Computer Architecture

End-sem Evaluation

Project Members:

Wrik Bhadra (MT18027)

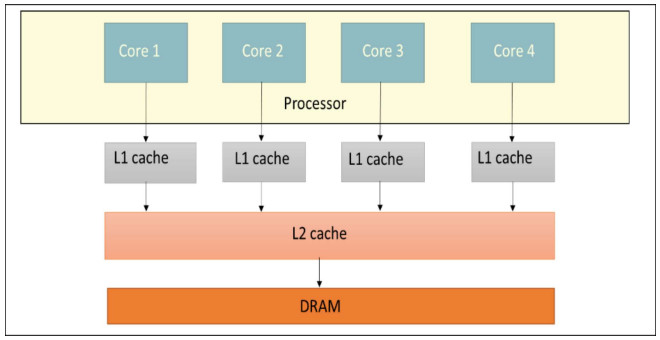
Shubham Kumar (MT18145)

Dhruv Kaushik (MT18037)

IIIT Delhi

New Delhi 110020, India

Classic Memory System



- MOESI snooping protocol as cache coherence protocol
- **Non-blocking Shared L2 cache** keeps servicing requests instead of blocking them despite having a miss.
- It has two associated buffers: Writeback buffer and MSHR buffer.
- Keeps miss related information in **Miss Status holding registers** (MSHR).

- **Cache Blocking:** Shared cache gets blocked If its any register (MSHR or Writeback) gets full. Can't service any further request from any core.
- It is not unblocked until internal buffers become free again.
- **Writeback Buffer** is used to store information of dirty cache lines.
- **Denial-of-Service (DoS) Attack :** Making lots of requests in a short time interval to a service provider, with an aim to block its service by overloading it.
- **Cache DoS attack** : A malicious application generates such frequent read/write requests that fills up read/write register associated with shared cache, leading to its blocking.
- **Attacker** (malicious) applications can't directly affect victim application, due to core/memory isolation.
- But Attacker can block shared cache to reduce victim application performance by performing Denial-of-Service (DoS) attacks on the shared cache.

Victim and Attack Programs

- **Victim Program**: MCF program of SPEC_int_2006 benchmark. MCF program is used for single-depot vehicle scheduling in public mass transportation.
- For performing **Write DoS attack**, we created the attacker code as shown in snippet:
- On execution, it almost always generates a cache miss, generating lots of misses leading to MSHR filling and cache blocking.
- It is also writing value in all requests made, thus simultaneously attacking Writeback buffer.
- Concurrent reads (read attacker) → stress MSHR.
- Concurrent writes (write attacker) → stress MSHR and WB Buffer.

```
#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;
}
```

Experiments

	Config_1	Config_2	Config_3	Config_4
CPU	TimingSimple (1 GHz)	TimingSimple (1 GHz)	TimingSimple (1 GHz)	**DerivO3CPU (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	-

Hardware configurations experimented

Experimented on various hardware configurations: with and without prefetcher, varying shared cache size and different types of processors (in-order and out-of-order).

- On each configuration above, performed 4 DoS attacks: baseline (4 Victim cores), single attack(3 victim, 1 attack core), double attack (2 victim, 2 attack cores), triple attack (1 victim, 3 attack cores).
- Noted various parameters related to L2 in generated stats file to analyze 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

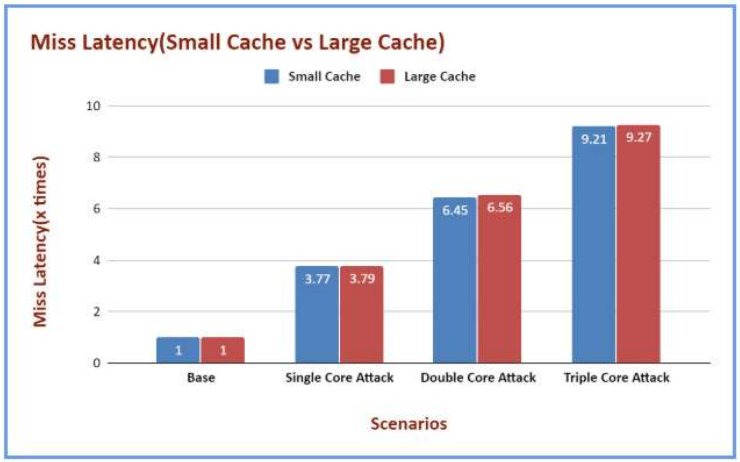


Fig: Overall Miss Rate Comparison for Config_1 vs Confiig_2

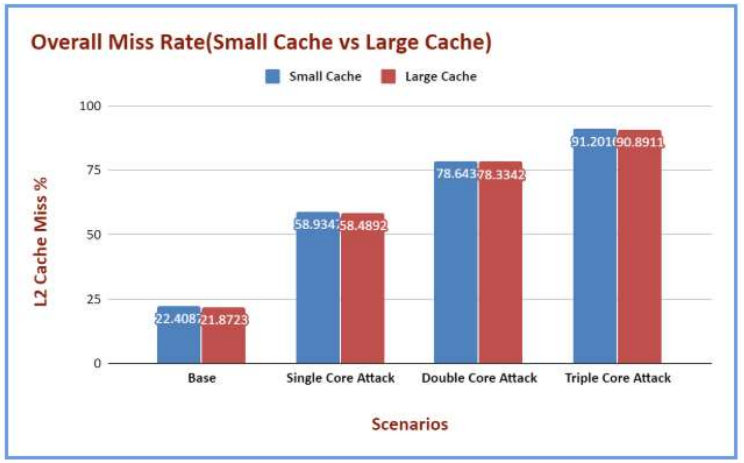


Fig: Miss Latency Comparison for Config_1 vs Config_2

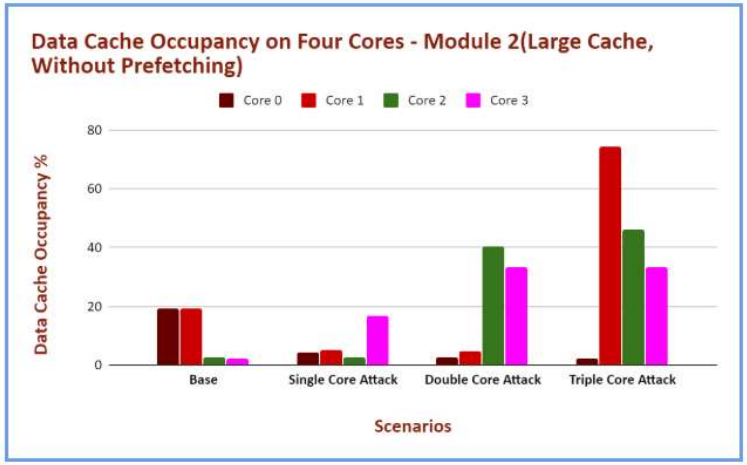


Fig: Core-wise L2 cache occupancy in Config_1

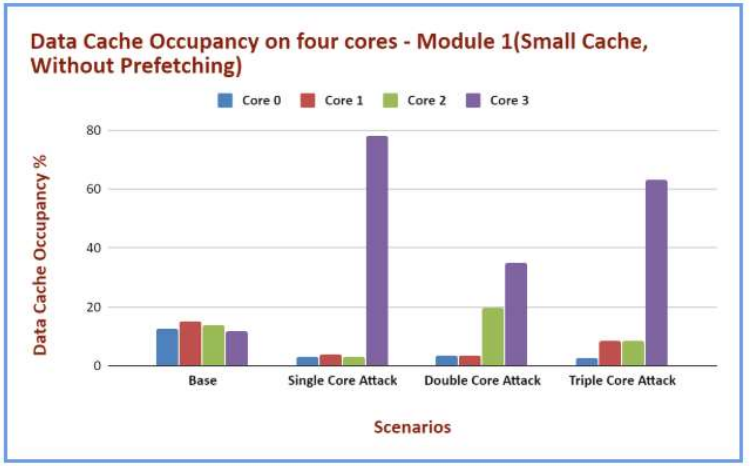


Fig: Core-wise L2 cache occupancy in Config_2

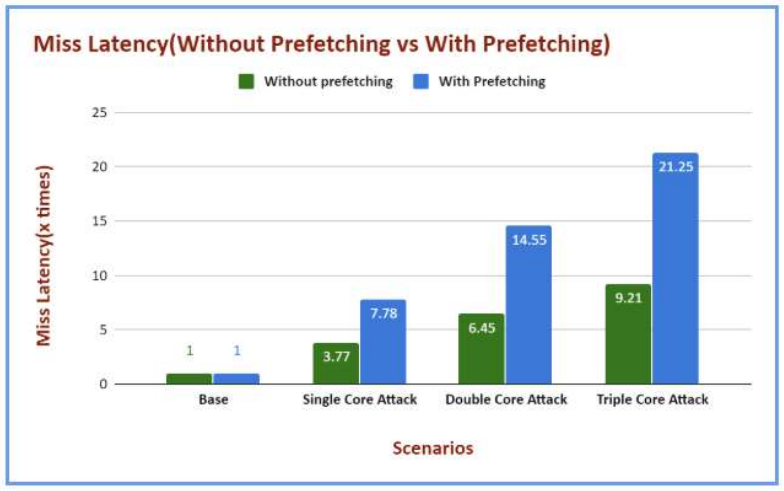


Fig : Overall Miss Latency Comparison for Config_1 vs Config_3

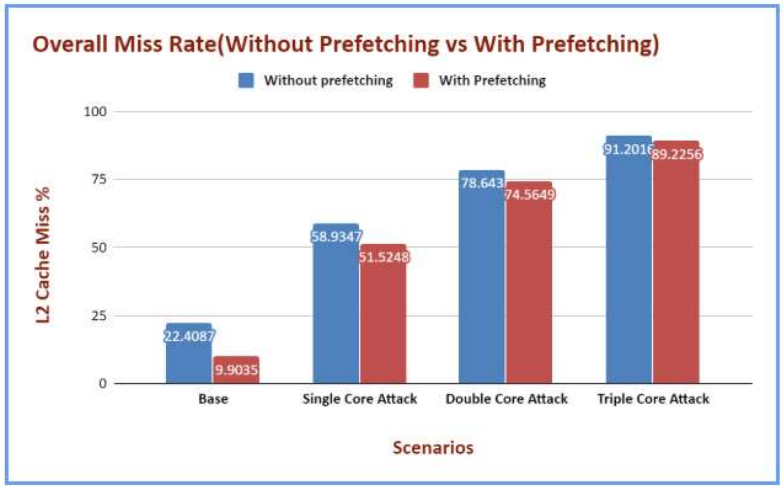


Fig : Overall Miss Rate Comparison for Config_1 (without prefetching) vs Config_3 (with prefetching)

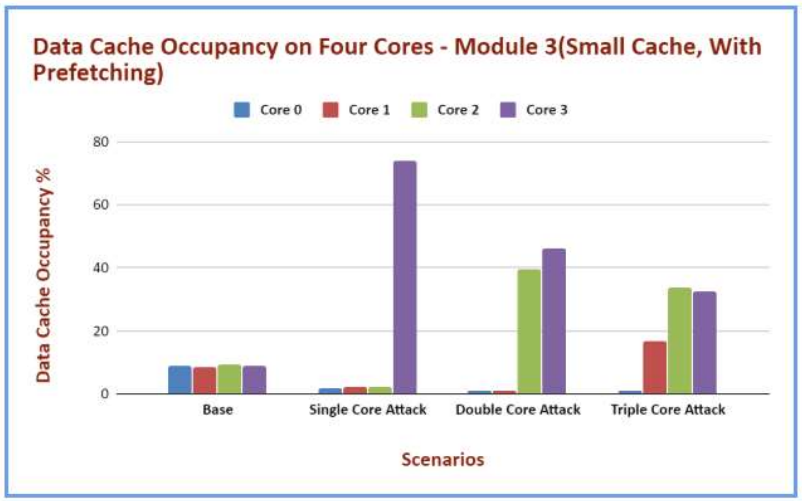


Fig : L2 cache occupancy by multiple cores in Config_3 (also named as Module 3)

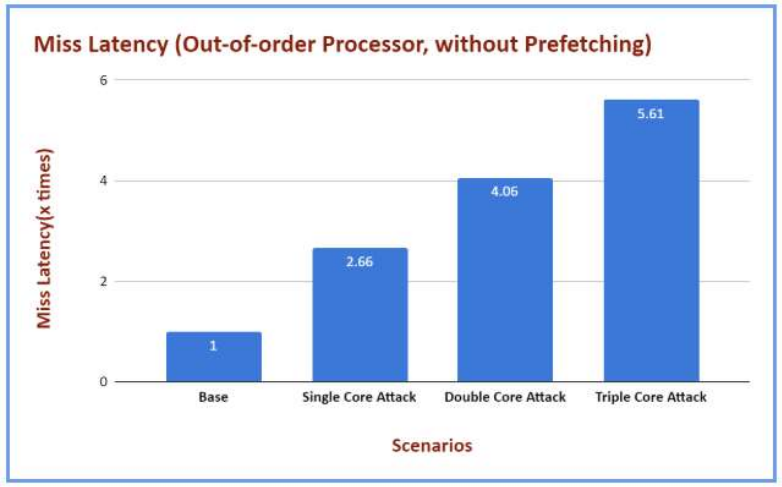


Fig : Overall Miss Latency for Config_4 (Out-of-Order processor)

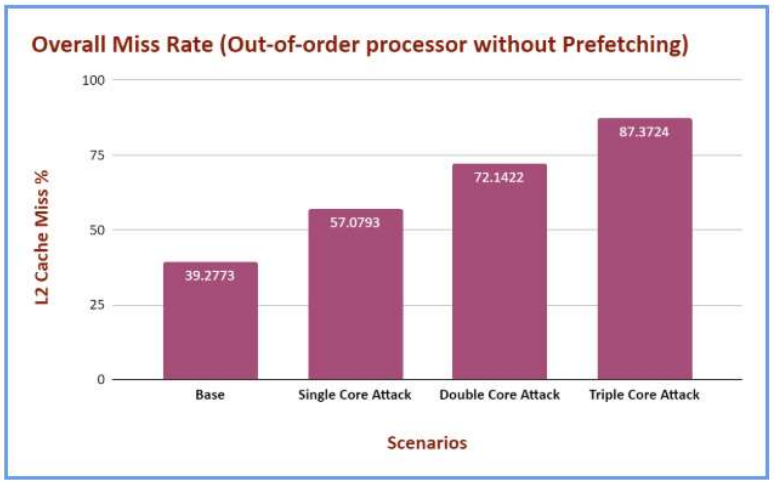


Fig : Overall Miss Rate for Config_4 (Out-of-Order processor)

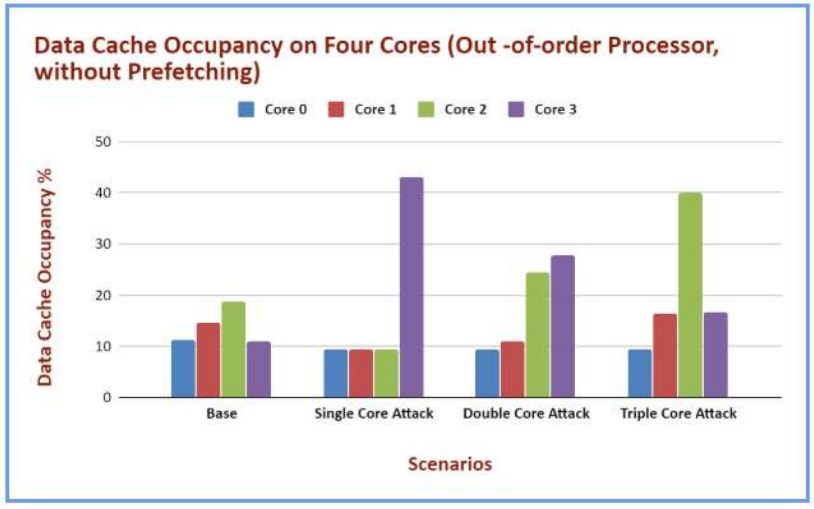


Fig : L2 cache occupancy by multiple cores in Config_4

Conclusions

- 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 In-order Processor, with Stride Prefetcher and small shared cache size(256 KB).
- 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
- In out-of-order processor, we found that the L2 cache overall miss rate and L2 overall cache miss latency does not increase as quickly as in case of in-order.
- In Base scenario, cache occupancy is almost uniform for all four cores.
- Though in single core attack only, we found that attacking core occupies more than 75% of the L2 cache. This occupation of L2 increases as the number of attacks increases from single to double to triple, thus giving less and less room to victim core in L2 to function.

References

1. Michael Bechtel and Heechul Yun , “Denial of Service Attacks on Shared Cache in Multicore: Analysis and Prevention”, 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)

2. Gem5 tutorial : http://gem5.org/Main_Page

3. Gem5 tutorial : http://learning.gem5.org/book/

4. N. Binkert et al., “The gem5 simulator,” ACM SIGARCH Computer Architecture News, vol. 39, no. 2, pp. 1–7, 2011.

5. Y. Kim, W. Yang, and O. Mutlu, “Ramulator: A fast and extensible DRAM Simulator”. IEEE Computer Architecture Letters, 2016

6. J. Hennessy and D. Patterson, “Computer Architecture: A Quantitative Approach”, 2011

Thank You