Creating and analyzing cache Denial of Service (DoS) attacks in a multicore system

P Akshay Kumar(MT19094)
CSE
IIITD
Delhi, India
akshay19094@iiitd.ac.in

Abhsishek Singh Chauhan(MT19085)
CSE
IIITD
Delhi, India
abhishek19085@iiitd.ac.in

Abhishek Pundir(MT19084)
CSE
IIITD
Delhi, India
abhishek19084@iiitd.ac.in

Vishal Sharma(MT19101)
CSE
IIITD
Delhi, India
vishal19101@iiitd.ac.in

Anannya Chottopadhaya(MT20139)
CSE
IIITD
Delhi, India
anannya20139@iiitd.ac.in

***Abstract*— In this paper, we mainly investigated the impact of denial-of-service (Dos) attack on shared caches on various multicore platforms. We tried analyzing what are the possible reasons for such attacks. How this is impacting the performance of the concerned system. It has been observed that with a more robust attacker script, we can simulate the attacks which produces result closer to what is described in the paper.**

Keywords—denial-of-service (DoS) attack, multicore platforms, MSHRs, write back buffers, prefetchers, SD-VBS

I. INTRODUCTION

Multi-core architectures are considered to be the turning point of modern computing system. Efficient multi-core computing platforms can be considered as the need of an hour for any embedded real-time system. Each of the tasks might have a strict timing requirement, which if get effected might hamper the entire performance of the entire system. The schedulability of a multi-core platform can be computed by taking into account the worst-case execution time for the individual tasks running simultaneously. Shared hardware resources such as shared caches, main memory contributes to unpredictable timing behavior.[9] This problem can further worsen in the presence of potential attackers.

If we consider the case of a modern car, which mainly comprises of distributive computer system where one can witness reach connectivity provided by the internal networks. Here, there can be a possibility of a potential attacker connecting to the car's internal networks and creating an obstacle in the smooth functioning of all the computer controlling system such as breaks, engines, power constraints. This can happen via downloaded 3rd party applications.[9] Even if we try to partition the OS and the cores to isolate those attacker programs, but the system being a multicore computing platform the attacker can still influence the timing behavior to a certain extent. The attacker mainly targets the shared hardware resources. In this paper, we will be focusing on how the attacker exploits the shared caches depicting the denial-of-service (DoS) attack.

Multicore processors use non-blocking caches for the purpose of employing memory-level parallelism (MLP). These are mainly used as the shared last level caches (LLC). In case of both the in-order and out of order processors multiple memory requests can be generated. But any further access to the non-blocking caches is denied once the internal hardware buffers are full. This can overall impact each of the cores. Now the caches can be accessed only when the internal buffers are available again. Once the caches are blocked each of the cores now will be required to wait for longer duration as accessing the main memory will slow down the process. If hypothetically, we consider an attacker's script where such shared cache blocking can be induced this could significantly hamper the timing behavior of the tasks under different cores.

In this paper, we took 3 multi-core processors into account out of which two are in-order (Minor CPU and HPI CPU) and one is out-of-order (DerivO3) and experimentally tried analyzing the feasibility of DoS attack on shared caches in each of these multicore platforms. We have done all our simulations for ARM ISA on Gem5. In the prior studies conducted so far, it was observed that such cache blocking is not much prominent in the in-order processors compared to that of out-of-order processors but through our experiments we could find significant impact of cache blocking in the in-order processors as well. Other prior studies in field experimentally stated that how cache partitioning is still ineffective in dealing with the problem of shared cache blocking in the multicore platforms.[9]

The following are the major contributions of our paper:

- We tried demonstrating the impact of DoS attacks with the help of experiments in both the in-order as well as out-of-order multicore platforms. We tried making the attacker script more robust and, in the process, analyzed the timing behavior.
- We experimentally tried analyzing the effect of writeback buffer sizes, L2 cache sizes and MSHRs on various CPUs that we took into account.

The research paper is mainly organized into four more sections. In section II we will give a brief overview of the concepts that are necessary for undergoing this research work. The section III will comprise the methodology part followed by conclusion in section IV.

II. BACKGROUND

A. Non- blocking cache

Non-blocking caches employees MLP which is useful in multi-core processors. MLP is well implemented in case of non-blocking caches as even after multiple number of cache misses are still under process, it can still be accessed.

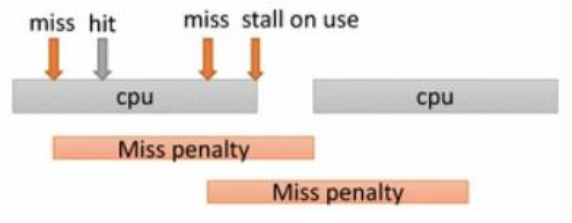

Figure 1: Non-blocking caches. [7]

A non-blocking caches consists of two main hardware structures.

- Write back (WB) buffers
- MSHRs

MSHRs deal with each of the cache misses that the non-blocking cache encounters. Each of the misses are registered in the MSHRs. Once the corresponding memory requests are served from the DRAM or any other lower-level memory hierarchy, the MSHRs are cleared. In the process, the cache continues to serve further requests from multiple cores at the same time. However, it needs to be taken into account that the degree of MLP that can be supported by a non-blocking cache is also dependent on number of MSHRs.[3] Further cache hit requests can get denied once all the MSHRs are already used up.

The WB buffers in the non-blocking cache mainly stores the evicted cache lines on temporary basis so that they can be written back to the main memory. WB buffers are efficient in reducing the number of stalls or pipeline contention and thus directly impacts the performance in a positive way.[3]

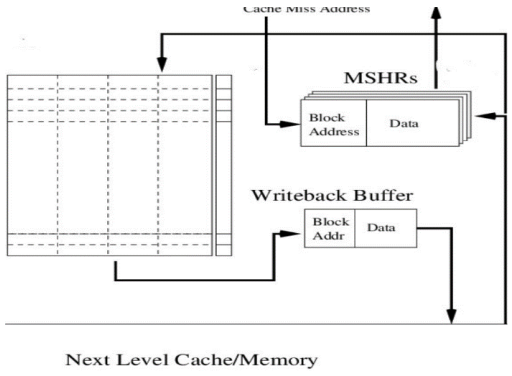

Figure 2: Block diagram of non-blocking caches.[2]

It needs to be taken into account that, if either of WB buffers or MSHRs gets exhausted, the shared caches will get blocked. This might hamper the task of each of the cores of the multicore platform.

B. *Hardware prefetchers*

Here the term prefetching depicts fetching memory blocks in advance which are most likely to be used in the near future. This can reduce the cache miss penalty to a significant extent thus showing a positive impact in the performance. But it needs to be kept in mind that only successful prefetching can lead to improvement in performance. There might be chances of incorrect prefetching which might lead to unnecessary cache blocking.[5] The hardware prefetchers mainly follows the cache memory access pattern and based on the pattern predicts the future memory access.

III. METHODOLOGY

We assumed a dual core processor where the victim as well as the attacker are co-located. It is also assumed that there is isolation between the core and the memory during the runtime. The attacker script will be run on the attacker core whereas the victim's script will run on the separate core. The task of the attacker is to perform or run unwanted programs in the attacker's core.[8] In this paper, we mainly analyzed how an attacker's script could be instrumental in influencing the timing behavior of the programs running on the victim's core.

The 4-core system architecture designed by is as follows:

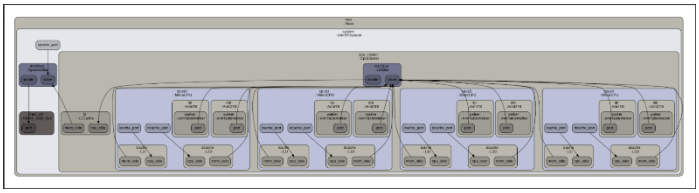

Figure 3: 4-core system architecture

A. *Experimentally demonstrating cache DoS attack on shared caches*

We wrote two attacker script namely- Bwread and Bwwrite. The main purpose of the attacker here is to create multiple number of cache misses thus exhausting the MSHRs as well as the WB buffers. Thus, inducing shared cache blocking.
The following is a snapshot of the final Bwread script:

```
#include<stdio.h>
#include<vector>
#define mem_size 8*1024*1024
#define ll long long
using namespace std;
int main()
{
      //char read_arr[mem_size];
      vector<char> c(mem_size,'A');
      ll x=65;
      ll b=0;
      for(ll i=0;i<mem_size;i+=64LL)
      {
            b+=((c[i]%26)+x)%100;
      }
      return 0;
}
```

Figure 4: BwRead attack

The Bwread attack code will keep on iterating the read_arr. Once the array size exceeds the cache size this will lead to cache misses thus loading the MSHRs. So, it can be said that the read attack will mainly target the MSHRs for inducing cache blocking.
The following is a snippet of the final Bwwrite script:

```
#include<stdio.h>
#include<vector>
#define mem_size 8*1024*1024
#define ll long long
using namespace std;
int main()
{
      //char read_arr[mem_size];
      vector<char> c(mem_size,'A');
      ll x=65;
      for(ll i=0;i<mem_size;i+=64LL)
      {
            c[i]=(i%26)+x;
      }
      return 0;
}
```

Figure 5: Bwwrite script (write attack)

The write attack will mainly target the MSHRs and WB buffers. Here the same iteration over the read_arr[] is performed but instead of the memory load operation it performs the memory store operations thus exhausting the cache both the MSHRs as well as the WB buffers. As once the cache miss is encountered this time it will induce both write back of the dirty lines as well cache fills.

B. Multicore platforms

In order to carry out the analysis of how DoS attacks are impacting the efficiency of various multicore platforms, we explored the various architectures Cortex A53, Cortex A7 on different multicore platforms such as Raspberry Pi 2, Raspberry Pi 3, Odroid C2. We have considered 2 in-order designs and one out-of-order design.

Platform	HPI	Minor	DerivO3
CPU	4x Cortex-A53 In-order	4x Cortex-A7 In-order	4x Cortex-A15 Out-of-order

Table 1: CPU Types

We experimentally tried analyzing the timing behavior in the presence of attacker's scripts in each of these systems. At first, we simply run the victim task without an attacker. However, we kept on increasing the number of attackers on the corresponding cores. In case of minor CPU as well as the HPI CPU, it has been observed that as we kept on increasing the number of attackers (Bwread) the process started slowing down slightly. Almost the same trend is observed when we changed the attacker script to write attack (Bwwrite). This observation tends to be in line with the prior studies that in-order processors are not much prone to shared cache blocking due to exhaustion of number of MSHRs as at a time there will be only one memory access in case of in-order cores.

When the overall analysis was considered we observed that the out-of-order processor gets more impacted by the attacker compared to that of the in-order. The possible reason can be out-of-order core can more easily lead to MSHR contention due to number of concurrent cache accesses. When each these encounters caches misses, the MSHRs are more likely to get exhausted but since the generation of concurrent cache access is limited in case of in-order cores.[6] This effect is not much evident in in-order cores.

When we tried analyzing the impact on the overall miss rate, we found on increasing the attackers task miss rate showed a huge rise compared to when the victim's task is run without attacker. The following are snippets from our observation how the execution time and overall miss rate gets impacted in case of in-order and out-of-order cores in ARM ISA. The baseline configuration used for impact analysis is as follows:

CPU Type	4x Cortex-A53 (HPI)	4x Cortex-A7 (Minor)	4x Cortex-A15 (DerivO3)
Private Cache Shared Cache	32/32KB 512KB (16-way)	32/32KB 512KB (16-way)	32/32KB 2MB (16-way)
Memory	1GB DDR3	2GB DDR3	2GB DDR3

Table 2: Baseline configuration for impact analysis

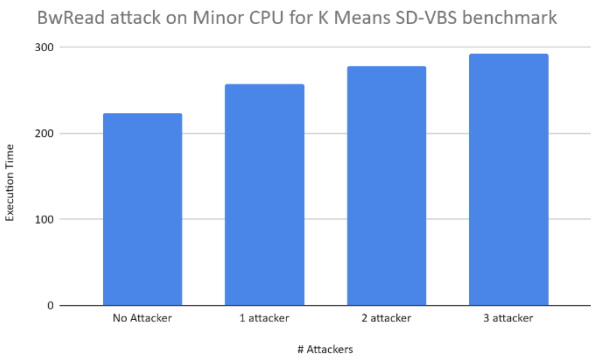

Figure 6: In the presence of Bwread attack on minor CPU

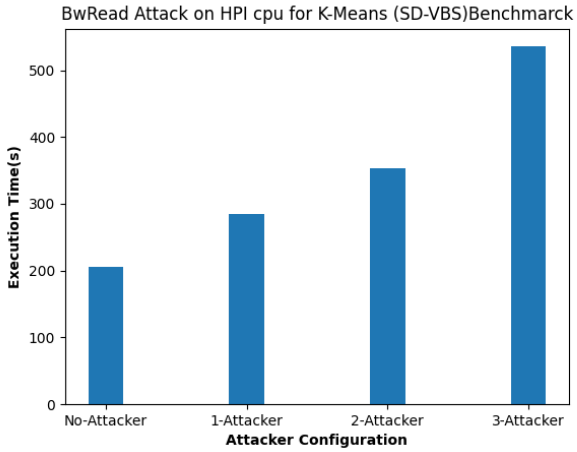

Figure 7: In the presence of Bwread attack on HPI CPU

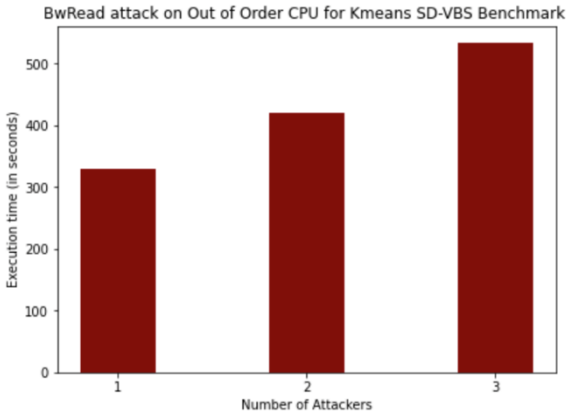

Figure 8: In the presence of Bwread attack on DerivO3 CPU

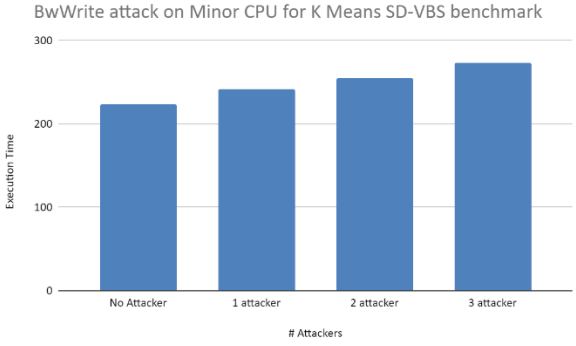

Figure 9: In the presence of BwWrite attack on minor CPU

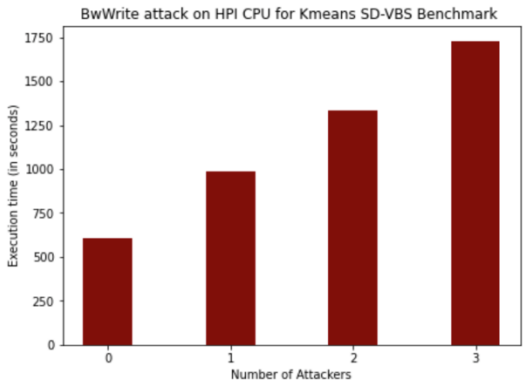

Figure 10: In the presence of BwWrite attack on HPI CPU

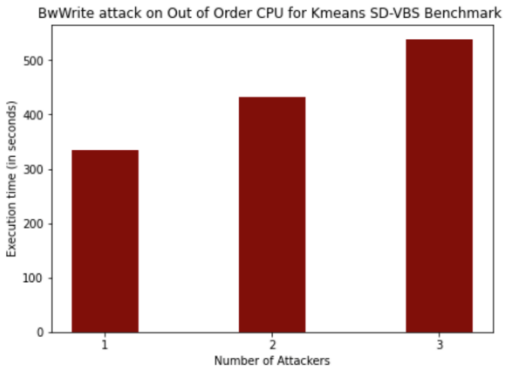

Figure 11: In the presence of BwWrite attack on DerivO3 CPU

C. *Impact of L2 cache size*

For the purpose of analyzing the performance of each of the processors (2 types of in-order and one out-of-order) more broadly we tried observing the trend of how the execution is slowing down when the L2 cache sizes is varied. We kept on varying the L2 cache size first in the presence of only Bwread attack, then in the presence of only Bwwrite attack and then in the presence of both the scripts (Bwread and Bwwrite) The following are a snippets for minor CPU (Odroid C2):

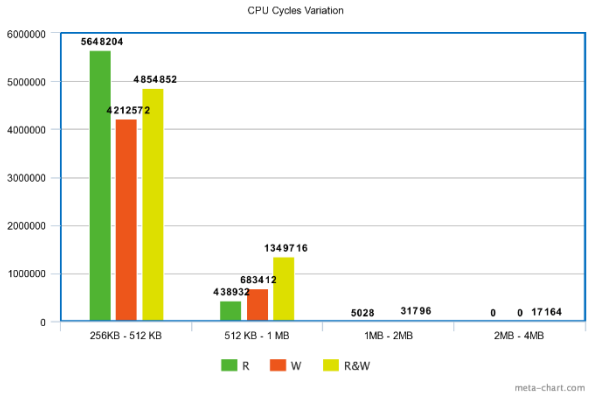

Figure 12: Impact of L2 Cache Size (variation) on CPU Cycles (Minor CPU)

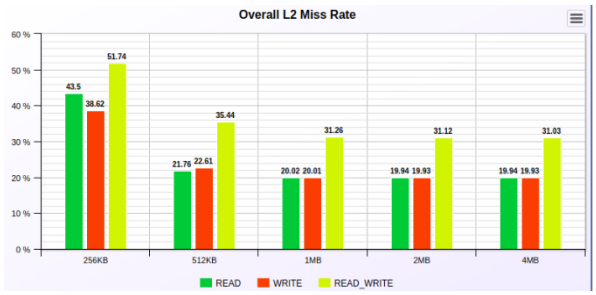

Figure 13: Impact of L2 Cache Size on L2 Miss Rate (Minor CPU)

The following are a snippet for Out-of-order processor(DerivO3)

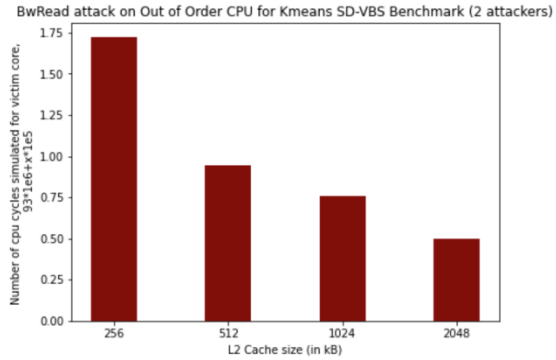

Figure 14: Impact of L2 Cache Size on BwRead attack CPU Cycles (DeriveO3 CPU)

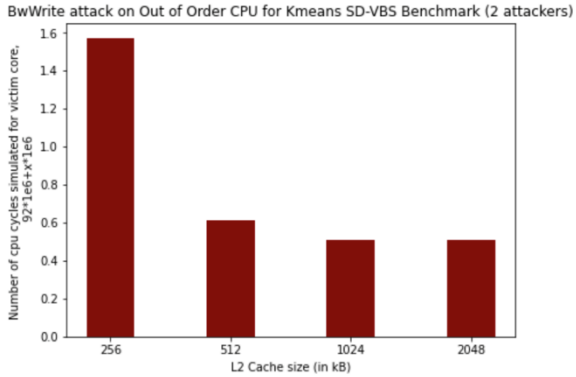

Figure 15: Impact of L2 Cache Size on BwRead attack CPU Cycles (DeriveO3 CPU)

D. *Impact of MSHRs*

We did similar observations by varying the number of MSHRs and keeping the other parameters to be constant. We kept on varying the number of MSHRs first in the presence of only Bwread attack, then in the presence of only Bwwrite attack and then in the presence of both Bwread and Bwwrite attack.

The following are the snippets for DerivO3 CPU :

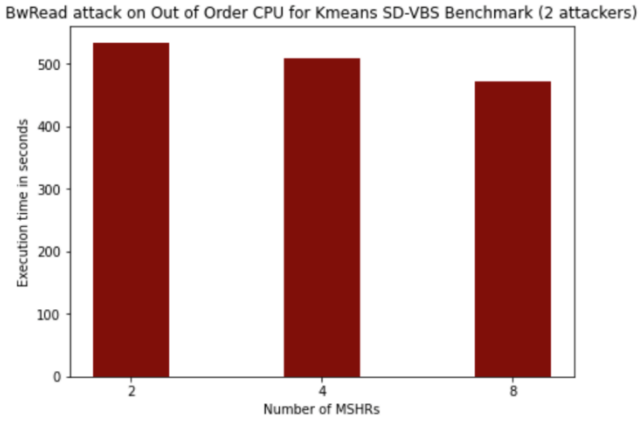

Figure 16: Impact of MSHRs on DerivO3 CPU

The baseline configuration for impact analysis of hardware prefetchers and writeback buffers is as follows:

Core	Quad-core, 1.5 GHz, IQ: 96, ROB: 128, LSQ: 48/48
L1-I/D caches	Private 32 kB (2-way), Private 32 kB (4-way), MSHRs: 1 (I), 3 (D), Writeback Buffer: 1 (I), 3(D)
L1-D PF	Stride, Degree: 5, Queue size: 5
L2 cache	Shared 512 kB (16-way), MSHRs: 24, Writeback Buffer: 8, hit latency: 12, LRU
L2 PF	Stride, Degree: 8, Queue size: 8
DRAM controller	Read/write buffers: 64, open-adaptive page policy
DRAM module	DDR3@800MHz, 1 rank, 8 banks

TABLE II: Baseline simulation parameters for Gem5 and Ramulator.

Table 3: Baseline configuration for impact analysis of hardware prefetchers and writeback buffers.

E. Impact of hardware prefetchers

We experimentally observed that when hardware prefetchers comes into play, the execution further slows down. The possible reason could be due to the blocking of L2 cache, which can again occur due to the increase in the number of caches writebacks. The number of caches writebacks increases due to the additional prefetch requests.[8] We performed the experiment for the prefetcher on L1D and prefetcher on L2 separately.

The following is the plot on how the hardware prefetchers can impact the timing behavior for minor CPU (Odroid C2) for Disparity program:

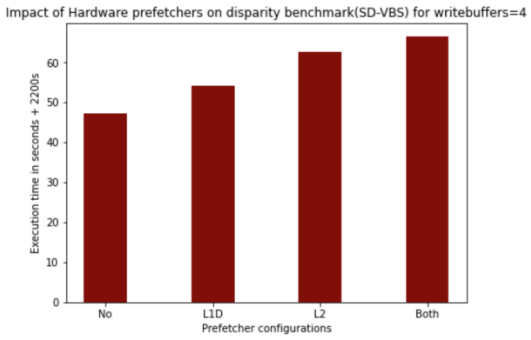

Figure 17: Impact of hardware prefetchers on execution time (Minor CPU)

F. Impact of writeback buffers

We further carried out experiments to analyze how the writeback buffer size could possibly impact the performance in the presence of DoS attack. We experimentally observed that the as on increasing the L2 writeback buffer size, the execution time improves. The possible reason could be, now the L2 cache will blocked for significantly lesser duration.[1] The following is the plot on how the writeback buffers can impact the timing behavior for minor CPU (Odroid C2) under Disparity program.

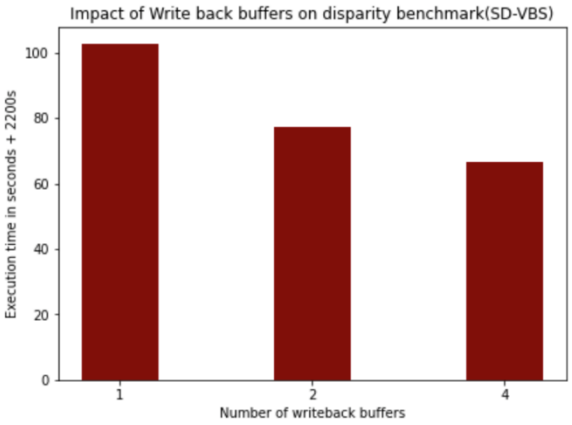

Figure 18: Impact of writeback buffers on execution time (Minor CPU)

IV. CONCLUSION

In this paper, we experimentally analyzed how severely is the DoS attack is prominent in the shared cache for various multicore platforms. We experimented in in-order as well as out-of-order processors on real world benchmarks. Other than that, we studied the effectiveness of DoS attack in the presence of hardware prefetchers and writeback buffers.

REFERENCES

[1] J. P. Shen and M. H. Lipasti. *Modern Processor Design: Fundamentals of Superscalar Processors*. Waveland Press, 2013.

[2] G. E. Suh, S. Devadas, and L. Rudolph. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *High-Performance Computer Architecture (HPCA)*. IEEE, 2002.

[3] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2011.

[4] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor. Sd-vbs: The san diego vision benchmark suite. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 55–64. IEEE, 2009.

[5] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni. Real-Time Cache Management Framework for Multi-core Architectures. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2013.

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011

[7] R. E. Kessler and M. D. Hill. Page placement algorithms for large real-indexed caches. *ACM Transactions on Computer Systems (TOCS)*, 10(4):338–359, 1992.

[8] M. Bechtel and H. Yun, "Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention," 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Montreal, QC, Canada 2019, pp. 357-367, doi: 10.1109/RTAS.2019.00037.

[9] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu. Mise: Providing performance predictability and improving fairness in shared main memory systems. In *High Performance Computer Architecture (HPCA)*, pages 639–650. IEEE, 2013.

[10] P. Greenhalgh. big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7. *ARM White paper*, 2011.

APPENDIX:

THINGS THAT DIDN'T WORK

Following are the list of observations and analysis that carried out while the duration of the project but didn't get the expected trends as per the prior studies:

- Impact of Bwread and Bwwrite attack on HPI CPU
- Impact of write back buffers on write attacks on HPI CPU
- Impact of hardware prefetchers on HPI CPU
- Impact of DoS attack on performance on varying cache sizes for HPI CPU
- Impact of DoS attack on performance on varying MSHRs for HPI CPU
- We performed the same analysis for HPI CPU under SB-VBS benchmark (Kmeans, PCA, disparity)
- The analysis for minor CPU didn't give expected result for PCA
- We tried implementing the OS-based solution as well but couldn't reach a conclusion.

This represents the work that we have done but since it was not following the required trend or was not upto the mark, it had to be discarded.
We worked on two scripts, we changed the script for writeback buffer and hardware prefetchers specifically as we were not getting the required trend. The below work has been done on Old attacking script.

Metric for Execution time:

Initially we tried with Linux time command in order to obtain the execution time for the various simulations but the trend was not as expected and also with the suggestion of Mitali Ma'am we switched to CPU cycles and host_seconds value of the m5out/stats.txt file for measuring the performance of the program.

Old attacking scripts:

BwRead:

```
#include<stdio.h>
#define mem_size 1024*1024
int main()
{
	char read_arr[mem_size];
	long long int var=0;
	for(int i=0;i<mem_size;i++)
	{
		var=read_arr[i];
		//printf("%c:%d\n",read_arr[i],i+1);
	}
	return 0;
}
```

BwWrite:

```
#include<stdio.h>
#define mem_size 1024*1024
int main()
{
	char ch='a';
	char read_arr[mem_size];
	for(int i=0;i<mem_size;i++)
	{
		read_arr[i]=ch;
		//printf("%c:%d\n",ch,i+1);
	}
	return 0;
}
```

We did the following simulations on the old attacking script:

BwRead and BwWrite attacks on three benchmarks (PCA, Disparity, Susan):

CPU	Benchmark	Attackers	Attack type
Minor	Susan	0,1,2,3	BwRead and BwWrite
	PCA	0,1,2,3	
	Disparity	0,1,2,3	
	K-Means	0, 1, 2, 3,	
HPI	Susan	0,1,2,3	BwRead and BwWrite
	PCA	0,1,2,3	
	Disparity	0,1,2,3	
	K-means	0,1,2,3	

We have carried out the above simulations on Minor and HPI for BwRead and BwWrite attacks but as BwWrite is much more computing intensive operation so it must increase the execution time substantially as compared to BwRead. But the changes we observed were not that significant. So we tried to change the attacking scripts to get better results and thus had to re-run all the simulations for K means on minor and HPI as well as the out of order CPU.

Impact of writeback buffer:

These configurations did not give the required trend for writeback buffers i.e. when the number of writeback buffers were increased the execution time of the (program + attacker script) started to increase which was not expected. The following configurations were tried but we got the same trend on all of these.

CPU	Benchmark	Writeback buffer size	Attacker Configuration
Minor (old script)	Susan	8,16,24	0,1,2,3
	K means	1, 2, 8, 16	0,1,2,3
	Disparity	1, 2, 8, 16	2,3
	PCA	1, 2, 8, 16	1,2,3
HPI (old script)	Susan	8, 16, 24	0,1,2,3
	K means	1, 2, 8, 16	0,1,2,3
	Disparity	1, 2, 4, 6, 8	0,1,2,3
	PCA	1, 2, 8, 16	0,1,2,3

Impact of Hardware Prefetchers:

CPU	Benchmark	Prefetcher configuration
Minor	K means	No prefetcher
		L1D
		L2
		Both enabled
HPI	K means	No prefetcher
		L1D
		L2
		Both enabled

Impact of MSHRs:

We have tried running simulations on Susan, K-means, PCA and disparity for different number of MSHRs but the L2 overall miss rate and the CPU cycles were not changing, i.e. when we varied the number of MSHRs, the both L2 overall miss rate and the number of CPU cycles were constant for Read, Write and Read & Write attacks. Although the time that we got from Linux time command was changing on a different number of MSHRs but the trend was not as expected probably because the time command does not give us actual time taken on that particular victim core rather it gives the overall time taken by the program.

Bwread:

CPU	Benchmark	# MSHRs
Minor	Susan	4, 8, 16, 32
	K-means	4, 8,16, 24, 32
	PCA	4, 8, 16, 24, 32
	Disparity	4, 8, 16, 24, 32
HPI	Susan	16, 24, 32, 40
	K-means	16, 24, 32, 40
	PCA	16, 24, 32, 40
	Disparity	16, 24, 32, 40

Bwwrite:

CPU	Benchmark	# MSHRs
Minor	Susan	4, 8, 16, 32
	K-means	4, 8, 16, 24, 32
	PCA	4, 8, 16, 24, 32

	Disparity	4, 8, 16, 24, 32
HPI	Susan	16, 24, 32, 40
	K-means	16, 24, 32, 40
	PCA	16, 24, 32, 40
	Disparity	16, 24, 32, 40

Bwread & Bwwrite together:

CPU	Benchmark	# MSHRs
HPI	Susan	16, 24, 32, 40
	K-means	16, 24, 32, 40
	PCA	16, 24, 32, 40
	Disparity	16, 24, 32, 40
Minor	Susan	4, 8, 16, 32
	K-means	4, 8, 16, 24, 32
	PCA	4, 8, 16, 24, 32
	Disparity	4, 8, 16, 24, 32

The new scripts:

BwRead:

```
#include<stdio.h>
#include<vector>
#define mem_size 8*1024*1024
#define ll long long
using namespace std;
int main()
{
	//char read_arr[mem_size];
	vector<char> c(mem_size,'A');
	ll x=65;
	ll b=0;
	for(ll i=0;i<mem_size;i+=64LL)
	{
		b+=((c[i]%26)+x)%100;
	}
	return 0;
}
```

BwWrite:

```
#include<stdio.h>
#include<vector>
#define mem_size 8*1024*1024
#define ll long long
using namespace std;
int main()
{
	//char read_arr[mem_size];
	vector<char> c(mem_size,'A');
	ll x=65;
	for(ll i=0;i<mem_size;i+=64LL)
	{
		c[i]=(i%26)+x;
	}
	return 0;
}
```

Impact of Writeback buffer:

CPU	Benchmark	Writeback buffer size	Attacker configuration
Minor (new script)	K-Means	1,2, 5, 8,16,24	2,3
	Disparity	1, 2, 4, 6, 8, 16	2,3
HPI (new script)	K-means	8, 16, 24	0,1,2,3
	Disparity	8, 6, 24	0,1,2,3
	PCA	8, 6, 24	0,1,2,3

For disparity benchmark at write back buffer size of 1,2,4 we observed the required trend, the same has been put in the report and the presentation as well.

Impact of Hardware Prefetchers:

CPU	Benchmark	Prefetcher configuration With 3 BwWrite attackers
Minor	K means	No prefetcher
		L1D
		L2
		Both enabled

The impact of hardware prefetchers with 3 BwWrite attackers gave the required trend for disparity benchmark, these results have been put in the report as well as the presentation.

OS based solution:

We tried integrating CommMonitor in the system configuration but were only able to get latency stats. We tried adding more DPRINTF to the CommMonitor debug flag to print average read, write bandwidth in “comm_monitor.cc” but we ran into errors when we were rebuilding gem5