

# Victim buffer impact on CPU performance

## --- Computer Architecture Technical Report Phase Two

Danhua Guo

George Chatzimilioudis

Department of Computer Science and Engineering

University of California, Riverside

{dguo, gchatzim}@cs.ucr.edu

## 1 Introduction

Attempting to decrease the Cycles per Instruction (CPI) of an out-of-order execution we implemented a *victim buffer* into the SimpleSim simulator. This change should affect the miss penalty and thus the CPI. We introduce two different kinds of victim buffers. The difference lies in the way it interacts with the Instruction Cache Level 1 (IL1) and the way it gives the required data to the CPU. Simulations show a speedup in performance for both victim buffer models. The result in this report is helpful to design a CPU with higher performance.

## 2 Benchmarks

We used four applications in NetBench: *crc*, *tl*, *md5* and *dh*. These applications can be divided into two categories: *crc* and *tl* are in the micro level, *dh* and *md5* are in the application level. The application introduction can be found at [8]:

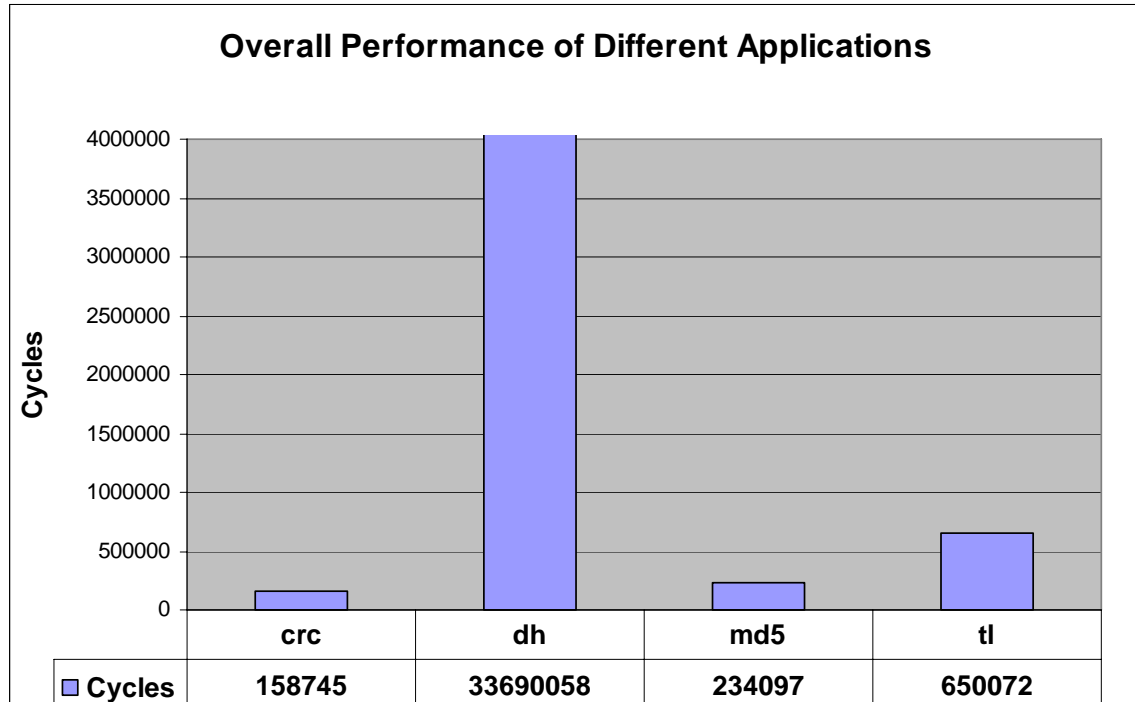
*CRC*: The CRC-32 checksum calculates a checksum based on a cyclic redundancy check as described in ISO 3309. CRC-32 is used in Ethernet and ATM Adaptation Layer 5 (AAL-5) checksum calculation.

*TL*: TL is the table lookup routine common to all routing processes. We have used radix-tree routing table which was used in several UNIX systems.

*MD5*: Message Digest algorithm creates a cryptographically secure signature for each outgoing packet, which is checked at the destination. If the received packet does not match the signature, then the receiver will detect it and discard the packet. The MD5 algorithm is commonly used for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private key under a public-key cryptosystem. MD5 is much more reliable than checksum and many other commonly used methods.

*DH*: Diffie-Hellman (DH) is a common public key encryption-decryption mechanism. It is the security protocol employed in several Virtual Private Networks (VPN's).

The complexity of the four applications is shown in Figure 1.



*Figure 2.1*

It is clear that dh is the most time consuming application we use, followed by tl and md5. crc on the other hand, costs least.

## 3 Performance Metrics

In order to measure the CPU performance, cycles per instruction (CPI) are considered. On the other hand, since two different victim buffer models are provided, we need to see which one of them is better than the other. As a result, IL1 miss rate is introduced. In addition, for each of the victim buffer model, we changed the number of entries for the cache to get the optimized configuration, with the victim buffer miss rate as a reference.

## 4 Experiment Environment

Before we start, let us first have an idea of how the cache is organized and which kind of parameters can be used to do the measurement. In SimpleScalar, there are five attributes in a description of cache: cache name (e.g. il1, il2, dl1, dl2, etc.), number of sets (e.g. 256, 512, etc.), block size (512, 1024, etc.), associativity (1, 2, 4, etc.) and replacement type (l for LRU, f for FIFO and r for Random). By combination of the details above, a typical example of a cache on instruction level 1, with a 2-way-32 sets of 512 size block and FIFO replacement policy is like: il1:512:32::2:f.

The cache size is calculated by the multiplication of the number of sets, the block size and the associativity. E.g. using the previous example, the cache size =  $512 * 32 * 2 = 32K$ .

In this experiment, we have the basic set up for SimpleScalar as following:

1. IL1: 512:32:1:l;

2. IL2=DL2: 1024:64:4:1, which is a unified level 2 cache;
3. DL1: 128:32:4:1.

#### 4.1 Original memory hierarchy

Let us first take a look at the original memory hierarchy in SimpleScalar. Figure 4.1 shows the mechanism of the memory system without a victim buffer. When the CPU asks for an instruction, it first goes into instruction level 1 (iL1). If there is a hit, the instruction is returned. Otherwise the request is passed to the next level of cache. At the second level cache, a swap will take place when the instruction is found. The result instruction from iL2 will be put into the place in iL1, where the original block is discarded depending on the replacement policy (LRU, FIFO or Random), and then return back to the CPU from iL1. If a miss occurs, the instruction request will be passed onto the main memory at the next level, which will takes a longer latency.

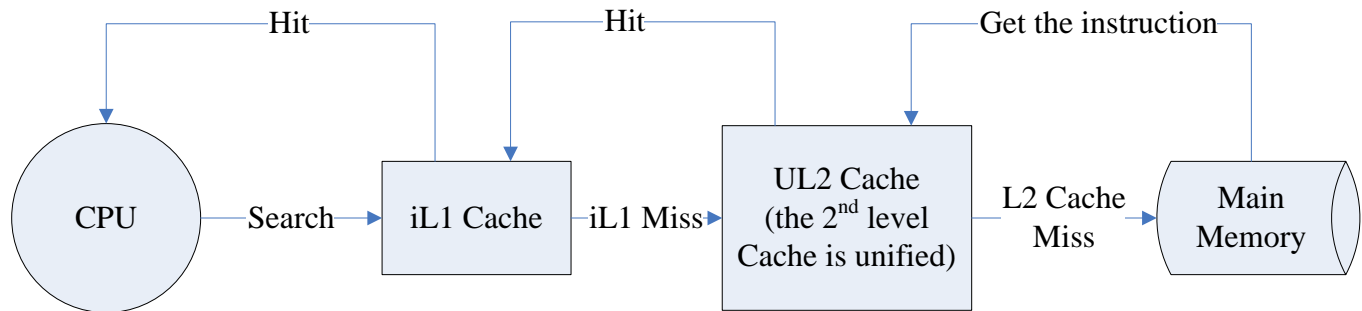


Figure 4.1 The original memory hierarchy in SimpleScalar

#### 4.2 About victim buffer

We have added a victim buffer for the level 1 instruction cache. The reason why we choose instruction rather than data cache is due to the out-weight number of instruction memory access, compared with the data memory access for network applications in NetBench

According the concept of victim buffer, when a block in IL1 is supposed to be replaced by the required instruction from the next level, instead of just discard it, we use a separate cache to save this “victim”. The temporary locality tells us the most recent accessed instruction has the highest possibility to be accessed again in the future. Therefore with this victim buffer saving all the replaced blocks from IL2, we may save some CPU time by looking for the missed instruction first in the victim buffer and then go the next level if there is a miss.

We use a fully associative cache as the victim buffer, since the size of the cache is small therefore we do not have conflict misses and the hit ratio is greater. In our implementation, a perfect parallel comparator hardware that has no latency is considered. Furthermore, the hit-latency of the victim buffer is set to 1.

In the following sections, we will introduce two different victim buffer models. The difference lies in the way they interact with the IL1 and the CPU.

#### 4.3 “Swap” victim buffer (SVB)

Figure 4.2 shows the first victim buffer model we used. When there is a miss at iL1, it first discards one of its blocks and put it into the victim buffer. Then if the required instruction is found in the victim

buffer, the result is returned to iL1 (We call this a “swap”). Otherwise, the request is passed on to the next level. Since the victim buffer is also embedded in the chip together with iL1, the latency is almost the same and much smaller than level 2 cache. And that is where we made our enhancement to the system. Meanwhile, there are some interesting points that could be further discussed: 1. When the discarded block is written from iL1 to the victim buffer, it has to override one of the blocks that already exists in the cache. This process is not stable because there is a chance (25% for a 4-entry fully associative victim buffer) of overriding the instruction block that is required by the CPU. 2. The organization of the victim buffer. We know that with a larger victim buffer, more iL1 discarded blocks could be buffered, which saves our time if there is a hit in the victim buffer. Meanwhile, it takes longer to match the block tag if there are too many entries in the cache. So what is the best configuration of the victim buffer?

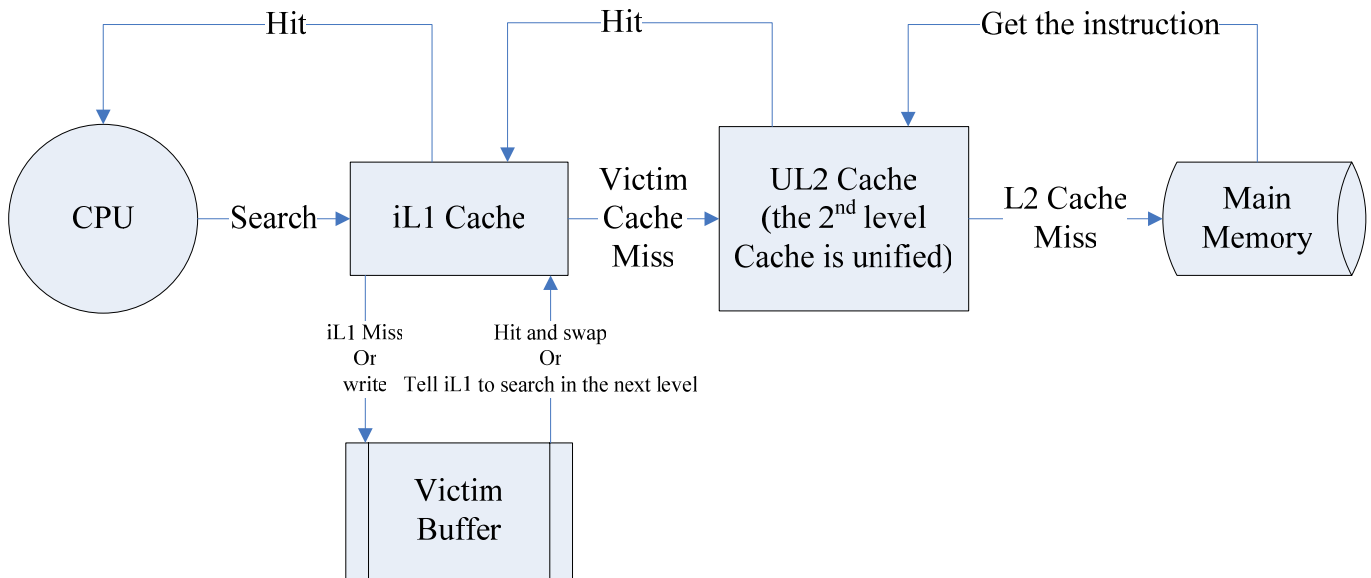


Figure 4.2 The memory hierarchy with a “swap” Victim buffer

### 4.3 “Non-swap” victim buffer (NVB)

Our second implementation of the victim buffer differs in the way that it interacts with the IL1 cache and the CPU. As we can see in Figure 4.3, the victim buffer is connected directly to the CPU. Every time we have a miss in the IL1 cache we search the victim buffer before looking into the lower level memory (UL2 and Main Memory). If we have a hit in the victim buffer we send the information directly to the CPU without first writing the data to the IL1 cache and getting the evicted block back (swap). This saves us time and we can see the difference in CPI in section 5. If we have a miss in the victim buffer then the IL1 continues looking for the block in the lower level memory and when it gets the new block it gives the evicted block to the victim buffer. There is no write-back for instruction blocks, thus the blocks that get evicted from the victim buffer just get deleted.

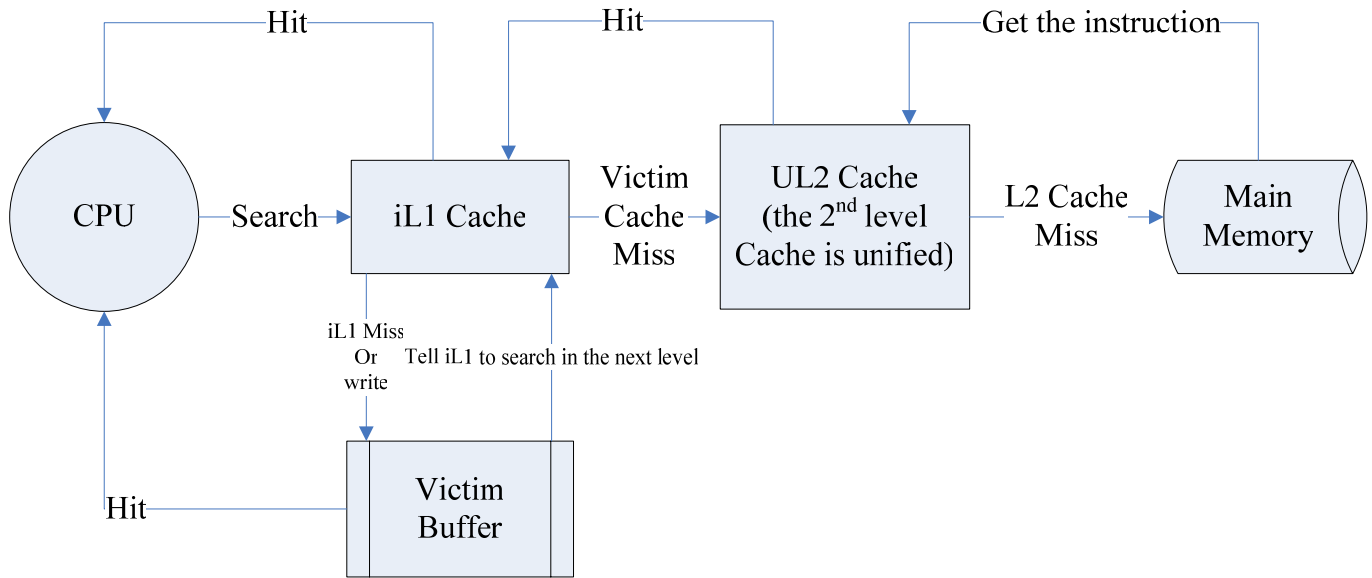


Figure 4.3 The memory hierarchy with a “non-swap” Victim buffer

The following passage focus on the discussion above and the result is compared based on the statistics from SimpleScalar.

## 5 Performance Evaluation Results

### 5.1 Simulation for SVB performance

In order to evaluate the performance of the victim buffers, we need to know how it affects, or enhance the CPU if possible. Meanwhile, we should also figure out when the victim buffers have the best performance with certain configurations.

#### 5.1.1 CPI

From Figure 5.1 we can see that the CPI reduced with the increase of the number of entries for the victim buffer. However, once the entry number reaches 64, the impact of victim cache on the CPU falls down.

In addition, we see that the SVB has a better influence on the application of *md5* and *dh*. This is probably because they have greater temporal locality in their instructions, which means the recently used instructions are more probably to be used in the near future.

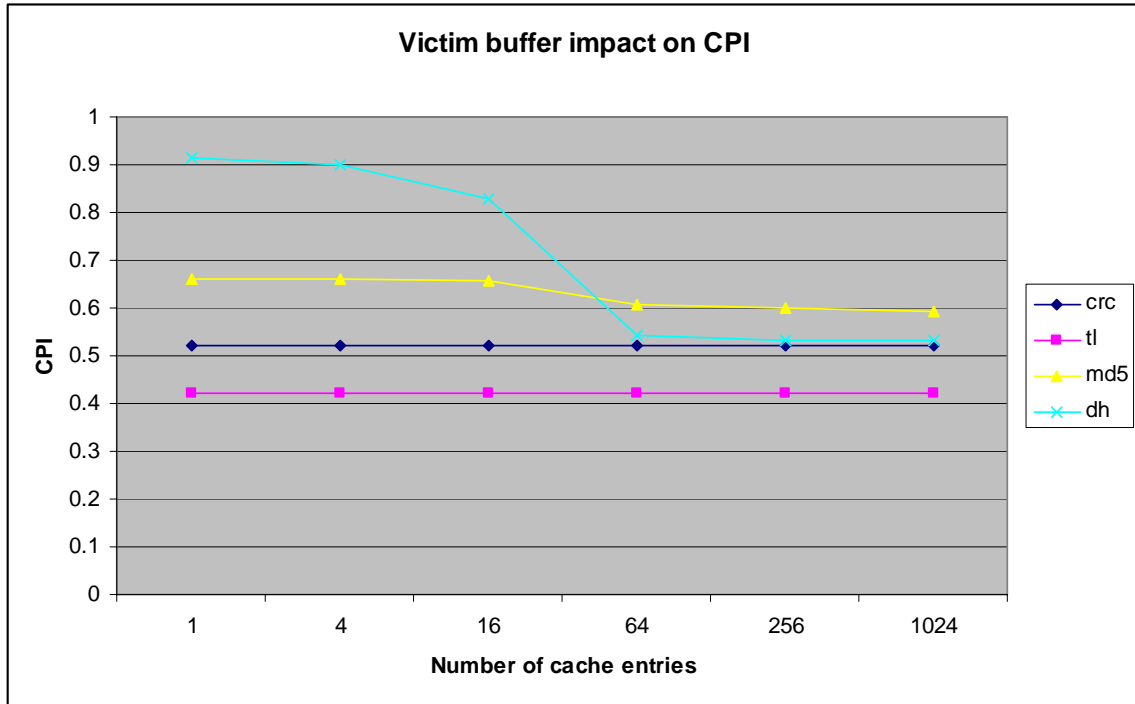


Figure 5.1

### 5.1.2 SVB miss rate

In Figure 5.2, it is clear that the miss rate drops with the increase of the size of the buffer. We know that the bigger a fully associative cache is, the more complicated and expensive the hardware is. Therefore, a fully associative victim buffer of 64-entry shows the best cost-effective result.

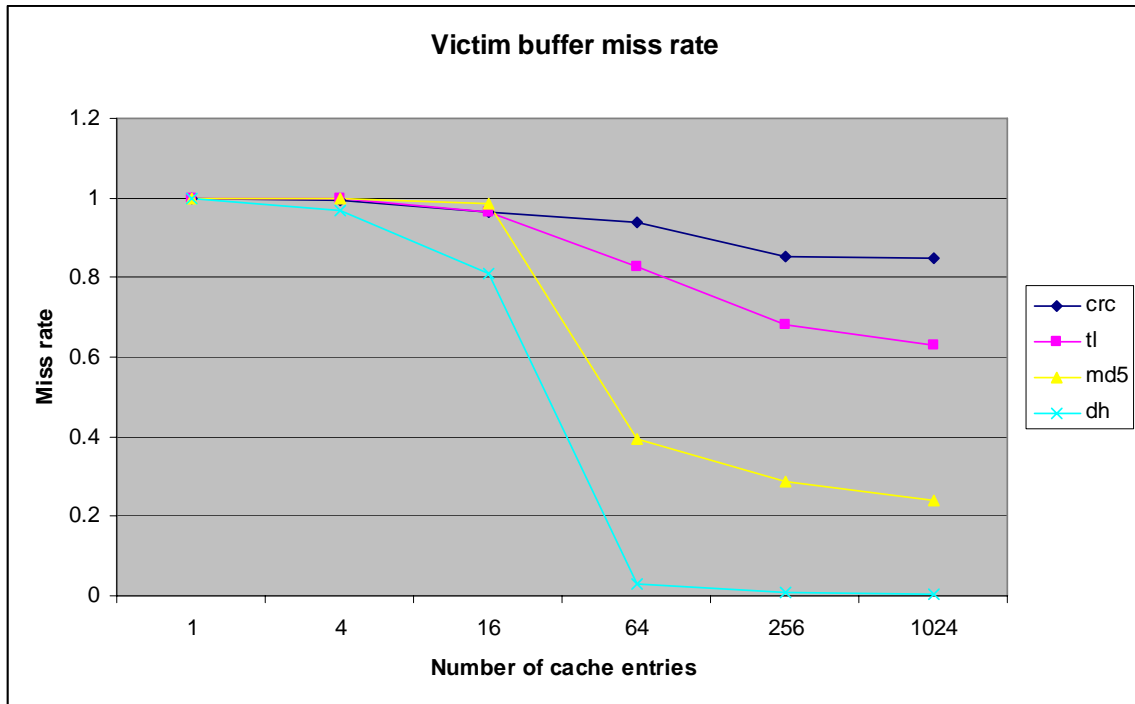


Figure 5.2

## 5.2 Simulation for NVB performance

The explanation of performance metrics for NVB is almost the same as SVB. A reference can be easily reached from previous sections.

### 5.2.1 CPI

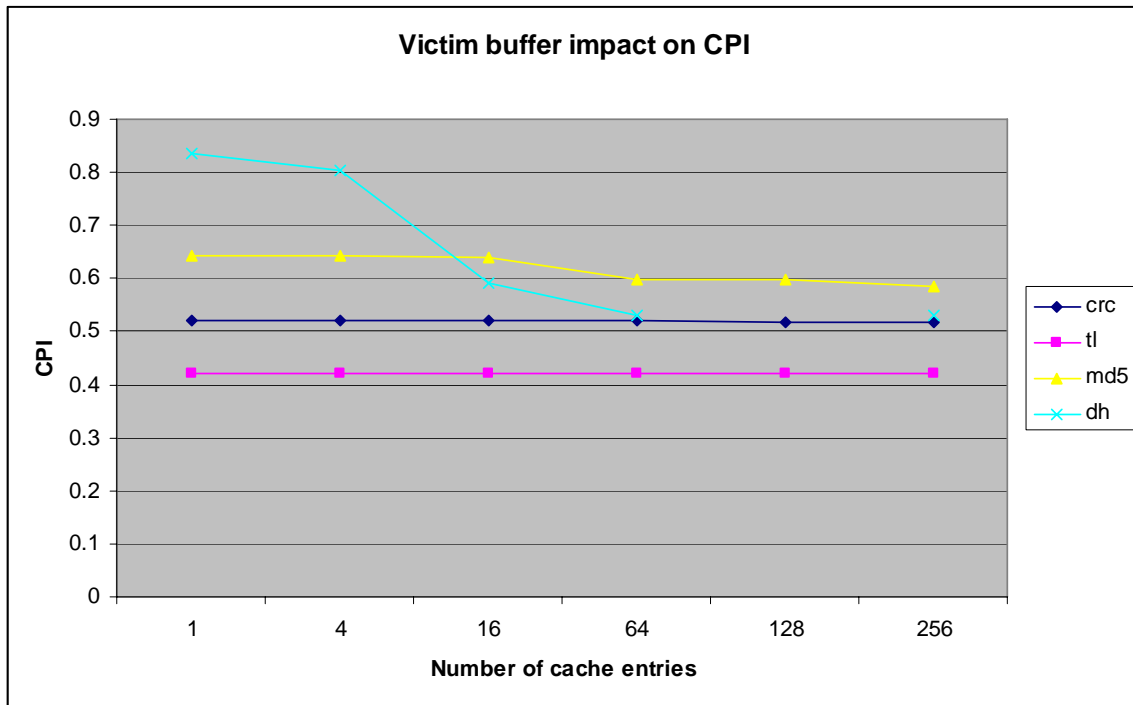


Figure 5.3

## 5.2.2 NVB miss rate

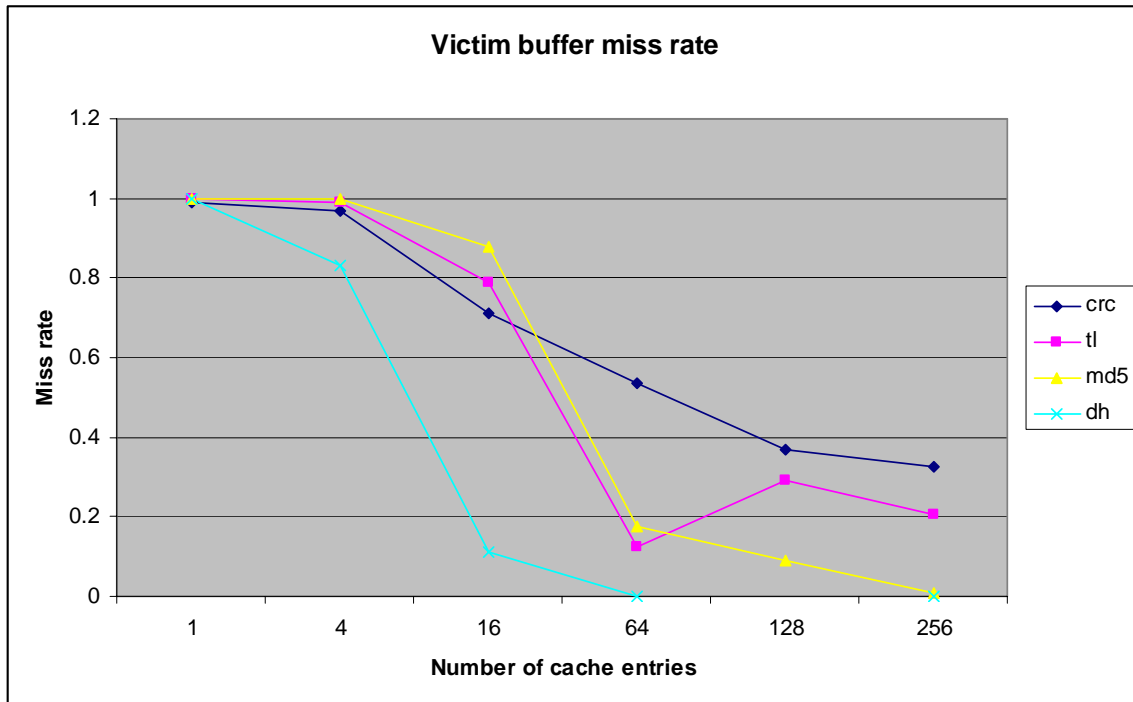


Figure 5.4

## 5.3 Comparison between with and without victim buffer

Our comparison can be divided into two steps. Firstly, we concerned with the overall performance of victim buffer (both SVB and NVB) over the original memory hierarchy. Later, we want to know which of the two victim buffer models has a better performance.

### 5.3.1 CPI

We can see that both the SVB and NVB give a better performance on CPI over the original set up, especially for the application of md5 and dh.

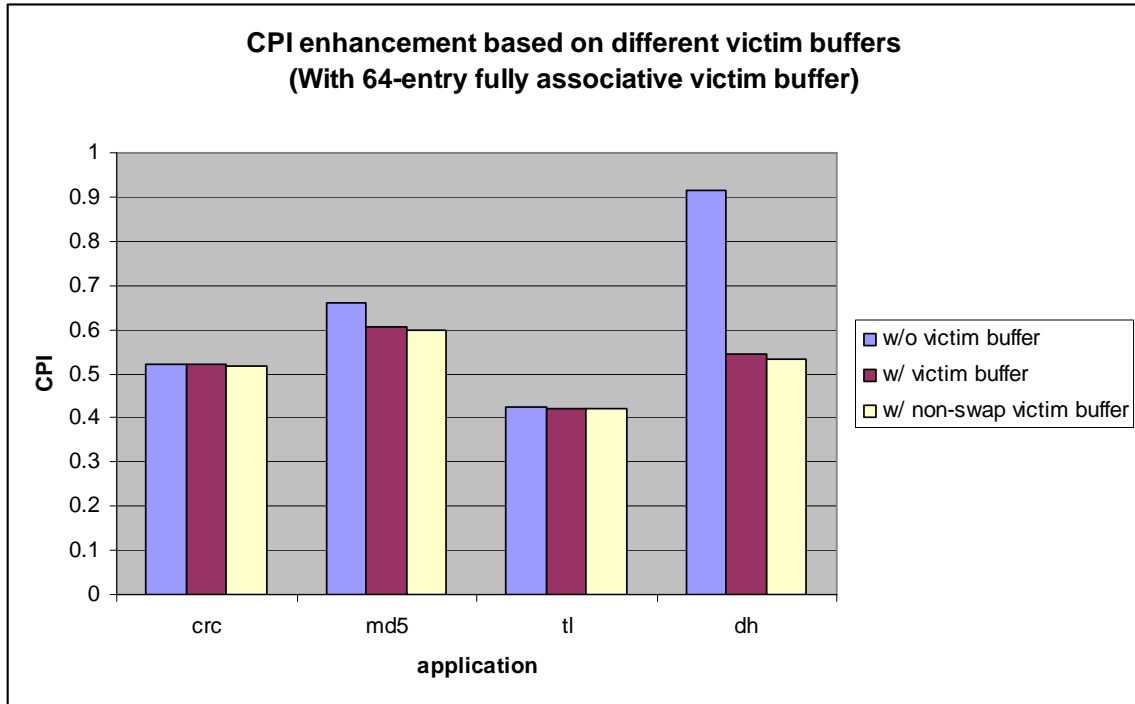


Figure 5.5

### 5.3.2 IL1 miss rate

From the miss rate of IL1, it is clearly shown that the NVB brings an increase. This takes place because the blocks between the victim buffer and IL1 never swapped, and the hit takes place in the victim buffer instead of the IL1 cache (Miss rate of victim buffer is very low. Figure 5.4).

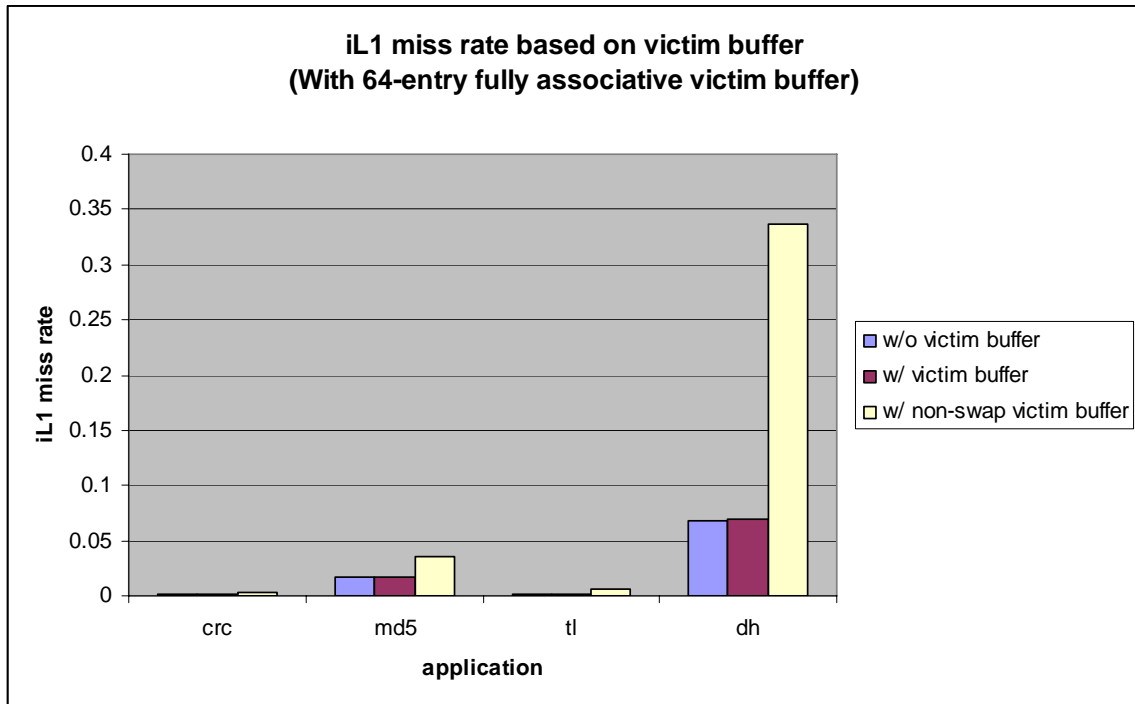


Figure 5.6

## 6 Conclusion

The performance of CPU lies in many factors. In this report, we find that it can be improved by implementing a victim buffer for IL1. It is also compared between the two different models of interaction with the IL1. The hit latency of the victim buffer should be considered more realistic according to the size of the buffer for future research.

## 7 References

- [1]. *NetBench: A Benchmarking Suite for Network Processors*  
Gokhan Memik William H. Mangione-Smith Wendong Hu
- [2]. *SimpleScalar Hacker's Guide*  
Todd Austin
- [3]. *Architectural Analysis of Cryptographic Applications for Network Processors*  
Haiyong Xie, Li Zhou, and Laxmi Bhuyan
- [4]. *A Flexible Accelerator for Layer 7 Networking Applications*  
Gokhan Memik and William H. Mangione-Smith
- [5]. *SimpleScalar Simulation Tool Set*  
Yan Luo <http://www.cs.ucr.edu/~yluo/cs162/lab1.html>
- [6]. *Performance Evaluation with Benchmarks (I)*  
Yan Luo <http://www.cs.ucr.edu/~yluo/cs162/lab2.html>
- [7]. *How to Write a Good Report*  
Yan Luo <http://www.cs.ucr.edu/~yluo/cs162/report.html>
- [8]. *NetBench: A Benchmarking Suite for Network Processor*  
Gokhan Memik
- [9]. *A Comparison of Software Code Reordering and Victim Buffers*  
Iris Bahar, Brad Calder and Dirk Grunwald