Journal of Computer Science 7 (10): 1448-1457, 2011
ISSN 1549-3636
© 2011 Science Publications

Composite Pseudo Associative Cache with Victim Cache for Mobile Processors

Lakshmi Deepika Bobbala, Monobrata Debnath and Byeong Kil Lee
Department of Electrical and Computer Engineering,
University of Texas at San Antonio, San Antonio, Texas, USA

Abstract: Problem statement: Multi-core trends are becoming dominant, creating sophisticated and complicated cache structures. One of the easiest ways to design cache memory for increasing performance is to double the cache size. The big cache size is directly related to the area and power consumption. Especially in mobile processors, simple increase of the cache size may significantly affect its chip area and power. Without increasing the size of the cache, we propose a novel method to improve the overall performance. **Approach:** We proposed a composite cache mechanism for 1 and L2 cache to maximize cache performance within a given cache size. This technique could be used without increasing cache size and set associatively by emphasizing primary way utilization and pseudo-associatively. We also added victim cache to composite pseudo associative cache for further improvement. **Results:** Based on our experiments with the sampled SPEC CPU2006 workload, the proposed cache mechanism showed the remarkable reduction in cache misses without affecting the size. **Conclusion/Recommendation:** The variation of performance improvement depends on benchmark, cache size and set associatively, but the proposed scheme shows more sensitivity to cache size increase than set associatively increase.

Key word: Associatively increase, mobile processors, power consumption, pseudo-associativity, increasing cache, replacement policy, remarkable reduction, composite pseudo, direct memory, further improvement

INTRODUCTION

While processors with multi-cores already thrived in general purpose processor area, mobile processor companies are recently starting to release their multi-core version which are used in net books, smart phones or Tablet PCs. Many design issues presented in general-purpose processors are more critical in mobile processors. Especially, power and heating problems are the prominent issues in battery operated mobile devices. In order to maximize the effectiveness of applying multi-core technologies to mobile processors, those problems need to be solved with appropriate solutions for future mobile processor designs.

In general, memory subsystem takes a large portion of the die area in the microprocessors and caches consumeover 40% of a processor's total power (ShivaKumar and Jouppi, 2001). The reduction of cache size and power consumption is one of the main design goals for mobile computing devices.

One of the easiest ways to design cache memory for increasing performance is to double the cache size. In mobile processors, however, simple increase of the cache size significantly affects chip area and power. As multi-coretrends are becoming dominant, cache structures turns outto be sophisticated and complicated.

The bigger shared level-2 (L2) caches are demanded for higher cache performance, but the big cache size is directly related to the area of interconnection and related power consumption. Similarly, higher performance L1 caches are required without increasing the size of the cache. As shown in Fig. 1, cache performance can be significantly increased by doubling cache sizes and increasing set associatively, but it results in hardware cost, larger area and power consumption.

To address this issue, in this study, we propose a composite cache mechanism to maximize cache performance within a given cache size. We have also added victim cache for further improvement in performance. Generally, not all the sets require same associatively most of the time and the utilization of the ways is biased to the first way (Abella and

Corresponding Author: Lakshmi Deepika, Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, Texas, USA

González, 2006). V-way cache is one of the pseudo associative techniques (Qureshi, *et al*., 2005). This technique can be used without, size and set associatively by increasing cache utilization and pseudo-associatively.

In our experiments, we use the SPEC CPU2006 benchmark suite for simulation workloads, since current mobile internet devices (e.g., net book) are required to run same application that are used in general-purpose processor.

Based on our experiments with the sampled SPEC CPU2006 workload, the proposed cache mechanism shows remarkable reduction in cache misses.

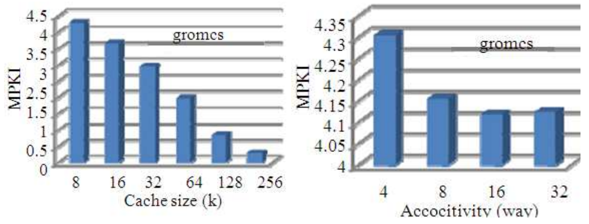

Fig 1: Cache misses (misses per kilo instruction) with cache size and set-associativity variations

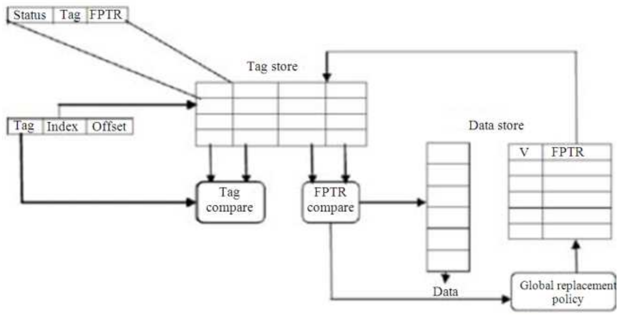

Fig. 2: V way Cache with 4-way set associativity

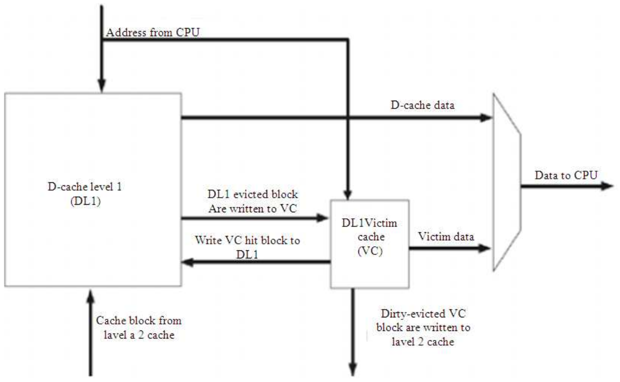
Address from CPU
D-cache data
Data to CPU
D-cache level 1 (DL1)
DL1 evicted block Are written to VC
Write VC hit block to DL1
DL1Victim cache (VC)
Victim data
Cache block from lavel a 2 cache
Dirty-evicted VC block are written to lavel 2 cache

Fig. 3: Victim cache structure

Related work: There have been several approaches to investigate cache organizations on level-1 and level-2 caches for CMPs (Qureshi, *et al*., 2005; Liu *et al* 2004; Davanam and Lee, 2010). Here we are describing some of the related techniques.

V-way cache: (Qureshi *et al*., 2005). Proposed Utility-based Cache Partitioning which is a low overhead, high performance, runtime mechanism to partition shared level-2 caches. This work achieved improvement of miss rate for L2 cache by choosing a pseudo associative cache structure and implementing global replacement policy. They proposed a novel cache structure in which they have separate tag and data stores as shown in the Fig. 2. In their cache structure, the tag and data store are independent of each other. The number of tag entries is doubled so as to provide more space for tags. The entries may be tripled but increase in size also needs to be taken into consideration. Generally, if the tag store is doubled, the increase in size of cache will be 2 to 3% **(**Qureshi *et al*., 2005). Each tag will be having a valid bit and dirty bit to show the status and FPTR which will be pointer associated with the tag pointing to the unique data in data store. For data store, each entry will have availed bit and RPTR which will be pointing to unique tag entry. As the tag and data structure are decoupled, data can be mapped to tag globally and there will not be one tone corresponding relationship between them. As each set will be having different demand, this global mapping will help to reduce the miss rate. Form this technique; they achieved an average miss rate reduction of 13% **(**Qureshi *et al*., 2005).

Heterogeneous way size cache: (Abella and González, 2006). Proposed a Heterogeneous Way-size Cache, in which different cache ways have different sizes. They applied this mechanism to L1 and L2 caches with dynamically adaptive version. Their experiments proved that only a small fraction of sets require some associativity most of the time and the numbers of sets that make effective use of given degree of associativity decreases as the associativity increases **(**Abella and González, 2006). They designed a heterogeneous way size cache based on this observation such that different ways can have different sizes and number of sets in each way should be power of 2.

Victim cache: A victim cache, as shown in Fig. 3, is a small fully set associative cache used along with L1 cache to improve the miss rate. Even though there is a tradeoff of area and delay due to the use of victim cache, the reduced miss rate will compensate them. That is why we generally use 4-6 victim lines so that delay and area required do not dominate the miss rate reduction.

A fully associative software managed cache design: Indirect index cache (Hallnor *et al*., 2000) used a novel replacement algorithm called generational replacement and give the miss rate performance nearly equal to fully associative cache.

Related works: (Liu *et al*., 2004). Proposed Shared Processor-based Split Leaches, statically allocating some private banks to each competing applications based on profile information. It might be problematic to profile if several applications are executed concurrently. While some research results focus on reducing access time (Jouppi, 1990; Peir, *et al*., 1998; Puzak, 1985; Seznec, 1993; Batson and Vijaykumar, 2001: Chishti *et al*., 2003), other approaches are based on predicting the way where the data is stored (Belady, 1966; Calder and Elmer, 1996; Inoue *et al*., 1999). The adaptive group associative cache (Peir, *et al*., 1998) proposes to improve the performance of first level cache but its benefit reduces by increasing associativity. Prime modulo hashing (Jouppi *et al*., 1990) and skewed associativity (Seznec, 1993). Suggests distributing memory access uniformly across cache sets by targeting the indexing function but they suffer from negative effects of local data replacement due to static mapping of tag entries to data lines.Satisg

Composite pseudo associative cache with victim cache: The work presented here expands on the initial work done in (Qureshi *et al*., 2005). We have modified such that the cache structure is applicable for both L1 and L2 caches.

Limitations on performance improvement: Modern microprocessors including general purpose processors and embedded/mobile processors need to run wide range of applications. While some applications are sensitive and increased their performance as cache size increases, some other applications might have less sensitivity or show the saturation in performance improvement. As shown in Fig. 1, applications show the performances alteration (or small improvement) on the cache size increase and set associativity increase. This means cache size increase is not a perfect solution for all application workloads because each application has different 3architectural behavior to hardware resources. Especially for battery-powered mobile processors, cache size increases hold not is considered as a high priority solution due to higher power requirement and larger area requirement. In Sock design using IPs, another simple way to improve cache performance is to increase set associativity with the penalty of additional hardware cost and more power consumption. Full-way set associative cache with ideal replacement method can provide large performance improvement, but it is impractical to be implemented. By increasing cache set associativity, we can expect certain level of performance improvement, but still the degree of improvement is saturating at some point in most of the applications.

Cache way (sub-array) utilization: Generally, LRU replacement policy is popularly used in cache designs with several methods of implementations. However, its way (physical way in sub-arrays) utilization can be categorized into two patterns. A half of benchmarks (e.g., vortex) in SPEC CPU2006 show the biased way utilization with LRU replacement policy; the rest of benchmarks (e.g., bzip2) show relatively balanced distribution. Based on the observations, not all the sets require some associativity most of the time and the utilization of the ways are biased to the primary way(s) (Abella and González, 2006). Several schemes have been introduced to address this issue such as heterogeneous way-size cache, in which different cache ways have different sizes. Most of them need complex logics to handle different size of ways and to add adaptive features.

Composite pseudo associative cache: The proposed composite pseudo-associative cache is designed for level-1 and level-2 caches(Bobbala *et al*., 2010). Based on the study from (Abella *et al*., 2006*)*. Only a small fraction of sets require some associativity most of the time and the utilization of the ways are biased to the first way. As the associativity increases, actual set utilization will be decreased more. In order to address this issue, (Abella and González, 2006). Use heterogeneous way-size cache (Abella and González, 2006) ith the penalty of access time and architectural complexity. In this study, however, we proposed composite cache mechanism emphasizing primary way utilization and pseudo-associativity by reconfiguring cache structure. Figure. 4 shows a basic concept of the proposed cache scheme. As the first step, data arrays of the cache are divided into two parts: the first half of data cache for the primary way; the rest of them for all other ways.

However, tag arrays need to be preserved as in traditional cache in order to use the features of set associativity. If one wants to design a 4-way 256KB cache, 128KB will be used for a primary way and another128KB will be shared by other ways. Also, four tag arrays for each way need to be maintained for getting full benefits from 4-way associativity.

However, we need a special mechanism to provide the pseudo-associativity on how to share the other data array by three tag arrays. Several different approaches for pseudo-associativity have been proposed to improve the miss rate of a set associative cache, but in our experiment v-way cache Qureshi *et al.*, 2005) was selected (TDR=2) and integrated into the proposed scheme for enabling the pseudo associativity of non-primary ways.

Operation of composite pseudo oassociative cache: In the primary way, each block will have valid bit for status information, tag bits and FPTR to map to particular data line for each tag. Each time the tag gets accessed, it will update the count that will in turn be used for the choice of tag victim. (Bobbala *et al.*,2010 & Bobbala *et al.*, 2011).

Least Recently Used (LRU) replacement policy is used for the primary way for selecting the tag victim. Data will have RPTR, access and reuse bits such that RPTR can be used to point FPTR using direct mapping. Access and reuse bits can be used to select victims using LRU and reuse replacement policy. Random replacement policy is also implemented in the simulator for choosing data victim. For the primary way, there is one to one correspondence between the tag and data; for example, each tag is mapped to unique data in the data store. For all the other ways, the tag store will have the tag along with valid bit and FPTR. The data will have availed bit, RPTR pointing to one of the tag entries in the tag store. Tag store will use LRU replacement policy to update the replacement information where the data store will use reuse replacement policy (Puzak, 1985). For the both tag data, if the valid bit is unset then the information is considered to be invalid. This is applicable for primary way also. The TDR (Tag to Data Ratio) is taken as 2 through hotel the simulations. This value is taken is 2 because is the optimum value when area, power is also taken into consideration. Figure 5 shows the algorithm that we have implemented.

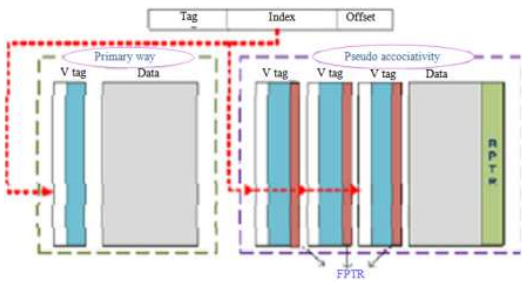

Fig 4: Composite pseudo associative cache with4-way set associative

Operation of composite pseudo associative cache with victim cache: Victim caching is implemented for the primary way of the Composite Pseudo associative cache (CPS). Separate tag and data stores are introduced for victim cache, associativity_vc is added to the input parameter that defines the number of victim lines and fptr-vc is added to the tag to point the data lines in the data store of victim cache. The structure of the CPS cache is kept constant. Figure 6 shows the basic concept and Fig. 7 shows the algorithm of Composite pseudo associative cache with victim cache operation.

Simulations: We modified cache-slim (Burger and Austin, 1997) simulator which is a trace driven simulator to implement our cache structure.

The work load is selected (Table 1) as SPEC CPU 2006(SPEC) Integer and float ing point suite (Qureshi *et al*., 2005). To generate the L1 and L2missed trace information, we used SPEC CPU 2006 integer and floating-point suite and the Simple scalar (Burger and Austin, 1997) Alpha binaries with skipped initialization phase.

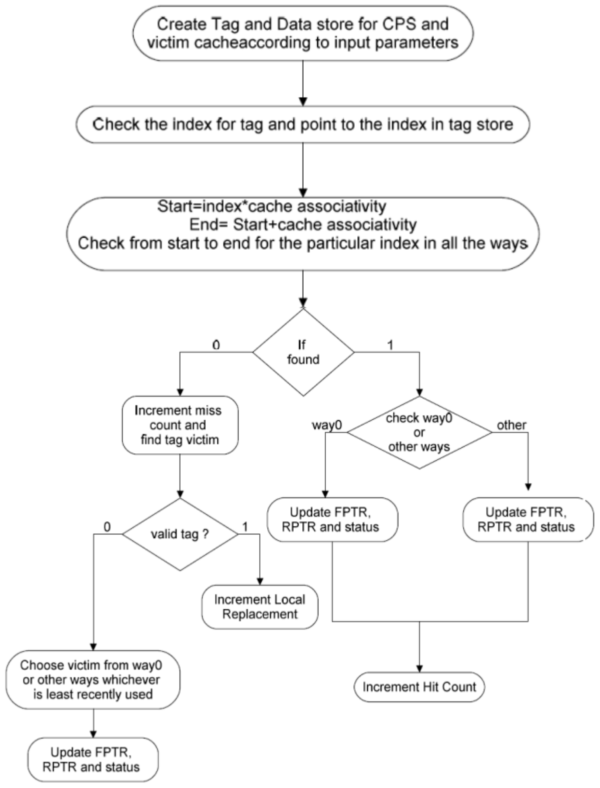
Create Tag and Data store for CPS and victim cacheaccording to input parameters
Check the index for tag and point to the index in tag store
Start=index*cache associativity
End= Start+cache associativity
Check from start to end for the particular index in all the ways
If found
0
1
Increment miss count and find tag victim
way0
check way0 or other ways
other
Update FPTR, RPTR and status
Update FPTR, RPTR and status
valid tag ?
0
1
Increment Local Replacement
Choose victim from way0 or other ways whichever is least recently used
Increment Hit Count
Update FPTR, RPTR and status

Fig. 5: Algorithm for composite-pseudoassociative cache

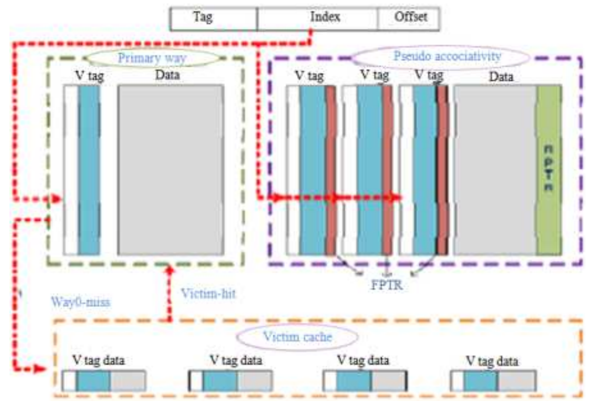

Fig. 6: Composite pseudo associative cache with victim cache

Table 1: Description of SPEC2006 benchmarks

Benchmark	Short description
Astar	It is derived from a portable 2D path-finding library that is used in games artificial intelligence
Bzip2	Compression application which is to help isolate the workdone to only the CPU and memory subsystem
Gobmk	Artificial Intelligence i.e., game playing
Hammer	Protein sequence analysis
Libquantum	A library for the simulation of a quantum\computer
Mcf	Application for combinatorial optimization/single-depot vehicle scheduling
Perl	A cut-down version of Perl v5.8.7,\the popular scripting language
Sjeng	A program that plays chess and several chess variants, such as drop-chess
Bwaves	Computational Fluid Dynamics
Leslie3d	The primary solver used to investigate a wide array of turbulence phenomena
Grimaces	Chemistry of molecular dynamics Gems FDTD Computational Electromagnetics
Milc	Application for simulations of four dimensional SU (3) lattice gauge theory on MIMD parallel machines
Namd	A parallel program for the simulation of large bio molecular systems
Soplex	A linear program using the Simplex algorithm
Zeusmp	An application to solve problems in three spatial imensions with a wide variety of boundary conditions

First 500 million instructions were fast forwarded and the following 500 million instructions are simulated with the ref input data sets. The baseline configuration will be normal cache and v way cache. The results for the traditional cache are obtained by giving Tag to Data Ratio (TDR) = 1 for the v way cache. This result is compared against the results of the composite pseudo associative cache (cps) and composite pseudo associative cache with victim caching (cpsv).

Miss rate Per Kilo Instruction (MPKI) and miss rate are used as measurements for miss rate. Two types of simulation analysis are performed.

A processor model that is considered in the v way cache (Qureshi *et al*., 2005) is used and the results are evaluated. This processor has16KB, 64B line size and 2-way set-associative L1 cache.

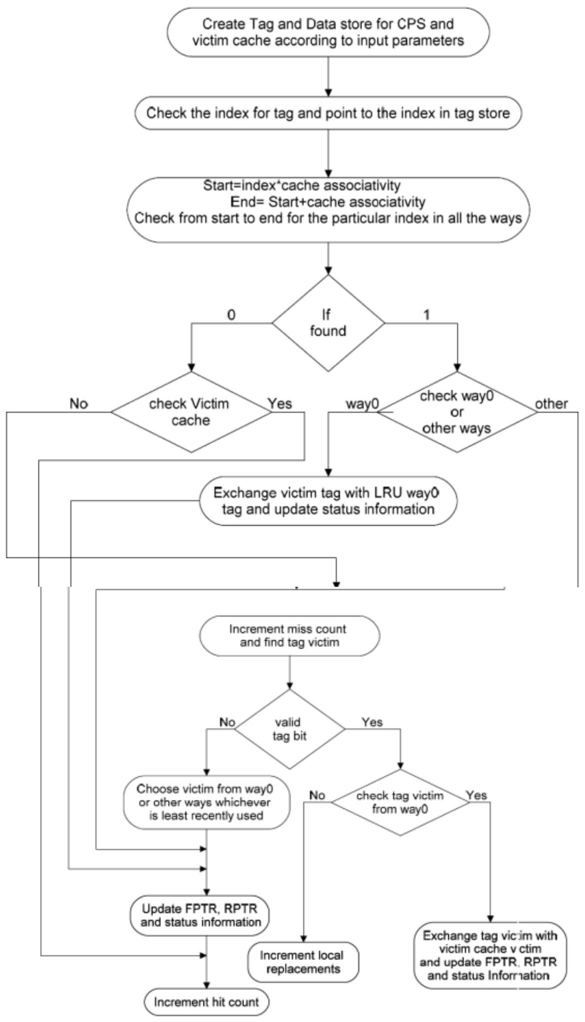
Create Tag and Data store for CPS and victim cache according to input parameters
Check the index for tag and point to the index in tag store
Start=index*cache associativity
End= Start+cache associativity
Check from start to end for the particular index in all the ways
If found
0
1
check Victim cache
No
Yes
check way0 or other ways
way0
other
Exchange victim tag with LRU way0 tag and update status information
Increment miss count and find tag victim
valid tag bit
No
Yes
Choose victim from way0 or other ways whichever is least recently used
check tag victim from way0
No
Yes
Update FPTR, RPTR and status information
Increment local replacements
Exchange tag victim with victim cache victim and update FPTR, RPTR and status Information
Increment hit count

Fig. 7: Algorithm for composite pseudo associative cache with victim cache

The replacement algorithm for L1 cache is LRU. For Simplex and bzip2, L2 cache is of size 256KB, 128B line size and 8-way set-associative.

For the rest of benchmarks, L2cache is 16KB, 4way set associative and 128B block size because miss rate almost reduced to zero for 256KB size. Individual variations of MPKI of various parameter sure also measured. The parameters for the L1 and L2caches are chosen such that both the extremities are covered.L1 cache simulations are done for cache sizes 8,16, 32, 64 and 128 KB. The associativity is varied as 2, 4 and 8. The line size is varied as 64B and 128B.The associatively of the victim cache is varied as 4, 8, 16and 32. L2 cache simulations are done for cache sizes 8,16, 32, 64, 128, 256, 512 and 1024KB.The 8KB cache is considered for L2 cache simulations because for some of the benchmarks 8KB CPS cache with 32victim lines achieved the miss rate equal to traditional cache of 64KB. All these simulations are done to visualize the miss rate variation across different cache sizes. The associativity is changed as 4, 8, 16 and 32. The line size is kept constant at 128B. L1 and L2 simulations are done for the combinations of all the above parameters for normal, v way, composite pseudo associative and composite pseudo associative with victim cache. The following benchmarks are tested.

RESULTS

Miss rate analysis of results: Graphs for only ceta in benchmarks are captured due to the limited space. In all the graphs (Figs. 8-14), x-axis represents different associativities for different cache sizes and y-axis is showing Misses per Kilo Instruction.

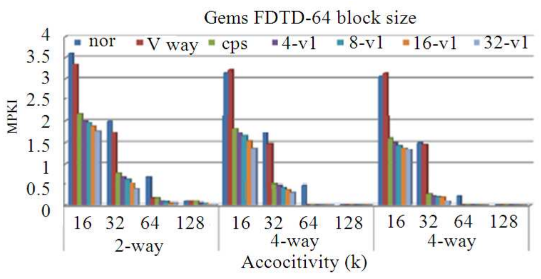

Fig. 8: MPKI with various set-associativity and cache sizes for L1 cache-Gems FDTD

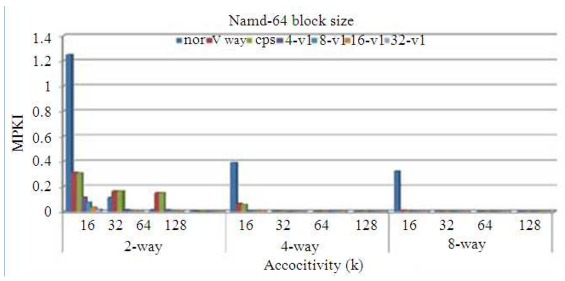

Fig. 9: MPKI with various set-associativity and cache sizes for L1 cache-namd

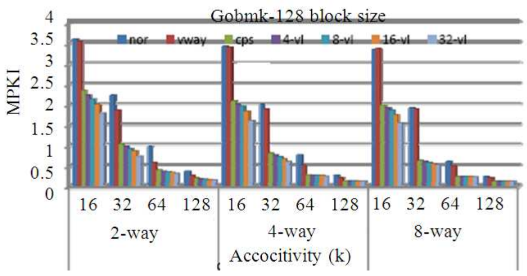

Fig. 10: MPKI with various set-associativity and cache sizes for L1 cache-named

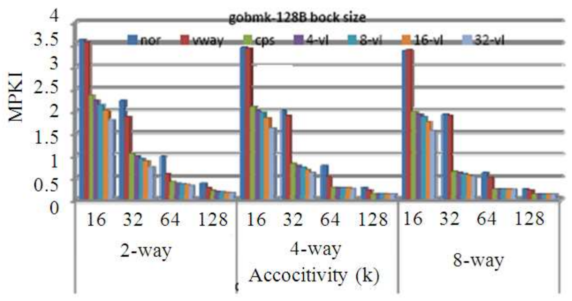

Fig. 11: MPKI with various set-associativity andcache sizes for L1 cache-soplex

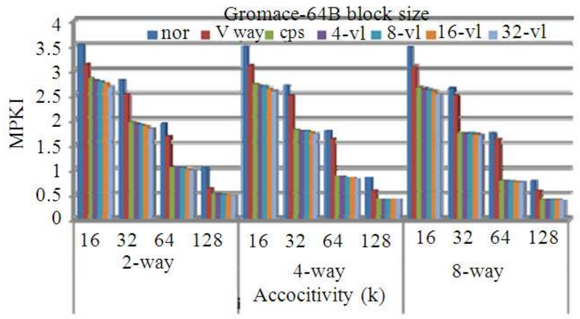

Fig. 12: MPKI with various set-associativity and cache sizes for L2 cache-gromcs

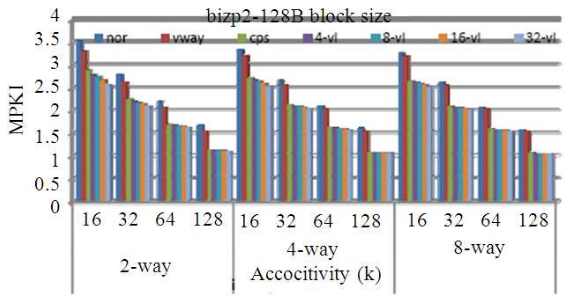

Fig. 13: MPKI with various set-associativity and cache sizes for L2 cache-bzip2

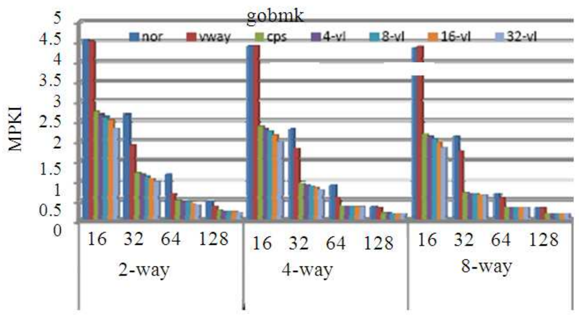

Fig. 14: MPKI with various set-associativity and cache sizes for L2 cache-gobmk

Also, we used following notations:

- Nor: Traditional set associative cache v way: V-way cache
- Cps: Composite pseudo associative cache
- 4-vl: Composite pseudo associative cache with victim cache
- 8-vl: Composite pseudo associative cache with victim cache
- 16-vl: Composite pseudo associative cache with victim cache
- 32-vl: Composite pseudo associative cache with victim cache

Below is the terminology that we used to analyze the miss rate improvement. This is applicable for both L1 and L2cache simulations. Maximum reduction of miss rate measured for both L1 and L2 caches for all the simulations as shown in Table 2-4:

- Intermediate performance-0.5 to 1.0 decrement in MPKI
- Good Performance-More than 1.0 decrement in MPKI
- Less Improvement in performance-Less than 0.5 decrement in MPKI
- No Improvement in Performance-MPKI increased when compared to other structures.
- CPS with neither VC to nor - decrement in miss rate
- From CPS with 32 victim lines to traditional set associative cache.
- CPS with VC to v way-decrement in miss rate from
- CPS with 32 victim lines to v way cache

Observations from L1 cache results: The benchmarks leslie3d, libquantum and mcf are computational intensive benchmarks. The increment in line size reduces MPKI. It is not affected by cache size, associativity or cache structure. This might be due to repetition of particular sized blocks. Increase in cache size is more when compared to the increase in associativity. For the benchmarks Astar, bwaves, bzip2, soplex, hmmer and gromcs, there is an intermediate improvement in MPKI using the composite pseudo associative cache.

For Namd, Gems FDTD and gobmk, there is good improvement inMPKI using composite pseudo associative cache. The benchmarks libquantum, mcf, gromcs, leslie3d and zeugma have less improvement in MPKI using composite pseudo associative cache. Bwaves is the only benchmark which could get good performance improvement by adding victim caching to the CPS.

Observations from L2 cache results: The bench mark named is not considered for analysis. MPKI is reduced to zero at the lowest configuration of cache. For the benchmarks milc and libquantum, increment in line size-reduces MPKI like L1 cache.

Table 2: Analysis of results for L2 cache simulations

Benchmark	Comments
Astar	Overall CPS with victim lines has good performance
Bwaves	Overall CPS with victim lines has good performance
GemsFDTD	Over all CPS has good performance
Gobmk	Over all CPS has good performance
Gromcs	Over all CPS has Intermediate performance
Hmmer	Over all CPS has Intermediate Performance
Leslie3d	Over all CPS has less performance
Libquantum	CPS has no performance improvement
Mcf	CPS has no performance improvement

Table 3: Analysis of Results for L1 cache simulations

Benchmark	CPS with to nor-L1	VCCPS with vway-L1	VCCPS with VC to nor-L2	CPS with VC to vway-L2
Astar	12.22	5.81	15.1	6.2
Bwaves	13.26	14.26	3.5	1
Bzip2	7.79	5.79	8.1	5.25
Gemfdtd	17.98	15.74	--	--
Hmmer	8.1	12.6	5.82	7.42
Gobmnk	22.63	22.63	8	6.35
Gromcs	8.67	4.51	7.08	7.08
Lelie3d	7.87	12.21	3.46	--
Soplex	13.54	9.36	11.68	--
Zeusmp	2.4	2.4	15.7	--
Hmmer	8.1	12.6	5.82	--
Lelie3d	7.87	12.21	3.46	--

Table 4: Results for L1 and L2 cache simulations for the alpha processor configuration

Benchmark	Comments
Astar	Overall CPS with victim cache has good reduction in MPKI
Bwaves	CPS performance is less than victim cache
Gobmk	Overall CPS with victim cache has good performance
Gromcs	Overall CPS with victim cache has good performance
Hammer	Overall CPS with victim cache has good performance
Leslie3d	Overall CPS with victim cache has relatively Less performance improvement
Libquantum	CPS has no performance improvement
Mcf	CPS has no performance improvement
Soplex	CPS has intermediate improvement in MPKI when compa-red to v way
Zeusmp	CPS has less improvement in miss rate when compared to v way
Bzip2	Overall CPS with victim cache has good performance

Table 5: Percentage numbers of hits for 256KB, 8 way associative composite pseudo associative caches for astar benchmark

Way	Number of hits (%)	
0	5959086	(39.21)
1	1349358	(8.88)
2	1331165	(8.76)
3	1401675	(9.22)
4	1305575	(8.59)
5	1288913	(8.48)
6	1288585	(8.48)
7	1273056	(8.38)

MPKI not affected by cache size, associativity or cache structure. This might be due to repetition of particular sized blocks. For rest of the benchmarks, v way showed good performance than CPS at the cache size 8KB and CPS is good than v way when the cache size more than 8KB.This is because as the number of entries in primary way increases as the cache size increases. Then the direct memory part of the CPS cache works well as it gets more data which is having locality.

For bwaves, change in cache size and associativity has no effect. MPKI got reduced only with the use of composite pseudo associative cache structure. Astar, soplex, bzip2, Gombak and hammer**,** there is intermediate improvement in the MPKI. For the benchmarks bwaves, leslie3d and zeusmp, there is less improvement in the MPKI. For gromcs, there is good improvement in MPKI. We observe that, in all the L2 cache simulations, there is less impact of introducing victim cache.

Storage cost and delay analysis for CPS and V way: A physical address space of 36 bits is assumed for the below analysis. Block size of 128 bytes is assumed. Then the number of tag bits will be 36-$\log_2$ (sets)-$\log_2$ (block size). The number of tag store entries is assumed as 2048 and the associativity is 8 for the traditional cache. Table 5 shows the hit rates obtained by composite pseudo associative cache when simulated for astar benchmark whose cache size is 256KB and associativity is 8. It has miss rate of 0.88%. Table 6 describes the storage cost analysis for different types of caches.

From the Table 5, maximum number of hit entries fromway0 comes around 40%. CPS needs an extra multiplexer to select between the different ways. But, it has direct memory cache whose latency will be less when compared to other types of caches. As the number of hit's in way0 is far bigger when compared to that of other ways, the latency of the CPS cache will be less than v way cach.

Table 6: Storage cost analysis for traditional cache, v way cache and composite pseudo associative cache

Storage	Traditional cache	VWAY	CPS
Each tag-store entry Contains	(bits)		
Status (v+dirty+LRU)	5000	5	5
Tag	2100	20	20
FPTR		11	11
Total number of tag bits	2600	36	36
Each data-store entry Contains	(bits)		
Status (v+reuse)		3	3
Data	128*8	128*8	128*8
RPTR		12	12
Total number of bits in data store entry	1024	1039	1039
Number of tag store Entries	2048	4096	
	256+3584=3840		
Number of data store Entries	2048	2048	2048
Size of tag store	6.7KB	18.43KB	17.2KB
Size of data store	256KB	259KB	259KB
Total size of cache	262.7KB	277.4KB	276.2K

This is true for all the benchmarks who achieved intermediate-and good performance compared to that of v way cache because the MPKI has reduced to the increase of number of hits in the direct memory cache. From Table 6, it is evident that CPS occupies less area when compared to v way cache. So, we can achieve the savings in delay and area for the composite pseudo associative cache when compared to v way cache for the benchmarks who achieved intermediate and good performance.

CONCLUSION

The performance of cache is very important in the memory design as it will have huge impact on the speed and power of the processor. As the usage of multi-core processors in mobile devices is becoming prevalent, there is a need for the high performance caches with minimum area. Composite pseudo associative with victim cache is one of such techniques which attempt to increase the performance of the cache without increasing the area. This work is developed from the simulator of v way cache (Puzak, 1985). Composite pseudo associative cache uses direct memory cache as primary way and pseudo associative cache for other ways. A victim cache is added to the primary way of the composite pseudo associative cache for further improvement in the performance. A 16KB, 2-way set associative, 64B block size L1 CPS cache with 32 victim lines outperforms the traditional cache by 16.7% and v way cache by 8.86%. A 256KB, 8-way set-associative, 128B block size L2 cache is considered for soplex andbzip2 benchmarks where as 16KB, 4-way set-associative, 128B block size is considered for rest of the bench marks.This is due to the reduction of miss rate to very less for the other benchmarks at 256KB, 8-way configuration. CPS cache with victim cache achieved an average of 8.7% better per for mance when compared to traditional set associative cache and 5.47% better than v way cache. Simulations for different cache sizes and associativity are done for choosing the best configuration.

CPS cache will also help to reduce the overall latency and size when compared to the v way cache. This in turn will reduce the power. The results from this cache configuration can be effectively used for multi-core designs in mobile processors for which area and power is a major constraint. Future work will concentrate on more effective designs for pseudo associative cache, estimation of exact power and latency (using Wilton *et al.*, 1996) implementing composite pseudo associative cache technique on multi-core simulators and retrieving the exact trace data for multi-core mobile processors.

REFERENCES

Abella, J. and A. González, 2006. Heterogeneous waym size cache. Proceeding of the International Conference on Supercomputing, (ICS '06), ACM New York, USA, pp: 239-248. DOI: 10.1145/1183401.1183436

Albonesi, D. and H. Selective 1999. Cache Ways: On demand cache resource allocation. Proceedings of 32nd Annual International Symposium on Micro Architecture, (MICRO-32), IEEE Xplore Press, Haifa, Israel, pp: 248-259. DOI: 10.1109/MICRO.1999.809463

Batson, B.M. and T.N. Vijaykumar, 2001. Reactive Associative Caches. Proceeding of the 2001 International Conference on Parallel Architectures and Compilation Techniques, Sep. 8-12, IEEE Xplore Press, Barcelona, Spain, 49-60. DOI: 10.1109/PACT.2001.953287

Belady, L.A., 1966. A study of replacement algorithms for a virtual storage computer. IBM Syst. J., 5: 78-101. DOI: 10.1147/sj.52.0078

Bobbala, L.D, Byeong K.L, 2011. Hybrid way cache for mobile processors. Proceeding of the eigth International Conference on Information Technology, Apriel 11-13, IEEE Xplore Press, Las Vegas, NV, USA, pp: 707-712. DOI: 10.1109/ITNG.2011.125

Bobbala, L.D, Salvatierra. J,Lee B.K, 2010 Composite pseudo associative cache for mobile processors. Proceeding of the 18th IEEE International Symposium on Modelling Analysis and Simulation of Computer and Telecommunication systems, Aug. 17-19, IEEE Xplore Press, Miami Beach, FL, pp: 394-396. DOI: 10.1109/MASCOTS.2010.49

Burger, D.C., and T.M. Austin, 1997. The simple scalar tool set, version 2.0. ACM SIGARCH Comput. Archit. News, 25: 13-25. DOI: 10.1145/268806.268810

Calder, D.G.B., and J. Elmer, 1996. Predictive sequential associative cache. Proceedings of the IEEE International Symposium on High Performance Computer Architecture, Feb. 3-7, IEEE Xplore Press, San Jose, CA , USA, pp: 244-253. DOI: 10.1109/HPCA.1996.501190

Chishti, Z., M.D. Powell and T.N. Vijaykumar, 2003. Distanceassociativity for high-performance energy-efficient nonuni form cache architectures. Proceedings of the 36th Annual ACM/IEEE Int. Symposium on Micro architecture, (MICRO 36), IEEE Computer Society Washington, DC, USA, pp: 55-66.

Davanam, N., and B.K. Lee, April 2010. Towards smaller-sized cache formobile pro cessors using shared set associativity. Proceeding of the 7th International Conference on Information Technology New Generation, April 12-14, IEEE Xplore Press, Las Vegas, NV, pp: 1-6. DOI: 10.1109/ITNG.2010.120

Hallnor, E.G. and S.K. Reinhardt, 2000. A fully associative software managed cache design. Proceedings of the 27th Annual International Symposium on Computer Architecture, (ISCA '00), ACM New York, NY, USA, pp: 107-116.DOI: 10.1145/339647.339660

Inoue, K., T. Ishihara and K. Murakami, 1999. Way-predictive set associative cache for high performance and low. Proceedings 1999 International Symposium of Low power Electronics and Design, (ISLPED '99), ACM New York, USA, pp: 272-275. DOI: 10.1145/313817.313948

Jouppi, N.P., 1990. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetchbuffers. Proceedings of the 17th Annual International Symposium on Computer Architecture, May 28-31, IEEE Xplore Press, Seattle, WA , USA, pp: 364-373. DOI: 10.1109/ISCA.1990.134547

Liu, C, A. Sivasubramaniam and M. Kandemir, 2004. Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs. Proceedings IEE, Software, Feb. 14-18, IEEE Xplore Press, USA, pp: 176-185. DOI: 10.1109/HPCA.2004.10017

Peir, J.K., Y. Lee, and W.W. Hsu, 1998. Capturing dynamic memory reference behavior with adaptive cache topology. Proceeding of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems, (ASPLOS-VIII), ACM New York, USA, pp: 240-250. DOI: 10.1145/291069.291053

Puzak, T.R., 1985. Analysis of cache replacement algorithms. University of Massachusetts.

Qureshi, M.K., D. Thompson and Y.N. Patt, 2005 The V Way Cache: Demand Based Associativity via Global Replacement. Proc. Int. Symp. Comput. Arc., 33: 176-185. DOI: 10.1145/1080695.1070015

Seznec, A., 1993. A case for two way skewed associative caches. Proc. Ann. Int. Sympos. Comput. Arch., 21: 169-178. DOI: 10.1145/173682.165152

ShivaKumar P. and N. Jouppi, 2001. CACTI 3.0: An Integrated Cache Timing Power and Area Model. Western Research Laboratory.

Wilton, S.E. and N. Jouppi, 1996. Cacti: An enhanced cache access and cycle time model. IEEE J. Solid State Circ., 31: 677-688. DOI: 10.1109/4.509850