

# Alternative Home: Balancing Distributed CMP Coherence Directory

Zhuo Huang<sup>\*</sup>, Xudong Shi<sup>+</sup>, Ye Xia<sup>\*</sup> and Jih-Kwon Peir<sup>\*</sup>

<sup>\*</sup>*Department of CISE*

*University of Florida, FL 32611, USA*

*{zhuang,yxl,peir}@cise.ufl.edu*

<sup>+</sup>*Google Inc.*

*Mountain View, CA 94043, USA*

*xdshi@google.com*

**Abstract** - The design of a scalable sparse coherence directory for future many-core CMPs faces a new challenge when the directory is distributed among multiple cores (homes) for maintaining cache coherence. Due to the uneven distribution of the cached block addresses among distributed homes, severe conflicts may occur on a few *hot-homes* where more than the average number of the cached blocks needs to be recorded. In this paper, we introduce the concept of *alternative home* to alleviate the hot-home conflict. The state and locations of a block can be recorded in one of two possible homes. Basically, the alternative home extends a direct-mapping for a unique home to a two-way mapping for locating an empty directory slot from two possible homes. The performance evaluations based on multi-threaded and multi-programmed workloads demonstrate significant reduction of L2 misses per instruction with the alternative home approach.

## I. INTRODUCTION

*Chip Multiprocessors (CMPs)* have become an industry standard for achieving a higher chip-level IPC (Instruction-Per-Cycle) [18,11]. Recently, the Intel's Tara-scale computing project pushes the number of on-die cores to tens or even hundreds [24,4]. One key requirement in many-core CMPs is their intelligent cache organization and cache coherency mechanism among many cores and cache modules [4]. We anticipate a class of CMP organization that consists of many cache modules, each local to a core. Multiple pairs of core and cache module are interconnected through a 2D-mesh network to form a tile-like interconnected organization [4]. In addition, the 3D memory+logic stacking technology will likely be mature in handling off-chip memory accesses [6]. Block-based interleaved memory modules are located directly above the associated core-cache pairs and can be accessed through the fast vertical interconnect.

In such an architecture with a large number of cores and cache modules, it becomes inherently difficult to locate a copy (or copies) of a requested data block and to keep them coherent. When a requested block is missing locally, one strategy is to costly search all other modules. To avoid broadcasting and searching all cache modules, directory-based

cache-coherence mechanisms were the choice for building scalable cache-coherent multiprocessors [9]. The memory-based directory [8] is very expensive and unnecessary since the cached block is only a small fraction of the total memory. A cache-based directory duplicates all individual cache directories and still requires to lookup all directories [23]. In a more efficient approach, a *sparse* directory uses a small fraction of the full memory directory organized in a set-associative fashion to record only those cached memory blocks [10,17]. One key issue is that the sparse directory must record all the cached blocks. Upon replacing a block in the directory, the respective block must be invalidated in all cache modules.

The design of a sparse coherence directory for future CMPs faces a new challenge when the directory is distributed among multiple cores. Based on the block address, a miss request can be forwarded to the *home* where the state and the locations of the block are recorded [15]. Upon a cache miss, the requested block can be fetched from the memory module located above the home. However, due to an uneven distribution of cached block addresses, the required size of the distributed directory at a home can vary significantly. When the directory size is partitioned equally among homes, insufficient directory space in some *hot-homes* where more than average cached blocks are recorded causes inadvertent block invalidations.

In this paper, we introduce a new distributed CMP coherence directory that uses an *alternative home* to alleviate the hot-home conflict. The state and locations of a block can be recorded in one of two possible homes. Each home location is determined by a different hashing function of the block address. Fundamentally, the alternative home extends a direct-mapping for a unique home to a two-way mapping for locating an empty directory slot from two possible homes. We consider different randomizations such as skew-associative [20] on selecting the alternative home. We also borrow a classical load balancing technique to place each missed block in the home with more empty space [16,3]. Performance evaluations based on multi-threaded and multi-programmed

workloads demonstrate significant reductions of block invalidations due to insufficient directory space using the alternative home approach. For SPECJBB2005, we observe that a reduction of 30-50% of the L2 miss per instruction is achievable in comparison with the regular direct-mapped single-home approach.

The paper organized as follow. In section II, we demonstrate the severity of unevenness of cached block distribution among the distributed home directories. In section III, we describe various distributed directory designs including the regular direct-mapped home, the randomized home selection, and the proposed alternative home. Section IV describes the performance evaluation methods, the workloads, and related simulation parameters. The performance results are presented and discussed in section V. The related work is given in section VI. Finally, we conclude the paper in section VII.

## II. BLOCK DISTRIBUTION AMONG HOMES

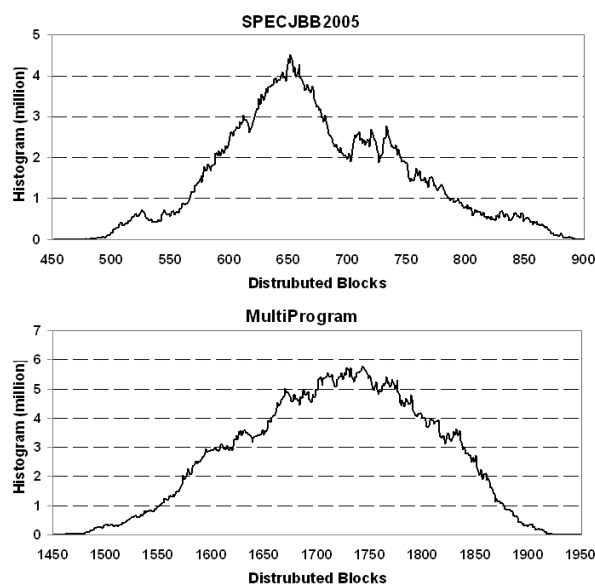


Fig. 1. Histogram of block distribution

In Fig. 1, the histograms of blocks distributed among homes are plotted for SPECJBB2005 and a multi-program workload with mixed SPEC programs (referred as MultiProgram). These workloads ran on a Simics 3.0 [14] whole-system simulation environment. We simulated a CMP with 64 cores, and each core has a private 128KB, 8-way, L2 cache with 64-byte blocks. A total of 8MB L2 is simulated. We simulated distributed coherence directories among 64 cores with infinite size. Each cached block has a unique home determined by the last 6 bits of the block address. Multiple counters are maintained to count the number

of cached blocks that each home must record when a block is loaded or removed from a core.

Fig.1 demonstrates severe disparity in the number of blocks distributed among homes. It also shows a significant difference in data sharing between SPECJBB2005 and MultiProgram. For SPECJBB-2005, the number of blocks that must be recorded in each home ranges from 450 to 900 with an average about 660. For MultiProgram, the range increases to between 1450 and 1950 with an average about 1740. Due to data sharing, the number of distinct blocks is much smaller in SPECJBB2005 than that in MultiProgram. In both workloads, the wide range in the number of recorded blocks in each home makes the directory design difficult. Severe block invalidations will be encountered unless a bigger directory is provided that can record extra 20-40% of the average number of cached blocks.

## III. DISTRIBUTED CMP COHERENCE DIRECTORY

In this section, we describe the fundamental CMP coherence directory design when an alternative home is available for each cached block.

### A. Randomized Home

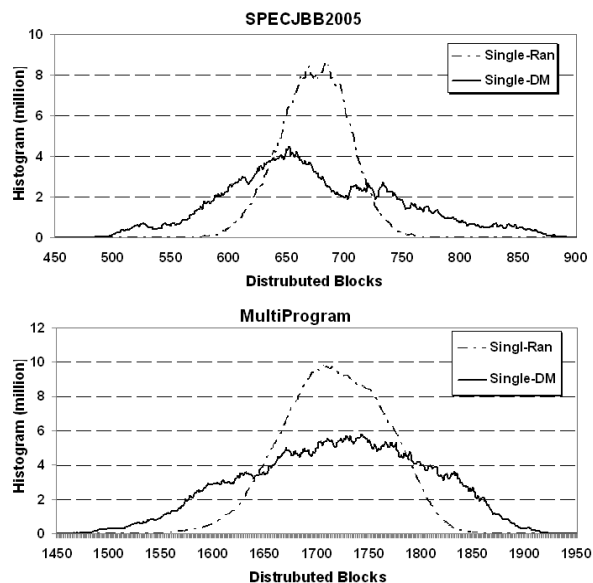


Fig. 2. Histogram of blocks distributed into a direct-mapped and a randomized coherence home

Consider a CMP with  $n$  cores, each hosting a part of the coherence directory. A straightforward home selection scheme, call *direct-mapped*, is to take the lower-order  $\log_2 n$  index bits of a block address to determine the home where the state and locations of the block is recorded. With the 3D memory-logic stacking technology, all memory blocks can be

allocated based on the  $\log_2 n$  index bits directly above the home directory, which can be fetched through the vertical interconnect when they are not present in the caches [6]. Although popular, this approach suffers the potential hot-home conflicts as shown in Section II.

To remedy the hot-home conflicts, an alternative approach is to randomize the home selection. One straightforward scheme is to select the home by exclusive-oring the lower-order  $\log_2 n$  bits with the adjacent higher-order  $\log_2 n$  bits of the block address [21]. The memory block allocation can be adjusted accordingly. In Fig. 2, we include the histogram from the randomized home selection (*Single-ran*) and the direct-mapped scheme (*Single-DM*). The results show that randomized home selection indeed improves the evenness of the block distribution. However, a home still must record a wide range of cached blocks.

### B. Alternative Home

The proposed alternative home records the state and locations of a cached block in one of two possible homes, determined by two different hash functions applied to the block address. With two homes, there are two key mechanisms for balancing the block distribution. The first is which of the two homes a block is actually placed into. For this, we can borrow ideas from an abstract load-balancing model of sending  $N$  balls to  $N$  bins. If, for each ball, we choose a random bin and place the ball into the chosen bin, the average number of balls per bin is clearly 1. However, the bin with the maximum number of balls has about  $\log N$  balls, which is larger than the average. On the other hand, if we choose two bins *randomly* for each ball and *place the ball in the bin with fewer balls*, then, the most loaded bin has about  $\log(\log N)$  balls, an exponential reduction in the maximum load. This drastic improvement of load balance using such a simple trick has been rigorously proven [3, 16].

The second key is the randomness of the hash functions used to select the two homes. In this paper, we consider a simple direct-mapped approach to determine the first home, called the *primary* home. For selecting the *secondary* home, we consider two approaches. The first approach is to simply organize the two homes as two-way set associative, where the secondary home can be decided by flipping the most significant bit of the  $\log_2 n$  index bits. Although not randomized, this simple two-way set-associative selection balances the blocks between a fixed pair of homes. The second approach is to select the secondary home using the straightforward randomization function as described in Section III.A.

The selection of homes is complicated by the fact that each distributed directory is implemented as set-

associative. Set selection within each home also plays a role in the number of cache invalidations. For set selection, we consider the same two hash functions, i.e. the direct-mapped and the straightforward randomization similar to that in the home selection.

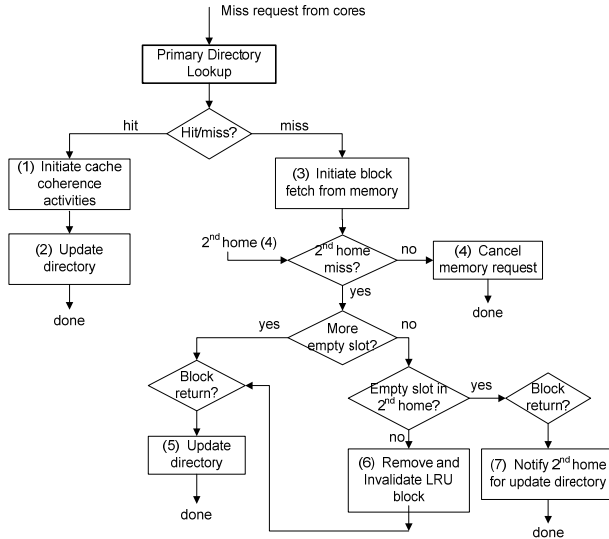
### C. Directory Lookup and Update

When a local cache miss occurs, the requested block address is forwarded to both homes. The state and locations of the block can be found in one and only one home directory if the block is located in one or more caches. Proper cache coherence activities can be initiated. When the block does not exist in either home directory, the requested block is missing from all caches. In this case, the primary home initiates a memory request to the next lower-level of the memory hierarchy. To avoid extra delay, the primary home can initiate the memory access once the block is missing in the primary home. An empty slot in either home directory must be picked to record the newly cached block. When no empty slot is available in either directory, one of the existing blocks must be removed. When a block is replaced from caches, a notification must be sent to both homes to update the directory. Fig. 3 shows the flow of the coherence operations in both the primary and the secondary homes.

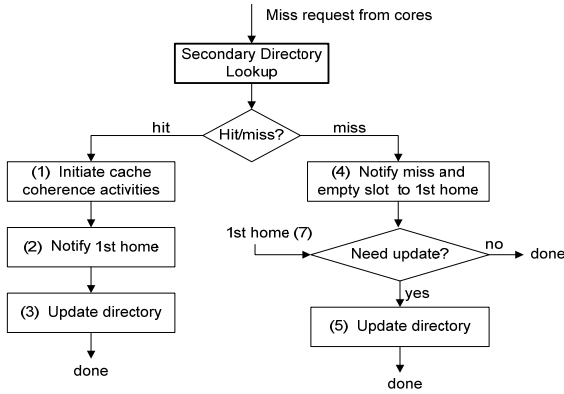
#### Primary Home:

- When a core's cache miss comes, the directory is searched. In case of a hit, the primary home triggers necessary coherence activities without delay and the directory is updated afterwards as indicated by boxes (1) and (2) in the figure.
- When the requested block is absent in the primary home, a memory request is initiated without delay for fetching the missed block (see (3)). Such a request is canceled if a notification from the secondary home indicates the block is in caches (see (4)). The fetched block, if received early, needs to wait for the notification from the secondary home to avoid any race condition.
- There are two cases if the notification from the secondary home also indicates a cache miss of the requested block. First, if the primary home has more empty slots in the target set, the state and locations of the missed block is recorded upon the return of the missed block (see (5)). Second, if the primary home set has less empty slots than the secondary home set for the missed block, the primary home returns an acknowledgement for the secondary home to record the missed block (see (7)).
- When none of the above two cases is encountered, there is no empty slot in either home. The primary home will remove the LRU block from the set

where the missed block is located. Invalidation is sent to caches to invalidate the LRU block (see (6)).



(a) Cache coherency operations in the Primary Home



(b) Cache coherency operations in the Secondary Home

Fig. 3. Coherency activities on primary and secondary homes

### Secondary Home:

- Upon receiving a cache miss, the directory is searched. In case of a hit, the secondary home triggers necessary coherence activities without any delay and the directory is updated afterwards. Meanwhile, a notification is sent to the primary home for canceling the memory request as indicated by boxes (1), (2) and (3) in the figure.
- When the requested block is not recorded in the secondary home, a notification of the miss along with available empty slots is sent to the primary home (see (4)).
- From the acknowledgement from the primary home, the secondary home records the state and locations of the missed block in its directory (see (5)).

## IV. EVALUATION METHODOLOGY

This section describes the simulation methods based on the Simics simulation tool, the simulated architecture parameters, and the selected multithreaded and multiprogrammed workloads.

### A. Simulator and Parameters

We use Virtutech Simics 3.0 [14], a whole-system execution-driven simulator, to evaluate an in-order x86 64-core CMP with Simics Micro-architecture Interface (MAI). To compare various coherence directory designs, we develop detailed cycle-by-cycle cache hierarchies, coherence directories, and their interconnection models. Each core has its own L1 instruction and data caches as well as an inclusive private L2 cache. The MOESI coherence protocol is applied to maintain L2 cache coherence through a distributed sparse coherence directory, which is evenly distributed among 64 cores.

Table 1. Simulation parameters

CMP and Caches
64 cores, 4GHz, in-order cores
L1-I/D: 8KB/8KB, 4-way, 64B line, write-back, 1-cycle
Private L2: 128KB, 8-way, 64B line, write-back, inclusive, 15 cycles, MOESI protocol
Remote L2: 35 cycles + network latency (see next page)
DRAM access: 225 Cycles + network latency
Request/response queues to/from directory: 8 entries
Coherence Directory
Main directory: Distributed 64 Banks, MOESI
Request/response queues to/from each core: 8 entries
Primary directory access latency: 10 cycles
Remote block access latency: 10 additional cycles
Interconnection Network
Network topology: 2-D 8*8 Mesh
Point-to-point wire delay: 1 cycle
Routing delay: 1 cycle per router
Message Packing/Unpacking delay: 1 cycle
Message Routing: Route-ahead to destination

When a requested block is not in the local L2 module, the request is routed to the corresponding coherence directory home (or homes) based on the block address and the directory organization. If the block is not in any of the CMP caches, the block will be fetched from the associated DRAM module.

Each core has one request queue for sending requests to the network and one incoming request queue for receiving requests from the network. Each core is attached to a router, which routes traffic to the four directly-connected routers. Our simulator keeps track of the states of all the requests, queues and routers. The delay for each request is carefully calculated by enqueueing, dequeuing, router stages,

routing conflicts and directory/remote L2 cache/main memory access, etc. We assume the point-to-point wiring delay between two routers is 1 cycle and the routing takes only 1 cycle at each router [12]. For simplicity, we simulate a simple route-ahead scheme such that each message is routed from the source to destination atomically. We do take the conflicts along the path into consideration that may lengthen the routing delay. Given a light traffic load, <15% for SPECJBB2005 and <5% for MultiProgram for the alternative home scheme, this simple routing strategy is reasonable. Table 1 summarizes a few parameters in our simulation.

### B. Workload Selection

SPECJBB2005 (java server) is a java-based 3-tier online transaction processing system. We simulate 64 warehouses. We skip first 5000 transactions, and then simulate 250 transactions after warming up the structures with 250 transactions. We also simulate a mixed SPEC2000 (Mcf, Parser, Twolf, and Vpr, Ampmp, Art, Mesa, and Swim) and SPEC2006 (Astar, Bzip2, Gcc, Gobmk, Libquantum, Mcf, Sjeng, and Xalan) applications, each with 4 copies. To alleviate perturbations among simulations of multiple directory configurations, we run multiple simulations for each directory configuration of each workload and inserted small random noises (perturbations) in the latency of main memory access. However, we still experience noticeable differences due to different instruction executions among simulated directory configurations. To make a fair comparison, we decide to collect an execution trace based on a direct-mapped single home with infinite directory and use the trace to compare various distributed directory designs.

## V. PERFORMANCE RESULT

Five CMP coherence directory organizations including direct-mapped single home (*single-DM*), randomized single home (*single-ran*), set-associative two homes (*2home-2way*), randomized two homes with fully-associative directory (*2home-ran-full*) and randomized two homes with random set selection (*2home-ran-set*) are compared. The set selection within each home for the first three organizations is direct-mapped. For *2home-ran-set*, the set selection in the primary home is direct-mapped while it is randomized in the secondary home; both use the adjacent higher-order bits next to the bits that are used for home selection. *2home-ran-full* is included in the evaluation to serve as a performance upper bond. The block distribution, the hit/miss improvement, and the invalidation traffic will be presented and compared.

### A. Cached Block Distribution

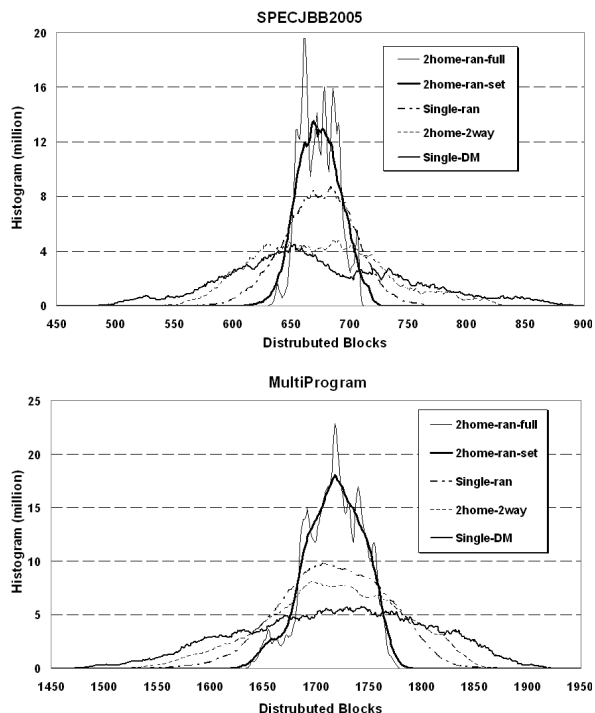


Fig. 4. Histogram of block distributions of five distributed coherence directories

The histograms of cached blocks distributed among 64 cores for the five simulated coherence directory organizations are plotted in Fig. 4. Clearly, *2home-ran-set* shows a more even block distribution than that of *single-DM*, *single-ran* and *2home-2way* for both workloads. With randomized set selection, the evenness in block distribution of *2home-ran-set* closely matches to that in *2home-ran-full*, which requires an unrealistic fully-associative directory. *2home-2way* does not help much for distributing the block more evenly. This is due to the fact that *2home-2way* has the two home restricted to a fixed pair. Effectively, it is equivalent to a single home using a directory with doubled set-associativity. *Single-ran* can balance the blocks better than that of *2home-2way*.

### B. Cache Misses and Invalidation Traffic

The evenness of the cached block distribution is reflected in cache invalidations and L2 cache misses. Fig. 6 shows the normalized L2 cache misses per instruction for *single-DM*, *single-ran*, and *2home-ran-set* under four directory sizes. The normalization is with respect to *single-DM* with the smallest directory. From the results for SPECJBB2005, we can make two important observations. First, as expected, *2home-ran-set* has the fewest L2 misses per instruction followed by *single-ran*, and then by *single-DM*. Comparing

with *single-DM* and *single-ran*, the improvement in L2 misses per instructions in *2home-ran-set* is very significant with about 30-50% and 9-12%, respectively. Second, the directory size plays an important role in the L2 misses per instruction. L2 misses per instruction increase substantially for small directory sizes. However, *2home-ran-set* can compensate the reduction of directory size. For example, *2home-ran-set* with a 576-entry directory reduces the L2 misses per instruction compared with the *single-DM* with 768-entry directory. The improvement of MultiProgram is not as significant, but still follows the same trend; *2home-ran-set* outperforms *single-DM* and *single-ran* by about 2-5%.

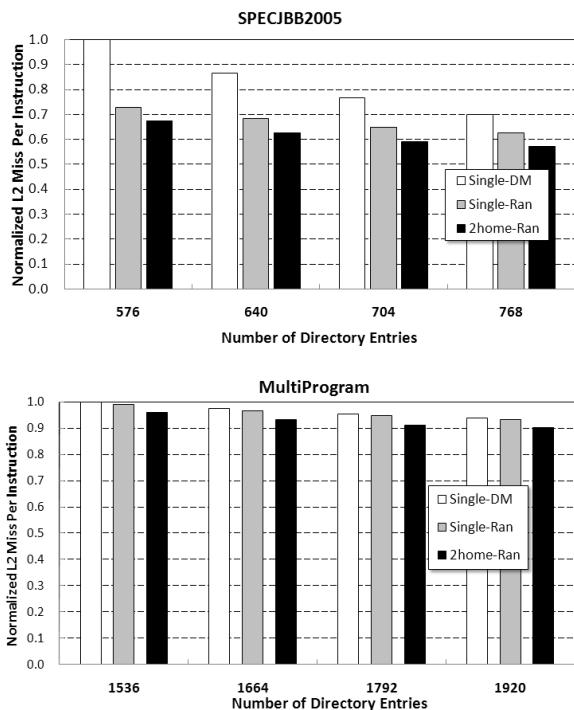


Fig. 5. L2 cache miss comparison

In Fig. 6, we show the total cache invalidations under *single-DM*, *single-ran*, and *2home-ran-set* with four coherence directory sizes. The invalidations are further categorized into the normal coherence invalidations due to updates and the invalidations due to insufficient directory space. We can observe that for SPECJBB2005, a majority of the cache invalidations are due to block updates, while for MultiProgram, the invalidations are mainly due to the directory size constraint. The three directory schemes have substantial differences in the amount of invalidations caused by insufficient directory size. With randomized two homes and randomized sets in each home, *2home-ran-set* produces a much smaller amount of cache invalidations than that of *single-DM*, and *single-ran*. This smaller

amount of invalidations is the primary reason for lowered L2 misses per instruction as described in Fig. 5.

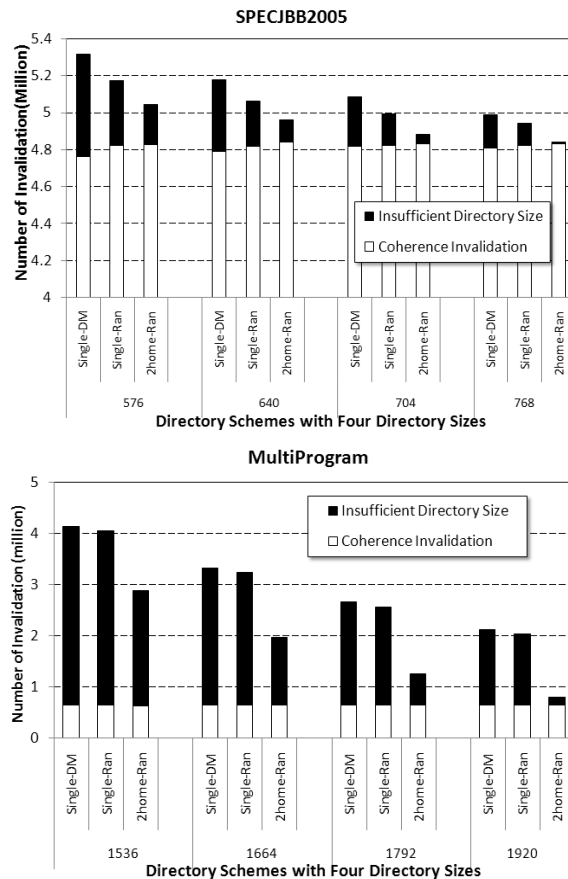


Fig. 6. Comparison of cache invalidation

It is important to point out that due to data sharing, SPECJBB2005 requires the directory size much smaller than the total L2 cache sizes. In this study, we consider the costly full presence bits in each directory entry to record all the sharers. The needed directory size increases significantly if the sparse pointers are used to record wide sharers using multiple directory entries [13]. Notice also that for MultiProgram, sizeable invalidations are still experience for *single-DM* and *single-ran* even when the directory size is close to the L2 cache size. This is due to the set conflicts within each home.

## VI. RELATED WORK

There has been a long history in designing directory-based cache coherence mechanisms for shared-memory multiprocessor systems [2]. The sparse directory approach uses a small fraction of the full memory directory to record cached memory blocks [10,17]. Studies have shown that the number of entries in such a directory must be significantly larger than the total number of cache blocks to avoid inadvertent cache invalidations. In a recent virtual

hierarchy design [15], a 2-level directory is maintained in a Virtual Machine (VM) environment. The level-1 coherence directory is combined with the L2 tag array in the dynamically mapped home tile located within each VM domain. No inclusion is maintained between the home L2 directory and all the cached blocks. Any unresolved accesses will be sent to the level-2 directory. If the block is predicted to be on-chip, the request is broadcast to all cores.

There have been many works in alleviating conflict misses [1,2,5,7,22]. One interesting work is the skewed cache [22] which randomized the cache index by excluding-oring the index bits with adjacent higher-order bits. The V-way cache [20] alleviates the conflict by doubling the cache directory size. It may not solve the hot-set problem since the expanded directory entries in the hot sets can still be occupied. The adaptive cache insertion scheme [19] dynamically selects blocks to be inserted into the LRU position, instead of the MRU position for handling cache capacity problem for a sequence of requests with long reuse distances that cannot fit into the limited cache by the LRU replacement policy.

In future many-core CMPs with 3-D DRAM-logic stacking technology, it is efficient to interconnect many cores and cache modules in a tiled structure using a 2-D mesh network while placing the DRAM modules right above the processor cores for fast accesses [4]. The coherence directory can be distributed among the cores in line with the memory block allocation. To our knowledge, this is the first work to consider an alternative home in implementing a distributed sparse coherence directory. This alternative home solution can be complimentary to the virtual hierarchy design [15] for alleviating home conflicts at their first-level coherence directory.

## VII. CONCLUSION

In this paper, we describe an efficient cache coherence mechanism for future CMPs with many cores and many cache modules. We argue in favor of a directory-based approach. However, the design of a low-cost coherence directory with small size and small set-associativity for future CMPs must handle the hot-home conflicts when the directory is distributed to all cores. We introduce and evaluate an alternative home approach that allows each cached block to be recorded in one of two possible homes. By examining two possible homes randomly and selecting the one with more empty space, the cached blocks can be more evenly distributed among all homes. Comparing with the traditional single-home approach, the proposed approach shows significant reduction in block invalidations and in cache misses per instruction.

## REFERENCES

- [1] M. E. Acacio. "A Two-Level Directory Architecture for Highly Scalable cc-NUMA Multiprocessors," *IEEE Trans. on Parallel and Distributed Systems Vol. 16 (1)*, Jan. 2005.
- [2] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz. "An Evaluation of Directory Schemes for Cache Coherence," *Proc. of the 15th Int'l Symp. Computer Architecture*, May 1988.
- [3] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced Allocations," *SIAM Journal on Computing*, vol. 29, no. 1, pp. 180-200, Feb. 2000.
- [4] M. Azimi, et al, "Integration Challenges and Tradeoffs for Tera-scale Architectures," *Intel Technology Journal*, vol. 11(3), Aug. 2007.
- [5] B. Beckmann and D. Wood, "Managing wire delay in large chip-multiprocessor caches," *Proc. of the 37th Int'l Symp. on Microarchitecture*, Dec. 2004.
- [6] B. Black et al, "Die Stacking(3D) Microarchitecture", *Proc. of the 39th Int'l Symp. on Microarchitecture*, Dec. 2006
- [7] F. Bodin and A. Sez nec, "Skewed Associativity Improves Performance and Enhances Predictability," *IEEE Trans. on Computers*, 46(5), May 1997.
- [8] L. M. Censier and P. Feautrier, "A New Solution to Coherence Problems in Multicache Systems," *IEEE Transactions on Computers*, c-27(12):1112-1118, Dec. 1978.
- [9] D. E. Culler, J. P. Singh, and A. Gupta. "Parallel Computer Architecture: A Hardware/Software Approach," *Morgan Kaufmann Publishers, Inc.*, 1999.
- [10] A. Gupta, W.-D. Weber, and T. Mowry. "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes," *Proc. of Int'l Conf. ICPP '90*, Aug. 1990.
- [11] Intel Core Duo Processor: The Next Leap in Microprocessor Architecture. *Technology@Intel Magazine*, Feb. 2006.
- [12] A.Kumar, L. Peh, P. Kundu, and N. K. Jha. "Express Virtual Channels: Towards the Ideal Interconnection Fabric". *Proc. of the 34th Int'l Symp. on Computer Architecture*, June 2007.
- [13] D. J. Lilja, and S. Ambalavanan, "A Superassociative Tagged Cache Coherence Directory," 1994 IEEE int'l Conf. on Computer Design, Oct. 1994.
- [14] P. S. Magnusson et al. "Simics: A Full System Simulation Platform," *IEEE Computer*, Feb. 2002.
- [15] M. R. Marty and M. D. Hill, "Virtual Hierarchies to Support Server Consolidation," *Proc. of the 34th Int'l Symp. on Computer Architecture*, June 2007.
- [16] M. Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing," *Ph.D. dissertation, University of California, Berkeley*, 1996.
- [17] B. O'Krafka and A. Newton. "An Empirical Evaluation of Two Memory-Efficient Directory Methods," *Proc. of the 17th Int'l Symp. Computer Architecture*, May 1990.
- [18] K. Olukotun et al. "The Case for a Single-Chip Multiprocessor," *Proc of the 7th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 1996.
- [19] M.K. Qureshi et al, "Adaptive Insertion Policies for High Performance Caching", *Proc. of the 34th Int'l Symp. on Computer Architecture*, June 2007.
- [20] M. K. Qureshi, D. Thompson, and Y. N. Patt, "The V-Way Cache: Demand-Based Associativity via Global Replacement," *32nd Int'l Symp. on Computer Architecture*, June 2005.
- [21] A. Sez nec. "A Case for Two-Way Skewed-Associative Cache," *Proc. of the 20th Int'l Symp. on Computer Architecture*, pp. 169-178, May 1993.
- [22] M. Spjuth, M. Karlsson and E. Hagersten, "Skewed caches from a low-power perspective", *Proc of the 2nd conf. on Computing frontier*, May 2005.
- [23] C. K. Tang. "Cache Design in the Tightly Coupled Multiprocessor System," *AFIPS Conf. Proceedings, National Computer Conference*, June 1976.
- [24] S. Vangal, et al., "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS," *IEEE Int'l Solid-State Circuits Conf.*, 2007, Feb. 2007.