I J C T A, 9(20), 2016, pp. 231-243
© International Science Press

A Survey on Last level Cache Partitioning Techniques in Chip Multi-Processors

Jobin Jose* and N. Ramasubramanian**

ABSTRACT

Chip Multi Processors (CMPs), a new generation of multicore architecture emerged as the base of System on Chip(SoC) paradigm. Multiple processing cores are packed into a single chip here. Each core is capable of executing simple and complex applications in parallel. Memory is being considered as a scarce resource for the application. The multilevel memory hierarchy that involves various levels of caches are being accommodated into the chip in shared/private mode for faster access of data. Of these, the Last Level Cache (LLC) is being shared amongst various processing cores. Multiple applications are accessing the same L2 cache, and thus increases the conflicts in cache entries and this acts as a source of contention. These applications may produce the destructive interference between cores and at the same time occurs cache misses. This will cause a serious effect on the performance and efficiency in terms of energy of the system. In order to mitigate these problems, the shared LLC is being partitioned amongst the applications. Depending on the cache organization, different schemes are used for partitioning that includes an initial placement and replacement policies that helps to develop a good system which will improve the utilization of cache space. This paper presents a study on various cache partitioning schemes used for the LLC in terms of replacement policy, simulating tools and cache design and analyze in term of performance and found that dynamic way based partitioning techniques performs better in terms of energy.

Keywords: Chip Multi Processors; Multi level memory; Last level cache;LLC; Cache partitioning; Replacement policy; cache misses.

1. INTRODUCTION

The revolution of multi-core processors has turned into a new direction after the introduction of Chip multiprocessors (CMPs)[1] that solve many disadvantages of uni-processor systems such as scalability, energy and area efficiency. The CMPs improves the throughput and the degree of parallelism at various levels. With the introduction of CMPs, the multi core architecture has moved from desktop computers to embedded systems. Multiple small cores are accommodated in the small die area that makes the CMP more area efficient. The power consumption and the resource sharing via wires are still appears as performance hindrance factors for it. The basic element in the CMP architecture consists of homogeneous or heterogeneous processing element(PE), that includes cores and the on- chip cache memory together. The cores use caches to reduce both the latency in memory accees and traffic. The cache in the system, are shared via properly designed interconnection networks that has organized as shared or private, of which the shared cache provides better performance, as this was accessed by multiple cores. As the number of cores in the multiprocessor system increased, the shared cache has become a source of contention which has increased the memory pressure and thus affects the utilization efficiency. The sharing also causes interference due to multiple accesses by different cores. This often produces a large amount of misses in the system. The private cache is often small in capacity, that can be accommodated as on chip caches which reduces the access latency and provides high utilization. Although the private caches provides good access efficiency, it cannot be used in the environments where high memory is demanded. Due to the wire delays in the inter

* Department of Computer Science and Engineering National Institute of Technology, Tiruchirappalli, *Email: jobin16981@gmail.com*

** Department of Computer Science and Engineering National Institute of Technology, Tiruchirappalli, *Email: nrs@nitt.edu*

connection, it is very difficult to provide a uniform access to the cache and sufficient bandwidth to the accessing applications.

One method to solve these problems is partitioning the cache, that logically partitions the large cache space into several parts, whose granularity depends on the application demand, that is executing in each core. The applications can be categorized as high demand applications, saturated applications and low demand applications, based on the cache demand. In order to have better utilization, the high demand applications are to be treated appropriately. The partitioning scheme has two parts: one is the allocation scheme and another is the replacement policy. The former deals with the placement of blocks in the cache and the latter deals with the replacement of blocks upon eviction. The placement of blocks deals with the initial as well as placement of blocks, that can be determined by the utility. The partitioning is available in two variations: static and dynamic, where the former predetermines the amount of cache blocks needed for the workload at the beginning[4]and the latter adjusts the cache sizes dynamically wherein the allocation policy determines the size and number of the partition[8,23] that improves the performance, throughput, fairness[10,27], QoS etc. and a partitioning scheme that enforces those sizes. This paper aims to provide a survey about various cache partitioning techniques that has been applied to CMP structure and analyze the performance.

The subsequent organization of the paper is as follows: Section II provides the design issues in partitioning, III describes the various categories of cache partitioning schemes. Section IV describes the comparative analysis of different cache partitioning schemes in CMPs. Concluding remarks are given in Section IV.

2. DESIGN ISSUES IN CACHE PARTITIONING

There are various issues in partitioning cache, which are discussed below:

1. Cache Indexing: Indexing techniques are at different levels of cache may incur conflict misses[2], which is costlier and produces memory overload.
2. Resizing penalty: The resizing of cache blocks during partition incurs certain impact on the applications that are accessing the blocks in the cache. The high performance applications may not get sufficient memory for its efficient execution and also certain data may be flushed out[24].
3. Associativity: The term implies the mapping of data into the cache. The level of associativity usually decreases while partition into ways. The unrestricted placement of cache lines sometimes increases the associativity [2,24]. High degrees of associativity risk increasing memory access latency.
4. Scalability: The partitions can get the size from other partitions [2,21], based on the demand of accessing applications.
5. Workload: The application workload may be selected in such a way that it should be memory intensive, having large working set size and good locality. Homogeneous mix or heterogeneous mix of applications can be considered depending on the characteristics of the method. This categorization is possible based on their cache space sensitivity[22] and MKPI and CPI.
6. Cache Sensitivity: The sensitivity of a cache is measured in terms of hit rates and miss rates in the cache. This is a very good measure of evaluating the execution time of each process and thus to minimize the worst case execution times[6].
7. Memory fragmentation: Equal-sized cache partitioning causes memory fragmentation [1,3].
8. Conflict misses: This may arise the eviction of data from the cache lines. These evictions can be managed applying suitably defined replacement policies [3].

9. Responsiveness: The wastage of memory by keeping them idle is a big source of performance degradation and consumes more energy .The partitioning scheme must take on full utilization[14] of all cache blocks. If portions of the cache is not being utilized fully, can be turned off[7,8].
10. Resource contention: The contention of resources often happens in shared memories as well as bandwidth. This may happen when higher priority tasks and low priority tasks are accessing the memory [18]. Once this is partitioned, contention in memories can be minimized as the bandwidth cannot be partitioned .But, adding more applications by oversubscribing will increase the contention and thus degrades the performance.
11. Access latency: The partitioning cause some latency in accessing the cache blocks. The latency of accessing the drowsy blocks is high in[16] as is the case of shared caches in[16].
12. Cache Conflict : Cache conflicts occur when some contents are assigned on to the same cache line[1] which is a very fundamental issue in caches, more specifically, in cache partitioning too. When multiple applications are competing for the resources to access, this issue will occur. This can be avoided by code placement and padding techniques in [6].

3. CACHE PARTITIONING TECHNIQUES

3.1. Way Based Partitioning

Generally, the partitioning schemes can be classified as static and dynamic partitioning methods which apply on real time applications whereas some may work also in mixed mode. The main aim of the partitioning methods discussed in this section is to improve the utilization[17].At the same time some schemes are used to provide isolation and interference reduction[8], energy minimization, in terms static and dynamic energy[5,12,16,18] and some schemes ensures fairness[10]. Majority of the partitioning schemes on caches works on way based partitioning that splits the cache space into two logical sections, that adopts various methodologies. Many of these schemes comes under dynamic schemes with one exception as in DCR+CP[5] and make-span[15], a static partitioning on shared caches can be employed to minimize the energy for the real time systems. A static profiling technique is employed to avail the benefit of partitioning on L2. The number of ways are statically allocated to each core and is determined at the earlier stage itself. This method also provides dynamic configuration of L1 caches along with the partitioning which minimizes the total energy, whereas in [15], provides a balance in partition size. The allocation does not change till the completion of execution of threads and ensures all threads finish at the same time provide more fairness.

The dynamic partitioning schemes, divides the cache space at run time that works on a variety of ways. The [6,7,8,9,10,11,12,14,16,17,18,21,24,25,26,27] comes under this category. Each employs partitioning based on the adaptation of various methodologies such as managed and unmanaged regions[8], data ownership based partitioning in to shared and private regions[11,12,16] with the difference is that, the [11] is based on real time task ownership. This ensures that private cache is accessible only by the owner task and the shared cache is accessible by all the tasks. During a stipulated time frame if the private space is not fully utilized by the owner, then the remaining space can be allocated to other non critical tasks. The CaPPS[17] uses slightly different terminology for partitioning called partial sharing on the basis of core ownership that divides cache into private and shared regions. The RWP [25] divides the cache into clean and dirty regions that minimizes the read misses by employing a read-write policy over the cache lines. The [4] uses a drowsy mode that divides the cache space into A/B partitions, in which the initial access is on partition A and the miss on item will lead to the access in partition B, which is in drowsy mode. The entire cache is partitioned on the granularity of threads described in [10] and a fairness value is assigned to each slice and is compared with the previous value and the greater value indicates the correct partition for fairness and that guarantees size on partitions. The partitioning scheme described in[27], meant for heterogeneous applications can be viewed as a unified shared cache that considers no associativity. This scheme splits the cache blocks equally among the applications initially and then reallocates the cache block to an application that has performance degradation due to the cache misses. Another scheme that employs into the LLC called NUCache[7], that separates the cache ways into Mainways and Deliways in a logical manner. This maintains set associativity while partitioning. All incoming blocks are initially placed in main ways and blocks from the delinquent PC are placed in the deliways on replacement, provided they should produce additional hits. Most of these schemes considered for review uses a unified cache structure with no guarantee on associativity initially, whereas vantage[8] highly associative cache and [6] uses both associative and direct mapped caches. The Liu et.al in[6] uses a shared cache as L2 that splits into different shares depending on the number of cores and assign tasks to cores with the aim of minimization of another aspect i.e Worst Case Execution Time(WCET) which we haven't examined earlier in this paper. Along with the task assignment, a locking mechanism is also being performed to restrict the size on each shares. With the motivation of improving the locality of reference, a bloom filter based scheme is proposed in[9]. The hash function based filter array is used to identify far misses that occur in each shared cache set.

The initial block placement of blocks into the cache does not incur any overhead, whereas PRETI requires future knowledge about all requests. The new cache line is inserted based on dynamic insertion policy that determines the partition in which the newline is inserted in place of an evicted line. The determination of the candidate for replacement and evictions of a cache line is done based on the frequency in which the line is being accessed. This can be accomplished by accommodating a counter and shadow tags in the structure. These counter determines the utility[8] of the cache line with minor improvements, such that vantage provides churn based management technique based on the insertion rate on each partition at any given time. In RWP[25], the utility monitor is used to determine the partition size which employs set sampling that maintains shadow directories for both clean and dirty lines. The technique proposed in[21]called cooperative partitioning that uses a utility based UCP lookahead algorithm to determine the optimal partition for allocation to the applications, whereas in[26] the victim block is determined by the utility value, checked by a utility predictor and the reuse distance. In contrary to utility, futility, a term which determines the uselessness of each cache line for ranking the cache line for replacement. This employs a replacement based algorithm called futility scaling[24] that guarantees partitions can grow/shrink their sizes.

The baseline replacement policy is LRU[8, 11, 12, 14, 16, 17, 24], which is a simple and common policy in almost all partitioning schemes with some minor modifications such as time stamp based LRU[24] clubbed with churn based management[8], replacement happens on a miss[16,17] and [25] uses an additional policy called MRU to determine the block placement. Whenever a miss happens in the main partition[14], the lookup is done on drowsy partition and if found, will be brought into partition by evicting certain lines. In[17] the oldest block which are either in the shared space or in the private space that produce a miss are replaced. The overhead occurs in the replacement policy, due to the addition of extra field in the tag. The unused ways in the centre region can be turned off which may incurs extra cycles in RECAP where as vantage performs evictions from the unmanaged region and demotions from the managed region. Before performing the replacement in CaPPS [17], the replacement candidates are to be occupied in the shared ways. RECAP does not provide isolation and requires extra circuitry for monitoring and partitioning.

A hardware implementation of dynamic partitioning scheme targeting MPSoCs using FPGA is described in [26]. Multiple applications from various processors can utilize different regions of the memory since it uses global address space which is partitioned. The cache management logic allocates cache to other cores while one core is searching for the resources. The availability of cache ways to the requesting cores is determined based on the state of the resources to be synchronized with the cores with the help of an API. To allocate cache ways, cache ways management unit(CMMU) along with block reference field logic is used. The released cache ways are flushed out in one tick of the clock cycle before reallocation. The reconfiguration of ways to cores are done by certain commands. The replacement policy used here is FIFO.

The other categories of partitioning methods will be described later in this section. Though they are, by nature, follows dynamic partitioning to a certain extend, their implementation is slightly different in some other context. That is the reason why we have mentioned it as a separate category. A main memory based partitioning scheme, meant for PCM main memory to minimize the write energy and limited endurance is proposed in[12].Though it is slightly out of our focus, the partitioning scheme alone has much significance here. This partitioning called WCP, is based on write back-aware policy that distributes the write bandwidth evenly across competing applications and minimizes the cache misses and writebacks.The partitioning decisions are taken based on the utility as described in[25] in terms of hit counts and shadow tags which is stored in hit counters and shadow tag arrays, that selects the valid or best partition by assigning weights, depending on the reduced miss rate and write backs. The write back minimization decisions on a way has been made depending on the write back avoidance distance corresponds to a way, when that way has the maximum stack distance. The replacement policy is LRU as described above with some additions such as implementation of a write queues balancing technique in order to distribute the write backs evenly across the queues to avoid the delay in removing a clean line or dirty line. WCP incurs some hardware overhead due to storage of utility and weights.

Considering the static and dynamic energy minimization as the prime motive, a cache partitioning scheme is proposed in[13] by karthik soundarrajan et al. that performs way alignment of data belonging to each core across all sets. The cooperation between cores is employed to migrate ways after partitioning . Whenever the way is unused by any core, it can be turned off to save energy similar to that in[16]. The scheme needs an extra circuitry in terms of registers for controlling the access to the ways by different cores that consumes more power than a regular cache and transferring ways across the cores creates performance overhead. A mix of static and dynamic Cache partitioning scheme, is proposed by Henry Cook et.al in[18] for improving the utilization of application responsiveness by co scheduling both foreground applications and background applications. The static partitioning is beneficial for foreground processes that achieve performance improvement and energy efficiency, while the dynamic cache capacity allocation framework based on utility is beneficial for background applications by detecting and responding the phase changes of the applications. More cache space is allocated when an application changes its phase until negative performance is seen. The data is not flushed when performing reallocation. If the cache size is close to the working set size excessive interference is taking place.

A tight bound on partitioning is provided in[19] called imbalanced partitioning that divides the cache set into preferred and non preferred segments whereas Flexi Way[20] divides into modules and the ways inside each modules, called subways are also considered for allocation and reconfiguration. Each core in [19] is prioritized to get the partitions for its application based on the minimum miss rate. This scheme lacks scalability and solution space. The unused ways in modules can be turned off, like in [13, 16].

Table 1
Static Partitioning Schemes

Technique	*Approach*	*Memory Type*	*Replacement Policy*	*Simulation Tool*	*Performance*	*Merits*	*Demerits*
DCR+CP[5]	Static profiling and partitioning ways equally amongst the cores	Uniform shared memory	NA	M5	Improvements in energy saving	Reduce inter-task inter-ferences, and improves energy	Only static energy can be determined
Make-span[15]	Partition a shared cache to Minimize the makespan of a set of threads.	Uniform shared memory	NA	NA	Better Shorter schedules	Ensures all the threads com-pletes at the same time	Applicable to one thread/ core with known miss rate

Dynamically repartitionable static non uniform cache access, DR-SNUCA based partitioning method is proposed in [21] in order to avoid excessive tag matching and interference in Dynamic NUCA. This employs a tag duplication and indirect cache addressing for efficient utilization. The indirect cache addressing can be achieved by employing separate tag array and data array for the cache. The tag duplication is used to provide uninterrupted execution during reconfiguration. FLexTCP[22], partitions the distributed shared LLC into clusters and the flexible capacity partitioning is done across the clusters, meant for tiled CMPs. Clusters consists of cache slices, in high cache demanded applications can steal capacity from other clusters if they are utilized much and the cache blocks are distributed across all L2 slices based on set-duelling approach. Cache sensitive applications can flush their replaced blocks into victim tiles. It reduces the average on chip cache access latency and incurs minimal dynamic repartitioning cost and takes more time for table lookup for mapping block to tiles in the cluster.

3.2. Page Coloring Based Partitioning

The software based approach requires the support of Operating system for making and effective implementation. Some of the methods are described below. COLORIS[28**],** a page coloring based partitioning mechanism that can be applied to the environment where there is more executable threads having varying memory requirements, than processor cores. This consists of a framework for management of memory that involves both static and dynamic partitioning. The distinct coloring can be applied to each cache partitions, whose numbers depends on the number of cores which are applied to each cores as local cache and thus provides cache isolation between processes. Where as, in light weight dynamic partitioning[29], coloring is done only on the dynamically allotted pages to the applications with the help of malloc allocator. Different applications allocate pages in different color in the color set, hence their accessed region in shared cache is partitioned based on the partition policy. The color set is changed adaptively with the change in application type whose categorization is based on the sensitivity to the cache memory. A slightly modified scheme is used in CAP [30] that allocates tasks to cores by considering the access patterns of the cache. The tasks having the same color can be grouped together and assigned to a processor core not individual task thereby reducing the interference due to self evictions and pre-emption delay. The scheduling of tasks is done based on various heuristics. The colors can be reclaimed in [10] or cache can be repartitioned if some colors are not needed or some are not sufficient, based on the cache utilization monitor. The reallocation of pages to the processes is done by determining the degree of hotness of a particular page and various policies. But in [11], changing the partition is done by changing the color-set of each allocated page for the application. Frequently allocated pages are assigned in a different color.

A scalable software defined cache called jigsaw is proposed by Nathan Beckmann et.al in[23].The scheme considers the LLC as a set of distributed banks similar to NUCA, that are logically partitioned as bank partitions or shares and lets the software to collect these shares and allocate data to it based on its classification. Dynamic reconfiguration of shares in terms of sizing in jigsaw based on the application demand, and this requires the UCP algorithm for monitoring the usage. It reduces latency and maximizes the hits by controlling the capacity of the cache. As far as longer cycles are concerned, the reconfigurations degrade the performance.

3.3. Hybrid Memory Based Partitioning

Majority of partitioning techniques on LLC focused on energy minimization. The hybrid memory technology based partitioning techniques are proposed in [32],[33]and [34] that aims to minimize the energy. The[32] and [34] uses STT-RAM and SRAM mix as hybrid last level cache(LLC) based on NUCA architecture, whereas the [33] uses DRAM along with NVM as the hybrid main memory. The selection of the partition is based on the utility. This uses way based partitioning, in which majority of ways are allocated to the STT-RAM and a very few to SRAM. The cache tags are of SRAM tags only. A Block is placed either in STTRAM or SRAM on read miss or write miss with the assumption that the former is read efficient and the latter is write efficient. The selection of victims on replacement is done based on LRU policy in [32,33,34]. This employs read-write aware cache architecture [31] which produces reduction in miss rate and energy of about 3.8% and 11% respectively. The [34] minimizes the high write pressure on STTRAM by providing two access aware policies, and thus improves the write utilization. The [33] is a hybrid memory aware partitioning (HAP) technique that splits the LLC into DRAM lines and low latency NVM lines depending on the source of data from the main memory based on the technology used.HAP sets appropriate bits either 1 or 0 in each line depending upon whether the data is from DRAM or NVM.HAP will be able to adjust the partition size dynamically.

4. COMPARISON BETWEEN VARIOUS PARTITIONING SCHEMES

The tables 1, 2 3 and 4 shows an analysis of various partitioning schemes based on their characteristics, memory type replacement policy performance, merits and demerits. From the review we can say that most of the partitioning techniques works on simulated environment of set-associative cache with LRU based replacement policy with the exception in[1],[14],[15] and [18]. Some partitioning schemes in [28-30], works on page level and that needs operating system support for allocation and replacement. By employing these methods on the LLC, the utilization is improved considerably and thus achieves reduction in energy.

Table 2
Dynamic Partitioning Schemes

Technique	*Approach*	*Memory Type*	*Replacement Policy*	*Simulation Tool*	*Performance*	*Merits*	*Demerits*
Cache partitioning with locking and task partitioning[6]	Equal L2 partitions and task assign ment is based on execution time, that can adjusts size after assignment with locking	Direct or Fully associative	NA	ARM11 architecture with abSInt tool	Reduction in WCET	No cache interference. Minimal capacity, conflict and Inter-core cache misses	Dependencies between tasks are not considered
NuCache[7]	Partitioned into deliways and Mainways. access by all lines in Mainways	Shared set associative memory	PC based	M5	Speed up of around 33% for 8 core processors	Hardware complexity doesnot increases with increase in cores and size	Multi threaded work loads are not considered.
Vantage[8]	Partition into managed and unmanged regions based on line.	Z cache Or set associiative cache	Time stamp based LRU	x86-64 simulator based on Pin	Efficient partitioning of unmanged regions. Good replacement.	Avoids interference and thus scalable. Good isolation	Restricts the size of partitions
Dynamic Partitioning[9]	Partitioning based on bloom filter with updated replacement policy	Shared set associative cache	LRU	Multi2Sim	Reduced miss rate	Improveslocality of reference	Hardware overhead. Consumes more software resources.
Fairness of shared caches [10]	Entire cache is partitioned per thread and a fairnees value is assigned per partition	–	Set Associative cache	Simics	Balanced execution in threads. The performance improvement in terms fairness	Increase the fairness of concurrently executing multithreads	Improved fairness will cause overhead in efficiency

(*contd...*)

(*Table 2 contd...*)

Technique	*Approach*	*Memory Type*	*Replacement Policy*	*Simulation Tool*	*Performance*	*Merits*	*Demerits*
PRETI[11]	Splits the cache space as private and shared caches based on the tasks accessibility	Set Associative	LRU	NA	Achieves schedulability and tighter WCET esti- mates	Enables high performance on all the tasks,	Extra logic is needed for the selection of the replaced block
WCP[12]	Based on write back aware policy that distributes the write band width evenly across applications	Set associative shared L3 cache	LRU	Simics	Improves throughout by 21% write reduction on PCM by 49%	Reduced number of write backs and misses.	Hardware overhead due to storage of u tility and weights
Cooperative Partitioning [13]	Co-operative migration of ways after partitioning into managed and unman- aged partitions.	Fully Asso- ciative	Modified LRU	Mars-x86Cacti	Dynamic and staticenergy savings	High performance. Faster trans- ferring of ways	Consumes more power as it needs extra circuitry.
Drowsy cache partitioning [14]	First partition is in high voltage and is accessed first, and the second partition is in drowsy mode and is accessed when the data is not found in the first one	Set associative cache	MRU	Multi2sim with CACTi	Average dynamic energy savings 20% Leakage energy 45%	Dynamically change the ways between caches.High temporal locality	Extra latency due to partitioning
RECAP[16]	Partitions the data within the cache into shared and private regions.	NUCA	LRU	gem5 and CACTi	Achieves dynamic and static energy savings.	No inter- ference. Reduced Cache contention. Energy mini- mization	Consumes more power because of the extra circuitry for moni- toring and partitioning. Extra latency for cache accesses.
CaPPS[17]	Cache to be organized as private to a particular core and shared partially by other cores	Set associative	LRU	Gem5	Reduces the average LLC miss rates by 25% and 17% as compared to baseline con- figurations and private partiti- oning	Low hardware overhead for CMPs	No mech- anism to handle unused ways
Phase change detection based partiti- oning[18]	Static and dynamic parti- tioning appro- aches to co- schedule both	Set Associative cache	Phase change based replace- ment	Proto typed on Intel's Sandy Bridge	Average energy improvements 12% and thro- ughput improv- ements 60%	Improves the utilization. Preserves the responsiveness	Adding more applications increases the contention for cache and

(*contd...*)

(*Table 2 contd...*)

Technique	*Approach*	*Memory Type*	*Replacement Policy*	*Simulation Tool*	*Performance*	*Merits*	*Demerits*
	foreground applications and back ground applications.						bandwidth
Imbalanced Cache Partiti-oning[19]	Way Partiti-oning shared LLC for balanced data –parallel applications.	Set associative cache	LRU	Simics GEMS	17% drop in miss rate8% drop in execution time	Prioritize each thread.	Scalability, Limited exploration of solution space
Flexi Way[20]	Logically divides the cache set into modules.	Shared L2	LRU	Sniper	Average energy saving of 22.4 % on quad-core machine	No unfair slow downs. Fine grained cache reconfiguration	Some valid blocks may lost due to turning off
DR-SNUCA [21]	Dynamic repartitionable shared cache with indirect addressing and set partitioning.	Set asoociative NUCA	LRU	PTLsim with DRAMSim	Reduced average L2 energy Consumption.	Energy efficiency and high cache utilization and reource guarantees.	Penaltyis due to indirect cache addressing.
FLexTCP[22]	Partitioning for clustered Tiled CMPs that partitions and resizes the cache based on the demand.	Fully associative cache	Victim tile lookup and tile migration	Simics Full system simu-lator	Improvements in Weighted Speedup	Reduce the average on-chip cache access atency minimal dynamic re-partitioning cost.	Randomly selects the processes for allocating the cache space. More time for table lookup
Futility scaling[24]	Partition the whole cache while main-taining high associativity even with a large number of partitions. Adjust the evictions of different partitions	Set associative, NUCA	Time stamp based LRU	Sniper simulator	FS improves performance from 6.0% - 13.7%	Improves the utility of cache capacity. Precise size of the partitions can be maintained.	Storage overhead
RWP[25]	Employing read-write policy that makes the cache into clean(read) and dirty(write only) partitions	Shared Cache	LRU	CMPSim	Provides 5.1%-6.2% of speedup compared to baseline scheme.	Does not require any information from pro-cessors or higher level caches	Extra storage overhead in storing the predictor
Automatic Partitioning [26]	Dynamic way-based cache partitioning at hardware level, building task-level time-	Unified cache	FIFO	Altera DE5 board with Statrix V FPGA	On anaverage reduce 14.93% and 12.56% cache miss on 2-core and 4-core architec	Reduces cache misses.Task level portioning of caches is done auto-matically.	Extra Circuitry is needed for cache and its way management

(*contd...*)

(*Table 2 contd...*)

Technique	*Approach*	*Memory Type*	*Replacement Policy*	*Simulation Tool*	*Performance*	*Merits*	*Demerits*
	triggered reconfigurable cache MPSoCs				tures		
Heterogeneous –aware cache partitioning [27]	The shared cache partitioning adjusts the size based on utility on both fairness and performance	Unified fully associative	LRU with fairness	Storage cache simulator together with Disksim	Average fairness and performance improvements.	Fair allocation of cache space among hete- rogeneous applications.	Throughput is less as the reuse distance is uniform

Table 3
Page Coloring Based Partitioning

Technique	*Approach*	*Memory Type*	*Replacement Policy*	*Simulation Tool*	*Performance*	*Merits*	*Demerits*
COLORIS [28]	Assign differ- ent color for different partitions.	Set a ssociative L3 cache	Lazy recoloring	Prototyped system	Reduces the interference in co-running workload	Eliminate cache conflict misses	Page migration re coloring overherad. Restricts memory space.
LightWeight dynamic partitioning [29]	Malloc allocator based dynamic cache partitioning with page coloring	Set associative	Minimum distance page copying	Operates n linux kernel 2.6.32	Improves performance of applications by 14.28% recol- oring cost is reduced by 55%	Reduces the overhead in recoloring.	Page copying strategy only works with application of larger data set and shorter reuse distance.
CAP[30]	Assigns tasks to cores by considering their usage of cache partitions	Shared L3 cache	–	Done on EPOS RTOS	It produces only 20% on deadline misses for HRT	Can avoid deadline misses	Requires OS support at a higher level pre-emption delay

Table 4
Hybrid Memory Based Partitioning

Technique	*Approach*	*Memory Type*	*Replacement Policy*	*Simulation Tool*	*Performance*	*Merits*	*Demerits*
Energy efficient hybrid cache partitioning[32]	Assigns cache blocks to a specific region of a cache based on region-based hybrid cache architecture	SRAM+ STTRAM (NUCA)	–	Gem5	Energy savings of hybrid cache is 11.0%, system	Improves the performance of the multi-core	The write delay and eviction on miss are not addressed
HAP[33]	Logically divides the cachespace	DRAM+NVM (NUCA)	LRU	Gem5	Improves the performance over LRU by	Reduces memory references	Cost of a miss Cost of a miss in NVM is

(*contd...*)

(*Table 2 contd...*)

Technique	*Approach*	*Memory Type*	*Replacement Policy*	*Simulation Tool*	*Performance*	*Merits*	*Demerits*
	into two partitions for DRAM and NVM lines.				54.3%		higher. Storage overhead
High endurance Hybrid cache [34]	Wear out-aware dynamic cache partiti-oning scheme	SRAM+ STTRAM (NUCA)	–	Gem5	89 times improvement incache lifetime over SRAM. Reduced energy consumption of 58%	Handles write pressure	Selection of replacement on a miss is not taken care.

5. CONCLUSION

This paper has presented an overview of various cache partitioning techniques, particularly in the area of chip multiprocessors. Many of these techniques attempt to reduce one or more issues related to caches, such as utilization, energy minimization, isolation, interference, fairness and worst case execution time by compromising some others. Most of these techniques are tested for its correctness based on different simulators. Among all these partitioning techniques the dynamic way partitioning techniques perform better, when compared to other. The page coloring based techniques requires operating system support to allocate the cache blocks and it restricts memory sizes and very complicated also. The dynamic partitioning methods is a promising approach for reducing the energy and life time of currently emerging hybrid memory paradigm. As the future research directions are for many core CMPs and GPUs to reduce the energy and to improve the utilization, these techniques will throw some insights into the development of an efficient cache system which maintains dynamic non uniform access and non volatility.

REFERENCES

[1] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, Kunyung Chang, "The Case for a Single-Chip Multiprocessor", *In Procedings of seventh International Symposium on Architectural Support for Programming Languages and Operating System,1996.*

[2] Hemant Salwan, "Eliminating Conflicts in Multi level CacheUisng XOR-Based Placement Techniques*", In Procedings of 2013, Internationa Conference onHigh Performance Computing &Communications, pp. 198-203.*

[3] Chih Yung Chang, Jang Ping Sheu, Hsi Chiuen Chen, "Reducing Cache Conflicts by Multi Level Cache Partitioning and Array Elements apping"*The Journal of Supercomputing, 22, 2002, pp.,197,219.*

[4] Lin et al., "Gaining Insights into Multi-Core Cache Partitioning: Bridging the Gap between Simulation and Real Systems," *In Procedings of the 14th IEEE International Conference on Hight Performance Computer Architecture(HPCA), 2008.pp. 367-378*

[5] Weixun Wang, Prabhat Mishra and Sanjay Ranka, "Dynamic Cache Reconfiguration and Partitioning for Energy Optimization in Real-Time Multicore **Sy**stems", In *Procedings of the 48th ACM Internatinal Conference on Design Automation (DAC) 2011, pp. 948-953*

[6] Tiatian Liu, Yingcho Zhao, Minming Li, Chun Jason Xue, "Joint task assignment and cache partitioning with cache locking for wcet minimization on MPSoC", *Journal on Parallel and Distributed computing,71, 2011, pp. 1473-1483*.

[7] R.Manikantan. Kaushik Rajan, R.Govindarajan, "Nucache: An Efficient Multicore Cache Organization Based on Next-Use Distance*", In Procedings of the 17th International Conference on High performance Computer Architetcure(HPCA),2011, pp. 243-253*

[8] Daniel Sanchez, Christos Kozyrakis, "Vantage: Scalable and Efficient Fine-Grain Cache Partitioning", *In Procedings of the 38th annual international symposium on Computer architecture (ISCA'11), pp. 57-68,2011,*

[9] Yang Zhang "Dynamic Cache Partitioning for Multi-core Systems", *MS thesis,2011*

[10] Fang Juan, Pu Jiang, "Fairness of Shared Cache Dynamic Partitioning in CMP*", International Journal of Advancements in Computing Technology July 2012, Vol4, Issue2, pp. 44.*

[11] Benjamin Lesage, Isabelle Puaut, André Seznec, "PRETI: Partitioned REal-TIme shared cache for mixed-criticality real-time systems*", In Procedings of the 20th International Conference on Real-Time and Network Systems RTNS2012.pp. 171-180.*

[12] Miao Zhou, Yu Du, Bruce Childers, Rami Melhem, and Daniel Mosse **"**Writeback-Aware Partitioning and Replacement for Last-Level Caches in Phase Change Main Memory Systems", *ACM Transactions on Architecture and Code Optimization, Vol. 8, No. 4, Article 53, 2012, Pp. 53.1-53.21.*

[13] Karthik T. Sundararajan, asileios Porpodas, Timothy M. Jones, Nigel P. Topham, Bj¨orn Franke, **"**Cooperative Partitioning: Energy-Efficient Cache Partitioning for High-Performance CMPs", *In Procedings of the IEEE 18th International Symposium on High Performance Computer Architecture (HPCA), 2012, pp. 1-12.*

[14] B. Fitzgerald, S. Lopez, and J. Sahuquillo, "Drowsy Cache Partitioning for Reduced Static and Dynamic Energy in the Cache Hierarchy", *In Procedings of the IEEE International Conference on Green Computing(GCC), 2013, pp. 1-6.*

[15] Pan Lai, Rui Fan, "Makespan-Optimal Cache Partitioning", In Procedings of the 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunicatin Systems,2013, pp. 202-211.

[16] Karthik T Sundararajan, Timothy M Jones, Nigel P Topham, "RECAP:Region-Aware Cache Partitioning", *In proceedings of the IEEE 31st International Conference on Computer Design (ICCD 2013),2013, pp. 291-304*

[17] Wei Zang and Ann Gordon-Ross, "Analytical Modelling of Partially Shared Caches in Embedded CMPs*", In Procedings of the 7th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies(UBICOMM) 2013.*

[18] Henry Cook, Miquel Moreto, Sara Bird, Khanh Dao, David A Patterson, Krste Asanovic, "A Hardware Evaluation of Cache Partitioning to Improve Utilization and Energy- Efficiency while Preserving Resonsiveness", *In Procedings of the 40th Annual International Symposium on Computer Architecture(ISCA'13).2013, pp. 308-319.*

[19] Abhishek Pan, Vijay S Pai, "Imbalnced Cache Partitioning for Balanced Data-Parallel Programs", *In Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture(MICRO'46),2013, pp. 297-309*.

[20] Sparsh Mittal, Zhao Zang, Jeffrey S.Vetter, "FlexiWay:A cache energy saving technique using fine grained cache reconfiguration", *In Procedings of the 31st International conference on Computer Design(ICCD)2013, pp. 100-107.*

[21] Ansuman Gupta, Jack Sampson, Michael Beford Taylor, "DR-SNUCA: An Energy-Scalable Dynamically Partitioned Cache", *In Procedings of the IEEE 31st International Conference on Computer Design(ICCD),2013, pp. 515-518*.

[22] Ahmad Samih, Xiaowei Jiang, Liang Han, and Yan Solihin, "Flexible Capacity Partitioning in Many-Core Tiled CMPs", *In Procedings of the 16th International symposium on Cluster, Cloud, and Grid Computing(CCGRID'2013), pp. 490-497.*

[23] Nathan Beckmann and Daniel Sanchez, "Jigsaw: Scalable Software-Defined Caches", *In Proceedings of the 22nd international conference on Parallel architectures and compilation techniques (PACT'13), pp. 213-224.*

[24] Ruisheng Wang, Lizhong Chen, "Futility Scaling:High-Associativity Cache Partitioning", *In Proceedings of the 47th IEEE./ACM International Symposium on Microarchitecture,2014, pp. 356-367.*

[25] Samira Khan, Alaa R. Alameldeen, Chris Wilkerson, Onur Mutluy "Improving Cache Performance Using Read-Write Partitioning", *In Procesings of the IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), 2014 pp. 452 – 463*

[26] Gang Chen, Biao Hu, Kai Huang, Alois Knoll, Di, Liu, Todor Stefanov, "Automatic cache partitioning and Time-Triggered scheduling for Real-time MPSoCs", *In proceedings of the IEEE International Conference on Reconfigurable Computing and FPGAs(ReConFig),2014.pp. 1-8*

[27] Yong Li, Dan Feng, Zhan Shi, "Heterogeneous –aware cache partitioning:Improving the fairness of shared storage cache" *Elsevier Journal on Parallel Computing, 2014, pp. 710-721*.

[28] Ying Ye, Richard West, Zhuoqun Cheng and Ye Li, "COLORIS: A Dynamic Cache Partitioning System Using Page Coloring", *In proceedings of the 23rd international conference on Parallel architectures and Compilation (PACT'14), 2014, pp..381-392*.

[29] Luden Zhang, Yi Liu, Rui Wang, Depei Qian, "Lightweight dynamic partitioning for last –level cache of multicore processor on real system*", Journel of Supercomputer, Springer, 2014, pp. 547-56.*

[30] Giovani Gracioli, Antonio Augusto Frohlich, "CAP: Color-Aware Task Partitioning for Multicore Real-Time Applications" *In proceedings of the IEEE International Conference on Emerging Technology and Factory Automation (ETFA), 2014, pp. 16-19*

[31] Anthony Gutierrez, Ronald G. Dreslinski, Trevor Mudge, "Evaluating Private vs. Shared Last-Level Caches for Energy Efficiency in Asymmetric Multi-Cores*", In Procedins of 2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV),2014, pp. 191-198.*

[32] Dongwoo Lee and Kiyoung Choi, "Energy-Efficient Partitioning of Hybrid Caches in Multi-Core"*In the Procedings of the 22nd International Conference on Very Large Scale Integration(VLSI-SoC) 2014, pp. 6-8.*

[33] Wei Wei, Dejun Jiang, Jin Xiong, Mingyu Chen, "HAP: Hybrid-memory-Aware Partition in Shared Last-Level Cache"*In Procedings of the 32nd International Conferenece on Computer Design(ICCD)2014, pp. 28-35.*

[34] Ing-Chao Lin, Jeng-Nian Chiou, "High-Endurance Hybrid Cache Design in CMP Architecture With Cache Partitioning and Access-Aware Policies", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 23, Issue.10, Oct.2015, pp. 2149-2161.*