WLRU CPU Cache Replacement Algorithm

(Thesis Format: Monograph)

by

Qufei Wang

Graduate Program in Computer Science

Submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy

Faculty of Graduate Studies
The University of Western Ontario
London, Ontario
December, 2006

Your file Votre référence
ISBN: 978-0-494-50498-7
Our file Notre référence
ISBN: 978-0-494-50498-7

THE UNIVERSITY OF WESTERN ONTARIO
FACULTY OF GRADUATE STUDIES

CERTIFICATE OF EXAMINATION

The thesis by
Qufei Wang
entitled

WLRU CPU CACHE REPLACEMENT ALGORITHM

is accepted in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy

Date	Chair of Examining Board

Abstract

A CPU consists of two parts, the CPU cores and the CPU caches. CPU caches are small but fast memories usually on the same die as the CPU cores. Recently used instructions and data are stored in CPU caches. Accessing CPU caches takes a quater to five nano seconds, but accessing the main memory takes 100 to 150 nano seconds. The main memory is so slow that the CPU is idle for more than 80% of the time waiting for memory accesses. This problem is known as the memory wall. The memory wall implies that faster or more CPU cores are of little use if the performance of CPU caches does not improve.

Generally, larger CPU caches have higher performance but the improvement is very small. A smarter CPU cache replacement algorithm is of more potential. The CPU cache replacement algorithm decides which cache content to be replaced. Currently, Least Recently Used (LRU) replacement and its variants are most widely used in CPUs. However, the performance of LRU is not satisfactory for applications of poor locality, such as network protocols and applications. We found that there is a pattern in the memory references of these applications that makes LRU fails. Based on this discovery, we developed a new CPU cache replacement called Weighted Least Recently Used (WLRU). Trace based simulations show that WLRU has significant improvement over LRU for applications of poor locality. For example, for web servers, WLRU has 50% fewer L2 cache misses than LRU. This means WLRU can immediately improve the performance of web servers by more than 200%.

CPU caches have been intensively studied in the past thirty years. WLRU has by far the biggest improvement. Our studies also indicate that WLRU is very close to the theoretical upper limit of cache replacement algorithms. This means any further improvement in CPU cache performance will have to come from changes to the software. In future work, we will investigate how to write OS and software to have better CPU cache performance.

Acknowledgements

I would like to gratefully acknowledge the supervision of Professor Hanan Lutfiyya during this work. Many thanks to her for her patience, tolerance and support.

I am grateful to all my friends in Computer Science Department, University of Western Ontario. From the staff, Janice Wiersma and Cheryl McGrath are especially thanked for their care and attention.

Finally, I am forever indebted to my wife Min and my parents. The support from Min is the source of strength helped me through the many years.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

The speed of CPUs is much faster than the speed of the main memory. CPU caches are used to bridge the speed gap. A CPU cache is a small memory which is usually on the same die as the CPU [dLJ03]. A CPU cache is much faster than the main memory but much smaller in size. Instructions and data recently accessed from the main memory are stored in the CPU cache. When the CPU requests an address, the CPU cache is checked. If found in the cache, it is called a *cache hit* otherwise it is called a *cache miss*. The proportion of addresses found in the cache is called the *cache hit rate*. The difference in accessing time between the main memory and the CPU cache is defined as the *cache miss penalty*. This work assumes that the cache miss penalty is measured using the number of CPU cycles needed to retrieve the information from the main memory. For example, if accessing the CPU cache requires only one CPU cycle but accessing the main memory requires 100 CPU cycles, the cache miss penalty is 100. Currently, the cache miss penalties of most CPUs are already much more than 100 [Jac03, Tho03, FH05]. Since most CPU caches are smaller than the program image in the main memory, when the CPU cache is full, then an existing cache entry is chosen to be replaced. A *cache replacement algorithm* decides the cache entry to be replaced. The most commonly used CPU cache replacement algorithm is Least Recently Used (LRU) replacement [PH05]. LRU replacement evicts the cache entry which is least recently accessed. The use of LRU is based on the assumption that programs exhibit the property of temporal locality, which is phrased as ‘recently accessed items are likely to be accessed in the near future [HP96].

In the past twenty years, the speed of CPUs doubled every 18 months, but the memory speed increased only 7% each year [HP02]. The speed gap between the CPU and the main memory keeps widening [1], but the CPU cache hit rate is seldom higher than 99% [HP02]. Assuming a cache hit rate of 99% and a cache penalty of 100, the CPU is idle for 50% of the time. Currently, main stream CPU speeds are between 2 to 4 GHz, and the main memory is clocked between 500 MHz to 800 MHz. Besides the data transfer time, the main memory made of DRAM (Dynamic Random Access Memory) also has a large latency. The latency of DRAM is the delay between the receiving of the read request and the readiness of data for transfer. The latency of the current DDR DRAM memory is at least 90 nano seconds, and the total transfer time of a cache line is around 120 ns [2]. Assuming a CPU speed of 1 GHz, the cache miss penalty is 120 CPU cycles. Faster CPU speeds have even larger cache miss penalties. Faster DRAM technologies helps little since the latency of these faster memory remains constant, if not even longer. The Semiconductor Industry Association (ISA) is now calculating cache miss penalties of more than 300 CPU cycles [FH05]. If the cache hit rate can not be improved, as the speed gap reaches a specific point, further increasing CPU speeds will not generate any gain in effective computing power. This is known as the *Memory Wall* problem [WM95].

The CPU cache is a dominant factor in computing power. Generally, a larger CPU cache has higher hit rates. However, there is a limit on the die for CPU caches. Recent processors have already spent 50% of the die area and more than 80% of the transistors on CPU caches [PHS98]. Larger CPU caches are unlikely unless revolutionary circuit technologies are used. This suggests other approaches to improve the CPU cache performance besides increasing the size of CPU caches should be examined.

One approach to improving CPU cache performance is to find better cache replacement algorithms. LRU is currently the most widely used CPU cache replacement. LRU was developed decades ago, and current computing environments are very different from that time.

1.2 Contributions

The contributions of this work include the following:

[1]Although the CPU speed stagnated in recent years, there are always faster CPUs coming. For example, IBM Power6 is targeted around 5G Hz. (http://realworldtech.com/page.cfm?ArticleID=RWT101606194731)

[2]source: www.powerlogix.com/downloads/SDRDDR.pdf

Property of Short Lifetime. This work presents an analysis of the pattern of memory references of programs. Of special interest is the study of inter-reference gaps (IRG) and reference counts of addresses. The reference count of an address is the number of times that the address is referenced. An Inter-Reference Gap (IRG) is defined as the number of references between two consecutive references of an address. Per set IRG values are IRG values of an individual cache set. Our studies find that the majority of per set IRG values are small. This is especially true at the first-level (L1) cache where it is found that 90% of the per set IRG values are of size one. At the level two (L2) cache, per set IRG values are still small. This provides strong evidence of temporal locality. However, our studies also show that a large portion of addresses have low reference counts. At the L2 cache, nearly 50% of all addresses are referenced only once, and nearly 90% of all addresses are referenced under ten times. This pattern is named the *property of short lifetime*. This suggests that LRU is less effective for programs that have a large portion of its addresses with low reference counts, since LRU does not distinguish between addresses with low reference counts and addresses with high reference counts, which turns out to be the case of many networked applications.

Development of a New Cache Replacement Algorithm. Based on the property of short lifetime a new cache replacement algorithm, which is a modification of LRU, was developed. This new algorithm is referred to as *Weighted Least Recently Used (WLRU)* . Simulations show that WLRU has significantly fewer cache misses than LRU for network protocols and applications. For other programs, such as SPEC benchmark programs, the difference in the hit rates of WLRU and LRU is unnoticeable. This means the superiority of WLRU over LRU for network protocols and applications does not harm the performance of traditional applications like SPEC benchmarks. WLRU can replace LRU in general purpose CPUs.

Example Circuit and Simulator. An example circuit of a CPU cache using WLRU replacement is presented in this work. The circuit shows that the cost of implementing WLRU is minimal. WLRU is requires less than 3% of more space than LRU. A trace based simulator is also developed. The simulator implements WLRU, LRU, pseudo-LRU replacements, and off-line optimal replacement. The simulator is written in Java and contains bookkeeping information not found in other simulators. This information is used to investigate the behavior of different cache replacements and designs.

1.3 Outline of Dissertation

The rest of this work is organized as follows.

Chapter 2 describes related research in cache replacement algorithms. Some background introduction to CPU cache designs is included. Cache replacements in fields other than CPU caches, such as database buffer caches, are introduced in chapter 2. Chapter 2 also discusses previous studies on the impact of cache performance on network protocols and applications.

Chapter 3 discusses the empirical analysis methods used for the study of the memory accesses of programs and the results of the analysis. The property of short lifetime is introduced in chapter 3.

Chapter 4 discusses the locality characteristics of network protocols and applications.

Chapter 5 presents a new CPU cache replacement algorithm called the *WLRU* replacement algorithm.

Chapter 6 presents an example hardware implementation of WLRU cache in CPU. The hardware cost of implementing WLRU replacement is analyzed and compared with the cost of implementing LRU replacement in CPU caches.

Chapter 7 describes the design of the CPU cache simulator. The CPU cache simulator in this work is different from other CPU cache simulators in that its focus is on the cache replacement algorithms. Other unique features include the victim analysis and a fast implementation of the off-line optimal replacement algorithm.

Chapter 8 presents a simulation comparison of the hit rates of WLRU and LRU replacement algorithms on the SPEC benchmark programs and network protocols and applications. Simulation results of the off-line optimal replacement (OPT) are provided to better understand the improvement of WLRU over LRU.

Chapter 9 presents conclusions and a plan for future research.

Chapter 2

Background and Related Research

CPU caches have been intensively studied for the last thirty years. This chapter briefly examines the design issues of current CPU caches and research on CPU cache performance of network protocols and applications.

2.1 Background on CPU Caches

This section introduces the basics of CPU cache design.

2.1.1 Memory Hierarchy

Modern CPUs have a hierarchy of memories. A higher level of memory is faster than a lower level of memory, but the higher level memory is also smaller in size and more expensive. The highest level or levels of memory are called the CPU cache. Currently, the CPU cache is on the same die as the CPU execution unit. CPU caches are always made of SRAM (Static Random Access Memory). The main memory is made of DRAM (Dynamic Random Access Memory). Visiting the main memory incurs a long latency, typically around 100 ns, and then fetching the data costs another 2 ns each word [1]. The time to visit the main memory is equal to several hundred CPU cycles. CPU caches can reduce the time to a single CPU cycle since CPU caches are made of SRAM and are usually on the same die as the CPU execution units. Figure 2.1 shows a hierarchy of memories. The first and the second levels

[1]source: www.powerlogix.com/downloads/SDRDDR.pdf

of the hierarchy are CPU caches, and the third level is the main memory. The fourth level of the hierarchy is the virtual memory on the disk storage. CPU caches contain a subset of the main memory.

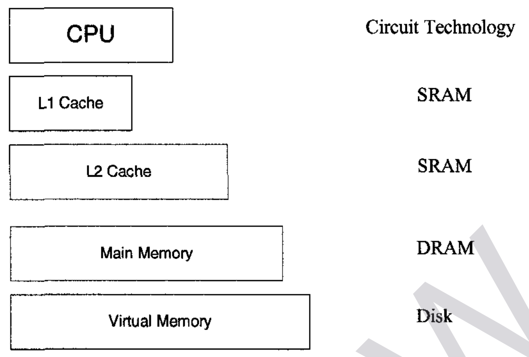


Figure 2.1: The structure of an four levels memory hierarchy.

2.1.2 Cache Lines and Cache Hits

The unit of transfer of data between the CPU execution unit and the cache is a word. Data transfer between the cache and the main memory is multiple memory words. This takes advantage of the spatial locality principle in that if one memory location is read then nearby memory locations are likely to be read [HP96]. Thus CPU caches are organized into cache lines where each cache line consists of the words read in a single transfer of data between the main memory and CPU cache. A cache line (depicted in Figure 2.2) consists of an address tag, status bits and data from the main memory. Transfer of more than one word also has advantages with respect to the memory bandwidth. The latency of visiting the main memory is amortized among multiple words. Cache lines also save space since multiple words share an address tag.

The part of the address of a main memory word is referred to as the address tag. When the CPU references a main memory word, the address tag part of the address of the main memory word is taken out. The tag of each cache line is compared with the address tag of the memory word being referenced by the CPU. If there is a match between a tag of a cache line and the address tag of the word then there is said to be a *cache hit*, otherwise it is a *cache miss*. In the case of a cache hit, the referenced word is directly accessed from the cache, avoiding the latency in retrieving the memory word from the main memory. In the case of a cache miss the referenced word is accessed from the main memory. There are two status bits in a cache line. The valid status bit is used to indicate that a cache line is not empty. The dirty status bit is set when the data in a cache line changes.

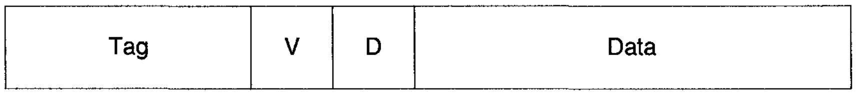


Figure 2.2: The structure of a CPU cache line.

2.1.3 Set Associative Caches

An important design aspect of CPU caches is determining where in the cache the retrieved data from main memory can be placed. If a main memory word can only be placed in a single cache location then the cache is called a *direct-mapped* cache. Figure 2.3(a) shows the mapping of a main memory word into a direct-mapped cache. The direct-mapped cache has m cache lines and the lowest log_2^m bits of the address is used to map a main memory word into the cache. When deciding cache hits or misses, direct-mapped caches only need to compare a single address tag. Thus direct-mapped caches are fast. The problem with direct-mapped cache is that it incurs more cache misses, which can be illustrated with the following example. Suppose a program generates a series of memory references such as the following: 0x1CD, 0x3CD, 0x1CD, 0x3CD. Both of the two memory words are mapped to the same cache line. This sequence of references causes a continuous stream of evictions and replacements of cache lines. Thus, direct-mapped caches are fast but also have lower hit rates. Studies [Prz90, HS89] found that reducing associativity from two-way to direct-mapped increases the miss rate by 25%.

Another approach would allow a unit of data transfer to be placed in any one of the cache lines in the cache. This is called a *fully-associative* cache. Replacement of data in a cache line only occurs when the entire cache has filled up. The replacement algorithm in a fully-associative cache can replace any cache line in the cache with the incoming cache line. Fully-associative caches are believed to have the highest hit rates [HR00] for a large number of replacement algorithms. The address tag is compared in parallel with all of the tags of all the cache lines in order to retrieve the data quickly. However, a CPU cache typically consists of hundreds of thousands of cache lines. The circuitry needed to do the parallel comparison of all tags is expensive. Thus, except for some very small caches, no CPU caches are fully associative [PH05].

A set associative cache combines concepts from direct-mapped cache and fully-associative cache. Cache lines are organized into cache sets. A main memory word can be placed in only one cache set but may be placed into any of the cache lines in the cache set. Essentially this means that a memory word can only be placed in a subset of the cache. A main memory address is divided into three fields: an address tag, set index and block offset. The set index field is used to determine the cache set. The address tag uniquely identifies the memory word and the offset is used to find the word within cache line. A set associative cache becomes a fully associative cache when the cache has only one cache set.

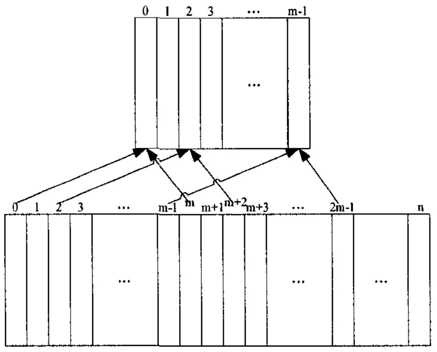
The number of cache lines in a cache set is referred to as the associativity of the cache. For example, if there are four cache lines in a cache set, the associativity is four, and the cache is called a four-way set associative cache. Figure 2.3(b) shows the mapping of a main memory word into a two-way set associative cache. The cache has the same number m of cache lines and is arranged into $m/2$ cache set. Each main memory word has two possible locations in the cache. The lowest $log_2^{m/2}$ bits of the address is used to map the word into the cache.

Figure 2.4 shows the structure an eight-way set associative CPU cache. The cache has 1024 cache sets. Each cache set has eight cache lines. Each cache line stores eight words. The address the CPU is currently referencing is stored in the address latch. The middle ten bits of the address latch is mapped to one of the 1024 cache sets. The lowest three bits is the block offset to index into the eight words of a cache line. The highest 19 bits form the address tag. All the eight address tags of a cache set are compared with the address tag in the address latch. If there is a match, the hit/miss signal indicates a cache hit or miss.

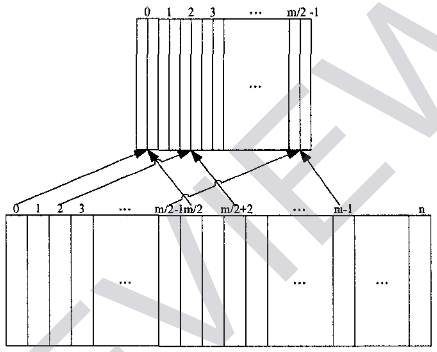
2.1.4 Multiple Level CPU Caches

Modern CPUs usually have a hierarchy of caches. Most of the current CPUs have two levels of CPU caches. The first level CPU cache is called the L1 cache and can be accessed in one or two cycles. The gate delay and wire delay limits the size of the L1 cache. Typically, the L1 cache is only 32KB or 64KB. The same speed constraint also limits the associativity of the L1 cache. To achieve high speed, the L1 cache may be direct-mapped.

The second level cache is called the L2 cache. L2 caches are typically accessed in around ten CPU cycles and are much larger than L1 caches [PH05]. The L2 cache usually has higher associativity. L2 caches can be 16 or 32 way associative. Besides the L1 and the L2 caches, some CPUs have a level three cache. L3 caches are slower and larger than L1 and L2 caches.



(a) Direct-mapped cache



(b) two-way set associative cache

Figure 2.3: The mapping of the main memory words into a direct-mapped cache and a two-way associative cache.

Currently, both the L1 and the L2 caches use LRU replacement. LRU replacement at L2 and L3 caches are actually Least Recently Missed replacement. The references at the L1 cache are invisible to the lower level caches. The most recently referenced or loaded address at the L2 or L3 cache is not necessarily the address which the CPU most recently referenced but the address most recently missed in the higher level cache. Since references to items in the L2 cache are not exactly what the CPU is currently referencing but misses from the L1 cache, LRU at L2 and lower level caches is actually least recently missed replacement algorithm. LRU at L2 cache does not exactly follow the definition of temporal locality [PHS98]. The hit rates of the LRU replacement at the L2 or L3 cache are low. This is considered by [PHS98] to be a problem of LRU.