1 cache directly and RD_LC increments its RR in TA. If the local LLC does not contain the request line, it forwards the request to the home LLC. When the home LLC misses the request line, it gets the line from main memory. Because the line is used for the first time, in other words, it is not reused by any core and is a NRL. RD_LC is not required to track locality information for any core and only allocates a tag entry in TA. What's more, V, S/C, and RR/FP are set to invalid. Then, the line is only sent to the request L1 cache and not replicated to its local LLC. On the other hand, if the request line exists in the home LLC, RD_LC permits several situations to happen depending on the V and S/C in TA. The pseudo code of the access request control algorithm based on V and S/C is shown in Figure 4.

(1) If V is invalid, the home line is a NRL so far. The line is reused for the first time by the request core and RD_LC needs to track locality information for the single core. RD_LC allocates a new entry in SLIA for the line for the reason that the line is just reused by only one core. In the SLIA entry, CID stores the ID of the request core; RM and HR is initialized to false and 0 respectively; and RP is set to the way of the related tag entry in TA. Simultaneously, the SLIA adopts LRU replacement policy, and the SLIA entry is promoted to the MRU position. Moreover, the V of the tag entry in TA is set to valid, which shows the home line is a SRL, and the FP is set to the way of the new SLIA entry.

(2) If V is valid and S/C is invalid, the home line is a SRL so far and FP points to a SLIA entry. RD_LC firstly gets the SLIA entry by FP and compares its CID with the ID of the request core (for example i). Then RD_LC carries out two different operations according to the comparison.

If the ID of the request core is equal to CID, the locality information stored in SLIA entry belongs to the request core and RD_LC updates the SLIA entry directly. The locality information is updated similar to the method in LADR. Specifically, if the RM is false, the HR is incremented and then if the HR is greater than or equal to the RT (for example 3), the RM is updated to true; otherwise, nothing is needed to do. What's more, the SLIA entry is promoted to the MRU position.

Otherwise, the home line becomes a MRL, therefore RD_LC needs to allocate a new entry in MLIA for the home line and initialize it. Then, the locality information of the original single core in SLIA entry is copied to the new MLIA entry and the locality information of the new request core is updated. Besides, the S/C of the tag entry in TA is set to true and

```
Access request algorithm
When the home line receives a request from core i:
// (1) NRL
 1: if V = invalid then
 2:     Allocate and initial a new entry in SLIA:
 3:         CID ← i; RM ← false; HR ← 0;
 4:         V ← valid; LRU ← setMRU; RP ← way.TA;
 5:     Update the entry in TA:
 6:         V ← valid; S/C ← invalid; FP ← way.SLIA;
// (2) SRL
 7: else if S/C = invalid then
 8:     Get the entry from SLIA by FP;
 9:     if CID = i then
10:         Update the entry in SLIA:
11:             if RM = false then
12:                 HR++;
13:                 if HR >= threshold then
14:                     RM ← true;
15:                 end if
16:             end if
17:             LRU ← setMRU;
18:     else
19:         Allocate and initial a new entry in MLIA:
20:             RM_{0-n} ← false; HR_{0-n} ← 0;
21:             V ← valid; LRU ← setMRU; RP ← way.TA;
22:         Copy the entry in SLIA to the new entry in MLIA;
23:         Update the entry in MLIA:
24:             if RM_i = false then
25:                 HR_i++;
26:                 if HR_i >= threshold then
27:                     RM_i ← true;
28:                 end if
29:             end if
30:         Update the entry in TA:
31:             S/C ← true; FP ← way.MLIA;
32:     end if
33:     Deallocate the entry in SLIA;
// (3) MRL
34: else
35:     Get the entry from MLIA by FP;
36:     Update the entry in MLIA:
37:         if RM_i = false then
38:             HR_i++;
39:             if HR_i >= threshold then
40:                 RM_i ← true;
41:             end if
42:         end if
43: end if
```

FIGURE 4. The pseudo code of the access request control algorithm based on V and S/C.

the FP is modified to point to the new MLIA entry. Finally, the old SLIA entry is deallocated.

(3) If V and S/C are all valid, the home line is a MRL so far and FP points to a MLIA entry. RD_LC still gets the MLIA entry by FP, and then updates the locality information of the corresponding core. Finally, the MLIA entry is promoted to the MRU position.

2) REPLACEMENT

As the system runs, more and more entries are needed in TA, SLIA and MLIA necessarily. All the three arrays face the situation to evict a victim entry in order to make room for the new entry. The pseudo code of the replacement algorithm is shown in Figure 5.

```
Replacement algorithm
// (1) Evict a TA entry
 1: if evicted entry = TA entry then
 2:   if line = home line then
 3:     if line = SRL/MRL then
 4:       Evict the linked entry;
 5:     end if
 6:   else
 7:     Back-invalidate the  L1 cache line;
 8:     if replica.state = M then
 8:       Send back the RR to the home LLC;
 9:     else
10:       Send back the data and RR to the home LLC;
11:     end if
12:     if home line = SRL/MRL then
13:       Update the relevant RM in SLIA or MLIA;
14:     end if
15:   end if
// (2) Evict a SLIA or MLIA entry
16: else
17:   V ← invalid; S/C ← invalid; FP ← invalid;
18: end if
```

FIGURE 5. The pseudo code of the replacement algorithm.

(1) If the TA needs to evict a victim entry, two different actions will be taken according to whether the evicted line is a home line or local line (replica). If the evicted line is a home line which is linked with a SLIA or MLIA entry, and the replacement of the TA entry will result in the eviction of the linked SLIA or MLIA entry. However, if the evicted line is a local line, the local LLC will send back-invalidation message to the L1 to invalidate the corresponding line. In addition, if the state of the local replica line is Modified (M), the local LLC will write back the data and RR to the home LLC, or the local LLC will only send back the RR. Then, when the home LLC receives the RR and is a SRL or MRL, it will update the replication decision (RM) according to the value of RR.

(2) If, a SLIA or MLIA entry is evicted, the V, S/C and FP in the linked TA entry will be set to invalid, and the corresponding cache line becomes to a NRL.

V. EXPERIMENT

A. EXPERIMENTAL METHODOLOGY

The verification work of optimization mechanisms in the actual hardware environment has a long verification cycle and is costly. Currently, most of the optimization mechanisms in the field of architecture research are performed by simulator simulation. We use gem5 [8], a modular discrete event driven simulator merged by the best aspects of the M5 [9] and GEMS [10] simulators, in full-system (FS) mode to evaluate our proposed RD_LC. Because we choose Limited$_3$ [7] for LADR as the baseline system, the experiment in this paper is set up as consistent as possible in LADR, such as processor core number, memory system, coherence protocol and on-chip network. The detailed memory system is modeled using the ruby modular in gem5 in which each core has its private L1I and L1D cache, and the LLC is equally divided into 64 slices that physically distributed among 64 tiles and logically shared by 64 cores. What's more, the relationship between the private L1 caches and the shared LLC is inclusive and the coherence among the private caches is maintained using a MESI protocol. The detailed basic system parameters and additional parameters of SLIA and MLIA in RD_LC are summarized in Table 1.

TABLE 1. Simulator configuration parameters.

Basic Parameters	
Cores	64 cores, TimingSimpleCPU/ Out-of-Order (O3)
ISA	Alpha
Cache block size	64 B
Cache hierarchy	Inclusive
L1 I/D cache	Private, 16KB, 4-way, LRU, 1 cycle
LLC (L2 cache)	Shared, 128KB/tile, 8-way, LRU, 6 cycles
Coherency protocol	MESI
Main memory	300 cycles
Network topology	8×8 mesh
Hop latency	6 cycles (2-router, 4-link)
RC_LC Parameters	
SLIA	544B, 16-way, LRU
MLIA	1600B, 16-way, LRU

As shown in Table 2, we measure the performance impact of RC_LC for LADR using the PARSEC [12] and SPLASH-2 [11] benchmark suits. The experimental results reported in the section V-B are collected from the Region-of-Interest (ROI) phase [15] of each multithread workload. For canneal, dedup, and ferret, the results shown in section V-B(2) and V-B(3) are from the simsmall input data set, while those shown in section V-B(4) are from simlarge input data set in PARSEC.

In the following section, we firstly analysis the additional access latency in RD_LC and the baseline Limited$_3$. Then, we compare the performance in term of execution time and network traffic of RD_LC and Limited$_3$. Moreover, we analyze the additional structures and storage overhead in Limited$_3$ and RD_LC.

B. EXPERIMENTAL RESULTS

1) ACCESS LATENCY

Both Limited$_3$ and RD_LC may introduce additional access latency for the local LLC tag access latency and the home

TABLE 2. Workloads and input size.

Benchmark suit	Workload	Input size
PARSEC	Canneal, Ferret Swaptions, Dedup	Simsmall
	Blackscholes Bodytrack	Simmedium
	Canneal, Dedup, Ferret	Simlarge
SPLASH-2	Lu_cb Radix Water_nsquared Water_spatial	1024*1024 matrix Radix = 1024 512 molecules 3375 molecules

LLC directory access latency. CACTI 6.5 [16] is used in this paper to model the access latency assuming 32nm technology.

On one hand, when a cache line is accessed as a local LLC replica, the RR bits will be accessed, which may increases the local LLC tag access latency. However, as shown in Table 3, the RR bits in Limited_3 and the RR/FP+S/C+V bits in RD_LC are accessed along with the original Tag+LRU bits in each LLC tag entry. Therefore, even considering the additional access latency introduced by Limited_3 and RD_LC, the local LLC Tag access latency still requires 1 cycle at 2GHZ clock.

TABLE 3. Local LLC tag access latency.

	Local LLC Tag access	Access latency (ns/cycle)
Local LLC Tag	Tag+LRU	0.269ns/ 1cycle
Limited_3	Tag+LRU+RR	0.269ns/ 1cycle
RD_LC	Tag+LRU+RR/FP+S/C+V	0.271ns/ 1cycle

On the other hand, when a cache line is accessed as a home LLC line, the locality information in Limited_3 is coupled with the directory sharer list and the locality information in RD_LC is stored in SLIA or MLIA which is decoupled from the LLC directory. Thus, when a home LLC line is accessed, the locality information in Limited_3 is accessed along with the directory sharer list and the locality information in SLIA or MLIA is accessed in parallel with the directory sharer list. As is showed in Table 4, the original sharer list access latency is 4 cycles, and considering the additional latency, the access delay in Limited_3 still requires 4 cycles. The access latencies of SLIA and MLIA are all 3 cycles which are hidden by the directory sharer list access latency (4 cycles). Of particular importance, according to observation in section IV-A(2), most LLC lines are NRLs ($\text{RD} = 0$), so in most cases the SLIA and MLIA will not be accessed and will not result in additional access latency.

Based on the above analysis, neither Limited_3 nor RD_LC introduces additional access latency for the local LLC tag access latency and the home LLC directory access latency.

TABLE 4. Home LLC directory access latency.

	Home LLC Directory access	Access latency (ns/ cycle)
Home LLC Directory	Sharer list	1.581ns/ 4cycles
Limited_3	Sharer list + Locality information	1.666ns/ 4cycles
RD_LC	Sharer list	1.581ns/ 4cycles
	SLIA entry	1.462ns/ 3cycles
	MLIA entry	1.471ns/ 3cycles

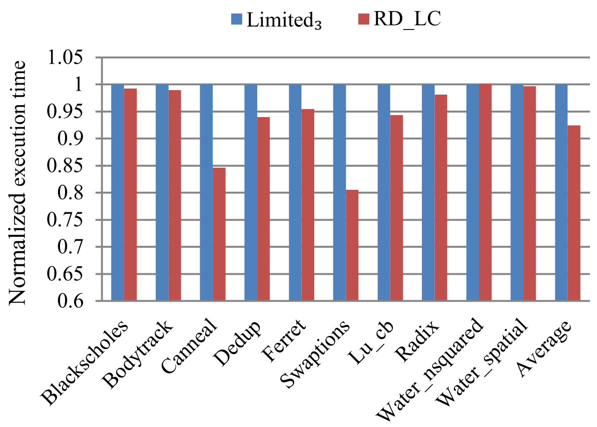

FIGURE 6. Normalized execution time to Limited_3.

2) EXECUTION TIME

Figure 6 shows the execution time of RD_LC normalized to the baseline Limited_3 [7] in LADR. In Figure 6, RD_LC improves the performance of almost all benchmarks compared to Limited_3 and the execution time of RD_LC is 7.56% less than Limited_3 on average. Because Limited_3 tracks locality information of only 3 cores for all lines indiscriminately while RD_LC adopts hybrid arrays and MLIA can stores complete locality information for MRLs ($\text{RD} > 3$) so that RD_LC makes more accurate decisions than Limited_3. So, RD_LC transforms more remote home LLC hits to local LLC hits, which reduces the LLC on-chip access latency and results in the overall performance improvement. Specially, the maximum performance gain happens at Swaptions benchmark with about 19.46% performance improvement. This is because Swaptions has relatively more MRLs, and Limited_3 cannot, but RD_LC can, makes accurate replication decisions for the extra cores based on their own locality information. However, RD_LC and Limited_3 have similar performance at the benchmarks Water_nsquared and Water_spatial because most of LLC lines in these benchmarks are NRLs or SRLs, which do not benefit from the MLIA structure in RD_LC. For this kind of benchmarks, although RD_LC cannot reduce the execution time, it saves the storage overhead by decoupling the SLIA from the TA which can be found in section V-B(3).

3) NETWORK TRAFFIC

Figure 7 depicts the on-chip network traffic of RD_LC normalized to Limited_3. Limited_3 cannot preserve complete locality information for MRLs, which will result in a certain amount of inaccurate replication decisions. So, when Limited$_3$ mistakenly replicates the cache lines with low reuse or excludes the replication with high reuse, the increased local LLC misses and the subsequent remote home LLC accesses will consume extra network traffic. RD_LC overcomes the shortcoming of Limited$_3$ by storing completing locality information in MLIA and has a network traffic less than Limited$_3$, 3.33% on average, as depicted in Figure 7.

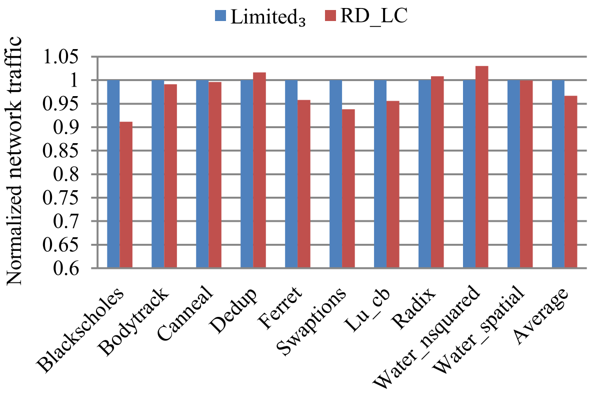

FIGURE 7. Normalized network traffic to Limited$_3$.

4) O3+SIMLARGE

The out-of-order (O3) CPU model is adequate for accurate architecture. Meanwhile, because modern workloads typically have a large working set, they need to be evaluated using the simlarge working set in PARSEC.

For a large working set, the number of NRLs (RD = 0) and SRLs (RD = 1) will be large due to frequent cache misses. In this case, Limited$_3$ stores the locality information of 3 cores for all cache lines, which will result in huge storage waste, as is detailed in section V-B(5). However, RD_LC does not need to store locality information for NRLs and only needs to store locality information of one core for SRLs, so that space can be saved without affecting performance. Moreover, although the number of MRLs (especially those with an RD > 3) is small, these cache lines are accessed frequently. Therefore, it means that storing only the locality information of 3 cores in Limited$_3$ will result in reduced the accuracy of data replication decisions and will degrade performance. RD_LC stores complete locality information for a small number of MRLs, which not only ensures the accuracy of replication decisions and system performance, but also does not introduce excessive hardware overhead. Figure 8 shows the normalized execution time and network traffic results with the O3 CPU model and the simlarge working set in PARSEC, indicating that RD_LC is better than Limited$_3$ in terms of execution time and network traffic.

5) STORAGE OVERHEAD

The additional structures and storage overhead in Limited$_3$ and RD_LC are shown in Table 5. In this paper, the particular configuration is a 64 tiled CMP, in which each tile has 32KB L1 caches, a 128KB LLC (L2 cache) slice, and a 64B cache block.

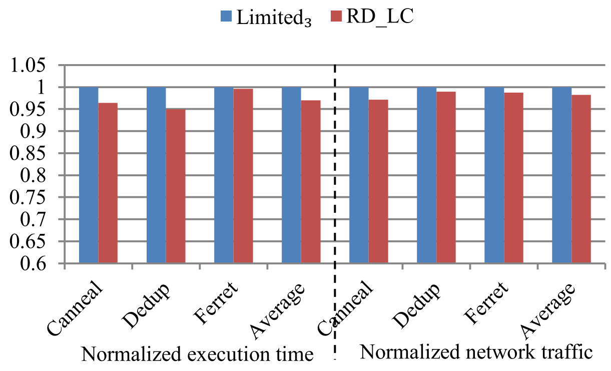

FIGURE 8. Normalized execution time and network traffic to Limited$_3$.

TABLE 5. The storage overhead in Limited $_3$ and RD_LC.

	Structure		Entry Size	Entries Number	Total Size	Overhead
Data	LLC		64 bytes	2 K	128 KB	0.0%
Limited$_3$	LLC TA	RR, CID, RM, HR	29 bits	2 K	7.25 KB	5.7%
RD_LC	LLC TA	V, S/C, RR/FP	6 bits	2 K	3.59 KB	2.8%
	SLIA		17 bits	256		
	CLIA		200 bits	64		

Limited$_3$ in LADR couples locality information on TA and does not distinguish home lines and local lines, so the coupled organization requires equivalent storage overhead for all LLC lines. Limited$_3$ adds a RR and three groups of CID, RM and HR for each line. The number of bits required by RR, CID, RM and HR is 2 bits, 6 bits, 1 bit and 2 bits respectively (assuming threshold is 3). Therefore, Limited$_3$ requires $(2+3\times(6+1+2))\times\frac{128}{64\times8} = 7.25$KB storage overhead. The total additional cost of Limited$_3$ is about 5.7% of the cache capacity.

RD_LC adopts the decoupled and hybrid locality information tracking organization. Increasing associations of SLIA and MLIA will be beneficial to their hit rate, but has a bad influence on storage overhead. Because the lengths of the FP in TA entry and the LRU bit in SLIA entry and MLIA entry are all equal to $log_2^{assoc\_SLIAorMLIA}$. As the same with association, the numbers of entries in SLIA and MLIA have the same impact on the hit rate and storage overhead. The result in section V-B(2) demonstrates that a way of 16 in SLIA and MLIA, and setting SLIA and MLIA to 256 and 64 entries respectively are enough for performance improvement. RD_LC only brings in 6 bits (1 bit for V, 1 bit for S/C, and 4 bits for RR/FP) for each TA entry, and adds SLIA and MLIA to track locality information. The length of a SLIA entry is 17bits (9 bits for one group of CID, RM, and HR, 1 bit for V, 4 bits for LRU, and 3 bits for RP). Differently, each MLIA entry contains 64 groups of RM and HR, 1 bit for V, 4 bits for LRU, and 3 bits for RP and needs 200 bits in total. In general, RD_LC requires 3.59KB extra costs and reduces the storage overhead by 51% compared to that of Limited_3.

VI. CONCLUSION

In this paper, we propose a novel RD_LC locality classifier for locality-aware data replication. We classify the LLC lines into three kinds: NRL, SRL, and MRL based on their RDs. And we design RD_LC as a decoupled and hybrid structure to track locality information for different kind of lines with two arrays: SLIA and MLIA. The decoupled structure avoids a waste of hardware space compared to the coupled structure in Limited_3. And the hybrid arrays permits tracking complete locality information for MRLs in MLIA. Our simulation results show that RD_LC achieves on average 7.56% and 3.33% of the performance improvement and the network traffic reduction compared to Limited_3. Moreover, the extra storage overhead for implementing RD_LC is 51% less than that of Limited_3.

REFERENCES

[1] S. Das and H. K. Kapoor, "Dynamic associativity management in tiled CMPs by runtime adaptation of fellow sets," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 8, pp. 2229–2243, Aug. 2017.

[2] S. Chakraborty and H. K. Kapoor, "Towards controlling chip temperature by dynamic cache reconfiguration in multiprocessors," in *Proc. Int. Conf. VLSI Design Int. Conf. Embedded Syst.*, Jan. 2017, pp. 75–80.

[3] S. Das and H. K. Kapoor, "Victim retention for reducing cache misses in tiled chip multiprocessors," *Microprocessors Microsyst.*, vol. 38, no. 4, pp. 263–275, 2014.

[4] M. Zhang and K. Asanovic, "Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors," in *Proc. Int. Symp. Comput. Archit.*, Jun. 2005, pp. 336–345.

[5] B. M. Beckmann, M. R. Marty, and D. A. Wood, "ASR: Adaptive selective replication for CMP caches," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2006, pp. 443–454.

[6] J. Wang, D. Wang, and H. Wang, "Dynamic reusability-based replication with network address mapping in CMPs," in *Proc. 17th Asia South Pacific Design Automat. Conf.*, 2012, pp. 487–492.

[7] G. Kurian, S. Devadas, and O. Khan, "Locality-aware data replication in the Last-Level Cache," in*Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2014, pp. 1–12.

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

[9] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, Jul./Aug. 2006.

[10] M. M. Martin, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, Nov. 2005.

[11] S. C. Woo, M. Ohara, and E. Torrie, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. Int. Symp. Comput. Archit.*, 1995, pp. 26–36.

[12] C. Bienia, S. Kumar, and J. P. Singh, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, 2008, pp. 72–81.

[13] J. Albericio, P. Ibáñez, and V. Viñals, "The reuse cache: Downsizing the shared last-level cache," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2013, pp. 310–321.

[14] N. Zhang, N. Zhang, and N. Zhang, "SelectDirectory: A selective directory for cache coherence in many-core architectures," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2015, pp. 175–180.

[15] X. Zhan, Y. Bao, and C. Bienia, "PARSEC3.0: A multicore benchmark suite with network stacks and SPLASH-2X," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 5, pp. 1–16, 2017.

[16] N. Muralimanohar and R. Balasubramonian, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2007, pp. 3–14.

[17] Q. Shi, G. Kurian, and F. Hijaz, "LDAC: Locality-aware data access control for large-scale multicore cache hierarchies," *Acm Trans. Archit. Code Optim.*, vol. 13, no. 4, p. 37, 2016.

[18] P.-A. Tsai, N. Beckmann, and D. Sanchez, "Nexus: A new approach to replication in distributed shared caches," in *Proc. 26th Int. Conf. Parallel Archit. Compilation Techn. (PACT)* , Sep. 2017, pp. 166–179.

[19] M. Bilal and S.-G. Kang, "A cache management scheme for efficient content eviction and replication in cache networks," *IEEE Access*, vol. 5, pp. 1692–1701, 2017.

[20] Y. Hassanzadeh-Nazarabadi and A. Küpçü, "Decentralized utility- and locality-aware replication for heterogeneous DHT-based P2P cloud storage systems," 2019, *arXiv:1907.11997*. [Online]. Available: https://arxiv.org/abs/1907.11997

QIANQIAN WU received the B.S. and M.S. degrees in computer science and technology from the Harbin Institute of Technology, where she is currently pursuing the Ph.D. degree in computer science and technology. Her research interests include computer architecture, high performance computing, and parallel computing.

ZHENZHOU JI received the Ph.D. degree in computer science from the Harbin Institute of Technology, in 2000. He is currently a Professor with the Harbin Institute of Technology. His research interests include computer architecture, parallel processing computer, and computer network security. He is the Vice Chairman of Computer Architecture at China Computer Federation (CCF).