Effective Utilisation of LLCs by Managing Associativity, Placement and Mapping

Thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

by

Shirshendu Das

Under the supervision of

Dr. Hemangee K. Kapoor

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

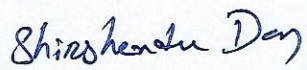
Guwahati - 781039, Assam, India

January 2016

Declaration of Authorship

I, Shirshendu Das, confirm that:

- The work contained in this thesis is original and has been done by myself and the general supervision of my supervisor.
- This work has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (data, theoretical analysis, results) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the reference.
- Whenever I have quoted from the work of others, the source is always given.

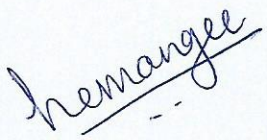


Shirshendu Das
Research Scholar
Department of CSE
Indian Institute of Technology Guwahati,
Guwahati, INDIA 781039,
shirshendu@iitg.ernet.in

Date: 25th January, 2016
Place: IIT Guwahati

Certificate

This is to certify that the thesis entitled **"Effective Utilisation of LLCs by Managing Associativity, Placement and Mapping"** being submitted by **Mr. Shirshendu Das** to the department of Computer science and Engineering, Indian Institute of Technology Guwahati, is a record of bona fide research work under my supervision and is worthy of consideration for the award of the degree of Doctor of Philosophy of the Institute.



Dr. Hemangee K. Kapoor
Associate Professor
Department of CSE
IIT Guwahati,
Guwahati, INDIA 781039,
hemangee@iitg.ernet.in

Date: 25th January, 2016
Place: IIT Guwahati

Dedicated to

my parents, my supervisor

and
my all friends at IIT Guwahati.

Acknowledgements

It is a real pleasure for me to thank all the people who have supported me during my PhD. First and foremost I thank my supervisor, Dr. Hemangee Kapoor for her extensive help and guidance throughout my PhD. My association with her has lasted for 6 years from my M.Tech days at IIT Guwahati. It is a wonderful part of my life to be associated with her through these years. I profoundly thank her for helping me become what I am today. Thank you ma'am for your support throughout my journey at IIT Guwahati. You have corrected my silly mistakes again and again and always with a smile. There are countless reasons to thank you.

I am grateful to my doctoral committee members: Dr. Santosh Biswas, Dr. Aryabhatta Sahu and Prof. Roy Paily Palathinkal for providing constructive suggestions related to my work. I sincerely feel that their suggestions and comments have helped me in shaping up my final thesis. My sincere thanks to Dr. Santosh Biswas and Dr. Kalpesh Kapoor for various academic/non-academic suggestions. Discussion with them is always fun. I would like to thank Prof. Diganta Goswami, the Head of the Department of Computer Science and Engineering and other faculty members for their support and help. I would also like to acknowledge the efforts devoted by all the teachers, right from my lower primary school till my PhD.

Life during a PhD takes many twists and turns. I am fortunate to have the company of Mayank, Nilkanta, Shashi, Shilpa, Basant, Pravati, Mamata, Srinivasa, Dipika, Satish and Sibaji. You all are very special for me. Life would not have been so picnic-full here without you all. I feel like writing a few lines about each of you, but I can only say that each one of you has a special place in my heart. Thank you Debanjan, Shuvendu, Khushboo, Awnish, Mrityunjay, Manojit, Anirban and Achyutmani. During my PhD, I had the opportunity to work with Shounak Chakraborty, Sukarn Agarwal, Prateek Halwe, Nagaraju Polavarapu and Surajit Das. I feel privileged to have exchanged very fruitful discussions with all of them. There are countless reasons to give a special thanks to Mayank Agarwal. This guy makes my life so effortless in IITG. I am extremely lucky that I have a friend like him. Mayank and Dipika are the two people whose support always lifted my spirits and added enthusiasm.

I would like to thank Mr. Nanu Alan Kachari, Mr. Bhriguraj Borah, Mr. Raktajit Pathak and all other departmental staffs for helping me in many occasions. My thanks to all security guards, janitors, housekeeping staffs, hostel and canteen staffs for making our life so easy in IITG. Many thanks to all of you for doing all the thankless jobs for us.

I am thankful to TATA Consultancy Services (TCS) for supporting me financially in my PhD via their TCS RSP program. The travel grants under TCS RSP program had given me the opportunity to visit many national and international conferences. This not only helped me broaden my vision but also got me new friends. Thank you TCS.

My special thanks to the assistant registrar of academic affairs Mr. Sunil Kumar Barua. Despite his extremely busy schedule, he has always been very caring and has always listened to our concerns genuinely. Many thanks to Mr. Rijumoni Dutta from academic affairs and Mr. Sunil Sarma from finance section for handling our TCS related stuff. I would like to thank the whole administration of IIT Guwahati for their exceptional work in maintaining this institute. I have never met anyone who did not fell in love with the beauty of this campus. The badminton facilities at IIT Guwahati are phenomenal. I have fallen in love with badminton here and now I can't imagine my life without badminton.

Finally, I would like to thank the most important persons of my life – my parents. Thank you for providing me all kinds of support starting from my childhood. They have always respected my thoughts and decisions. My elder brother and my sister-in-law are also my biggest strength of life. Being the youngest in my family, I feel I am extremely lucky to be brought up under the shadow of such beautiful people.

Abstract

Tiled based CMP (TCMP) has become the essential next generation scalable multicore architecture. The cores in TCMP commonly share a large sized Last Level Cache (LLC). NUCA is used in LLC to divide it into multiple banks such that each bank can be accessed independently. Static NUCA (SNUCA) has a fixed address mapping policy whereas dynamic NUCA (DNUCA) allows blocks to relocate nearer to the processing cores at runtime.

It has been observed that the LLC of the current TCMP architectures is not utilised properly. Better cache utilisation will reduce the number of misses in the cache and hence can improve performance. The utilisation issue of LLC can be divided into two categories: (a) local utilisation and (b) global utilisation. The memory accesses within a bank are not distributed uniformly among the sets. Some sets are used heavily while some others remain idle. Such utilisation issue is termed as the local utilisation issue. It has also been observed that the banks of the LLC are not carrying equal loads during the execution. Some banks are loaded heavily while some other banks remain almost unused. Better load distribution among the banks may improve the utilisation factor of the cache. Such inter-bank utilisation issue is termed as the global utilisation issue.

In this work we propose architectures to increase both the local and global utilisations of LLC for TCMP. Our first three proposals are for improving the local utilisation. We do this by allowing the heavily used set to use the idle ways of lightly used sets. Hence the associativity of each bank is managed dynamically. The three architectures we propose have different performance benefits and hardware requirements. To improve global utilisation we propose two DNUCA based TCMP architectures capable of distributing loads among multiple banks.

Experimental evaluation using full-system simulations has validated our claim of performance enhancement. The improvements in local utilisation give better performance in the range of 6.3-13.5% and those in global utilisation give 6.1-13% performance enhancement.

Contents

List of Figures

List of Tables

Abbreviations

CMP	Chip MultiProcessors
LLC	Last Level Cache
CPI	Cycle Per Instructions
LRU	Least Recently Used
MRU	Most Recently Used
AMAT	Average Memory Access Time
MPKI	Miss Per Thousand Instructions
IPC	Instructions Per Cycles
NUCA	Non Uniform Cache Access
NoC	Network on Chip
SNUCA	Static NUCA
DNUCA	Dynamic NUCA
TCMP	Tiled based CMP
DAM	Dynamic Associativity Management
T-SNUCA	SNUCA based TCMP
CMP-VR	Proposed DAM based architecture
CMP-SVR	Proposed DAM based architecture
FS-DAM	Proposed DAM based architecture
T-DNUCA	Proposed DNUCA Based TCMP
TLD-NUCA	Proposed DNUCA based TCMP
TGS	Tag array of CMP-VR
SA-TGS	Tag array of CMP-SVR
FS-TGS	Tag array of FS-DAM
H-Set	Heavily used set

L-Set	Lightly used set
RT	Reserve storage
NT	Normal storage
TLD	Tag Lookup Directory
TDR	Tag Data Ratio
EDP	Energy Delay Product (Energy × CPI)
RGP	Regrouping Period

Chapter 1

Introduction

As mentioned in Moore's law [4] the number of transistors on the chip increases with every new generation of the microprocessor. The main motive of adding more transistors is to provide better pipelining, ALU support and many other features. All these improvements enable faster performing processors due to better clock frequencies. However the frequency and computing power of a uniprocessor system has already reached its maximum limit few years ago. The amount of pipelining possible from a typical instruction set is hard to improve any more. Now it is not easy to create more powerful and/or faster processors than the current ones. As the CPU speeds rose to the 3-4 Ghz range the amount of power required to go faster started to become prohibitive. That is why the major CPU manufacturers switched strategy and started designing multi-core systems.

1.1 Chip-multiprocessors (CMPs)

Chip Multiprocessors (CMPs) are multi-core systems containing multiple processing units on the same die [5]. For various reasons CMPs are now the only way to design high performance systems. They avoid the above mentioned performance issues of single core systems by adding multiple, relatively simpler processor cores instead of just one huge core. The cores in the CMP can be either simple or

highly complex super-scalar processors. But the advantage with CMPs is that the performance can be improved further by adding more copies of the selected core in its every new generation.

The upcoming generations heavy workload demands higher throughput and parallelism. It is more likely that the demands of such workloads will increase in near future. Since a single core is unable to meet their needs, CMPs are the better candidates which can increase throughput by distributing multiple threads of execution among different cores. Moreover CMP has very low inter processor communication latency as compared to other conventional multi-chip processors thereby making more number of workloads suitable for parallel executions. However the limited parallelism in some workloads limits the performance of CMPs but such scenarios are quite lesser in number in present day situation.

1.2 Cache Memory

Cache memory is an essential part of computer architecture to improve performance. It is used to reduce the average time to access data from the main memory. Cache is a smaller and faster storage which stores some frequently used blocks of main memory. In case of single core systems the cache is integrated with the CPU chip. For any memory access, the processor first requests the cache. If the block is present in the cache then it directly sends the block to the processor otherwise it has to be fetched from the main memory. Fetching block from main memory takes much longer time than accessing it from the cache. The principle of temporal and spatial locality guarantees that the block once fetched from the main memory is more likely to be used multiple times before getting evicting from the cache [6]. Cache memory improves the performance of CPU by reducing the average Memory Access Time (AMAT). Lesser memory access implies lesser Cycles Per Instruction (CPI) or higher Instructions Per Cycle (IPC).

Most of the computer systems have multiple level of caches (L1, L2, etc). The L1 caches are divided into instruction cache and data cache while the other levels

of caches have no such divisions. Most of the works mentioned in the thesis are based on set associative cache which is considered as the best cache architecture as compared to direct mapped and fully associative cache [6]. The different data management policies of set associative cache are discussed in Chapter 2.

It has been observed that the performance of CPU heavily depends on the performance of its cache memory. Better hit rate in cache memory reduces the number of main memory accesses and hence improves CPI. The detail description about the cache architectures and its performance issues are given in [6]. Improving the performance of cache memory is a major area of research for many years [5, 7, 8]. In this thesis we only concentrate on the Last Level Cache (LLC) of CMPs. The main aim of the thesis is to improve the performance of LLC in CMPs.

1.3 Cache Memory Architectures of CMP

In CMP, the cores can have many levels of caches. In most of the cases the upper levels of cache are private to each core, i.e. each core has its own upper level caches. The Last Level of Cache (LLC) can be either private or shared. The performance of CMP largely depends on the performance of the LLC [5]. The importance of better LLC performance in CMP can be understood from the many recent research works [9, 7, 10, 11, 12, 13, 14, 15, 16]. In our entire work we considered two level cache hierarchy (L2 as LLC) though it is possible to extend it for more levels.

CMP cache architectures are mainly of two types [5]: (a) CMP with a private LLC and (b) CMP with a shared LLC. Both types of architectures have a separate L1 cache (considering two-level cache) with each core, but they differ in the physical placement of the LLC (L2). Private L2 caches are relatively small and placed physically very near to the core, thus having faster cache accesses. However as the cache size is small, it causes several capacity misses. Multiple copies of the same data block may be present in separate L2 caches, leading to maintain L2 level coherence. On the other hand, shared L2 is comparatively larger and only a single copy of each data block can be stored in it. All the requesting cores share

the same data block and the cache storage can be dynamically allocated to the cores depending on their workloads. Shared LLC increases hit time due to a much larger cache size compared to the private LLC. Based on the current large size applications, the shared LLC is a better design choice. However the performance of such a shared LLC needs to be improved in order to incorporate the growing demand for cache size [1]. In this thesis we consider only shared LLC and in the rest of the report the term LLC means shared LLC of CMP, unless otherwise mentioned.

The rest of the thesis assumes LLC as shared L2 cache of CMP.

1.3.1 Non Uniform Cache Access (NUCA)

Initially most of the cache structures were designed to have uniform access time regardless of the block address being accessed. For such Uniform Cache Access (UCA) architectures, the access time became a significant bottleneck for larger size caches [5, 17]. An alternative solution is to divide the large cache into multiple banks such that each bank can be accessed independently. The banks are connected over an interconnect like network on chip (NoC) [5]; a core can access its closer bank much faster than the farther banks because of lesser on chip communication distance. This kind of design is called Non Uniform Cache Access (NUCA) [17] and is the most promising LLC design in recent years.

NUCA is divided into two categories: static NUCA (SNUCA) and dynamic NUCA (DNUCA). In SNUCA a block always maps to a fixed bank which makes it easier to get the block. The disadvantage of SNUCA is that it cannot move the frequently accessed blocks to a bank closer to the requesting core. Hence the average cache access time increases in case of repeated accesses to the farther banks. In DNUCA, the banks are divided into a number of *banksets* and a block can reside in any bank within a particular bankset. Such a relaxation allows a heavily used block to migrate closer to the requesting core. DNUCA gives better performance than

SNUCA [17], though it has a major issue of *bankset searching* time. To search a block, all the banks within a particular bankset may need to be searched, which is expensive in terms of both energy and time. Reducing block searching time in DNUCA is a major research area [1, 18, 19].

Both SNUCA and DNUCA are mainly applied to the LLC. NUCA was initially proposed for single core systems and later has been adopted for CMPs. Implementing SNUCA on CMP is simple but DNUCA creates some additional issues [1, 18] due to multiple cores sharing the blocks. Also the search time of DNUCA increases with higher number of banks in CMPs. It has been observed that DNUCA can give better performance in CMP than SNUCA provided the searching mechanism is optimized [1]. To reduce the searching time some interesting techniques have been already proposed [18, 9, 19, 11].

CMPs can be divided into two parts based on their LLC placements: centralised and Tile based [5]. In centralised, the cache banks are placed in a contiguous area on the chip and the cores are placed on the two/four sides of the LLC. Such a centralised cache CMP structure has scalability issues. Tile based CMP (TCMP) has recently emerged as a scalable alternative to centralised cache CMP designs and is probably the architecture of choice for future many-core processors [5]. It consists of multiple tiles connected over an on-chip NoC. Each tile has a processor, a private L1-cache and an L2-cache (considering L2 as LLC). The L2-cache with each tile can be private, or shared among all the processors on the chip. Since we work on shared LLC we assume the L2-cache as shared and distributed among all the tiles. The slice of L2 located in each tile is called a bank.

The tile associated with a bank is called the *local-tile* of the bank. All other tiles are called *remote-tiles* with respect to that particular bank. Also the terms *local-bank* and *remote-bank* represent the local (within the same tile) and remote banks respectively, with respect to a particular tile or core. The detail description about the CMP cache architectures is given in the next chapter. Both SNUCA and DNUCA are already implemented for centralised cache CMPs but for the case of TCMP the DNUCA architecture is less explored.

An SNUCA based TCMP is considered as baseline design for all the works. The details about the baseline design is give in next chapter.

1.3.2 Objective of this Work

The main aim of this thesis is performance improvement of LLC in TCMP. We compare our proposed techniques with the existing techniques in order to demonstrate the improvements of our proposed architectures. In order to achieve the goal, the first three proposals are based on performance improvement of individual cache banks, wherein the technique used inside one bank is transparent from another. In the other two proposals the cache banks coordinate among themselves to improve the overall effectiveness. The motivation behind our proposals is discussed in the next section.

1.4 Motivation

Its has been observed that most of the current LLC architectures are not able to use the entire LLC storage [8, 20, 21, 22]. The memory accesses are non uniformly distributed among the sets as well as among the banks. Better utilisation of cache means more number of blocks are managed to be placed in cache memory. Hence, reduces the expensive main memory accesses. Also better utilised cache can make it possible to use smaller sized cache without any performance loss. Using a smaller sized cache is always beneficial in terms of energy and cost. Reduction in main memory accesses results in the reduction of Miss Per Thousand Instructions (MPKI). Reduced MPKI results better execution performance.

The LLC utilisation issues can be divided into two categories:

- local utilisation (within each bank).
- global utilisation (among the banks).

The local utilisation issue means the sets in a bank are not used uniformly [20, 8]; some sets are used heavily while some others remain underused. Such non uniform use of sets reduces the utilisation factor as the heavily used sets face more misses while underused sets remain idle. The loads in the LLC of TCMP are also not uniformly distributed among the banks. Some banks are heavily loaded while some other remains almost idle [22]. Such non-uniform load distribution among the banks is called the global utilisation issue.

The local utilisation of each bank can be improved by allowing the heavily used sets to increase their associativity dynamically without increasing the bank size. We call this technique as Dynamic Associativity Management (DAM). In our first three works we propose DAM based techniques to improve the cache performance. Block migration as used in DNUCA can reduce the block access time but additional mechanisms are required to improve the global utilisation. Also DNUCA is not well explored for TCMP as is done in the centralised cache CMP. Our other two works are based on improving the cache performance in TCMP by proposing a DNUCA based architecture. The first work is to propose a DNUCA based architecture for TCMP (T-DNUCA) with a mechanism to minimize search time and increase utilisation. The second work is to further minimise the search time of T-DNUCA by separating tag and data arrays.

The main aim of this entire research work is to improve the performance of LLC in TCMP. To do this our contributions are in the following directions:

- Increase the utilisation of LLC: The LLC utilisation can be improved by: (a) enhancing local utilisation and (b) enhancing global utilisation.
- Proposing DNUCA based design for TCMP: DNUCA gives better performance than SNUCA. Also uniform load distribution among the banks is possible with DNUCA based architectures.
- Reduce the search time of DNUCA based TCMP: The bankset searching time in DNUCA based TCMP must be optimal.

1.5 Proposed Works

1.5.1 CMP with Victim Retention (CMP-VR): a DAM based Technique

In our first work we propose a DAM based technique called CMP-VR to increase the utilisation of the cache sets within a bank (local utilisation). CMP-VR divides the ways of each set into two storage groups: normal storage (NT) and reserve storage (RT). NT behaves same as conventional cache and RT constitutes 25% to 50% ways of every set. In order to allow a set to access or use the reserve locations (ways) of any other set within the cache, a heavily used set can use the RT section of other underutilised sets to dynamically increase its associativity.

Whenever a block request arrives from L1, the block is searched simultaneously in its dedicated cache set and in the entire RT section. Note that, the dedicated cache set means the NT section of the set in which the block is mapped according to its index bits.

Searching the entire RT directly from the cache is an expensive operation as it requires to search all the sets within the cache (the RT section being made of ways from each cache set). To overcome this problem, CMP-VR uses an additional tag-array, TGS for the RT. Each entry in TGS stores the tag address of a block currently stored in RT. There is a one-to-one mapping between the TGS locations and RT locations, i.e. each location in TGS maps to a corresponding location in RT. Hence if a tag is found in TGS then based on its location (array index) the corresponding RT location can be directly computed. Each set maintains separate LRU replacement policy for its NT section while the entire RT section maintains a common global LRU replacement policy.

CMP-VR tries to distribute the load among all the cache sets uniformly. The technique is implemented on each bank separately and it is completely transparent from other components of the CMP. CMP-VR is compared with the baseline TCMP. It

gives 7.75% performance improvement (CPI) as compared to the baseline. CMP-VR having larger RT gives even better improvements but such configurations are not recommended to use for their additional hardware requirements.

1.5.2 CMP-SVR: an Improved DAM based Technique

CMP-VR improves the performance of TCMP but its fully associative TGS consumes significant amount of energy during the execution. Most of the previously proposed DAM based technique has hardware overheads. To minimize the hardware overheads of dynamic associativity management we proposed another technique, called CMP-SVR, which uses a set-associative TGS. In CMP-VR, there is no restriction on the dynamic associativity increase of a set; in worst case a heavily used set can consume the entire RT section. Experiments found that it is sufficient to allow a heavily used set to increase its associativity by double or triple. CMP-SVR restricts the maximum possible associativity increase of a set. Here the sets are grouped into multiple *fellow-groups* and all the sets belonging to the same group are called *fellow-sets*. A set can only use the reserve ways of its fellow-sets and hence each fellow-group has to maintain a separate RT section. In worst case a set can only use all the reserve ways of its fellow-sets. Since heavily used sets require limited number of additional ways [8], such restriction of CMP-SVR does not cause major performance degradation over CMP-VR. The CMP-VR can be considered as CMP-SVR having only one fellow-group.

The additional tag-array SA-TGS of CMP-SVR is set associative instead of fully associative. Each entry in SA-TGS has a one-to-one mapping with a corresponding location in the RT (same as in CMP-VR) and hence no forward/backward pointers are required as were needed in [8, 10]. CMP-SVR improves performance by 6.76% which is slightly less than CMP-VR. The main benefit of CMP-SVR is that it has very less energy overhead (1.8%) over the baseline.

1.5.3 Fellow Set based DAM (FS-DAM)

The main motive behind proposing CMP-SVR is to reduce the hardware overhead of CMP-VR. Though the hardware overhead is negligible in CMP-SVR it gives almost similar performance as CMP-VR and not more than that. To improve the performance of CMP-SVR we proposed a modified version of CMP-SVR called FS-DAM. In CMP-SVR the grouping process is static and does not depend on the loads of the set. Due to such static grouping, it may be possible that during the execution all the sets of some fellow-groups are heavily used. The dynamic nature of the sets cannot be predicted at the starting of the execution. The non-uniform load distribution of each individual set is handled by CMP-SVR, but the non-uniform load distribution among the fellow-groups is not handled properly. Some fellow-groups may contain more heavily used sets (H-Sets) while some other groups may be made of all lightly used sets (L-Sets). Since CMP-SVR performs almost same as CMP-VR (with lesser hardware requirements), making the fellow-groups based on the current set-loads must improve the performance.

The grouping is done at run-time such that the H-Sets can be distributed among all the fellow-groups. Such load based set distribution increases the utilisation of LLC. The sets are re-grouped after a fixed interval of execution to effectively utilise the dynamic behavior of the sets. The H-Sets are given higher priority in the RT section of each fellow-group while in CMP-SVR there is no such priority. FS-DAM gives better result than CMP-VR and CMP-SVR with much lesser energy overhead. FS-DAM is our improved and final DAM based technique to increase the utilisation of a bank. It improves the performance by 6.15% over CMP-SVR.

From now onwards the heavily used sets and lightly used sets are also referred as **H-Sets** and **L-Sets** respectively.

1.5.4 T-DNUCA: a DNUCA based TCMP Architecture

Proper implementation of DNUCA based CMP gives better performance than SNUCA [17, 1]. To improve the performance of TCMP we propose a DNUCA based architecture for TCMP (T-DNUCA). T-DNUCA has the following DNUCA properties:

- It divides the banks into multiple banksets; a block can be placed in any bank within a particular bankset.
- A heavily used block can be migrated from one bank to another within the same bankset.

The existing proposals to solve the issues of DNUCA are mostly for centralised cache CMPs and they cannot be directly applied to TCMP as their architectures are different.

The proposed T-DNUCA has the following properties:

- Every core has a *home-bank* in each bankset. The bank from each bankset having shortest distance from the core is its home-bank in the corresponding bankset.
- To search a block, the home-bank is searched first (local-search). If the block is not found then the remaining banks of the bankset are searched simultaneously (remote-search).
- To reduce the search time and energy consumption we proposed a placement strategy such that most of the blocks can be initially placed in the home-bank. Since most of the blocks are private to a particular core [1], placing them in their home-bank can reduce the remote-search.
- To utilise all the banks uniformly we propose a cascaded replacement policy. A block replaced from one bank is moved to another bank. The process helps a block evicted from a heavily used bank to be placed in a lightly used bank.

Since blocks are managed in a distributed manner, maintaining consistency is a major part of this work. We design the complete protocol to support the distributed behavior of TCMP by extending the MESI-CMP based protocol.

T-DNUCA is compared with baseline TCMP to show that the proposed T-DNUCA is a better architecture for TCMP. T-DNUCA has all the advantages of DNUCA like migration, better bank utilisation, etc. T-DNUCA gives 6.59% improvements over SNUCA based TCMP.

1.5.5 Improvement Over the T-DNUCA Architecture

Although the proposed block placement policy in T-DNUCA reduces the chances of remote-search, however the overheads cannot be completely ignored. Also since a block cannot be moved out of its bankset even after getting migrated to the closest bank (home-bank), the block may still be far from the requesting core due to the position of the bankset. Another issue is that all the banksets may not always be used uniformly. This encourages us for more improvements of T-DNUCA for better bank utilisation and minimization of search time.

1.5.5.1 T-DNUCA with Tag Lookup Directory (TLD-NUCA)

In this technique all the banks in LLC form a single bankset. Thus each bank can store any block enabling the heavily used block to be migrated directly to the local-bank. Also the banks can uniformly distribute loads among themselves by moving the blocks from one bank to another based on migration and cascading replacement policies. Since a block can be in any bank, all the banks have to be searched before declaring a hit or miss. To reduce the block searching time we propose a centralised tag storage concept. The tags of all the blocks stored in the L2 banks are also duplicated in a central storage called Tag Lookup Directory (TLD). The TLD is not necessarily to be designed as a single unit; it can be divided into multiple parts and distributed among the various regions of the chip. The TLD contains the tag and a pointer to the L2 bank where the block resides.

A block request (from L1) first comes to the local-bank. If the block is found in the local-bank then the request can be handled without interacting with the TLD, otherwise the request is forwarded to the TLD. The protocol designed for TLD-NUCA is different from T-DNUCA and gives better performance. TLD-NUCA gives 6.5% improvements over T-DNUCA and 13% over SNUCA based TCMP. Duplication of tags in TLD results in 10.5% of storage overhead and 3.08% of energy overhead. A three dimensional NoC based TLD-NUCA is also proposed for better scalability.

1.6 Summary

Tiled CMP (TCMP) is gradually becoming the main architecture of CMP. The performance of Last Level cache (LLC) in TCMP has a major impact on the performance of the entire CMP. This research work is motivated to improve the performance of LLC in TCMP by better utilisation of it.

To improve the LLC utilisation we worked in two directions: (i) increase the utilisation of each bank internally (local utilisation enhancement) and (ii) uniform distribution of loads among the banks (global utilisation enhancement). To improve local utilisation we used the concept of dynamic associativity management technique (DAM). The proposals CMP-VR, CMP-SVR and FS-DAM are proposed for this. The three techniques are proposed based on different mapping policies, performance and hardware overheads. In these techniques the heavily used sets are allowed to share the idle ways from other lightly used sets. CMP-VR enhance the local utilisation and hence improves the performance (CPI) by 7.75% as compared to baseline TCMP. CMP-SVR is proposed to reduce the hardware overheads of CMP-VR. The energy overhead of CMP-SVR is less than 2% as compared to the baseline TCMP. Both CMP-VR and CMP-SVR show almost similar performance improvements. To improve the performance even further or to enhance the local utilisation even more we proposed FS-DAM. FS-DAM has better control over the dynamic changes of the sets behavior. The hardware overhead of FS-DAM

is almost same as CMP-SVR. Hence FS-DAM is our most efficient DAM based policy.

To increase the LLC utilisation among the banks (global utilisation) we propose a DNUCA based architecture (T-DNUCA) for TCMP. Migration and cascading replacement used in T-DNUCA helps to distribute the load among multiple banks (within same bankset). A smart placement policy is also proposed to reduce the block searching time in T-DNUCA. Better block placement reduces the block searching time in T-DNUCA and hence improve performance. The T-DNUCA is further improved by proposing another technique called TLD-NUCA. TLD-NUCA is proposed to minimize the search time of T-DNUCA. It has only one bankset to allow better utilisation of all the banks. A centralised tag directory is used for faster block searching. TLD-NUCA gives 6.5% and 13% better performance than T-DNUCA and T-SNUCA respectively.

Since our main motive is to improve the performance of LLC, all the proposed architectures have some additional hardware requirements as compared to the baseline TCMP. But care has been taken to minimize this additional overheads. Our improved techniques like CMP-SVR, FS-DAM and T-DNUCA has almost negligible hardware overheads. The overheads of TLD-NUCA can also be reduced.

1.7 Thesis Organization

The rest of the thesis is organized as follows. The next chapter discusses the background details of LLC architectures in CMP. The existing works are also discussed in this chapter. Chapter 3 gives the experimental methodology of all the works mentioned in this thesis. The chapter discusses how the proposed architectures are modelled (simulated) and compared with other existing architectures. The proposed technique CMP-VR is discussed in Chapter 4. Chapter 5 discusses about CMP-SVR and FS-DAM. The DNUCA based TCMP design (T-DNUCA) is discussed in Chapter 6. TLD-NUCA is discussed in Chapter 7. Finally conclusion and future works are given in Chapter 8.

Chapter 2

Background

Previous chapter gives a basic overview about the CMP cache architecture. This chapter gives the detail description of LLC architecture in CMP and different existing works in this area.

As mentioned in Chapter 1, the cores in CMP can have many levels of caches. The structure of upper level caches in CMP and single core systems are almost similar because of its smaller size and faster access. The Last Level Cache (LLC) in CMP is larger in size and most of the time shared by all the cores. The performance of CMP largely depends on the performance of the LLC [5, 17]. Additional coherence mechanism is required in CMP for maintaining consistency among the multiple caches at different levels. Though the main target of this work is CMP but the cache structures of CMP are inspired from some cache architectures of single core systems. For better understanding of CMP cache architectures, this chapter gives a brief description about a promising single core LLC structure (NUCA) from which most of the CMP based LLCs are derived. The chapter also discuses about different cache properties which are common in both single core and CMP systems before discussing about the CMP based LLCs. In our entire work we considered a two level cache hierarchy (L2 as LLC) though it is possible to extend the works for more levels.

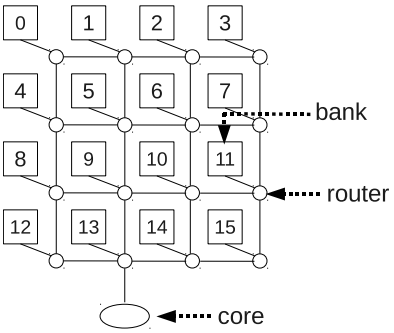


FIGURE 2.1: Example of NUCA architecture in single core systems.

2.1 Cache Architectures in Single Core Systems

Initially most of the cache structures were designed to have uniform access time regardless of the block being accessed. For such Uniform Cache Access (UCA) architectures, time became a significant bottleneck as the caches become larger [17, 5]. In an alternative solution, called NUCA, the large LLC is divided into multiple banks and each bank can be accessed as an independent cache. A core can access its closer bank much faster than the farther banks because of lesser on chip communication distance. Figure 2.1 shows an example of NUCA architecture designed for single core systems. It can be observed that bank-13 is closer to the core (C) as compared to bank-2. Any request from C to bank-2 must travel through the NoC and hence requires additional time. NUCA is divided into two categories: static NUCA (SNUCA) and dynamic NUCA (DNUCA). The architecture of both SNUCA and DNUCA are same but they differ in their data management policies.

The term *bank* is used for LLC banks in NUCA based designs. In this entire thesis the terms *bank*, *L2-bank* and *LLC-bank* represents the same entity. Each bank is assigned a number starting from 0. The term L_2^i such that $0 \leq i < N$, represents the i^{th} bank, where N is the number of banks in the system. The entire LLC is termed as either LLC or as L2.

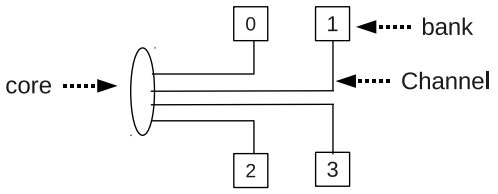


FIGURE 2.2: An example of SNUCA1.

2.1.1 SNUCA

In SNUCA a block always maps to a fixed bank which makes it easier to search the block. SNUCA is also divided into two categories: SNUCA1 and SNUCA2. Figure 2.2 shows an example of SNUCA1. Here each bank has its dedicated two-way channel connected to the controller. Such dedicated channels consumes a major area on the chip and hence restricts the number of banks that can be accommodated on the chip. SNUCA2 uses a two dimensional NoC to connect all the banks with the controller. Figure 2.1 is an example of SNUCA2 structure. The replacement of dedicated private channels by the NoC significantly reduces the area overheads and improves scalability. In SNUCA2 the LLC can now be divided into more number of banks for better performance. The architecture of SNUCA2 and DNUCA is similar but they differ in their data management policies.

From now onwards the term SNUCA is used for SNUCA2. SNUCA1 is never mentioned any more in this thesis.

2.1.1.1 Data management policy in SNUCA

Data management policy means how data (cache blocks) are managed in SNUCA. Three major issues of data management policy in any cache architecture are: **block mapping**, **block searching** and **block replacement**.

Mapping means where to place (map) a block within the cache or bank. A block address in conventional cache is divided into three parts: (a) *block index* (b) *set index* and (c) *tag address*. Figure 2.3 shows the mapping details of a conventional

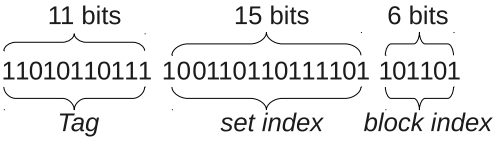


FIGURE 2.3: A 32-bit block address showing the mapping policy of conventional cache. The cache size is considered as 8MB. The associativity and block size are 4 and 64 bytes respectively. *set index* means the index bits required to identify a set within the cache.

cache. In a conventional cache (not divided into banks) mapping means using the *set index* bit from the block address to decide which set in the cache the block belongs to. In the case of NUCA architecture the LLC is divided into multiple banks. Hence bank selection is also a part of the mapping policy. SNUCA has fixed bank selection policy: based on the block address it always maps the block to a fixed particular bank. The block address in SNUCA is divided into four parts: (a) *block index* (b) *set index* (c) *bank index* and (d) *tag address*. Figure 2.4 shows the mapping details of SNUCA. The *bank index* depends on the number of banks and *set index* depends on the number of sets available in each bank. The order of *bank index* and *set index* given in the figure can be interchanged to allow *bank index* to appear just after *block index*. Making lower significant bits as *bank index* distributes the contiguous memory blocks into different banks.

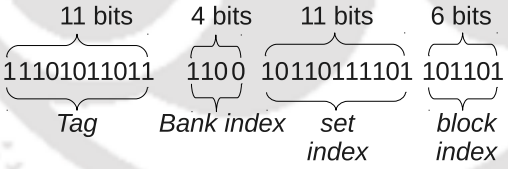


FIGURE 2.4: A 32-bit block address showing the mapping policy of SNUCA. The cache size is considered as 8MB with 16 banks, hence each bank is 512KB. Associativity of each bank is 4 and the block size is 64 bytes. *set-index* means the index required to identify a set within a bank (each bank has 2048 sets). *bank-index* is to identify the bank.

Searching: To search a block in conventional cache the assigned (mapped) set has to be searched. The mapping policy selects the set and then searching policy searches the block within that set. Each set has multiple ways (considering a set associative cache) and all the ways may require to be searched. This process is

called tag comparison and it can be done either in parallel or sequentially. Sequential tag comparison is slow but takes less hardware as only one set of comparators is required to compare all the tags. Parallel search is faster but requires separate comparators for each way in the cache. Even though parallel search is expensive it is used for most of the set associative caches for better performance [6]. The mapping policy in SNUCA selects the bank as well as the set (in the selected bank) where the block may be present. Once the bank and the set are mapped the searching process in SNUCA is same as in a conventional cache.

Replacement : The replacement policies used in SNUCA are local to every bank. Since each bank is considered as a conventional cache, the replacement policy in SNUCA and conventional cache are the same. Section 2.1.4 discuss about different replacement policies of cache memories.

The Complete Process: Whenever an L1 cache (say L_1^x) has a miss on a block (say B), it requests the corresponding L2 bank to get the block. The selection of the L2 bank is done based on the *bank index* as mentioned above. Consider the L2 bank selected as L_2^y. According to the property of SNUCA the block cannot be in any other bank except L_2^y. Once the request is sent to the proper L2 bank the controller of the corresponding bank maps the block to a particular set (say s_i) based on the *set index*. The searching policy as mentioned above searches the block in s_i and sends the block to L_1^x if found (i.e. tag matched). Otherwise, the block has to be fetched from the main memory before sending it to L_1^x. The controller of L_2^y sends a request to main memory for the block. If there is no free space available for the newly fetched block then a victim block from L_2^y is selected based on the local replacement policy to be replaced with the new block. The replacement process is done in parallel with the fetching process from main memory such that the fetched block can be placed immediately in L_2^y.

2.1.1.2 Disadvantage of SNUCA

SNUCA has fixed mapping policy and a block always maps to a fixed particular bank. Such a mapping policy makes the block searching process easier and faster.

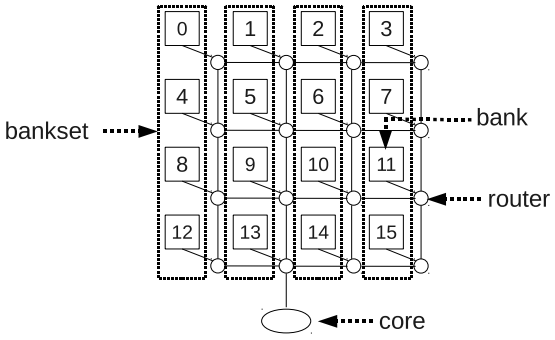


FIGURE 2.5: An example of DNUCA.

Although SNUCA design is very simple, it has performance issues due to its cache access latencies [17]. When a core requests a block which is far away from the core then there is no mechanism to bring the block closer. Also fixed mapping can make some banks heavily loaded while some other banks remain underused. SNUCA has no mechanism (not possible with fixed mapping) to dynamically balance the loads among multiple banks.

2.1.2 DNUCA

DNUCA, allows dynamic mapping of the cache blocks among the banks. The banks are divided into groups called *banksets* and a block can be placed in any bank within a given bankset. Such dynamic nature facilitates migration of heavily used blocks closer to the requesting cores. All the banks within a bankset are called *peer-banks*. Though the original DNUCA has not proposed any load balancing techniques among the banks, it can be implemented on a DNUCA structure. In this thesis we propose techniques for balancing load among the banks of DNUCA. Figure 2.5 shows a DNUCA with 16 banks. The banks are divided into 4 banksets as depicted in the figure.

2.1.2.1 Data management policy in DNUCA

Mapping: Since a block can be placed in any bank within the bankset the *bank index* of SNUCA is replaced with *bankset index* in DNUCA. Here the *bankset index*

maps a block to a particular bankset. The *set index* is same as in SNUCA used to map a block to a set. A set can be assumed as distributed to all banks within a particular bankset. For a particular block address a set is fixed in each peer-bank. Figure 2.6 shows the mapping policy of DNUCA for a block (say B). Based on the *bankset index* it can be decided that the block maps to bankset 1 (bs_1). Since the block is allowed to place in any bank within bs_1 each bank has a fixed set for the block. The *set index* shown in the figure decides that the block can be placed in the 1469^{th} set of any bank within bs_1.

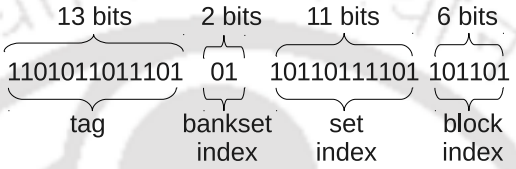


FIGURE 2.6: A 32-bit block address showing the mapping policy of DNUCA. The cache size is considered as 8MB with 16 banks, hence each bank is 512KB. Associativity of each bank is 4 and the block size is 64 bytes. *set index* means the index required to identify a set within a bank (each bank has 2048 sets). *bankset index* is to identify the bankset.

Searching: The bankset index and set index fix the set in which the block may be placed. However it can still be in any one bank within the bankset. Therefore in DNUCA, all the peer-banks may need to be searched to get a block. Such searching is called *bankset search* and is the major bottleneck of DNUCA. Expensive and time consuming bankset search may absorb all the benefits of DNUCA. The three basic mechanisms of bankset searching are: incremental search, multi-cast search and mixed search [17]. In case of incremental search the bankset is searched one bank at a time starting from the closest bank until the block is found or a miss is declared after searching till the last bank. Multicast search simultaneously sends (multicast) the search request to all the banks within the bankset. Search in all the banks is performed roughly in parallel depending upon the physical distance and the router delay. Incremental search minimizes the number of messages in the NoC and hence is less expensive in terms of energy consumption and network overhead. But average cache access time is longer in incremental search. Multicast search gives higher performance but is expensive in terms of network and energy consumption. Mixed search is a mixture of both incremental and multicast search

where first few banks in the bankset are searched simultaneously and remaining banks are searched incrementally. Though mixed search is a better option than the other two, the bankset search is still time consuming in DNUCA. Additional smart searching techniques also exist to minimize the search time [17]. But optimizing bankset searching time, especially for CMPs, is still an open issue [5]. Fortunately many innovative techniques are already proposed (discussed later) for bankset search. For single core DNUCA the three basic searching techniques are sufficient. Note that each particular bank behaves same as in a conventional cache and a block has to placed in its dedicated set.

Every bank in DNUCA behaves like a conventional cache and hence the replacement policy of each bank in DNUCA is same as in a conventional cache. The cache replacement policy is described in Section 2.1.4. In our proposal we use an inter bank replacement policy called cascading replacement. Chapter 6 and 7 gives the detail description about it.

2.1.3 Inclusive and Non-inclusive Cache

An inclusive cache design (considering a two-level cache hierarchy) means L2 has the copy of all the blocks currently in L1s. When a block needs to be evicted from L2 the block must be evicted from all the L1s where it currently resides. Coherence mechanism must take care of this. In non-inclusive cache, L2 does not necessarily store the backup copy of the blocks currently in L1s. A block in L1 may or may not be in L2. Another strict non inclusive design is also used for some architectures where the blocks in L1 and L2 are completely different. In this case not a single block is allowed to place in both the levels at the same time.

The advantage of inclusive design is its simplicity in block searching. When an L1 miss occurs the L2 cache can decide the current status of the block without contacting any other L1s. If the block is in the cache then definitely L2 has the block. The directory information of a block can be stored with the blocks in L2 to manage the coherence among the L1s easily. A block not in L2 means it is a miss

and has to be fetched from the main memory. The disadvantage of this design is the wastage of space in L2 for storing the blocks which are also in L1s.

Non-inclusive and exclusive designs has complex search mechanism. For any L1-miss all the other L1s as well as L2 has to be searched to get the block. The concept becomes complicated and expensive for CMP where multiple L1s exists.

2.1.4 Cache Replacement, Insertion and Promotion Policies

Replacement policy plays a major role in the performance improvement of CMP last level caches [5]. The most efficient replacement algorithm for cache is to always discard the information that will not be needed for the longest time in the future. This optimal result is referred to as Belady's optimal algorithm [23]. But it is not implementable in practice because it is generally impossible to predict how far in future the information will be needed. Recent studies found that there is a huge performance gap between LRU and the Belady's optimal algorithm and the miss rate using LRU policy can be up to 197% higher than the optimal replacement algorithm [24]. Most replacement algorithms maintain each set separately. The LRU replacement policy used in conventional caches is a local replacement policy and hence each set has its own recency data structure (*lru-list*). The first and last block of *lru-list* are the LRU and MRU (most recently used) blocks of that set respectively.

Other then replacement policy, insertion and promotion policies of a block also plays a vital role in cache performance [25]. In LRU policy a newly arrived block is inserted into the MRU position and after every access the block is promoted to MRU position. If a block is inserted (or promoted) into MRU, then in order to get it evicted, it has to wait till it becomes the LRU block of the set. The blocks which were requested only once or twice take unnecessarily long time to become LRU and get evicted. Such blocks are called *dead blocks* [26, 27] and they unnecessarily waste cache storage. In many applications there are some blocks which are used

multiple times but within a small time interval [28]. Such requests to a block over a smaller interval can be served by the L1 cache (except for the first time) and never need to request L2 again. So the chances of a block becoming dead in LLC is more than any upper level caches.

Pseudo-LRU has lesser hardware complexity as compared to LRU and performs almost same as LRU [29]. There are some other interesting proposals for cache replacement policies [30] but most of them have complex mechanisms and hardware overheads for maintaining aging bits, counters etc. However LRU performs better in many applications having higher temporal locality [5]. Hence, instead of completely removing the concept of LRU, researchers proposed many innovative techniques to improve the performance of LRU based policies [31, 32, 33, 34, 35]. Dynamic insertion policy (DIP) [31] is a technique proposed to solve the problem of dead blocks by inserting blocks directly into LRU position. But DIP also has disadvantages as it may repeatedly replace heavily used blocks [25].

Cache partitioning is also a major area of research in CMP [36, 25, 37, 38, 39, 40, 41]. Different way-based partitioning techniques have been proposed to resolve the disadvantages of DIP. When cores are executing their own applications, a proper partition of the cache among the cores makes a huge impact on performance. Way based partitioning divides the ways of each set among different cores based on their requirements. Such partitioning techniques can be handled by changing the replacement policy of LLC. Some way-based partition are statically fixed while dynamic partitioning techniques are also proposed. UCP [36], PIPP [25], CCPIP [42] are the examples of way partitioning in shared LLC of CMPs.

Though our proposals have not experimented with cache partitioning and replacement but these policies can be implemented on top of our any proposed designs. For most of our designs we consider basic LRU policy for local replacement. Better replacement policy and cache partitioning techniques can further improve the performance of the proposed policies but these are left as future work. We use additional mechanisms for block replacement and are discussed whenever required.

In Chapter 6 and 7 we proposed our own replacement policy for inter bank (global) replacement.

2.2 Cache Architectures in Chip Multiprocessors

NUCA was initially proposed for single core architectures, most of the above NUCA descriptions are for single core designs. The technology is gradually extended for CMPs and now almost all CMP architectures use NUCA based LLC design. Before discussing about the NUCA designs in CMP we first discuss about the LLC architectures of CMP.

The LLC architecture in CMP can be either shared or private. Based on the current large size applications the shared LLC is a better fit in requirements, but the performance of such shared LLC must be improved to incorporate the growing cache size demand. Many innovative techniques are already proposed to minimize the cache access time of shared LLC [18, 1, 19, 37, 43, 9].

2.2.1 CMP with Shared LLC

CMP having shared LLC can be categorized in two different ways:

- CMP with centralised shared LLC (CCMP).
- CMP with Tiled Shared LLC (TCMP).

In CCMP the LLC is considered as a centralised entity and placed in a contiguous area on the chip. All the cores are placed on the two/four sides of the LLC. One example of such centralised shared L2 cache is shown in Figure 2.7. Even though the cache is partitioned into multiple banks and the cache controller is distributed over all the banks, the LLC is still called as centralised because it occupies contiguous space on the chip.

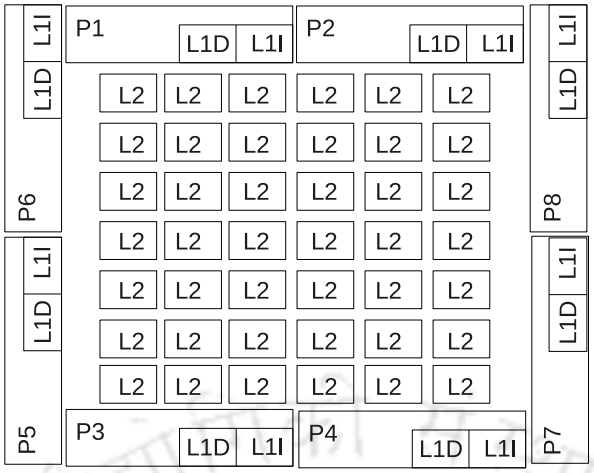
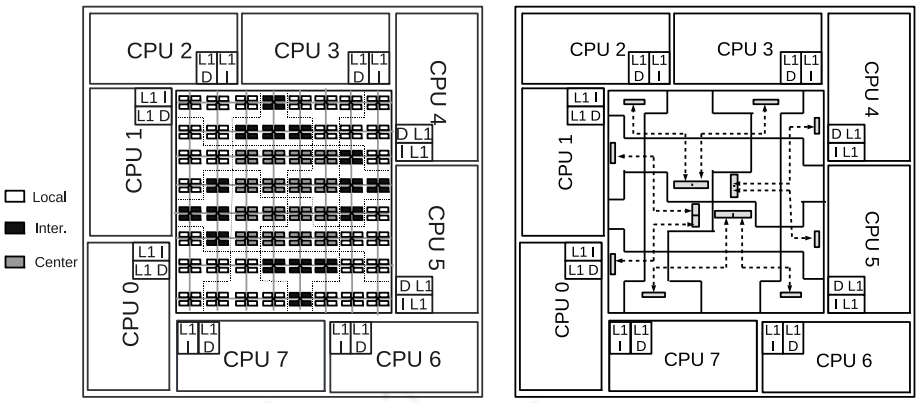


FIGURE 2.7: An example of CCMP.

This type of centralised architecture is good enough for CMPs with small number of cores, but when the number of cores increases, such architectures degrade performance [5, 44]. Researchers have later proposed an alternative of such centralised architecture which is called "CMP with Tiled shared LLC" or TCMP [5]. But since most of the existing NUCA based CMP architectures are proposed for CCMP, we first discuss about CCMP and its design issues. The next subsection discusses the complexities of implementing NUCA architectures for CCMP and also the existing techniques to solve the complexities. The details regarding TCMP is given separately in Section 2.2.3.

2.2.2 NUCA for CCMP

Implementing the basic SNUCA design for CCMP is straight forward but its performance improvement has always been of research interest [45]. We call the SNUCA architecture for CCMPs as C-SNUCA. The LLC banks of C-SNUCA have fixed mapping policy as in SNUCA (single core). The additional mechanism required here is the coherence protocol to maintain consistency among the different L1 caches which are private to each core. MESI-CMP [46] is an example of such a protocol. Since multiple cores shares the LLC, partitioning the storage of LLC among the cores based on their workload-demand is another area of research.



(a) a (b) b

FIGURE 2.8: C-DNUCA proposed in [1].

Section 2.1.4 already mentioned about the basic cache partitioning techniques. Some recent work in cache partitioning are [36, 25, 42].

DNUCA outperforms SNUCA in case of single core processors [17]. But extending it for CMP has some basic difficulties. As the size of the LLC in CMP increases the number of banks also increases. Hence the bankset searching time becomes a major challenge. Block migration may cause a block to ping-pong between two cores. Though the issue of migration can be handled easily but efficient bankset search is still an open area of research [5]. There already exists contributions for implementing DNUCA based CCMPs. However, most of these works had to suffer from the overheads of a fairly complex bankset search mechanism [5]. We call the DNUCA based CCMPs architecture as C-DNUCA.

2.2.2.1 Innovations in DNUCA based CCMPs

The first CMP based NUCA architecture was proposed by Beckmann and Wood [1]. They considered a CCMP architecture with eight cores. The banks are divided into multiple regions to allow a block to be migrated to a region closer to the requesting core. There are 16 regions, out of these 8 are called local regions (one for each core), 4 are called center regions and remaining 4 are called internal regions. Initial placement is somewhat random (based on the blocks tag bits).

From here, a block is allowed to gradually migrate to different regions based on the cores that access it. A block is migrated from the local region of one core to the local region of another core through the inter regions and center regions. To prevent ping-ponging a block among multiple regions migration is only allowed after a successive number of requests from a core. Figure 2.8 shows the proposed architecture of [1].

In this work the authors have reported that the amount of shared data in most of the workloads is less, but the frequency of accessing this data is very high. Hence the migration rule makes most of the shared data to be placed into the center regions, which are far away from all the cores. Normal RC based wires used as on chip interconnects are slow [47] and cannot access those center regions rapidly. Hence to reduce access latency the authors use high speed transmission lines [47] to connect directly each core to the center regions. Note that transmission lines can communicate data at the near speed of light. The details regarding transmission lines is out of our context. The most significant problem with the above architecture is the difficulty in locating a block. In a DNUCA that distributes the ways across the banks, it cannot statically determine the exactly location of a block. That means for every request it needs to search a set of banks to get the block (assuming it to be a hit). The authors use a multicast based mechanism for bankset search.

Another paper appeared shortly after [1] is [18]. Their experiments found that C-SNUCA has longer access time on average. C-DNUCA on the other hand shows better performance than C-SNUCA when migration is allowed. But they also mentioned that block searching in C-DNUCA is a major bottleneck. To reduce the requirement of accessing multiple banks, they implemented a distributed set of replicated partial tags. The partial tags for each block are stored at the top and bottom of every bankset (column). Such partial tags help to a reduced number of banks to be searched. But the additional tag storage required for partial tags creates a significant hardware overhead. Also the tag storages have to be updated with every on-chip change and a coherence mechanism needs to be implemented for these partial tag storages.

Re-NUCA [48] allows limited replication for shared blocks accessed by processors placed on the opposite sides of the chip. In particular, their solution allows at most two independent copies of the same block to be stored in the same shared LLC, each of them migrating towards the closest LLC side. An efficient data search algorithm for C-DNUCA has been proposed in [9] (HK-NUCA). In HK-NUCA each block has a statically mapped home bank whose purpose is to know which other NUCA banks have at least one of the data blocks that it manages. The home-bank based mechanism significantly reduces the search-time, network traffic as well as miss rate. DR-SNUCA [49] and ESP-NUCA [50] are proposed for reducing energy overhead of NUCA based architectures.

Chishti et al. proposed an alternative architecture for DNUCA called NuRAPID [10]. It gives better performance than DNUCA and also has less complexity. The original NuRAPID was proposed for single core architectures. The main contribution of the work is to separate the tag and data arrays. The tag arrays are separated from the data arrays and stored to a special storage called tag storage near the processing core. The tag storage stores the tag address of each block as well as a pointer to the location where the data is present. The data are stored in the banks. Such tag storages minimize the tag lookup time as the storage is nearer to the core. Also the forward pointer stored in the tag array allows to assess the block directly from the bank without searching any bank. The forward pointers allow data to be placed in any location of a bank irrespective of the *set index*. Such flexibility allows more number of heavily used blocks to be placed in the closer bank. The overhead in providing such a flexibility is that the data blocks need to maintain reverse pointers to identify their entry in the tag array, so that the corresponding tag can be updated when a move happens.

The CMP extension of NuRAPID is proposed in [19]. In this design each core has its own tag storage. Though it shows better performance than corresponding DNUCA, managing multiple tag storage is a major drawback of the work. The tag storages duplicate same content and need to be consistent with any changes in the cache. Hence coherence mechanism has to be implemented for them.

A shared/private hybrid LLC is proposed in [51]. The LLC is distributed into multiple banks same as in C-DNUCA but each bank reserves some ways as private to the local core. The blocks are initially placed into the private ways of the local bank of the requested core. In case of eviction, the block is placed into the shared ways of any bank. A prediction based method is used in [52] to minimize the bankset searching overheads. It uses the concept of Bloom filter [53] for possible location of a block in each bank. The design reduces the storage overhead by a factor of 10 as compared to the partial tag structure. But the use of Bloom filter introduces a non-trivial number of false positives.

Some other proposal for C-DNUCA are [54, 55, 56]. In [57] the migration of DNUCA is done earlier based on some prediction to reduce the hit time. In summary even after proposing many innovative designs the problem of minimizing the cost of bankset searching is still a challenging task. A better placement policy can reduce the requirement of total bank accesses per bankset search. Chapter 6 and 7 use this concept for better block placement.

Most of the DNUCA designs discussed above are purely made for CCMP and they cannot be directly applied on TCMP (more details are discussed in the next section). Considering the scalability issues of CCMP and the growing demand of TCMP [5] current researchers are giving more importance to TCMP than CCMP [58, 59, 60, 61]. The next section discuss about TCMP and its NUCA designs.

2.2.3 Tile Based CMP (TCMP)

Tile based CMP (TCMP) has recently emerged as a scalable alternative to current small-scale CCMP designs and will be probably the architecture of choice for future many-core processors [5, 58]. It consists of multiple tiles connected over an NoC. Each tile has a processor, a private L1 cache and a slice of L2-cache (considering L2 as LLC). The slice of L2-cache with each tile can be private, or shared among all the processors on the chip. We consider a shared L2-cache, where the slice

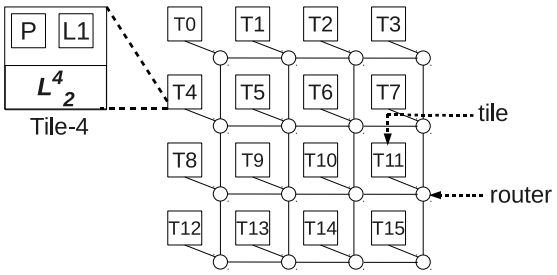


FIGURE 2.9: Tile based CMP (TCMP).

located in each tile is a bank. Inclusion is maintained between the L1 and L2 cache. Figure 2.9 shows an example of TCMP having 16 tiles.

For the simplicity of discussion this thesis uses few terms for TCMP components. The terms *local bank* and *remote bank* for a particular tile or core represents the bank within the same tile and banks in other tiles respectively. Each tile is numbered starting from 0. Each L1 cache and L2 bank are referred as L_1^i and L_2^i respectively; here $i; 0 \leq i < 16$ is the tile number. For example, in Figure 2.9, the L1 cache of the 5^{th} tile is represented as L_1^5 and the bank of the same tile is represented as L_2^5. For the same tile, L_2^5 is considered as its *local bank* while the other banks are consider as *remote banks*.

2.2.3.1 SNUCA based TCMP (T-SNUCA)

The basic SNUCA design for both CCMP and TCMP are easy to implement; the only difference is that in TCMP each bank is associated with a core, which was not the case for C-SNUCA. This thesis refers SNUCA based TCMP designs as T-SNUCA. SNUCA based TCMP is well explored [38, 62, 63, 64] compared to DNUCA for TCMP. In [65] the author proposed a pressure aware placement mechanism for TCMP. The TCMP shown in Figure 2.9 can be considered as a T-SNUCA provided it follows the data management policy of SNUCA as discussed in Section 2.1.1.1. Although the basic T-SNUCA design is straight forward many innovative variations of this design has been proposed to improve its performance. In this thesis we also propose T-SNUCA architectures for better LLC performance (cf. Chapter 4 and 5).

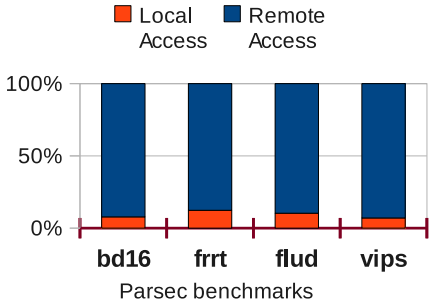


FIGURE 2.10: Distribution of local and remote bank access in T-SNUCA.

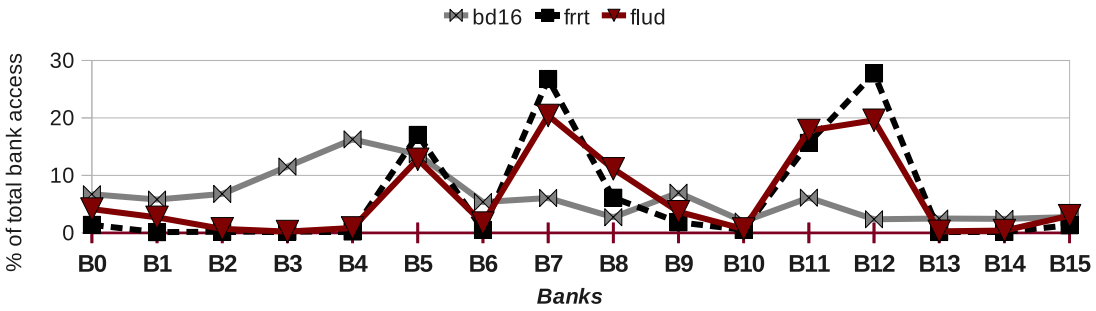


FIGURE 2.11: The load distribution among the banks of T-SNUCA.

T-SNUCA may have more number of accesses to a remote bank as the blocks are statically mapped and cannot be moved (migrated) into a closer bank. Therefore, T-SNUCA has higher average cache access latency. Figure 2.10 shows the number of *remote bank* accesses of T-SNUCA as compared to the *local bank* accesses. For this particular experiment four Parsec benchmarks [3] are used. The details regarding the experimental methodology and benchmarks are given in next Chapter. Also some *banks* in T-SNUCA can be used heavily while certain others are not. Figure 2.11 shows the bank usage of T-SNUCA over different Parsec benchmarks. Non uniform load distribution among the *banks* proves that the LLC in T-SNUCA is not getting utilised properly; thus a better utilisation of LLC can improve its performance.

2.2.3.2 DNUCA based TCMP (T-DNUCA)

DNUCA has advantages over SNUCA and hence, implementing DNUCA concept for TCMP can reduce the problems of T-SNUCA. The dynamic nature of DNUCA can move a block among the tiles (*banks*). Therefore, heavily used blocks can

be migrated closer to the requesting tiles (cores). It would also be possible to distribute the loads among multiple *banks* for better utilisation of the cache. As per our knowledge no relevant experiments have been done on DNUCA based TCMP.

2.3 Performance Enhancement of TCMP

The importance of TCMP and its future role in CMP designs are already explained by many researchers [63, 66, 58, 59]. Though the CCMP has not completely vanished and research works continues for CCMP, our research direction is towards TCMP. The target architecture of our research is TCMP and now onwards most of the discussions in this thesis are based of TCMP. CCMP is mentioned whenever necessary.

Since each bank in a TCMP behaves as an independent cache the performance of the entire LLC can be improved by improving the performance of every bank either locally or globally. Local bank improvement techniques can be implemented on any type of NUCA based CMP designs. The complete mechanism is transparent from outside the bank. Many such techniques are already proposed for both CCMP and TCMP [8, 20, 43]. In [24, 31, 67, 21, 68] better replacement policies have been proposed for individual banks. The policies can be implemented on each bank independently and transparent from outside. Such policies improve the performance of each bank independently. Way-based cache partitioning [36, 25] is also an example of individual bank improvements.

Global improvement techniques are based on improving the performance of the entire LLC. In this technique more than one bank coordinates together. For example in Co-operative caching [12] an evicted block from one bank is spilled into another bank to enable the block to remain in the cache for a longer duration.

There are many methods (either local or global) by which the performance of the cache can be improved. For example better block size, associativity, replacement

policy etc. But most of these ideas are already saturated [6]. An area where improvement is still possible is called better cache utilisation [69, 70]. As mentioned in Section 1.4 the LLCs are not utilised properly both locally [8] and globally [71, 72, 22]. Less utilisation factor makes some portions of the cache heavily used while some other portions remain idle. Proper distribution of loads among the entire LLC can increase the utilisation factor of LLC. Better utilisation means lesser miss rate (or MPKI) and hence better performance. Care must be taken to remove dead blocks from the LLC otherwise performance may not be improved even after load distribution [26, 27].

As discussed in Section 1.4 the LLC utilisation issue in TCMP can be divided into two categories: (a) local utilisation and (b) global utilisation. The first category is well explore [8, 21, 67, 20] but as per our knowledge very less work has been done on the second category (considering TCMP architecture). The next section discusses about the local utilisation enhancement techniques. The global techniques are discussed in Section 2.3.2.

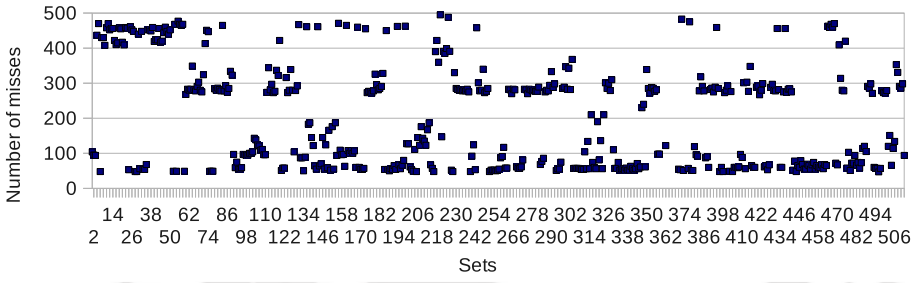


FIGURE 2.12: An example of the non-uniform behavior of sets within a particular bank.

2.3.1 Local Utilisation Enhancement

The non uniform load distribution among the sets within a bank is already discussed in Chapter 1. Figure 2.12 shows the distribution of loads among the sets of a particular bank. It can be observed that some sets are used heavily while some others are underused. This results in more misses in heavily used sets while underused sets may have idle ways. Uniform distribution of loads among all the sets can

improve the utilisation of the bank. The most popular technique for such uniform distribution is called Dynamic Associativity Management (DAM). As mentioned in Chapter 1 the DAM based techniques can manage the associativity of the cache (bank) dynamically without changing its size. Since DAM is a local technique most of the DAM based approaches can be implemented in both T-SNUCA and T-DNUCA.

2.3.1.1 Existing DAM based techniques

Though the main motive of all DAM based techniques is to dynamically manage the associativity, they differ in data management and architecture. Some promising DAM based techniques currently exists are V-Way [8], Z-Cache [20], Set Balancing Cache (SBC) [21], STEM [67], etc.

V-Way: In V-Way, the number of data blocks per set is same as in a corresponding conventional cache but the number of tag entries per set has been increased; it depends on a parameter called Tag-to-Data-Ratio (TDR). If the TDR value is 2 then number of tag entries per set is twice the number of data lines per set. Since the number of tags per set is more than the number of ways per set, there is no conventional one-to-one mapping between the tag and data lines and hence a data block can be placed in any set of the complete data array. A pair of forward and backward pointers are used for mapping between such tags and data lines. Although V-Way improves performance, it increases complexity on account of completely removing the static one-to-one tag-data mappings. Use of forward/backward pointers causes significant storage overhead. Energy consumption also increases because of doubling (considering TDR=2) the tag array. The cache access time is increased by one cycle for every access.

Z-Cache: It yields more associativity then the number of ways by increasing the number of replacement candidates. Most of the features in Z-cache are inherited from the Skew associative cache [73]. In Z-cache hit rates are increased by indexing the block with the help of H3 hash function [74]. When a miss occurs in a Z-cache the blocks that are in conflict for a particular way are not in conflict with the

other ways, increasing the number of replacement candidates. The searching for the replacement candidates in Z-cache is to be done in several steps. First, it walks in the tag array for searching of the replacement candidates by forming multiple levels and then at the last level it searches for the block that has not being accessed for a longer duration and evicts it from the Z-cache. This multilevel replacement policy is to be considered here because H3 hash function places block in a different position for each way in the cache. Finally, the relocation happens before placing the newly incoming block.This multi-level replacement policy is motivated by cuckoo hashing [75]. The negative aspect of this replacement policy is that it utilises more energy and extra bandwidth while searching in the tag array. Z-cache has very complex mechanism and cannot be implemented with conventional cache banks as it is based on a skewed associative cache.

Set Balancing Cache: In SBC [21] the sets are allowed to associate with each other for better cache utilisation. A heavily loaded set can statically or dynamically displace its block to another underused set. A saturation counter is used for each set to determine the sets as heavily used or underused. For each access of the set, the counter is incremented in the case of a miss and decremented in the case of a hit. A set is considered as heavily used when its saturation counter is $2K-1$ where K is the associativity of the cache. A set is considered as underused when its saturation counter is less than K. The sets neither heavily used nor underused are not associated with any other sets. In the case of static policy each set has a prefixed partner set and displacement is only possible if the partner set is underused. The association algorithm is responsible to fix the static association among the sets. Whenever a set tries to displace a block into its associated set the receiver set has to be underused. To check this a displacement limit is used for the saturation counter of the set. The static nature of SBC is upgraded in the next technique where the set association can be made dynamically. When a set (say s) when reaches the maximum saturation counter value it can displace its block to any other underused set. If the set (s) is not associated with any other set then a unassociated underused set is allowed to associate with s. The dynamic

nature gives better utilisation but complicates the mechanism. Additional hardwares is required to handle the dynamic association among the sets. A table called Destination Set Selector (DSS) is used to store the index of the most underused unassociated sets. The table must be updated for any change in the set behavior. For example if the size of DSS is 10 then care must be taken that the most underused 10 sets are present in DSS. To associate the sets dynamically another table called Association Table (AT) is required to provide the current partner of a set. Both static and dynamic version of SBC can increase the associativity of a set at most twice and not more than that. Also additional time is required to search the associated set as well as the DSS and the AT. Constant monitoring is required for the set behavior to update the DSS.

STEM: Replacement policies like LRU, DIP, PrLIFO [68] are responsible to determine how the LLC capacity is shared among the competing blocks of a working set. Such policies are called temporal LLC management. DAM based techniques are responsible to dynamically manage the overall capacity of the LLC by spatially partitioning it among the sets that are hosting different working sets thus performing spatial management. It has been observed that neither temporal nor spatial management can deliver the best performance while working independently. A combination of both policies is proposed in [67] called SpatioTEmporam Management (STEM). It can be consider as an extension of SBC combining with temporal management.

Other Notable Works Related to DAM: Hash-rehash cache [13] and sequential-associative cache [76] work on variable access time of increased associativity. In these works if the first attempt to access the cache is a miss then the hash function that maps the address to a set is changed and a new cache access is started. The technique was initially proposed for direct-mapped cache and [76] used this technique for minimizing the access time of a 2-way set associative cache. But todays large size LLC are typically 4 to 8 way associative and hence cannot be benefited from such techniques. In [77] the author proposed two new hashing functions: prime modulo and prime displacement, to uniformly distribute the cache accesses across the set. It improves the performance of todays large sized LLC

but cannot support inter-set distribution and hence a heavily used set cannot use the idle ways of the underutilised sets. Some more techniques related to DAM are [70, 78, 69, 79, 80].

Most of the existing DAM based technique have complex mechanisms and require significant hardware overheads to implement. In this work we proposed less expensive DAM based techniques with better performances as compared to the existing techniques. The proposed DAM based approaches are discussed in Chapter 4 and Chapter 5.

2.3.2 Global Utilisation Enhancement

The non-uniform nature of load distribution among the banks is already discussed in Chapter 1 as well as in Section 2.2.3.1. Such non uniform load distribution causes some banks to be used heavily while some others remain idle. The heavily loaded banks incur more misses and hence leads to expensive main memory accesses. The performance of the LLC can be improved if the idle banks can be used to share the loads of the heavily used banks.

Distributing loads among multiple banks depends on the LLC mapping policy. For example SNUCA cannot share the loads of one bank with another bank. DNUCA can distribute the loads of one bank with its other peer-banks (banks within same bankset). Though DNUCA is capable of distributing the loads, the original DNUCA design has no such mechanism implemented. Also as per our knowledge such load distributions have not yet been done for TCMPs.

Though the load balancing technique is not well explored for TCMPs some notable works in this area are done for private LLC based CMPs. In [12] the L2 banks are private to start with but become shared due to spilling of data. The evicted L2 cache blocks are spilled onto neighboring L2 bank thereby increasing the life time of the cache block. But this could not be scaled as the number of processors increased and the cache coherence engine became an overhead. In [7, 55], and [81] the processors are divided into clusters where each cluster has a L2 cache bank

and a directory which maintains the information about that cluster and another directory sits near the shared main memory which maintains information about all the clusters.

2.4 Summary

The main components for building today's computer systems are Chip Multiprocessors (CMPs), where multiple CPUs (cores) are placed on the same chip [5]. Most of the CMP architectures have a common large size Last Level Cache (LLC) shared by all the cores. Non Uniform Cache Access (NUCA) [17] divides the large cache into multiple banks such that each bank can be accessed with an optimal access latency. Since different banks can be accessed with different access latencies, a core can access its closer bank much faster than the farther banks. NUCA architectures are mainly of two types: static NUCA (SNUCA) and dynamic NUCA (DNUCA).

SNUCA has fixed mapping policy and a block always maps to a particular bank. Such a mapping policy makes the block searching process easier and faster. Although SNUCA design is very simple, it has performance issues due to the cache access latencies. When a core requests a block which is far away from the core, then there is no mechanism to bring the block closer. DNUCA, allows dynamic mapping of the cache blocks among the banks. The banks are divided into groups called *banksets* and a block can be placed in any bank within a given bankset. Such dynamic nature facilitates migration of heavily used blocks closer to the requesting cores. But in DNUCA the entire bankset has to be searched before declaring a cache miss. DNUCA gives better performance than SNUCA provided a optimized block searching technique is used [1].

Tile based CMP (TCMP) has recently emerged as a scalable alternative to current small-scale CMP designs and will be probably the architecture of choice for future many-core processors [5]. It consists of multiple tiles connected over an on-chip network. Each tile has a processor, a private L1-cache and a slice of LLC (L2).

Both SNUCA and DNUCA are initially proposed for centralised shared CMPs [9, 18, 19]. In centralised shared CMP, the shared LLC banks are placed in a common contiguous chip area. The growing demands of TCMP makes the research direction shifted from centralised CMP to TCMP. SNUCA is well explored for TCMP as compared to DNUCA.

The performance of Last Level cache (LLC) in TCMP has a major impact on the performance of the entire CMP. It has been observed that the LLC in TCMP has utilizing issues [8, 20]. Such utilisation issues can be divided into two categories:

- local utilisation (within each bank).
- global utilisation (among the banks).

The local utilisation issue means the sets in a bank are not used uniformly; some sets are used heavily while some other remains underused. Such non uniform use of sets reduces the utilisation factor as the heavily used sets face more misses while underused sets remain idle. The loads in the LLC of TCMP are also not uniformly distributed among the banks. Some banks are heavily loaded while some other remains almost idle [22]. Such non-uniform load distribution among the banks are called global utilisation issue.

To increase the local utilisation a technique called Dynamic Associativity Management (DAM) is used. In DAM the associativity of the sets are managed dynamically such that a heavily used set can use the idle ways of another set whenever required. DAM based techniques are local to the bank and the process is transparent from outside. Some promising DAM based techniques are [8, 20, 21, 67, 43]. In this work we proposed three DAM based techniques for better performance and minimizing hardware overheads.

There are some existing works for distributing the loads among multiple banks [12, 55, 18, 1] but most of them are not fit for TCMPs. In this work we proposed DNUCA based TCMP (T-DNUCA) such that blocks can be moved from one bank to another within the bankset. Such flexibility allows to distribute the loads among

all banks (within a bankset) equally. Minimizing the search time of DNUCA is a major area of research [5, 1, 9, 10, 11, 45]. We proposed some simple but efficient techniques to reduce the searching time of DNUCA.

2.5 Some Important Terms for Reference

There are some common terms defined in this chapter which are used in later chapters without any descriptions. The terms are re-defined here for reference and the numbers in bracket gives the section where it is defined/used.

- *LLC* or *L2*: Last Level Cache. The terms L2 and LLC have same meaning for this thesis.
- *bank*: The LLC is divided into multiple independent caches/modules called banks.
- *bankset*: Collection of banks where a block can be mapped. DNUCA divides the banks in multiple banksets. (Section 2.1.2)
- *peer-bank*: All the banks within a bankset are peer-banks (Section 2.1.2).
- *CCMP*: CMP with centralised LLC (Section 2.2).
- *TCMP*: CMP with Tiled LLC (Section 2.2.3).
- *NUCA*: Non-Uniform Cache Access (Section 2.1).
- *SNUCA*: Static NUCA (Section 2.1.1).
- *DNUCA*: Dynamic NUCA (Section 2.1.2).
- *C-SNUCA*: SNUCA design for CCMP (Section 2.2.2).
- *C-DNUCA*: DNUCA design for CCMP (Section 2.2.2).
- *T-SNUCA*: SNUCA design for TCMP (Section 2.2.3).
- *T-DNUCA*: DNUCA design for TCMP (Section 2.2.3).

- *local bank*: In TCMP, a bank is called local bank for a particular core/tile if it is placed in the same tile (Section 2.2.3).
- *remote bank*: For a particular tile in TCMP, all the banks except *local bank* are *remote banks* (Section 2.2.3).
- *bankset-search*: Searching a block in a particular bankset of DNUCA (Section 2.1.2).
- L_1^i: The L1 cache of the i^{th} tile in TCMP.
- L_2^i: The L2 bank of the i^{th} tile in TCMP.
- *DAM*: Dynamic Associativity Management Technique used to increase the associativity of the heavily used set during execution (Section 2.3.1).

Chapter 3

Experimental Methodology

This chapter discuss about the experimental methodologies used in our work. All the experiments mentioned in this thesis are done on full-system simulators. Full-system simulator can simulate an entire electronic system including CMPs. The machine (computer) on which the simulator runs is called the *host machine* and the computational environment created by the simulator is called the *target machine*. It provides virtual hardware, independent of the host machine. A full-system simulator typically includes modules for processing cores, memory systems, I/O devices, NoC etc. The difference of full-system simulator with instruction set simulator is that it allows real device drivers and operating systems to be run on the target machine. It provides a complete virtual machine and any real program can run on it. Since the simulators are nothing but programs, all the modules can be modified to meet the design requirements. For example a full-system simulator providing conventional cache architecture can be modified to support CMP-VR or Z-Cache [20]. The number of cores in the target machine can be changed by just changing a parameter. A brief history of computer architecture simulators and their importance in industrial and academic research is discussed in next section.

3.1 Computer Architecture Simulators

A computer architecture simulator or an architectural simulator is a software designed to model the computer hardware devices to analyze the performance of the modeled computer. An architectural simulator can either model: (a) target microprocessor called instruction set simulator or (b) entire computer system called full system simulator as mentioned above.

CMP systems are complex and highly integrated containing a number of cores, cache memories, controllers, NoCs etc on a single chip. Physical design of such CMPs are highly expensive [82]. Also for an experimental design, different configurations of the same architecture is required to be designed. For example, experimenting the performance of T-DNUCA we have to design it with different cache size and associativity or T-SNUCA with different number of banks and mapping policies. Prototyping each architectures with different configurations is almost impossible in academic research projects [83, 84, 85]. Also the real hardware or prototypes built using field-programmable gate-arrays (FPGA) does not have user friendly debugging environment. The expensive tools to design real hardwares are not always possible to use in academic research where variety of architectures need to be designed for a single research project. For example in this work we need to design T-SNUCA, T-DNUCA, TLD-NUCA, CMP-VR, CMP-SVR, FS-DAM, V-Way [8] etc. All the architectures must have to be designed with different cache sizes and associativity. Therefore computer architects use simulators for experimenting the accuracy and performance of different architectures in a timely and cost-effective manner [82, 86]. Most of the existing works described in Chapter 2 are implemented using such full-system simulators.

SimOS [87, 88] is one of the first machine simulator designed. It simulates the hardware of a machine by using services provided by the underlying operating system. The earliest architectural simulator which become widely popular is SimpleScalar [89] but it had no support for multi core architectures. Soon after SimpleScalar, many architectural simulators have been developed [83, 90, 91]. Since a simulator

has to run on a real machine the performance of simulator depends on the performance of its host machine. As the performance of real hardware increases, the full-system simulators can model the actual systems without sacrificing performance [82]. There are many full-system simulators available for various requirements [82, 85, 84, 92, 93, 94, 95, 96, 97].

Full-system simulator has two categories: functional and timing [85]. Functional simulators mainly focus on what the actual system does. It tries to achieve the same functions as the actual system (for which the simulator is designed) has. Timing simulators takes care of the real-time behavior of the system. It is not just restricted to model what the actual system does but it also models when to do a task. Timing simulator is required to compare the LLC performance in CMPs. For example, average network access latency and the average memory access time cannot be derived without simulating time.

3.1.1 Simics

Simics [82] is a full-system simulator used to generate a complete virtual machine on top of a host machine. It is flexible enough to support variety of tasks like microprocessor design, electronic system design and verification, operating system development etc. Simics supports models for UltraSparc, Alpha, x86, x86-64, PowerPC, IPF (Itanium), MIPS and ARM. It is fast enough to run realistic workloads, including the SPEC CPU2006 benchmark suite [98], Parsec benchmark suite [3], database bench-marks such as TPC-C, interactive desktop applications, and games.

Next generation architecture design is the main focus for most of the architectural research. In our work we proposed some TCMP based architectures which can perform better in future with the increasing workload demands. Full-system simulator like Simics is well suited for the purpose of designing such future hardwares without any physical overheads. Modeling of such future designs is also useful for industry where a new architecture can be quickly designed and verified on simics.

The designing of the real hardware and the supported softwares can be done in parallel as the simulated hardware is available for the softwares.

3.1.1.1 Restrictions on Simics

Simics has powerful capabilities. However its functional behavior restricts one from performing timing simulations that are required to simulate CMP based systems. A timed simulator called GEMS [85] has been proposed to work on top of Simics. It was designed to simulate the complete memory hierarchy of CMP. It manages the coherence protocol and the on-chip communications. The timing nature of GEMS helps to compare the performance of different CMP based architectures. The detail description of GEMS is given in next section. GEMS requires the use of Simics, thus decoupling functional execution from its timing models.

3.1.2 Basic Overview of GEMS

GEMS has three major modules: Ruby, Garnet [99] and Orion [100]. Ruby is used for modeling the entire memory system of any CMP based architecture. Each component like L1 cache, L2 banks, memory banks, directories etc. can be modeled in Ruby. These individual components are called "*machine*" in Ruby. Each machine is given a unique id called *machineID*. The *machineID* is required for the unique identification of a machine during the on-chip communications. All the machines in CMP communicates by the underlying NoC. The NoC in GEMS is managed by Garnet. Any CMP based architecture can be modeled on ruby and connection between its different components (machines) can be designed by Garnet. It simulates the real time scenarios to communicate a message through the NoC. Orion is used for modeling the energy consumed by the simulated system. The Garnet version we used follows X-Y routing. The on-chip communication cost is considered for all the experimental analysis done in this thesis.

The block request (load, store, fetch) from simics are passed to the Ruby module of GEMS. The first level of cache in ruby determines if the block is a hit or a

miss. If it is a hit then simics continues its execution. Otherwise request from the issuing core is stalled and GEMS simulates the cache miss. Ruby determines the timing-dependent functional simulation in Simics by controlling the timing of when Simics advances.

Each L1 cache is attached with a sequencer which is responsible for managing the requests from the corresponding core. The sequencer sends the request to L1. In case of CMP there can be multiple L1s and L2s available. Each of the machine has a controller and all the communications and operations of a machine are performed by the corresponding controller. GEMS provides a domain-specific language called SLICC to model the controllers. The purpose of a controller is to manage all the operations of a machine and also its communication with other *machines*. Maintaining coherence is a major concern in case of CMP based cache structure, hence coherence protocol must be implemented. The controllers of different modules manage the coherence protocol by communicating with each other through message passing. The messages are communicated from one module to another through the NoC (modeled by Garnet). SLICC is also responsible for designing the coherence protocol as it is a combined task of all the controllers.

3.1.2.1 CMP Architectures Supported by GEMS

GEMS only has T-SNUCA (cf. Section 2.2.3.1) implemented on it. The T-SNUCA architecture provided by GEMS is very robust and can be configured with different varieties. For example, cache size, number of banks, number of tiles, cache access latency, miss penalty, hit time, network protocols etc. can be changed by just changing their values in a configuration file. There are many other parameters that can be reset based on the configuration demand like number of virtual networks, block size, cache associativity, replacement policy, network flit size etc. Multiple coherence protocols are also available to support T-SNUCA. One of them is MESI-CMP which is the baseline protocol of our designs. MESI-CMP protocol is based on the MESI protocol proposed in [46]. Different network topology can be designed by changing the network configuration file in Garnet.

Any other architecture behaving different than T-SNUCA has to be implemented by modifying the modules of GEMS. Additional modules may also required to be designed. For example, implementation of T-DNUCA (cf. Section 6.1) in GEMS is not possible by just changing the configuration parameters. A modified architecture may require to change the SLICC based coherence files and other ruby modules like cache memory, replacement policy etc. The mapping and replacement policies as mentioned in Chapter 2 also need to be changed for different designs requirements. The cache memory architecture is also reprogrammed if its new behavior is not like a conventional cache. For example, to implement Z-Cache [20] or CMP-SVR (cf. Section 1.5.2) we need to design our own cache memory in GEMS. For our proposed architectures like T-DNUCA and TLD-NUCA (cf. Section 1.5.4) we have to model our own coherence protocol in SLICC. Rigorous testing has been done to guarantee consistency and liveness. Separate module has to be added for additional hardware support like TGS (in CMP-VR) and TLD (in TLD-NUCA). The modified GEMS requires to re-compile for generating the new architectures.

The operation of entire NoC can be modified according to the design requirements. In this work we consider the existing mesh based NoC (shown in Figure 2.9) as the NoC architecture for all the designs.

3.1.2.2 Result Analysis

Since GEMS is a full-system simulator (together with simics) it can run real workloads on the simulated (modeled) CMP architecture. During the execution it records different information. Some important information that GEMS records during execution are:

- Total Cycle Executed: The cycles executed in all the cores. Cycles executed in each core is also recorded.
- Total instruction executed: The instructions executed in all the cores. Instructions executed in each core is also recorded.

- Total L1 accesses/misses: The L1 accesses of all the L1 caches together. The individual L1 cache accesses are also recorded. The L1 misses are recorded similarly.
- Total L2 accesses/misses: Total number of misses in LLC (L2). The bank-wise distribution is not provided in the original GEMS. Though it can be implemented easily when required. The L2 misses are recorded in a similar manner.
- Average network latency: The average cycles required for each message to communicate through the NoC.
- NoC energy consumption: Total energy consumption of NoC during the execution.

Other important statistics provided by GEMS are: cycle per instructions (CPI), instructions per cycle (IPC), average memory access latency (including on-chip/off-chip communication time), average link utilisation etc. Miss Per Thousand Instructions (MPKI) is also provided by GEMS.

GEMS has a special module called Profiler for managing all the results during execution. The Profiler can be initialized at any time such that the results can be recorded for any specific time period during the execution. The Profiler can also be modified as per user requirements. For example we added a feature to profiler for recording the misses of every set in the banks.

3.1.3 Hardware analysis

For calculating the access time, dynamic energy, leakage power and area consumption of cache memories we used CACTI 6.0 [2]. It is a tool used to model memory related components for better understanding their performance trade-offs. The technology node we consider for CACTI is 32nm and transistor used are ITRS-High Performance (HP).

3.2 Benchmarks

As discussed in the previous section, full-system simulator like GEMS+simics can execute real workloads on the simulated architectures. Different results are collected from these executions based on which the performance of the architecture is analyzed. As CMP architectures are new, the hardware manufacturers or researchers face difficulty to get the testing for the architecture that represent real-world behavior accurately.

The Princeton Application Repository for Shared-Memory Computers (Parsec) [3] is a benchmark suite composed of multi-threaded applications. The applications can be used to evaluate and develop next-generation CMPs. It was collaboratively created by Intel and Princeton University to drive research efforts on future computer systems. Parsec is freely available and is used for both academic and industrial research. Some major objectives of Parsec are:

- Focus on multithreaded applications.
- Different input size for each workload.
- Programs from different real-world problems.

The benchmarks that were publicly available before Parsec are application specific and mostly available in un-parallelized version [3]. The version 2.1 of Parsec Benchmark Suite has 12 workloads and each of the workload is multithreaded and parallelized. The applications are chosen from different real-world areas like finance, media processing, computer vision, enterprise service and animation physics etc. Table 3.1 gives detail description about the Parsec workloads. Multithreaded applications shares/exchanges data among the threads. The data usages details of the benchmarks are given in Table 3.2. Both the tables are taken from [3], which also describe each property in detail. The workloads are also called programs, applications or benchmarks.

Program/ Benchmarks	Application Domain	Parallelisation		Working-Set
		Model	Granularity	
blackscholes	Financial Analysis	data-parallel	coarse	small
bodytrack	Computer Vision	data-parallel	medium	medium
canneal	Engineering	unstructured	fine	unbounded
dedup	Enterprise Storage	pipeline	medium	unbounded
facesim	Animation	data-parallel	coarse	large
ferret	Similarity Search	pipeline	medium	unbounded
fluidanimate	Animation	data-parallel	fine	large
freqmine	Data Mining	data-parallel	medium	unbounded
streamcluster	Data Mining	data-parallel	medium	medium
swaptions	Financial Analysis	data-parallel	coarse	medium
vips	Media Processing	data-parallel	coarse	medium
x264	Media Processing	pipeline	coarse	medium

TABLE 3.1: The inherent key characteristics of Parsec benchmarks. Detail descriptions are given in [3].

Program/ Benchmarks	Data Usage	
	Sharing	Exchange
blackscholes and swaptions	low	low
bodytrack and freqmine	high	medium
canneal, dedup, ferret and x264	high	high
facesim, fluidanimate, streamcluster and vips	low	medium

TABLE 3.2: The data usage behavior of Parsec benchmarks. Detail description is given in [3].

Each application has different size of input sets: small, medium, large, etc. User can run the applications with any input size based on the requirement and architectural demand.

Another popular benchmark suite used for CMP is SPEC-CPU 2006 [98]. One of the main reason for choosing Parsec is that it is freely available. SPLASH-2 [101] was a widely used benchmark suite for CMP architecture in the last decade. But due to its smaller input sizes it cannot be used for the current large sized LLCs.

3.2.1 Benchmark Description of Parsec

This section describes the properties of some Parsec benchmarks used in our work for performance comparison of different CMP based architectures. The detail description about the benchmarks is given in [3].

3.2.1.1 blackscholes

The blackscholes application is used to perform the financial analysis. It is an Intel's recognition, mining and synthesis (RMS) benchmark to analytically calculate the prices for a portfolio of European options with the Black-Scholes partial differential equation (PDE). Blackscholes requires to solve different variety of PDE for their application in financial analysis. The program is divided into multiple concurrent threads where each thread represents a work unit of the portfolio.

3.2.1.2 bodytrack

This benchmark tracks the three dimensional view of human body with multiple cameras. It uses an annealed practice filter to track 3D view using an edge and foreground silhouette. In bodytrack benchmark for an input video which contains many frames, a frame is selected at time stamp t and computes its likelihood. The likelihood is a degree of the 3D body model alignment with the foreground and edges in the images. The value of likelihood is computed by using the two attributes of an image named as the foreground map and the edge distance map.

Bodytrack has a persistent thread pool. The main thread sends the task to the thread pool whenever it reaches a parallel kernel. The main thread has to wait for the working threads to finish their execution before proceeding further.

3.2.1.3 facesim

This benchmark is an Intel RMS application and originally developed by Stanford University. It is an animation based application which takes a human face and a time sequence of muscle activations as input and computes a visually realistic animation of the modeled face. It simulates the underlying physics to get the visually realistic result. Human faces in particular are observed with more attention from users than other details of a virtual world, making their realistic presentation a key element for animations.

3.2.1.4 ferret

This application is used for content based similarity search of rich text data in Internet search engines. Rich text data includes audio, images, video, 3D shapes etc. It uses ferret toolkit [102] for searching. It has six modules. The first and last module are serial while the remaining four modules are parallel. The first module are used as a input and the last module are used as a output.

3.2.1.5 fluidanimate

The main reason for including this workload in Parsec benchmark is due to increasing importance of real time animation and the physical simulations for computer games. It is an Intel RMS application based on Smoothed Particle Hydrodynamics (SPH) method [103]. Fluidanimate uses a five kernel to simulate an incompressible fluid for interactive animation purposes. Fluidanimate generates an output by interpreting and discovering the surface of incompressible fluid.

3.2.1.6 freqmine

The freqmine application is used for Frequent Itemset Mining (FIMI) [104] with an array based version of the Frequent Pattern-growth method. It is an Intel RMS benchmark which was originally developed by Concordia University. FIMI is the basis of Association Rule Mining (ARM) which is a common data mining problem for areas like protein sequences, market data and log analysis etc. It is included in Parsec because of its increasing demand in data mining techniques. It is parallelized with OpenMP and uses three kernels executes in parallel.

3.2.1.7 swaptions

The main reason for including this benchmark is due to the increasing importance of Partial Differential equation(PDE) and the Monte Carlo simulation. It is used to price a portfolio of Swaptions by using the Heath-Jarrow-Morton (HJM)

framework [105]. It is an Intel RMS workload. The behavior of HJM model is non Markovian which prevents it to solve PDE in order to compute the prices. Therefore, Swaptions employs a Monte Carlo simulation. The program stores all the portfolio in the swaptions array. Each entry of the array represent a derivative. Swaption divides the array into the number of block which is equal to the number of thread. It assigns each block to a particular thread. In order to compute a price it iterates through all the swaptions and calls the function HJM_Swaption_Blocking.

3.2.1.8 vips

The application includes fundamental image operations such as transformation and convolution. It is based on the VASARI Image Processing System [106]. The VARSI system is able to construct multi-threaded image processing pipelines transparently on the fly. The image transformation pipeline of the vips benchmark has 18 stages. It is implemented in the VIPS operation im benchmark. All the 18 stages in Vips are implemented in the following kernels:

- **Crop**- This kernel removes the 100 pixels form all the edges.
- **Shrink**- This kernel shrinks the image by 10% by applying the matrix transformation.
- **Adjust white point and shadows**- This kernel brightens the white point and pull down the shadows in order to improve the visual quality of an image.
- **Sharpen**- This kernel enlarges the edges of an output image. It removes the blurring and gives better overall appearance of an output image.

3.2.1.9 x264

It is an H.264/AVC (Advanced Video Coding) video encoder. It includes the new features in encoding such as increased sample bit depth precision, higher-resolution color information, variable block-size motion compensation (VBSMC) or

context-adaptive binary arithmetic coding (CABAC). It allows the H.264 encoders to generate a higher output quality with a lower bit-rate at the expense of a significantly increased encoding and decoding time. It uses motion compensation technique to remove the data redundancy. The application is very flexible and used for different requirements like video conferencing to HD movie distribution. The H.264/AVC encoding is also required for the next-generation HD DVD or Blu-ray video players.

3.3 Experimental Procedure

Different multithreaded and multiprogrammed benchmarks are used for the experimental analysis. The next section describes about the multithreaded and multiprogrammed benchmarks we made from the Parsec benchmarks.

3.3.1 Multithreaded vs. Multiprogrammed Benchmarks

Each Parsec benchmark is a multithreaded application. The number of threads in each benchmarks is based on the input size. In some benchmarks the number of threads can be provided as a command line parameter. In all the benchmarks, the region where the actual multi-threading happens is called "Region Of Interest" (ROI). A magic instruction is inserted in each of the benchmark just before starting the ROI and just after finishing the ROI. The purpose of this magic instruction is to pause the benchmark execution. In simics the magic instruction make the entire simulator to pause its execution. The execution can only resume after manually pressing a "c" command. The purpose of inserting such magic instructions is discussed in the next section. In short the real parallelization of Parsec benchmarks happens in the ROI. The execution required before ROI is for initialization, reading input etc. The execution after the ROI is for output and program termination.

Multiprogrammed benchmarks are designed by combining multiple Parsec applications. Here combining means running together on different cores. For example,

vips and *ferret* can make a multiprogrammed benchmark where *vips* can be binded on first 8 cores and *ferret* can be binded on the next 8 cores (considering a 16 core CMP). Note that each Parsec application is a multithreaded application, hence even in multiprogrammed environment each application has multiple threads and they share the cores provided to the application. In case of multiprogrammed benchmarks the term benchmark means the combined workload and each individual multithreaded applications associated with the benchmark are called applications.

3.3.2 The Benchmarks Used in Our Experiments

Based on the multithreaded benchmarks provided by Parsec different benchmark combinations can be made. The benchmark combinations we used for experiments are either a single Parsec benchmark or a combination of more than one benchmark. Table 3.3 gives the details of the benchmarks we used for our experiments. Note that not all benchmarks mentioned in the table are used for experimenting all the architectures.

3.3.3 Benchmark Running Process

To execute a multi-threaded benchmark we run each benchmark on the *target machine* up to the starting of the ROI. Once the ROI is reached the benchmark automatically stops executing because of the magic instruction inserted there. Reaching ROI means the initialization of the benchmark is done and all the threads are created. After creating all the threads the benchmark is run for 50 million cycles for warm-up. The warm-up is necessary to avoid the compulsory misses in the cache and also allowing the NoC architecture to settle properly. After the first 50 million cycles the Ruby profiler is cleared and actual execution begins. Few benchmarks are executed up to its termination (i.e. the complete ROI) while the other benchmarks are execute up to a fixed number of cycles. Note that the number of cycles executed varies from benchmark to benchmark but is never less

Multithreaded benchmarks	
blackscholes (*blck*), bodytrack (*body*), facesim (*face*) ferret (*frrt*), fluidanimate (*flud*), freqmine (*frq*) swaptions (*swp*), *vips* and *x264*	
Multiprogrammed benchmarks	
Benchmark	**Details**
mx1 or *mix1*	Combination of *x264*, *swp*, *frq* and *flud*.
mx2 or *mix2*	Combination of *face* and *raytrace*.
mx3 or *mix3*	Combination of *vips* and *body*.
frt16	16 copies of *frrt*.
bd16	16 copies of *body*.
bdy2	2 copies of *body*.
vp4	4 copies of *vips*.
blk4	4 copies of *blackscholes*.
frq16	16 copies of *frq*.

TABLE 3.3: List of all the multithreaded and multiprogrammed benchmarks used for the experiments in this thesis. The multiprogrammed benchmarks are made of combining multiple multithreaded Parsec benchmarks. The details are discussed in section 3.3.1.

than 1 billion (unless the case of termination). The execution process fixed for a benchmark is same for all the architectures being compared. For example the benchmark *vips* runs for 1300 million cycles for all the designs like CMP-VR, V-Way, T-SNUCA, T-DNUCA etc.

To run a multiprogrammed benchmark the first step is to load all the applications (belonging to the benchmark) one by one. Each application is initially executed until its ROI and then binded with some cores. Note that each program may have multiple threads and the threads can also be binded with the cores assigned to each application. After binding an application to some cores the execution of that application remains paused. The binding process is repeated with the other applications. Once all the applications are attached to some cores, all of them are activated simultaneously. Activating the applications means they will start executing from their corresponding ROI. The warm-up and running policy is same as multithreaded benchmarks.

3.3.4 Comparing Two TCMP Architectures

For analyzing the performance of each proposed TCMP architecture we compare its performance with other existing TCMP architectures in terms of CPI, IPC, MPKI, Energy Consumption etc. To do this we designed all the TCMP architectures on simics+GEMS (full-system simulator) and then executed Parsec benchmarks on top of them. Different statistics are recorded during the execution of each benchmarks as discussed in Section 3.1.2.2. Based on this statistics the performance of two architectures is compared.

An architecture is designed with different configurations. For example with different cache size, associativity, block sharing capability etc. Details are discussed later whenever required. For all the architectures having different configurations, the process of running a particular benchmark is the same. The results for the benchmarks are reported individually and the geometric mean (average) of all the benchmarks is also given.

3.3.5 Cache Configurations

The entire thesis considers SNUCA based TCMP (T-SNUCA) as the baseline architecture. The architecture is already implemented in GEMS for different coherence protocols, cache sizes and replacement policies. The baseline architecture has 16 tiles as shown in Figure 3.1. MESI-CMP is used as coherence protocol for the L1 caches. Different cache size, associativity, replacement policy are used for the experimental versatility. The terms T-SNUCA or baseline are used interchangeably to represent the same design. The detailed configurations are mentioned separately whenever required in the later chapters.

For the entire thesis the term T-SNUCA, baseline TCMP or baseline represent a 16 core SNUCA based TCMP as shown in Figure 3.1.

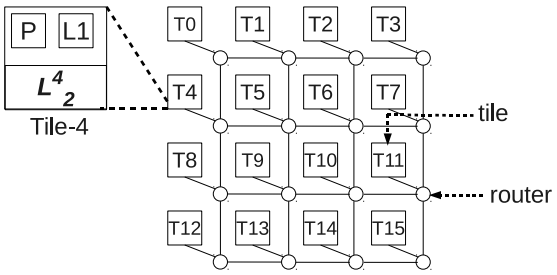


FIGURE 3.1: The baseline T-SNUCA.

For most of our experiments we used the LLC size as 2MB, 4MB and 8MB. More larger cache can be used but in that case the size of input provided by the benchmarks may not be sufficient. The size of the LLC is equally divided into all the banks. For example in our baseline design a 4MB LLC (L2) means each bank is of size 256KB (total 16 banks). The associativity of LLC means the associativity of each bank.

Chapter 4

Improving Local Utilisation by Victim Retention

As discussed in Chapter 2 the LLC banks of TCMP are not utilised properly. The memory accesses within a bank are distributed non-uniformly among the sets. Such non uniform distribution makes some set to be used heavily while some other remain idle. The problem is defined as local utilisation problem in Chapter 2. The Dynamic Associativity Management (DAM) technique used to improve local utilisation is also discussed in Chapter 2. DAM based techniques allow us to dynamically manage the associativity of the bank for better utilisations. This Chapter discusses our first proposed DAM based technique, called CMP-VR.

4.1 Introduction

Chip Multiprocessor with Victim Retention (CMP-VR) is our first DAM based policy proposed for local utilisation enhancement. The basic description about CMP-VR is already given in Chapter 1. The related works in the area of DAM are discussed in Chapter 2. Hence instead of discussing them again we start this chapter with a motivational analysis regarding why local utilisation enhancement is necessary.

All the DAM based policies are applicable to each LLC bank independently. The baseline architecture is considered as a 16 core TCMP or T-SNUCA (used interchangeably) as discussed in Section 3.3.5. CMP-VR is applied to each bank separately and the process is transparent from outside the bank. Other inter-bank data management policies like co-operative caching [12] and victim-replication [7] can be implemented on top of it.

The rest of the chapter is organized as follows. The next section gives a motivational analysis regarding why enhancement of local utilisation is required. Section 4.3 gives the detail description about CMP-VR. The experimental analysis of CMP-VR is given in Section 4.4. Section 4.5 gives the summary of this chapter.

4.2 Motivation

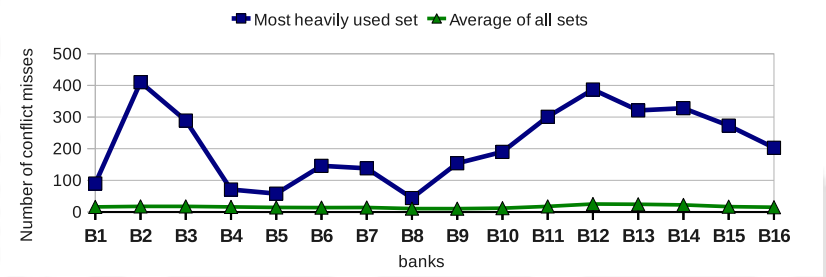


FIGURE 4.1: The number of misses in the most heavily used set as compared to the average misses of all sets (considering the statistics of six Parsec benchmarks [3] together).

In [8], the authors mentioned about the non-uniform behavior of cache sets. We have done an experiment for a TCMP having 4MB LLC to observe the usages of sets. The size of each bank is 256KB and they are 4-way set associative. Six Parsec [3] benchmarks are used for the simulation. Figure 4.1 shows the result of the experiment. The benchmarks are run for either termination or 1 billion cycles, whichever comes earlier. The details about the benchmark running process is discussed in section 3.3.3. It shows the number of misses in the most heavily used set as compared to the average misses of all the sets. The result of each bank is given separately ($B1, \ldots, B16$ through X-axis). For each bank, the values

Banks		Benchmarks					
		flud	**frrt**	**frq**	**vips**	**swp**	**x264**
B1	Max (%):	247.34	344.68	854.25	234.02	1094.69	580.11
	Min (%):	-21.89	-38.72	-72.44	-41.50	-68.72	-100.00
B2	Max (%):	155.43	213.88	5552.28	192.06	284.18	290.91
	Min (%):	-17.71	-30.16	-95.66	-35.98	-52.64	-100.00
B3	Max (%):	277.87	223.74	1225.10	282.93	339.75	545.76
	Min (%):	-19.33	-28.40	-61.85	-41.46	-66.03	-100.00
B4	Max (%):	177.38	226.33	504.19	186.31	386.98	291.09
	Min (%):	-13.58	-26.20	-50.29	-37.90	-70.23	-100.00
B5	Max (%):	188.35	201.85	261.49	178.27	231.41	384.52
	Min (%):	-17.31	-19.26	-65.13	-38.53	-42.15	-100.00
B6	Max (%):	147.72	183.99	431.93	392.87	240.60	340.30
	Min (%):	-21.22	-16.37	-100.00	-48.87	-51.88	-100.00
B7	Max (%):	159.71	152.17	356.95	349.30	402.36	509.09
	Min (%):	-17.57	-18.48	-55.38	-46.85	-100.00	-100.00
B8	Max (%):	143.16	175.63	261.89	189.12	595.35	632.10
	Min (%):	-8.38	-9.35	-47.62	-52.72	-100.00	-100.00
B9	Max (%):	791.25	394.30	676.23	253.36	783.17	1169.89
	Min (%):	-17.79	-13.58	-51.70	-35.31	-100.00	-100.00
B10	Max (%):	782.12	302.22	993.84	394.88	1189.49	571.36
	Min (%):	-34.14	-13.65	-65.73	-48.90	-100.00	-100.00
B11	Max (%):	576.64	219.29	568.89	181.68	353.55	483.29
	Min (%):	-63.60	-20.26	-100.00	-42.19	-67.86	-100.00
B12	Max (%):	412.54	293.14	520.29	201.90	426.07	616.87
	Min (%):	-77.61	-27.84	-80.73	-49.53	-64.49	-100.00
B13	Max (%):	363.96	260.23	727.46	279.86	359.18	805.81
	Min (%):	-76.19	-19.93	-78.60	-41.98	-67.35	-100.00
B14	Max (%):	451.95	247.87	650.85	238.91	269.79	525.75
	Min (%):	-71.06	-20.68	-100.00	-43.96	-70.02	-100.00
B15	Max (%):	660.16	343.86	1164.96	316.07	1059.95	384.79
	Min (%):	-51.64	-19.09	-100.00	-41.12	-53.92	-100.00
B16	Max (%):	716.55	248.59	1489.82	190.16	280.04	698.18
	Min (%):	-36.46	-27.14	-100.00	-32.52	-53.33	-100.00

TABLE 4.1: Percentage difference in misses with respect to the average misses of all the sets in a bank. A value of $x\%$ implies increase by $x\%$ over the average case and a value of $-x\%$ implies reduction by $x\%$ over the average case. Maximum values are for the most heavily used set within the bank and the minimum values are for the most underused set within the bank. (Result shown for six benchmarks and for all the 16 banks.)

are calculated considering the statistics of all the six benchmarks together. In each bank, the heavily used set has far more number of misses than the average misses of all the sets within the bank. A more detailed result is shown in Table 4.1. It shows the comparison of misses for the most heavily used set and the most underused set with the average misses of all the sets. The results are given for each benchmark and for each bank. For example, during the execution of vips, L_2^2 (i.e. $B2$ in figure) has a set whose misses are 192.06% more than the average misses of all the sets. Also there is a set which has 41.50% less (-41.50% more) misses than the average misses.

Figure 4.2 shows the set-wise distribution of misses in the most heavily used bank. The result is shown for only one Parsec benchmark (x264). Higher misses in a set imply that the set is heavily used, while lesser misses in a set indicate an underused or idle set.

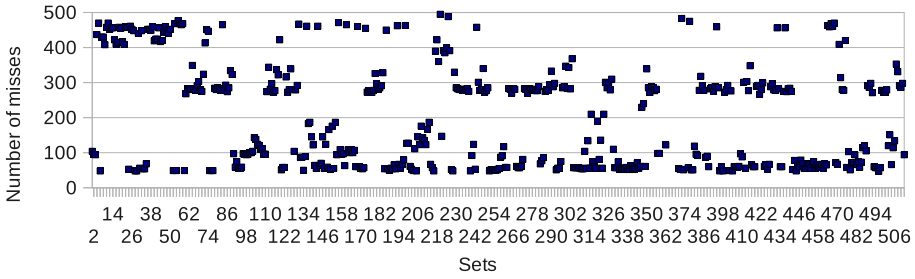


FIGURE 4.2: Set wise distribution of misses for the most heavily used bank.

From the above analysis it is clear that the usage of sets in the banks is not uniform. A traditional set associative cache has one-to-one mapping between its tag entries and data lines which makes searching simpler and less expensive. But it cannot dynamically change the associativity of any set according to its run-time requirements. Both [8] and [20] can increase the associativity dynamically but they removed the traditional one-to-one mapping between the tag and data blocks completely. In [8] this mapping is maintained by a pair of forward and backward pointers whereas in [20], it is maintained by a separate hash function for each way in the set. In CMP-VR, the traditional mapping remains same for $(100 - x)\%$ of ways, where x is the percentage of ways reserved from each set. Comparing with the other similar implementations [8, 20], the hit in reserve storage may

be a bit more expensive but the hit in regular storage takes less time. Also the maximum dynamic associativity increase of heavily used sets is more in CMP-VR than V-way [8].

4.3 CMP-VR

The objective of CMP-VR is to: (a) retain the evicted cache blocks on the chip for the longest possible time and (b) increase the bank utilisation. The first objective can also be achieved by using a victim cache, however the separate victim cache adds to the area overhead. Also using separate victim cache the second objective cannot be achieved. The banks remains locally underutilised and some sets face more number of misses while some others remain idle. Allowing the heavily used sets to use the idle ways of underused set can achieve both the objectives mentioned above. Hence techniques like CMP-VR are beneficial as compared to victim caches.

4.3.1 Cache Architecture

Each cache-bank is divided into two parts: regular storage (NT) and reserve storage (RT). This is done by dividing every set into two parts. RT takes significantly less space (25%-50% of the total cache space). NT behaves as conventional cache while RT is used as reserve storage. The RT section of each set can be used by all the sets in the cache. In particular if a set needs extra space it can store its new block in the RT section of other ways. Whenever a block is evicted from the NT, it is moved to the MRU position of RT. When a cache miss occurs on a set, the block is either in the RT or needs to be fetched from the main memory. To make searching in RT faster, a separate tag-array (TGS) is maintained. Each entry in TGS stores the tag address of a block currently stored in RT. Finding a tag in TGS means the block is currently in RT. There is a one-to-one mapping between the TGS locations and RT locations, i.e. each location in TGS maps to a corresponding location in RT. Hence if a tag is found in TGS then based on its

location (array index) the corresponding RT location can be directly calculated. The number of entries in TGS is $S \times R$, where S is the number of sets and R is the number of reserve ways in each set.

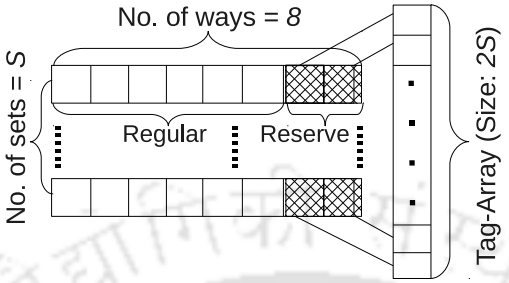


FIGURE 4.3: Example of a bank in CMP-VR.

Figure 4.3 demonstrates the structure of an LLC bank and its associated TGS. In order to facilitate block search and replacement in RT, the TGS and RT mapping needs to be maintained consistent. An LRU list is maintained for the TGS to get the LRU entry to be replaced with newer ones. For example, if the index i in TGS is the LRU entry, then the corresponding block in RT will be replaced by the new block, evicted from NT. In other words, a global LRU replacement policy is maintained for RT.

4.3.2 Operations of CMP-VR

All the cache-hits within the NT are treated as in regular bank, without any changes. The modifications are only required for the misses in NT. The necessary changes are discussed below:

1. *On NT miss (block not in NT):* Search the TGS. Upon a tag match in TGS identify the corresponding block in RT and move it to NT. If tag match fails in TGS, it is a cache-miss.

2. *Move a block B from RT to NT:* If free space (way) is available in the desired set of NT, then move B in that free space. Otherwise, swap B with the LRU block in the corresponding set of (NT).

3. *On Cache Miss:* Bring block from main memory and store it in the desired set of NT.

4. *Store a newly arrived block B into NT:* If free space (way) is available in the desired set of NT then place B into the free space. Otherwise make space in NT by evicting its LRU block. This evicted block moves to RT.

5. *Store block V (evicted from RT) into RT :* Search the TGS. If space available in RT then move V into the free space of RT. Otherwise select a victim block (V') from RT (based on LRU) and replace it with V. The new victim V' is evicted from the cache bank and written back to the main memory (if required). Update TGS entry by overwriting the tag of V' by the tag of V.

Coherence action of invalidation for shared blocks in L1 is initiated only if the block is evicted from RT (i.e. from the chip). In other words, the RT is transparent to the L1s, sharing blocks residing in RT. Also the usage of NT in one bank is completely transparent to the other banks and hence other inter-banking data management policies can be implemented on top of it.

4.3.3 Additional TaG Storage (TGS)

The above description gives an idea of what CMP-VR is and how it works. As mentioned above, CMP-VR has an extra tag-array called TGS, which is searched on every cache miss in NT. The TGS access time overhead should not hamper the performance gain of CMP-VR. The details regarding access time of TGS as well as that of CMP-VR are discussed below.

4.3.3.1 Structure of the TGS

The structure of TGS is similar to a fully associative TLB (translation lookaside buffer). For searching TGS, all the entries in it are compared simultaneously. Comparing all the tag entries simultaneously needs a lot of comparators and may

lead to energy overhead. To reduce the overheads of TGS it is divided into multiple segments of 16-ways. All the segments are searched simultaneously, hence the total search time of TGS is the access time of each segment. The detailed analysis about the energy overhead of CMP-VR is given in Section 4.4.4.2.

4.3.3.2 Access time of CMP-VR

This section discusses about the time to access a block within the bank, after the request reaches the bank. The request may come from a separate tile and according to TCMP architecture the routing time is different based on the distance between the home tile and the requesting tile. But the routing concepts of CMP-VR and the baseline design are same. Hence, NoC routing time of CMP-VR has not been discussed in this section. The actual cache access time is *NoC routing time+bank access time*.

In the case of standard cache architecture [2], for each cache bank, the address of the requested block is provided as input to a decoder, which then activates a wordline in both the data array as well as the tag array (not to be confused with TGS). The content of the entire set are placed on the bitlines, which are then sensed. The multiple tags read out of the tag array are compared with the input address to check for a match. The multiplexers are then appropriately selected to forward the data to the requesting core. The logical organization of a cache bank is shown in Figure 4.4. The bank access time in such an architecture depends on both tag array and data array and it is termed as the normal bank access time ($nbat$).

The tag array and TGS are not the same entity. The former is a part of normal cache architecture as mentioned in [2] while the latter is the proposed additional tag storage required for RT. Figure 4.5 explains the flow of a block (say B) request in CMP-VR. The request is done by an L1 cache (say L_1^x) when it suffers a cache miss. Considering the static bank mapping policy of baseline TCMP the block maps to a fixed L2 bank (say L_2^y). The details of address decoding, tag comparison and multiplexing are abstracted out. The routing details about how the request

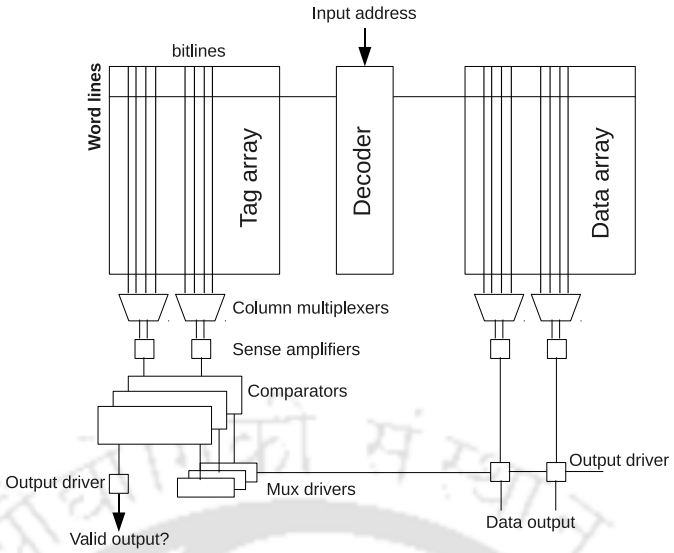


FIGURE 4.4: A logical organization of cache bank [2].

reaches L_2^y from the requesting L_1^x are also ignored. The flow only shows what happens when the request arrives at L_2^y. The CMP-VR is applied to each bank separately.

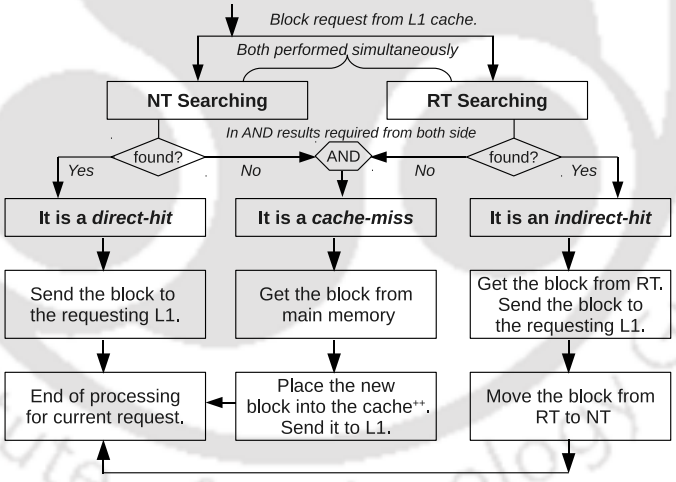


FIGURE 4.5: Request flow in CMP-VR after the request reaches the home tile. ++Placing a block in cache memory may need to move another block to RT. The block movement and replacement procedures are discussed in Section 4.3.2.

When a request arrives, all the three components i.e. tag array, data array and TGS are searched simultaneously. Tag array and data array are part of normal cache (in CMP-VR, it means NT) and in the figure it mentioned as **RT searching**.

Components	Parameters
No. of tiles	16
Processor	UltraSPARCIII+
L1 I/D cache	64KB, 4-way
LLC size	2MB and 4MB
bank size	128KB 4-way and 256KB 4-way
Memory	1GB, 4KB/page

TABLE 4.2: System Parameters.

The TGS access time in worst case is considered to be equal to *nbat* [†]. If the block is found in NT then it is a *direct-hit* and the hit time is *NoC routing time*+*nbat*. Since TGS takes *nbat* as worst case access time, it finishes searching in parallel with NT searching. Now if the tag is not found in TGS then it is a cache miss and the request has to go to the main memory. However, if the tag is found in TGS, then it is an *indirect-hit* and the block needs to be read from the data array; this in worst case takes another *nbat* amount of time. So in the case of an indirect-hit, the cache access time (including both RT and NT) is *NoC routing time*$+2 \times$*nbat*.

4.4 Experimental Setup

In order to evaluate CMP-VR, simulations were performed by running benchmarks on a multi-core simulator GEMS [85], with the help of SIMICS [82] a full-system functional simulator. The detail description of experimental procedures are discussed in Chapter 3. The configuration details of the processor, cache memory and main memory used in the experiments are given in Table 4.2.

Eight Parsec benchmarks are used for this experiment: **flud**, **body**, **frrt**, **vp4**, **swp**, **face**, **x264** and **blck**. The details about the benchmarks are mentioned in Section 3.3.2. MESI-CMP based cache controller has been used in GEMS. For implementing CMP-VR, the cache controller part of GEMS needs to be modified. Note that, the behavior of the L1 caches remains unchanged.

[†]The access time (based on CACTI) of each segment is less than the bank access time. For example in case of a 256KB, 4-way associative bank, the access time is 0.67ns while the access time of each TGS segment is less than 0.2ns. Hence the simultaneous search of both NT and TGS can be finished in *nbat* time.

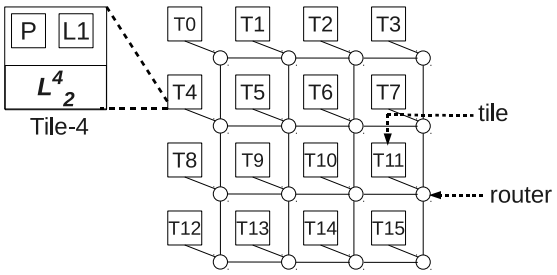


FIGURE 4.6: The baseline TCMP or T-SNUCA.

4.4.1 Simulation Results and Analysis

CMP-VR is designed on top of the baseline TCMP (T-SNUCA) as shown in Figure 4.6. Note that the figure is exactly same as the Figure 3.1. It is shown here again for completeness. The concept of CMP-VR is applied to each bank of the baseline TCMP. The size of LLC in CMP-VR is equal to the size of LLC in baseline.

The number of L2 misses in CMP-VR and in the baseline can be calculated as follows:

- Total L2 misses in CMP-VR=
  Total L1 miss - (Total NT hit + Total RT hit).
- Total L2 misses in baseline = Total L1 miss - Total L2 hit.

The term **xMyW** represents a baseline TCMP having LLC of size xMB. The LLC has 16 equal sized banks. The associativity of each bank is y. Also the term **xMyW-zR** represents a TCMP having LLC of size xMB with sixteen y-way set associative banks and in each bank, z% of y ways ($z/100 * y$) from each set are reserved for RT. For example, **4M4W-25R** represents an LLC of size 4MB and having sixteen equal sized banks. Each bank is 4-way set associative and 25% ways (i.e. 0.25*4=1) from each set are reserved for RT.

The performance of CMP-VR is analyzed with different associativity and reserve storage areas. The complete list is given below:

2M-4W: This category has a 2MB 4-way set-associative LLC as baseline (2M4W) to compare with the following CMP-VR configurations.

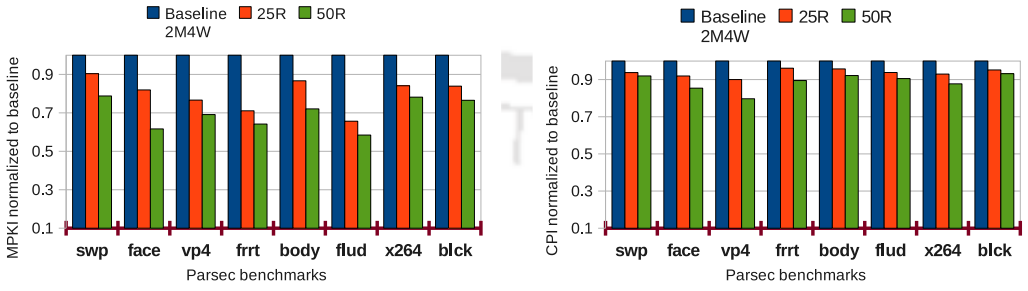
- 2M4W-25R: 2MB 4-way set associative LLC having 25% ways (1 way) from each set as reserve.
- 2M4W-50R: 2MB 4-way set associative LLC having 50% ways (2 ways) from each set as reserve.

4M-4W: This category has a 4MB 4-way set-associative LLC as baseline (4M4W), to compare with the following CMP-VR configurations.

- 4M4W-25R: 4MB 4-way set associative LLC having 25% ways (1 way) from each set as reserve.
- 4M4W-50R: 4MB 4-way set associative LLC having 50% ways (2 ways) from each set as reserve.

4.4.1.1 2M-4W

The size of each L2 bank in this configuration is 128KB and the total L2 cache size is 2MB (128KB*16). Figure 4.7 shows the performance comparison of CMP-VR with the baseline design. The result of CMP-VR is given for two different configurations: 2M4W-25R and 2M4W-50R. Each graph in the figure shows the results of different performance metrics normalized to the baseline design value.



(a) MPKI. (b) CPI.

FIGURE 4.7: Normalized performance comparison of CMP-VR (2M4W-25R and 2M4W-50R) with baseline (2M4W) design.

	CPI			MPKI		
Benchmarks	**25R-O-B**	**50R-O-B**	**50R-O-25R**	**25R-O-B**	**50R-O-B**	**50R-O-25R**
swp	6.25	8.04	1.91	9.61	21.21	12.84
face	8.07	14.62	7.13	18.08	38.37	24.76
vp4	10.00	20.31	11.46	23.36	30.89	9.82
frrt	3.86	10.51	6.92	28.92	35.83	9.73
body	4.27	7.83	3.72	13.33	27.94	16.85
flud	6.18	9.46	3.50	34.34	41.55	10.98
x264	7.05	12.29	5.64	15.91	21.83	7.04
blck	4.82	6.82	2.11	16.10	23.46	8.77
Average	**6.33**	**11.33**	**5.34**	**20.35**	**30.52**	**12.77**

TABLE 4.3: Performance improvement (in %) chart for 2M-4W. 25R-O-B means percentage of improvement in 2M4W-25R over baseline (2M4W); 50R-O-B means percentage of improvement in 2M4W-50R over baseline (2M4W); and 50R-O-25R means percentage of improvement in 2M4W-50R over 2M4W-25R.

Figure 4.7(a) shows that 2M4W-25R gets 9.61% to 34.34% reduction in MPKI with an average of 20.35%. In other words, on average, 20.35% times the blocks are found in RT, resulting in hit, whereas they would be termed as misses in the baseline design. Similarly, 2M4W-50R gets 21.21% to 41.5% reduction is MPKI with an average of 30.52%. Figure 4.7(b) shows the performance comparison in terms of CPI. It shows that in the case of 2M4W-25R, CPI improves by 3.86% to 10% with an average improvement of 6.33% and in case of 2M4W-50R the improvement is 6.82% to 20.31% with an average of 11.33%.

Table 4.3 shows the performance improvement in terms of both MPKI and CPI for each benchmark. The performance of both proposed configurations (2M4W-25R and 2M4W-50R) are compared with the baseline (2M4W) as well as they are compared between themselves.

In the results shown, it can be observed that for some benchmarks there is a large improvement in the MPKI, however the CPI improved is not as expected. This is because of the more number of indirect-hits, which takes twice the time taken by direct-hits. Figure 4.8 shows the distribution of direct-hits and indirect-hits in each benchmark. Though the CPI is not improving proportionally (with the improving MPKIs) but there is performance gain over baseline.

Comparison with higher associativity baseline: Increasing the associativity of baseline (keeping the cache size same) improves the performance because of the less number of conflict misses. However, CMP-VR performs even better than the

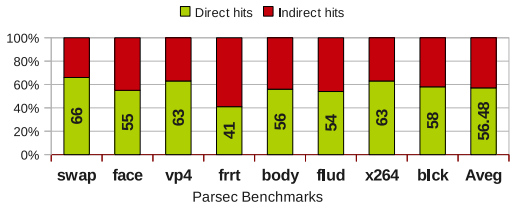
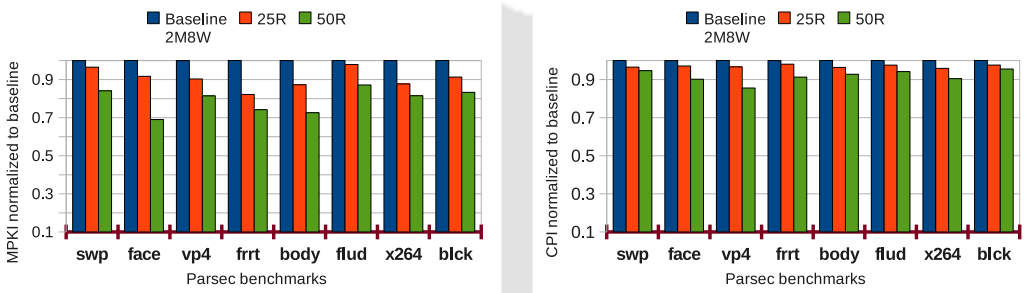


FIGURE 4.8: Distribution of direct-hits and indirect-hits in CMP-VR.

–	25R-O-B	50R-O-B	25R-O-2B	50R-O-2B
Average CPI improvement(%)	6.33	11.33	3.02	8.20
Average MPKI improvement(%)	20.35	30.52	9.50	21.06

TABLE 4.4: Average performance improvement (in %) of CMP-VR, considering all the eight benchmarks. 25R-O-B means improvement of 2M4W-25R over the first baseline (2M4W) and 25R-O-2B means improvement of 2M4W-25R over the second baseline (2M8W). 50R-O-B and 50R-O-2B also have similar meaning.

baseline with double the associativity (2M8W) compared to the original baseline (2M4W).



(a) MPKI.

(b) CPI.

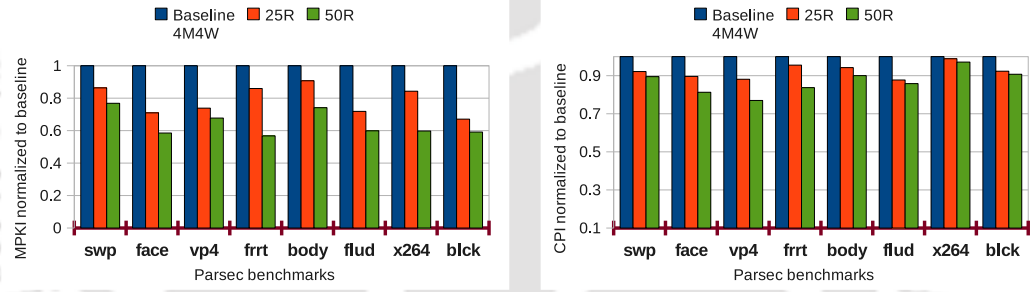
FIGURE 4.9: Normalized performance comparison of CMP-VR (2M4W-25R and 2M4W-50R) with baseline (2M8W) having higher associativity.

The performance of both 2M4W-25R and 2M4W-50R is compared with an 8-way set associative baseline cache (2M8W). Figure 4.9(a) shows this comparison in terms of MPKI. It shows that, in the case of 2M4W-25R, the MPKI improved by 2.06% to 17.80% with an average of 9.5%. On the other hand, the MPKI of 2M4W-50R improves by 12.82% to 27.40% with an average of 21.06%. Figure 4.9(b) shows the improvement in terms of CPI. It shows that 2M4W-25R improves CPI by 1.91% to 4.12% with an average of 3.02% and 2M4W-50R improves CPI by 4.45% to 14.42% with an average of 8.20%.

Table 4.4 shows the average improvement of both 2M4W-25R and 2M4W-50R over the two baseline design (2M4W and 2M8W). From the analysis, it is clear that CMP-VR having 25% reserve storage (25R) improves performance of the baseline design but a CMP-VR with 50% reserve storage (50R) improves performance even more.

4.4.1.2 4M-4W

The bank size in this configuration is 256KB and the total L2 cache size is 4MB (256KB*16). Figure 4.10 shows the performance comparison of CMP-VR with the baseline design. The two different configurations of CMP-VR used in this section are: 4M4W-25R and 4M4W-50R.



(a) MPKI. (b) CPI.

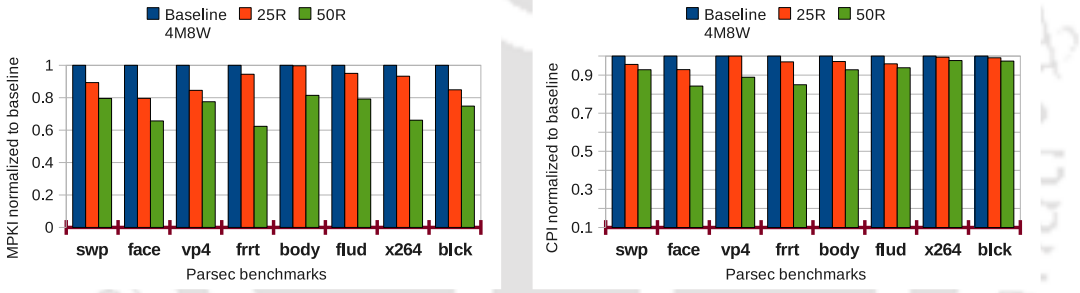
FIGURE 4.10: Normalized performance comparison of CMP-VR (4M4W-25R and 4M4W-50R) with baseline (4M4W) design.

Figure 4.10(a) shows that 4M4W-25R gets 9.28% to 32.92% reduction in MPKI with an average of 21.55% and the same for 4M4W-50R is 23.67% to 43.25% with an average of 36.29%. The performance comparison in terms of CPI is shown in Figure 4.10(b). Here, 4M4W-25R improves CPI by 1.12% to 12.29% with an average of 7.75% and in the case of 4M4W-50R the improvement is 2.88% to 23.03% with an average of 13.31%. Table 4.5 shows the performance improvement in terms of both MPKI and CPI for each benchmark.

	CPI			MPKI		
Benchmarks	25R-O-B	50R-O-B	50R-O-25R	25R-O-B	50R-O-B	50R-O-25R
swp	7.84	10.54	2.93	13.67	23.14	10.97
face	10.39	18.71	9.28	29.04	41.46	17.50
vp4	11.88	23.03	12.65	26.13	32.26	8.30
frrt	4.46	16.29	12.38	14.03	43.25	33.98
body	5.79	9.99	4.46	9.28	25.88	18.30
flud	12.29	14.18	2.15	28.14	40.12	16.67
x264	1.12	2.88	1.78	15.71	40.24	29.10
blck	7.67	9.27	1.73	32.92	40.86	11.83
Average	**7.75**	**13.31**	**6.02**	**21.55**	**36.29**	**18.78**

TABLE 4.5: Performance improvement (in %) chart for 4M-4W. 25R-O-B means percentage of improvement in 4M4W-25R over 4M4W (baseline); 50R-O-B means percentage of improvement in 4M4W-50R over 4M4W; and 50R-O-25R means percentage of improvement in 4M4W-50R over 4M4W-25R.

Comparison with higher associativity baseline: Comparison of 4M4W-25R and 4M4W-50R with higher associativity baseline (4M8W) is shown in Figure 4.11.



(a) MPKI. (b) CPI.

FIGURE 4.11: Normalized performance comparison of CMP-VR (4M4W-25R and 4M4W-50R) with baseline (4M8W) having higher associativity.

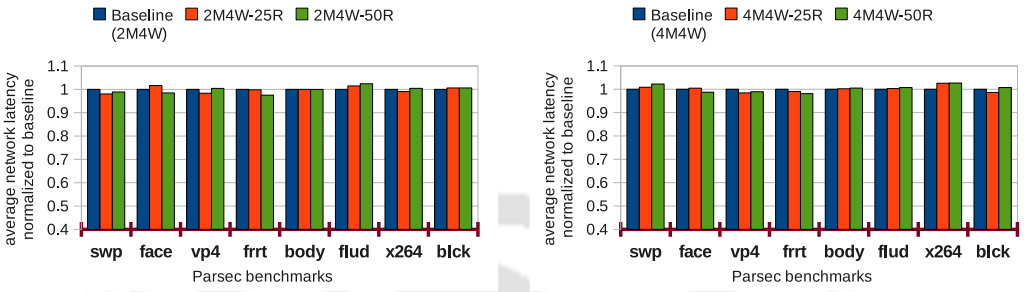
–	25R-O-B	50R-O-B	25R-O-2B	50R-O-2B
Average CPI improvement(%)	7.75	13.31	2.69	8.55
Average MPKI improvement(%)	21.55	36.29	10.11	27.00

TABLE 4.6: Average performance improvement (in %) of CMP-VR, considering all the eight benchmarks. 25R-O-B means improvement of 4M4W-25R over the first baseline (4M4W) and 25R-O-2B means improvement of 4M4W-25R over the second baseline (4M8W). 50-R-O-B and 50R-O-2B have similar meaning.

Figure 4.11(a) shows that in case of 4M4W-25R, the average MPKI improvement is 10.11% while the same for 4M4W-50R is 27%. Figure 4.11(b) shows that 4M4W-25R improves CPI by 2.69% and 4M4W-50R improves CPI by 10.11%.

Table 4.6 shows the average improvement of both 4M4W-25R and 4M4W-50R over the two baseline designs: 4M4W and 4M8W.

4.4.2 Communication Cost in CMP-VR



(a) 2M-4W. (b) 4M-4W.

FIGURE 4.12: Average network latency comparison of CMP-VR with the baseline.

As mentioned in Section 4.3.3.2, the structure and configuration of the on-chip network in CMP-VR is exactly same as the baseline. Though the bank access time in CMP-VR varies (double in the case of an indirect-hit) from baseline it should not affect the on-chip communication time. Figure 4.12 shows the comparison of CMP-VR with the baseline in terms of average network latency. Figure 4.12(a) shows the latency comparison for 2M-4W and Figure 4.12(b) shows comparison for 4M-4W. For either configuration (2M-4W or 4M-4W) the corresponding figure shows that the communication cost of the baseline as well as that of CMP-VR (both 25R and 50R) are almost same. In the case of 2M-4W, 2M4W-25R has -2.0% to 1.63% increase in average network latency as compared to baseline while the same for 2M4W-50R varies from -2.53% to 2.34%. Such variations in latency falls within the margin of error of the simulator and are thus considered to be the same. Note that the communication time calculated for this analysis is only for the L1s requests to L2. Since the Tiled CMP has shared LLC (L2), the blocks in L2 are distributed throughout the chip and the L1 requests need to travel through the NoC to reach the corresponding bank of L2. The average communication latency

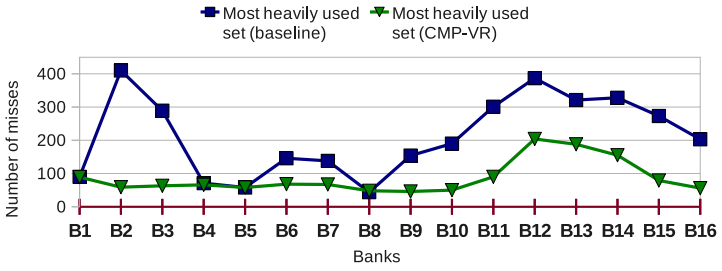


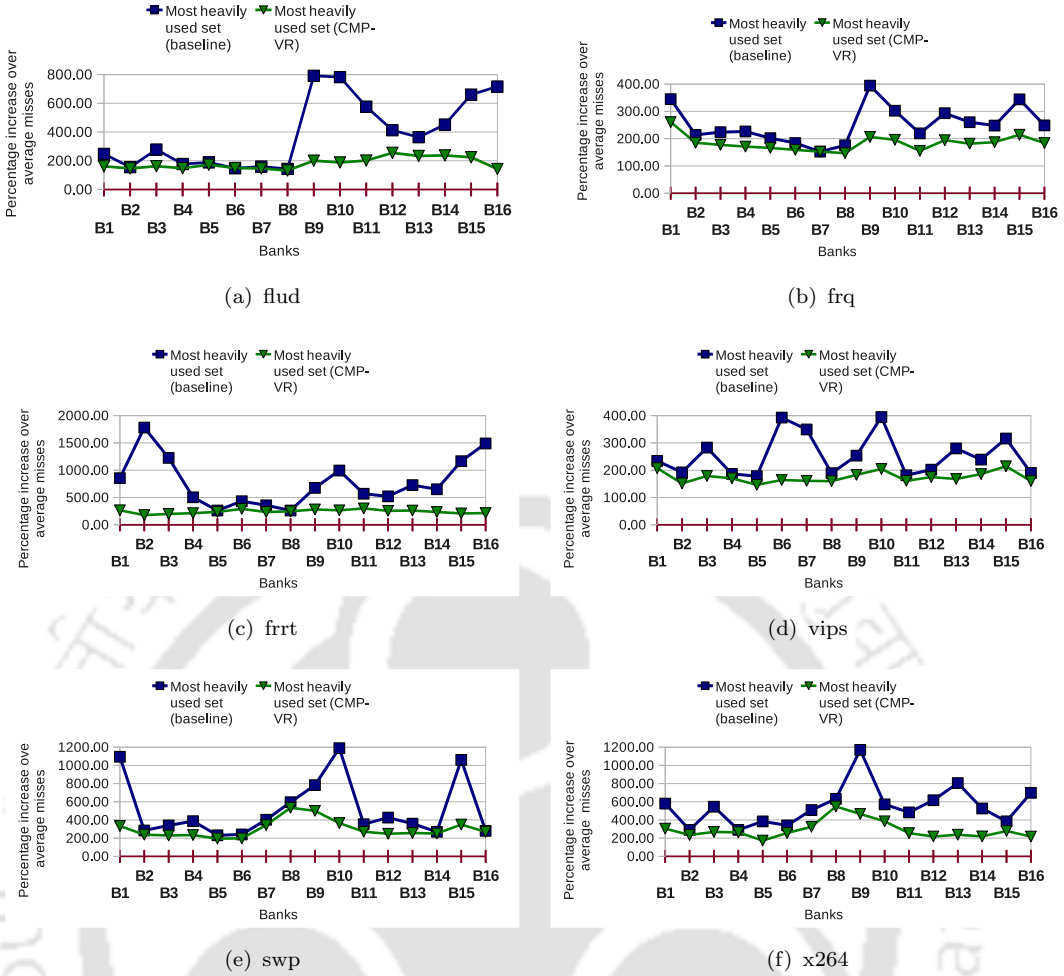
FIGURE 4.13: Conflict misses by the most heavily used set of both baseline (4M4W) and CMP-VR (4M4W-25R) (considering the statistics of six Parsec benchmarks [3] together).

of CMP-VR will be less if the communication cost (on-chip) of L2 miss is also considered.

4.4.3 Effect of CMP-VR on Heavily Used Sets

As mentioned in Section 4.1, the baseline cache has certain sets being used heavily relative to other sets. Consequently, heavily used sets have higher number of misses. An experimental analysis for such heavily used sets is given in Figure 4.1 as well as in Table 4.1. Figure 4.1, shows the number of misses in the most heavily used set as compared to the average number of misses of all the sets. A similar experiment is also performed for CMP-VR (4M4W-25R) to compare the results with the baseline. Figure 4.13 compares the number of misses in the most heavily used set of both baseline and CMP-VR. The comparison shows that CMP-VR significantly reduces the misses of heavily used sets.

Figure 4.14 shows the percentage increase of the number of misses in the most heavily used set compared to the average misses of all the sets. The graphs are shown for the baseline (4M4W) and CMP-VR (4M4W-25R). Each sub-figure represents one Parsec benchmark. Note that, the results for baseline (4M4W) used here are taken from the Table 4.1.



(a) flud (b) frq

(c) frrt (d) vips

(e) swp (f) x264

FIGURE 4.14: Benchmark-wise comparison of the percentage increase of misses (as compared to the average number of misses of all the banks) between the most heavily used set of baseline and CMP-VR.

4.4.4 Overheads

4.4.4.1 Replacement Overheads

The overheads of maintaining LRU based global replacement policy for TGS is ignored during the simulation. Maintaining such policy may increase the overhead of CMP-VR during actual execution. To remove such overheads we also analysed CMP-VR with a cost effective frequency based global replacement policy (for TGS) proposed in [8]. The policy was proposed to reduce the overhead of LRU in global replacement policies. In this policy, every TGS entry has to associate with a two-bit reuse counter. A Reuse Counter Table (RCT) is used for managing

all the counters. A PTR register is used to store the pointer of the RCT location from where the replacement process start searching its next victim. When a block is initially placed in RT the corresponding reuse counter in TGS is set as zero. For each subsequent access to the TGS entry, the corresponding reuse counter is incremented based on saturating arithmetic. To select a victim, the replacement engine searches the RCT for a reuse counter equal to zero. The searching starts with the index stored in PTR and keep decrementing all non-zero counters until a counter with zero reaches. The process is done in a circular fashion.

CMP-VR shows only 1.4% less improvements while using the cost effective replacement policy for TGS. Note that the replacement policy for banks remain LRU.

4.4.4.2 Energy Overheads

Presence of TGS in CMP-VR requires additional energy consumption as compared to the baseline design. The details of TGS were already discussed in Section 4.3.3. This section analyse the energy overhead of CMP-VR as compared to the baseline. CACTI 6.0 [2] is used to compute the energy consumption of both the baseline and CMP-VR. The modifications made for CMP-VR are internal to the tiles (cache-banks) and the complete mechanism is transparent from outside. All the inter-bank routing mechanisms remain the same for CMP-VR and the baseline. Table 4.7 shows the static and dynamic energy required for a bank in both baseline and CMP-VR. The static power consumption is shown in milliwatt (mW) while dynamic energy is shown in nanojoule (nJ). Each bank in CMP-VR has a TGS and the additional energy required by the banks of CMP-VR is for the TGS. It can be observed from the table that CMP-VR having more number of reserve ways require more additional energy.

Table 4.8 shows the improvements of CMP-VR with the total energy overheads as compared to the baseline. The total static energy consumption by the LLC in both baseline and CMP-VR is computed by multiplying the static power (in mW) obtained from CACTI (cf. Table 4.7) with the total execution time and the number

Bank Size	Assoc.	Static Power (mW)			Dynamic Energy (nJ)		
		Baseline	25R	50R	Baseline	25R	50R
128KB	4	285.876	317.876	349.876	0.0444	0.0512	0.0581
	8	274.04	306.04	338.04	0.0469	0.0538	0.0606
256KB	4	541.639	605.639	669.639	0.0718	0.0855	0.0992
	8	546.325	610.325	674.325	0.0739	0.0876	0.1014

TABLE 4.7: The static power and dynamic energy (per access) consumption of a bank in both baseline and CMP-VR. The values are calculated in CACTI 6.0. Note that the TCMP we used has 16 banks. Hence bank size of 128KB means 2MB of total LLC. Similarly 256KB of bank means 4MB LLC.

of banks. The total dynamic energy is calculated by multiplying the dynamic energy obtained from CACTI (energy per access) with the total number of LLC accesses. CMP-VR having 50% reserve storage (50R) performs better than CMP-VR having 25% reserve storage (25R). But 50R makes the TGS searching more expensive as compared to 25R due to its higher energy requirements. Since the overheads of CMP-VR having 50R is significantly more only 25R is recommended for use.

Configuration	CMP-VR	Baseline	Improvements (%)		Overheads (%)	
			CPI	MPKI	Static	Dynamic
2M-4W	2M4W-25R	2M4W	6.33	20.35	11.19	12.90
	2M4W-50R	2M4W	11.33	30.52	22.38	28.01
4M-4W	4M4W-25R	4M4W	7.75	21.55	11.81	16.48
	4M4W-50R	4M4W	13.31	36.29	23.63	35.16

TABLE 4.8: Comparison of CPI improvements, MPKI improvements and the TGS energy overhead of CMP-VR with baseline. The terms used in this table are explained in Section 4.4.1.

4.5 Summary

In this chapter we have presented CMP-VR (Chip-Multiprocessor with Victim Retention), an approach to improve cache performance by enhancing the local utilisation factor of each bank. The objective of this approach is to distribute the loads of heavily used sets among the underused sets. It also allows to retain the selected victim cache blocks on the chip for the longest possible time. In each cache bank, some number of ways from every set are reserved to store a victim block

from a heavily used set. In this way the load of heavily used sets are distributed among the less heavily used sets. Whereas other similar approaches like [20, 8] completely separate the traditional one-to-one mapping between the data lines and their corresponding tag entry, CMP-VR does this only for the reserve storage. Experimental comparison is done between CMP-VR and T-SNUCA (baseline) for different cache sizes and associativity. The simulation results show that the best improvements of CMP-VR are 36.29% and 13.31% in terms of MPKI and CPI respectively for a 4MB 4 way set associative cache. It also gives better performance while compared with a higher associative (8-way) baseline of the same cache size. Reduction in CPI and MPKI guarantees performance improvement. Due to the additional energy requirements, the highly improved configurations of CMP-VR are not recommended to use. A configuration of CMP-VR which is recommended to use gives 7.55% improvements in CPI and 21.55% improvements in MPKI.

Chapter 5

Victim Retention Using Static and Dynamic fellow-sets

This chapter proposes two alternative techniques to reduce the overheads of CMP-VR as well as improve the performance. The fully associative TGS of CMP-VR is replaced with a set associative additional tag array. The improved technique has less than 2% energy overhead over T-SNUCA whereas the same overhead in CMP-VR is more than 20%.

5.1 Introduction

A major challenge in CMP-VR is to reduce the energy consumed by its fully associative TGS. In CMP-VR, reserving 50% ways from each set (50R) gives better performance (in terms of CPI and MPKI) compared to reserving 25% ways from each set (25R). But because of the higher energy overhead it has not been recommended to use 50R. Making the additional tag array as set associative instead of fully associative can reduce the hardware overhead drastically.

CMP-VR has no restriction about the dynamic associativity increase of a set; in worst case a heavily used set can consume the entire RT section. Experiments found that it is sufficient to allow a heavily used set to increase its associativity

by double or triple [8]. V-Way [8] used this concept and allowed to increase associativity up to twice the normal associativity (TDR[1]=2). Though V-Way consumes less energy than CMP-VR, its storage overhead is more than CMP-VR (cf. Table 5.8). This motivated us to design new DAM based technique, called CMP-SVR, with minimum energy overheads. The performance of CMP-SVR is almost same as that of CMP-VR.

CMP-SVR significantly reduces the hardware overhead (especially energy) of CMP-VR but it cannot improve the performance more than CMP-VR. Hence our next goal is to enhance the performance of CMP-SVR. For this we propose another DAM based technique called FS-DAM, which provides better performance than both CMP-VR and CMP-SVR with almost same hardware overheads as in CMP-SVR.

Throughout this chapter the heavily used sets are termed as **H-Sets** and lightly used sets are termed as **L-Sets**. The procedure of classifying the sets either as **L-set** or **H-Set** is discussed later in this chapter.

The rest of the chapter is organized as follows. Next section gives the detail description about CMP-SVR. The experimental analysis of CMP-SVR is given in Section 5.3. Limitations of CMP-SVR and the possible improvements are discussed in Section 5.4. FS-DAM is discussed in Section 5.5. Section 5.6 gives the experimental analysis of FS-DAM. The summary of the chapter is given in Section 5.7.

5.2 CMP-SVR

In CMP-SVR the sets are grouped into multiple *fellow-groups*; all the sets belonging to the same group are called *fellow-sets*. A set can only use the reserve ways of its fellow-sets and hence each fellow-group has to maintain a separate RT section. CMP-SVR restricts the maximum possible associativity increase of a set.

[1]TDR means tag-to-data ratio, i.e. the ratio of the number of tag entries to the number of data entries.

In worst case a set can only use all the reserve ways of its fellow-sets. Since H-Sets require limited number of additional ways, such restriction of CMP-SVR does not degrade the performance as compared to CMP-VR. The CMP-VR can be termed as a CMP-SVR having only one fellow-group. Set balancing cache [21] proposed a similar technique but it allows only two sets to share ways together. CMP-SVR can be configured for different fellow-group sizes.

The additional tag-array, called SA-TGS, in CMP-SVR is set associative instead of fully associative. Each entry in SA-TGS has one-to-one mapping with a corresponding location in RT (same as in CMP-VR) and hence no forward/backward pointers are required as were needed in [8] [10]. Since both the cache and SA-TGS are set-associative the rest of the chapter uses the term *cacheSet* to represent the sets of cache and *tgsSet* to represent the sets of SA-TGS. Similarly, *cacheWay* and *tgsWay* are used. Also the terms *set* and *way* represent the cacheSet and cacheWay respectively, unless otherwise specified.

ALGORITHM 1: ***SVRFlwGrp*** divides the *cacheSets* into fellow-groups in CMP-SVR.

Input: $S \longleftarrow$ total number of *cacheSets* in the cache.
Input: $F \longleftarrow$ fellow-group size: total number of *cacheSets* in each fellow-group.
Output: list of all fellow-groups.

```
1  𝕋 ⟵ S/F; // Total number of fellow-groups in cache.
2  List<Index> fellow-group; // Each fellow-group is a List of cacheSets.
3  List<fellow-group> all-groups; // List of fellow-groups.
4  fellow-group-index← 0
5  for cacheSet-index← 0 to S − 1 do
       /* Add cacheSet (cacheSet-index) to fellow-group
          (fellow-group-index).                                    */
6      all-groups.get(fellow-group-index).add(cacheSet-index)
7      fellow-group-index++
8      if fellow-group-index == 𝕋 then
9          fellow-group-index ← 0
10     end
11 end
12 return all-groups
```

The fellow-groups in CMP-SVR are statically decided. Starting from *set-0*, a set is assigned to each fellow-group. When all the fellow-groups are assigned one set each, the process is repeated for the remaining unassigned sets. The algorithm

used to create the fellow-groups is given in Algorithm 1. Now consider an A-way set associative cache having S number of cacheSets, R number of cacheWays from each cacheSet reserved for RT and F as fellow-group size. The fellow-group sizes (F) are in powers of 2. The structure of the SA-TGS is:

$$Number\ of\ entries\ in\ SA\text{-}TGS\ = R * S \tag{5.1}$$

$$Associativity\ of\ SA\text{-}TGS\ (A^t) = R * F \tag{5.2}$$

$$Number\ of\ tgsSets\ (S') = S/F \tag{5.3}$$

Each SA-TGS entry is statically mapped to a location in RT. An SA-TGS entry having tgsSet number as tgs_s and tgsWay number as tgs_w is statically mapped to the RT location calculated by the following two equations.

$$cacheSet\ index = ((tgs_w/R) * S') + tgs_s \tag{5.4}$$

$$cacheWay\ index = (A - R) + (tgs_w\%R) \tag{5.5}$$

Each fellow-group has a separate RT section and one dedicated tgsSet in SA-TGS to manage the RT. To search the RT section, the dedicated tgsSet is searched. Since the fellow-groups are created based on Algorithm 1, given any cacheSet index $s_i; 0 \leq s_i < S$, the corresponding tgsSet index can be identified by the Equation 5.6.

$$tgsSet\ index = s_i\%S' \tag{5.6}$$

The mapping between RT locations and the SA-TGS entries are statically fixed. Given a cacheSet index $s_i; 0 \leq s_i < S$ and cacheWay index as $w_j; (A - R) \leq w_j < A$, the corresponding tgsWay index can be calculated by the following equation.

$$tgsWay\ index = (s_i/S') * R + (w_j - (A - R)) \tag{5.7}$$

Note that, all the division operations in the above formulas are integer divisions (e.g. $1/2 = 0, 3/2 = 1$, etc.) and '%' is the modulo operator used in high level

programming languages. The set and way numbering of both cache and SA-TGS starts from 0.

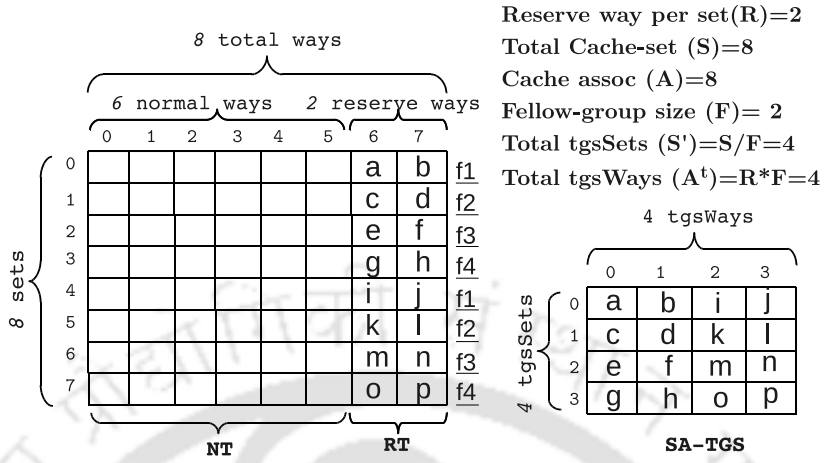


FIGURE 5.1: CMP-SVR: way distribution, fellow sets and associative mapping into SA-TGS.

Example: An example of a 4KB 8-way set associative cache ($A = 8$) and its corresponding SA-TGS is shown in Figure 5.1. Assuming block size as 64B the cache has 8 sets ($S = 8$). Considering reserve ways per set (R) as 2 and fellow-group size (F) as 2, the total number of *tgsSets* (S') and *tgsWays* (A^t) can be calculated as $S' = S/F = 4$ and $A^t = R * F = 4$. There are four fellow-groups $f1, f2, f3$ and $f4$. Each cacheSet is labeled with the fellow-group id in which it belongs. The letters $(a, b, \ldots, p)$ in RT as well as in SA-TGS are labels to show the one-to-one mapping between a RT location and its corresponding SA-TGS location; they do not represent any block content.

Both CMP-VR and CMP-SVR have same data management policy as discussed in Section 4.3.2. The difference between CMP-VR and CMP-SVR is the structure of the additional tag array (TGS for CMP-VR and SA-TGS for CMP-SVR) and the one-to-one mapping policy with RT. To search the RT section CMP-SVR only searches the corresponding tagSet in SA-TGS while in case of CMP-VR the entire fully associative TGS has to be searched.

5.2.1 Additional Search Time

Since *indirect hits* require to access the cache twice: first simultaneously with SA-TGS search and second after the tag is found in SA-TGS, the time required for *indirect hits* is twice the time required for a *direct-hit*. The time to detect a miss remains same. The detail discussion was given in section 4.3.3.2 during the description of CMP-VR.

5.2.2 Implementation of CMP-SVR in TCMP

As mentioned in Section 2.3.1 that in case of TCMP based architectures, the DAM based policies are implemented on each bank separately, hence the concept of CMP-SVR discussed above also does the same. It is applied to each bank separately and the process is transparent from outside the bank. Logically it is also possible to apply CMP-SVR on some selective banks without applying on all the banks. But we have not considered such designs for our experiments. Since CMP-SVR is a local utilisation enhancement technique other inter-bank utilisation enhancement policies like T-DNUCA (cf. Chapter 6), Co-Operative Caching [12], HK-NUCA [9] etc can be combined with it.

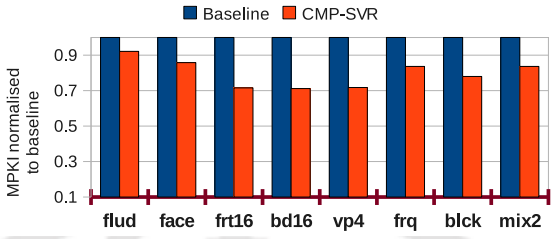
5.3 Experimental Analysis of CMP-SVR

As mentioned in Section 3.3.5 the baseline architecture for each of our proposed technique is a 16 core T-SNUCA (cf. Figure 3.1). CMP-SVR is implemented on top of the baseline architecture as done for CMP-VR. It is compared with both T-SNUCA and CMP-VR.

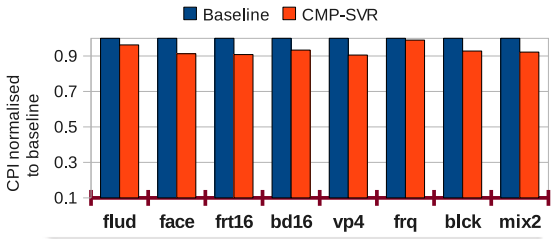
CMP-SVR is implemented separately on each tile of the TCMP, i.e., each tile has its own NT, RT and SA-TGS. The detail experimental procedures are discussed in section 3.3.3. The configurations about the processor, cache memory and main memory used for the experiment are given in Table 5.1. Eight multi-threaded

Component	Parameters
No. of tiles	16
Processor	UltraSPARCIII+
L1 I/D cache	64KB, 4-way
LLC size, bank size	4MB, 256KB 4-way
Memory	1GB, 4KB/page
CMP-SVR: reserve ways per set (R)	2
CMP-SVR: fellow-group size (F)	4

TABLE 5.1: System Parameters



(a) MPKI



(b) CPI

FIGURE 5.2: Normalized performance comparison of CMP-SVR with baseline design.

benchmarks from Parsec benchmark suite has been used for the simulation. The benchmarks are: **flud**, **face**, **frt16**, **bd16**, **vp4**, **frq**, **blck** and **mix2**. The detail description about the benchmarks are given in section 3.3.2.

5.3.1 Simulation Results and Analysis

We first compare our proposed method with baseline TCMP. The number of L2 misses in CMP-SVR and in baseline can be calculated as follows:

- Total L2 misses in CMP-SVR =
  Total L1 miss - (Total *direct-hit* + Total *indirect-hit*).
- Total L2 misses in baseline = Total L1 miss - Total L2 hit.

We consider the total size of the L2 cache as 4MB, hence the size of each bank is 4MB/16=256KB. The more detailed configurations regarding both baseline as well as CMP-SVR are given in table 5.1.

Figure 5.2 shows the performance comparison of CMP-SVR with the baseline design. Both graphs in the figure show the results normalized to the corresponding values of the baseline design.

Figure 5.2(a) shows that CMP-SVR gets 7.86% to 28.8% reduction in MPKI with an average of 20.61%. In other words, on average, 20.61% times the blocks are found in RT, resulting in an *indirect-hit*, whereas they would be termed as misses in the baseline design. Figure 5.2(b) shows the performance comparison in terms of CPI. It shows that in case of CMP-SVR, CPI improves by 1.04% to 9.45% with an average improvement of 6.76%.

5.3.2 Hardware Overheads

The additional storage (compared to the baseline) required in CMP-SVR depends on the size of SA-TGS. Each SA-TGS entry stores a tag address and a validity bit. For instance consider the cache shown in Figure 5.1. It is a 4KB 8-way set associative cache having block size of 64 bytes. Both reserve ways per set and fellow-group size are given as 2. The SA-TGS shown in the figure has 4 ways and 4 sets, hence total 16 entries (details given in Section 5.2). Considering 36-bit address space, each cache block has a tag address of 27 bits. Each SA-TGS entry stores 27(tag)+1(validity)=28 bits. So the total storage requirement of SA-TGS is 16*28=448 bits = 56 bytes, which is only 1.36% of the total cache size. So the additional storage requirements in CMP-SVR is 1.36% of the baseline.

We use Cacti 6.0 [2] to find the power/energy consumption for both baseline and CMP-SVR. All the changes done by CMP-SVR in TCMP are internal for the tiles and has no changes in the NoC. The energy overhead in CMP-SVR is 1.8% as compared to the baseline while the same overhead in case of CMP-VR is more than 20%.

5.3.3 Comparison with CMP-VR

The result of CMP-SVR has been compared with CMP-VR. The configuration used for CMP-VR is same as mentioned in Table 5.1 (except the fellow-group size, as CMP-VR has no fellow group concept). Both CMP-SVR and CMP-VR has been compared with same sized baseline. Table 5.2 shows the improvements of CMP-VR as well as CMP-SVR over the baseline design. CMP-SVR has much lesser energy overhead compared to CMP-VR. In particular CMP-VR has 20% energy overhead whereas in case of CMP-SVR the overhead is 1.8%. The storage overheads of both the architecture is almost same. The performance of CMP-SVR is slightly less than CMP-VR.

Cache Design	**Improvements (%)**		**Energy Overheads (%)**
	CPI	MPKI	
CMP-VR	7.75	21.55	23.63
CMP-SVR	6.76	20.61	1.8

TABLE 5.2: Improvements and energy overhead of CMP-SVR and CMP-VR over the baseline.

5.3.4 Analysis of the Results

In particular CMP-SVR with set-associative SA-TGS has much lesser energy overhead (1.8%) as compared to CMP-VR (23.63%). CMP-SVR shows 6.5% improvement over baseline, which is slightly less than CMP-VR.

5.4 Performance Enhancement of CMP-SVR

CMP-SVR reduces the hardware overhead of CMP-VR significantly but it cannot improve the performance more than CMP-VR. After proposing CMP-SVR we analyzed the behavior of it and observed that it is possible to increase its performance with almost same hardware overheads. We proposed an extension of CMP-SVR called FS-DAM which gives better performance than both CMP-VR and CMP-SVR.

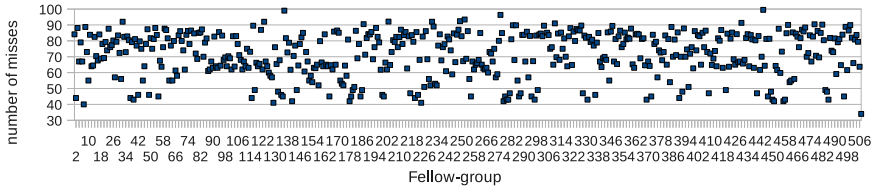


FIGURE 5.3: fellow-group usage in CMP-SVR. Size of the fellow-groups considered is 2.

Benchmark	Period (million)	B0	B1	B2	B3	Average*
fluid	20 to 30	18.95/25.00	21.88/14.84	21.48/16.60	15.43/19.92	25.18/22.72
	40 to 50	23.83/27.54	27.34/21.09	22.66/29.88	28.12/18.36	26.18/23.86
ferret	20 to 30	23.73/22.17	19.14/24.90	17.09/23.83	17.87/23.24	23.24/22.48
	40 to 50	29.00/19.73	20.02/23.83	19.53/22.46	19.34/24.32	25.08/23.57
freqmine	20 to 30	23.05/25.39	25.98/20.90	22.66/25.78	27.15/23.44	25.28/25.58
	40 to 50	23.44/17.19	17.77/25.59	25.78/15.82	26.95/18.55	24.05/22.37

TABLE 5.3: The percentage of fellow-groups in CMP-SVR having only one type of *cacheSets* (H-Sets or L-Sets). The results are calculated on the 16-core baseline TCMP, applying CMP-SVR on each bank separately. The results of four banks (out of 16) and the average of all 16 banks are given. The values are given in (X/Y) format, X is the percentage of fellow-groups having only H-Sets and Y is the percentage of fellow-groups having only L-Sets. *Average of all 16 banks.

The main limitation of CMP-SVR is the static grouping of its fellow-groups which never change during the execution. The grouping is done based on *cacheSet* indexes and does not consider the set-loads. It may be possible that the H-Sets are not evenly distributed over the fellow-groups. Table 5.3 shows the distribution of *cacheSets* among the fellow-groups in CMP-SVR. The table shows that on average, 25% fellow-groups are made of only H-Sets while 24% fellow-groups are made of only L-Sets. Remaining groups have a mixture of L-Sets and H-Sets. The groups having only H-Sets shows higher conflict misses and the groups having only L-Sets may be underused. Figure 5.3 shows the group usage of a L2-bank in CMP-SVR. The usage is recorded after 30 million cycles of execution. The figure shows that the groups are not used uniformly. Such type of non-uniform grouping in CMP-SVR restricts their performance improvements. A grouping based on the set-loads can increase the utilization of the cache and may also improve performance.

The non-uniform nature of sets mentioned in CMP-SVR and V-Way is based on the overall set usage, but the behavior of a set is not permanent; the usages of

sets may change dynamically as the execution proceeds. Such changes in the usage also change the category (H-Set/L-Set) of sets dynamically. Based on experimental analysis we found that, on average 47% sets change their category after every 20 million cycles of execution. Here category change means change of category from H-Set to L-Set or vice versa. Mechanism to handle such changes is absent in CMP-SVR and V-Way. Taking into account such periodic changes of set behaviors can have a positive impact on performance.

The proposed FS-DAM technique uses the set-load metric to group them in various fellow-groups. The H-Sets are equally distributed among the fellow-groups. After a fixed cycles of execution, the grouping is done again based on the current set-loads. Such grouping and re-grouping process utilizes the cache more efficiently and hence improves the performance as compared to CMP-SVR. The detailed discussion about FS-DAM is given in the next section.

5.5 Proposed Architecture: FS-DAM

The main objective of FS-DAM is to improve the performance of CMP-SVR without increasing the storage, area and energy overheads. The data management technique of FS-DAM is similar to CMP-SVR. Each set has some number of ways reserved for the RT section and the sets are grouped into multiple fellow-groups. A set can only use the reserved ways from its fellow-sets, and all the fellow-groups are non-overlapping. However, unlike CMP-SVR, the fellow-groups are formed based on the set-loads and re-grouped after every fixed interval of execution. The necessity of re-grouping is already discussed in Section 5.4. This dynamic nature of group construction makes the structure of FS-DAM different from that of CMP-SVR.

5.5.1 Some common terms used for FS-DAM

Given a block B, the address of the block is termed as $addr(B)$. Each block is mapped to a *cacheSet* based on the set indexes of $addr(B)$, which is called the *home-set* of B and is termed as $hs(B)$. A set can belong to only one fellow-group. The fellow-group to which the set belongs is called the *home-group*. Each fellow-group has its own RT section and a set can use the RT section of its *home-group*. The RT of the *home-group* of set s is termed as $RT(s)$ and the NT section of the same set is termed as $NT(s)$. The terms *cacheSet*, *cacheWay*, *tgsSet* and *tgsWay* are already described in Section 5.1.

5.5.2 FS-TGS and its dynamic mapping with RT

The additional tag-array in FS-DAM, termed as FS-TGS, is same as SA-TGS of CMP-SVR. For a same cache configuration both FS-TGS and SA-TGS have equal *tgsSets* and *tgsWays*. Given an M-way set associative cache having S number of *cacheSet*s, R number of reserved *cacheWay*s per *cacheSet* and fellow-group size as F, the structure of the FS-TGS is given by the Equations 5.8, 5.9 and 5.10. Note that the equations are similar to the Equations 5.1, 5.2 and 5.3.

$$Number\ of\ entries\ in\ FS\text{-}TGS(E') = R * S \tag{5.8}$$

$$Associativity\ of\ FS\text{-}TGS(A') = R * F \tag{5.9}$$

$$Number\ of\ tgsSets\ in\ FS\text{-}TGS(S'') = S/F \tag{5.10}$$

The difference between FS-TGS (of FS-DAM) and SA-TGS (of CMP-SVR) is in the mechanism to maintain the one-to-one relationship between the FS-TGS locations and the corresponding RT locations. The grouping in CMP-SVR is done based on the set indexes. Therefore, the one-to-one mapping between the RT and the SA-TGS entries can be calculated by the Equations 5.4, 5.5, 5.6, and 5.7. In FS-DAM the grouping is done based on the set-loads; irrespective of the set index, a set can be in any fellow-group based on its current load. Hence the equations used for CMP-SVR cannot be directly used for FS-DAM. Some additional information has to be maintained for such dynamic groupings [21, 67].

The fellow-groups in FS-DAM are represented as follows:

$$G = \{g_0, g_1, \ldots, g_{S''}\},$$

Where g_i is the i^{th} fellow-group of sets. Note that there are S'' number of fellow-groups. Every fellow-group g_i has F number of *cacheSets*, called fellow-sets.

$$g_i = \{f_0, f_1, \ldots, f_{F-1}\}, 0 \leq i < S''$$

In each fellow-group g_i, a fellow set is denoted by $f_j (0 \leq j < F)$. Each fellow-set represents a unique *cacheSet* between 0 to S. For example $g_2 = \{2, 23, 34\}$ means the fellow-group g_2 has three fellow-sets: 2^{nd}, 23^{rd} and 34^{th} sets of the cache. The subscript j in each fellow-set is called the fellow-set number which represents its order in the fellow-group. In the above example, entry "2" indicates 2^{nd} set from cache and 0^{th} index in the group.

In FS-DAM the i^{th} fellow-group (starting from 0) is mapped to the i^{th} *tgsSet* of FS-TGS. All the reserve *cacheWays* of g_i are mapped with the *tgsWays* of i^{th} *tgsSet* (ts_i). The mappings are done in index order, i.e., the reserve ways of first fellow-set (f_0) in g_i are mapped first into the tgsWays of ts_i. The first reserve way of f_0 maps with the 0^{th} *tgsWay* of ts_i. The next reserve way of f_0 maps to the next *tgsWay* of ts_i. Once all the reserve ways of f_0 are mapped, the reserve ways of f_1 start to map in the successive ways of ts_i. In this manner all the reserve ways of g_i map with the *tgsWays* of ts_i in FS-TGS. The associativity of FS-TGS, A', and the number of total reserve ways in g_i must always be the same. Hence it can be observed that the mapping between the fellow-groups and their corresponding *tgsSet* in FS-TGS is static. Given a FS-TGS location (t_s, t_w) where t_s is the *tgsSet* number and t_w is the *tgsWay* number, we have:

$$\textit{fellow-group number} = t_s$$

$$\textit{fellow-set number within the above fellow-group} = t_w/R$$

But since the fellow-groups are made (re-grouped) dynamically the *cacheSets* are not statically mapped to the fellow-groups. Hence the above formula can only give the fellow-group number and the fellow-set number but cannot give the actual *cacheSet* number ($f_{t'_w/R}$). To store the *cacheSet* number for a particular fellow-set number an additional table is used called the *SetMapper*. The *SetMapper* has S'' rows and F columns; each row represents a fellow-group and each column represents a fellow-set. An entry, $SetMapper[l][k] = m$, $(0 \leq l < S'', 0 \leq k < F, 0 \leq m < S)$, means k^{th} fellow-set of l^{th} fellow-group is the m^{th} *cacheSet*. Algorithm 2 shows the complete FS-TGS to cache mappings.

ALGORITHM 2: FS-TGS and cache mapping

TgsToCache(tgs_r, tgs_w) **begin**
  Input: FS-TGS location (tgs_r, tgs_w)
  Output: Cache memory location (in RT).
  /* tgs_r is the tgsSet index and tgs_w is the tgsWay index. */
  /* R: reserve ways per cacheSet, M: associativity of the cache. */
  cachewayID=$(M - R) + (tgs_w \% R)$
  cachesetID=**SetMapper**$[tgs_r][tgs_w/R]$
  return *(cachesetID, cachewayID)*
end

CacheToTgs(*ch_set*) **begin**
  /* The function is called for indirect-search. */
  Input: Cache-set index (*ch_set*)
  Output: corresponding tgsSet in FS-TGS.
  /* To search a block in the home-group's RT, the corresponding tgsSet in FS-TGS is searched. */
  return *SetPointer*[*ch_set*];
end

A similar mapping from cache to FS-TGS is also required to search a block in its RT section. Each fellow-group has a separate RT section and a set can use the RT section of its fellow-group. To search a block in RT the corresponding tgsSet in FS-TGS has to be searched. As mentioned above, each RT section has a separate tgsSet in FS-TGS. An additional array called *SetPointer* is used to map each *cacheSet* with the corresponding *tgsSet* in FS-TGS. Each *cacheSet* has an entry in *SetPointer*. An entry $SetPointer[p] = t$ means cacheSet p maps to tgsSet t. Figure 5.4, shows an example of FS-DAM.

FS-TGS same as SA-TGS needs to store few additional bits to distinguish the block addresses having same tag but maps to different home-set.

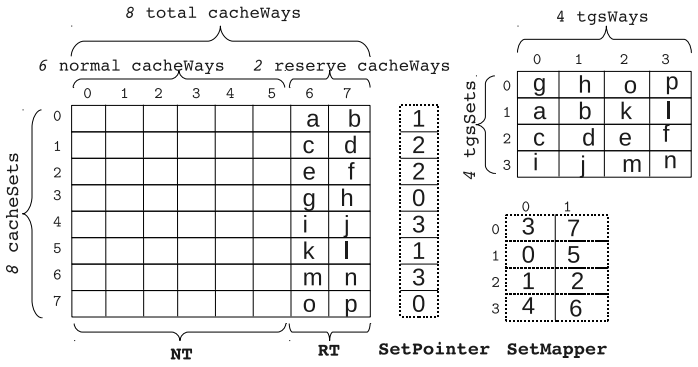


FIGURE 5.4: An example of FS-DAM. Total cacheSets (S) = 8, Total cacheWays (M) = 8, Reserve cacheWays per Cacheset (R) = 2, Fellow-group size (F) = 2, Total tgsSets in FS-TGS (S'') = S/F = 4 and Total tgsWays per tgsSet (A') = $R * F$ = 4.

5.5.3 Operations of FS-DAM

The major actions to be defined for FS-DAM are: (a) Initialization, (b) Normal execution and (c) Re-grouping. These are discussed in detail below:

5.5.3.1 Initialization

Initially FS-DAM starts with a random grouping of cacheSets. The cacheSets are randomly divided into S'' fellow-groups (each of size F). Once the fellow-groups (G) are ready, the FS-TGS, *Setmapper* and *SetPointer* are initialized based on the initial grouping. Re-grouping based on set-loads is performed after a fixed cycles of execution.

5.5.3.2 Normal execution

For every block request to the cache, the block (say B) is searched in the NT section of its *home-set* and also parallely searched in the RT section of its *home-group*. Searching in NT section is called *normal search* and searching in RT section is called *indirect search*. Normal searching is same as in the conventional set associative cache; for a block B, the $NT(hs(B))$ is searched. For indirect search, a particular *tgsSet* in FS-TGS has to be searched. The *tgsSet* which needs to be searched is $SetPointer[hs(B)]$: (say ts_B). The tag part of $addr(B)$ is compared with the tag stored in all the tgsWays of ts_B. If the tag is matched with a

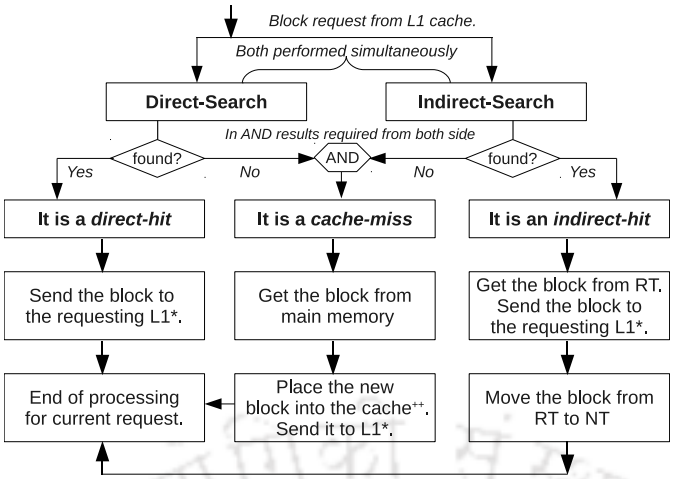


FIGURE 5.5: The flow diagram of FS-DAM: normal operation. The diagram shows the flow only after the request reaches the L2 cache controller. * Sending a block to the requested L1 may needs some coherence steps to do first. ++ Placing a block in cache memory may need to move another block to RT. The block movement and replacement procedures are discussed in Section 5.5.3.2

tgsWay (say tw_B) then the corresponding location in the cache $RT(hs(B))$, has the block B. Given the tgsSet and tgsWay, the corresponding cache location can be calculated by the procedure "TGSTOCACHE" (given in Algorithm 2). Both normal and indirect searches are performed simultaneously. If the block is found by normal search then it is a *direct-hit* and if the block is found in indirect searching then it is an *indirect-hit*. In the case of an indirect-hit the block is moved back to $NT(hs(B))$.

Moving the block from RT to NT is easy provided NT has a free (invalid) way, otherwise the block is swapped with the LRU block of NT. During replacement, instead of removing a victim block completely from the cache it moves the block to RT. Moving a victim block (say V) to RT is easy if RT has a free (invalid) location, otherwise the LRU block of RT is replaced with V. Searching of RT is indirect; instead of searching RT it searches FS-TGS which has a dedicated entry for each corresponding RT locations. Since *indirect hits* require to access the cache twice: first during indirect search and second after the tag is found in FS-TGS, the cache access time is twice the time required for a *direct hit*. Moving and swapping blocks between NT and RT need to change the entries in FS-TGS. Figure 5.5 gives the flowchart of the normal operations of FS-DAM.

ALGORITHM 3: Grouping based on current set-loads.

```
MakeFellowGroup() begin
    Input: S ⟵ total number of cacheSets in the cache.
    Input: F ⟵ fellow-group size- total number of sets in each fellow-group.
    Output: list of all fellow-groups.
    𝔻 ⟵ S/F; // total number of fellow-groups in cache.
    List<Index> fellow-group; // Each fellow-group is a List of cacheSet
       indexs.
    List<fellow-group> all-groups; // List of fellow-groups.
    List<Index> cache_miss; /* cache_miss stores the current misses of all
       the cacheSets.                                                  */
    List<Index> sorted_set ⟵ Sort all the cacheSets based on their current misses.
    for i ← 0 to 𝔻 do
        fg_i ⟵ Create an empty fellow-group.
        for j ← 0 to F do
            next_set_index=sorted_set[i + j * 𝔻]
            fg_i.add(next_set_index);
        end
        all_groups.add(fg_i)
    end
    return all_groups
end
```

5.5.3.3 Re-grouping

The purpose of re-grouping is already discussed in Section 5.4. After every fixed cycles of execution, the cacheSets are re-grouped based on the current set-loads. The interval after which the re-grouping occurs is called re-grouping period (RGP). The grouping mechanism is based on the current set-loads. Algorithm 3 describes this process. According to the algorithm, an H-Set with higher conflict misses (among the H-Sets) combines with an L-Set having lesser conflict misses (among the L-Sets). The sets are sorted based on the conflict misses and then divide into D number of segments, where D is the total number of fellow-groups. For making a new group it selects the first unselected set from each segment. The selected sets from each segment are marked as selected. Such grouping based on conflict misses allows the cache to be used more efficiently. Combining H-Sets with L-Sets enables the idle ways of L-Sets to be utilized by the H-Set and thus improving MPKI. The details regarding re-grouping is discussed in Section 5.5.4.

5.5.4 The Re-grouping procedure

Re-grouping modifies the fellow-groups and consequently the RT section. The blocks in the RT may need to re-organize as their home-set may now belong to a different fellow-group. Re-organization of RT blocks are done by three ways: (a) swapping the blocks between two fellow-groups, (b) Moving the blocks from one fellow-group to another, and (c) removing the blocks from the cache. For a particular block B, if its home-set ($hs(B)$) joins a new fellow-group g_j, then the block has to be moved from the current RT location to the RT of g_j (if it is currently not in RT of g_j).

If the RT(g_j) has a free location available then B will be moved there, otherwise it checks if any block swapping is possible or not. When the first and second options cannot be used, the block is removed from the cache. Note that the re-organization is performed only in the RT section and the NT section remains the same. After the re-organization, SetMapper, SetPointer and FS-TGS are updated accordingly. The details about mapping between the cache and the FS-TGS is mentioned in Section 5.5.2.

5.5.4.1 Choosing the re-grouping period (RGP)

The RGP is taken as 10 million cycles. Most of the applications from PARSEC benchmark suite [3] show best performance with RGP of 10 million cycles. However certain applications perform better with RGP as more than 10 million cycles. Figure 5.6 shows the MPKI of five benchmarks having RGP as 10, 20 and 30 million cycles. It can be observed that 10 million cycles gives better results for majority of applications. But since different applications performs better with different RGP, an additional condition is added for triggering the re-categorization process.

After every RGP cycles of execution, the standard deviation of the conflict misses of all the fellow-groups is calculated and if the deviation is very less than it means the groups are uniformly used with the current grouping, so the re-grouping process

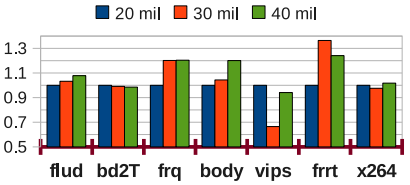


FIGURE 5.6: MPKI of FS-DAM for different RC-periods. The results shown are normalized to the result of 10 million cycles.

is skipped until the next RGP. This prevents unnecessary re-categorization after every RGP in case the set usages have not highly deviated.

5.5.4.2 Cost of re-grouping

The re-grouping processes has some overhead in terms of both time and hardware. During re-grouping some blocks have to be moved from one location to another and some require to write-back into main memory. The algorithm of re-grouping is designed such that during the regrouping, each block in RT will access at most twice. The performance of FS-DAM shown in this chapter are after considering all the cycles required for re-grouping. Also the additional condition for re-grouping as mentioned above avoids the unnecessary re-groupings. The hardware overheads of re-grouping is discussed in Section 5.6.5.

5.5.5 Summary of FS-DAM

THERE ARE THREE MAIN PROCESSES IN FS-DAM.		
Initialization	**Normal Execution**	**Re-grouping Process**
Starts with a random grouping of CacheSets (Section 5.5.3.1).	Normal execution means the data management policy of FS-DAM. Figure 5.5 shows the data management policy of FS-DAM. The detail discussion is given in Section 5.5.3.2.	After a fixed interval of normal execution the fellow-groups of FS-DAM are re-grouped based on the current set-loads (Section 5.5.4).

5.6 Experimental Evaluation

FS-DAM is compared with baseline, CMP-SVR, V-Way, Z-Cache and SBC. The baseline is a 16 core TCMP as mentioned in Section 3.3.5.

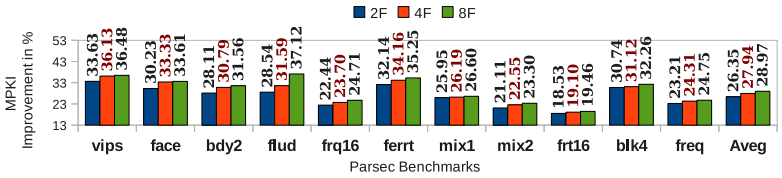
5.6.1 Experimental Setup

All the DAM based policies are separately implemented for each L2-bank of the baseline. Simulations are performed by running benchmarks on GEMS [85], with the help of SIMICS [82]. Note that, the behavior of the L1 caches remain unchanged. The detailed configuration about the processor, cache memory and main memory used for the experiments is given in Table 5.4.

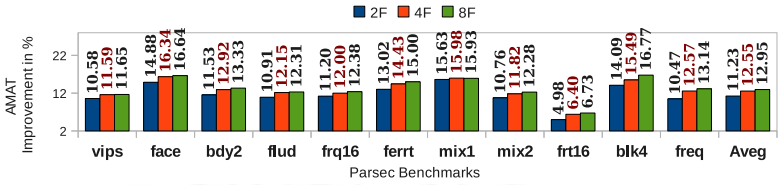
Component	**Parameters**
No. of tiles / Processor	16 / UltraSPARCIII+
L1 I/D cache	64KB, 4-way
LLC size	2MB and 4MB
bank size	128KB 4-way / 256KB 4-way
Memory	1GB, 4KB/page
Reserve ways per set (R)	50%
Fellow-group size (F)	2, 4 and 8

TABLE 5.4: System Parameters.

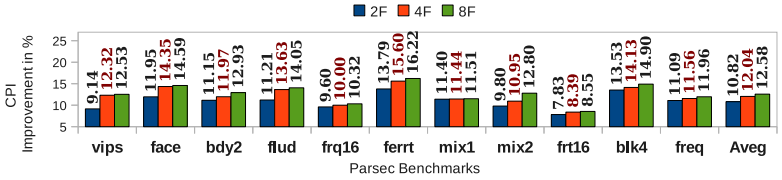
The size of LLC (L2) is considered as 2MB and 4MB for the experiments. There are 16 tiles (L2-banks) in the baseline. In the case of 2MB L2 cache, the size of each L2-bank (total 16 L2-banks) is 128KB and in the case of 4MB L2 cache, the size of each L2-bank is 256KB. The experiments are done for two different configurations. The first configuration, termed as **C1**, has 128KB 4-way associative L2-banks; the second configuration, termed as **C2**, has 256KB 4-way associative L2-banks. Eleven multithreaded benchmarks from PARSEC [3] benchmark suite are used for simulation. The benchmarks are: *vips*, *face*, *ferrt*, *freq*, *bdy2*, *mix1*, *mix2*, *blk4* and *frt16*. The details about the benchmarks are discussed in Section 3.3.2. The experimental procedure are discussed in section 3.3.3.



(a) MPKI (**C1**).



(b) AMAT (**C1**).



(c) CPI (**C1**).

FIGURE 5.7: Improvements in FS-DAM over baseline TCMP. xF means FS-DAM with fellow-group size x. The cache configuration is **C1**.

5.6.2 Comparisons for C1

5.6.2.1 Comparison with baseline

Figure 5.7 shows the improvement of FS-DAM as compared to baseline. Three different configurations of FS-DAM based on the fellow-group sizes are used for comparison. The terms 2F, 4F and 8F means FS-DAM having fellow-group size as 2, 4 and 8 respectively. The improvement in terms of MPKI is given in Figure 5.7(a). The average improvements are 26.35%, 27.94% and 28.97% for 2F, 4F and 8F respectively. Reduction (improvement) in MPKI results in more number of cache hits and hence reduces the number of expensive main memory accesses. Reduction in main-memory accesses improves the average memory access time (AMAT). Figure 5.7(b) shows the improvements of FS-DAM in terms of AMAT over baseline. The improvements are 11.23% (2F), 12.55% (4F) and 12.95% (8F). Improvement in MPKI as well as in AMAT means improvement in CPI. Figure

	fellow-group size (F) =2			fellow-group size (F)=4			fellow-group size (F) =8		
Benchmarks	**MPKI**	**AMAT**	**CPI**	**MPKI**	**AMAT**	**CPI**	**MPKI**	**AMAT**	**CPI**
vips	17.86	05.21	02.93	19.83	05.55	05.12	20.01	05.34	05.79
face	16.54	06.61	06.21	17.45	06.66	06.50	17.40	06.80	06.38
bdy2	14.52	06.11	04.76	15.82	06.35	05.18	16.42	06.82	06.16
flud	10.38	05.32	05.47	11.81	05.04	06.52	18.33	05.05	06.94
frq16	13.00	05.67	04.54	13.76	06.37	04.58	14.85	06.32	04.86
ferrt	16.07	05.39	07.42	14.26	06.17	07.60	14.37	06.70	07.46
mix1	11.90	08.71	05.95	10.45	08.25	05.92	10.83	08.23	05.95
mix2	11.21	05.27	05.27	12.46	05.92	06.21	12.77	06.09	07.84
frt16	10.37	03.02	05.48	10.87	03.46	05.97	10.73	03.47	05.86
blk4	14.04	08.11	06.91	13.69	08.80	07.17	13.97	10.05	07.97
freq	11.14	05.29	05.10	11.62	06.43	05.40	11.82	07.01	05.73
Average	**13.13**	**05.69**	**05.32**	**13.56**	**06.11**	**5.95**	**14.38**	**06.32**	**06.38**

TABLE 5.5: Improvements (in %) of FS-DAM over CMP-SVR. The improvements are shown in terms of MPKI, AMAT and CPI for three different fellow-group sizes. The cache configuration is **C1**.

5.7(c) shows that the average CPI improvement of FS-DAM as compared to baseline are: 10.82% (2F), 12.04% (4F) and 12.58% (8F).

The improvement of FS-DAM over baseline is because of the better utilisation of the storage alloted to each bank. The heavily used sets can store more number of blocks than the bank associativity. It has been observed that the hit rate of FS-DAM is better than baseline. The improvement in hit rate is because of the indirect hits of FS-DAM. The heavily used set can use the reserve ways of underused sets and hence generates indirect-hits which would have a miss in case of baseline even though there are idle ways in other sets.

5.6.2.2 Comparison with CMP-SVR

FS-DAM is proposed to handle the issues in CMP-SVR. The index based static fellow-group creation in CMP-SVR restricts the utilisation enhancement in the banks. FS-DAM creates fellow-groups based on the set-loads and also re-grouped them dynamically if required. Table 5.5 shows the improvement of FS-DAM over CMP-SVR for 2F, 4F and 8F. The same fellow-group size is used for both FS-DAM and CMP-SVR for the comparisons. The table shows that the FS-DAM gives 13.13% (2F), 13.56% (4F), 14.38% (8F) improvements in terms of MPKI as compared to CMP-SVR. The improvements in AMAT are 5.69% (2F), 6.11% (4F)

and 6.32% (8F). Improved MPKI and AMAT results 5.32% (2F), 5.95% (4F) and 6.38% (8F) improvements in CPI.

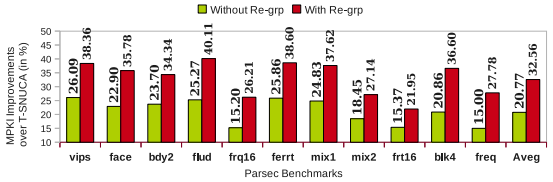


FIGURE 5.8: MPKI improvements in FS-DAM with and without re-grouping. The cache configuration considered for this experiment is **C2**.

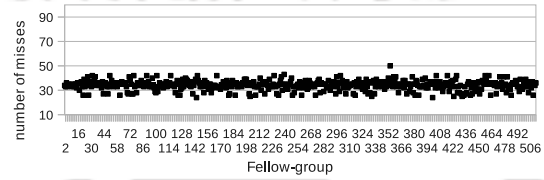


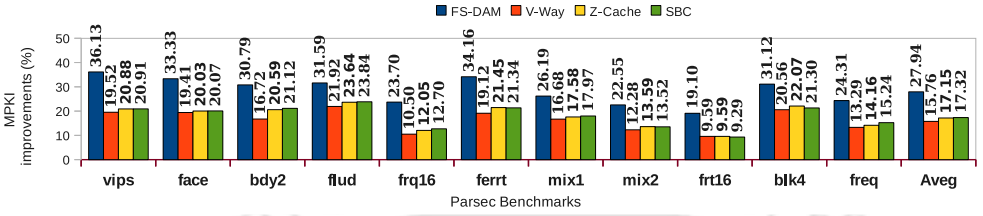
FIGURE 5.9: Load distribution among the fellow-groups of FS-DAM.

The improvement of FS-DAM over CMP-SVR is obtained due to two reasons: (a) better load distribution among the fellow-groups and (b) re-organising the fellow-groups based on the current demand. Figure 5.9 shows the load distribution among the fellow-groups of FS-DAM. The figure shows the comparison for the same benchmark and same bank as was considered for Figure 5.3, to show the non-uniform load distribution among the fellow-sets of CMP-SVR. In comparison with Figure 5.3, we have a uniform distribution in Figure 5.9. The importance of re-grouping in FS-DAM, which is not present in CMP-SVR, can be observed from Figure 5.8. The figure shows the MPKI improvement of FS-DAM with and without re-grouping, while compared to the baseline. It can be observed that the improvement is only 20.77% without re-grouping while using re-grouping the improvement is 32.56%. Note that the improvement shown without re-grouping is also better than CMP-SVR because of the set-load based fellow-group creation.

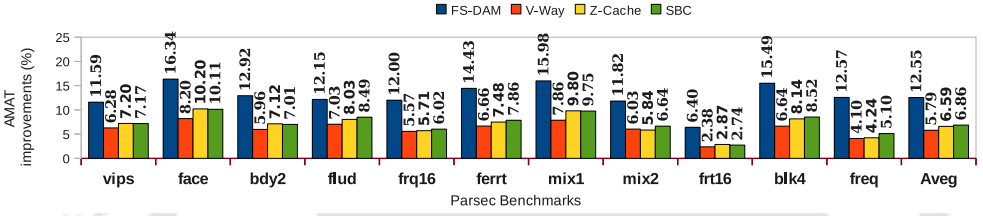
5.6.2.3 Different fellow-group sizes

Higher fellow-group size shows more improvement as the sets get more chances to distribute their load. But as the fellow-group size increases the associativity of

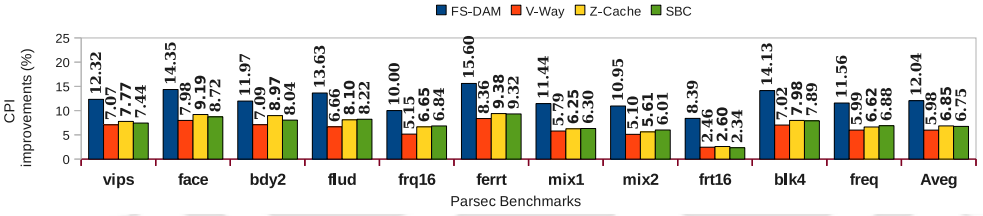
FS-TGS also increases and hence hardware overhead also increases. Section 5.6.5 gives the detail discussion about the hardware overheads of FS-DAM. Since the improvements shown for 4F and 8F in the above comparisons are almost similar, most of remaining comparison shown in this chapter use only 4F to compare FS-DAM with other techniques.



(a) MPKI (**C1**).



(b) AMAT (**C1**).



(c) CPI (**C1**).

FIGURE 5.10: The improvements of FS-DAM over V-Way, Z-Cache and SBC while compared to baseline. The cache configuration is **C1**.

5.6.2.4 Comparison with other techniques

FS-DAM is also compared with V-Way, Z-Cache and SBC. The same cache configuration is taken for all the experiments. For V-Way the Tag to Data Ratio (TDR) is taken as 2, which means the associativity of each set can be increased twice at max. For Z-Cache the replacement candidate is considered as three level.

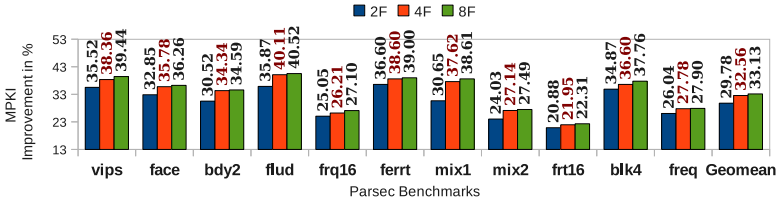
Figure 5.10 shows the improvements of FS-DAM, V-Way, Z-Cache and SBC over baseline. The improvements over MPKI is shown in Figure 5.10(a). It can be observed that the average improvement of FS-DAM over baseline is 27.94% while other techniques improve by 15.76% (V-Way), 17.15% (Z-Cache) and 17.32% (SBC). The AMAT improvements are shown in Figure 5.10(b). The average improvements in terms of AMAT are 12.55% (FS-DAM), 5.79% (V-Way), 6.59% (Z-Cache) and 6.86% (SBC). The improvements in terms of CPI (Figure 5.10(c)) are 12.04%(FS-DAM), 5.98% (V-Way), 6.85% (Z-Cache) and 6.75% (SBC). It can be observed that FS-DAM shows better performance than the other three existing techniques. The performance of V-Way, Z-Cache and SBC are almost similar to CMP-SVR. The load based fellow-group creation and dynamic re-grouping based on requirements gives FS-DAM better improvements. The performance enhancement of FS-DAM over these existing techniques is discussed in Section 5.6.4.

5.6.3 Comparison for C2

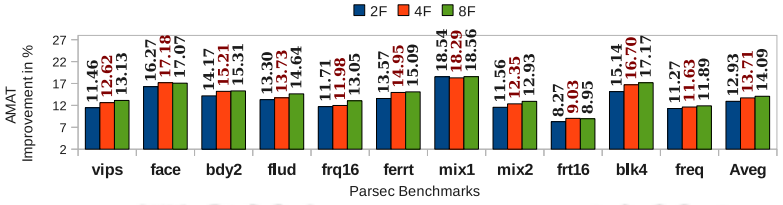
FS-DAM is also compared for the configuration C2 as mentioned in Section 5.6.1. Figure 5.11 shows the improvement of FS-DAM over baseline for C2. Same as C1 (cf. Section 5.6.2) three different fellow-group sizes viz. 2F, 4F and 8F are used for the experiment. The improvements in terms of MPKI is shown in Figure 5.11(a). Compared to the baseline, FS-DAM shows 29.78%, 32.56% and 33.13% average improvements for 2F, 4F and 8F respectively. The AMAT improvements, as shown in Figure 5.11(b) are 12.93% (2F), 13.71% (4F) and 14.09%(8F). Figure 5.11(c) shows the improvements in CPI, which are 12.14% (2F), 13.47% (4F) and 13.79%(8F).

5.6.3.1 Comparison with other techniques

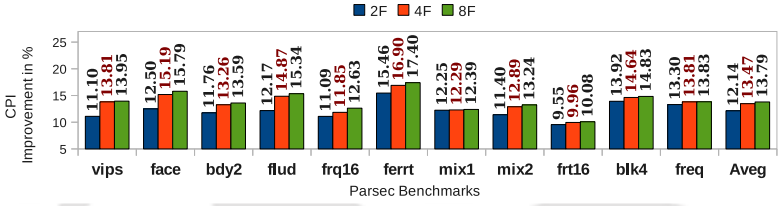
This section compares the FS-DAM (4F) as well the other techniques with baseline. Instead of showing separate comparison of FS-DAM with CMP-SVR (as shown for C1 in section 5.6.2) here all the techniques viz. FS-DAM, CMP-SVR,



(a) MPKI (**C2**).



(b) AMAT (**C2**).

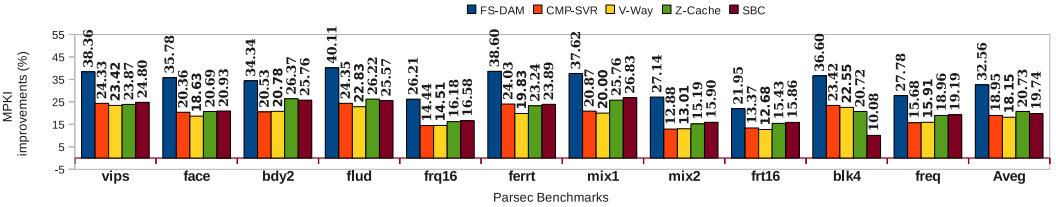


(c) CPI (**C2**).

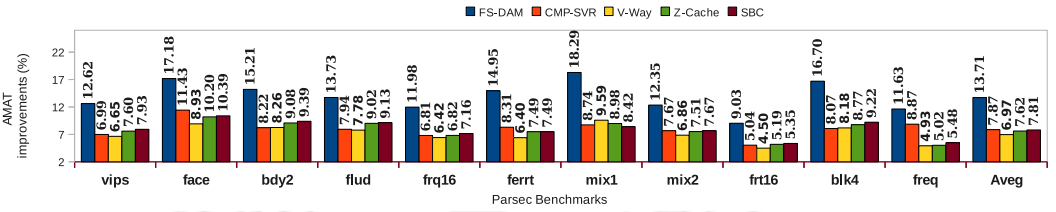
FIGURE 5.11: Improvements of FS-DAM over baseline for the configuration **C2**.

V-Way, Z-Cache and SBC are compared with baseline together. Figure 5.12 shows the improvements in terms of MPKI, AMAT and CPI. The average improvements for MPKI, as shown in Figure 5.12(a) are 32.56% (FS-DAM), 18.95% (CMP-SVR), 18.15%(V-Way), 20.73% (Z-Cache) and 19.74% (SBC). The AMAT improvements (Figure 5.12(b)) are 13.71% (FS-DAM), 7.87% (CMP-SVR), 6.97%(V-Way), 7.62% (Z-Cache) and 7.81% (SBC). The improvements of CPI is based on the improvements of MPKI and AMAT as the other parameters are remain same for all the configurations. Figure 5.12(c) shows the improvements of CPI in all the techniques while compared to baseline. The average improvements are 13.47% (FS-DAM), 7.27% (CMP-SVR), 6.53%(V-Way), 7.66% (Z-Cache) and 7.80% (SBC).

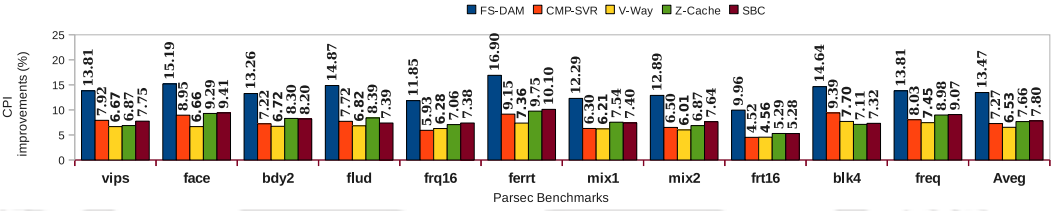
It can be observed that FS-DAM shows better improvements than all other existing techniques used for the experiment.



(a) MPKI (**C2**).



(b) AMAT (**C2**).



(c) CPI (**C2**).

FIGURE 5.12: Improvements in FS-DAM, CMP-SVR, V-Way, Z-Cache and SBC while compared to baseline. The cache configuration is **C2**.

Improvements over	**Improvements in 2MB LLC (C1)**			**Improvements in 4MB LLC (C2)**		
	MPKI (in %)	*AMAT (in %)*	*CPI (in %)*	*MPKI (in %)*	*AMAT (in %)*	*CPI (in %)*
Baseline	26.35	11.23	10.82	32.56	13.71	13.47
CMP-SVR	13.56	06.11	05.95	16.74	6.62	6.15
V-Way	14.23	7.06	06.31	17.56	7.20	07.39
Z-Cache	12.74	06.17	05.28	14.65	06.49	06.21
SBC	12.56	05.94	05.41	14.83	06.27	06.07

TABLE 5.6: Improvements (in %) of FS-DAM over baseline design and the other existing techniques: CMP-SVR (SVR), V-Way, Z-Cache and SBC.

5.6.4 Result analysis

From the above comparisons of both C1 and C2 it can be summarised that FS-DAM gives better performance than the other existing techniques like CMP-SVR, V-Way, Z-Cache and SBC. Table 5.6 shows the improvements of FS-DAM over all the four existing techniques used in this chapter. The improvements shown are the average improvements over all the eleven benchmarks used. Also the improvement for both cache configurations i.e. C1 and C2 are given in the table.

5.6.5 Hardware overheads

In this section the hardware requirements of FS-DAM is compared with the baseline. The hardware overhead of two architectures are compared in terms of storage overhead, area overhead and energy overhead. Area consumed by LLC in different designs are calculated using CACTI 6.0 [2]. The energy consumption of the different components of LLC are calculated separately and then combined together to show the total energy consumption.

5.6.5.1 Energy Overhead

There are two types of energy consumed by each component: static and dynamic. Dynamic energy is consumed during every access of the component, whereas the static energy represents the leakage power of the component. In the case of the baseline, the components are banks and NoC, while in the case of FS-DAM an additional component called FS-TGS is needed. CACTI 6.0 is used to calculate the static energy per unit time and dynamic energy per access for each bank and FS-TGS. These static energy values when multiplied with the total execution time give the total static energy consumption of the component. The dynamic energy from CACTI while multiplied with the total number of accesses gets the total dynamic energy consumed by the component. The values like total execution time and the total number of accesses are obtained from the simulation.

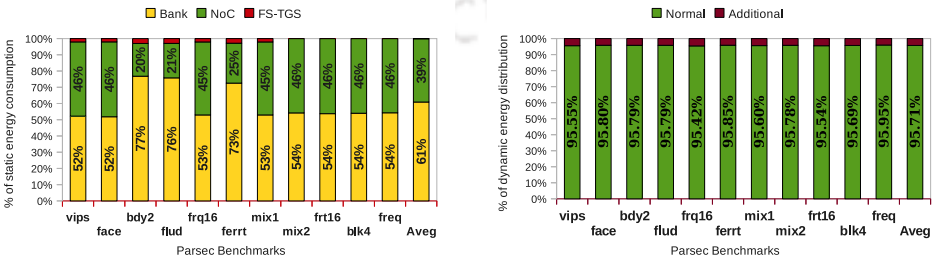
The improvements gained by FS-DAM over baseline is after considering all the cycles required for re-groupings. The static energy required for re-grouping is included in the static energy consumed by the banks during the execution time. But re-grouping requires some additional cache accesses for the adjustment of the reserve blocks as discussed in Section 5.5.4. The number of such accesses cannot be more than twice the number of reserve locations during each re-grouping process. These additional cache accesses slightly increase the total dynamic energy consumption of FS-DAM. Since FS-TGS is always active, the execution time of the bank and FS-TGS is the same. The dynamic energy is depend on the number

	Additional energy consumption required in %										
	vips	*face*	*bdy2*	*flud*	*frq16*	*ferrt*	*mix1*	*mix2*	*frt16*	*blk4*	*freq*	*Average*
Static Energy	3.14	1.97	3.13	2.97	2.20	3.60	2.04	-0.08	0.07	-0.01	-0.01	1.73
Dynamic Energy	2.41	2.62	2.10	2.99	2.23	1.71	2.99	0.48	1.56	2.96	2.00	2.19

TABLE 5.7: The energy overhead of FS-DAM over the baseline design.

of FS-TGS accesses. Since FS-TGS has to be searched for all the block requests from L1, the number of FS-TGS accesses is the same as the number of L2 accesses.

Figure 5.13(a) shows the static energy consumption breakdown for FS-DAM. The energy consumption of three components: bank, NoC and FS-TGS are shown separately. It can be observed that the average energy consumption of FS-TGS is only 2% of the total energy consumption. The NoC consumption of both baseline and FS-DAM remains same as the DAM based policies are internal to each bank. The distribution of dynamic energy consumption in FS-DAM is shown in Figure 5.13(b). In this figure the energy consumption of different components are divided into two groups: normal and additional. Normal means all the components common in both baseline and FS-DAM, i.e. bank, NoC. Additional means the dynamic energy required by FS-TGS and re-grouping. It can be observed that the total dynamic energy consumed by FS-TGS and re-grouping process is only 4.2% of the total dynamic energy consumption by the LLC. The energy required by re-grouping is 0.48% to 0.05% with an average of 0.3%. Table 5.7 shows the additional energy required by FS-DAM over baseline. The static energy overhead of FS-DAM over baseline is only 1.73% while the overhead in dynamic energy is 2.19%.



(a) Static energy.

(b) Dynamic energy.

FIGURE 5.13: Energy consumption breakdown for FS-DAM.

Baseline L2-bank	V-Way overhead (in %)	CMP-SVR overhead (in %)			FS-DAM overheads (in %)		
		F=2	F=4	F=8	F=2	F=4	F=8
256KB, 4-way	15.32	2.27	4.63	9.44	3.11	5.43	10.20
128KB, 4-way	14.8	2.27	4.63	9.44	3.03	5.34	10.11

TABLE 5.8: The storage overhead of DAM based techniques over baseline cache. Rx: x% ways per set reserve for RT, F: fellow-group size.

5.6.5.2 Storage and Area Overhead

The storage overheads of CMP-SVR and FS-DAM depend on the size of their additional tag-arrays. Each additional tag-array entry stores a tag address and a validity bit. Note that both CMP-SVR and FS-DAM need some additional bits for separating the tag of different fellow-sets in their additional tag array. For example, in case of fellow-group size 4, the number of additional bits required for each entry is 2 bits. The number of entries in the additional tag-array is the same for both CMP-SVR and FS-DAM. In addition, FS-DAM needs two tables called **SetMapper** and **SetPointer**. Table 5.8 compares the storage overhead of V-Way, CMP-SVR and FS-DAM. The storage overhead of V-Way are calculated based on the storage required for the forward and backward pointers. From the table it can be observed that the overhead in both CMP-SVR and FS-DAM increases while the fellow-group size increases. This is because of the above mentioned additional bits required to separate the tag of different fellow-sets. The overheads of CMP-SVR remains same irrespective of the bank sizes but varies for FS-DAM. Increasing the banks size while keeping the associativity same increases the number of sets in each bank. Hence the storage required for SetPointer and SetMapper increases.

Cacti 6.0 [2] is used to calculate the area consumption of FS-DAM with different values of F. Table 5.9 shows the percentage overheads of FS-DAM as compared to the baseline design. The overhead of FS-DAM is between 1.6% to 3.5%. In the case of a 256KB 4-way associative L2-bank having R=2 and F=2, the area overhead of FS-DAM is 3.38%.

	L2-bank: 256KB-4way			L2-bank: 512KB-4way		
	F=2	F=4	F=8	F=2	F=4	F=8
Area overhead (in %)	3.380	3.397	3.486	3.128	3.518	3.551

TABLE 5.9: The area overhead of FS-DAM over the baseline design. Rx: x% ways per set reserve for RT, F: fellow-group size.

5.7 Summary

DAM based techniques allow the heavily used sets to distribute their load among the lightly used sets improving the overall cache utilization. The chapter proposed a DAM-based technique, FS-DAM, to demonstrate better performance improvement as compared to other existing techniques. FS-DAM is mainly proposed to resolve the issues of existing CMP-SVR. In CMP-SVR some number of ways (25% to 50%) from each set are reserved for RT and the remaining ways belong to NT. The sets are divided into groups called fellow-groups and a set can use the reserve ways of its fellow sets to increase its associativity during execution. The heavily used sets can use the reserve ways to increase the associativity dynamically.

But though CMP-SVR improves performance the formation of its static fellow-groups restricts the improvements. It has been observed that some fellow-groups have more number of heavily used sets than the other fellow-groups. As a result the cache loads are not uniformly distributed among the fellow-groups. Also the behavior of sets change dynamically: a lightly used set may become heavily used after a number of execution cycles. This chapter studied the behavior of each set of CMP-SVR in detail and proposed FS-DAM, which gave better performance than other DAM based techniques like CMP-SVR, V-Way, Z-Cache and SBC. FS-DAM creates fellow-group based on the current set-loads and the heavily used sets are evenly distributed among all the fellow-groups. Such distribution increases the utilization of the cache and hence improves performance. Full system simulation shows an average of 16.74% and 6.15% improvements, in FS-DAM as compared to CMP-SVR, in terms of MPKI and CPI respectively. The improvement over Z-Cache is 14.65% and 6.21% in terms of MPKI and CPI respectively. FS-DAM

also shows improvement while compared with V-Way and SBC. The static energy overhead of FS-DAM over baseline is just 1.73% while the dynamic energy overhead is 2.19%.

Chapter 6

Dynamic NUCA Framework for Tiled CMPs

In Chapter 2 it has been mentioned that the banks of TCMP are not used uniformly. Some banks are heavily loaded while some others used very lightly. Such utilisation issue is referred as global utilisation issue (cf. Section 2.3.2). Global utilisation can be improved by uniformly distributing the loads among the banks. SNUCA cannot move blocks from one bank to another hence such distributions are not possible with T-SNUCA. DNUCA can move block from one bank to another hence load distribution among the banks is possible in DNUCA based designs.

This chapter proposes a DNUCA based TCMP architecture, called T-DNUCA, to enhance the global utilisation by distributing the loads among multiple banks. The main contribution of T-DNUCA is : (a) distributes loads among multiple banks, (b) heavily used blocks are migrated to the closer banks and (c) better placement policy to reduce LLC access time. Simulation results show that T-DNUCA improves performance by 31.11% and 6.59% in terms of MPKI and CPI respectively as compared to T-SNUCA.

6.1 Introduction

SNUCA has fixed mapping policy and a block always maps to a particular bank. Such a mapping policy makes the block searching process easier and faster. Although SNUCA design is very simple, it has performance issues due to the cache access latencies. When a core requests a block which is far away from the core, then there is no mechanism to bring the block closer. DNUCA allows dynamic mapping of the cache blocks among the banks. The banks are divided into groups called *banksets* and a block can be placed in any bank within a given bankset. Such dynamic nature facilitates migration of heavily used blocks closer to the requesting cores.

DNUCA allows a block to be placed in any bank within a bankset, thus necessitating the entire bankset to be searched before declaring a cache miss. The three basic mechanisms for bankset search are: incremental search, multi-cast search and mixed search [17]. Additional smart searching techniques also exist to minimize the search time [1, 9]. But optimizing bankset searching time is still an open issue [5]. The details regarding DNUCA block search is discussed in Section 2.1.2.1. DNUCA can distribute the load among the banks of same bankset (*peer-banks*). Though most of the DNUCA designs have only migration facility to move blocks from one bank to another, the flexibility of placing a block in any peer-bank makes it possible to move blocks from a heavily loaded bank to another bank.

Both SNUCA and DNUCA described above are well explored for centralised shared CMPs. In Chapter 2 we referred them as C-SNUCA and C-DNUCA respectively. But as mentioned in Section 2.2.3.2 DNUCA is less explored for TCMPs. The TCMP architecture and some common terms related to TCMP are already discussed in Section 2.2.3. For completeness, Figure 6.1 shows the baseline TCMP (cf. Section 3.3.5) . The shared L2 cache is divided into 16 banks and each tile has a bank. The terms "*local bank*" and "*remote bank*" w.r.t. a core represent the local (within the same tile) and remote banks respectively, for that particular core.

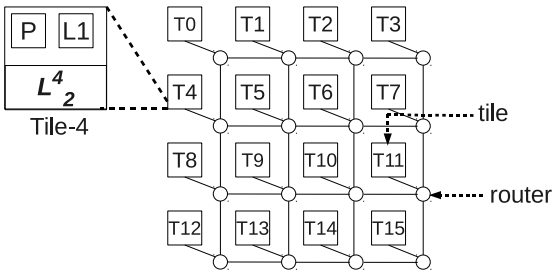


FIGURE 6.1: An example of TCMP. This is a SNUCA based TCMP or T-SNUCA. The TCMP is considered as baseline for our proposed architectures.

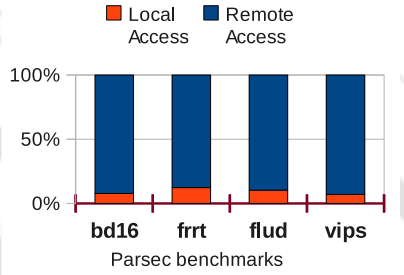


FIGURE 6.2: Distribution of hits among the local banks and remote banks for T-SNUCA.

Due to its distributed shared banks, TCMP can be designed based on the concept of SNUCA and DNUCA. In Chapter 2 we referred them as T-SNUCA and T-DNUCA respectively. T-SNUCA is already implemented and well explored [38, 62] compared to DNUCA based designs for TCMP. Due to its static mapping policy, T-SNUCA has some disadvantages. It has more number of accesses to the remote banks than the local bank as the blocks are statically mapped and cannot be moved (migrated) into a closer bank. The requesting core (tile) has to forward the request to the bank where the block maps. Therefore, T-SNUCA has higher average cache access latency. Figure 6.2 shows the number of remote bank accesses of T-SNUCA as compared to the local bank accesses. The non-uniform load distribution among the T-SNUCA banks is shown in Figure 6.3.

From the above discussion it is clear that a TCMP with lesser remote bank accesses and having block migration capability among the banks can be a better option as compared to T-SNUCA. DNUCA has some advantages over SNUCA and hence, implementing DNUCA concept for TCMP can reduce the problems of T-SNUCA. The dynamic nature of DNUCA can move a block among the banks. Therefore, heavily used blocks can be migrated closer to the requesting tile (core). It would

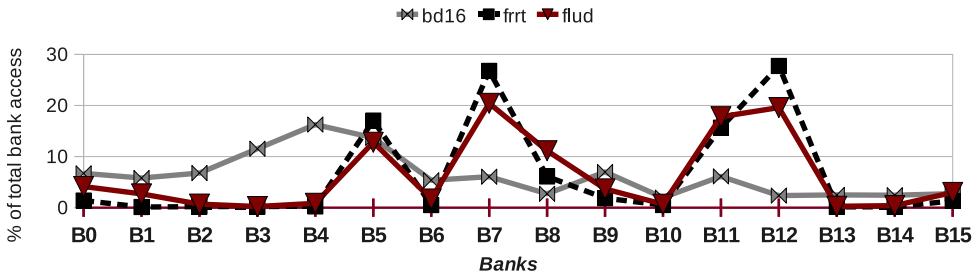


FIGURE 6.3: Comparison of bank usages in T-SNUCA.

also be feasible to spill the victim block (during replacement) of one bank into another bank, thus giving the victim block another chance to remain in the cache. Such interbank movement of victim blocks can distribute the load of a heavily used bank among the other underused banks.

Issues that need to be handled differently in TCMP based DNUCA: Implementing DNUCA for TCMP is not identical to the original DNUCA proposed for centralised CMPs (CCMPs). In TCMP every bank is associated with a core, which is not the case in CCMPs. The core in each tile performs execution and hence the heavily used blocks by the core have to be placed nearer to the core.

Within a bankset, each bank is associated with a core at the same time being accessible by other cores. These nearby cores would benefit loading data in such a bank compared to other banks of a bankset. Now consider gradual migration of a block within a bankset. As this block moves through intermediate tiles, it will evict some blocks from these intermediate nodes. Note that, these evicted blocks may have been cached by other nearby cores.

Placement of a newly fetched block is also an issue. Most of the original DNUCA policy places a newly fetched block into the farthest bank and gradually tries to move it closer [18, 1]. The centralised LLC structure has no core associated with the central banks, by gradual migration they try to settle the shared blocks into the central banks to enable faster access from all the cores. But TCMP has core associated with all the banks, hence settling shared blocks into central bank may remove the important blocks from that bank. Also the DNUCA problems like bankset search and migration control must also be handled properly such that

they should not nullify the advantages gained over T-SNUCA. Using smart search policies to speedup the bankset searching are also expensive in terms of area, storage and energy consumption [5].

Our proposed T-DNUCA uses a hybrid search policy to search a block within the bankset. In its first attempt it only searches the bank within the bankset that is closest from the requesting core. If there is a miss then search request is multicast to the remaining banks of the bankset. Migration is allowed to move heavily used bank directly into the closest bank of the requesting core. The newly fetched blocks are placed to the closest bank of the requesting core. Earlier experiment [1] shows that most of the blocks requested by a core are private to the requesting core even though the application running is a shared memory based application. Placing the incoming blocks to the nearest bank reduces the average bank access time. Also since most of the cache hits can be served by the closest bank the multicast search requirement is less and hence one can afford not using any smart searching techniques. The requirement of migration is also less due to the same reason.

The rest of the Chapter is organized as follows. The detail description of proposed T-DNUCA is given in the next Section. Experimental analysis is given in Section 6.3. Section 6.4 gives the summary of the Chapter.

6.2 T-DNUCA

The proposed T-DNUCA is a TCMP with DNUCA based data management policy. But since each tile in TCMP has a core, the data management policy of T-DNUCA is not exactly same as C-DNUCA. A 4x4 (16 tiles) T-DNUCA is shown in Figure 6.4. Same as other TCMP, each tile has its own processing core, L1 cache and a slice (bank) of distributed shared L2 (LLC) cache.

As mentioned in Section 2.2.3 each tile is given a number starting from 0. The associated bank with each tile is denoted as L_2^n where $0 \leq n < N$; N means total

number of banks. For example, the bank of tile-0 is referred as L_2^0. Each bank is a set associative cache and all the banks are equal in size.

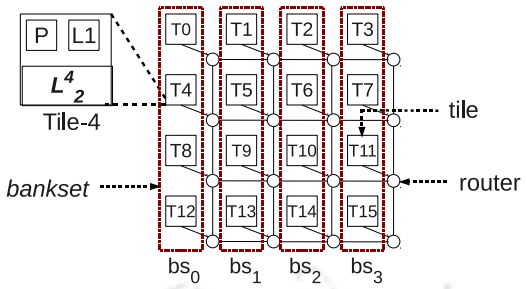


FIGURE 6.4: An example of T-DNUCA.

The banks are grouped into multiple banksets same as in original DNUCA. For example in Figure 6.4, the entire T-DNUCA is divided into four banksets, e.g., bs_0, bs_1, bs_2 and bs_3. The mapping of banks with the banksets used for this Chapter are:

$$bs_0 = \{L_2^0, L_2^4, L_2^8, L_2^{12}\}$$

$$bs_1 = \{L_2^1, L_2^5, L_2^9, L_2^{13}\}$$

$$bs_2 = \{L_2^2, L_2^6, L_2^{10}, L_2^{14}\}$$

$$bs_3 = \{L_2^3, L_2^7, L_2^{11}, L_2^{15}\}$$

The mapping of banks with the bankset can be done in a variety of methods. The mapping shown above is a column-wise mapping where each column (from 4x4 TCMP) is considered a bankset. However it can be row-wise or distributed throughout the chip. The number of banksets can also vary. Rest of the chapter assumes a T-DNUCA having four column-wise banksets as shown in Figure 6.4.

6.2.1 Some Common Terms Used for T-DNUCA

Some of the terms mentioned here are already mentioned in Chapter 2.

local-bank : The bank associated with each tile is called the *local-bank* for that

particular tile. For example L_2^0 is the local-bank for tile 0.

***home-bank* :** For a given tile (or core) the bank of a bankset which has the least Manhattan distance from this tile (or core) is called the *home-bank*. For example, the home-bank of tile-0 in bs_1 is L_2^1, whereas in bs_0 the home-bank is L_2^0 (itself).

peer-bank: All the banks within a particular bankset are called *peer-banks*. For example, all the banks associated with bs_0 are peer-banks.

6.2.2 Mapping Policy

In T-DNUCA a block maps to a bankset and not to a particular bank (as is done in T-SNUCA). The block can be placed in any bank within the bankset but not outside the bankset. For example, if a block is mapped to bs_0 then it can be placed in any of the four banks (L_2^0, L_2^4, L_2^8 and L_2^{12}) associated with bs_0. The placement of a block within a particular bank is same as traditional block placement for set associative cache and always maps to a fixed set based on the index bits. The mapping policy is explained in Figure 6.5 with a 32 bit block-address.

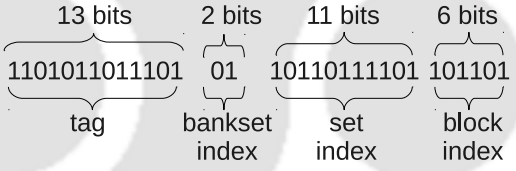


FIGURE 6.5: A 32-bit block address showing mapping policy of T-DNUCA. The cache size is considered as 8MB, hence each bank is 512KB. Associativity of each bank is 4 and the block size is 64 bytes. *set index* means the index required to identify a set within a bank (each bank has 2048 sets). *bankset index* is to identify the bankset.

In the figure the most significant 11 bits are tag address. The next four bits are used for bank selection in T-SNUCA (cf. Figure 2.4). Since in T-DNUCA blocks are mapped to bankset instead of bank only two bits (out of 4) are required as bankset-index. The remaining two bits become part of the tag. The next bits are the index-bit for the banks, based on these bits each bank maps a block to a particular set, same as traditional set-associative cache. Note that the mapping policy in T-DNUCA only decides which bankset the block can be placed and not any particular bank within the bankset.

6.2.3 Handling L1-cache Miss in T-DNUCA

On a cache miss in any L1 cache the request for the block (say B) has to be forwarded to L2 (LLC). The requested block can be in any bank within the selected bankset. The bankset where the block B maps is selected based on the bankset-index. The situation is same as DNUCA and the entire bankset needs to be searched to decide hit or miss. T-DNUCA follows a different method to search a block within a bankset as compared to DNUCA.

When a request (read or write) is sent from L1 to L2, the requesting L1 bank controller (say of tile-j) first decides the bankset (say bs_i) for the requested block (B) and then sends the request to the home-bank (say $L_2^{h_{ji}}$) of tile-j in bs_i. After reaching the request to the home-bank ($L_2^{h_{ji}}$), the block is first searched only in $L_2^{h_{ji}}$, same as a conventional set associative cache. If the block is found in $L_2^{h_{ji}}$ then it is a hit (called *local-hit*) and the block is sent to the requesting L1 cache. But if the requesting block is not found in $L_2^{h_{ji}}$ then a multicast search request is sent to all the peer-banks of $L_2^{h_{ji}}$. If any of the peer-bank has the block then it sends the block directly to the requesting L1 and also sends a *block-found* message to $L_2^{h_{ji}}$; otherwise it sends a *block-not-found* message to $L_2^{h_{ji}}$. After sending the multicast search request, $L_2^{h_{ji}}$ waits for the response from its peer-banks. If it receives *block-found* message from any of the peer-bank then it is a hit (called remote hit) and ignores the remaining messages. Otherwise, it has to wait for the *block-not-found* message from all its peer-banks to declare a cache miss. If all the peer-banks send *block-not-found* message then the block is not in the cache (L2) and needs to be fetched from the main memory.

6.2.4 Initial Block Placement

Most of the block accesses by a core in CMP are private to only that particular core [1]. We use this information and place the newly incoming blocks from the main memory into the home-bank (of the requesting core) with an aim to reduce the access time for such blocks. For example, if the core in tile 7 requests for a

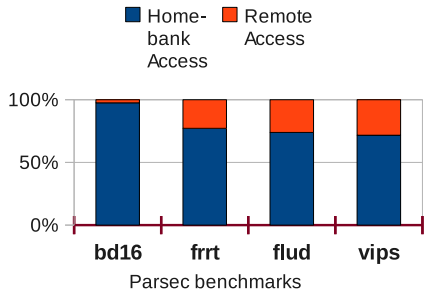


FIGURE 6.6: Distribution of hits among the home-banks and the remote banks of T-DNUCA.

block which maps to bs_0 and the block is not in the cache then the block will be fetched from the main memory and placed in L_2^4 (the home-bank of tile 7 in bs_0). Also, placing these blocks into home-bank makes the searching policy perform better as most of the hits will be local hits and will not require to go for multicast-based bankset search. Figure 6.6 shows how local hit dominates the remote hits in T-DNUCA. Multicast bankset search is faster but expensive in terms of energy consumptions. That was the reason that the previous works tried to avoid it by using other smart search policies like partial tag comparison [18]. But such smart search policies have also disadvantages in terms of storage, coherence communication etc [5]. Placing the newly incoming block initially into home-bank and searching home-bank first, makes it possible to afford the occasional expensive multicast search for T-DNUCA.

6.2.5 Block Migration

Migrating a heavily used block closer to a requesting core can reduce the access latency. In DNUCA, a block is gradually migrated towards the requesting core. In T-DNUCA each bank is associated with a core and is home-bank for more than one core. Gradual migration may hamper the benefit gained by loading blocks directly in home-bank. Therefore in T-DNUCA, migration policy brings the heavily used block directly into the home-bank of the requesting core.

T-DNUCA uses information about consecutive access to invoke migration of a block and prevents the ping-ponging effect [5, 1]. We consider two consecutive

accesses by a core in order to migrate a block. But migration is an expensive operation as the block movement consumes network bandwidth and energy. Also mechanism has to be maintained so that no false cache-miss occurs for the block while it is in transition. T-DNUCA allows migration but the number of migrations is much lesser due to the placement policy (as discussed above).

6.2.6 Replacement

Replacement plays an important role in the performance of LLC. In T-DNUCA, there are two types of replacements: (a) Local replacement and (b) Cascading replacement. Local replacement occurs within each bank locally. The purpose of local replacement is to choose a victim block from a particular bank. Cascading replacement tries to place the victim block of local replacement into another peer-bank. For example, assume that local replacement evicts a block V from the bank L_2^0 of bankset bs_0. Now instead of completely removing V from the cache, cascading replacement tries to spill it into another peer-bank. So if the cascading replacement feature is turned on, the block V is forwarded to the nearest peer-bank L_2^4. On reaching L_2^4, the block is placed into L_2^4 if any empty place is available, otherwise V is written in place of V' of L_2^4. Eviction of V' for placing V depends on some conditions. The simplest condition is that the V will replace V' if V' is older than V. To determine this the reuse counts of V and V' are compared. The block with less reuse is selected as the victim.

If the condition fails then V can be forwarded to the next neighbor peer-bank or removed from the cache. The number of times this can take place depends on a parameter called the *cascading number*. If the cascading number is 1 then V has to be removed from the cache (considering the fact that it fails to insert in L_2^4). Otherwise, if the cascading number is more than 1 then V is forwarded to the next peer-bank (L_2^8). But when V is placed into L_2^4 by replacing a new victim V' then the new victim V' is forwarded in place of V to the next peer-bank. Note that, forwarding is only possible if the cascading number is more than 1. The process continues until the victim block reaches the last bank according to the cascading

number or it gets placed in a free location of some peer-bank. Cascading number cannot be more than the number of peer-banks that a bank has. The cascading replacement may increase the complexity of the system but the entire process is done in background (when the newly requested block is fetched from the main memory) hence does not affect the performance.

6.2.7 Different design varieties (migration and replacement)

Cascading replacement helps a heavily used bank to use another bank (within the bankset) for storing its blocks. It is different from migration, as migration brings a heavily used block nearer to the requesting core while cascading moves it away from the core. But the benefit is, it remains in the cache and next access may be a hit. This is highly beneficial when the banks are not used uniformly. Figure 2.11 shows that for most of the applications, banks are not used uniformly. Hence cascading a local victim block into another peer-bank reduces the cache misses and improves the performance. For small size applications or application with lower temporal locality, cascading may not show too much improvements because the cascaded blocks may not be reused in future. But it makes the replacement policy as global replacement policy and even in case of uniform bank access, it can improve performance by removing the global victim instead of local victim. Migration improves the cache performance by reducing the access latency of heavily used blocks.

Both migration and cascading replacement are independent and one can be used without using another. Both cascading replacement and migration helps to improve the global utilisation of the entire LLC.

6.3 T-DNUCA: Experimental Analysis

The main motive of this work is to implement the concept of DNUCA in TCMP to improve the performance of T-SNUCA. Therefore T-DNUCA is compared here

Component	Parameters
No. of tiles	16
Processor	UltraSPARCIII+
Block size	64bytes
L1 I/D cache	64KB, 4-way
Total LLC (L2) size	4MB
bank size	256KB 4-way
Memory bank	1GB, 4KB/page
bank access Latency	6
Network latency is handled by GARNET [99], a module of GEMS.	

TABLE 6.1: System Parameters.

with T-SNUCA. Also as mentioned in Section 6.2.7, T-DNUCA can be configured in different ways in terms of migration and cascading number. But all the configurations have same bankset distribution as shown in Figure 6.4. Also the replacement policy chosen is LRU for both local and cascading replacement.

In order to evaluate the proposed T-DNUCA, we performed simulations by running benchmarks on a multi-core simulator GEMS [85] with the help of SIMICS [82], a full-system functional simulator. The configuration details of the processor, cache memory and main memory used for both T-DNUCA and T-SNUCA is given in Table 6.1. Nine Parsec benchmarks are used for the experiment: **vips**, **mx1**, **frrt**, **frq**, **body**, **mx2**, **flud**, **mx3** and **blck**. The details regarding the benchmarks are discussed in Section 3.3.2

MESI-CMP based protocol is used for both T-SNUCA and T-DNUCA. We implemented the MESI-CMP based cache controller for T-DNUCA in GEMS. The controller is responsible for all the data management policies of T-DNUCA, e.g. address mapping, bankset selecting, bankset searching, cascading replacement, block migration and initial block placement. Maintaining coherence among the L1 caches is also the responsibility of the implemented T-DNUCA controller.

Various configurations of T-DNUCA based on migration and cascading replacement are used to analyze the performance. The configurations are:

M0C0 : Without migration and no cascading replacement.

M0C1 : Without migration and cascading number as 1.

M0C3 : Without migration and cascading number as 3.

M1C1 : With migration and cascading number as 1.

M1C3 : With migration and cascading number as 3.

Whatever be the combination of migration and cascading number the initial placement policy is always a plus point for T-DNUCA.

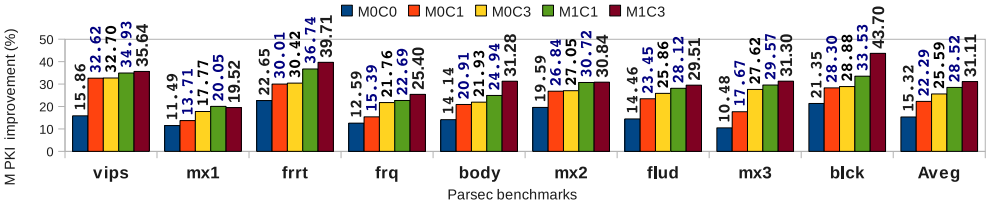
6.3.1 Comparison with T-SNUCA

T-DNUCA having different configuration (as mentioned above) is compared with T-SNUCA. In some graphs of this section, the Y-axis values of bars are shown above the bar. It shows the improvement (in %) of the corresponding T-DNUCA configuration.

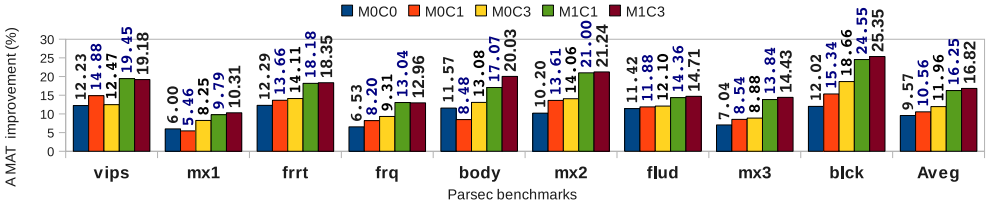
Figure 6.7(a) shows the *MPKI* comparison of T-DNUCA with T-SNUCA. The T-DNUCA having configuration M1C3 gets 19.5% to 43.7% reduction in *MPKI* as compared to T-SNUCA. On average, M1C3 improves *MPKI* by 31.11%. Similarly, the configurations M0C0, M0C1, M0C3 and M1C1 improve MPKI by 15.32%, 22.29%, 25.59% and 28.52% respectively.

Migration cannot directly increase or decrease the number of misses in T-DNUCA. But from the Figure 6.7(a), it can be observed that the MPKI of M1C1 and M1C3 are less than the other three configurations. This is because migration helps to reduce the average memory access time (AMAT). Reduced AMAT means more number of instructions can be executed and hence less MPKI.

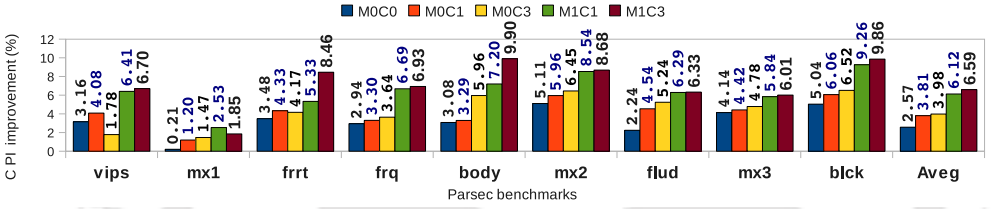
The cascading number 1 is sufficient for most of the benchmarks, hence the average improvements in M1C1 and M1C3 are almost same. M1C3 is beneficial when the load distribution among the banks is extremely non-uniform and a heavily loaded bank needs more banks to distribute its load. In case of *frrt*, *body* and *blck*, M1C3 gives better performance than M1C1. The importance of cascading replacement can be observed by comparing the improvements of M0C0 with other four. The



(a) MPKI improvement (in %) of T-DNUCA over T-SNUCA



(b) AMAT improvement (in %) of T-DNUCA over T-SNUCA



(c) CPI improvement (in %) of T-DNUCA over T-SNUCA

FIGURE 6.7: The performance comparison of T-DNUCA with T-SNUCA. Improvement of five T-DNUCA configurations over T-SNUCA is shown. "Aveg" means Average improvement.

absence of cascading replacement in M0C0 cause less improvements (in MPKI) as compared to the other configurations having cascading replacement. Similarly the importance of migration can be observed by comparing M1C1 and M1C3 with the other configurations.

T-DNUCA improves MPKI for three reasons: (a) cascading replacement, (b) migration and (c) smart placement. Cascading replacement helps to improve the global utilisation of the LLC while migration and smart placement reduces the cache access time. Reduction in MPKI also reduce the average memory access time (AMAT). Better utilisation and reduced MPKI means the LLC can serve more block requests without contacting main memory, hence improves AMAT. Figure 6.7(b) shows the improvements of T-DNUCA over T-SNUCA in terms of AMAT. The average improvement of T-DNUCA in terms of AMAT are 9.57% (M0C0), 10.56% (M0C1), 11.96% (M0C3), 16.25% (M1C1) and 16.82% (M1C3).

In some exceptional cases, improved MPKI does not relatively improve AMAT. For example in *blck*, the MPKI improvements of M1C3 is 10% better than M1C1 but both the configurations have almost similar AMAT. This is because of the higher number of remote bank accesses.

CPI comparison is shown in Figure 6.7(c). It shows that T-DNUCA having configurations M0C0, M0C1, M0C3, M1C1 and M1C3 outperforms T-SNUCA by 2.57%, 3.81%, 3.98%, 6.12% and 6.59% respectively. The improvement of CPI is not always in the rhythm of *MPKI*. In some cases even after a better *MPKI* improvement the CPI improvement is less. The reason behind this is the higher number of remote bank accesses. When the number of remote hits are more and migration is not allowed, the CPI does not improve. As the basic property of DNUCA, allowing migration improves CPI.

In all cases, cascading replacement improves CPI. But the improvements with high cascading number (3) is not significantly better than a lower cascading number (1). With high cascading number, the evicted blocks get distributed among all the peer-banks and may increase the access latency in some cases. But on average T-DNUCA with migration and cascading replacement outperforms T-SNUCA as well as the other T-DNUCA configurations.

The average improvements of the different configurations of T-DNUCA over T-SNUCA are shown in Table 6.2. The improvements are shown in terms of MPKI, AMAT and CPI.

Improvements	**T-DNUCA configurations**				
in terms of	M0C0	M0C1	M0C3	M1C1	M1C3
MPKI	15.32	22.29	25.59	28.52	31.11
AMAT	9.57	10.56	11.96	16.25	16.82
CPI	2.57	3.81	3.98	6.12	6.59

TABLE 6.2: Average improvements (in %) of T-DNUCA over T-SNUCA.

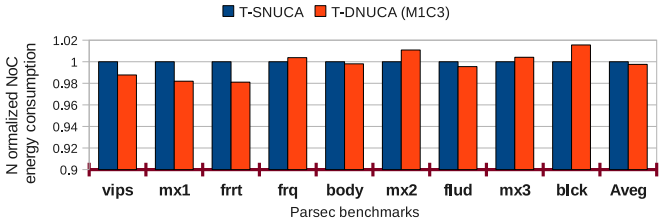


FIGURE 6.8: The normalized network energy comparison of T-DNUCA with T-SNUCA. "Aveg" means Average improvement.

6.3.2 Hardware Requirements

Network energy: The smart placement policy of T-DNUCA results more local-hits, hence reduces the on-chip communications required for serving a block request. Also migrating a block to home-bank reduces the communication cost. But cascading replacement and migration require a block to be moved through the NoC. The multicast search request to access a block requires more on-chip communications. Hence the advantage (in terms of network consumption) gained by better placement and global utilisation is being nullified. Figure 6.8 shows the network energy consumption of T-DNUCA and T-SNUCA. It can be observed that T-DNUCA consumes almost same network energy as compared to T-SNUCA. But it gives better performance in terms of MPKI, AMAT and CPI. The block movements in T-DNUCA happens in background when the execution of other blocks continues.

Total energy consumption: The total energy consumptions of T-DNUCA is almost same as T-SNUCA. Even after reducing MPKI the total energy consumption remains same because of the domination of static energy as the number of components in both T-SNUCA and T-DNUCA are same. The detail description about the energy consumption of T-DNUCA is given in the next chapter.

Storage overhead: The storage requirements of both T-DNUCA and T-SNUCA can be considered as same. T-DNUCA may require some additional storage to store the migrated/cascaded blocks temporarily. But such overheads are negligible.

6.3.3 Comparison Among the Different T-DNUCA Configurations

The performance gained by M1C3 is not much higher than M1C1. Table 6.3 shows the percentage of blocks permanently removed from the cache due to cascading replacement. It shows that on average, M1C3 removes only 8.65% lesser blocks than M1C1. But M1C3 distributes these blocks among all the peer-banks and increases the access time. Therefore the benefits gained by higher cascading number is spent on higher communication cost. Also such block distribution increases the network traffic and hence consumes more energy. Hence we suggest that M1C1 is the best configuration to run T-DNUCA.

Benchmarks	M1C1	M1C3	Diff. between M1C1 and M1C3
frrt	34%	11%	23.49
blck	93%	90%	3.33
mx1	80%	67%	12.77
mx2	94%	88%	5.62
Average	–	–	**8.65**

TABLE 6.3: The percentage of evicted blocks permanently removed from cache (L2) during the cascading replacement process.

6.4 Summary

In this chapter we proposed a dynamic NUCA design for tile based CMPs. In case of centralised CMPs the DNUCA performs better than SNUCA with smarter block searching techniques [5]. Static NUCA (SNUCA) has a fixed address mapping policy whereas dynamic NUCA (DNUCA) allows blocks to relocate nearer to the processing core at runtime. SNUCA is well understood and explored for tiled CMPs whereas the same is not the case for DNUCA. Since there are architectural differences in centralised CMPs and TCMPs, the T-DNUCA design is different than the original DNUCA design. Each tile in TCMP is associated with a core which is not the case in centralised CMPs.

Different configurations of T-DNUCA based on migration and cascading replacement are compared with T-SNUCA and are found to outperform T-SNUCA. With migration enabled and maximum cascading number (M1C3) the T-DNUCA improvements over T-SNUCA are 31.11% and 6.59% in terms of *MPKI* and *CPI* respectively. The proposed migration and placement policies lead to considerable improvements in performance.

Chapter 7

Mechanism for Block Distribution and Fast Searching in T-DNUCA

Our first DNUCA based TCMP (T-DNUCA) is discussed in the previous chapter. T-DNUCA improves performance by better global utilisation and smart initial placement policy. This chapter proposes a new DNUCA variants for better utilisation and access time, called TLD-NUCA. It improves the performance of T-DNUCA by better global utilisation and reduced on-chip communication.

7.1 Introduction

T-DNUCA discussed in the previous chapter has the following DNUCA properties:

- It divides the banks into multiple banksets; a block can be placed in any bank within a particular bankset.
- A heavily used block can be migrated from one bank to another within the same bankset.

In addition to these two basic DNUCA properties T-DNUCA can distribute the loads of a heavily used bank to the other underused banks. Also smart placement

policy has been used to reduce the hit time. For a particular core, C, the closest bank from a bankset (bs) is called the *home-bank* for C in bs. Here the closest means minimum Manhattan distance. A core has a home-bank in each bankset. If C needs to access a block which belongs to bs then C must first send the request to its home-bank in bs. If the block is found in the home-bank then it is sent to C, otherwise the request is multicast to all the remaining banks of bs. If any of the bank has the block then it sends the block directly to C. Otherwise, a miss occurs and home-bank brings the block from the main memory before sending it to C. Though the two-level access requires some additional time but due to the use of efficient placement and migration policy the average hit time of T-DNUCA is less than T-SNUCA. Both these polices try to reduce the need for the two-level search.

Although T-DNUCA improves performance as compared to T-SNUCA it can be further improved on the following aspect:

- The communication with the home-bank is mandatory in T-DNUCA. Since all the communications are done through the NoC, minimizing mandatory communication can reduce average hit time.
- The multicast search used is expensive in terms of energy consumption.
- The loads can be distributed among the banks within the bankset but there is no mechanism to balance the loads among multiple banksets.

This chapter proposes another technique called TLD-NUCA for better LLC utilisation and performance. TLD-NUCA considers a single bankset for the entire LLC for better utilisation of the banks. Single bankset means a block can be placed in any bank within the LLC. The expensive multicast search of T-DNUCA is replaced with a centralised tag lookup mechanism. TLD-NUCA has an additional Tag Lookup Directory (TLD) placed in the central tiles. This TLD stores the information about all the blocks in the LLC. Each entry in TLD stores the tag address of a block and the *bank-id* where it currently resides. Single bankset makes the local-bank as home-bank hence the mandatory communication required

in T-DNUCA between the requesting core and home-bank is not required. Experimental analysis found that TLD-NUCA gives better performance than T-SNUCA as well as T-DNUCA.

As compared to T-DNUCA the average improvements of TLD-NUCA are 6.5% and 16.8% in terms of CPI and MPKI respectively. The improvements compared to baseline (T-SNUCA) are 13% (CPI) and 42.9% (MPKI). The energy consumption of TLD-NUCA as compared to the baseline is 3.08% more while the storage consumption is 10.5% more. Though TLD-NUCA has additional energy requirements its improved performance reduces the EDP by 11%, compared to the baseline.

The purpose of global utilisation enhancement and DNUCA based TCMP are already discussed in Chapter 2 and Chapter 6. Hence we start this chapter with a motivational discussion about why TLD-NUCA is required design after T-DNUCA. The rest of the chapter is organized as follows. The motivation behind TLD-NUCA are discussed in next section. Section 7.3 discuses the proposed TLD-NUCA. Experimental analysis of TLD-NUCA are given in Section 7.4. Finally, summary of the chapter is given in Section 7.7.

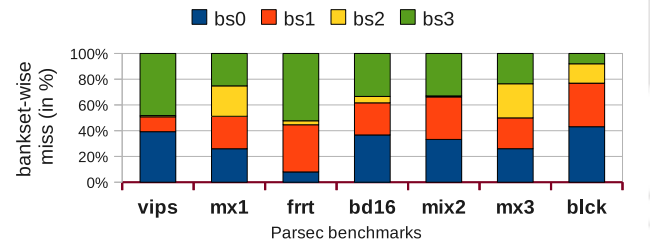


FIGURE 7.1: Load distribution among the banksets of T-DNUCA. The T-DNUCA architecture considered for this experiments has four banksets: $bs0, bs1, bs2$ and $bs3$.

7.2 Motivation

In T-DNUCA the loads can be distributed among the banks within the same bankset but a bankset cannot share loads with other banksets. Figure 7.1 shows an example of how the loads are distributed among the banksets in T-DNUCA.

Benchmarks	vips	frrt	body	flud	freq	blck	Aveg
bpbs=4	1.25	1.26	1.18	1.26	1.34	1.22	1.25
bpbs=8	0.75	0.73	0.76	0.76	0.78	0.77	0.75

TABLE 7.1: Mandatory communications required in T-DNUCA in terms of hop count, to communicate between the requesting core and its home-bank. The benchmarks are taken from Parsec benchmarks suite [3]. "Aveg" means average of all the benchmarks and "bpbs" means banks per bankset.

From the figure it can be observed that the loads are not uniformly distributed. Therefore a better load balancing technique can increase the utilisation factor of T-DNUCA. Also it is mandatory for T-DNUCA to first communicate with the home-bank for every LLC request (cf. Section 6.2.3). Network communication is required for the request to reach the home-bank, as the home-bank is not always the local-bank. Table 7.1 shows the average home-bank communication cost in terms of hop counts. Minimizing these costs can reduce the search time.

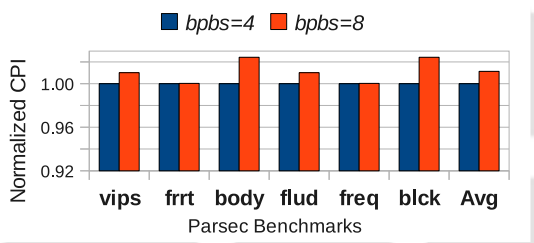


FIGURE 7.2: Normalized CPI comparison of T-DNUCA having *bpbs* as 4 and 8. The results are normalized with *bpbs*=4. "Aveg" means average of all benchmarks.

In T-DNUCA the multicast search is not required always. This is on account of the smart placement policy. However this cost may increase with the increase the number of banks per bankset (*bpbs*). Higher *bpbs* increases the network communication which results in higher energy consumption by the network. The block searching time also increases with higher *bpbs*. Though larger bankset can utilise the cache better than a smaller bankset, it cannot improve the performance of T-DNUCA. Figure 7.2 shows the performance comparison of two T-DNUCA architectures having *bpbs* as 4 and 8. For T-DNUCA having *bpbs*=8, the average CPI degradation is 1.13% as compared to *bpbs*=4. Bankset with higher *bpbs* must improve bank utilisation and reduce the number of misses in LLC, but due to the additional delay in bankset searching, such reductions are not able to improve

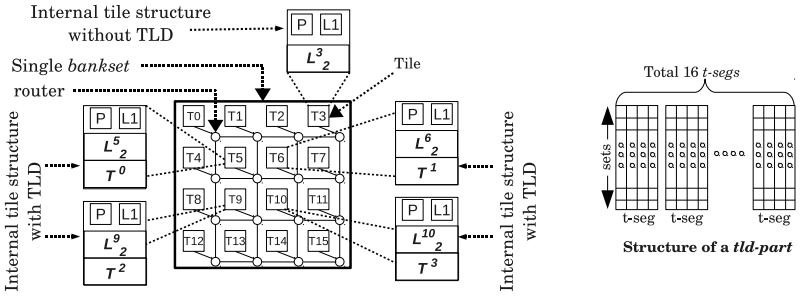


FIGURE 7.3: An example of TLD-NUCA having 4 *tld-parts* placed in Tile 5, 6, 9 and 10. All other tiles are normal tiles (without TLD).

performance. Hence a better search technique has to be implemented to use larger banksets. These all factors have motivated us to design a TCMP architecture having better LLC utilisation and performance than T-DNUCA. The proposed architecture is called TLD-NUCA.

7.3 TLD-NUCA

TLD-NUCA is proposed to improve the utilisation of LLC with larger banksets. In case of a 16-core TCMP we consider all the banks to form one bankset. Since all the banks belong to the same bankset, the home-bank and local-bank for each core is the same. Hence no additional communication cost is required between the requesting core and its home-bank. Also the loads can be distributed among all the banks within the LLC. Multicast search is replaced with a centralised directory search. The tag array of all the banks are duplicated in a centralised directory called Tag Lookup Directory (TLD). The TLD has the information of all the blocks currently present in the LLC. When a block is not found in the local-bank the TLD is contacted to get the *bank-id* where the block resides. Given a block address the TLD gives the *bank-id* where the block currently resides. If the block is not in the LLC then TLD also has no entry for the block and hence it is a cache-miss. The TLD-NUCA considered for this section have 16-cores (tiles). The scalability of TLD-NUCA for more number of tiles is discussed in Section 7.5.

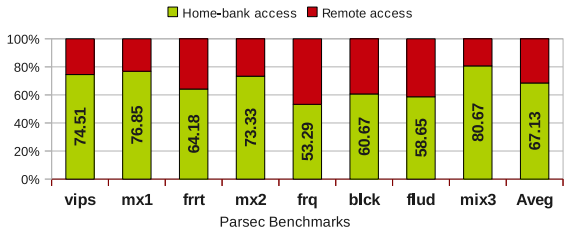


FIGURE 7.4: The LLC access distribution of TLD-NUCA among local bank and remote banks.

For handling multiple requests at a time the TLD is divided into multiple parts (*tld-parts*). Figure 7.3 shows a TLD-NUCA having TLD divided into 4 tld-parts placed in Tile-5, Tile-6, Tile-9 and Tile-10. This thesis assumes that the tld-parts are placed with some central tiles. Such TLD placement policy may convert the central tiles as hotspot of the chip. However, on account of the smart placement policies (as used in T-DNUCA) the number of TLD requests is less and can be managed without making the tiles as hotspot. Figure 7.4 shows the distribution of local and remote accesses in TLD-NUCA. On average 67% requests are local. Hence the probability of getting multiple simultaneous request on the same tld-part is minimised. The complete energy breakdown of TLD-NUCA is given in Section 7.4.6. The tld-parts are referred to as $T^y; 0 \leq y < K$, where K is the total number of tld-parts.

7.3.1 Mapping Policies

The mapping policy of TLD-NUCA is almost the same as T-DNUCA (cf. Section 6.2.2). The only difference is that TLD-NUCA does not require any bit for bankset selection as there is a single bankset. Each bank behaves as an independent cache and uses the conventional set indexing method to map a block to a particular set within the bank. The LLC in TLD-NUCA can be assumed as a highly associative cache whose ways are distributed among the banks. For example a 4MB 64-way associative LLC can be distributed into 16 banks, each having 4 ways such that the size of each cache bank is 256KB. A block belonging to set i of an unpartitioned cache can be placed in any bank because each bank has 4 ways for the set i. Placement of a block within a bank is based on set indexing.

The structure of TLD is like a two dimensional array or a set associative array where each entry stores a tag address (t_g) and the *bank-id* where the block (having t_g) is currently placed. The content stored in each entry of TLD is called a *tld-block*. Each set in TLD represents a set in the unpartitioned cache i.e. the associativity of TLD is the associativity of each bank multiplied by the number of banks. To support high throughput the sets in TLD are distributed into multiple *tld-parts*. To avoid the overheads of high associativity, each tld-part is divided into multiple segments (*t-segs*) as done in [12]. Figure 7.3 shows the structure of a tld-part. In the figure the 64-way associative *tld-part* is divided into 16, 4-way associative *t-segs*. All the t-segs (in a particular tld-part) are searched in parallel hence the search time of TLD is same as the tag-array search time of each bank. Though the search time is less the TLD requires some additional hardware to manage all the tld-parts and t-segs. The details regarding the hardware overheads of TLD are discussed in Section 7.4.6.3.

There is a one-to-one mapping with the TLD entries and the bank ways. Considering the associativity of each bank as M and total banks as N the associativity of TLD is $M \times N$. Given a location $(i', j') : 0 \leq i' < S, 0 \leq j' < M$ in a bank $L_2^k; 0 \leq k < N$, where i' is set index, j' is way index and S is the total number of sets, the corresponding location in TLD is:

$$t_r = i' : 0 \leq t_r < S$$

$$t_c = (k \times M) + j' : 0 \leq t_c < (N \times M)$$

Where t_r is the TLD set index and t_c is the TLD way index. The partitions of TLD are done setwise i.e. the sets are distributed in multiple parts and the associativity remains same in all the parts. The mapping of a block address in TLD is based on: (a) bits for selecting TLD and (b) bits for TLD set index. The bits required combining both is the set index bits of each bank. For example, if the set index bits required for banks is 10 and number of TLDs are 4, then out of 10 bits 2 bits are required for selecting TLD and the rest of the bits are used for selecting TLD sets. No replacement policy is required in TLD as there is a one-to-one mapping

with banks. Replacement are initiated only in banks. TLD is updated based on the request (update/remove) from the banks.

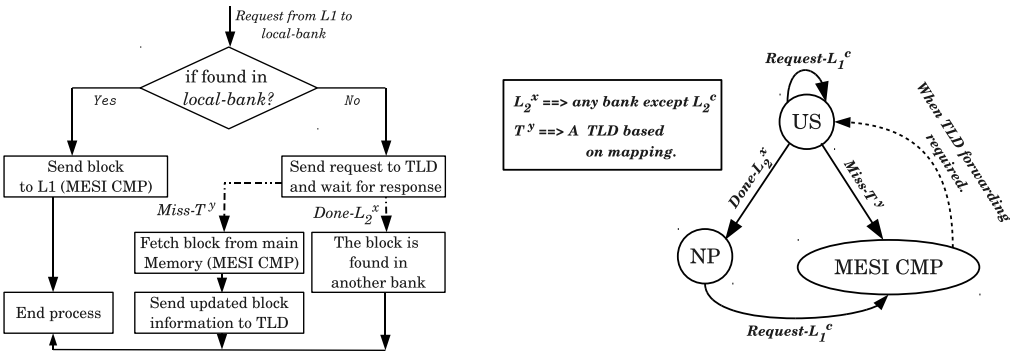
7.3.2 Operations of TLD-NUCA

Since the cores in TCMP have their own L1 caches and all the cores share a common L2 cache, the coherence must be maintained among the L1s. The baseline TCMP (T-SNUCA) is assumed to be working on MESI-CMP protocol [46]. Other protocols can also be used for TCMP designs. In this thesis both T-DNUCA and TLD-NUCA are designed extending the concept of MESI-CMP. This protocol can operate T-SNUCA but needs additional support for the dynamic nature of LLC in T-DNUCA as well as TLD-NUCA. This section discusses about the operations of TLD-NUCA and the additional coherence mechanisms required. Each TLD and bank has a separate controller. Only the additional mechanisms required for TLD-NUCA are discussed here; the operations controlled by existing MESI-CMP are not described.

All the controllers of TCMP based architectures communicate among themselves using message passing, through the NoC. In this thesis the messages are represented by *MessageName-[Sender]*. For example *Getx*-[L_1^2] means a *Getx* request from L_1^2. The messages can have different format like: request, response, data and control. For simplicity, only a generic format is used here. All messages send the block address and sender information embedded along with other informations. The details of the informations are mentioned whenever required. Below we discuss various scenarios and the protocol support provided for the same.

7.3.2.1 Block request from L1

Consider a core C requesting for a block B which is not in its L1. So the corresponding L1 (L_1^c) sends a request to the local-bank of LLC (L_2^c).



(a) Flow chart for the request processing in a bank L_2^C. The dotted arrows are event based: they triggers only when the event labeled with them occurs.

(b) State diagram in L_2^c for searching a block in *remote-banks*.

FIGURE 7.5: Managing LLC block request in TLD-NUCA.

If the block is found in L_2^c then it is called *local-hit* and the block is sent to L_1^c. Note that it may not be possible to send the block directly (or immediately) to L_1^c in the case when the block is shared exclusively with another L1 (or is in some transient state). Such conditions are managed by MESI-CMP and no additional mechanisms are required.

When the block is not found in L_2^c the other banks have to be searched. The current status of B in L_2^c must be NP (not present). The request from L_1^c is first served by MESI-CMP. Since the block is not in L_2^c it may be present in *remote-banks*. To get the block from remote-banks the controller of L_2^c needs to communicate with other banks with the help of the TLD. The additional mechanisms required here are shown in Figure 7.5(a). L_2^c sends a search request for block B to the TLD; the block is now considered as Under Search (US) in L_2^c. Any other request for the same block will not be sent to TLD until the response for current request arrives. A diagram showing the state transitions of a block (in L_2^c) during the process of remote searching is given in Figure 7.5(b). The two transitions possible from the initial state, US, are:

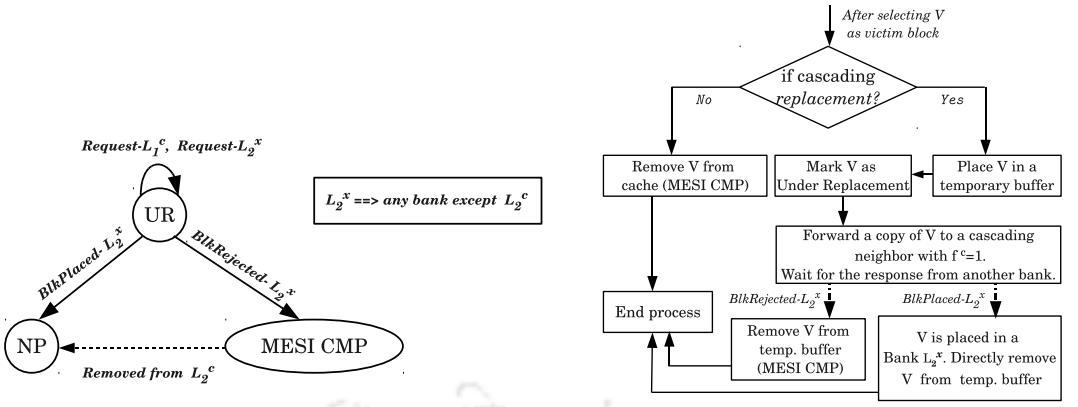
- L_2^c receives *Done*-$[L_2^x]$: It means the block is in $L_2^x; 0 \leq x < N$ and $x \neq c$. Now it is the responsibility of L_2^x to send it to L_1^c. The state of the block (B) in L_2^c becomes NP. The bank L_2^x is called *target-bank*.

- L_2^c receives *Miss*-$[T^y]$ from TLD $(T^y; 0 \leq y < K)$: it means that block is not in any bank. L_2^c considers this as cache miss and brings the block from the main memory. The newly fetched block may need to replace another block in L_2^c. In this case LRU policy selects a victim (V). The victim block is placed in a special buffer to make room for B. The replacement of V now depends on the replacement policy of TLD-NUCA which is discussed separately. Once a free space is created for B in L_2^c, the rest of the fetching process can be completed by existing MESI-CMP. The TLD has to be informed about the existence of B in L_2^c through an update-location request *Update#i*-$[L_2^c]$. It informs TLD that the block is currently in L_2^c with the way index of i.

7.3.2.2 Replacement

TLD-NUCA also uses cascading replacement as in T-DNUCA. The entire LLC is considered as a single bankset and a block can be moved from one bank to any other bank. Each bank has four neighbors to cascade a victim block and one can select any of the four for cascading the block. But allowing cascading of block in all the directions may in future revert the block back to the same bank. Therefore we restrict the neighbors to which to cascade the block. For a bank L_2^b the cascading neighbors $CSN(L_2^b)$ are the neighbors in the right and bottom directions. The right neighbor of a rightmost bank is considered as the first bank in the same row and the bottom neighbor of a bottom most bank is considered as the first bank in the same column. For example, cf. Fig 7.3, $CSN(L_2^1) = \{L_2^5, L_2^2\}, CSN(L_2^3) = \{L_2^7, L_2^0\}, CSN(L_2^{15}) = \{L_2^3, L_2^{12}\}$. Same as in T-DNUCA, a cascading number (CN) is used to allow maximum forwarding per cascaded replacement.

To evict a block V from a bank L_2^c the LRU replacement policy is used. Figure 7.6(b) summarizes the replacement details of TLD-DNUCA in a bank L_2^c. If the cascading replacement is not allowed then replacement process is local to L_2^c and can be handled by the existing MESI-CMP. When the cascading replacement is



(a) State diagram in L_2^c for cascading replacement.

(b) Flow diagram showing complete process of cascading replacement in L_2^c.

FIGURE 7.6: Managing Cascading request in TLD-NUCA.

allowed the victim V is forwarded to a neighbor bank $L_2^n \in CSN(L_2^c)$. A forward counter ($f^c = 1$) is also sent along with the block.

Figure 7.6(a) shows a state diagram of V in L_2^c once cascading replacement is invoked. When the block is sent to L_2^n the status of V in L_2^c is Under Replacement (UR). The block remains in a special buffer of L_2^c until a response comes from L_2^n (or may be from another bank if $CN > 1$). Any request for V either from the L1 cache of the *home-tile* (*Request*-$[L_1^{home}]$) or forwarded through TLD (*Request*-$[L_2^x]$; $x \neq c$) will have to wait[1]. There are two possible responses:

- *BlkPlaced*-$[L_2^x]$: it means the block is placed in L_2^x. Here L_2^x may or may not be L_2^n. Because if L_2^n cannot place the block and $CN > 1$ then L_2^n forwards the block to another bank from $CSN(L_2^n)$ (the maximum number of such forwarding allowed per cascading replacement is controlled by CN). When L_2^c receives the message it removes the block from its special buffer and the state of V in L_2^c becomes NP. L_2^c does not require to send any information to TLD in this case.

- *BlkRejected*-$[L_2^x]$: it means that the block is not placed in any bank and hence it must be removed from the cache. The removal of V from L_2^c is done by

[1]Such cases are very rare as the evicted block V is an LRU block in L_2^c, hence most of the time the process of cascading replacement finishes before such request arrives. On average only 0.0045% of the total block requests have to wait for such intermediate status.

MESI-CMP but after the removal, a block-removed message (*Removed*-$[L_2^c]$) has to be sent to TLD.

In both of the above responses, $0 \leq x < N$ and $x \neq c$. The summary of the method to replace a block is shown in Figure 7.6(b). Next we need to discuss the method of cascading, i.e. the details of processing done when a victim block is sent to its neighbor in CSN.

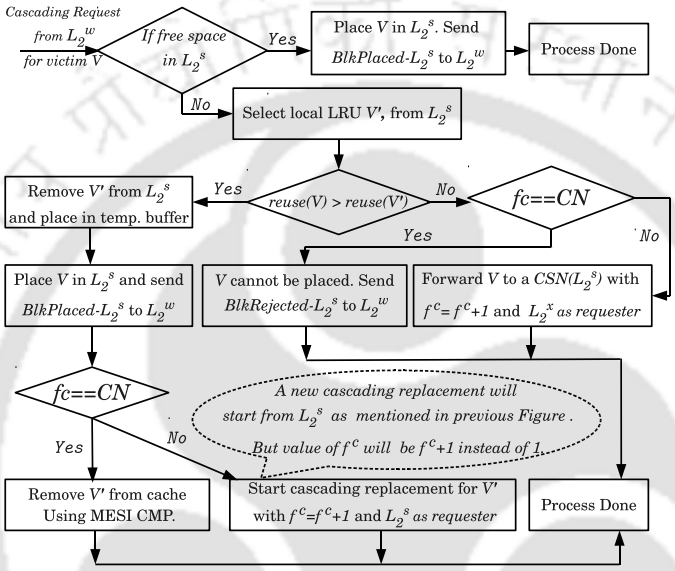


FIGURE 7.7: Handling incoming cascading blocks in L_2^s. V is considered as incoming block and V' as the local LRU block. The term *reuse(V)* means reuse counter of V.

The process to manage a cascaded block request from a bank (say L_2^w) to another bank (say L_2^s; $s \neq w$) is shown in Figure 7.7. On receiving the cascade request of V (from L_2^w), L_2^s stores it directly if any free space is available. Otherwise it compares the reuse counter of V with the reuse counter of its LRU block (V'). If the reuse counter of V is more than V' then V in placed on L_2^s evicting V'. The message *BlkPlaced*-$[L_2^s]$ is sent to L_2^w. If the forward counter $f^c = CN$ then no more forwarding is possible and V' is removed from the cache. The removal is done through the local replacement process managed by MESI-CMP. Otherwise (when $f^c < CN$), V' is forwarded to a bank from $CSN(L_2^s)$ with $f^c = f^c + 1$. A 4 bit reuse counter is used for each block in LLC.

V cannot be placed in L_2^s if its reuse counter is less than that of V'. In that case *BlkRejected*-$[L_2^s]$ is sent to L_2^w provided the value of f^c is CN. Otherwise (when $f^c < CN$) the block V is forwarded to a bank from $CSN(L_2^s)$ with $f^c = f^c + 1$.

For an $N \times N$ TCMP the value of CN cannot be more than $N - 1$. Choosing neighbor always from CSN guarantees that a cascading process starting from L_2^c never revisits the same bank on account of forwarding.

7.3.2.3 Migration

Consider the requesting core for a block B as C and the block is currently in L_2^m $(m \neq c)$. The controller of L_2^m tracks the access of each block. If any block in L_2^m is consecutively requested by a core (C) for more than Q times, then the block is forwarded to the local-bank of the requesting core: L_2^c. The forwarding is almost same as cascading replacement. The only difference is that the migrated block must be placed in L_2^c. The victim block (if any) from L_2^c follows the cascading replacement policy. A 6-bit counter is used for each block to manage the migration: 4 bits are used for the tile-id of the previous requester and 2 bits are for consecutive requests. With 2 bits of consecutive request counter, the value of Q is 3.

7.3.2.4 Target Bank

Consider the situation when an L1 requests for a block B to the local-bank L_2^c, and that L_2^c does not have the block. The controller of L_2^c forwards the request to TLD. The TLD has information about the bank (L_2^t) in which the block B currently resides. It forwards the request to L_2^t. In normal situations t and c are not same but there are some exceptional cases when both t and c can be same. Such exceptional cases are discussed in Section 7.3.2.6.

In normal condition the block is with L_2^t and it sends the block to the requesting L1 directly. The process is handled by MESI-CMP, but after completing the process, L_2^t sends an acknowledgement message *Done*-$[L_2^t]$ to L_2^c and TLD.

It may be also possible that the block is in a condition that it cannot process any request. The situation may arise in three cases : (a) the block is removed from L_2^t. (b) the block is under replacement process and (c) the block is under movement (cascading replacement/migration). In any of these three cases TLD may wrongly forward a request to L_2^t until it gets updated about the changes in L_2^t. In this case L_2^t sends a message *Wait*-$[L_2^t]$ to the TLD, informing TLD to wait until further information (regarding block B) is received. The further information to TLD for B can be either *Removed*-$[L_2^t]$ or *Update#i*-$[L_2^x]$; $x \notin \{t, c\}$. These are discussed in detail in the next sub-section.

7.3.2.5 TLD

The structure of TLD is explained earlier. The corresponding *tld-block* for a block B is referred to as *tld(B)*. A *tld-block* can be in any of the four states:

- *Not present (NP)* - the *tld-block* is not in TLD.
- *Normal (NR)* - the *tld-block* has a valid tag entry and also the *bank-id* where the corresponding block resides.
- *Miss Lock (ML)* - the block is currently being fetched (or under fetching) by some bank from the main memory.
- *Wait (WT)* - the block is either under replacement or movement in some bank and therefore any more requests (for block B) have to wait.

The state diagram of TLD for a *tld-block* is given in Figure 7.8. When a block (B) is first requested by a core (C) it will not be in its local-bank and hence sends a request to TLD (as discussed in Section 7.3.2.1). If the block is not in cache, initial status of the corresponding *tld-block* (*tld(B)*) is *NP*. TLD sends a response *Miss*-$[T^y]$ to L_2^c and changes the state of *tld(B)* to *ML*. Here the miss lock is necessary to prevent any other bank from fetching the block simultaneously. TLD will be forced to stall any other requests for the same block until L_2^c sends *Update#i*-$[L_2^c]$

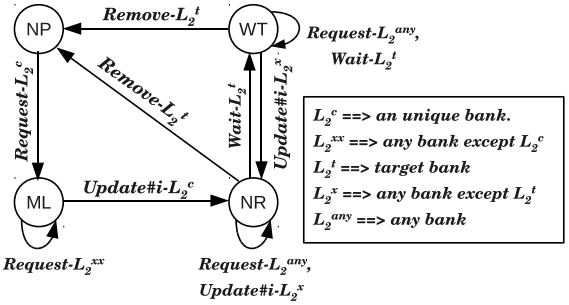


FIGURE 7.8: State diagram describing the status of a *tld-block* in TLD. The letter c, x and y are used to represent the sender banks uniquely. The corresponding actions requires to perform in each transition is not shown.

to T^y. Receiving this message from L_2^c means L_2^c fetched the block from main memory (or is in the process of fetching data but the address is already allocated). In that case the status of *tld(B)* is changed from *ML* to *NR*. The status of *tld(B)* is NR means the block B is definitely present in some bank, called target bank (say L_2^t). Since L_2^c just fetched the block hence currently $t = c$. Now TLD can forward any further request for B to the target bank (L_2^t).

The operations of a target bank on receiving a block request from TLD is already discussed in Section 7.3.2.4. If TLD receives *Wait*-$[L_2^t]$, as response from the target bank of B then the status of *tld(B)* is changed to *WT*. It means the block is either under replacement or movement; TLD has to wait until the task is completed. Multiple wait signals from the target bank keep the *tld(B)* in *WT*. The status is *WT* until a further request (*Removed*-$[L_2^t]$ or *Update#i*-$[L_2^x]$; $x \neq t$) arrives from a bank with the latest status of B. Receiving *Removed*-$[L_2^t]$ from target bank means the block is removed from the cache and hence has to be removed from TLD. If TLD receives *Update#i*-$[L_2^x]$ it means the block is moved (cascaded/migrated) to L_2^x and placed in the i^{th} way. TLD can now forward future block requests for B to L_2^x. The status of *tld(B)* is changed to *NR* and makes L_2^x as the target bank.

7.3.2.6 Additional conditions to be handled

The major operations of TLD-NUCA are discussed above but there are some other conditions that need to be taken care of by the controllers. The messages

communicated through NoC may reach the destination out of order (depending on the NoC routing logic). The controllers must have mechanism to handle the out of order messages without any loss of consistency. For example, in a bank (say L_2^c) when a block V replaces V' (consider no cascading) then V' is temporarily stored in a buffer for removal and the cache space is allocated to V. L_2^c sends two messages to TLD separately: (a) after V' gets completely removed from the cache (*Removed*-[L_2^c]), and (b) after V is placed (*Update#i*-[L_2^c]). These two messages may reach one after another in TLD. The two scenarios are explained below:

- If *Removed*-[L_2^c] reaches first then the entry *tld(V′)* gets removed from the TLD. Later when *Update#i*-[L_2^C] reaches, *tld(V)* is placed in place of the previous *tld(V′)*.
- If *Update#i*-[L_2^c] for block V reaches first then to make space for *tld(V)*, the existing *tld(V′)* must be stored in some temporary buffer of TLD. Later when *Removed*-[L_2^c] arrives, the *tld(V′)* from the TLDs temporary buffer is deleted.

Note that an additional buffer is required in the second case. The out of order behavior of all the messages is handled accurately.

It may be possible that TLD forwards a block request to a target bank (L_2^t) which is the same bank (L_2^c) that requested for the block. The situation may arise when L_2^c does not have the block initially and forwards the request to TLD but before TLD can decide about its actual location in LLC, the block gets moved to L_2^c due to either migration or cascading replacement. In this case the target bank does not require to inform the requesting core with the message *Done*-[L_2^t] (discussed in section 7.3.2.4) because both are the same bank.

7.4 Experimental analysis

In order to evaluate TLD-NUCA, the simulation has been performed by using multi-core cycle accurate simulator GEMS [85] which runs on top of SIMICS [82],

Component	Parameters
No. of tiles, Processor, L1 cache size	16, UltraSPARCIII+, 64KB 4-way
Total LLC (L2) size	2MB and 4MB
bank size	256KB 4 way and 128KB 4-way
Block Size, Memory bank	64KB, 1GB 4KB/page
Router pipeline stage	5-stage
Access latency L1/L2/TLD	2/6/3 cycles

TABLE 7.2: System Parameters

a full-system functional simulator. The performance of TLD-NUCA is compared with both T-SNUCA and T-DNUCA. All the TCMP architectures (T-SNUCA, T-DNUCA and TLD-NUCA) are implemented on top of the baseline TCMP as shown in Figure 3.1. GEMS is capable of simulating the entire memory system of CMP. The complete protocol as discussed in Section 7.3.2 is implemented in GEMS. The configuration details of the processor, cache memory and main memory used are given in Table 7.2. The Parsec benchmarks used for this simulation are: *vips*, *body*, *frrt*, *flud*, *frq*, *blck*, *mx1*, *mx2* and *mx3*. The detail description about the benchmarks are given in Table 3.3. The benchmarks are executed on each simulated architecture to evaluate the performance of it.

7.4.1 Configuration of T-DNUCA and TLD-NUCA used

Various configurations of T-DNUCA and TLD-NUCA are used for the experiment. In general the T-DNUCA configurations are represented as MxCy, where x can be either 0 or 1 and $0 \leq y < bpbs-1$. 0 after M means migration not allowed and 0 after C means cascading replacement not allowed. Putting $y > 0$ after C means cascading replacement is allowed with a cascading number of y. The concept of cascading number is discussed in Section 7.3.2.2. Based on the migration and cascading replacement two T-DNUCA configurations are used for the experiments: **M1C1** and **M1C3**. Other configurations of T-DNUCA like M1C2, M0C1, M0C3 etc are not considered here because as in our previous work [107] we found that M1C1 and M1C3 are the best configurations for T-DNUCA. Since TLD-NUCA uses the same concept of migration and cascading replacement, similar configurations are used for TLD-NUCA. But considering the different number of tld-parts,

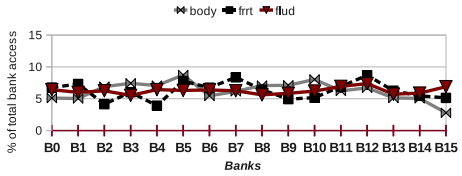


FIGURE 7.9: Bank usage in TLD-NUCA.

the TLD-NUCA configurations can be generalised as Tn-MxCy; $n > 1$, $x \in \{0, 1\}$ and $0 \leq y < N$. N means total number of banks (TLD-NUCA has only one bankset). Four configurations of TLD-NUCA are used for the experiments: **T2-M1C1**, **T2-M1C3**, **T4-M1C1** and **T4-M1C3**. The configurations mentioned here can be used for any LLC sizes. As mentioned in Table 5.1 the LLC size considered for the experiments are 2MB and 4MB. Experimental analysis of both LLC sizes are shown separately.

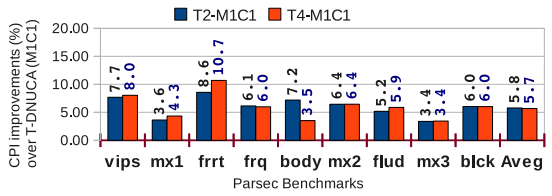
7.4.2 Comparison with T-DNUCA for 4MB LLC

TLD-NUCA is compared with T-DNUCA for the above mentioned configurations. All the experiments done in this section are for 4MB LLC equally divided into 16, 4-way associative banks. Figure 7.10 shows the comparison when T-DNUCA uses the configuration M1C1 i.e. migration with cascading number as 1. For each configuration of T-DNUCA two different TLD-NUCA configurations are compared based on the number of TLDs as 2 and 4. The TLD-NUCA configuration compared with M1C1 are T2-M1C1 and T4-M1C1. From the Figure 7.10(a) it can be observed that TLD-NUCA improves CPI by almost 6% for both T2-M1C1 and T4-M1C1 as compared to M1C1. The improvement in both TLD-NUCA configurations are same because of the less number of remote searches required. Most of the block requests are served by the local banks without communicating with TLD hence two tld-parts can serve the requests without any congestions. The CPI improvements happen for two reasons:

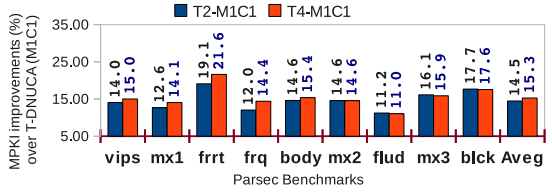
(a) Better cache utilisation by distributing loads among the banks through migration and cascading replacement. Figure 7.9 shows the distribution of loads among the banks.

(b) Minimize the remote searching time by using TLD.

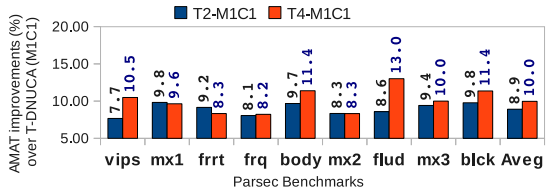
Case (a) also results in improvements of Miss Per Thousand Instructions (MPKI). Figure 7.10(b) shows the improvements of TLD-NUCA over T-DNUCA in terms of MPKI. The improvements are 14.5% and 15.3% for T2-M1C1 and T4-M1C1 respectively. Case (a) and case (b) jointly improves the Average Memory Access Time (AMAT). Figure 7.10(c) shows the improvements in terms of AMAT.



(a) CPI



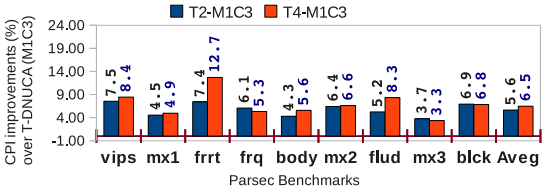
(b) MPKI



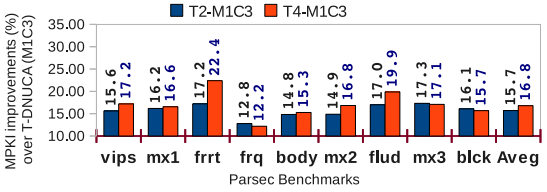
(c) AMAT

FIGURE 7.10: Improvement of TLD-NUCA over T-DNUCA having configuration as **M1C1** and LLC size as **4MB**.

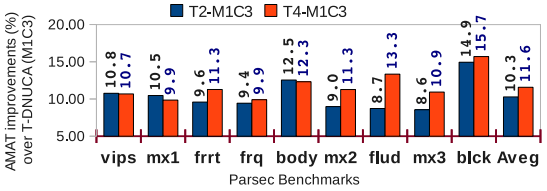
Comparison with the T-DNUCA configuration, M1C3, is shown in Figure 7.11. The two TLD-NUCA configurations used to compare with M1C3 are T2-M1C3 and T4-M1C3. Figure 7.11(a), 7.11(b) and 7.11(c) shows the improvements in terms of CPI, MPKI and AMAT respectively. The average CPI improvements are 5.6% and 6.5% for T2-M1C3 and T4-M1C3 respectively. The average improvements in terms



(a) CPI



(b) MPKI



(c) AMAT

FIGURE 7.11: Improvement of TLD-NUCA over T-DNUCA having configuration as **M1C3** and LLC size as **4MB**.

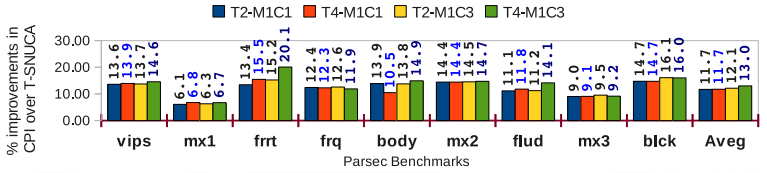
of MPKI are 15.7% (T2-M1C3) and 16.8% (T4-M1C3). AMAT improvements are 10.3% (T2-M1C3) and 11.6% (T4-M1C3).

7.4.2.1 M1C1 v/s M1C3

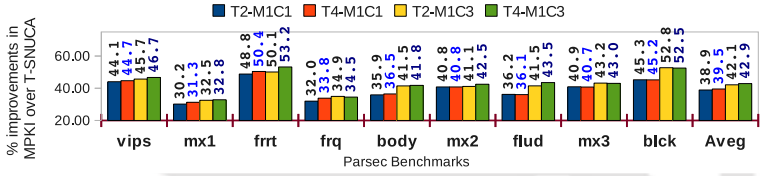
It can be observed that the improvements for M1C3 are slightly better than M1C1. This is because of using higher cascading number (3 instead of 1). Higher cascading number allows better uniform distribution of loads among the banks. Note that the cascading replacement is done in background while the normal execution of other blocks continues. But higher cascading number means more number of block movements through the NoC which can degrade the NoC performance as well as increase the NoC energy consumption. This is the reason why the maximum cascading number is not considered more than 3.

7.4.3 Comparison with T-SNUCA for 4MB LLC

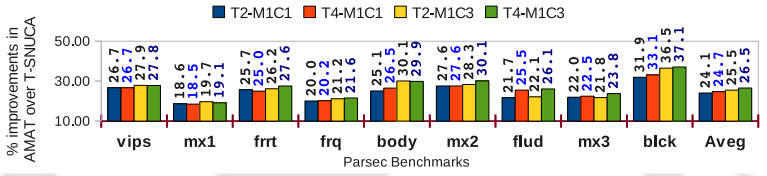
The experiments done in this section are for 4MB LLC having 16, 4-way associative banks. In [107] it has been mentioned that T-DNUCA gives better performance than T-SNUCA. The previous section shows that TLD-NUCA improves performance even better than T-DNUCA. Which means TLD-NUCA obviously improves performance of T-SNUCA. The four TLD-NUCA configurations used to compare with T-SNUCA are T2-M1C1, T4-M1C1, T2-M1C3 and T4-M1C3.



(a) CPI



(b) MPKI

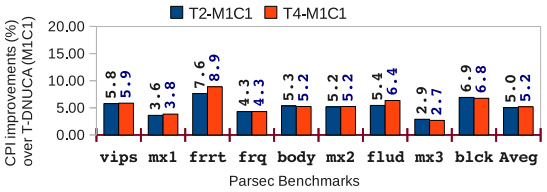


(c) AMAT

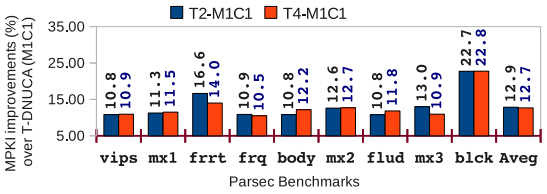
FIGURE 7.12: Performance comparison of TLD-NUCA with T-SNUCA having LLC size as **4MB**.

Figure 7.12 shows the comparison of TLD-NUCA with T-SNUCA. The CPI comparison is shown in Figure 7.12(a). It can be observed that the average CPI improvements are 11.7% (T2-M1C1), 11.7% (T4-M1C1), 12.1% (T2-M1C3) and 13% (T4-M1C3). All the four configurations show almost similar results. The reason for such similar results are already discussed in Section 7.4.2. The comparison in terms of MPKI is shown in Figure 7.12(b). The average improvement in MPKI is 38.9% (T2-M1C1), 39.5% (T4-M1C1), 42.1% (T2-M1C3) and 42.9% (T4-M1C3). As shown in Figure 2.11 the load distributions among the banks is

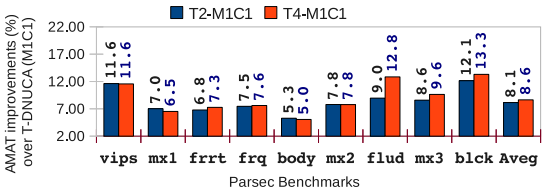
not uniform in T-SNUCA because of its fixed mapping policy. TLD-NUCA can distribute the loads among the banks. Also the initial placement policy reduces the average cache access time of TLD-NUCA as compared to T-SNUCA. The AMAT improvements in TLD-NUCA over T-SNUCA are shown in Figure 7.12(c). On average the improvements are 24.1% (T2-M1C1), 24.7% (T4-M1C1), 25.5% (T2-M1C3) and 26.5% (T4-M1C3).



(a) CPI



(b) MPKI

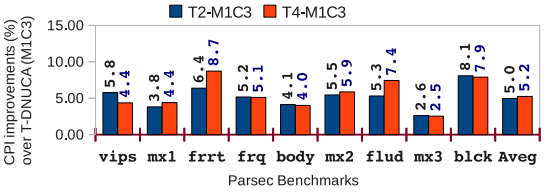


(c) AMAT

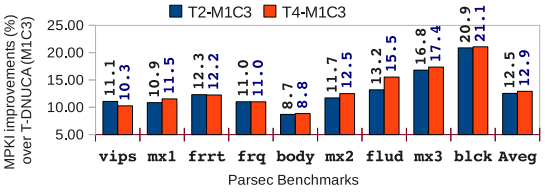
FIGURE 7.13: Comparison of TLD-NUCA with T-DNUCA having configuration as **M1C1** and LLC size as **2MB**.

7.4.4 Comparisons with 2MB LLC

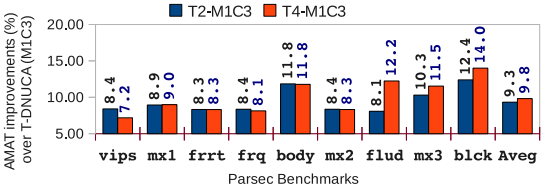
TLD-NUCA is also compared with 2MB LCC. All the comparisons done for 4MB are also done for 2MB. Figure 7.13 shows the comparison of TLD-NUCA with T-DNUCA for configuration M1C1. The comparisons in terms of CPI, MPKI and



(a) CPI



(b) MPKI



(c) AMAT

FIGURE 7.14: Comparison of TLD-NUCA with T-DNUCA having configuration as **M1C3** and LLC size as **2MB**.

AMAT are shown in Figure 7.13(a), 7.13(b) and 7.13(c) respectively. The improvements in 2MB LLC are slightly less than a 4MB LLC. The average improvements of CPI while compared to M1C1 are 5% (T2-M1C1) and 5.2% (T4-M1C1). MPKI is improved by 12.9% (T2-M1C1) and 12.7% (T4-M1C1). The comparison of TLD-NUCA with T-DNUCA having configuration M1C3 is shown in Figure 7.14. The improvements in M1C3 are almost same as M1C1.

The comparison with T-SNUCA is shown in Figure 7.15. The four configurations compared with T-SNUCA are T2-M1C1, T4-M1C1, T4-M1C1 and T4-M1C3. Figure 7.15(a), 7.15(b) and 7.15(c) shows the comparisons in terms of CPI, MPKI and AMAT respectively.

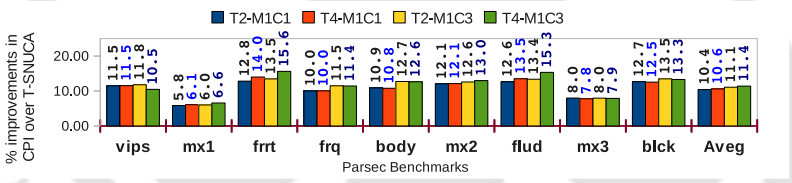
	Comparison with T-DNUCA				Comparison with T-SNUCA			
Source Architecture	T-DNUCA (M1C1)		T-DNUCA (M1C3)		Baseline T-SNUCA (cf. Figure 3.3.5)			
Target Architecture	T2-M1C1	T4-M1C1	T2-M1C3	T4-M1C3	T2-M1C1	T4-M1C1	T2-M1C3	T4-M1C3
CPI improvements:	5.8%	5.7%	5.6%	6.5%	11.7%	11.7%	12.1%	13%
MPKI improvements:	14.5%	15.3%	15.7%	16.8%	38.9%	39.5%	42.1%	42.9%
AMAT improvements:	8.9%	10%	10.3%	11.6%	24.1%	24.7%	25.5%	26.5%

TABLE 7.3: The average improvements of TLD-NUCA over T-DNUCA and T-SNUCA having **4MB LLC**. Target Architectures mentioned in the table are TLD-NUCA configurations. The improvements are shown for Target Architecture as compared with the corresponding configuration of Source Architecture.

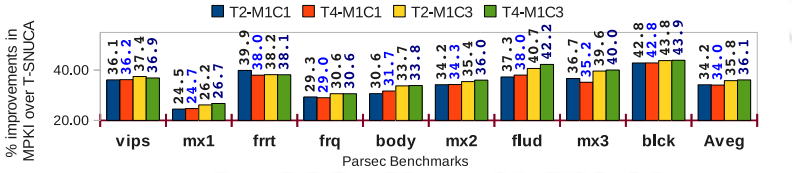
7.4.5 Result analysis of different configurations

Table 7.3 shows the average improvements of various TLD-NUCA configurations compared with both T-SNUCA and T-DNUCA. The results shown in this table are for 4MB LLC. The improvements are almost same in all the TLD-NUCA configurations. There are two reasons for these similarities:

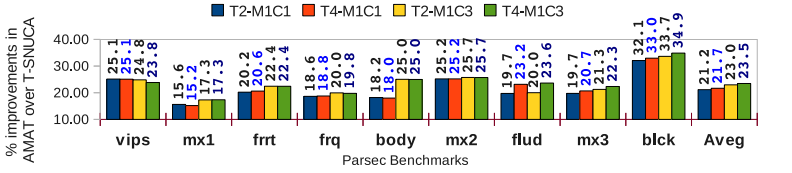
- More number of tld-parts are not required for the experiments performed.



(a) CPI



(b) MPKI



(c) AMAT

FIGURE 7.15: Comparison of TLD-NUCA with T-SNUCA having LLC size as **2MB**.

	Comparison with T-DNUCA				Comparison with T-SNUCA			
Source Architecture	T-DNUCA (M1C1)		T-DNUCA (M1C3)		Baseline T-SNUCA (cf. Figure 3.3.5)			
Target Architecture	T2-M1C1	T4-M1C1	T2-M1C3	T4-M1C3	T2-M1C1	T4-M1C1	T2-M1C3	T4-M1C3
CPI improvements:	5%	5.2%	5%	5.2%	10.4%	10.6%	11.1%	11.4%
MPKI improvements:	12.9%	12.7%	12.5%	12.9%	34.2%	34%	35.8%	36.1%
AMAT improvements:	8.1%	8.6%	9.3%	9.8%	21.2%	21.7%	23%	23.5%

TABLE 7.4: The average improvements of TLD-NUCA over T-DNUCA and T-SNUCA having **2MB LLC**. Target Architectures mentioned in the table are TLD-NUCA configurations. The improvements are shown for Target Architecture as compared with the corresponding configuration of Source Architecture.

Two tld-parts can serve all the requests without any major congestion. As discussed in Section 7.3 usage of smart placement policy reduces the requirements of remote search.

- The benefits gained by higher cascading number is partially consumed by NoC for more block movements.

The average improvements of TLD-NUCA configurations for 2MB LLC are given in Table 7.4.

7.4.6 Hardware Overheads

TLD-NUCA has some additional hardware overheads because of its TLD. The hardware overheads are calculated in terms of energy, storage and area.

7.4.6.1 Energy Overhead

The energy model used in this thesis is influenced from [72]. The total energy consumed by LLC for executing a particular application is given in Equation 7.1. The term E_X represents energy consumed by component X.

$$E_{total} = E_{cache} + E_{ntw} + E_{tld} \tag{7.1}$$

The total energy is calculated as the energy consumed by the four different components: cache banks (cache), NoC (ntw), TLD (tld) and off chip accesses. There are

two types of energy consumed by each components: static (st) and dynamic (dy). Dynamic energy is consumed during every cache access whereas the static energy represents the leakage power. The total energy consumption of cache banks is calculated from the equations 7.2, 7.3 and 7.4. The term E^p_X means energy consumed by X per access, P_X means leakage power consumed by X.

$$E_{cache} = E_{cache\_dy} + E_{cache\_st} \tag{7.2}$$

$$E_{cache\_dy} = totalBankAccesses \times E^p_{bank} \tag{7.3}$$

$$E_{cache\_st} = (no.\ of\ banks \times P_{bank}) \times execTime \tag{7.4}$$

The value of E^p_{bank} and P_{bank} are calculated from CACTI 6.0 [2]. Rest of the values are obtained from the full system simulation. The term $execTime$ means the total execution time of TLD-NUCA. The NoC energy consumption is provided by the two modules of GEMS called Garnet and Orion. Equations 7.5, 7.6 and 7.7 gives the total energy consumption of TLD.

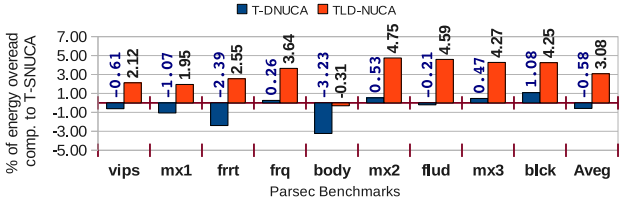
$$E_{tld} = E_{tld\_dy} + E_{tld\_st} \tag{7.5}$$

$$E_{tld\_dy} = no.\ of\ requests \times E^p_{tld} \tag{7.6}$$

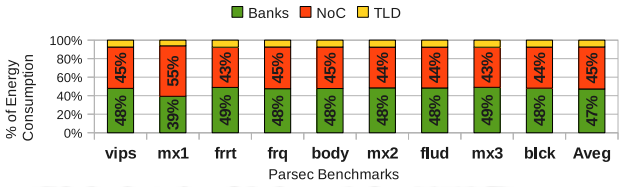
$$E_{tld\_st} = (no.\ of\ tlds \times P_{tld}) \times execTime \tag{7.7}$$

Although logically each *tld-part* has very high associativity, it is actually segmented into multiple t-segs having associativity same as the associativity of a bank. Each t-seg duplicates the tag-array contents of S/p sets from a particular bank, where S is the total sets per bank and p is the number of tld-parts. Hence the energy consumption of a t-seg is considered as the energy consumption of tag-array for a bank having S/p sets. The value of E^p_{tld} and P_{tld} are calculated by adding the energy consumption of all the t-segs in a tld-part.

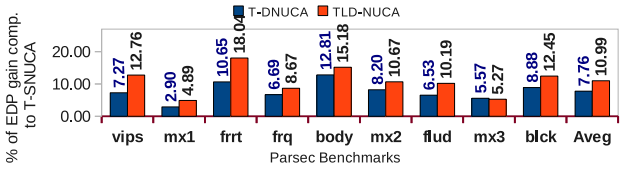
The total energy overhead of both T-DNUCA and TLD-NUCA as compared to T-SNUCA is shown in Figure 7.16(a). Though T-DNUCA has many multicast requests and block movements the energy consumption remains same as T-SNUCA.



(a) Total energy overhead compared to T-SNUCA.



(b) The energy breakdown of TLD-NUCA.



(c) EDP gain compared to T-SNUCA.

FIGURE 7.16: Comparison of different energy consumption parameters of TLD-NUCA and T-DNUCA with T-SNUCA. The experiments are done on a 4MB LLC having each bank as 256KB 4-way associative. For both T-DNUCA and TLD-NUCA the cascading number considered as 3 and migration is on.

The main reason for this is the smart placement policy which reduces the chances of remote search and hence reduces the network communication as well as the number of additional bank accesses required for remote search. TLD-NUCA on average has 3.08% energy overhead as compared to T-SNUCA. The reason for this overhead is the centralised TLD. Figure 7.16(b) shows the energy breakdown of the different components of TLD-NUCA. The off-chip energy is not shown here. It can be observed that TLD on average require 8% of the total energy consumed by TLD-NUCA. Though TLD requires 8% energy the total energy overhead of TLD-NUCA (as shown in Figure 7.16(a)) is less (3.08%) because of the improvements in network energy consumption. TLD-NUCA requires no mandatory communication between the requesting core and the home-bank, hence require less communication than T-DNUCA. Also no multicast search request is needed to go through NoC. TLD-NUCA requires 13.4% less energy for NoC as compared to T-SNUCA. Though TLD-NUCA has additional energy overhead it gives better EDP (energy

LLC					
A	LLC size:	4MB	J	Tag bits:	44 bits
B	Number of banks:	16	K	Sharers bits:	4 bits
C	Size of each bank:	$(A/B) = 256$KB	L	Dirty bit:	1 bit
D	Block size:	64 bytes	M	Total tag size:	$(J + K + L)$=49 bits
E	Address space:	64 bits	N	Total tag array size:	$(I \times M) = 3211264$ bits
F	Bank associativity:	4 way	O	Total counter bits:	10 bits (per block)
G	Total sets per bank:	$(C/(F \times D)) = 1024$	P	Counter storage:	$(O \times H \times B)$=655360 bits
H	Total block per bank:	$(G \times F) = 4096$	Q	Total LLC Size:	$(N + A + P)$=37421056 bits
I	Total tag per LLC:	$(H \times B) = 65536$			
TLD					
R	Total tld-parts:	4	V	Bits for bank-id:	4 bits
S	Set per tld-part:	$(G/R) = 256$	W	Dirty bit:	1 bit
T	TLD associativity:	$(F \times B) = 64$	X	Total bits per entry:	$(J + V + W) = 49$ bits
U	Total TLD entries:	$(R \times S \times T)$=65536 bits	Y	Total TLD size:	$(U \times X)$= 3211264 bits
Total TLD-NUCA storage: $(Q + Y)$= 40632320 bits **Total T-SNUCA storage:** 36765696 bits **Storage overhead of TLD-NUCA:** 10.5%					

TABLE 7.5: Storage overhead calculation of TLD-NUCA over T-SNUCA.

× CPI) than T-DNUCA. Figure 7.16(c) shows the total EDP gain of T-DNUCA and TLD-NUCA as compared to T-SNUCA. The reason of better EDP is the improvement in CPI.

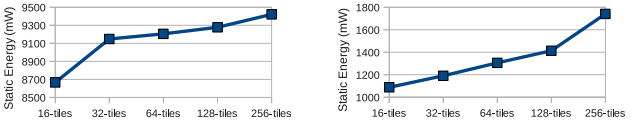
7.4.6.2 Storage and area overhead

TLD-NUCA requires additional storage for the TLD. Table 7.5 gives the storage details of the LLC in TLD-NUCA. The table shows that a TLD-NUCA having 4MB LLC with 16 4-way associative banks and 4 tld-parts have 10.5% of storage overhead as compared to T-SNUCA. The overhead remains same for all LLC size and associativity. The area overhead of TLD-NUCA having 4MB LLC, divided into 16 4-way associative banks is 11.47% while the same for 2MB LLC is 10.6%.

7.4.6.3 Energy overhead of highly associative TLD

The total energy consumption of TLD-NUCA (as shown in Section 7.4.6.1) is only 3.08% more than T-SNUCA, this overhead increases with the increase in TLD associativity. The associativity of the TLD increases as the bank associativity increases. For example, in case of a 16-core TLD-NUCA having 8-way associative banks, the associativity of TLD is 128. Hence the associativity of each t-segs will

be 8. Experimental analysis found that the energy overhead of such TLD-NUCA is 3.6% more than the corresponding T-SNUCA.



(a) LLC (b) TLD

FIGURE 7.17: The static energy required per cycle in LLC (all banks) and TLD (all tld-parts). The values are calculated on CACTI.

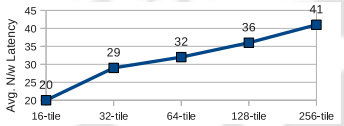


FIGURE 7.18: The increase in average network latency for TLD-NUCA (single bankset).

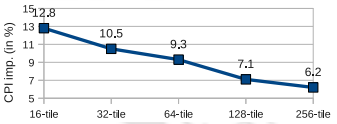


FIGURE 7.19: The CPI improvements of TLD-NUCA (single bankset) over T-SNUCA.

The energy consumption of TLD also increases with increasing number of cores in TLD-NUCA. Figure 7.17 shows the total static power consumption per cycle for both LLC and TLD having different number of tiles. It can be observed that though the energy consumption of TLD increases the corresponding consumption of LLC also increases. Such relative increase in energy consumption limits the overhead of TLD during execution. But there are other constraints which reduces the performance of TLD-NUCA having more than 16 cores. Section 7.5 discusses it in detail and also proposes an alternative TLD-NUCA design to support high number of cores without degrading performance.

7.5 Scalability of TLD-NUCA

Placing more number of tiles in a single bankset has reduces the performance of TLD-NUCA. The main reason for such reduction are TLD throughput and network latency. Since the number of banks increases the TLD becomes a bottleneck to support multiple request in parallel. Also more number of tiles increases the

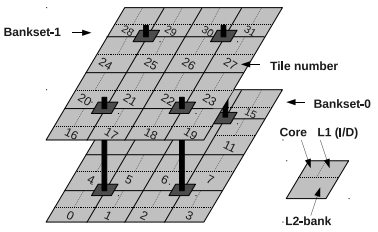


FIGURE 7.20: Example of TLD-NUCA having 32-cores distributed in two banksets.

Improvements	vips	mx1	frrt	frq	mx2	mx3	blck	Average
MPKI (32-Tiles)	28.89	23.19	37.59	25.94	29.62	30.01	38.24	30.06
CPI (32-Tiles)	10.35	4.64	11.49	9.39	10.80	6.76	10.77	8.8
MPKI (64-Tiles)	28.49	23.27	36.09	26.17	30.85	32.76	39.11	30.5
CPI (64-Tiles)	10.66	4.68	11.86	9.49	11.10	7.00	10.91	9

TABLE 7.6: Improvements (%) of 3D TLD-NUCA (multiple banksets) over the corresponding 3D T-SNUCA.

on-chip communication distance between tiles and TLD. Figure 7.18 shows the increase of average network latency in TLD-NUCA with increasing number of tiles. The reduction in performance improvement is shown in Figure 7.19. Hence a slightly different technique is used to design TLD-NUCA having more than 16 tiles. The tiles are divided into multiple banksets of size 16. Each bankset has its own TLD as described in Section 7.3.1. Same as T-DNUCA a block can only be placed in a particular bankset. But once the banks are divided into multiple banksets the mandatory communication to the home-bank (cf. Section 7.1) becomes obvious. To reduce such mandatory communication a three dimensional NoC structure as proposed in [108] is considered as the communication backbone for such TLD-NUCA. The banksets are stacked on top of each other and some vertical bus pillars are installed to communicate among the planes. Figure 7.20 shows an example of TLD-NUCA having 32 tiles. The 3D architecture of TLD-NUCA makes the design highly scalable and the TLD does not become a bottleneck for the scalability of the design. Such three dimensional TLD-NUCA is a better architecture than T-DNUCA even after using multiple banksets like T-DNUCA because: (a) the vertical communication among the banksets is very fast [108] hence, the mandatory communication takes very less time as compared to T-DNUCA (b) use of larger bankset always gives better utilisation than T-DNUCA.

Number of Tiles	vips	mx1	frrt	frq	mx2	mx3	blck	Average
32-Tiles	2.63	2.76	2.04	2.78	3.49	3.17	3.03	2.8
64-Tiles	2.76	2.81	2.18	2.92	3.61	3.29	3.15	2.92

TABLE 7.7: The energy overhead (in %) of 3D TLD-NUCA over 3D T-SNUCA.

Table 7.6 shows the improvements of TLD-NUCA over T-SNUCA for both 32-core and 64-core designs. It can be observed the improvements are slightly less than the improvement shown by 16-core TLD-NUCA. This is because of the mandatory communication cost between the requesting core and the home-bank. The energy overheads for TLD-NUCA having different number of cores is given in Table 7.7. All the overheads are calculated using the similar technique discussed in Section 7.4.6.

7.6 Comparison of TLD-NUCA with Private LLC Architecture

Use of larger bankset, smart placement and cascading replacement makes the shared LLC based TLD-NUCA closer to a private LLC base CMP, called Cooperative Caching (CC) [12]. In CC each core has its own private LLC and each LLC can spill its block into another LLC when required. The Central Coherence Engine (CCE) is used to search a block from all the private LLCs. The blocks which are not shared are called singlet. Though TLD-NUCA and CC has similar architecture but they differ in technical aspects.

In CC the shared blocks are allowed to replicate in local tiles for faster access in future. But whenever a shared block is requested in exclusive mode by some core, the replicas in different tiles has to be invalidated. The CCE has to handle such invalidations.

The singlet block evicted from one tile is spilled into another tile. To place the evicted block (say V) into target tile the LRU block (say V') from that tile has to be removed. It may be possible that the spilled block V is older than V'. Such spill

% Improvements	vips	mx1	frrt	frq	mx2	mx3	blck	Average
MPKI	12.94	6.74	5.69	6.86	7.13	10.42	10.57	11.68
AMAT	10.02	11.07	12.73	10.21	12.09	14.47	11.77	8.28
CPI	4.38	3.66	6.53	4.48	5.75	4.93	4.57	4.82

TABLE 7.8: Improvements (%) of 16-core TLD-NUCA over 16-core Cooperative Cache.

actions are not beneficial as it may remove important blocks to store less important blocks. We found that on average 46% spilled blocks of CC are replacing more important blocks from target banks. In TLD-NUCA the placement of a cascaded block is decided based on a reuse counter. Hence TLD-NUCA has more chance to keep the important blocks into cache. The replacement policy of CC first searches for shared blocks to evict and if no shared blocks are available then it selects the LRU block as the victim. Such policy makes the replacement process complicated and time consuming. Also information has to be maintained about the behavior (shared/singlet) of each block. The CCE is responsible to update the status of a block from shared to singlet.

Another issue with CC is scalability. The cluster based alternatives of CC requires multiple levels of directories to handle clusters. TLD-NUCA has a three dimensional architecture for supporting TCMP having more than 16 tiles. No two level directory is required for such TLD-NUCA designs.

Experimental analysis found that TLD-NUCA performs better than CC. The improvement of TLD-NUCA over CC is for two reasons:

- Better utilisation: TLD-NUCA always removes less important blocks from the cache hence the dead blocks are replaced by the valid blocks. The utilisation also improves by not allowing replicated blocks. As discussed above the advantage of replication is partially consumed by the mechanism required to manage it.

- Cache access time: In CC, the LLC level invalidation and replacement policy consumes additional time to process a request. Also removal of important blocks results in unnecessary misses and hence need to bring blocks from main memory.

Table 7.8 shows the improvements of TLD-NUCA over CC in terms of MPKI, AMAT and CPI. On average, TLD-NUCA (16-core) has 11.68% (MPKI), 8.28% (AMAT) and 4.82% (CPI) improvements over CC. Since CC has scalability issues [12, 5] the comparison of TLD-NUCA with CC for more than 16 cores is not done.

7.7 Summary

In this chapter we proposed a DNUCA based TCMP called TLD-NUCA. In TLD-NUCA all the banks belong to a single bankset and the loads can be distributed much better among the banks. The blocks in TLD-NUCA can be placed in any bank. A newly fetched block is placed to the local-bank of the requesting core, hence removes the requirements of the mandatory communication between the requesting core and the home-bank, as required in T-DNUCA. The smart placement policy results most of the L2 requests to be served by the local banks without communicating with other banks. Searching the remote banks is done through a centralised directory called TLD. The TLD stores the bank-id where a block currently resides. When a block is not found in its local-bank the TLD is contacted for the bank-id. Hence instead of searching all the banks the TLD helps to access the target bank directly. Use of TLD makes it possible to use single bankset for the entire LCC, otherwise the expensive searching mechanism used in T-DNUCA cannot afford it. Using TLD and removing the mandatory on-chip communications, TLD-NUCA improves the performance (CPI) by 6.5% and 13% as compared to T-DNUCA and T-SNUCA respectively. Comparing with CC which is a private LLC based design, TLD-NUCA gives 4.8% improvements in terms of CPI. TLD-NUCA has 3.08% energy overheads as compared to T-SNUCA. Scalability of TLD-NUCA is done by multi-layered 3D tiled architecture using one bankset (of 16 banks) per layer. It gives 8.8% and 9% improvements over T-SNUCA for the designs having 32-cores and 64-cores respectively.

Since majority of the block requests can be served by the local banks without contacting TLD, the entire portion of TLD is not required to accessed at all time.

Shutting down some TLD portion is not possible as the corresponding locations may contain valid informations. The best option is to use the concept of Drowsy cache [109] and keep the inactive locations in low power mode.

Chapter 8

Conclusion and Future Work

This research work is motivated to improve the performance of LLC in TCMP by better utilisation of it. To improve the LLC utilisation we worked in two directions: (i) improve the local utilisation of each bank and (ii) improve the global utilisation by distributing the loads among all the banks. This chapter sums up the major contributions of this thesis along with the future direction of research.

8.1 Summary of Contributions

To improve the local utilisation we used the concept of dynamic associativity management (DAM). CMP-VR, CMP-SVR and FS-DAM are proposed for this. In these techniques the heavily used sets are allowed to share the idle ways from other lightly used sets. CMP-VR reserves some ways from each set as reserve storage (RT); the rest of the storage is called normal storage (NT). A heavily used set can use the reserve storage of any other sets. In this manner the associativity of the entire cache can be managed dynamically. An additional tag array called TGS is required in CMP-VR for managing the RT section. CMP-VR improves the local utilisation by distributing the loads of heavily used sets with others sets. Improvement in utilisation causes reduction in MPKI. Reduced MPKI also reduces the number of main memory accesses hence improves the AMAT. Improved AMAT

allows more number of instructions to be executed within a particular time interval. Hence the overall CPI improves.

CMP-VR has some additional hardware overheads for the use of its fully associative TGS. CMP-SVR is proposed to reduce the hardware overheads of CMP-VR. It significantly reduces the energy consumption. The energy consumption of CMP-SVR as compared to baseline is less than 2%. The performance of CMP-SVR is slightly less than CMP-VR but it is still 6.76% better than the baseline. After observing the behavior of CMP-SVR we found that the performance of CMP-SVR can be further improved by better distribution of its fellow-sets. Hence we proposed FS-DAM. The hardware overheads of FS-DAM is slightly more than CMP-SVR but it improves the performance by 6.15% as compared to CMP-SVR.

To increase the global utilisation among the banks of TCMP, we first proposed a DNUCA based architecture called T-DNUCA. It divides the banks into multiple banksets and a block can be placed in any bank within a particular bankset. Migration and cascading replacement used in T-DNUCA helps to distribute the loads among multiple banks (within same bankset). A smart placement policy is also proposed to reduce the block searching time in T-DNUCA. Better block placement reduces the block searching time in T-DNUCA and hence improve performance. T-DNUCA gives 6.59% improvements over the baseline architecture (T-SNUCA).

The T-DNUCA is further improved by proposing TLD-NUCA. It minimizes the search time of T-DNUCA and allows to further improve the global utilisation. It has only a single bankset for better global utilisation of all the banks. A centralised tag directory (TLD) is used for faster block searching. TLD-NUCA gives 13% better performance than T-SNUCA and 6.5% better than T-DNUCA. The centralised TLD causes additional hardware overheads. The energy consumption of TLD-NUCA is 3.08% more than T-SNUCA. However, this overhead can be reduced by several energy optimization techniques from the literature.

Figure 8.1 summaries the contributions of this thesis.

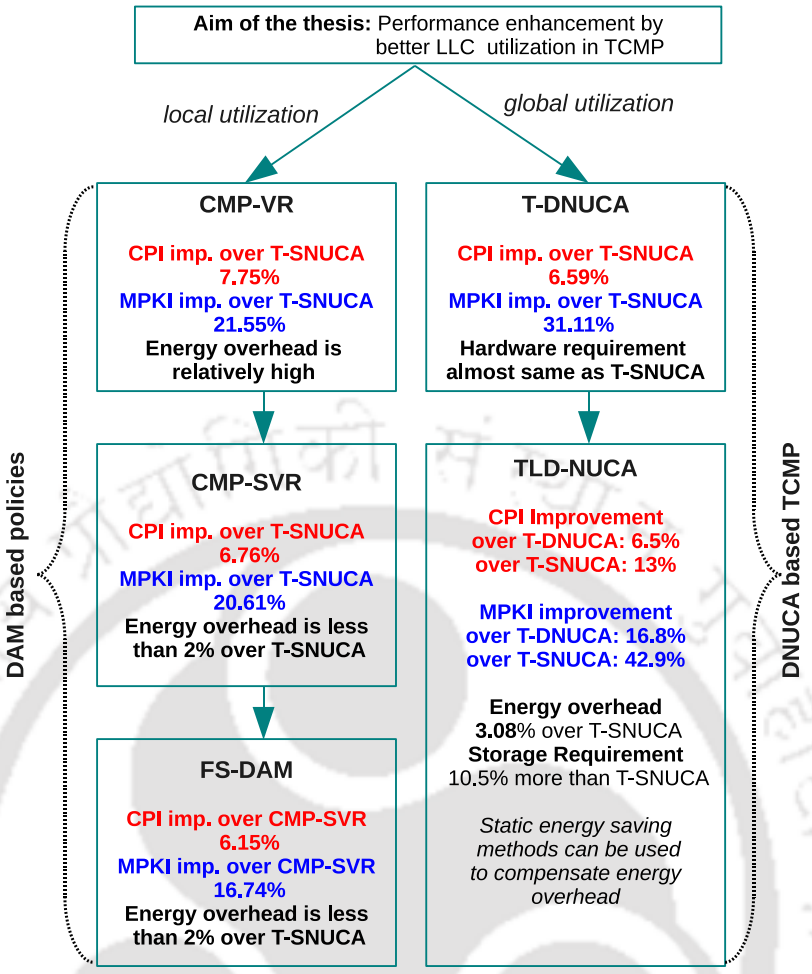


FIGURE 8.1: Summary of the thesis contributions. Note that the performance of the proposed architectures shown in this figures are only for one configuration per architecture (**4MB**). The architectures are experimented for various cache configurations.

8.2 Scope for Future Work

The contributions of this thesis can be extended in several ways. We outline some of the possible future research directions:

- T-DNUCA and TLD-NUCA improve performance for better global utilisation. Implementing the DAM based policies like CMP-SVR and FS-DAM in each bank of TLD-NUCA (or T-DNUCA) may further improve their local utilisation.

- The hardware overhead of TLD-NUCA (both energy and storage) can be reduced by using smaller sized banks with better local utilisation. It has been

observed that on average 24% blocks in L2 are useless as they are exclusively owned by some L1 [110]. These blocks, called stale blocks, cannot be used without contacting the owner. Such blocks can be removed from the bank to allow other blocks to be placed there. Doing so can improve the local utilisation factor of each bank.

Bibliography

[1] B. M. Beckmann and D. A. Wood, "Managing Wire Delay in Large Chip-Multiprocessor Caches," in *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2004, pp. 319–330.

[2] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2007, pp. 3–14.

[3] C. Bienia, "Benchmarking Modern Multiprocessors," Ph.D. dissertation, Princeton University, Jan. 2011. [Online]. Available: http://parsec.cs.princeton.edu/

[4] G. Moore, "Cramming More Components Onto Integrated Circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, Jan. 1998.

[5] R. Balasubramonian, N. P. Jouppi, and N. Muralimanohar, *Multi-Core Cache Hierarchies*. Morgan and Claypool Publishers, 2011.

[6] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 2006.

[7] M. Zhang and K. Asanovic, "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors," *Proceedings of the 32nd annual international symposium on Computer Architecture (ISCA)*, vol. 0, pp. 336–345, 2005.

[8] M. K. Qureshi, D. Thompson, and Y. N. Patt, "The V-Way Cache: Demand Based Associativity via Global Replacement," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 2, pp. 544–555, May. 2005.

[9] J. Lira, C. Molina, and A. Gonzalez, "HK-NUCA: Boosting Data Searches in Dynamic Non-Uniform Cache Architectures for Chip Multiprocessors," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, May. 2011, pp. 419–430.

[10] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2003, pp. 55–66.

[11] M. Kandemir, F. Li, M. Irwin, and S. W. Son, "A novel migration-based NUCA design for Chip Multiprocessors," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2008, pp. 1–12.

[12] J. Chang and G. S. Sohi, "Cooperative Caching for Chip Multiprocessors," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2006, pp. 264–276.

[13] A. Agarwal, J. Hennessy, and M. Horowitz, "Cache performance of operating system and multiprogramming workloads," *ACM Transactions on Computer Systems*, vol. 6, no. 4, pp. 393–431, Nov. 1988.

[14] A. El-Moursy and F. Sibai, "V-Set Cache Design for LLC of Multi-core Processors," in *Proceedings of the IEEE 14th International Conference on High Performance Computing and Communication and IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS)*, 2012, pp. 995–1000.

[15] S. Gupta, H. Gao, and H. Zhou, "Adaptive Cache Bypassing for Inclusive Last Level Caches," in *Proceedings of the IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS)*, 2013, pp. 1243–1253.

[16] A. Jaleel, M. Mattina, and B. Jacob, "Last level cache (LLC) performance of data mining workloads on a CMP - a case study of parallel bioinformatics workloads," in *Proceedings of The 12th International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2006, pp. 88–98.

[17] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," *ACM SIGOPS Operating Systems Review*, vol. 36, pp. 211–222, Oct. 2002.

[18] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A NUCA substrate for flexible CMP cache sharing," in *Proceedings of the 19th annual international conference on Supercomputing (ICS)*, 2005, pp. 31–40.

[19] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Optimizing Replication, Communication, and Capacity Allocation in CMPs," *ACM SIGARCH Computer Architecture News*, vol. 33, pp. 357–368, May. 2005.

[20] D. Sanchez and C. Kozyrakis, "The ZCache: Decoupling Ways and Associativity," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2010, pp. 187–198.

[21] D. Rolán, B. B. Fraguela, and R. Doallo, "Adaptive Line Placement with the Set Balancing Cache," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 529–540.

[22] H. Kapoor, S. Das, and S. Chakraborty, "Static energy reduction by performance linked cache capacity management in Tiled CMPs," in *Proceedings of the 30th ACM/SIGAPP Symposium On Applied Computing (SAC)*, 2015.

[23] L. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[24] W. Wong and J.-L. Baer, "Modified LRU policies for improving second-level cache behavior," in *Proceedings of the 6th International Symposium on High-Performance Computer Architecture (HPCA)*, 2000, pp. 49–60.

[25] Y. Xie and G. H. Loh, "PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 174–183, Jun. 2009.

[26] A. chow Lai, "Dead-block prediction and dead-block correlating prefetchers," in *In Proceedings of the 28th International Symposium on Computer Architecture (ISCA)*, 2001, pp. 144–154.

[27] M. Kharbutli and Y. Solihin, "Counter-Based Cache Replacement and Bypassing Algorithms," *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 433–447, Apr. 2008.

[28] Z. Huang, Z. Mingfa, and X. Limin, "LvtPPP: Live-Time Protected Pseudopartitioning of Multicore Shared Caches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 8, pp. 1622–1632, Aug. 2013.

[29] H. Ghasemzadeh and S. Fatemi, "Pseudo-FIFO Architecture of LRU Replacement Algorithm," in *Proceedings of the IEEE 9th International Multitopic Conference (INMIC)*, Dec. 2005, pp. 1–7.

[30] M. Zahran, "Cache Replacement Policy Revisited," in *Proceedings of the The Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD) held in conjunction with the International Symposium on Computer Architecture (ISCA)*, Jun. 2007.

[31] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 381–391, Jun. 2007.

[32] A. Jain, A. Shrivastava, and C. Chakrabarti, "LA-LRU: A Latency-Aware Replacement Policy for Variation Tolerant Caches," in *Proceedings of the 24th International Conference on VLSI Design (VLSI Design)*, 2011, pp. 298–303.

[33] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, "A Case for MLP-Aware Cache Replacement," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2, pp. 167–178, May. 2006.

[34] K. Morales and B. K. Lee, "Fixed Segmented LRU cache replacement scheme with selective caching," in *Proceedings of the IEEE 31st International Performance Computing and Communications Conference (IPCCC)*, 2012, pp. 199–200.

[35] F. Juan and L. Chengyan, "An Improved Multi-core Shared Cache Replacement Algorithm," in *11th International Symposium on Distributed Computing and Applications to Business, Engineering Science (DCABES)*, 2012, pp. 13–17.

[36] M. K. Qureshi and Y. N. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2006, pp. 423–432.

[37] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip," in *Proceedings of the conference on Design, automation and test in Europe (DATE)*, 2004, pp. 20 890–.

[38] M. Zhang and K. Asanovic, "Victim replication: maximizing capacity while hiding wire delay in tiled chip multiprocessors," in *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*, 2005, pp. 336–345.

[39] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler, and D. Burger, "On-Chip Interconnection Networks of the TRIPS Chip," *IEEE Micro*, vol. 27, no. 5, pp. 41–50, Sep. 2007.

[40] M. B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal, "Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures," in

Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA), 2003, pp. 341–.

[41] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip network," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2004, pp. 188–197.

[42] P. Halwe, S. Das, and H. Kapoor, "Towards a Better Cache Utilization Using Controlled Cache Partitioning," in *IEEE 11th International Conference on Dependable, Autonomic and Secure Computing (DASC)*, Dec. 2013, pp. 179–186.

[43] C. J. Janraj, T. Kalyan, T. Warrier, and M. Mutyam, "Way Sharing Set Associative Cache Architecture," in *Proceedings of the 25th International Conference on VLSI Design (VLSID)*, 2012, pp. 251–256.

[44] L. Hao, L. Tao, L. Guanghui, and X. Lunguo, "Private cache partitioning: A method to reduce the off-chip missrate of concurrently executing applications in Chip-Multiprocessors," in *Proceedings of the 3rd International Conference on Computer Research and Development (ICCRD)*, vol. 4, Mar. 2011, pp. 254–259.

[45] H. K. Kapoor, L. Chatterjee, and R. Yarlagadda, "Clustered Caching for Improving Performance and Energy requirements in NoC based Multiprocessors," in *Proceedings of the International Conference on Computer Design (CDES)*.

[46] M. S. Papamarcos and J. H. Patel, "A Low-overhead Coherence Solution for Multiprocessors with Private Cache Memories," *ACM SIGARCH Computer Architecture News*, vol. 12, no. 3, pp. 348–354, Jan. 1984.

[47] R. Chang, N. Talwalkar, C. Yue, and S. Wong, "Near speed-of-light signaling over on-chip electrical interconnects," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 5, pp. 834–838, May. 2003.

[48] P. Foglia, C. A. Prete, M. Solinas, and G. Monni, "Re-NUCA: Boosting CMP Performance Through Block Replication," in *Proceedings of the 2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, 2010, pp. 199–206.

[49] A. Gupta, J. Sampson, and M. Taylor, "DR-SNUCA: An energy-scalable dynamically partitioned cache," in *Proceedings of the IEEE 31st International Conference on Computer Design (ICCD)*, oct. 2013, pp. 515–518.

[50] J. Merino, V. Puente, and J. Gregorio, "ESP-NUCA: A low-cost adaptive Non-Uniform Cache Architecture," in *Proceedings of the IEEE 16th International Symposium on High Performance Computer Architecture (HPCA)*, Jan. 2010, pp. 1–10.

[51] H. Dybdahl and P. Stenstrom, "An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors," in *Proceedings of the IEEE 13th International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2007, pp. 2–12.

[52] R. Ricci, S. Barrus, D. Gebhardt, and R. Balasubramonian, "Leveraging Bloom Filters for Smart Search Within NUCA Caches," in *In Proceedings of the 7th Workshop on Complexity-Effective Design, held in conjuction with ISCA-33*, june 2006, pp. 2–12.

[53] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[54] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan, "Enabling Software Management for Multicore Caches with a Lightweight Hardware Support," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 14:1–14:12.

[55] E. Herrero, J. Gonzalez, and R. Canal, "Distributed Cooperative Caching: An Energy Efficient Memory Scheme for Chip Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 5, pp. 853–861, May. 2012.

[56] E. Speight, H. Shafi, L. Zhang, and R. Rajamony, "Adaptive mechanisms and policies for managing cache hierarchies in chip multiprocessors," in *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*, Jun. 2005, pp. 346–356.

[57] J. Lira, T. Jones, C. Molina, and A. Gonzalez, "Beforehand Migration on D-NUCA Caches," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, oct. 2011, pp. 197–198.

[58] W. J. Dally and J. Balfour, "Author Retrospective for Design Tradeoffs for Tiled CMP On-chip Networks," in *Proceedings of the 25th ACM International Conference on Supercomputing*, 2014, pp. 77–79.

[59] J. Balfour and W. J. Dally, "Design Tradeoffs for Tiled CMP On-chip Networks," in *Proceedings of the 20th Annual International Conference on Supercomputing*, 2006, pp. 187–198.

[60] L. Han, J. An, D. Gao, X. Fan, X. Ren, and T. Yao, "A survey on cache coherence for tiled many-core processor," in *Proceedings of the IEEE International Conference on Signal Processing, Communication and Computing (ICSPCC)*, Aug. 2012, pp. 114–118.

[61] G. Kurian, O. Khan, and S. Devadas, "The Locality-aware Adaptive Cache Coherence Protocol," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 523–534, Jun. 2013.

[62] X. Wang, M. Yang, Y. Jiang, and P. Liu, "On an efficient NoC multicasting scheme in support of multiple applications running on irregular subnetworks," *Microprocessors and Microsystems*, vol. 35, no. 2, pp. 119–129, Mar. 2011.

[63] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: near-optimal block placement and replication in distributed caches," *ACM SIGARCH Computer Architecture News*, vol. 37, pp. 184–195, Jun. 2009.

[64] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir, "Design and Management of 3D Chip Multiprocessors Using Network-in-Memory," *ACM SIGARCH Computer Architecture News*, vol. 34, pp. 130–141, May. 2006.

[65] M. Hammoud, S. Cho, and R. G. Melhem, "Cache Equalizer: A Placement Mechanism for Chip Multiprocessor Distributed Shared Caches," in *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers (HiPEAC)*, 2011.

[66] J. Balfour and W. J. Dally, "Design Tradeoffs for Tiled CMP On-chip Networks," in *Proceedings of the 20th Annual International Conference on Supercomputing*, 2006, pp. 187–198.

[67] D. Zhan, H. Jiang, and S. C. Seth, "STEM: Spatiotemporal Management of Capacity for Intra-core Last Level Caches," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Washington, DC, USA, 2010, pp. 163–174.

[68] M. Chaudhuri, "Pseudo-LIFO: The Foundation of a New Family of Replacement Policies for Last-level Caches," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 401–412.

[69] D. Rolán, D. Andrade, B. B. Fraguela, and R. Doallo, "A Fine-grained Thread-aware Management Policy for Shared Caches," *Concurrency and Computation:Practice and Experience*, vol. 26, no. 6, pp. 1355–1374, Apr. 2014.

[70] J. R. Diamond, D. S. Fussell, and S. W. Keckler, "Arbitrary Modulus Indexing," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014, pp. 140–152.

[71] P. Foglia and M. Comparetti, "A workload independent energy reduction strategy for D-NUCA caches," *The Journal of Supercomputing*, vol. 68, pp. 157–182, Oct. 2013.

[72] A. Bardine, P. Foglia, G. Gabrielli, and C. A. Prete, "Analysis of Static and Dynamic Energy Consumption in NUCA Caches: Initial Results," in *Proceedings of the Workshop on MEmory Performance: Dealing with Applications, Systems and Architecture (MEDEA)*, 2007, pp. 105–112.

[73] A. Seznec, "A Case for Two-way Skewed-associative Caches," in *Proceedings of the 20th Annual International Symposium on Computer Architecture (ISCA)*, 1993, pp. 169–178.

[74] J. L. Carter and M. N. Wegman, "Universal Classes of Hash Functions (Extended Abstract)," in *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing (STOC)*, 1977, pp. 106–112.

[75] R. Pagh and F. F. Rodler, "Cuckoo Hashing," in *Proceedings of the 9th Annual European Symposium on Algorithms (ESA)*, 2001, pp. 121–133.

[76] B. Calder, D. Grunwald, and J. Emer, "Predictive sequential associative cache," in *Proceedings of the Second International Symposium on High-Performance Computer Architecture*, 1996, pp. 244–253.

[77] M. Kharbutli, K. Irwin, Y. Solihin, and J. Lee, "Using prime numbers for cache indexing to eliminate conflict misses," in *Proceedings of the IEE Proceedings Software*, 2004, pp. 288–299.

[78] L. Zhang, D. Strukov, H. Saadeldeen, D. Fan, M. Zhang, and D. Franklin, "SpongeDirectory: Flexible Sparse Directories Utilizing Multi-level Memristors," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation (PACT)*, 2014, pp. 61–74.

[79] J. Cong, K. Gururaj, H. Huang, C. Liu, G. Reinman, and Y. Zou, "An Energy-efficient Adaptive Hybrid Cache," in *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design (ISLPED)*, 2011, pp. 67–72.

[80] S. M. Khan, D. A. Jiménez, D. Burger, and B. Falsafi, "Using Dead Blocks As a Virtual Victim Cache," in *Proceedings of the 19th International Conference*

on Parallel Architectures and Compilation Techniques (PACT), 2010, pp. 489–500.

[81] E. Herrero, J. González, and R. Canal, "Elastic Cooperative Caching: An Autonomous Dynamically Adaptive Memory Hierarchy for Chip Multiprocessors," in *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA)*, 2010, pp. 419–428.

[82] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[83] L. Schaelicke and M. Parker, "The Design and Utility of the ML-RSIM System Simulator," *Journal of Systems Architecture*, vol. 52, no. 5, pp. 283–297, May. 2006.

[84] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *ACM SIGARCH Computer Architure News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[85] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Nov. 2005.

[86] A. Gutierrez, J. Pusdesris, R. G. Dreslinski, T. N. Mudge, C. Sudanthi, C. D. Emmons, M. Hayenga, and N. C. Paver, "Sources of error in full-system simulation," in *Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 13–22.

[87] M. Rosenblum and M. Varadarajan, "SimOS: A Fast Operating System Simulation Environment," Stanford, CA, USA, Tech. Rep., 1994.

[88] M. Rosenblum, E. Bugnion, S. Devine, and S. A. Herrod, "Using the SimOS Machine Simulator to Study Complex Computer Systems," *ACM Transactions on Modeling and Computer Simulation*, vol. 7, no. 1, pp. 78–103, Jan. 1997.

[89] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.

[90] S. K. Sastry Hari, M.-L. Li, P. Ramachandran, B. Choi, and S. V. Adve, "mSWAT: Low-cost Hardware Fault Detection and Diagnosis for Multicore Systems," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 122–132.

[91] J. An, X. Fan, S. Zhang, and D. Wang, "An Efficient Verification Method for Microprocessors Based on the Virtual Machine," in *Proceedings of the 1st International Conference on Embedded Software and Systems*, 2005, pp. 514–521.

[92] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, Jul. 2006.

[93] C. J. Mauer, M. D. Hill, and D. A. Wood, "Full-system Timing-first Simulation," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1, pp. 108–116, Jun. 2002.

[94] N. Qu, Y. Zhao, X. Guan, and X. Cheng, "Unichos: A Full System Simulator for Thin Client Platform," in *Proceedings of the ACM Symposium on Applied Computing (SAC)*, 2007, pp. 1552–1556.

[95] H. M. Nyew, N. Onder, S. Onder, and Z. Wang, "Verifying Microarchitecture Simulators Using Event Traces," in *Proceedings of the 28th ACM International Conference on Supercomputing (ICS)*, 2014, pp. 323–332.

[96] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega, "COTSon: Infrastructure for Full System Simulation," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 1, pp. 52–61, Jan. 2009.

[97] F. J. Ridruejo, J. Miguel-Alonso, and J. Navaridas, "Full-system Simulation of Distributed Memory Multicomputers," *Cluster Computing*, vol. 12, no. 3, pp. 309–322, Sep. 2009.

[98] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, sept 2006.

[99] N. Agarwal, T. Krishna, L.-S. Peh, and N. Jha, "GARNET: A Detailed on-chip network model inside a full-system simulator," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2009, pp. 33–42.

[100] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: a power-performance simulator for interconnection networks," in *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-35)*, 2002, pp. 294–305.

[101] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proceedings of the INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE*, 1995, pp. 24–36.

[102] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Ferret: A Toolkit for Content-based Similarity Search of Feature-rich Data," in *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2006, pp. 317–330.

[103] M. Müller, D. Charypar, and M. Gross, "Particle-based Fluid Simulation for Interactive Applications," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 2003, pp. 154–159.

[104] G. Grahne and J. Zhu, "Fast Algorithms for Frequent Itemset Mining Using FP-Trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 10, pp. 1347–1362, Oct. 2005.

[105] D. Heath, R. Jarrow, and A. Morton, "Bond Pricing and the Term Structure of Interest Rates: A Discrete Time Approximation," *Journal of Financial and Quantitative Analysis*, vol. 25, pp. 419–440, Dec. 1990.

[106] K. Martinez and J. Cupitt, "VIPS - a highly tuned image processing software architecture," in *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, vol. 2, sept 2005, pp. II–574–7.

[107] S. Das and H. K. Kapoor, "Exploration of Migration and Replacement Policies for Dynamic NUCA over Tiled CMPs," in *Proceedings of the 28th International Conference on VLSI Design (VLSID)*, 2015.

[108] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir, "Design and Management of 3D Chip Multiprocessors Using Network-in-Memory," in *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA)*, 2006, pp. 130–141.

[109] B. Fitzgerald, S. Lopez, and J. Sahuquillo, "Drowsy Cache Partitioning for Reduced Static and Dynamic Energy in the Cache Hierarchy," *International Green Computing Conference (IGCC)*, pp. 1–6, Jun. 2013.

[110] A. Basu, D. Hower, M. Hill, and M. Swift, "FreshCache: Statically and dynamically exploiting dataless ways," in *Proceedings of the IEEE 31st International Conference on Computer Design (ICCD)*, oct. 2013, pp. 286–293.

Publications Related to Thesis

Journals:

- **Shirshendu Das**, and Hemangee K. Kapoor, "Victim Retention for Reducing Cache Misses in Tiled Chip Multiprocessors," *Journal of Microprocessors and Microsystems (Elsevier)*, 38(4), 2013, pp. 263-275.

Conference Proceedings:

- **Shirshendu Das** and Hemangee K. Kapoor, "Exploration of Migration and Replacement Policies for Dynamic NUCA over Tiled CMPs," *28th International Conference on VLSI Design-2015 (VLSID 2015)*, pp.141-146, Bangalore, India.
- **Shirshendu Das**, Dhantu Buragohain and Hemangee K. Kapoor, "A Reduced Overhead Replacement Policy for Chip Multiprocessors having Victim Retention," *International Conference on Electronics and Communication Systems (ICECS 2014)*, pp.1,6, 13-14,Coimbatore, India.
- **Shirshendu Das** and Hemangee K. Kapoor, "Dynamic associativity management using fellow sets," *4th International Symposium on Electronic System Design (ISED 2013)*, pp.133-137, NTU, Singapore.
- **Shirshendu Das**, N. Polavarapu, Prateek D. Halwe and Hemangee K. Kapoor, "Random-LRU: A Replacement Policy For Chip Multiprocessors," *17th International Symposium on VLSI Design and Test (VDAT-2013)*, pp.204-213, Jaipur, India.