REUSED FREQUENCY-BASED REPLACEMENT POLICY WITH PROGRAM COUNTER PREDICTOR ON VARIOUS MEMORY ACCESS TYPES FOR LAST LEVEL CACHE MEMORY

YEE MING CHUNG

UNIVERSITI SAINS MALAYSIA

2019

REUSED FREQUENCY-BASED REPLACEMENT POLICY WITH PROGRAM COUNTER PREDICTOR ON VARIOUS MEMORY ACCESS TYPES FOR LAST LEVEL CACHE MEMORY

by

YEE MING CHUNG

**Thesis submitted in fulfilment of the requirements
for the degree of
Doctor of Philosophy**

July 2019

ACKNOWLEGEMENTS

This dissertation is dedicated to everyone in the field of microprocessor architecture especially in Last-level cache memory who is going through the journey as professional researcher at novice level that needed guidance for similar research. Throughout the entire years of earning Ph.D qualification, my supervisor Assoc. Prof. Dr. Zaini Abdul Halim has supported me in various ways namely her guidance and advice. Thanks to her dedication, she enables my research and thesis to be completed in a timely manner. In addition, I would like to express my gratitude to the School of Electrical and Electronic Engineering staff namely Mr. Jamaludin Bin Mohamad for his support and permission to allow the usage of computers in the Microprocessor laboratory. This research would not have been possible completed in a timely manner without the computers in the laboratory. Lastly, my parents whom are important people in my life give me the encouragement and morale support to go through this highly demanding intellectual journey. I would also like to thank my father particular for his willingness to support me financially.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BRRIP	Bimodal Re-reference Interval Prediction
CPU	Central Processor Unit
EHC	Expected Hit Count
FP	Frequency Priority
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HP	Hit Priority
HHT	Hit History Table
IPC	Instruction Per Cycle
L1	Level 1
L2	Level 2
LIME	Less Is MorE
LLC	Last-Level Cache
LRU	Least Recently Used
LFU	Least Frequently Used
LRFU	Least Recently Frequently Used
MRU	Most Recently Used
NRU	Not Recently Used
OPT	Optimal
PC	Program Counter
PCHM	Program Counter Hit Miss
PCHT	Program Counter Hit Table
PCMT	Program Counter Miss Table
ReD	Reused Detection
RRIP	Re-reference Interval Prediction
RRPV	Re-reference Prediction Value
RFP	Reused Frequency Promotion
RFO	Read For Only
SRRIP	Static Re-reference Interval Prediction
SRRIP-HP	Static Re-reference Interval Prediction Hit Priority
SRRIP-FP	Static Re-reference Interval Prediction Frequency Priority
SHiP	Signature-based Hit Predictor

SHCT	Signature Hit Counter Table
SPEC	Standard Performance Evaluation Corporation

POLISI PENGGANTIAN BERDASARKAN FREKUENSI GUNA SEMULA BERSAMA PERAMAL PEMBILANG PROGRAM PADA PELBAGAI JENIS MEMORI AKSES UNTUK ARAS AKHIR INGATAN SORAKAN

ABSTRAK

Pada masa ini, haluan utama penyelidikan dalam ingatan sorakan mikroprosesor adalah ingatan sorakan aras akhir (*LLC*). Saiz *LLC* biasanya antara 2 MB hingga 20 MB. Penyelidikan menumpukan pada peningkatan prestasi *LLC* dari segi *Instruction Per Cycle* (*IPC*). Polisi penggantian *Static Re-ference Interval Prediction Hit Priority (SRRIP-HP)* boleh membuang ingatan sorakan yang kerap digunakan dan akan digunakan tidak lama lagi. Dalam tesis ini, polisi penggantian baru yang menggunakan penggunaan semula frekuensi dan *Re-reference Interval Value (RRPV)* dicadangkan untuk mengurangkan masalah *SRRIP-HP* dan bertujuan meningkatkan prestasi dari segi *IPC*. Polisi penggantian ini dipanggil Promosi Frekuensi Digunakan Semula (*RFP*). Walau bagaimanapun, *RFP* terlalu awal membuang ingatan sorakan yang baru dimasukkan pada program yang *recency-friendly*. Hal ini kerana pemasukan frekuensi yang digunakan semula dan *RRPV* adalah secara statik dan tanpa mengetahui ingatan sorakan yang mana perlu disimpan lebih lama. Oleh itu, peramal blok mati yang baru dipanggil peramal *Program Counter Hit Miss (PCHM)* digunakan untuk mengubah keputusan pemasukan *RFP*. Peramal *PCHM* membenarkan ingatan sorakan yang diramalkan akan digunakan semula pada masa depan disimpan lebih lama dalam ingatan *LLC*. Ini dapat meningkatkan peluangnya untuk digunakan semula dan meningkatkan prestasi.

Gabungan *RFP* dan Peramal *PCHM* dipanggil *RFP-PCHM*. Walau bagaimanapun, *RFP-PCHM* mempunyai prestasi yang buruk dalam beberapa program kerana pelbagai jenis akses memori mengemaskini pada Peramal *PCHM* yang sama. Jenis-jenis akses memori adalah *load/RFO*, *prefetching* dan *writeback.* Tingkah laku kekerapan akses memori ini mungkin berbeza. Oleh itu, operasi mengubah pengemaskinian dan pemasukan Peramal *PCHM* dari *RFP-PCHM*; dan juga operasi mengubah keputusan promosi *RFP-PCHM* harus berdasarkan jenis akses memori. Polisi penggantian ini disebut *RFP-PCHM-I*. Dalam penilaian prestasi, *RFP*, *RFP-PCHM* dan *RFP-PCHM-I* telah diuji pada penilaian *single-thread*, *multi-thread* dan *multi-program*. Eksperimen ini diuji dengan menggunakan penanda aras *SPEC CPU 2006* dan *Cloudsuite* pada simulator *ChampSim*. Polisi penggantian *Least Recently Used* (*LRU*) digunakan sebagai penanda aras. *LRU* membuang ingatan sorakan yang paling lama tidak digunakan dalam memori *LLC*. *RFP*, *RFP-PCHM* dan *RFP-PCHM-I* mempunyai prestasi yang lebih baik berbanding dengan *LRU*, di mana prestasi masing-masing ialah 0.72%, 0.77% dan 2.52% dari segi purata geometri berwajaran *IPC* yang dinormalisasi. *RFP-PCHM-I* juga secara keseluruhan mempunyai prestasi yang lebih baik daripada *SHiP++* dan *Leeway*. Kesimpulannya, penggunaan semula frekuensi bagi *RFP* telah meningkatkan prestasi *SRRIP-HP*. Tambahan pula, dengan adanya Peramal *PCHM* bagi *RFP-PCHM* juga telah meningkatkan prestasi *RFP*. Di samping itu, pengubahsuaian bagi pengemaskinian dan keputusan kemasukan bagi peramal *PCHM* dan pengubahsuaian keputusan promosi bagi *RFP-PCHM* telah juga meningkatkan prestasi *RFP-PCHM*.

REUSED FREQUENCY-BASED REPLACEMENT POLICY WITH PROGRAM COUNTER PREDICTOR ON VARIOUS MEMORY ACCESS TYPES FOR LAST-LEVEL CACHE MEMORY

ABSTRACT

Currently, the main trend of microprocessor cache research is on the last level cache (LLC) memory. The typical LLC memory size is between 2 MB to 20 MB. The researches focus on improving the performance of LLC memory in terms of instruction per cycle (IPC) by using replacement policy. Static Re-reference Interval Prediction Hit Priority (SRRIP-HP) replacement policy can evict cache line that is frequently used and might be reused very soon. In this thesis, a new replacement which uses reused frequency and Re-reference Interval Value (RRPV) is proposed to mitigate the problem of SRRIP-HP with the aim to improve performance in terms of IPC. This replacement policy is called as Reused Frequency Promotion (RFP). However, the proposed RFP evicts newly inserted lines too early on recency-friendly benchmarks. This is because RFP inserts the reused frequency and RRPV in static manner without knowing which lines need to be kept longer. Therefore, a new dead block predictor called Program Counter Hit Miss (PCHM) Predictor is use to modify the insertion decision of RFP. The PCHM Predictor allows line that is predicted to be reused in future very soon to be stored longer in LLC memory. This can increase its chance of being reused and improve performance. The combination of RFP and PCHM Predictor is called RFP-PCHM. However, RFP-PCHM performs worse on some benchmark due to different type of memory access are updating the PCHM Predictor. Each type of memory access namely load/RFO, prefetching and writeback has different reused behavior. Therefore, the updating and insertion decision of

PCHM predictor from RFP-PCHM is modified; and the promotion decision of RFP-PCHM is also modified based on the type of memory access namely load/RFO, prefetching and writeback. This replacement policy is called RFP-PCHM-I. In the performance evaluations, the proposed RFP, RFP-PCHM and RFP-PCHM-I are tested at single-thread, multi-thread and multi-program evaluation using SPEC CPU 2006 and Cloudsuite benchmarks on ChampSim simulator. Least Recently Used (LRU) replacement policy is used as baseline. LRU evicts cache line that is the least recently used in LLC memory. The RFP, RFP-PCHM and RFP-PCHM-I outperform LRU with 0.72%, 0.77% and 2.52% respectively in terms of weighted geometric mean of normalized IPC. RFP-PCHM-I also overall outperforms the Signature-based hit predictor ++ (SHiP++) and Leeway. In conclusion, the reused frequency of RFP have further improved the performance of SRRIP-HP. In addition, the PCHM predictor of RFP-PCHM have further improved the performance of RFP. The modification of updating and insertion decision of PCHM predictor; and promotion decision of RFP-PCHM have further improve the performance of RFP-PCHM.

CHAPTER 1

INTRODUCTION

1.1 Introduction

In recent decades, the advancement of semiconductor fabrication technology continues to shrink the size of transistor in microprocessor. This leads to more transistors being packed in a given die size and allows more functions to be included in microprocessor designs. There are many research and development activities on improving the performance of microprocessors which results in fast and power efficient processors. However, main memory (Random Access Memory) and primary storage (Hard Disk) do not have the same rapid advancement as microprocessor. The long memory latency from these memories has negatively impacted the performance of microprocessor (Chu and Park, 2014; Das and Kapoor, 2016; Liu *et al.*, 2017; Xue *et al.*, 2017; Bartolini, Foglia and Prete, 2018).

Cache memory is a solution to migrate the problem of long memory latency from main memory (Xue *et al.*, 2017). The cache memory is a fast but low capacity memory (P Michaud, 2016). Cache memory works by exploiting the spatial and temporal locality of programs. It stores data and instructions that have higher chance of being reused by processor’s cores. Most of the programs only execute a small part of its entire codes at one time. Therefore, it is not necessary to load all the data and instructions into memory. This works in favor of cache memory because it is usually small in capacity and can be expensive to fabricate large cache memory. There has been a trend for quite some time now to integrate two-level or three-level cache memory hierarchy to further improve the performance of microprocessor.

Since last decade, microprocessors have not much improvements in terms of clock speed due to "power wall" problem (Shi *et al.*, 2016). Engineers in microprocessor field have moved to design multi-core processor as a way to achieve performance improvement (Das and Kapoor, 2016; Kedar, Mendelson and Cidon, 2017; Liu *et al.*, 2017; Valls *et al.*, 2017). Those multi-core processors have two-level or three-level cache memory hierarchy. Normally, the last level cache (LLC) memory is shared among the cores (Kharbutli and Sheikh, 2014; Sridharan and Seznec, 2017). Each core has its own private cache memory namely level 1 (L1) or level 2 (L2) or both which cannot be accessed by other cores (Fernández-Pascual, Ros and Acacio, 2017). LLC memory is L2 and L3 of two-level and three-level cache memory hierarchy.

Cache memory utilizes 40% to 60% of total microprocessor silicon area and consumes as much as 50% of total power consumption of entire microprocessor (Chu and Park, 2014; Asad *et al.*, 2017; Chakraborty and Kapoor, 2017; Kim, Lee and Kim, 2017; Yasir Qadri *et al.*, 2017). LLC is the largest cache memory compared to L1 and L2 cache memory. It is between 2 MB to 20 MB depending on the architecture design of the processor and the market segments that the designers are aiming for. While, a L1 cache memory can be between 16 KB to 64 KB and L2 cache memory can be between 64 KB to 512 KB for each core. The power consumption and silicon area usage of LLC is significant larger than L1 and L2 cache memory.

The performance and power consumption of cache memory can be improved by using replacement policy. Example of replacement policies are Re-reference Interval Prediction (RRIP) (Jaleel *et al.*, 2010) and Signature-based Hit Predictor (SHiP) (Wu *et al.*, 2011). A good replacement policy can evict lines that will not be used soon while keeping the lines longer that will be used soon. This can improve the performance of microprocessor because the cache memory is able to handle demands from processor’s cores with minimum delay. If the replacement policy evicts the wrong lines, the processor’s cores have to wait longer for the data or instructions to be retrieved from main memory or primary storage. This can degrade the performance of the processor if the replacement policy frequently evicts the wrong lines. Power consumption can be improved because a good replacement policy can avoid the processor from accessing main memory and primary storage which consumes more power.

In the cache memory, there are two events, namely hit and miss. Hit occurs if the requested data or instructions are in the cache memory. Miss occurs if the requested data or instructions are not in the cache memory.

A cache memory replacement policy is a subsystem of cache memory, which the main function (eviction operation) is to evict cache line when the cache memory set is fully occupied and miss occurs. Besides eviction operation, cache memory replacement policy tasks involve insertion and promotion operation. When a new line is stored in the cache memory after eviction operation, the insertion operation needs to update the associated replacement policy information of the line. When a hit occurs (data or instructions are in cache memory), the promotion operation also needs to update the replacement policy information of the line. The information determines whether the line will be considered as replacement or not during eviction operation at miss. For example, information maintained by Least Recently Used (LRU) replacement policy is called as recency. At miss (data or instructions are in cache memory), LRU evicts the least recently used line based on the recency value.

During insertion, LRU inserts the line as most recently used. At hit, LRU promotes the line as most recently used.

Figure 1.1 shows the operation of replacement policy using LRU as an example on 4-way set associative cache memory. Each rectangular represents the recency of cache line where recency of 0 means the most recently used and recency of 3 means the least recently used. During eviction operation, LRU looks for cache line with recency of 3 as replacement as shown in Figure 1.1 a). It searches logically from way-0 to way-3 and found cache line at way-3 is the candidate of replacement. After eviction, new cache line is stored in LLC memory and the recency needs to be updated as shown in Figure 1.1 b). The update is called as insertion. Using Figure 1.1 a) as initial condition, the insertion operation is done by assigning recency value as 0 at way-3. The other recency values are increased by one. During hit, the recency also needs to be updated as shown in Figure 1.1 c). LRU assigned the recency value as 0 at way-2 because hit occurs at way-2. Then, the other recency values are increased by one.

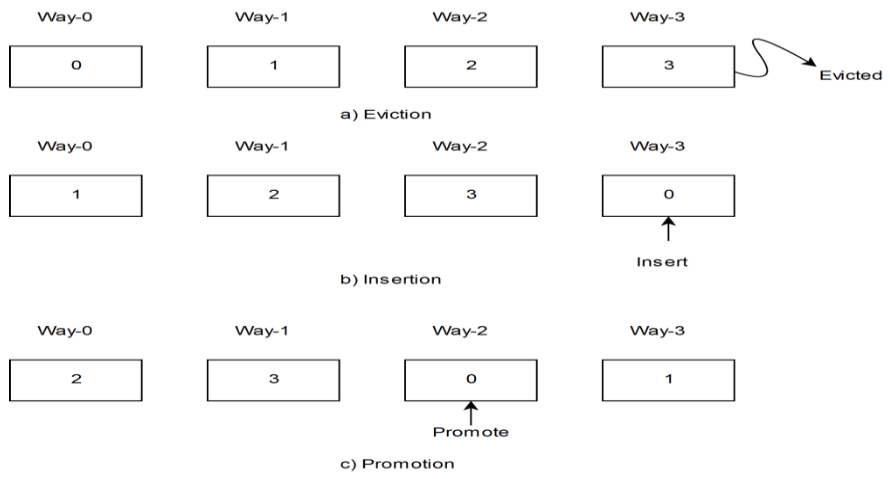


Figure 1.1 LRU Operation on 4-way set associative

In recent years, there have been a lot of researches focused on LLC replacement policy because there is a performance gap between conventional replacement policy namely LRU and theoretical Optimal replacement policy (Do *et al.*, 2015). This gap shows that there is a room of performance improvement on LLC replacement policy. Optimal is an offline replacement policy which required looking ahead for cache line that will be reused in furthest in future as replacement. It requires significant modification of the simulator to allow Optimal to check the future access memory stream to identify cache line that will be reused far in future during miss.

Furthermore, LLC replacement policy can improve the performance and power consumption of multi-core processor by avoiding memory access that has high latency and power consumption penalty (Park, Lee and Kim, 2016). The multi-core processor configurations give a new challenge to be efficiently managed of the LLC utilization due to competition of multiple program threads on shared LLC (Yin *et al.*, 2016).

Instruction per cycle (IPC) is the conventional metric used to measure the performance of microprocessor. The IPC is obtained from simulation result using simulator. Normally, the simulations are ran between 100 million to 8 billion instructions (Faldu and Grot, 2017; Jain and Lin, 2016; Jiménez and Teran, 2017; Vakil-Ghahani *et al.*, 2018; Young *et al.*, 2017). In 2nd International Cache Replacement Championship, the current performance improvement of replacement policies in terms of IPC over LRU at LLC memory is between 0.5% and 3.3% which are mentioned in Section 2.5 (Crc2.ece.tamu.edu, 2017).

1.2 Motivation and Research Problems

Re-reference interval prediction (RRIP) is a hardware efficient replacement policy based on reused prediction. RRIP keeps lines that are predicted to be re-

referenced in near-intermediate future longer while keeps lines that are predicted to be re-referenced in distant future for short time at LLC memory (Peng, Yu and Zhu, 2015).

Each cache line is assigned with RRIP value which represents the likelihood of reused in future. The RRIP value is called Re-reference Prediction Value (RRPV). The RRPV can be from 0 until 3. The high RRPV means the line are predicted to be re-referenced in distant future while low value RRPV means the line are predicted to be re-referenced in near-intermediate future. However, the RRIP replacement policy does not consider the reused frequency of each line. If the RRPVs are the same in the cache set, the first line will be evicted, normally line at way-0. There is a possibility for frequently reused line to be evicted even though it has the highest reused frequency value.

Figure 1.2 shows the operation of RRIP where the value in square box represents RRPV. When miss occurs and cache memory is full, RRIP evicts cache line with RRPV of 3 as shown in Figure 1.2 a). RRIP searches logically from way-0 to way-3 for the first line with RRPV of 3 to be evicted. RRPV of 3 means the line would not be reused or will be reused in distant future. After eviction, RRIP always insert the new line with RRPV as 2 as shown in Figure 1.2 b), which the line is predicted to be reused far in future. This line will be evicted soon because its time in cache memory will be short. When hit occurs, the RRPV is promoted to 0 which means the line is predicted to be reused in near-intermediate future as shown in Figure 1.2 c). This can increased the chance the line will be reused in future because it stay longer in cache memory.

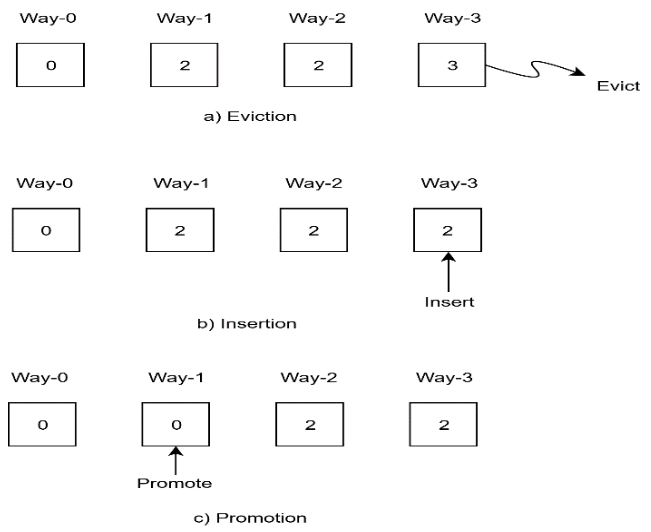
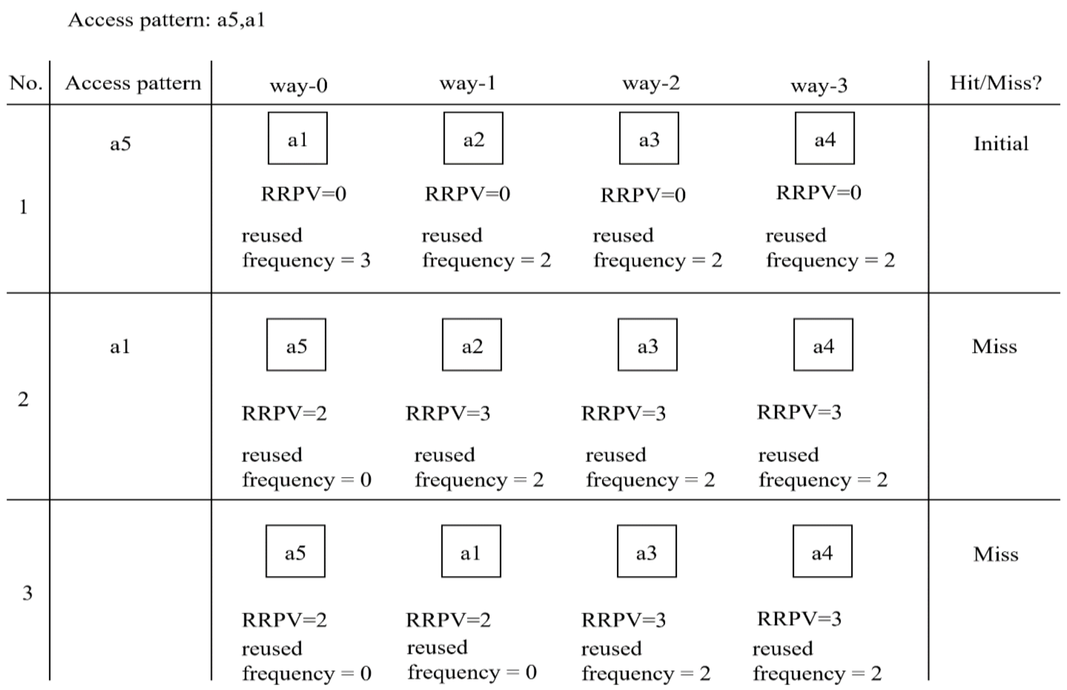


Figure 1.2 RRIP Operation on 4-way set associative

Figure 1.3 shows the problem of RRIP with a theoretical example and detail operation. At initial condition as shown in Figure 1.3 at row No.1, the a1, a2, a3 and a4 line have been reused (reused frequency) many times where a1 has been reused 3 times while a2, a3 and a4 have been reused 2 times. The RRPV for all lines are 0 which represents near-intermediate re-reference interval prediction. This means the lines will be reused very soon. When a5 is accessed, it is a miss since there is no a5 resides in the cache lines. According to eviction operation of RRIP, it searches logically from way-0 to way-3 for the line with RRPV of 3. However, there is no such line. So, RRIP increased all the RRPVs of all lines by 1 and continue to search again. The process of searching and increasing are repeated until the first line with RRPV of 3 is found. This line is the candidate of the replacement. As shown in Figure 1.3 at row No.2, line at way-0 is selected as replacement. The RRPV of a5 is updated as 2 which represents long re-reference interval prediction. This means the a5 line is predicted to be reused in distant future. Next, a1 is accessed and a miss has occured. RRIP evicts line at way-1 because it is the first line with RRPV of 3. The a1 is updated with RRPV of 2 as shown in row No.3.

RRIP does not keep any reused frequency information of each memory access of line in the LLC memory. RRIP can evict line that has been reused frequently and will be reused again very soon. It is shown in Figure 1.3 where RRIP evicts the line at way-0 that will be reused very soon. If RRIP keeps the reused frequency information, it can avoid evicting that line by evicting other line instead. This will avoid a miss and can improve the performance if RRIP is reused frequency aware.

To the best of author’s knowledge in this research field, there is no research to use the reused frequency and the RRPV to make replacement decision in LLC memory replacement policy. Hence, a new replacement policy is proposed which uses the reused frequency and the RRPV to make replacement decision in this thesis with the goal of performance improvements in terms of IPC. The used of reused frequency for the new proposed replacement policy is different compared to Least Frequently Used (LFU) and Least Recently Frequently Used (LRFU). LFU evicts randomly when there are more than one least frequently used lines. The proposed replacement policy uses RRPV to find line that is predicted to be re-reference in distant future in the group of least reused lines. On the other hand, LRFU uses floating point number to represent the likehood of being reused while the proposed replacement policy uses simple counter to count the reused frequency. LRFU is more complex in terms of calculating the floating point number.


Access pattern: a5,a1

No.	Access pattern	way-0	way-1	way-2	way-3	Hit/Miss?
1	a5	a1 RRPV=0 reused frequency = 3	a2 RRPV=0 reused frequency = 2	a3 RRPV=0 reused frequency = 2	a4 RRPV=0 reused frequency = 2	Initial
2	a1	a5 RRPV=2 reused frequency = 0	a2 RRPV=3 reused frequency = 2	a3 RRPV=3 reused frequency = 2	a4 RRPV=3 reused frequency = 2	Miss
3		a5 RRPV=2 reused frequency = 0	a1 RRPV=2 reused frequency = 0	a3 RRPV=3 reused frequency = 2	a4 RRPV=3 reused frequency = 2	Miss

Figure 1.3 RRIP Problem

A dead block predictor is another method to improve the performance of LLC memory replacement policy in terms of IPC (Diaz Maag *et al.*, 2017; Faldu and Grot, 2017; Jain and Lin, 2016; Jiajun *et al.*, 2017; Wu *et al.*, 2011; Vakil-Ghahani *et al.*, 2018; Young *et al.*, 2017). The predictor works by modifying the insertion decision of the replacement policy. The cache lines will be evicted later or sooner based on its prediction whether the new incoming line will be reused or not in future. One of the replacement policies that is using the dead block predictor is Signature-based Hit Predictor (SHiP).

SHiP has been used to improve the existing RRIP replacement policy performance by modifying its insertion decision (Wu *et al.*, 2011). SHiP introduces three different signatures which are program counter (PC), memory region (Mem) and instruction sequence (ISeq); and named the SHiP replacement policy as SHiP-PC, SHiP-Mem and SHiP-ISeq respectively. The SHiP predicts whether a line will be reused in future based on the counter value associated with the signature in the signature history counter table (SHCT). If the counter value of SHCT is larger than zero, it means the line is predicted to be reused soon. Hence, SHiP inserts the RRPV as 2 which allow the line to be stored in LLC memory longer. While, the line is predicted to be reused far in future if the counter value is zero. Therefore, SHiP inserts the RRPV as 3 which allow the line to be evicted very soon. SHiP works better than RRIP because it can make better insertion decision by using reused behaviors of instructions which share the same signature (Wu *et al.*, 2011; Sardashti and Wood, 2017). The improvement of SHiP over LRU is 9.7% in average while the RRIP is only 5.5% (Wu *et al.*, 2011).

However, SHiP does not make any different prediction when the SHCT counter reaches saturation because lines have the highest chance of being reused very soon compared to counter value that is more than zero but not saturated (Young *et al.*, 2017). This is an opportunity to further improve the performance of SHiP. In addition, the updating of SHCT is imbalance during hit and miss (Young *et al.*, 2017). Every time a hit occurs at a line, the associated SHCT counter is updated. However, SHCT is only updated once if the evicted line is not reused at all during miss. This can become a problem because the SHCT cannot be updated fast enough when the program behavior changes from high reused to no reuse as SHCT needs more time and many misses to change from high counter value to low counter value.

To address these problems, a new dead block predictor based on SHiP is proposed. This new dead block predictor is used to modify the proposed replacement policy insertion decision in this thesis.

Modern multi-core processors have prefetcher implemented which is another method to improve performance (Gupta, Gao and Zhou, 2013; Li *et al.*, 2014).

Prefetcher fetches instruction or data that may be used or may not be used into LLC memory before the program is accessing it (Panda and Balachandran, 2016). When processor's core access data or instructions at LLC memory due to program miss, this type of memory access is called load. Prefetching and load are the type of memory accesses in microprocessor that may have different reused frequency behavior. Writeback is another type of memory access. It is accessed by the processor's core to maintain data coherence in memory hierarchy.

If data or instructions are not available at LLC memory during prefetching access, this will trigger a miss. The replacement policy needs to evict a line and requests memory access to main memory. The evicted line might be reused in future while the incoming prefetched lines from main memory may or may not be reused at all. This introduces a problem at LLC memory because prefetching memory accesses essentially competing LLC memory space and may interfere with replacement policy eviction decision if the prefetched line is not reused for program execution (Panda, 2016; Sridharan, Panda and Seznec, 2017). Recent study has shown that 95% of prefetched lines are not reused after their first load hit (Seshadri *et al.*, 2015). Besides prefetching, writeback access is another memory access that can compete with LLC memory resources. Writeback memory access may not improve the execution speed of program because the memory latency due to a miss of writeback access can be hidden (Jiajun *et al.*, 2017, Lee, Kim and Chung, 2018).

To the best of author's knowledge, there are not many replacement policy researches that consider and treat the load, writeback and prefetching memory access differently in terms of promotion and insertion operation replacement policy (Diaz Maag *et al.*, 2017; Faldu and Grot, 2017; Jiajun *et al.*, 2017; Jiménez and Teran, 2017; Young *et al.,* 2017). This provides an opportunity to make different insertion and promotion replacement policy decision based on type of memory access.

In this thesis, the proposed replacement policy promotion decision is modified based on type of memory access namely load, prefetching and writeback. In addition, the updating and insertion decision of the proposed dead block predictor is also modified based on the type of memory access.

1.3 Objectives

In short, the main objective of this thesis is to investigate the performance in terms of IPC of the new replacement policy which use reused frequency and RRPV in making decision of replacement. In addition, the performance of the proposed replacement policy is investigated further using a proposed dead block predictor; and by modifying the insertion and promotion decision based on the type of memory accesses. Below are the objectives of the thesis:

(i) To improve the performance in terms of IPC over RRIP at LLC memory by using the proposed replacement policy that uses reused frequency and RRPV to make replacement policy decision.

(ii) To improve the performance in terms of IPC over the proposed replacement policy in (i) by using the proposed dead block predictor based on SHiP.

(iii) To improve the performance in terms of IPC by modifying the proposed dead block predictor in (ii) to make updating and insertion decision differently; and modifying the proposed replacement policy in (ii) to make different promotion decision based on type of memory access namely, load, writeback and prefetching memory access.

1.4 Scope Limitation

In this thesis, the main focus is on the research of algorithm-based replacement policy by using software simulation to evaluate the performance on LLC memory. The proposed replacement policy is developed with the intention of hardware implementation in future. Therefore, the additional storage hardware is presented in this thesis because it is easier to be estimated. In this thesis, there is no hardware-based development on field programmable gate array (FPGA) or using silicon die fabrication.

There are three evaluations considered in this thesis namely single-thread, multi-thread and multi-program. A single-thread is a sequence of program instructions execution that is managed by operating system scheduler as shown in Figure 1.4 a). Multi-program is basically a group of single-thread program which runs individually on each core. Each thread in the multi-program setup has its own codes (instructions) and data as shown in Figure 1.4 b). Multi-thread is a single program with a parent thread that can issue multiple child threads which are distributed and run among all the cores. These programs have a group of common data and codes that share among all cores as shown in Figure 1.4 c). In this thesis, single-thread evaluation is evaluated at single-core. Although multi-thread and multi-program evaluation can be evaluated in large number of cores, only quad-core configuration is evaluated in this thesis.

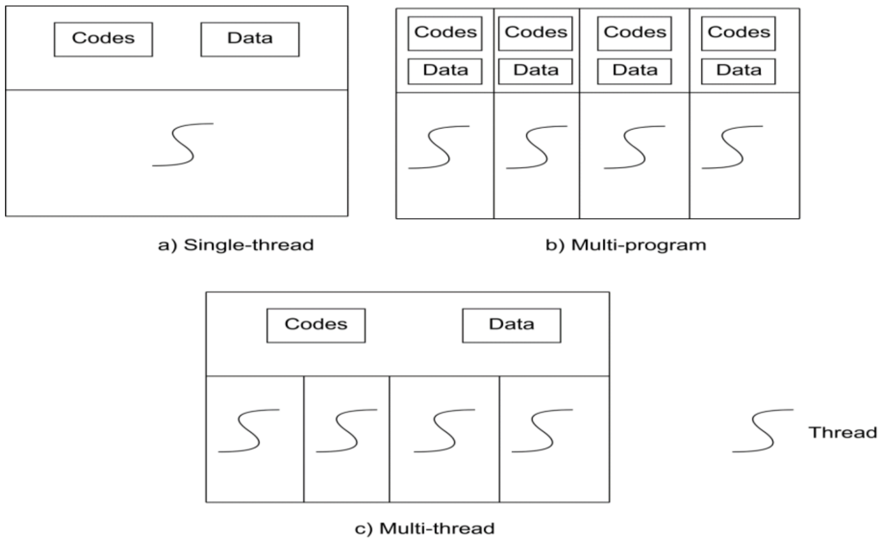


Figure 1.4 Visualization of a) single-thread, b) multi-program, and c) multi-thread

Furthermore, the thesis is not about design exploration in terms of cache size, line size and associative at LLC memory. Therefore, these parameters are fixed where the line size is 64 bytes and the associative is 16 at LLC memory. The LLC memory size is 2 MB and 8 MB for single-core and quad-core configuration respectively. Only single-core and quad-core are evaluated because the simulator only supports these configurations. Simulator used is called as ChampSim. It is modeling an quad-core Intel i7 Skylake processor with 8 MB LLC memory.

1.5 Outline of Thesis

In this thesis, six chapters are presented. The first chapter is introduction of the thesis. This chapter briefly explains the background of research topic, the motivation, the research problems, the objectives of research and the research scope. The second chapter is literature review of previous researches on LLC memory replacement policy. In addition, this chapter also elaborates more on the background of cache memory field, the benchmarks and the simulator used. In chapter three, the reused frequency analysis of SPEC CPU 2006 and Cloudsuite benchmarks are presented. This chapter also describes the proposed replacement policy in detail along with its performance.

Chapter four describes the proposed dead block predictor and the performance of the proposed replacement policy with the proposed predictor. In chapter five, a number of modifications on the updating of proposed dead block predictor; insertion and promotion operation of the proposed replacement policy on different type of memory accesses are discussed. Chapter five also presents the performance of the modified proposed replacement policy by comparing its performance with previous related works. Lastly, chapter six concludes the thesis findings.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, the background of LLC memory is presented. Detail reviews of related replacement policies at LLC memory are also presented. In addition, the simulator used for this research is also discussed. Prefetching is also discussed in this Chapter. The details of benchmarks used in this study are also presented.

2.2 Background of LLC memory

Typical memory access patterns found in LLC memory are recency-friendly, thrashing, streaming or scanning and mixed (Jaleel *et al.*, 2010). The recency-friendly access pattern occurs when memory access have high temporal locality. The thrashing access pattern occurs when there is a cyclic access pattern that is larger than the cache size. The streaming or scanning access pattern occurs when a series of memory access does not show any reused behavior. This access pattern is normally seen in streaming video or scanning a larger file. The mixed access pattern is a mixture of recency-friendly and a burst of streaming or scanning access patterns.

The characteristic of cache memory are categorized into split or unified, mapping function, write policy and inclusivity. For split cache architecture, the data and instruction are stored separately in data and instructions array respectively. While the unified cache architecture store all the data and instructions in a single array.

LLC memory capacity is smaller compared to main memory. The main memory can be a few Gigabytes in size. There are three methods to map the main memory address to a cache memory. The first one is called set-associative. The main memory address is divided into three fields namely, tag, set and word. The LLC memory is divided into a few arrays. If there are two arrays, the set-associative is called 2-way and so on in base number of 2. Each array has of its own tag, data or instructions array. All the tag arrays are accessed simultaneously to find the address. If matched, the associated data or instruction array is accessed. Any data or instructions can be stored in one of the data or instruction arrays. The placement of data or instructions in the array is dependent on replacement policy decision. The number of candidates available for replacement is based on the number of associativity. A typical associativity of LLC memory is 16-way. The energy constraint of microprocessor does not allow the associativity of LLC memory to increase further around 16-way (Sridharan and Seznec, 2017).

The second method of mapping is direct mapping. The address is also divided into tag, line and word field. Direct mapping is a special case of set-associative where the associativity is one. If there is a miss, only one candidate is available for replacement.

Fully-associative only has single tag; and data or instruction array. The main memory address is divided into tag and word fields. Data or instructions can be stored at any lines in the array. When finding the right address, all tags are checked simultaneously.

Write policy is used to maintain data coherence among all cores. When a cache line is updated at high level cache namely L1 or L2, the write policy needs to decide when to update all other copies of the same data which may be located at L1 or L2 or LLC or main memory. Write back policy will only update the data if the line which stores the data is selected for replacement. Write through policy will update all the data copies every time the new data are written.

The inclusivity of cache memory keeps a copy of lines from higher level cache memory at lower level cache memory such as LLC memory. The redundant lines at LLC memory are wasting memory resources that might not to be used at all. Furthermore, these redundant lines do not help to improve much performance because the processor's core will mostly access high level cache memory than LLC memory. Therefore, LLC memory becomes less effective in utilization.

In this thesis, 16-way set-associative of unified LLC memory with cache line size of 64 bytes is used. The writeback and non-inclusivity are also used in this thesis. These cache characteristics modeled on the simulator as shown in Section 2.7.

There are many factors that make the utilization of LLC less effective. One of the factors is the dead block phenomena (Zhibin, Mingfa and Limin, 2013). Dead block is referring to line that is not referenced again by processor's cores. These lines are not referenced at all or only re-referenced a few times for a short period and later become useless in the entire time at LLC memory. Another issue is cache pollution where high reused line get evicted instead of low reused line (Shahtouri and Ma, 2015). Workload such as scanning a file that is larger than the cache size exhibits cache pollution. The diverse memory behavior of various applications in LLC memory of multi-core processor is a problem for cache memory replacement policy. This is because the replacement policy needs to be aware of this diversity of workload mix and able to assign a fair amount of resources for each application (Sridharan and Seznec, 2017).

Least Recently Used (LRU) is the replacement policy baseline used when comparing performance of replacement policy for microprocessor (Qureshi *et al.*, 2007; Wang *et al.*, 2018). This replacement policy evicts cache line that is least recently used by using recency information which is associated with each cache line. LRU does not perform as good as Optimal replacement policy at LLC memory because of diverse reused characteristic between multiple concurrent running programs which interfered each other (Park, Park and Mahlke, 2016). Hence, there is a need to research a better replacement policy than LRU at LLC memory for performance improvement.

2.2.1 Basic Architecture of LLC Memory

Basic architecture of LLC memory consists of tag array, data or instruction array, cache controller and replacement policy. Example of basic architecture of 4-way LLC memory is shown in Figure 2.1. The tag array contains a portion of main memory address, valid bit and dirty bit. The first few most significant bits (MSB) of main memory address are stored in tag array. The "**Tag**" array is used to find whether the data or instructions are available in cache memory or not. The "**Valid**" bit is to indicate whether there are data or instruction stored or not. The "**Dirty**" bit is to indicate whether the line has been written or not. The data or instruction array is further divided into 4 arrays because 4-way set associative. Each row in the array is called cache line or line. This is where at least one data or instruction is stored.

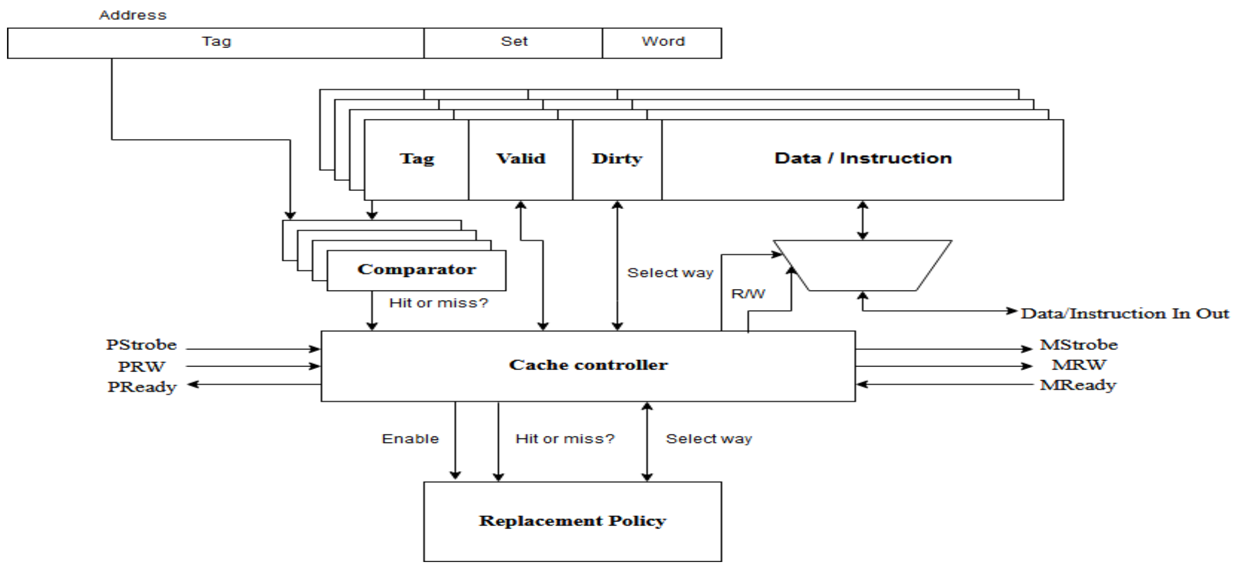


Figure 2.1 Basic architecture of 4-way associative (Stallings, 2016)

The cache controller is a finite state machine (FSM) that receives read or write data or instructions access from processor’s cores via PStrobe which enable the controller itself and PRW for read or write signal. The cache controller determines whether the data or instruction is in the cache memory by checking the tag array for the specific MSB address. If the data or instructions are there, the cache controller activates the right latches or multiplexers using Select way and R/W that controls the data flow of data or instructions array for read and write. The cache controller will send PReady if the data or instructions are ready to be sent to processor’s core. If a miss occurs and LLC memory is full, the cache controller requests the replacement policy for a replacement using Enable, hit or miss and select way signal. While at the same time, the cache controller passes the accesses to main memory via MStrobe and MRW signal. The cache controller will read the data or instructions from main memory if the MReady signal is set. The cache controller also decides whether the line needs to be updated at main memory first to maintain data coherence before the new incoming instructions or data are written on the evicted line. During hit, the cache controller also sends signals to replacement policy to update the replacement policy information.

The replacement policy decides which line to be evicted in the LLC cache set based on the information. The replacement policy stores information such as recency for LRU. The replacement policy tells which line to be evicted to the cache controller via select way.

The performance of microprocessor measures in terms of IPC is normally depending on the number of hits. The IPC is calculated as *Total of instructions run / Total number of clock cycle*. A hit in LLC memory can reduce the total clock cycle that the microprocessor's core has to wait for the data or instructions to arrive. If miss occurs in LLC memory, the microprocessor's core has to wait longer because data or instructions are being retrieved from slower main memory. Typically, the memory latency of LLC memory when hit occurs is around 20 clock cycles. The 20 cycles is calculated at the moment the microprocessor's core requested until data or instructions arrived from LLC memory. When a miss occurs at LLC memory, the typical memory latency to retrieve data or instructions from main memory to microprocessor's core is around 200 clock cycles. If there are multiple misses, the *Total number of clock cycle* will increased. This decreases the IPC because the inverse relationship of IPC with total number of clock cycle. On the other hand, more hits at LLC memory means the *Total number of clock cycle* will decrease. This increased the IPC. The higher value of IPC means higher performance because it takes less time to execute all the instructions.

2.3 Background of Basic LLC Replacement Policy Operation

Figure 2.2 shows the basic operation of LRU replacement policy on a 4-way associative which is divided into three namely eviction, insertion and promotion operations. When a memory access is requested by processor’s core, the address of the memory access is check against Tag in the LLC memory as shown in operation ①. If Tag is matched with the address, then a hit has occurs. This means data or instructions are available in LLC memory. When a hit occurs, it is an opportunity to update the LRU replacement policy information of the line. This operation ② is called promotion operation and it occurs only during hit. As shown in Figure 2.2, the recency of the hit line is updated as 0 to represent it is the most recently used line. At the same time a copy of data or instructions are send to processor’s core.

If no match is found between Tag and address in LLC memory, then is miss has occurs. This means data or instructions are not available in LLC memory. During miss, eviction operation is done first then follow by insertion operation. The eviction operation involves evicting lines to make memory space for new incoming block from main memory. The eviction operation uses the replacement policy information to determine the best candidate of replacement as shown in ②. In this case, LRU find line with recency of 3 as replacement which represents the least recently used. At the same time, the memory request is send to main memory. LLC waits for the data or instructions to arrive from main memory. After arrived, insertion operation updates the replacement policy information for the new incoming block from main memory as shown in ③. During this operation, the recency information is updated as 0. At the same time, a copy of data or instructions are stored at LLC memory and sent to processor’s core.

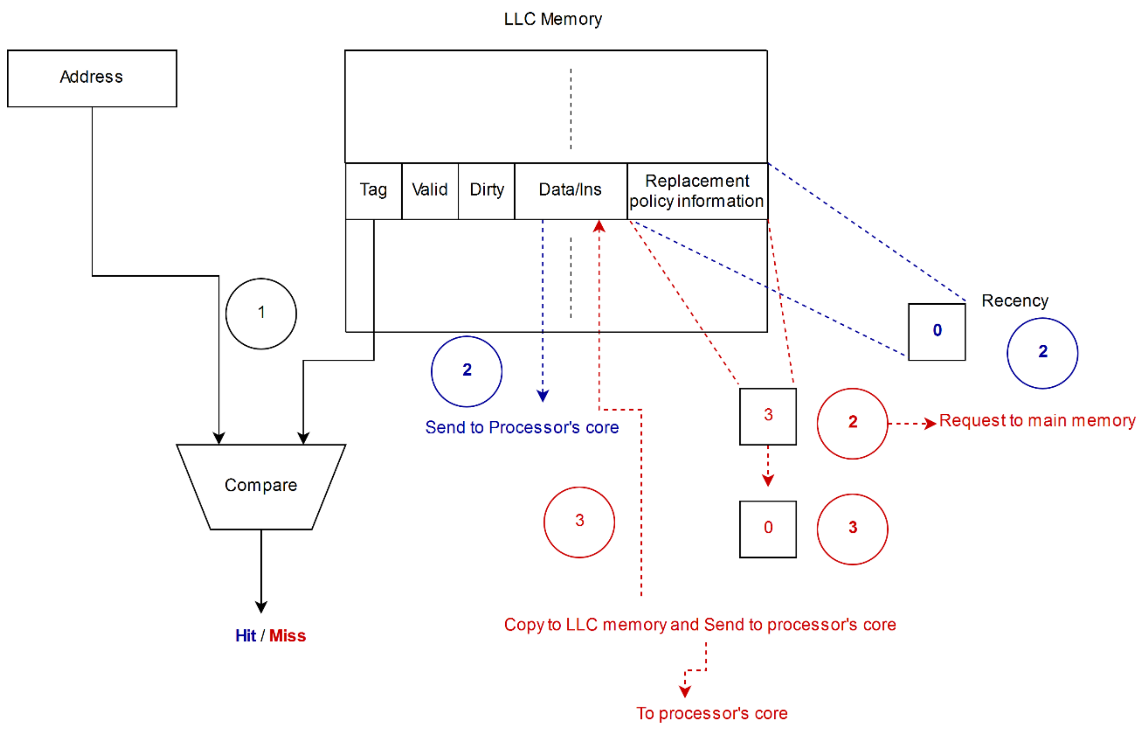


Figure 2.2 Operation of Promotion, Eviction and Insertion of LRU

2.4 Conventional replacement policy

Least Recently Used (LRU) and Least Frequently Used (LFU) replacement policy are the conventional and commonly used replacement policies which are developed decades ago (Ji *et al.*, 2017). LRU and LFU are based on recency and frequency of the referenced line in the cache respectively. In this section, detailed information is presented on these conventional replacement policies.

2.4.1 Least Recently Used (LRU)

One of the conventional replacement policies that are still currently used as a comparison in research is LRU. The LRU replacement policy keep tracks the recency of lines in cache memory by using chain-based concept (Wei *et al.*, 2017). The recency of lines are arranged logically from left to right as shown in Figure 2.3 where the value in each square box represents recency. The most recently used (MRU) line is placed at the most left the chain which represented by recency value of

0. While, the least recently used (LRU) is placed at the most right of the chain which represented by value recency of 3. LRU replacement policy evicts line at LRU position in the chain.

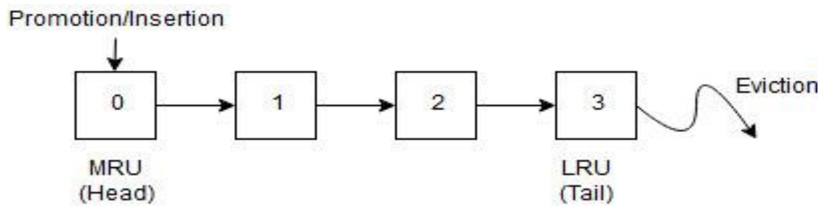


Figure 2.3 LRU chain

At higher level cache memory namely, L1 or L2, LRU is the usual replacement policy used. LRU keeps line that has short reused distance or in other words, high temporal locality (Jin *et al.*, 2013). The reused distance of the line means the number of unique line that has been accessed at a cache memory set before that line is referenced again. Let say, line A has been referenced, then followed by four unique lines at the same set. Next, line A is referenced again. So, the reused distance is four since four unique lines are referenced before line A is referenced again.

The operation of the LRU can be explained in terms of promotion, eviction and insertion. When a hit occurs, the promotion operation will move the line to MRU position. All the lines that are positioned at left side of the hit line is shifted rightward by one position before that hit line is moved to MRU position. When a miss occurs, LRU replacement policy selects the line at LRU position for replacement during eviction operation. After eviction operation, LRU replacement places the new incoming line at MRU position after the entire chain is shifted rightwards by one position. This insertion operation assumes that the incoming line will be re-referenced very soon. Therefore, it is inserted at the most recently used