Cache DoS attacks on Shared Cache in Multi-core System

Dhruv Kaushik, MT18037
Indraprastha Institute of Information Technology, Delhi
India
dhruv18037@iiitd.ac.in

Shubham Kumar, MT18145
Indraprastha Institute of Information Technology, Delhi
India
shubham18145@iiitd.ac.in

Wrik Bhadra, MT18027
Indraprastha Institute of Information Technology, Delhi
India
wrik18027@iiitd.ac.in

I. Introduction

In Denial-of-Service (DoS) attack, attacker makes lots of requests in a short time interval to a service provider, with an aim to block its service by overloading it. In Cache DoS attack, the attacker generates such frequent read/write requests that fills up read/write register associated with shared cache, leading to its blocking. This project aims to perform and analyse Denial-of-service (DoS) attacks on shared cache in multicore systems. By executing DoS attacker task, we are able to increase execution time for victim task running on a separate dedicated core.

Attacker application can't directly affect victim application, due to core/memory isolation. So, it attacks on the non-blocking shared cache.

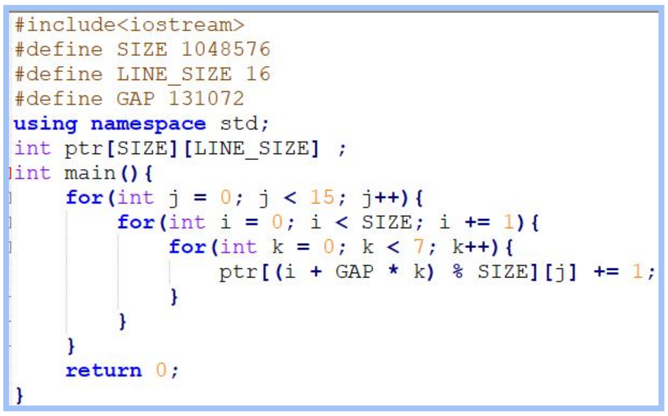
Non-blocking cache keeps servicing requests instead of blocking them despite having a miss. It keeps the miss related information in Miss Status holding registers (MSHR). Non-blocking caches allow concurrent memory accesses from multiple cores and they are often used as last-level caches in multi-core systems. It has two associated buffers: Writeback buffer and MSHR buffer. It behaves like blocking cache when any of the two buffers becomes full and deny servicing requests. It is not unblocked until internal buffers become free again. Writeback Buffer is used to store information of dirty cache lines.

In this project, we have performed Write DoS attack, which targets both the above buffers in the shared non-blocking cache at last level (level-2). This leads to high frequency of Cache blocking and thus reducing the system's performance. We simulated embedded multicore platform and executed victim tasks and attacker tasks on it with varying configurations. We are able to successfully perform Cache DoS attack on shared cache in multicore system.

II. Victim and attack programs

We used MCF program of SPEC 2006 int benchmark as victim program. MCF program is used for single-depot vehicle scheduling in public mass transportation. [2]

For performing Write DoS attack, we created the attacker code as shown in figure 1. On execution, it almost always generates a cache miss, thus generating lots of misses leading to MSHR filling and cache blocking. Also note that it is also writing value in all requests made, thus simultaneously filling Writeback buffer. Thus, it's a two way attack.



```
#include<iostream>
#define SIZE 1048576
#define LINE_SIZE 16
#define GAP 131072
using namespace std;
int ptr[SIZE][LINE_SIZE] ;
int main(){
    for(int j = 0; j < 15; j++){
        for(int i = 0; i < SIZE; i += 1){
            for(int k = 0; k < 7; k++){
                ptr[(i + GAP * k) % SIZE][j] += 1;
            }
        }
    }
    return 0;
}
```

Figure 1: Write attack code snippet

III. Methodology

1. Basic Architecture

We first created a basic Multicore architecture consisting of 4 cores and 2-level caches, among which 1st level cache is private cache and is individually associated with each processor. The 2nd level cache is the non-blocking cache, shared among all processors, on which the attacker makes DoS attack. The system architecture follows the Classic Memory System model and uses MOESI snooping protocol as cache coherence protocol.

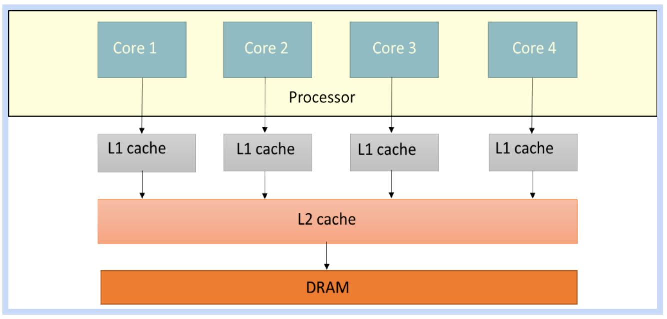


Fig. 2 : Computer Architecture proposed to be used in this project

2. *Experiments*

We experimented on various hardware configurations: with prefetcher, without prefetcher, varying shared cache size and different types of processor (in-order and out-of-order). The various system configurations are tabulated in table 1:

	Config_1	Config_2	Config_3	Config_4
CPU	TimingSimple (1 GHz)	TimingSimple (1 GHz)	TimingSimple (1 GHz)	TimingSimple (1 GHz)
L1 cache size	64 KB	64 KB	64 KB	64 KB
L2 cache size	256 KB	512 KB	256 KB	256 KB
L2 set assoc.	8 way	16 way	8 way	8 way
L2 mshrs	20	40	20	20
Writeback buf size	8	16	8	8
Prefetching	-	-	Stride prefetching	-

Table 1: Hardware configurations experimented

For, each of the configuration mentioned in Table 1, we performed 4 DoS attacks: 4 Victim cores (baseline case for that architecture), 3 victim cores and 1 attack core (single attack), 2 victim cores and 2 attack cores (double attack), 1 victim core and 3 attack cores (triple attack). As the victim program (MCF) is a heavy program w.r.t the configurations mentioned in Table 1, we run the program for 200 million instructions and noted various parameters related to L2 in generated stats file to analyse L2 performance.

We selected these few parameters as well indicators of L2 performance: l2cache.overall_miss_rate (The miss rate for all accesses), l2cache.overall_miss_latency (Total number of cycles spent waiting for all misses), l2cache.tags.occ_percent::cpu_id.data (Average percentage of L2 cache occupancy by CPU cpu_id where id varies from 0 to 3). We have plotted below the results on these parameters obtained for all the Table 1 configurations after DoS attack.

IV. Results

After performing experiments on simulated multi-core platforms, we obtained interesting results. We have plotted interesting observations as bar graphs. These graphs comprise of L2 Overall Miss Latency, L2 Overall Miss rate, L2 cache occupancy related plots are provided below:

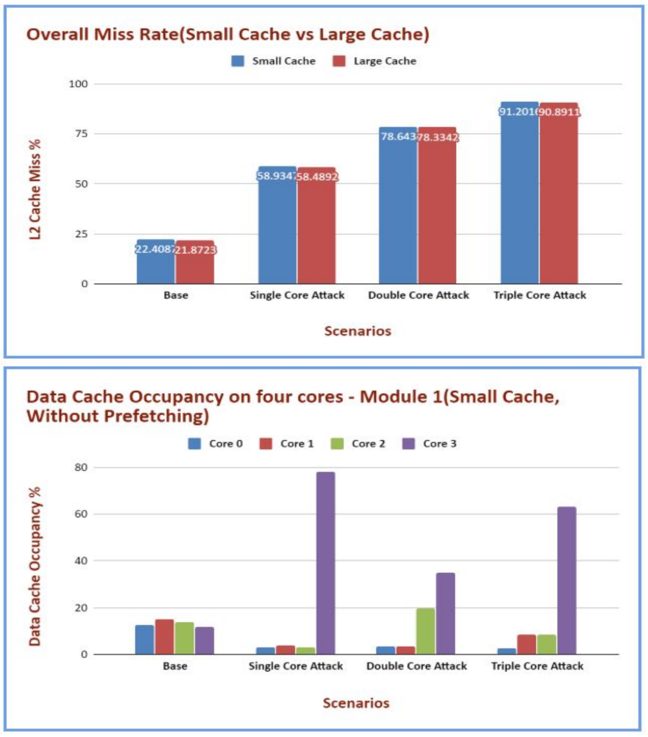


Fig3: (a) Overall Miss Rate Comparison for Config_1 vs Config_2 (b) L2 cache occupancy in Config_1

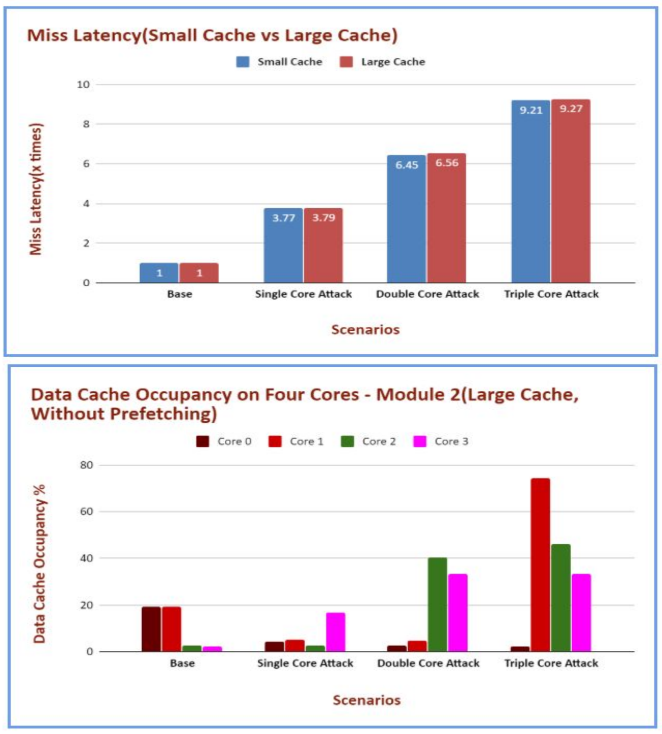


Fig4: (a) Miss Latency Comparison for Config_1 vs Config_2 (b) L2 cache occupancy in Config_2

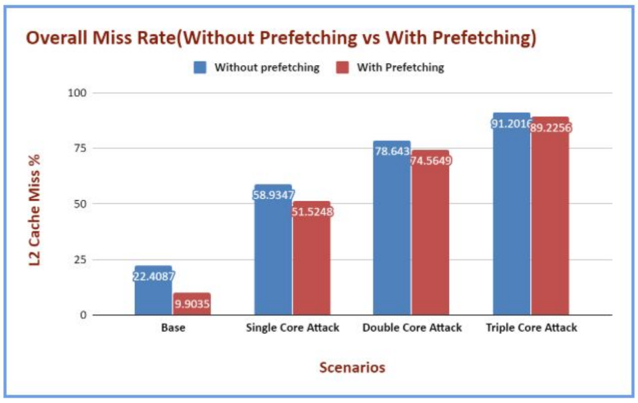


Fig 5 : Overall Miss Rate Comparison for Config_1 (without prefetching) vs Config_3 (with prefetching)

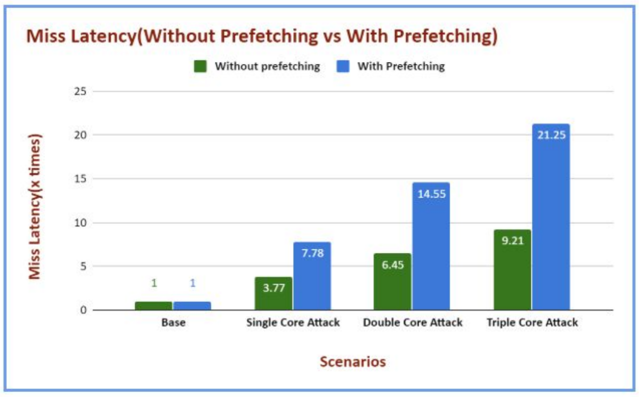


Fig 6 : Overall Miss Latency Comparison for Config_1 vs Config_3

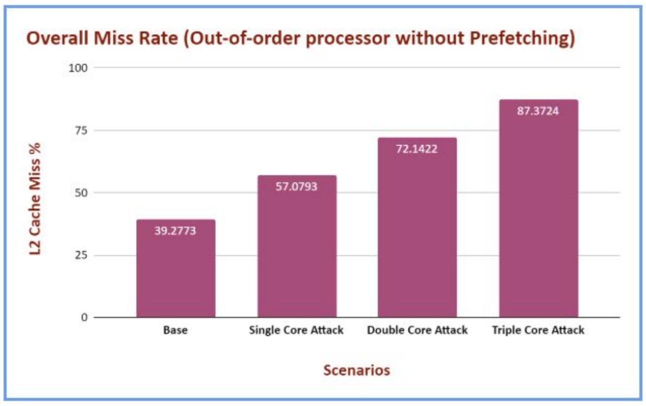


Fig 7 : Overall Miss Rate for Config_4 (Out-of-Order processor)

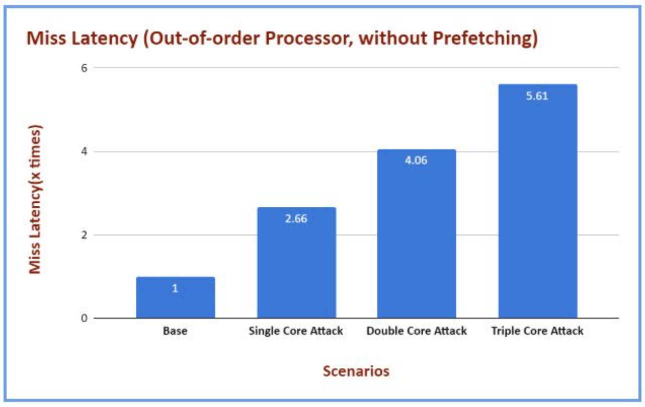


Fig 8 : Overall Miss Latency for Config_4 (Out-of-Order processor)

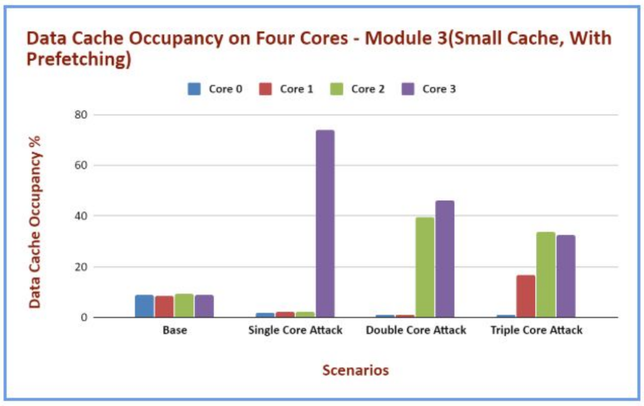


Fig 9 : L2 cache occupancy by multiple cores in Config_3 (also named as Module 3)

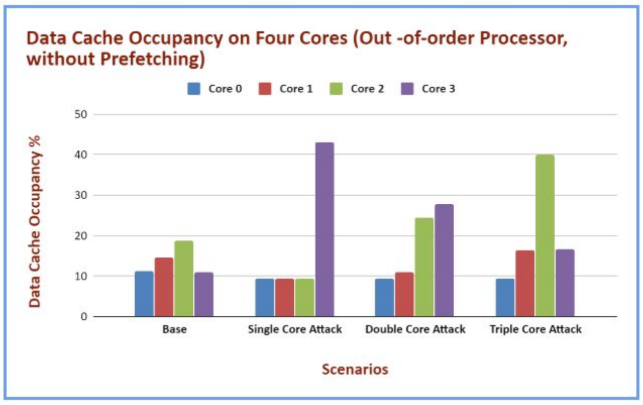


Fig 10 : L2 cache occupancy by multiple cores in Config_4

V. Conclusion

In our project, we tried to perform DoS attacks on shared caches in multicore platforms. After performing experiments, we can observe that Processors using prefetchers and Out of order processors also faced slowdown or increased L2 miss latency when attacker code was executed. The highest increase(21x) in L2 overall miss latency is observed in case of Inorder Processor, with Stride Prefetcher and small shared cache size(256 KB). Also, we observed that increasing the shared L2 cache size from small(256 KB) to large(512 KB) did not have much impact on L2 overall miss rate and L2 overall miss latency.

One interesting insight that is gained by observing DoS attack on out of order processors. The L2 cache overall miss rate and L2 overall cache miss latency does not increase as quickly as in case of inorder processors. Thus DoS attack is more successful on Inorder processors compared to complex Out of Order processors.

We also plotted graphs to analyze which core is occupying how much L2 data cache. For Base scenario(when there is no attacker code running), cache occupancy is almost uniform for all four cores and it is low for all of the cores. However, as we move to Single core attack scenario(when there is attacker program running on Core 3), Core 3 occupies major portion(greater than 70%) of L2 data cache in all cases. In case of Double core attack, second and third cores in together contribute to maximum L2 cache occupancy. When three cores are running attacker program, then these three cores in together occupy major portion of L2 Data cache. Thus, we simulated multi-core platforms with different configurations and tried performing DoS attacks on them.

VI. References

[1] Bechtel, Michael & Yun, Heechul. (2019). Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. 357-367. 10.1109/RTAS.2019.00037.

[2] https://www.spec.org/cpu2006/Docs/429.mcf.html

[3] https://www.mail-archive.com/gem5-users@gem5.org/

[4] http://gem5.org/Documentation

[5] https://en.wikipedia.org/wiki/Cache_prefetching

[6] https://www.ics.uci.edu/~amrm/slides/amrm_structure/pta/tsld049.htm

[7] https://courses.cs.washington.edu/courses/csep548/06au/lectures/cacheAdv.pdf

[8] https://en.wikipedia.org/wiki/Write_buffer

[9] http://hpca23.cse.tamu.edu/taco/utsa-www/cs5513-fall07/lecture5.html

[10] https://en.wikipedia.org/wiki/Out-of-order_execution