Compressed Sharer Tracking and Relinquishment Coherence for Superior Directory Efficiency of Chip Multiprocessors

Wei Shu and Nian-Feng Tzeng

Abstract—To lower on-chip SRAM area overhead for chip multiprocessors (CMPs), this work treats a novel directory design which compresses present-bit vectors (PVs) by dropping "runs of zeros" commonly existing and lets PVs be transformed to their variations after sharer relinquishment for hashing alternative table sets to lift table utilization. Featured with relinquishment coherence and compressed sharer tracking (ReCoST), the proposed design attains superior directory efficiency and maintains "exact" directory representations, as a result of dropping abound long runs of zeros present in PVs. According to full-system simulation using gem5 for a range of core counts under PARSEC benchmarks, ReCoST is found to enjoy 3.21× (or 2.64×) more efficiency in directory storage than conventional bit-tracking directories (or the best directory known so far, called SCD) for a 64-core CMP under monotasking (or multitasking) workloads while ensuring execution slowdowns to stay within 2.4 percent (or 3.3 percent).

Index Terms—Chip multi-processors, coherence protocols, directory storage, hashing keys, multitasking, present-bit vectors

1 Introduction

A conventional directory-based chip multiprocessor (CMP) suffers from excessive directory area overhead when its core count increases, due to sharer tracking via present-bit vectors (PVs) maintained in the last level cache (LLC) of all cores [9]. The PV of each LLC block in such a CMP has the length equal to the core count, swiftly raising directory area overhead as the core count grows. Various techniques have been considered to contain directory storage for scalability improvement. Previous techniques all come at the expense of limiting the concurrent private caching degree, impeding performance due to network traffic elevation and/or increased hardware complexity.

This work pursues a novel directory design for its area overhead reduction, realized by two orthogonal mechanisms. First, compressing PVs by dropping long "runs of zeros" commonly present therein, so that resultant PVs can be kept in a hash-based, set-associative table, called the sharer pattern table (SPT), with reduced storage. Second, letting PVs with hashing conflicts be transformed to their variations after sharer relinquishment for hashing alternative SPT sets to lift table utilization. The width of SPT is shrunk to accommodate both compressed PVs and regular PVs uniformly. One unique feature of our design is that reducible patterns can still maintain "exact" sharer trackings for low coherent traffic and simple coherent logics. The proposed directory design is featured with relinquishment coherence and compressed sharer tracking (ReCoST) to exhibit superior directory efficiency.

Long runs of zeros frequently exist in PVs due chiefly to (1) a given data block is often accessed by only a fraction of active threads spawned from a task, limiting the program parallelism inherently [7], yielding abundant zeros in its associated PV, and (2) OS thread scheduling is domain-aware and affinity-cognizant (e.g., under the Linux kernel [5]). Threads spawned from a task in sequence are usually assigned to neighboring cores for good execution performance [8]. Private caches in those scheduled cores may hold shared data blocks simultaneously, but cores outside the dispatched domain never cache the data blocks, giving rise to long runs of zeros in the PVs. Load balancing also help to keep those threads of a given task within a domain, as the result of thread affinity. The insight of PVs likely to possess long runs of zeros signifies plentiful PV width reduction opportunities to yield reducible PVs, thus lowering directory area overhead. ReCoST also fits well to the contemporary multitasked scheduling domain partition approaches [7], [8], [9], [10], [11], [12], [13], [14] for directory storage overhead reduction.

ReCoST also explores an ingenuitive feature to resolve hashing conflicts. Upon a conflict, a PV is transformed into its variation by relinquishing one shared copy, with the corresponding bit in the PV reset. The PV variation is likely to hash an alternative SPT set. Hence, ReCoST can attempt multiple candidate sets for a given PV via relinquishent coherence support to lift SPT utilization. This way avoids an expensive provision for handling SPT overflows altogether.

ReCoST relies on two salient features to achieve superior directory storage efficiency, as follows:

- *Leverage the insight of scheduling*. OS kernel domain-aware and affinity-cognizant thread scheduling leads to abundant long "runs of zeros" that can be dropped for storage overhead savings while retaining "exact" sharer tracking for simple coherent logics and low coherent traffic without broadcasting.
- *Intelligent relinquishment coherence*. This distint feature permits our design to hash any PV to multiple SPT sets or even to avoid taking any SPT entry altogether by encoding a pattern, thus avoiding overflows.

We employ gem5 [3] with linux kernel to evaluate ReCoST under PARSEC benchmarks extensively, measuring the results for a range of core counts under both monotasking and multitasking. Our evaluation confirms that ReCoST under monotasked workloads on 64 cores enjoys 3.21× more directory storage efficiency than conventional bit-map directories while ensuring execution performance degradation to stay within 2.4 percent. Under multi-tasking workloads, simulation results reveal that the average execution slowdown is no more than 3.3 percent (or 2.3 percent) for equal (or weighted) core domain partitioning. When compared with the best directory design known so far (called SCD [15]), ReCoST is 2.64× more directory storage-efficient. Furthermore, it maintains "exact" sharer tracking to enjoy markedly lower invalidation-induced traffic (by a factor of up to 9.67× reduction for 64 cores) than its earlier inexact directory compression counterpart, SPACE [19].

2 Related Work

2.1 Directory Field Reduction

Directory storage reduction techniques fall into the following four categories, outlined in sequence next.

(i) *Compressed sharer tracking:* A compressed directory structure makes use of OS Distance-Aware Round-Robin (DARR) page mapping to achieve sharer tracking exactly [17]. However, DARR trades hardware complexity for storage efficiency. Inexact tracking allows each present bit or pointer to denote a cluster of cores, curbing directory storage overhead, irrespective of the core count. SPACE [19] achieved directory storage reduction through deduplicating present-bit vectors by keeping distinct vectors in a separate set-associative table.

• *The authors are with the Center for Advanced Computer Studies, University of Louisiana, Lafayette, LA 70504. E-mail: {wxs0569, tzeng}@louisiana.edu.*

Manuscript received 30 Aug. 2016; revised 26 Feb. 2017; accepted 30 Mar. 2017. Date of publication 24 Apr. 2017; date of current version 16 Oct. 2017.
Recommended for acceptance by R. F. DeMara.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2017.2698043

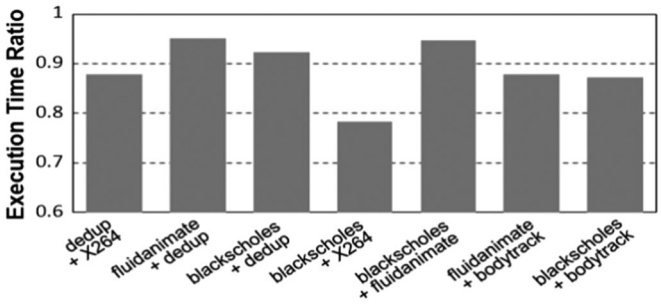

Fig. 1. Execution time results for two PARSEC benchmarks executed simultaneously on 64 cores, denoted by the ratios of those under domain partitioning against those without partitioning, where benchmarks are detailed in Table 2.

It suffers from inexact representations which often lead to unnecessarily high invalidation traffic. Imprecise tracking does not track all cached blocks in the LLC directory precisely for storage reduction [1], [2], [11].

(ii) *Hierarchical directory constructs:* Hierarchical directory constructs for cache coherence fit naturally to hierarchical architectures for large multiprocessors [10], but they impose additional lookups on the critical path, hence hurting execution performance. The scalable coherence directory (SCD) [15] follows a virtual hierarchical constructs for directory storage reduction by using multiple levels of present-bit vector representations.

(iii) *Inherent parallelism restriction:* Coherence domain restriction (CDR) [7] maintains the shared memory notion dynamically per task over a small number of system cores to bound directory area overhead. In fact, the restriction of inherent parallelism [7] can benefit our ReCoST, with its use of narrow SPT to hold reducible PVs for high storage savings potentially.

(iv) *Relaxed memory consistency:* A cache coherence design upon relaxed memory consistency model [13], [14], [15], [16], [17], [18], dubbed TSO-CC [6], has done away with sharer tracking. Such a design solely relys on self-invalidation and detection of potential sharers to satisfy the TSO memory consistency model lazily. However, TSO-CC has to maintain block states for proper block self-invalidation upon a read miss [6], complicating cache control logics.

2.2 Hash Collisions/Overflow Handling

A salient feature of ReCoST is the avoided provision of a hash table. ReCoST has its ingenuity that solves the overflow issue brought along by hashing. While many known directory structures employ multiple hash functions to improve table utilization and lower table overflows (for example, SCD [15] and others [12], [13], [14], [15], [16], [17], [18], [19]), overflows can never be avoided entirely under these structures. They usually either involve sharer eviction chains with potentially excessive latencies or may resort to an expensive provision for handling hash table overflows, such as fully associative TCAMs in HaRP [12]. Relying on sharer relinquishment, on the other hand, ReCoST requires no overflow provision while maintaining exact PV (present-bit vector) representations.

3 Proposed Approach

This work exploits the insights of (1) kernel domain-aware and affinity-cognizant scheduling for reducing PV width (without keeping long runs of zeros) and (2) pinning down tasks in partitioned domains for better performance. In addition, relinquishment coherence is followed to boost hash-based SPT utilization in two aspects under ReCoST: (1) locating alternative SPT sets for a given PV which is hashed to a full set, and (2) enhancing the chance of PV reduction by relinquishment coherence.

3.1 Motivation and Insights

3.1.1 Domain-Aware and Affinity-Cognizant Scheduling

CMPs cores are managed by the Linux kernel in a wrapped around circulary linked-list style. Upon thread spawning, the scheduler is responsible for finding one target core to run the newly spawned thread [5]. It is done in a load-balancing fashion, resorting to a local domain traversal algorithm to check the work queue of each core to choose the relatively idle core to assign the thread. The load-balancer usually selects the nearest neighboring core or a core within the same domain if its workload is under a threshold and bind the thread to that core [5]. Therefore, threads are commonly assigned to neighboring cores, within one domain.

From the aforementioned insights of spawn thread scheduling and binding processes, it is clear that the data blocks of a given application are *never cached* by cores outside the assigned domain, resulting in long runs of zeros in their associated PVs to permit compressed sharer tracking. Such insights are explored by ReCoST for the first time to lift cache directory storage efficiency.

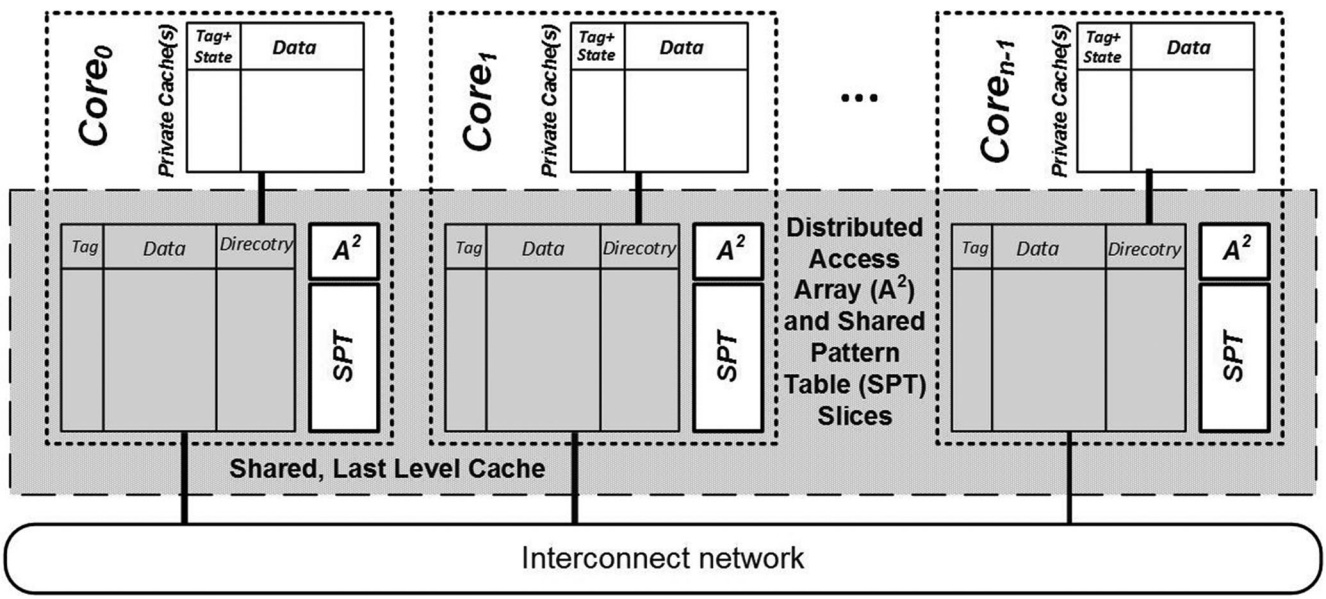

Fig. 2. CMP with shared directory-based LLC, where A^2 and SPT (in bold, solid boxes) are ReCoST-specific.

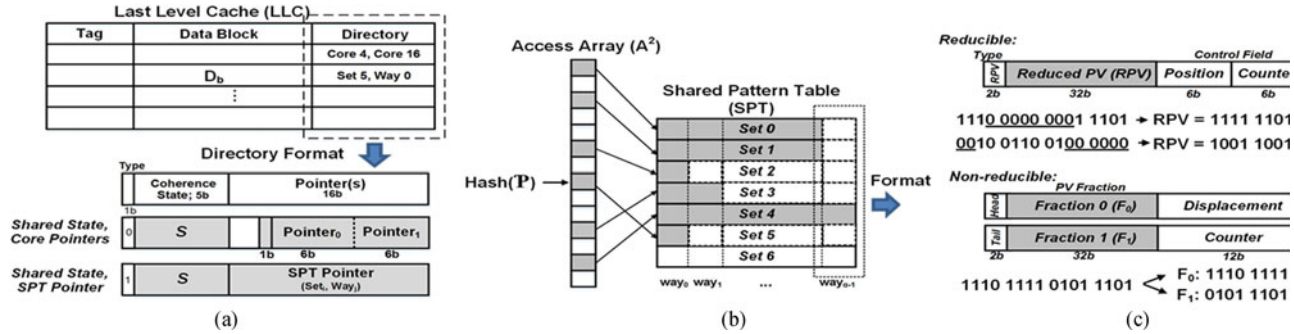

Fig. 3. (a) LLC directory format. (b) Hashing present-bit vector ($\mathcal{P}$) of data block D_b for SPT insertion, with table entry format shown in (c).

3.1.2 Multitasking and Execution Domain Partitioning

With a growing core count, CMPs may run multiple tasks at the same time (under multitasking) routinely. Under multitasking runs, it has been found that *binding those tasks statically to cores* or restricting the program execution domain [7] results in performnance speedups, because those binded threads of a task tend to enjoy plentiful hot hits in private caches and in TLBs and to suffer less from communication interference over the interconnect [7]. The results we observed reinforce that pinning separate tasks to partitioned core domains is generally advantageous, as shown in Fig. 1 for two concurrent banchmarks run on 64 cores. When the spawned 64 threads of each benchmark are pinned to one domain with 32 cores statically, an execution performance gain is evident to range from 4.9 to 21 percent over the unpinned scenario (where 128 spawned threads of two benchmarks share 64 cores). This task pindown by binding threads to core domains naturally yields *abundant reducible PVs*.

Note that reducible occupancy never reaches 100 percent. This is because OS kernel and housekeeping threads (e.g., PID scheduler, kthread, load-balancer, thread synchronizer, etc.) typically involve all cores, thereby yielding non-reducible PVs for the data blocks and instruction blocks of those threads. The directory thus should efficiently accommodate both reducible PVs and non-reducible PVs flexibly.

3.2 Proposed LLC Directory Structure

The overall LLC construct of a CMP with n cores is shown in Fig. 2, where each core has private caches and one slice of shared LLC. The set-associative SPT is fragmented into n slices. Each LLC cache block has a pointer, which points to one SPT entry if the type bit = "1", as depicted in Fig. 3a. The pointer field contains up to two core IDs, if its type bit = "0". One slice of the Access Array (A^2) is included in each core for enhancing hashing efficiency.

3.2.1 Reducible SPT

ReCoST employs one single function for hashing a PV to one SPT set. SPT entries are structured to accommodate both reducible PVs and regular (non-reducible) PVs efficiently, with their format depicted in Fig. 3c. Each entry contains four fields: Type, PV Fraction (PVF), Position, and Counter. An entry can be available for a reducible PV (RPV) or a non-reducible PV, which takes an entry pair. Under the reduction factor, γ, equal to 2, the length of an RPV is $n/2$ for a CMP with n cores. The starting position of the run of zeros for a stored RPVis recorded in its associated Position field. Counter field keeps track of the number of LLC data blocks which refer to this PV. In case the counter reaches its largest representable value, no additional LLC block may point to its associated PV any more, resorting to relinquishment coherence to search for another candidate set. An entry is released and becomes available when the associated counter equals 0.

Note that a PV is deemed wrap-around when specifying a run of zeros. The two examples shown in Fig. 3c have their runs of eight (8) zeros starting respectively at the positions of 3 and 10, for $n = 16$. Upon an insertion operation, a PV is transformed to an RPV by dropping a run of $n/2$ zeros to enter an available entry in the set hashed by the PV.

A regular PV takes up two entries, which are from separate sets in the same SPT slice (resident to a core, see Fig. 3c) for better access performance, with the head entry in a set, say Set SH, identified by hashing the PV, and the tail entry in another set chosen randomly from {ST | T = H+1, H+2, ...}. The head entry collapses both Position and Counter fields to serve as a pointer for storing the displacement of its paired tail entry. The tail entry combines both Position and Counter fields to serve as a counter, as illustrated in Fig. 3c.

3.2.2 Access Array for Hashing Enhancement

ReCoST adopts a hinge-like fashion to flexibly associate patterns with SPT sets through an **access array** (A^2), as illustrated in Fig. 3b. This fashion achieves exceedingly high SPT utilization. Like SPT, A^2 is sliced and distributed across all cores, for high scalability and avoiding a potential performance bottleneck.

Access Array (A^2) elements are initialized with nil. Every entry is set to point to one SPT entry when hashing a PV to its corresponding A^2 entry in the first time. Every entry is reset if its corresponding SPT set is freed. Let T and α be the total number of entries and the set associated degree for SPT, respectively, then SPT has $\Lambda = T/\alpha$ sets. In practice, the size of A^2 is chosen to be a prime number, R, which is larger than Λ (the number of sets in SPT), permitting dynmaically assigning sets to PVs. Such dynamic assignment greately enhanced SPT utilization. A^2 takes negligible storage ($\lceil lg\,\Lambda \rceil R$ bits in total) and can involve more than Λ elements inexpensively to result in nearly full SPT utilization. By choosing $R > \Lambda$ is a cost-effective way to (1) lower the probability of excessive conflict misses and (2) drop the mean number of PVs mapping to each set. It naturally makes coherent inquiries spread more evenly for better performance than what would otherwise be when PVs are hashed to reference SPT sets directly.

The use of A^2 does not lengthen read miss latencies and is off critical path of PV read operations, because the data blocks required by read misses that causing the associated PV bit altered can be furnished to those cores without waiting for the resulted SPT pointer change. This ensures the job execution slowdown on CMP with the ReCoST directory construct is usually negligible as detailed in Section 4.3.

3.3 Relinquishment Coherence

Under high SPT occupancy, a new PV is likely to hash to an SPT set with its entries all taken, suffering from a conflict miss. To deal with conflict misses and boost SPT utilization, ReCoST employs relinquishment coherence for identifying other candidate sets which have available entries. A PV hashes variations of its undersets to locate multiple candidate sets. Additionally, relinquishment coherence can transform non-reducible PVs to reducible ones by appropriately dropping certain existing

TABLE 1
Parameters of the Evaluation System

Core	3 GHz, X86 Out-of-Order; 3 GHz, ALPHA In-Order (explained in Section 4.1), 64 cores
L1 (Private)	32 KB I-Cache, 64 KB D-Cache, 2-way, 2-cycle latency, pseudo_LRU
L2 (Shared LLC)	4 MB/core, fully shared, 64 tiles 16-way, 14-cycle latency
Protocol	MESI, LLC inclusive, non-silent eviction
NoC	8×8 mesh, 2-cycle link latency
Memory	3 GB, 200-cycle latency

TABLE 2
A^2 and SPT Cofiguration

Parameter	$\xi = 100\%$	$\xi = 75\%$
SPT sets (Λ)	4,043	3,011
SPT associativity (α)	16	16
bits/SPT entry	46	46
A^2 entries (R)	4,919	3,761
Aug. factor (ρ)	1.22	1.24
bits/A^2 entry	12	12
total	368 KB	276 KB

sharer(s), before they are stored in SPT to lift storage utilization. This hashing overflow handling represents a unique and desirable feature for ReCoST, because it eliminates the need of a table provision (like extra fully-associative entries or a larger table that needs rehashing all patterns in existence) for dealing with table overflows.

ReCoST utilizes the same hash function keyed with variations of a PV at hand to hash alternative candidate sets. Given a PV, say $\mathcal{P}$, let the variation of $\mathcal{P}$ obtained by setting the nonzero bit at the ith position (with the leftmost bit as position 0) in $\mathcal{P}$ to "0" be denoted by $\mathcal{P}^1(\sim \mathrm{NZ}_i)$. Hashing $\mathcal{P}^1(\sim \mathrm{NZ}_i)$ is to yield one candidate set, which is likely to be different from the set determined by hashing $\mathcal{P}$. If there are β nonzero bits in $\mathcal{P}$, as many as β-1 additional candidate sets can then be identified via hashing $\mathcal{P}^1(\sim \mathrm{NZ}_i)$ for $0 \leq i < n$. Given a resulting PV (with Core 6 being the new sharer, denoted by NZ_6) under a 16-core CMP, $\mathcal{P} = 0010\ 01\underline{1}0\ 0100\ 0000$, for example, three additional candidate sets are identified by hashing $\mathcal{P}^1(\sim \mathrm{NZ}_2) = 0000\ 01\underline{1}0\ 0100\ 0000$, $\mathcal{P}^1(\sim \mathrm{NZ}_5) = 0010\ 00\underline{1}0\ 0100\ 0000$, and $\mathcal{P}^1(\sim \mathrm{NZ}_9) = 0010\ 01\underline{1}0\ 0000\ 0000$, besides the fully-occupied set identified by hashing $\mathcal{P}$ itself. If insertion such variation to candidate sets succeed, the corresponding data block cached in L1 of core *i* is invalidated.

During application execution, it is observed that shared data sets of a multi-threaded application are cached privately within domains and often involved cores in an order of increasing (or decreasing) IDs. Hence, the farthest non-zero bit in a run of "1" usually has held the associated block in its L1 for the longest time, among all cores that cache the block. Relinquishing the block from the private cache of that "farthest away" core is likely a desirable choice. In addition, choosing the target core for relinquishment in this way avoids "data thrashing". Given $\mathcal{P} = 0111\ 11\underline{1}0\ 0011\ 0010$ (with the read miss originated from Core 6, where Core 0 is denoted by the leftmost bit), the first alternative candidate set explored is given by the hash key of $\mathcal{P}_1(\sim \mathrm{NZ}_2) = 0011\ 11\underline{1}0\ 0011\ 0010$, provided that $\mathcal{P}$ and $\mathcal{P}_1(\sim \mathrm{NZ}_2)$ are hashed to different elements of A^2. The next alternative candidate set, if required, will be determined by $= \mathcal{P}_1(\sim \mathrm{NZ}_3) = 0111\ 11\underline{1}0\ 0011\ 0000$.

4 Evaluation and Results

Simulation evaluation has been conducted to assess ReCoST for core counts n = 64 using the PARSEC benchmark suite [3], with performance results compared with those of its conventional counterpart. ReCoST is also compared with earlier coherence directory organizations, namely, SCD (in terms of storage efficiency) [15] and SPACE (in terms of traffic and storage efficiency) [19].

4.1 Evaluation Methodology

Design Modeling. We use gem5 [3], an event-driven simulator, to model our proposed ReCoST directory-based design. The simulator runs in the full system mode, employing the Linux kernel 2.6, for evaluating CMPs with up to 64 nodes that execute benchmarks concurrently. The evaluation parameters assumed are listed in Table 1.

Workloads. We scale PARSEC [4] benchmarks blackscholes (BL), bodytrack (BT), canneal (CA), dedup (DU), ferret (FR), fluidanimate (FL), freqmine (FQ), streamcluster (SC), swaptions (SW), vips (VP) and x264 inputset from simsmall to simlarge accordingly for measuring our directory design. Every benchmark was set to spwan 64 threads for 64-core CMPs.

All workloads (except vips) were run on the X86 Out-of-Order cores, with vips on the ALPHA In-Order cores due to incomplete dependency in X86 diskimage. Six benchmarks (i.e., DU, x264, FA, BL, FA, BT) were selected to compose 7 mixes of workloads (listed along Fig. 7's *X*-axis and shown in the rightmost 8 bars), each comprising 2 benchmarks with similar ROI values. Every workload mix involves 128 threads (with 64 for every benchmark).

Baseline Configuration. We model conventional directory configuration as our baseline, in which each LLC block is associated with its home directory that tracks sharers using a present-bit vector, with its size equal to the core count. As LLC is distributed across 64 nodes, the home directory is also distributed accordingly. An L1 read miss or replacement invokes a message to request the data in its home LLC block, with a data reply sent back to the requesting core. This baseline configuration exhibits ideal performance but involves the largest directory storage than its counterparts.

ReCoST Latency Model and Configuration. An LLC block with the core pointer format is subject to a latency of 2 cycles for pointer encoding/decoding. If the directory field of an LLC block points to a local SPT entry, the latency of such SPT slice interrogation is 2 cycles, with 2 additional cycles required to get the paired entry in the same slice for a non-reducible PV. On the other hand, interrogation for the block pointer referring to a remote SPT slice involves an extra latency of one round-trip over the NoC plus associated traffic due to request/reply control messages. The upper bound of the total number of date blocks $\mathcal{F} = 64 \times 64\ KB/64B = 2^{16}$. Let SPT with $\mathcal{F}$ entries be referred to as 100 percent coverage of L1 data blocks in all cores together. In general, SPT can be sized with ξ-coverage, for $\xi \leq 100\%$, to involve ($\xi \times \mathcal{F}$) entries. SPT entries are set-associative, with each set pointed by an A^2 element, as shown in Fig. 3b. A^2 is augmented by *a factor of* ρ (meaning the number of A^2 entries = $\rho \times$(the number of SPT sets)) for better SPT utilization. Overall storage taken by both SPT and A^2 combined is listed in Table 2. In our evaluation, we gather and compare the performance results of ReCoST under two coverage levels, $\xi = 100\%$ and $\xi = 75\%$.

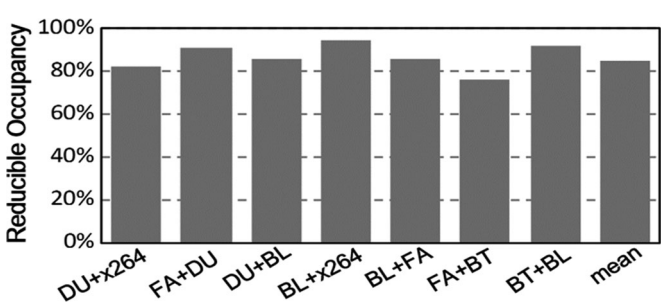

Fig. 4. Reducible occupancy in SPT for 64 cores under multitasking with threads pinned to cores.

TABLE 3
Directory Storage Overhead Under 100 Percent Coverage

Cores	ReCoST storage	SCD storage	SCD versus. ReCoST	Baseline versus. ReCoST	SPACE versus. ReCoST
16	2.9%	10.3%	3.55×	1.41×	∼1.00×
64	4.2%	11.1%	2.64×	3.21×	1.02×
256	4.9%	12.5%	2.55×	10.4×	1.08×
1,024	6.5%	15.8%	2.43×	30.9×	1.23×

4.2 Comparative Evaluation

Reducible Occupancy. Reducible occupancy is defined as the ratio of the number of reducible PVs to the total number of taken SPT entries. Reducible PVs during multitasking runs dominate those held in SPT, as illustrated in Fig. 4, where ξ equals 100 percent. Mean reducible occupancy over seven mixes of two benchmark codes examined reaches 82 percent. The outcomes of reducible PVs under monotasking can be found in [16].

Reducible occupancy under both monotasking and multitasking is expected to increase as CMPs scale up, since limited application parallelism and domain-aware OS scheduling will restrict the sharers of a given LLC data block, irrespective of the core count. However, reducible occupancy cannot reach 100 percent since application threads always co-exist with OS kernel (e.g., kthread, loadbalance, synchronization API etc., for hardware resource management) that typically involve all cores, thus yielding non-reducible PVs for those data blocks and instruction blocks pertinent to the kernel threads.

4.2.1 Storage Overhead

ReCoST storage overhead is calculated under $\xi = 100\%$. A^2 takes negligible storage (< 7.5 KB) to improve SPT utilization resulting from better hash distribution, and total storage taken by A^2 is accounted for by lowering the number of SPT sets down from 4,096 to 4,043 under 100 percent coverage. Other parameters of A^2 and SPT can be referred to Table 2.

Directory storage overhead is qualtified by the ratio of directory bits per block to the block size for a range of core counts, with its amounts listed in Table 3. As can be seen, directory storage overhead for ReCoST is low, ranging from 2.9 percent for 16 cores to 6.5 percent for 1,024 cores. Storage overhead amounts for the earlier 2-D SCD configuration [15] are included in Table 3 for comparison. ReCoST enjoys storage efficiency by more than 2.64 × (or 2.43 ×) over its SCD counterpart (which is about twice as efficient as the hierarchical directory, see [15]) for 64 cores (or 1,024 cores). When compared with its baseline counterpart, ReCoST enjoys 3.21× more storage efficiency for 64 cores and 30.9× more efficiency for 1,024 cores.

It should be noted that ReCoST achieves far better storage efficiency than SCD, despite that its every PV access involves simply one or two direct SPT entry fetch(es) using a SPT pointer or plus the pointer to its paired entry as opposed to 52 concurrent fetches to interrogate 52 candidate entries in the tag Zcache under SCD.

TABLE 4
Coherent Traffic and Execution Time Ratios of SPACE versus ReCoST Under 64 Cores

Bench-mark	Coherent Messages	Exec. Time	Bench-mark	Coherent Messages	Exec. Time
BL	1.18×	1.19×	FQ	2.94×	1.77×
BT	1.76×	1.88×	SC	1.39×	1.19×
CA	1.66×	1.24×	SW	4.29×	2.7×
DU	1.40×	1.74×	VP	9.67×	3.45×
FR	3.25×	2.13×	x264	1.88×	1.62×
FA	1.63×	1.55×	Mean	2.27×	1.86×

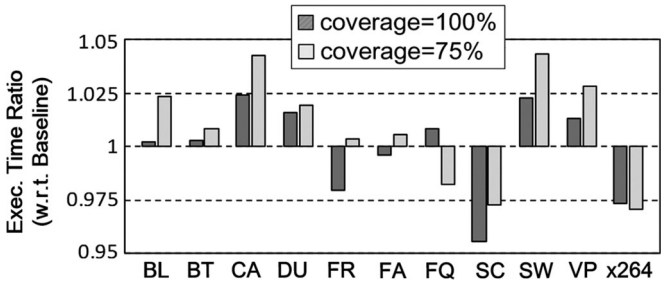

Fig. 5. Normalized execution times (w.r.t. those of the baseline) under monotasking for 64 cores under $\xi = 100$ and 75 percent.

Besides, ReCoST is found to exhibit negligible execution slowdown with respect to its baseline.

ReCoST exhibits a slighterly smaller mean footprint per PV held in SPT than its SPACE counterpart while have a much better execution time than SPACE. Total storage overhead of SPACE versus that of ReCoST is listed in the last column of Table 3, revealing that ReCoST exhibits higher storage efficiency as the core count rises (to 1.23 × for $n = 1024$). Moreover, SPACE holding PVs inexactly represented often overwhelm directory induced coherent traffic significantly, as will be explained in the next section.

4.2.2 Coherence-Induced Traffic

To better understand how coherent-induced traffic and execution slowdown are resulted from inexact tracking, we obtained total coherence-induced traffic and execution time respectively under SPACE and under ReCoST for 64 cores. Table 4 lists the ratio of the coherent-induced message count under SPACE to that under ReCoST. All benchmarks are seen to exhibit significant coherernt message reduction (by a factor up to 9.67× for 64 cores), which transforms to various execution efficiency gains (up to 3.45×) when compaired with those under SPACE. This is mainly due to inexact sharer tracking under SPACE that leads to considerable coherent traffic (for invalidation) unnecessarily. The inexact tracking degree worsens as execution progresses when more and more bits of sharing patterns are set to "1", elevating excessive invalidation-induced messages for long execution. In extreme cases, patterns with all nonzero bits exist and they require *broadcasting* to all cores upon invalidation, yielding coherent traffic surge (to 9.67× for vips (VP)). This excessive coherent traffic interfers with other traffic, resulting in marked execution slowdowns (by up to 3.45× when compaired with ReCoST).

In contrast, ReCoST always tracks sharers exactly, with SPT entries reclaimed and reused more efficiently. A conflicting PV either is hashed to an alternative candidate set after sharer relinquishment, or falls back to the core pointer format so as to avoid the use of any SPT entry. ReCoST heightens the entry reusable probability, while keeping the coherence-induced message count low. Because of its salient exact tracking feature, ReCoST

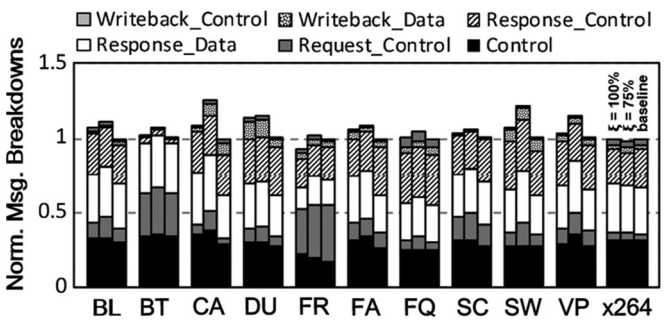

Fig. 6. Normalized NoC message breakdowns (w.r.t. those of the baseline shown in the rightmost bar of each workload) under monotasking for 64 cores, with $\xi = 100\%$ (or 75 percent) denoted in the leftmost (or middle) bar of each workload.

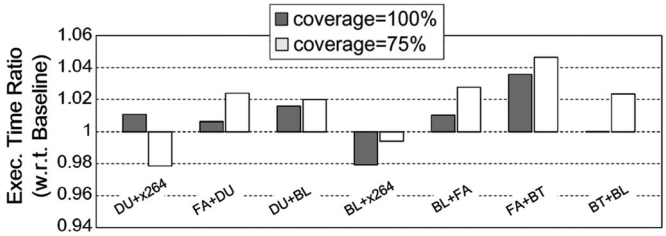

Fig. 7. Normalized execution times (w.r.t. those of the baseline) under multitasking for 64 cores.

maintians execution efficiency far better than SPACE (by an average of 1.86 ×), as can be observed in Table 4.

4.3 ReCoST Performance

4.3.1 Monotasking Performance

Execution Time. The execution time results of monotasking for 64 cores under two ξ values (100 and 75 percent) are depicted inside the left box of Fig. 5. It is clear that the ReCoST directory with $\xi = 100\%$ has negligible execution time degradation (by less than 1.3 percent) except canneal and swaptions. Canneal is known to have non-deterministic simulation input characteristics inherently, since its parallelization granularity is fine, its input data set is unbounded, and its data sharing rate is high [4]. Swaptions is likely to have its shared data stay in private caches very long, and any promotion for reducible PVs could prematurely relinquish active data blocks in private caches to cause slight performance degradation. If the SPT size is reduced by 25 percent ($\xi = 75\%$), execution time slowdowns are within 4.5 percent for all benchmarks, signifying high directory storage efficiency that renders execution performance insensitive to the SPT size.

Interestingly, three benchmarks (ferret, streamcluster, and x264) enjoy faster execution (by 2.1, 4.7, and 2.6 percent, respectively; see Fig. 5) under ReCoST than under its baseline counterpart for $\xi = 100\%$. They appear to benefit from relinquishment coherence, which relinquishes "oldest" sharers and may help the L1 replacement policy. Furthermore, these three benchmarks may not require their cached blocks in L1 to stay very long, so that preinvalidating L1 cached blocks heuristically under relinquishment coherence can help the L1 replacement policy through checking consecutive multiple cores, causing their overall execution time drop slightly when compared with those of the baseline.

NoC Traffic Results. Message count breakdowns is depicted in Fig. 6. All the benchmarks examined by our simulation exibit negligible traffic overhead, with the worst case of extra NoC traffic under ReCoST contained within 15 percent for $\xi = 100\%$ (DU). Extra traffic is due to relinquishment trials, slightly heightening coherent traffic, when SPT table utilization is high. This insiginificant traffic overhead is in sharp contrast to that under SPACE, where the amount of invalidation and ack messages can be up to 9.67 × higher (from Table 4), due to its inexact tracking.

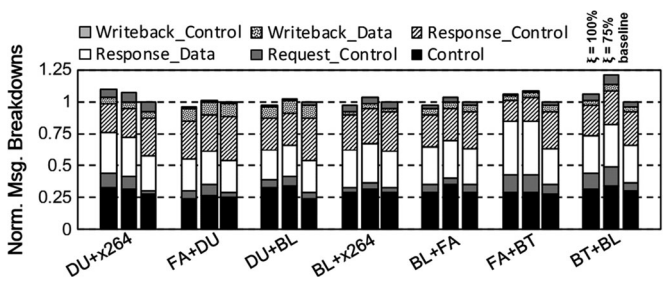

Fig. 8. Normalized NoC message breakdowns (w.r.t. those of the baseline shown in the rightmost bar of each workload mix) under mutitasking with equal domain partitioning for 64 cores, with $\xi = 100\%$ (or 75 percent) denoted in the leftmost (or middle) bar of each workload.

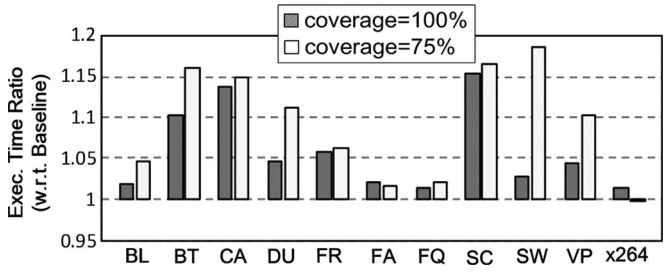

Fig. 9. Normalized execution results (w.r.t. those of the baseline) under monotasking for 16 cores and $\xi = 100\%$.

Benchmarks with lighter sharing and coarser parallelism usually exhibit lower traffic overhead. Benchmarks possesses high sharing and fine granularity (CA), experiencing more traffic overhead, due to excessive request_control messages.

4.3.2 Multitasking Performance

Multitasked applications exhibit better performance when pinned onto separate domains of cores, as explained in earlier sections. Normalized execution time results for multitasking workloads of 64 cores are depicted in Fig. 7. They reveal that the ReCoST directory with $\xi = 100\%$ experiences negligible execution time degradation (by less than 2.5 percent) for all workload mixes except that of FA + BT, which faces some 3.3 percent degradation. In contrast, the workload of BL + x264 enjoys an execution speedup (of some 2 percent) likely due to (1) better cache block replacement (by considering multiple neighboring cores) and (2) constructive invaliddation of private cache blocks, that result from relinquishment coherence as stated in Section 4.3.1. When the coverage level drops to 75 percent, the execution time of a given workload mix tends to grow, since fewer SPT entries then cause more relinquishment coherence traffic that lengthens its execution.

Execution performance is correlated to PV reducible occupancy depicted in Fig. 4. The execution results of FA + BT are worst among all workload mixes shown in Fig. 7, with its reducible occupancy (of 76 percent) lower than the average (of 86 percent), according to Fig. 4. This is due to the reason that both FA and BT employ and heavily invoke the same synchronization primitive codes [4]. Hence, LLC blocks that hold such primitive codes are likely to have non-reducible PVs as they are referred by cores in both domains, rendering low reducible occupancy in SPT. On the other hand, the mix of BL + x264 exhibits the best execution performance for $\xi = 100\%$, with its reducible occupancy topped at 93 percent from Fig. 4.

The breakdowns of NoC traffic are depicted in Fig. 8, where the results are normalized shown in the rightmost bar of each workload. It is apparent the total traffic increases are negligible under the coverage level of 100 percent, except for the mix of DU + x264 with a rise of some 8 percent. If the SPT size is shrinked to $\xi = 75\%$, traffic overhead is still contained except for the mix of BT+BL to

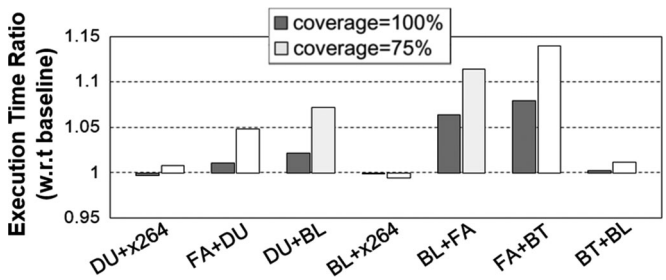

Fig. 10. Normalized execution results (w.r.t. those of the baseline) under multitasking for 16 cores and $\xi = 100\%$ (or 75 percent).

exhibit a noticeable surge (of 21 percent) due mostly to Request_Control and Control messages caused by relinquishment coherence. This is because data blocks under such a workload mix typically stay long in private caches, and any invalidation to those cached blocks caused by relinquishment coherence prematurely is to prompt extra access misses on those blocks subsequently, involving Request_Control and Control messages.

4.3.3 Scalability Consideration

ReCoST benefits more to CMPs with larger core counts, due to higher chances of long runs of zeros existing in PVs for storage efficiency improvement. If the core count is low, storage overhead incurred for pairing two SPT entries for holding one regular PV outweighs the limited potential savings due to reducible PVs, negatively impacting storage efficiency. Limited storage savings for a small CMP (e.g., with 16 or fewer cores) result from then scarce reducible PVs since PARSEC benchmarks have adequate sharing and fine-grained parallelism on such a CMP [4]. Meanwhile, directory access latency and traffic penalties inherent to ReCoST are then translated to more noticeable execution slowdowns (up to 18 percent for 16 cores), as demonstrated in Fig. 9. ReCoST can be noticeably inferior to those of its baseline counterpart in terms of execution performance for 16 cores, unlike what are depicted in Fig. 4, where the execution results of 64 cores are found to suffer from little slowdowns. ReCoST is clearly better suitable for a larger core count, exhibiting high scalability. In the case of multitasking, ReCoST again has lighter execution slowdowns for 64 cores than for 16 cores, as evidenced by contrasting Fig. 4 with Fig. 9. While slowdown ratios for 16 cores with 100 percent coverage under multitasking shown in Fig. 10 are up to 7.8 percent (or 13.9 percent under $\xi = 75\%$), the ratios are dropped down to no more than 3.8 percent (or 4.2 percent) for 64 cores as depicted in Fig. 7. ReCoST exhibits good scalability under multitasking.

Although not evaluated, ReCoST with its core count higher than 64 is expected to have even less performance degradation while delivering further improved storage efficiency. Under a larger reduction factor (say, $\gamma = 4$ or larger), ReCoST fragments every PV into γ equal pieces, as outlined in Section 3.4, to effectively support large-scale coherence. It benefits from the naturally restrained data sharing degree, due to not only the inherent parallism, but also the fine-grained domain-aware scheduling, which motivates this work.

5 Conclusion

This work has investigated an efficient on-chip CMP directory design that is featured with relinquishment coherence and compressed sharer tracking (ReCoST) for superior directory storage efficiency. By dropping long runs of zeros in present-bit vectors (PVs) and storing distinct PVs in a hash-based, set-associative sharer pattern table (SPT), directory storage overhead is reduced. Relinquishment coherence is adopted to boost SPT utilization while ensuring exact sharer tracking. ReCoST for a CMP with 64 cores is 3.21 $\times$ more storage-efficient than the conventional bit-tracking directory while ensuring negligible execution performance slowdowns under all PARSEC benchmarks examined for both monotasking and multitasking workloads. Compared with its earlier directory compression counterpart called SPACE [19], ReCoST achieves substantial reduction in invalidation-induced traffic (by a factor up to 9.67 $\times$ for 64 cores) due to its lower NoC message count. It is 2.64 $\times$ more directory storage-efficient than the best previous directory design known as 2-D SCD [15], exhibiting superior storage efficiency.

Acknowledgments

This work was supported in part by US National Science Foundation under Award Number: CCF-1423302.

References

[1] M. Acacio, J. González, J. García, and J. Duato, "A new scalable directory architecture for large-scale multiprocessors," in *Proc. 7th Int. Symp. High-Performance Comput. Archit.*, Jan. 2001, pp. 97–106.
[2] A. Agarwal, et al., "An evaluation of directory schemes for cache coherence," in *Proc. 15th Int. Symp. Comput. Archit.*, May.1988, pp. 280–289.
[3] N. Binkert, et al., "The gem5 simulator." *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.
[4] C. Bienia, et al., "The PARSEC benchmark suite: Characterization and architectural implications." in *Proc. 17th Int. Conf. Parallel Archit. Compilation Techn.*, 2008, pp. 72–81.
[5] D. P. Bovet and M. Cesati, *Understanding Linux Kernel*, 3rd ed. Springfield, MO, USA: O'Reilly, 2005, ch. 7, pp. 258–260.
[6] M. Elver and V. Nagarajan, "TSO-CC: Consistency directed cache coherence for TSO," in *Proc. IEEE 20th Int. Symp. High Performance Comput. Archit.*, Feb. 2014, pp. 165–176.
[7] Y. Fu, T. M. Nguyen, and D. Wentzlaff, "Coherence domain restriction on large scale systems," in *Proc. 48th ACM/IEEE Int. Symp. Microarchitecture*, Dec. 2015, pp. 686–698.
[8] B. Lepers, V. Quéma, and A. Fedorova "Thread and memory placement on NUMA systems: Asymmetry matters," in *Proc. USENIX Annu. Tech. Conf.*, Jul. 2015, pp. 277–289.
[9] M. Martin, M. Hill, and D. Sorin, "Why on-chip cache coherence is here to stay," *Commun. ACM*, vol. 55, pp. 78–89, Jul. 2012.
[10] M. Marty and M. Hill, "Virtual hierarchies," *IEEE Micro*, vol. 28, no. 1, pp. 99–109, Jan. 2008.
[11] S. Mukherjee and M. Hill, "An evaluation of directory protocols for medium-scale shared-memory multiprocessors," in *Proc. 8th Int. Conf. Supercomputing*, Jul. 1994, pp. 64–74.
[12] F. Pong and N.-F. Tzeng, "HaRP: Rapid packet classification via hashing round-down prefixes," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 7, pp. 1105–1119, Jul. 2011.
[13] A. Ros and S. Kaxiras, "Racer: TSO consistency via race detection," in *Proc. 49th IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2016, pp. 1–13.
[14] A. Ros and M. Acacio, "DASC-DIR: A low-overhead coherence directory for many-core processors," *J. Supercomputing*, vol. 71, no. 3, pp. 781–807, Mar. 2015.
[15] D. Sanchez and C. Kozyrakis, "SCD: A scalable coherence directory with flexible sharer set encoding," in *Proc. IEEE 18th Int. Symp. High Performance Comput. Archit.*, Feb. 2012, pp. 1–12.
[16] W. Shu and N. Tzeng, "Relinquishment coherence for enhancing directory efficiency in chip multiprocessors," in *Proc. 34th IEEE Int. Conf. Comput. Des.*, Oct. 2016, pp. 372–375.
[17] R. Simoni and M. Horowitz, "Dynamic pointer allocation for scalable cache coherence directories," in *Proc. Int. Symp. Shared Memory Multiprocessing*, Apr. 1991, pp. 72–81.
[18] X. Yu, et al., "Tardis 2.0: Optimized time traveling coherence for relaxed consistency models," in *Proc. 25th Int. Conf. Parallel Archit. Compilation Techn.*, 2016 , pp. 261–274.
[19] H. Zhao, A. Shriraman, and S. Dwarkadas, "SPACE: Sharing pattern-based directory coherence for multicore scalability," in *Proc. 19th Int. Conf. Parallel Archit. Compilation Techn.*, Oct. 2010, pp. 135–146.