Sindh Univ. Res. Jour. (Sci. Ser.) Vol.43 (1-A)43-50 (2011)

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Improving Memory Performance Using Cache Optimizations in Chip Multiprocessors

H. Khatoon and S. H. Mirza

Computer and Information Systems Engineering Department, NED University of Engineering and Technology, Karachi, Pakistan
hkhatoon@neduet.edu.pk, shahid_h_mirza@yahoo.com

cor
Rec

Abstract: The processor-memory bandwidth in current generation processors is the main bottleneck due to a number of processor cores sharing it through the same bus/ processor-memory interface. As a result, the on-chip memory hierarchy in multi core processors has assumed the role of one of the most important resources that should be managed efficiently to alleviate the above problem. Effective utilization of this resource is therefore an important aspect of memory hierarchy design of multi core processors. This is currently an important area of research with a large number of research publications that have proposed a number of techniques to solve the problem. These include novel techniques that were not used earlier either in single core processors or the conventional multiprocessors. This paper presents a survey of all such techniques proposed in recent publications. The major contribution of this paper is the assessment of effectiveness of some of the techniques that were implemented in recent chip multiprocessors. Cache optimization techniques that were identified for single core processors but have not been implemented in multi core processors are also examined to predict their effectiveness.

Keywords: On-chip cache hierarchy; cache optimizations; NUCA cache; prefetching; victim cache; chip multiprocessors.

INTRODUCTION

The on-chip memory and its effective utilization in multi core processors is the prime focus of this paper. With the increasing number of cores on a single chip, this strategy will determine the overall memory performance and hence the performance of the applications running on such systems. The workload running on these systems is a mix of multiple programs or multiple processes belonging to the same program. The overall performance would therefore not only be determined from the throughput of multiple programs but also from the performance of programs comprising of multiple parallel processes running on multiple cores of the same chip. The on-chip cache hierarchy needs to be designed with the best possible configuration and optimizations to serve the above purpose.

A large number of cache optimization techniques have been implemented in different types of computer architecture. Some of the techniques have been successfully implemented both in single core processors and in conventional multiprocessors with a resultant improvement in performance. A detailed account of seventeen of these well tested techniques is given in (Hennessy, *et al.,* 2006). Although most of the optimizations mentioned in (Hennessy, *et al.,* 2006) have been implemented in single core processors or in conventional multiprocessors, they have also found their usefulness in multi core processors, as these are thought to be the basic techniques that are expected to be successful in all types of architecture. Some of these techniques that are successfully implemented in multi core processors are: small and simple first-level cache, multi-level caches, non-blocking caches, and prefetching of code and data through hardware and software techniques. In most of the Chip Multiprocessors (CMPs), multi-level cache consists of two-levels with the second-level cache being the main focus of improvements. As a result, a number of new and innovative techniques have been developed for this level of cache. Although the performance of first-level cache is also important, the current design of first-level cache is considered to be almost an optimized one with very few innovations possible. But the design of second-level cache has a large room for improvement and is therefore the main focus of most of the research targeted towards its optimization.

In the next section, a brief account of all cache optimizations implemented in multi core processors taken from recent publications shall be presented. The optimizations implemented in single core processors and multiprocessors that have not been explored for multi core processors shall be presented in section 3.1. Those optimizations that have been explored for multi core processors but were found to be ineffective and in some cases have caused degradation in the overall performance will be presented in section 3.2. Section 4 gives possible research directions and concludes this paper.

MATERIALS AND METHODS

Optimizations Implemented Successfully

A number of cache optimization techniques that were implemented in single core processors were successfully implemented in multi core processors. Multi-level cache with the current structure of two-level has been implemented since the very first multi core processor visualized in **(Fig.1).** In this configuration, the first-level cache is private to each core and coherence is maintained between them with MESI or MOESI

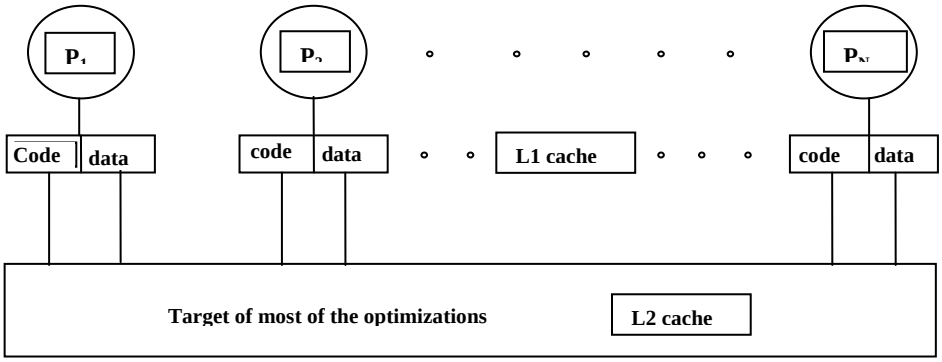

Fig.1 Block diagram representing on-chip memory hierarchy in CMPs

protocols (Villa, *et al.*, 2005). The second-level cache has been implemented with different design options in various architectures. In general, the second-level cache is shared among all cores with a number of optimizations to be discussed in this section.

One of the major innovations in the design of the second-level cache is NUCA (Non Uniform Cache Architecture) cache (Kim, *et al*., 2003). The reason for building NUCA organization is that the second-level cache is made much larger than the first-level to satisfy the design requirements of multi-level cache. The result is a slower access time with the increasing cache size. This problem is resolved by dividing the cache into banks. The context of a specific core is kept in a bank physically closer to it gaining improvement in the speed of access. A number of variants of NUCA have evolved over the last few years with many innovations implemented in current generation processors. Reactive NUCA (Hardavellas, *et al.*, 2010) performs optimal cache block placement by classifying blocks at run-time and placing data close to the core that uses them. D-NUCA (Kim, *et al..,* 2003) is another variant of NUCA architecture that dynamically places frequently accessed data in banks closer to the core and less frequently accessed blocks in farther banks. Thus data migration is allowed at run time. Other variants of NUCA are outlined in (Dybdahl, *et al*., 2007, Lin, *et al.,* 2008 and Zhang *et al*., 2005). An issue which is considered to be important for on-chip cache hierarchy is whether there is a need for it to follow the property of inclusion (Hsu *et al.,* 2005). In order to satisfy this property, the blocks present in the first-level cache should be present at all other levels. Many researchers have pointed out that enforcing this property results in wastage of cache space, suggesting that a better cache utilization is achieved if this property is not made mandatory, especially in large scale multi core processors. Although this is still being debated, some earlier results (Jouppi *et al.,* 1994) show improvement with the exclusion property in single-core, two-level caches. It is expected that similar results can be achieved in multi core processors, because the on chip memory is not wasted by replicating at different levels. The increase in the total available memory through the property of exclusion may ultimately result in better performance.

Cache parameters such as block size, associativity, cache size, write policy and coherence protocols directly affect the performance of the on-chip memory hierarchy. Since different applications have different demands with respect to each of the above parameters, (Tao *et al.* 2008) have proposed reconfigurable cache architecture with the parameters being transparent to application programmers who may

set the values according to the requirements of their applications. Although this is difficult to achieve practically requiring the programmer to be architecture aware, the research groups working on such reconfigurable architectures are optimistic and have predicted positive results.

A recent publication by (Hammoud *et al.* 2010) outlines a novel technique called DPAP (Dynamic Pressure-aware Associative Placement) for cache blocks. This scheme decouples the mapping of memory blocks to cache from their physical addresses and places them according to their pressure or frequency of use. The pressure is recorded at the group granularity level which is later used to place an incoming block in a cache block that belongs to a group that has the minimum pressure.

A question associated with second-level cache and NUCA organization is whether the cache should be shared or private with respect to each core. This issue has been explored by a number of researchers with conflicting results supporting the respective configurations. Hsu *et al.* (2005) has pointed out that for small cache designs, shared cache gives a better performance but for large cache designs, the advantage is not so significant. On the contrary, the study conducted by (Tao *et al.* 2008) using various benchmarks show that shared L2 gives better performance for majority of the applications. (Haakon *et al.*, 2007) have proposed to implement an adaptive shared/private cache partitioning where the amount of shared space among cores is controlled dynamically. This shows that application dependent features are important to decide about shared/private cache configuration. The control for this feature remains in hardware but is made application-aware by coupling it with the counters and registers meant for recording misses and tags of evicted blocks. Most of the current generation processors are equipped with PMUs (Processor Monitoring Units) that provide the above measurements. R-NUCA (Hardavwllas, *et al.*, 2010), a variant of NUCA, deals with the above issue in a more formal way. It alleviates the problem of both private and shared cache designs with significant power savings by classifying blocks on the basis of access patterns at run time and places it near the requesting cores. Address mapping is managed through a simple lookup that saves time and power.

Another improvement for multi core processors is more architectural support for non-blocking cache. To allow maximum miss level parallelism, the Miss Handling Architecture (MHA) requires a number of additional components as proposed in (Jahre, *et al.*, 2007). Since a higher miss-level parallelism may add to congestion in the access path affecting the overall performance, (Jahre *et al.,* 2007) have proposed that a balance is required between the miss-level parallelism and congestion in both on-chip and off-chip interconnects.

In order to improve cache space utilization, a number of techniques have been suggested. One such design option is bypassing of cache accesses that are transient referred to as block bypassing (Dybdahl, *et al.,* 2007). Block bypassing is proposed for second-level cache and it requires the monitoring of reuse behavior of cache blocks. Blocks that are classified as bypassed are kept only in the first-level cache. Load requests for bypassed blocks are used to generate an early miss request, improving the miss penalty.

A strategy that is similar to reconfigurable cache architecture with application/ programmer transparent parameters is to have a software controlled cache. This scheme allows the operating system or the application developer to become the software-based cache controller and adapt the cache parameters according to the run-time conditions. One such technique is demonstrated in (Mori, *et al*., 2009) and is named Cache-Core architecture. Using heterogeneous multi core processor with local memory that can be configured as software controlled cache, the core that is not allocated a thread is made to work as a shared L2 cache managed through software.

Prefetching of instructions and data has been a useful strategy for improving the performance of every level of memory hierarchy. A number of prefecthing strategies for the cache hierarchy are explored in (Tao, *et al*., 2008 and Ebrahimi, *et al*., 2009). After analyzing five prefetching schemes namely always prefetch, on-miss prefetch, tagged prefetch, stride-based prefetch and delta prefetch, it is inferred through supporting data that tagged prefetching is the most effective to reduce the miss rate (Tao, *et al*., 2008). Whether this also gives the best overall performance is not clear and needs to be investigated. As pointed out in (Ebrahimi, *et al*., 2009), prefetching may interfere with demand fetches. This problem is more acute in CMPs because of multiple cores generating prefetch requests. This may lead to performance degradation that needs to be controlled. (Ebrahimi, *et al.* 2009) have proposed a solution based on hierarchy of prefetch controls that combine local and global prefetcher interference to balance the benefits of prefetching with that of the overall system performance. Address correlated prefetching are effective for irregular access patterns that are repetitive. Earlier, this scheme was not practical for implementation because it required a large amount of metadata that could not be stored on processor. (Wenisch *et al.* 2010) have suggested an

innovative technique to make the off-chip storage of metadata practical, thus allowing the effective use of the scheme in current generation processors.

Because of the growing power concerns in multi core processors, in-order processors are a preferred architecture, but this results in performance degradation because of stalls due to various dependences. This can be overcome by using iCFP (in-order continual flow pipeline), a technique proposed by (Hilton *et al.* 2010). This technique uses the runahead execution mode, a mode of execution entered by an in-order processor when it encounters a miss, increasing the Miss Level Parallelism (MLP). All miss-dependent instructions are saved in a slice buffer whereas the miss-independent instructions are executed and retired speculatively. When the miss returns, it executes the instructions saved in the slice buffer. (Hilton, *et al.*, 2010) have shown that iCFP improves performance of in-order processors with the additional advantage of low power consumption.

Use of adaptive shared/private NUCA cache partitioning to improve the overall miss rate is given by (Dybdahl, *et al.*, 2007). Second-level cache as NUCA is generally organized as per core partition. If a core runs out of cache space, the evicted block is relocated to the partition of another core, thus utilizing some cache space as a shared one. An uncontrolled allocation may result in performance degradation due to pollution. An adaptive scheme to dynamically control the shared space attempts to maximize the overall performance. This is done by protecting the most recently used data in the last-level cache. An improvement in suggested in (Qureshi, 2009) through adaptive spill/receive policy, which is a dynamic scheme used not only to reduce but also to control pollution by defining spiller and receiver partitions. Use of Miss Rate Curves (MRC) for on-line optimization is proposed in (Tam, *et al*., 2009) to support the decision making process for the above schemes, but obtaining online MRC has its overhead which makes it un-practical. (Tam *et al.* 2009) have proposed to obtain on-line efficient MRC termed as Rapid MRC through the use of PMUs (Processor Monitoring Units) available in all current generation processors. The paper approximates L2 MRC with low overhead and compares RapidMRC of 30 standard application benchmarks with that of real MRCs. Stack algorithm is a common method to generate MRC which maintains an LRU stack for recent memory accesses. The stack distance of every memory access is calculated to speculate for the next access to be a hit or a miss. A histogram *Hist (dist)* shows all memory accesses with a stack distance of *dist*. The number of misses for a memory size *size*, *Miss (size)* is calculated by the following expression

$$Miss(size) = \sum_{dist=size+1}^{\infty} Hist(dist)$$

This then generates an MRC that is normalized over a fixed probing period using MPKI (Number of Misses Per Kilo Instructions), given by the following expression

$$MPKI(size) = 1000 \times \frac{Miss(size)}{CPUInstructions}$$

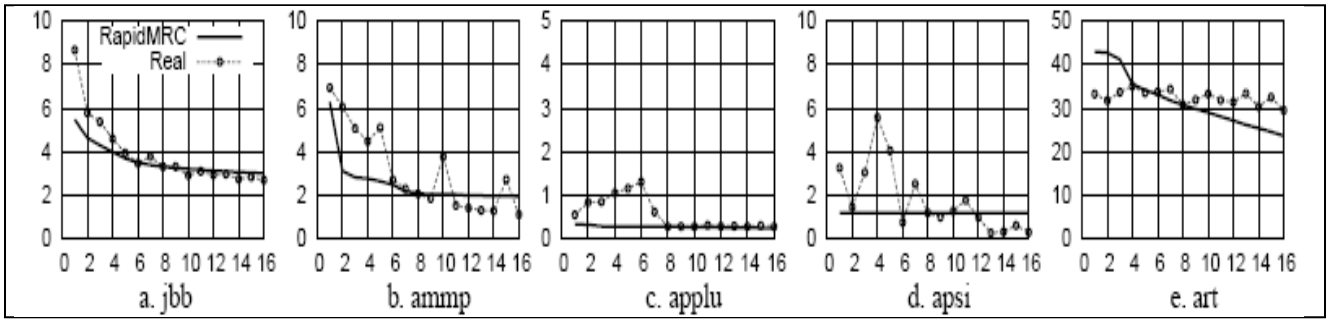

Fig. 2. Online Rapid MRC vs Offline Real MRCs. X-axis gives the allocated L2 partition in terms of the number of colors and Y-axis gives the resulting L2 cache miss rate (MPKI) (taken from Tam *et. al.* 2009)

where *CPUInstructions* is the length of the probing period. **(Fig.2)** shows the comparison of online RapidMRC with that of real (Offline) MRC (taken from (Tam, *et al*, 2009)). For most of the applications, online MRCs (RapidMRC) are close to real MRCs obtained offline.

Other innovations include addition of hardware to enhance a previously implemented optimization. The LRU block replacement policy gives a poor performance if the working set is larger than L2. This is because a larger number of less reused blocks occupy cache space. Bypassing these less reused blocks with the

help of less reused filter improves the overall performance as outlined in (Xiang, *et al.*, 2009). Such blocks are identified by a reuse frequency predictor. Adding a filter buffer to this optimization to temporarily save such blocks avoids more misses. Use of filters also reduces the overall coherence traffic that increases with the increase in the number of cores, causing congestion at various resources. Design and implementation of Blue Gene/P snoop filter, presented in (Salapura, *et al.*, 2008) attempts to filter snoop requests that are not in cache. This is augmented with streaming registers with a marked improvement in the overall performance.

Cooperative Caching (Chang, *et al.,* 2006) combines the strengths of both private and shared caching. It is a framework in which data used locally are kept in private caches and the data that are shared globally are kept in shared cache. Modified policies were simulated and experiments were conducted by Chang and Sohi which is summarized in (Chang, *et al.*, 2006). The results thus obtained were encouraging in terms of off-chip miss rate and local cache hit rate.

To reduce off-chip communication latency in case of a miss in cache, a 3D-stacked MRAM is proposed in (Sun, *et al*., 2009). The implementation requires addition of extra hardware at the L2 cache to memory interface.

RESULTS AND DISCUSSION

Proposed Cache Optimizations

A number of cache optimization techniques were successfully implemented in single core processors or single core multiprocessors but have not yet been tried in multi core processors. Some of these techniques are discussed in this section with a prediction of their effectiveness in multi core processors.

Trace cache (Hennessy, *et al.*, 2006) allows to cache dynamic traces of executed instructions including taken branches. This cache requires a branch predictor to dynamically decide the execution path of programs. A cache block is utilized more efficiently in a trace cache but the overall cache utilization is not efficient because the same instructions may be present in a number of blocks. Due to some problems, trace cache has been implemented in only a selected number of single core processors. Because it is less efficient in terms of power and area utilization, it may not be an effective mechanism for multi core processors. Moreover, the control of trace cache is complex that may add to the complexity of the overall cache control system.

Victim cache is an optimization technique where a small, fully associative cache is placed between the cache and its refill path. The victim cache is filled with the blocks evicted from cache due to block replacement. A miss in the cache is first checked in the victim cache before the request is sent to the main memory. In a multi core processor, this technique has not been implemented in the same form but a similar technique is used in second-level NUCA caches that use shared/private configuration. In NUCA cache with partitions private to each core, when a core runs out of cache space and a block is evicted due to block replacement, instead of discarding this block totally, it is stored in the partition of another core in anticipation that it may be needed again (Dybdahl, *et al.*, 2007). Although this scheme is similar to victim cache but it is implemented at the cost of another core's cache space. Some controlled schemes have also been suggested in (Tam, *et al.,* 2009), but all these schemes use the partition of another core. A better implementation would be a victim cache which is a small, one to four entry fully associative cache that helps in significant improvement in miss rate. A dedicated victim cache per core would also avoid pollution of cache partition belonging to another core. Since the victim cache contains recently evicted blocks, it shall reduce both the miss penalty and miss rate of the cache hierarchy.

One major issue which has not been investigated is whether the cache design should always conform to the shared memory paradigm as this requires policies for coherence and consistency with an overhead that affects the performance of on-chip memory hierarchy. If the message passing paradigm is used for inter-core communication and sharing of data, all caches will remain private to each core without causing interference due to coherence and consistency traffic. In shared memory paradigm, a large wait time is incurred in synchronization for access to shared variables. Use of message passing paradigm would remove the overhead of synchronization wait time.

Various compiler-based optimization techniques are suggested by a number of researchers to improve the overall cache utilization (Chen, *et al.*, 2008). These techniques are effective for efficient utilization of cache space and may be equally effective for CMPs. All future compilers designed for multi core processors should take into account the existence of parallelism at the chip level. Other features that need to be considered is that the inter core communication and synchronization overhead is much smaller than what is observed between processors of conventional multiprocessors. This advantage can be exploited in the design of compilers and algorithms where inter-core communication is more efficient than communication among multiprocessors in SMPs.

Search in cache is carried out after translation from virtual to physical address. The translation time adds an overhead to all cache accesses which adds to the

critical memory access time. A solution implemented in some processors is to implement virtual address cache. The search in these caches is done in parallel with the address translation process. But the solution was found to be complex and was not as effective due to high overheads for larger caches. A brief account of the problems of virtually addressed caches is given in (Hennessy, *et al.*, 2006). A possible solution to some of the problems is to use small first level cache. Observing the design of cache hierarchy of multi core processors, the first-level cache is relatively smaller in CMPs than in single core processors. This technique can therefore be considered for implementation without the problems seen in single core processors.

Use of write buffers in both write-through and write-back cache improves the overall memory access time. The number and size of write buffer is an important parameter that determines the effectiveness of this technique. An optimization to best utilize the write buffers is to use write merging (Hennessy, *et al*., 2006). This technique allows fast write for multiple writes and better performance even with a smaller number of write buffers. The same technique can be applied to CMPs for effective utilization of write buffers.

Most of the first-level cache is two-way set-associative. In order to improve the hit time of the cache, way prediction can be used that gives the hit time of a direct-mapped cache and the miss rate of a set associative cache (Hennessy, *et al.*, 2006). Extra bits are added to each cache block to predict the way for next cache access. This scheme works well in single core processors and it can be effective and beneficial for multi core processors too. There is an additional requirement of block predictor that predicts the next block in the set. Only a single tag is compared and the multiplexer is set earlier to select the predicted block. With the help of accurate predictors, way prediction can be effective for CMPs.

As in single core processors, pipelined access for first-level cache is an effective way of reducing the overall cache access time in multi core processors. This method increases the hit time of individual accesses to the cache but the overall effective access time is reduced. The penalty increases for mispredicted branches, a problem that can be overcome with the use of efficient branch predictors. This technique is expected to optimize the average cache access time of individual cores with a number of accesses pipelined to overlap the access time. DSBC (Dynamic Set Balancing Cache) tries to make use of an under-utilized set by associating it with another set in the same cache (Rolan, *et al.,* 2009). Although suggested for single-core processors, this scheme can be extended for multi core processors with a few modifications.

All the cache optimization techniques that were discussed in this section need to be investigated by conducting experiments. Results inferred from such investigation can then be applied for future generation multi core processors.

Ineffective Cache Optimizations

The optimization techniques presented in Section 3.1 needs to be implemented to determine their effectiveness. A few optimizations were tested for multi core processors and were found to be ineffective. As more optimizations are tested, one may find more such techniques as not being useful for multi core processors. The following paragraphs give a brief account of the tested techniques that were not successful in CMPs.

Cache affinity is a policy decision taken by the operating system to schedule processes on specific cores. The decision is based on the behavior of a process that has its context in a cache and is expected to reuse the contents as a result of temporal locality. After a context switch, when a process is rescheduled, it is allocated to the same processor, assuming that its context may still be present in the cache, reducing the compulsory or cold start misses. This scheme has improved the performance in conventional multiprocessors (SMPs). On investigation of this scheme in multi core processors and summarized in (Kazempour, *et al.*, 2008), it was observed that the performance improvement in multi core uniprocessors (CMPs) is not significant, but the performance is good in case of multi core multiprocessors (SMPs based on CMPs).

Trace cache may not be an effective technique for multi core processors because it wastes memory space due to repetition of instructions in more than one block because it contains dynamic sequence of instructions. It also has relatively higher power consumption. Although it is a promising technique in theoretical terms, it may not work for large scale multi core processors.

Since a large number of optimizations that have been discussed in the previous sub-section have not been tested, this section contains very few instances. As a part of our PhD project, we plan to test most of the techniques mentioned in Section 3.1 and report the results in subsequent publications.

CONCLUSION AND FUTURE DIRECTIONS

This paper forms part of the guideline for future work for researchers interested in optimization of memory hierarchy for scalable multi core processors, as it presents a survey of all such techniques proposed in recent publications. The techniques are also presented

along with the comments about their effectiveness. A summary of all the optimization techniques discussed in this paper is presented in **(Table 1)**.

Table I. Summary of All Cache Optimization Techniques in CMPs

Optimizations (Level 1 cache)	
Trace Cache	Not explored
Virtually Addressed Cache	Not explored
Way Prediction	Not explored
Pipelined Cache Access	Not explored
Optimizations (Level 2 Cache)	
NUCA	Tested and implemented
D-NUCA	Tested and implemented
R-NUCA	Proposed and Explored
Adaptive Shared/Private Cache	Proposed and Tested
Adaptive Spill/Receive Policy	Proposed and Explored
Use of MRC to support the abov	Proposed and tested
Block bypassing	Proposed and tested
Cache-Core Architecture	Proposed and tested
Cooperative Caching	Proposed
3D Stacked MRAM	Proposed
Victim Cache	Not explored
Write Buffers with Write Merging	Not explored
Dynamic Set Balancing Cache (DSBC)	Not explored
Optimizations (Both L1 and L 2 Caches)	
Use Property of Exclusion	Proposed
Reconfigurable Cache	Proposed and explored
Dynamic Pressure Aware Placement(DPAP	Proposed and explored
Non-Blocking Cache – MHA	Proposed and explored
Prefetching of Instructions and data	
Tagged Prefetch	Proposed and tested
Hierarchy of Prefetch Controls	Proposed and tested
Address-Correlated Prefetching	Proposed and tested
iCFP (In-order Control Flow Pipeline)	Proposed and explored
Compiler-based Optimizations	Not explored
Cache Affinity	Explored but ineffective

The effect of the mechanisms and policies of operating system on the memory hierarchy, especially the on-chip cache hierarchy is another direction of research that can be explored. High coherence traffic gives rise to congestion at the first level cache. Directory-based coherence protocols may reduce the overall coherence traffic but this comes with the cost of maintaining the directory and keeping it updated. These and other research directions shall be explored in future research.

ACKNOWLEDGEMENT

This is the extended version of our own paper presented and published in "International Conference on Computer and Emerging Technologies" (ICCET 2011) held on 22-23 April 2011 at Shah Abdul Latif University, Khairpur, Sindh, Pakistan.

REFERENCES

Papers in Journals/Proceedings/Symposia

Books H. (2006) *Computer Architecture–A Quantitative Approach,* 4th Edition, Morgan Kaufmann Publishers Euro-Par), 2008, 151-161.

Chang A. (2006) *Cooperative Caching for Chip Multiprocessors*, Proceedings of the 33rd Annual International Symposium on Computer Architecture, 264-276.

Chen A. (2008) *Code Restructuring for Improving Cache Performance in MPSoCs*, IEEE Transactions on Parallel and Distributed Systems, Vol. 19, No. (**9**): 1201-1214.

Dybdahl, H., and P. Stenström, (2007) *An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors*, Proceedings of the IEEE 13th International Symposium on High Performance Computer Architecture, 02-12.

Dybdahl H., and P. Stenström, (2006) *Enhancing Last-Level Cache Performance by Block Bypassing and Early Miss Determination,* Asia-Pacific Computer Systems Architecture Conference (ACSAC), LNCS(**4186**):52-66.

Ebrahimi, E., O. Mutlu, C. J. Lee, and Y.N. Patt, (2009) *Coordinated Control of Multiple Prefetchers in Multi-Core Systems,* Proceedings of the 42nd International Symposium on Micro-architecture (MICRO), 327-336.

Hammoud, M., S. Cho, and R.G. Melhem, (2010) *A Dynamic Pressure-Aware Associative Placement Strategy for Large Scale Chip Multiprocessors*, IEEE Computer Architecture Letters, Vol. 9, No.(**1**): 29-32.

Hardavellas, N., M. Ferdman, B. Falsafi, and A. Ailamaki, (2010) *Near-Optimal Cache Block Placement with Reactive Non-uniform Cache Architecture,* IEEE Micro, Vol. (**1**): 20-28.

Hilton, A., S. Nagarakatte, and A. Roth, (2010) *iCFP: Tolerating All-Level Cache Miss in In-order Processor*, IEEE Micro, Vol. (**1**): 12-19.

Hsu, L., R. Iyer, S. Makineni, S. Reinhardt, and D. Newell, (2005) *Exploring the Cache Design Space for Large Scale CMPs*, ACM SIGARCH Computer Architecture News, Vol. 33, Issue (**4**): 24-33.

Jahre, M., and L. Natvig, (2007) *Performance Effects of a Cache Miss Handling Architecture in a Multi core Processor,* Norwegian International Conference.

Jouppi M. and Wilton, (1994) *Tradeoffs in Two-Level On-Chip Caching,* ACM SIGARCH Computer

Architecture News: Proceedings of 21st Annual International Conference on Computer Architecture (ICSA), Vol. 22, Issue (**2**): 34-45.

Kazempour, V., A. Fedorova, and P. Alagheband, (2008) *Performance Implications of Cache Affinity on Multicore Processors,* Proceedings of the 14th International EuroPar Conference on Parallel Processing, 151-161.

Kim, C., D. Burger, and S. W. Keckler, (2003) *NUCA: A Non-Uniform Cache Access Architecture for Wire-Delay Dominated On-Chip Caches,* IEEE Micro, Vol. (**6**): 99-107

Lin J. (2008) *Gaining Insights into Multi core Cache Partitioning: Bridging the Gap between Simulation and Real Systems*, IEEE, 367-378.

Mori S., (2009) *The Cache-Core Architecture to Enhance the Memory Performance on Multi-Core Processors*, 2009 International Conference on Parallel and Distributed Computing, Applications and Technologies, Published by IEEE Computer Society, 445-450.

Qureshi, M., (2009) *Adaptive Spill-Receive for Robust High-Performance Caching in CMPs*, Proceedings of the IEEE 15th International Conference on High Performance Computer Architecture (HPCA), 45-54.

Rolán, D., B. Fraguela, and R. Doallo, (2009) *Adaptive Line Placement with the Set balancing Cache*, 42nd Annual IEEE/ACM International Symposium on Micro-architecture (MICRO-42), 529-540.

Salapura, B. G. (2008) *Design and Implementation of Blue Gene/P Snoop Filter,* Proceedings of the 14th International Symposium on High Performance Computer Architecture (HPCA), 05-14.

Sun, G., X. Dong, Y. Xie, and J. Li, (2009) *A Novel Architecture of the 3D Stacked MRAM L2 Cache for CMPs,* IEEE 15th International Symposium on High Performance Computer Architecture (HPCA), 239-249.

Tam, D. K., R. Azimi, L.B. Soares, and M. Stumm, (2009) *RapidMRC: Approximating L2 Miss Rate Curves on Commodity Systems for Online Optimizations*, Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 121-132.

Tao, J., M. Kunze, and W. Karl, (2008) *Evaluating the Cache Architecture of Multi core Processors*, 16th Euromicro Conference on Parallel, Distributed and Network-based Processing, 12-19

Villa, F. J., M. E. Acacio,, and J. M. Garc´ıa, (2005) *Memory Subsystem Characterization in a 16-Core Snoop-based Chip-Multiprocessor,* Proc of the 1st International Conference on High Performance Computing and Communication (HPCC), 213-222.

Wenisch, T .F., M. Ferdman, A, Ailamaki, B. Falsafi, and A. Moshovos, (2010) *Making Address Correlated Prefetching Practical,* IEEE Micro, Vol. (**1**): 50-59.

Xiang, L., T. Chen, Q. Shi, and W. Hu, (2009) *Less Reused Filter: Improving L2 Cache Performance via Filtering Less Reused Lines*, Proceedings of the 23rd International Conference on Supercomputing (ICS), 68-79

Zhang S. (2005) *Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors,* 32nd International Symposium on Computer Architecture (ICSA-32), 336-345.