An Efficient Data Access Policy in shared Last Level Cache

NITIN CHATURVEDI
EEE, Department
Birla Institute of Technology & Science
Pilani, INDIA
nitin80@bits-pilani.ac.in

S GURUNARAYANAN
EEE, Department
Birla Institute of Technology & Science
Pilani, INDIA
sguru@bits-pilani.ac.in

Abstract: - Future multi-core systems will execute massive memory intensive applications with significant data sharing. On chip memory latency further increases as more cores are added since diameter of most on chip networks increases with increase in number of cores, which makes it difficult to implement caches with single uniform access latency, leading to non-uniform cache architectures (NUCA). Data movement and their management further impacts memory access latency and consume power. We observed that previous D-NUCA design have used a costly data access scheme to search data in the NUCA cache in order to obtain significant performance benefits. In this paper, we propose an efficient and implementable data access algorithm for D-NUCA design using a set of pointers with each bank. Our scheme relies on low-overhead and highly accurate in-hardware pointers to reduce miss latency and on-chip network contention. Using simulations of 8-core multi-core, we show that our proposed data search mechanism in D-NUCA design reduces 40% dynamic energy consumed per memory request and outperforms multicast access policy by an average performance speedup of 6%.

Key-Words: - Non-Uniform Cache Architecture (NUCA), Last Level Cache (LLC), Multi-core Processors (CMP)

1 Introduction

As Multi-core Processors have become the predominant topology for the leading processors, the critical components including cache of the system are also integrated along with processing cores on a single chip. Cache hierarchy is of primary concern as it can be dominant in controlling overall throughput. With advent of every new technology there is an exponential increase in multi-core processor (CMP) cache sizes accompanied by growing on-chip wire delays which makes it difficult to implement traditional caches with single, uniform access latency. Non-Uniform Cache Architecture (NUCA) designs have been proposed to address this issue. In order to reduce the dominance of wire delays in upcoming new fabrication technologies, a NUCA logically divides the entire shared cache memory into smaller multiple banks where a cache line placed in the closer banks are accessed by the processor cores with reduced access latencies as compared to cache lines in banks that are placed further away from the requesting core. Conventionally, non-uniform organizations have been classified as static non-uniform cache architecture (S-NUCA) and dynamic non uniform cache architecture (D-NUCA). A cache line is always mapped to a fixed unique bank in the static organization as compared to dynamic organization where a cache line can be mapped in different banks of the shared cache. D-NUCA provides dynamic features like migration of cache line between multiple banks and moves frequently accessed data close to the requesting core. Multiple possible locations of data and its migration between multiple banks, complicates data access policy in the D-NUCA organizations. Previous research from both academia and industry proposed different schemes to access data in the heavily banked caches, which includes prioritizing banks which are frequently used, serial access scheme, in this scheme all the banks in the bank set are accessed serially by starting from the bank that is close to the requester core and continues to search the requested data block in the other banks, till the data block is found, or it results in a miss in the entire NUCA cache. This scheme minimizes energy consumption at the cost of reduced performance. Finally, parallel access scheme, in this scheme the entire bank set is accessed in parallel thereby providing much better

performance as compared to serial access with penalty in-terms of the increased energy consumption and on-chip network traffic. However, these techniques used costly access scheme [2], [3], [4] to search data in the NUCA cache before sending request to the upper-level memory. To address this problem, in this paper we propose an efficient and implementable data access algorithm for D-NUCA designs in CMP architectures. It utilizes the migration feature and provides fast and power efficient access to data which is located in any one of the banks of the bank set. Moreover, this scheme implements an efficient and low-cost search mechanism to reduce miss latency and on-chip network traffic. The rest of the paper is organized as follows: The next section describes the related work. Section III provides detailed explanation of proposed architecture and simulation environment. Section IV provides proposed implementation details followed by results that are presented in section V and finally conclusions are given in section VI.

2 Related Work

Research on cache memory organization in chip multiprocessors mostly concentrated on the last level cache designs. Different proposals have been made to manage last level cache as: either private to each core, or shared among all the cores or it can be a hybrid combination of both shared and private organizations [1, 2, 3, 4 and 6]. Private LLC organization provides limited cache capacity to a thread with large working set size, where as it can lead to inefficient cache utilization if some threads have small working set sizes. On the other hand, shared cache organizations provides flexibility for threads to share and store data at different locations in the cache and hence provides larger storage for applications with large working set size. Shared organization, have longer average access latency and higher on chip network traffic as compared to private organizations, however their off-chip miss rate are low as compared to private organization because data is not replicated in LLC. The influence of wire delays in shared LLC design leads to non-uniform access latencies. To address this problem of non-uniform access latencies, Kim et al. [1] introduced the original non uniform cache architecture (NUCA) as shown in Fig. 1. In shared NUCA, the whole LLC is partitioned into smaller banks and it provides nearer cache banks to have lower access latencies as compared to farther banks, thus mitigating the effects of on chip wire-delays. To improve CMP performance with shared D-NUCA, the capability of migration scheme depends on an accurate data access scheme which was complex and difficult to implement. The importance of data access scheme in dynamic NUCA organizations was first emphasized by Kim [1].

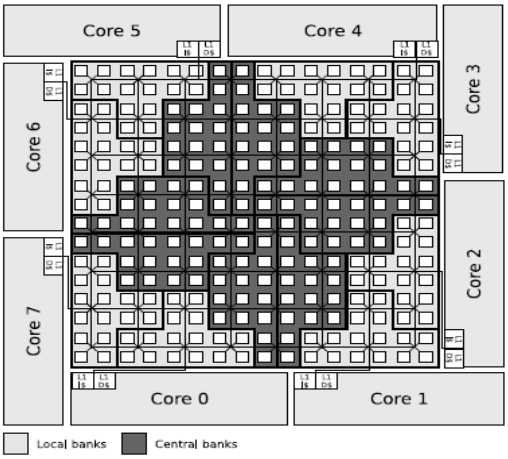

Fig. 1: Non-uniform cache architecture (NUCA)

Although block migration schemes were proposed to improve D-NUCA benefits, but it is limited by the quality of the bank access scheme with in NUCA. This work was further extended by Huh et al. [3] on D-NUCA and analyzed different NUCA organizations, he also came to the same conclusion that, dynamic cache organization performs better that the other organizations but data access policy becomes a critical issue in the heavily banked shared cache designs, since then researchers from industry and academia proposed different studies using NUCA organization in the literature that manage: block placement [3], [10], [15], [17], block migration [13], [15], [26] replacement [16], [31] and access scheme [18], [29]. Multi-core architectures brought additional challenges to the multi-banked NUCA organization. Chisthi et al. [21] also proposed an alternative NUCA design called NuRAPID, in which the Last level cache is divided into a few large banks instead of many smaller banks for higher reliability, efficiency and lower data migration rates with further extension to accommodate a limited number of cores. The concept of cooperative caching in multi-core processor systems was introduced by Chang et al. [25], where each processor core has a local L2 cache and cache consistency, sharing are achieved by listening in on all the L2 cache traffic and cooperating in decreasing the conflicts and increase

the overall capacity. Another variant of NUCA is proposed by Liu et al. Beckmann and Wood [3] analysis shows that block migration policy is less effective for CMP because 40-60% of total hits in commercial workloads were satisfied in the central banks. A novel Nahalal cache architecture was proposed by Guz et al. [24] in which the L2 cache is divided into two different partitions: (i) the private L2 partition for each core and (ii) and a separate L2 partition that is shared among all cores [27] by introduced a fully shared multi-banked L2 cache in which the most frequently shared data blocks are placed in a central bank located centrally to all cores. In this design, the shared banks are at the centre of the processor cores enclose the shared elements, and private L2 caches are located close to the cores. Kim et al. [1] analyzed two distinct access schemes: incremental search and multi-cast access. Incremental access scheme, accesses all the banks sequentially from the closest to the farthest bank, whereas multi-cast search accesses all the NUCA banks in parallel. Another way to improve data access algorithm is to introduce tag replication information along the NUCA cache [3], [5], [15], but this approach leads to increased access time, energy consumption and die area. Muralimanohar et al. [17] alleviates the interconnection delay bottleneck. They proposed the use of two different physical wires to connect NUCA banks, one of these wires provided lower latency and the other provides wider bandwidth. Abhishek et al. [30] proposed dynamic directories to eliminate large fraction of on-chip interconnect traversals and reduce power consumption. Akioka et al. [29] divided the entire NUCA into rings to denote a set of banks that exhibit the same access latency and introduced a Last Access based policy. This policy first accesses the ring that satisfied the previous request for the same data. They showed that, compared with parallel and serial access, LAB reduces energy consumption while maintaining similar performance. However, an accurate implementation is not addressed in terms of die area and access time. Keun et al. [32] argues that thread migration can better exploit shared data locality for NUCA design but it requires complex online predictors.

3 Search Policy: Owner bank tracks data movement

3.1 Baseline Architecture

We base our design on a distributed shared LLC design derived from Kim et al.'s Dynamic non-uniform cache architecture [1]. The shared L2 cache is distributed on the chip in multiple banks and these multiple banks are inter-connected via on-chip 2D mesh interconnection network. This cache organization is not a limitation, as the policy we explain can easily apply to different cache organizations. We first define few terms to ease describing our baseline architecture.

Owner Bank: The bank in which data is mapped first time during off-chip access using address mapping scheme.

Bank clusters: A group of eight banks compose a bank cluster and the entire NUCA banks (128 banks) are divided into 16 bank cluster shown by red dotted box in Fig. 2. Each bank cluster consists of a single bank of each bank set.

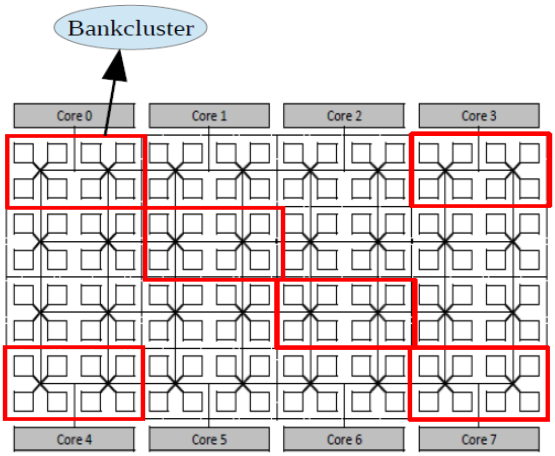

Fig. 2: Multibank NUCA with bank clusters

Bank set: All the banks that compose NUCA cache are treated as a set-associative structure as shown in Fig. 3, where in each bank holds one way of the set, which is called as bank sets.

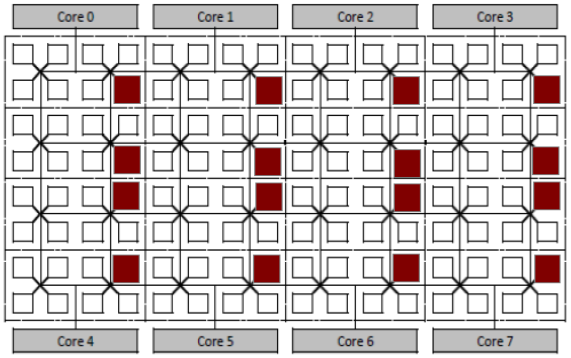

Fig. 3: Each bank holds one way of the set (16-way bank set associative)

As shown in Fig. 3, the entire NUCA cache is partitioned into 128 banks, which is logically organized into 16-way bank-set associative structure (Grey color banks constitute a bank set). Now, the groups of eight banks (bank cluster) that are located close to the cores are called local banks whereas the other eight banks that are located at the center of the NUCA cache are called central banks. Therefore, in bank-set associative NUCA cache a data block can have 16 possible placements (eight local banks and eight central banks). The address mapping of the incoming data block during first reference when it comes from off chip memory is statically predetermined by selecting lower bits of the data block address as shown in Fig. 4.

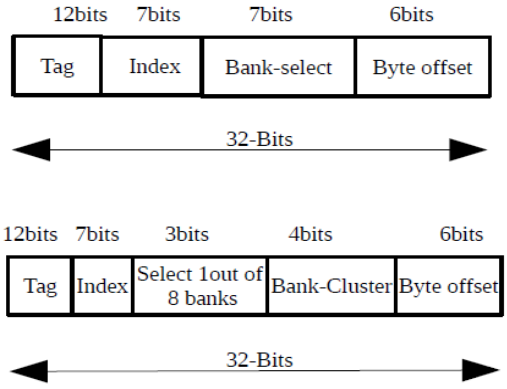

Fig. 4: Address Interpretation

The LRU data block in the referenced set of this bank would be evicted if the set is completely occupied by data blocks. Once the data block is in the bank of NUCA cache, the migration policy is used to determine its optimal position. We assume gradual promotion as migration policy for data blocks that has been widely used in the literature [2], [3]. Gradual migration moves data block one step close to the core that has initiated the memory request. In Ideal D-NUCA, a data block can be mapped into any cache bank to maximize placement flexibility of the block. However, the overhead of searching a data block in that scenario may be too large as each bank in entire NUCA must be searched. This can be done either through a centralized tag store or by broadcasting the tags to all the banks. To address this issue, our baseline architecture allows data blocks to be mapped (migrated) only to one bank-set. Our baseline D-NUCA design uses a two-step multicast data access algorithm. In the First step, it broadcasts a block request to the local banks that are close to the core that has launched the memory request, and to the eight other banks in the bank set called central banks. If all nine requests results in a miss, then in second step, the request is sent in parallel to the remaining seven banks from the requested data's bank-set. Finally, if the request misses in the remaining 7 banks, then the request would be forwarded to the off-chip memory. Therefore, when we evaluate NUCA further, we will assume the same NUCA architecture described above in this section, but we will use our proposed data access algorithm to find the exact location of data instead of the two step multicast data access algorithm.

3.1.2 Owner Bank

In the proposed scheme, the owner banks keep track of a set of data blocks within the multi-banked NUCA cache. Before explaining the access policy, we introduced the term owner which should first be explained to better understand the rest of the scheme. Multiple banks from different bank clusters compose a bank-set and a cache line can be placed in any of these banks in the bank-set as shown in Fig. 3, but only one of the banks in the bank-set is called its owner bank in NUCA. The ownership of the data block is statically determined using the lower bits of the data block's address described in Fig. 4. Therefore, each bank in heavily banked cache acts as owner bank, and every individual bank manages equal number of data blocks. The owner bank keeps track of which other NUCA banks have the data blocks that it manages. For dynamic tracking of data movement within bank set, a set of bits are used to point to the other banks called owner pointers, it assigns a single bit to each bank from the bank-set so that there are 16 extra bits within a set. If the data block is migrated from owner bank to some other bank, then the corresponding bit is set to 1. If the bit is zero, it means none of the data block from this owner bank is moved to some other bank. So, there are 16 bits that acts as a pointer for each cache-set in the NUCA bank. We have used pointer at the cache-set level to increase the precision of our search policy to reduce on-chip network traffic which in turn improves access latency.

3.1.3 Working of the Proposed Scheme

The working of the proposed access policy along with the details to owner pointer management tasks is presented in this section. As a result of compulsory miss, a data block is fetched in the cache hierarchy. After its initial placement in owner bank the migration policy moves data blocks in the multiple banks in bank-sets depending on the

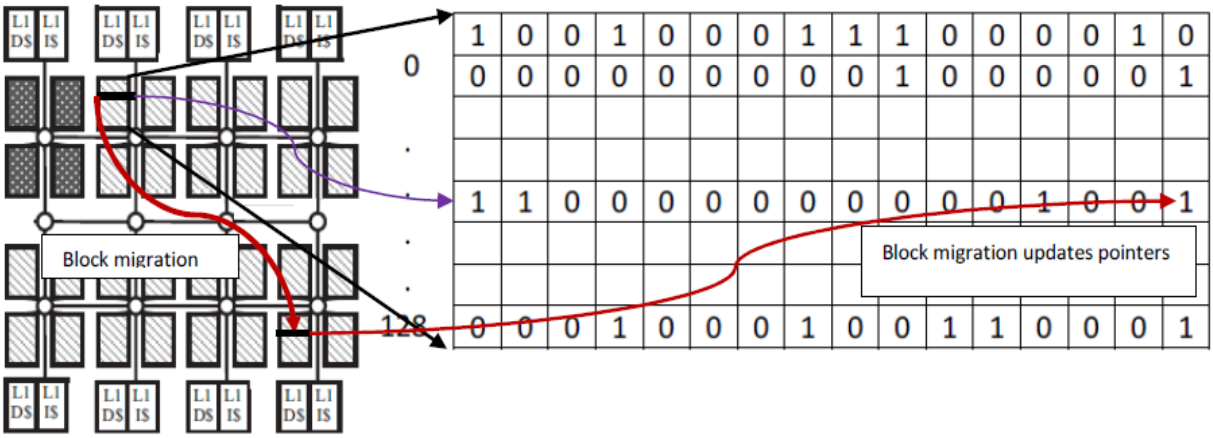

Fig. 5: Multibank NUCA with set pointers

workload. This makes data search a key challenge as a data block can be in local banks or in central banks. In order to utilize benefits offered by migration feature, the L2 cache controller are designed to start search mechanism into the closest NUCA bank of the requesting core, where the data block can be placed within the cache with reduced access latency. Now, if the cache line is found then this closest bank will send the requested cache line to the requesting core with minimum access latency and on-chip network usage. In case of a miss, the request is passed to the owner bank of the cache line. Now again a fresh search starts by accessing the requested data's owner bank in NUCA by using the lower address bits of the block address. If the requested data block is found, the request results in a hit and the block is forwarded to the requesting core and the search scheme is completed. In case of a miss in owner bank, the owner bank checks the corresponding set pointer to identify which other banks in the bank set may store requested data block. Then in the next step the request is forwarded, in parallel to all these banks in bank-set which have their bit sets to 1 in the corresponding set pointer as shown in Fig. 5. In case of hit in any of the bank in the corresponding bank set, the block is forwarded to the requesting core and the search is complete. This reduces the on-chip traffic as the memory request is forwarded to only few banks in parallel instead of sending it to all the remaining banks. In the worst case, if the cache line is not found, the request is finally forwarded to the main memory. In our analysis we have considered few special cases, that further accelerate the proposed data access scheme. For example, during the initial search into the bank closer to the core, the same bank is actually the requested data block's owner bank, which results in reduced steps in the search policy. In another special case, if none of the bits from set pointer in the owner bank is set (0), then the request would be forwarded to the main memory. Algorithm-1 demonstrates how lookup carried out on every reference from the cores. Let the set C = {C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7} represent the cores as described in the baseline NUCA architecture. Let L1 = { $L1_0$, $L1_1$, $L1_2$, $L1_4$, $L1_5$, $L1_6$, $L1_7$} be the respective private L1 caches. We use BC_{local} and BC_{owner} to refer to the local and owner bank-clusters respectively. Also assumed is a LRU based replacement policy, implemented using a queue.

Algorithm 1## To be carried out on every request
1: function handle Read/ write Request
2: INPUT: $ReadReq_j$ from C_i € C
3:Begin:
4:Lookup $L1_i$
5:if (hit)
6:Load $cacheLine_j$
7:$LRUQueue_{set}$.movetoEnd ($cacheLine_j$)
8:else
9:Fwd $ReadReq_j$ → BC_{local}
10:if (hit)
11:Load $CacheLine_j$
12:$LRUQueue_{set}$.movetoEnd ($cacheLine_j$)
13:else // local bank-cluster miss
14:send request to the owner bank
15:if (hit)
16: send $CacheLine_j$ to the requesting core
17: $LRUQueue_{set}$.movetoEnd ($cacheLine_j$)
18: else (miss in owner banks)
19: lookup set-pointers in the corresponding set
20: if status (set pointer bits) =1
21:send request in parallel to all the banks (step 20)
22: if (hit)
23: send $CacheLine_j$ to the requesting core
24: $LRUQueue_{set}$.movetoEnd ($cacheLine_j$)
25 : else (miss)
26:Fwd $ReadReq_j$ → off-chip memory
27:endif
27:endif
28:endif
29:endif

3.1.4 Managing Set Pointers

To ensure the correct operation and accuracy of this data search policy, the set pointers keeps the updated status. For example, during migration, a block is moved from one bank to the other bank, therefore to dynamically update pointer bits in the corresponding set of the owner bank, an update message is sent to the owner bank. In other case, if this cache line is evicted then the owner bank must be notified, to reset the corresponding bits in the set pointer. There are only few cases that require set pointer update: 1) a new data entry to the bank during first reference, 2) a data block eviction from the cache bank, and 3) a data block migration to the other banks. During initial data placement in the owner bank from main memory, the set pointer update is not necessary. However, in case of the cache line eviction from a bank, the corresponding owner bank must be notified dynamically to update set pointer. When an incoming data block enters to the NUCA bank from the upper-level memory, it is mapped to its owner bank, thus updating the set pointer is not necessary.

Algorithm 2 ## Block ownership (owner bank)
1: function handle set pointers at owner bank
2: INPUT: ReadReq $_j$ from C_i € C
3:Begin:
4:Lookup $L1_i$
5:if (compulsory Miss)
6: Fwd ReadReq $_j$ → off-chip memory
7: Block allocation (depending on address)
8: set bank as owner bank of the block
8:s send $CacheLine_j$ to the requesting core C
9:else (non compulsory miss)
10: Fwd ReadReq $_j$ → BC_{local}
11: execute (algorithm-1)
12: endif
13: if (frequent request from Cj,k,l belongs to C)
14: migration algorithm
15: notification send to owner bank
16: update corresponding set-pointer
17:elsif (data block eviction)
18: lookup entire set
19: if (another block from same owner bank)
20: silent eviction
21: else
22: notification send to owner bank
23: update corresponding set-pointer
24: insert new cacheline
25: $LRUQueue_{set}$ .movetoEnd (cacheLine $_j$)
26:endif
27:endif
28:endif

However, if there is an eviction in a bank then set pointer must be updated by sending a notification message to the corresponding owner bank. Finally in order to ensure correctness and efficiency the cache line movements must be synchronized with the modifications made to the set pointers, to prevent misalignments that provoke extra network messages, but it does maintains the correctness of the proposed scheme. Algorithm-2 demonstrates how to keep track of the migrated blocks.

3.1.5 Cache Coherence Protocol

Our work is based on a directory protocol and this protocol does not need ordered interconnect to satisfy coherency. We also believe that future CMP will be based on directory like structure to provide coherence and it can scale to large on chip cores. To sustain correctness and to implement different read and write scenarios, cache coherence protocols utilize transition states. Our protocol implementation inherits such transition states from the baseline cache coherence protocols and used these transient states to maintain coherent view and correctness. In the proposed cache access scheme, for any cache line that does not exhibit a complex sharing and therefore search mechanism, the implemented protocol works similar to the baseline cache coherence protocol which is basically enforcing a write-invalidate policy for all cache lines in the shared NUCA. Our proposed scheme along with protocols is implemented on top of the write-invalidate directory protocol, which is modified baseline MOESI protocol. The exclusive state (clean) obviates upgrade misses to non-shared data. Race conditions are handled using busy or active states for each request. Fig. 6, briefly describes how a write-invalidate based protocol works for a simple data sharing example. The arrows present a specific location in the system with hypothetical time line. From left to right, these locations are the requester core, the L2 shared cache which also includes the directory that is co-located, the consumer cores, and the main memory. For clarity, the example assumes a single requesting core a single consumer core of the cache line. Also, the cache line is assumed to be loaded and modified earlier so that the cache line actually exists in the requester's cache. Similarly, the cache line also exists in the shared L2 cache. We assume the initial state of the cache line is OWNED in the requester's cache and SHARED in the consumer's core cache.

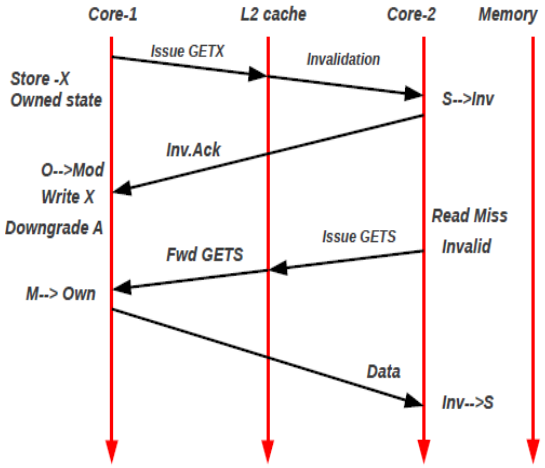

Fig. 6: Directory based write invalidation Protocol

The directory is co-located with each cache line and it tracks different cores. Fig. 7, shows how a request is forwarded to the L2 bank containing data block with a sequence diagram.

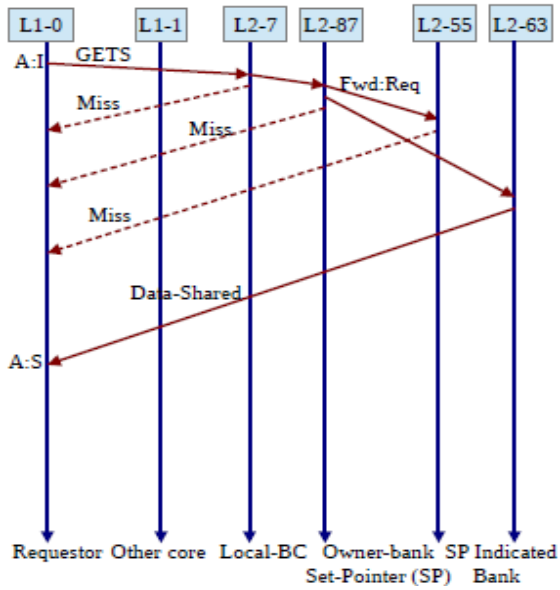

Fig. 7: Directory based write invalidation Protocol

3.1.6 Verification of Coherence Protocol

Modified MESI based directory protocol relies on the baseline coherence protocol for correctness. However, the modifications made to the base MESI protocol can easily create race conditions which need to be verified and tested. For the verification, we have utilized the stress tests provided by GEMS tools set. This synthetic testing mechanism generates excessive race conditions on the cache coherence protocol to identify potential coherence bugs.

4 Experimental setup

In this section, we describe our evaluation methodology and all the results are obtained with the system configuration described in Table I.

4.1.1 Simulation Environment

We simulate the entire system using virtutech simics full-system simulator [6] extended with the GEMS toolset [4]. GEMS is an event driven simulator that provides a complete memory-system timing model that enabled us to model the multi-banked NUCA cache architecture. Furthermore, the RUBY memory system simulator provides support to implement baseline system memory hierarchy.

Table I. System Configuration

Configuration Parameters	
No of Cores	8
Core Mode	Single Thread
Frequency	1Ghz
L1-Data Cache	32kb, 64 bytes
L1-Instruction Cache	32kb, 64 bytes
Shared L2 Cache	8 Mb, 128 banks
Bank Size	64 Kb, 8-Way, 64bytes

This includes on chip interconnection network parameters, bank access time, mapping, replacement policies etc. In ruby each cache bank has its own controller and using domain specific language called SLICC we can specify with precision the coherence protocol. This environment allows us to simulate a complete multiprocessor system that is running a commercial operating system without any modification and it accurately models the network contention introduced during the simulation. The simulated system is organized as a single CMP that consists of eight UltraSPARC IIIi homogeneous cores with layout depicted in Fig. 2. Each processor core has its own first-level cache (data and instructions) and is connected to a node of the network. The Last level of the memory hierarchy is the D-NUCA distributed in 128 banks connected to the cores via switches. We used MOESI based directory protocol to maintain correctness and

robustness in the memory subsystem. The main system configuration parameters used in our simulations are shown in Table I. To quantitatively analyze the proposed scheme, we used two different scenarios: 1) Multi-programmed and 2) Parallel applications. The first one executes in parallel a set of eight different SPEC CPU2006 workloads with the reference input and fast forwarded to the beginning of the main loops. Table II outlines the workloads that make up this scenario. The Parallel workload simulates the complete set of applications from the PARSEC v2.0 benchmark suite [8] with the simlarge input data sets. This benchmark suite contains 13 programs from different areas such as, computer vision, image processing, financial analytics, video encoding and animation physics.

Table II. Benchmarks

Benchmarks	Applications	Input
PARSEC	Blackscholes, bodytrack, canneal, racesim, fluidanimate, x264, raytrace,swaptions, streamcluster	Sim-large Input
SPEC2006	Mix or Different applications, gcc, ibm, astar, mcf, soplex, perlbench	Reference Input

The method for the simulations involves first skipping both the initialization and thread creation phases, and then fast-forwarding while warming up the cache for 500 million cycles. Then finally, we performed a detailed simulation for 500 million cycles. We use the aggregate number of user instructions committed per cycle as performance metric, which is proportional to the overall system throughput [28].

4.1.2 Energy Calculations

To evaluate the energy consumed by the multi-banked NUCA cache and the off-chip memory. We used a similar energy model to that adopted by Bardine et al. [10] which allowed us to calculate the dynamic energy dissipated by the banks in LLC cache in addition to energy required to access the off-chip memory. As shown below, the total energy consumed by the NUCA memory system is the sum of all three components:

$$E_{Dynamic} = E_{network} + E_{banks} + E_{off-chip}$$

Therefore, total dynamic energy consumed is the sum of the dynamic energy consumed by the NUCA cache, which is energy consumed in network plus the static energy consumed in banks and also the energy dissipated during the off-chip memory ($O_{ff-chip}$). CACTI 6.0 has been used to compute energy dissipated in banks, while GEMS toolset also integrates a power model based on Orion [10] that is used to evaluate the dynamic power consumed by the on-chip network (Energy in network). Energy consumed per memory access is the metrics and it is based on the energy per instruction (EPI) metric [23] which is commonly used for the analysis of the energy consumed by the whole processor. This metric works independently of the amount of time required to process an instruction and is ideal for throughput performance.

5 Results

This section analyses the impact on performance and hardware overhead using the proposed bank access policy with set ownership pointer in the baseline architecture.

5.1.1 Performance Improvement

We used two separate scenarios to analyze the performance results obtained with our proposed scheme. In the first case we have selected applications from the parsec benchmark that shows high cache miss rate like streamcluster, canneal, vips and multiple applications from SPEC2006 to make workload. We have observed that our scheme outperforms the baseline architecture scheme by 8% as shown in Fig. 8. By taking advantage of set pointers with owner bank, memory requests are directly forwarded and satisfied by only accessing the cache banks that can have the requested data which significantly lowers network traffic,

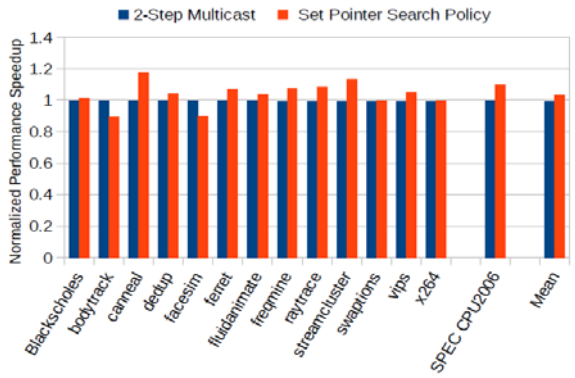

Fig. 8: Normalized Performance

thus it reduces time to resolve cache miss before the request is forwarded to the main memory. Therefore, for the applications with higher miss rates, the impact on the performance is even better. In the second scenario, we have observed applications with low miss rate, like raytrace, dedup, swaptions and x264. In this scenario both the scheme take equal access latencies when request hit in the closest banks. While the two-step multicast scheme introduces 8 extra messages to the on chip network whereas set pointer scheme sends only one message to the network. Therefore, this scheme reduces congestion as shown in Fig. 9, due to reduce on chip network traffic (on average 45%) and results in performance improvement of nearly 4% as compared to baseline architecture.

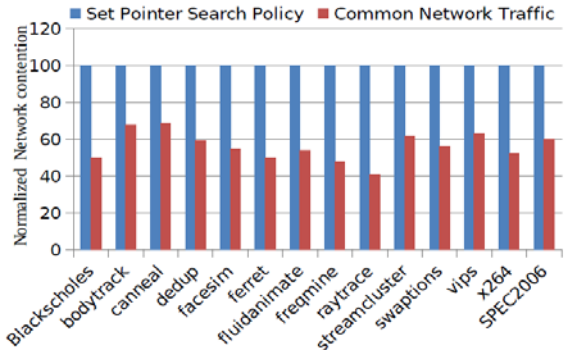

Fig. 9: Network Contention

We have observed negligible performance improvement for the applications with very high hit rate like bodytrack. We assume that the applications running on future processors will follow the first case: where workloads with large working sets and multiple applications will run simultaneously. Fig. 8, shows the performance improvement obtained with the set pointers that we evaluated, as compared to baseline architecture. We found that our proposed scheme outperforms two-step multicast scheme by an average of 6%. In general our scheme improves performance for almost all the PARSEC applications, by achieving more than 8-10% improvement in canneal, ferret, streamcluster applications. In case of SPEC2006 multi programmed workloads an average improvement of 10% is observed.

5.1.2 Energy Results

Fig. 10, shows the dynamic energy consumption of each benchmark using the proposed data access scheme relative to the baseline two-step multicast data access scheme. The energy reduction can be primarily attributed to the reduction in network traffic. It is important to note that our energy model does not account for the off-chip memory. Therefore, for benchmarks where our proposal improves the L2 performance, the energy benefits will in fact be higher. We observed that the proposed scheme improves energy consumption of the NUCA cache by more than 40% as compared to the two-step data access in baseline architecture.

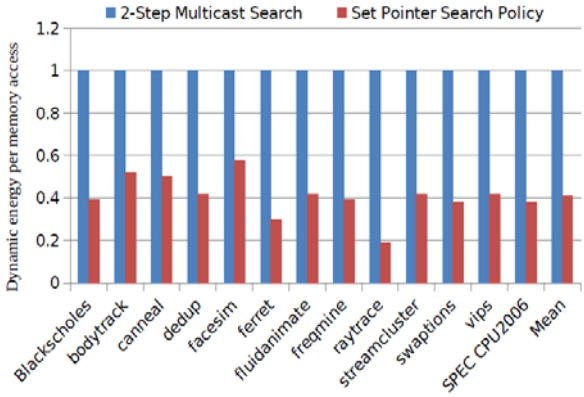

Fig. 10: Dynamic Energy

To summarize, the proposed data access scheme provides reduced energy consumption and increased performance as compared to the two-step multicast scheme. It is important to balance the on chip data locality and off-chip miss rate and overall our scheme achieves the best trade-off.

5.1.3 Set Pointer Implementation overhead

This search policy requires additional hardware to implement set pointers. As shown in Fig. 5, the set pointers requires 256 bytes for each bank, so as per our baseline configuration with 128 banks and total shared cache size of 8 Mb, the extra bytes required by set pointer is nearly 32 Kb, which adds extra 1.2% of the total cache size. In addition to extra storage, the proposed scheme requires extra

comparators which slightly complicate cache design. This data search scheme is proposed for heavily-banked cache architecture, but it can be easily adapted to tiled multi-core architectures.

6 Conclusions

Future chip multiprocessors will be based on tiled architectures with large shared L2 cache. The increasing wire delay makes data locality a key performance bottleneck. Many existing data migration schemes for NUCA caches succeed in concentrating the most frequently accessed data in the banks with the smallest access latency. However, this migration of data blocks increases complexity of data access policy. In order to address this situation, we have proposed a data access policy. This hybrid policy uses both serial and parallel search to probe NUCA banks with optimal access latency and make D-NUCA promising non uniform cache architecture.

References:

[1] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *Procs. Of the 10th Intl. Conf. on Architectural Support for Programming Languages and operating Systems*, 2002.

[2] B. M. Beckmann and D. A. Wood, "Managing wire delay in large chip-multiprocessor caches," in *Procs. Of the 37thInternational Symposium on Micro architecture*, 2004.

[3] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A nuca substrate for flexible cmp cache sharing, "in *Procs. Of the 19th ACM International Conference on Supercomputing*, 2005.

[4] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," in *Computer Architecture News*, 2005

[5] R. Ricci, S. Barrus, and R. Balasubramonian, "Leveraging bloom filters for smart search within nuca caches," in *Procs.of the 7th Workshop on Complexity-Effective Design*, 2006.

[6] Alessandro Bardine, Pierfrancesco Foglia, "Way adaptable D-NUCA caches" in International Journal of High Performance Systems Architecture (IJHPSA), Volume 2 Issue 3/4, August 2010, Pages 215-228.

[7] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren,G. Hallberg, J. H¨ogberg, F. Larsson, A. Moestedt, and B. Werner, *Simics: A Full system Simulator Platform*. Computer, 2002, vol. 35-2, pp. 50–58.

[8] Starvrou, Costas, Paraskevas "Chip multiprocessor based on data-driven multithreading model" in International Journal of High Performance Systems Architecture (IJHPSA), Volume 1 Issue 1, April 2007, Pages 34-43.

[9] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Procs. Of the International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[10] A. Bardine, P. Foglia, G. Gabrielli, and C. A. Prete, "Analysis of static and dynamic energy consumption in nuca caches: Initial results," in *Procs. Of the Workshop on Memory Performance: Dealing with Applications, Systems and Architecture*, 2007.

[11] H. S. Wang, X. Zhu, L. S. Peh, and S. Malik, "Orion: A power- performance simulator for interconnection networks," in *Procs. of the 35th International Symposium on Microarchitecture*, 2002.

[12] N. Nedja, A. S Nery "A massively parallel hardware architecture for ray-tracing" in International Journal of High Performance Systems Architecture (IJHPSA), Volume 2 Issue 1,December2009,Pages 26-34.

[13] Micron, "System power calculator," in *http: //www.micron.com/*, 2009.

[14] M. Hammoud, S. Cho, and R. Melhem, "Dynamic cache clustering for chip multiprocessors," in *Procs. Of the Intl. Conference on supercomputing*, 2009.

[15] M. Kandemir, F. Li, M. J. Irwin, and S. W. Son, "A novel migration-based nuca design for chip multiprocessors," in *Procs. Of the international Conference on Supercomputing*, 2008.

[16] J. Lira, C. Molina, and A. Gonz´alez, "Last bank: dealing with address reuse in non-uniform cache architecture for cmps," in *Procs. of the International Conference on Parallel and Distributed Computing*, 2009.

[17] N. Muralimanohar and R. Balasubramonian, "Interconnect design considerations for large

nuca caches," in *Procs. Of the 34th international Symposium on Computer Architecture*, 2007.

[18] M. Chaudhuri, "Pagenuca: Selected policies for page-grain locality management in large shared chip-multiprocessor caches," in *Procs. of the 15th IEEE Symposium on High-Performance Computer Architecture*, 2009

[19] J. Merino, V. Puente, and J. A. Gregorio, "Sp-nuca: A cost effective dynamic non-uniform cache architecture," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 2, pp.64–71, May 2008.

[20] M. Hammoud, S. Cho, and R. Melhem, "Acm: An efficient approach for managing shared caches in chip multiprocessors," in *Procs. Of the 4th Intl. Conference on High-performance and Embedded Architectures*, 2009.

[21] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Distance associativity for high-performance energy-efficient non-uniform cache architectures," in *Procs. Of the 36thInternational Symposium on Micro architecture*, 2003.

[22] L. Hsu, R. Iyer, S. Makineni, S. Reinhardt, and D. Newell. "Exploring the cache design space for large scale cmps." *SIGARCH Computer Architecture News*, 33(4): pages 24–33, 2005.

[23] Shekhar Srikantaiah, Mahmut Kandemir, Mary Jane Irwin. "Adaptive Set Pinning: Managing Shared Caches in Chip Multiprocessors." Proceedings of ASPLOS'08, ACM/IEEE, pages 135-144 March, 2008.

[24] Z. Guz, I. Keidar, A. Kolodny, and U. Weiser, "Nahalal: Cache organization for chip multiprocessors," *IEEE Comput.Archit. Lett.*, vol. 6, no. 1, 2007.

[25] J. Chang and G. S. Sohi, "Cooperative caching for chip multiprocessors," in *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*, 2006, pp. 264–276

[26] C. Liu, A. Sivasubramaniam, M. Kandemir, and M. Irwin, "Enhancing L2 organization for CMPs with a center cell," in *IPDPS 2006, 20th international Parallel and Distributed Processing Symposium*, April 2006, p. 10.

[27] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi,"Optimizing nuca organizations and wiring delays alternatives for large caches with cacti 6.0," in *Procs. Of the 40th International symposium on microarchitecture*, 2007.

[28] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "Simflex: Statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, no. 4, pp.18–31, 2006.

[29] S. Akioka F. Li, K Malkowski, P. Raghavan, M. Kandemir, and M. J. Irwin, "Ring data location prediction scheme for non-uniform cache architectures," in Procs. Of the International Conference on Computer Design, 2008.

[30] A. Das et al., "Dynamic Directories: A Mechanism for Reducing On-Chip Interconnect Power in Multicores," Proc. 2012 Conf. Design, Automation, and Test in Europe (DATE 12), EDA Consortium, 2012, pp. 479-484.

[31] Aamer Jaleel, Hashem H. Najaf-abadi, Samantika Subramaniam, Simon C. Steely Jr., and Joel Emer CRUISE: Cache Replacement and Utility-aware Scheduling ASPLOS'12, March 3–7, 2012, London, England, UK.

[32] Keun Sup Shim, Mieszko Lis, Omer Khan and Srinivas Devadas "Judicious Thread Migration When Accessing Distributed Shared Caches" *Procs.* Of the Third Workshop on Computer Architecture and Operating System Co-design (CAOS), 2012.