Cache Bank-Aware Denial-of-Service Attacks on Multicore ARM Processors

Michael Bechtel, Heechul Yun
University of Kansas, USA.
{mbechtel, heechul.yun}@ku.edu

***Abstract*—In this paper, we identify that bank contention in the shared last-level cache (LLC) of multicore processors can cause significant execution time slowdown to cross-core victims even when the cache is partitioned. We propose a Cache Bank-Aware Denial-of-Service (DoS) attack, which is specially engineered to induce bank contention. This is accomplished by leveraging publicly available cache bank mapping information on commercial off-the-shelf (COTS) ARM multicore processors to generate a large number of concurrent memory accesses to a particular cache bank. Importantly, this attack can be mounted from the user-space to delay cross-core victims without requiring any special privilege or OS level support. We implement and evaluate the proposed DoS attack on two ARM multicore platforms using both synthetic and real-world workloads as victim tasks. The results show that the proposed Cache Bank-Aware DoS attack causes up to 9.7X slowdown in synthetic workloads and 2.3X slowdown in real-world workloads even when state-of-the-art isolation techniques, namely LLC partitioning and DRAM bandwidth throttling, are used to protect the victim. In other words, the proposed attack can effectively bypass existing DoS attack mitigation techniques. We identify LLC bandwidth throttling and LLC bank partitioning as potential mitigation solutions and discuss their limitations.**

I. Introduction

Multicore computing platforms are increasingly utilized in safety-critical cyber physical systems as they can offer significant performance improvements while simultaneously meeting size, weight, and power (SWaP) constraints. However, in multicore platforms, contention for shared hardware resources is a major challenge as it can break the temporal isolation needed for critical real-time tasks. Moreover, such contention can be intentionally induced by malicious attackers with the goal of jeopardizing system predictability and safety. Such attacks are called *Microarchitectural Denial-of-Service (DoS)* attacks [5], [8], [41] and they are especially problematic in networked embedded systems, such as connected vehicles and edge servers, where multiple applications or virtual machines can be co-scheduled on a single multicore platform [16].

Microarchitectural DoS attacks target various shared hardware resources in a platform's memory hierarchy. Prior works have shown that partitioning the cache space [28], [47] alone is not sufficient to provide temporal isolation on a multicore system. This is because the cache may still have other internal shared resources, such as miss-status-holding registers (MSHRs) and write-back buffers, that can be exploited to mount successful DoS attacks [8], [41]. DRAM bank contention is another well-known source of interference, which led to many proposals for partitioning them to improve isolation [14], [24], [40]. Recently, it has been shown that DRAM bank contention can exacerbate the cache blocking problem by slowing down accesses to DRAM from the cache [5].

In this paper, we identify that bank contention in the shared last-level cache (LLC) can also be exploited for DoS attacks. Similar to DRAM, CPU caches, which are made of static random access memory (SRAM), commonly employ banked architectures to enable parallel accesses and improve throughput. Much like DRAM bank contention, the SRAM banks in a cache can also suffer from bank contention if multiple accesses are made to the same bank simultaneously [34].

Based on this insight, we propose a *Cache Bank-Aware DoS attack*, which is designed to induce bank contention on the shared last-level cache (LLC) in a multicore processor. Our attack leverages LLC bank address mapping information and generates many accesses to a particular cache bank in the LLC to maximize bank contention, which in turn slows down victim accesses to that same bank. We implement and validate our proposed Cache Bank-Aware DoS attack on two popular embedded multicore platforms, a Raspberry Pi 4 and an Nvidia Jetson Nano, using both synthetic and real-world representative workloads. We find that our attack can generate up to 9.7X slowdown to cross-core victim tasks running in isolation. Furthermore, we show that existing isolation mechanisms, such as cache partitioning and DRAM bandwidth throttling [5], [8], [41], are unable to effectively mitigate our attack. This is because, unlike prior DoS attacks that are designed to generate many LLC misses [5], [8], [41], our Cache Bank-Aware DoS attack generates LLC hits, not misses, and thus does not consume any DRAM bandwidth.

To defend against our Cache Bank-Aware DoS attack, we explore both software- and hardware-based prevention mechanisms. We first propose a software-based solution, which extends MemGuard [49], to throttle the LLC bandwidth (not DRAM bandwidth) of the offending cores. We show that it can mitigate our attack but it comes at a very high performance hit to best-effort tasks on the throttled cores. We also evaluate a cache bank partitioning solution and show that it can effectively prevent our attack without significant performance impacts, although it would require hardware support for general applications.

This paper makes the following **contributions**:

- We identify that cache bank contention can be exploited by malicious actors for deploying effective Microarchitectural DoS attacks in multicore processors.

- We propose and implement a *Cache Bank-Aware DoS attack* that is specifically designed to induce bank contention in the shared LLC of a multicore processor.
- We show that our attack bypasses existing DoS attack mitigation mechanisms, namely LLC partitioning and DRAM bandwidth throttling.
- We explore software- and hardware-based mitigation strategies for addressing our proposed DoS attack and evaluate their pros and cons.

The remainder of this paper is organized as follows: Section II provides necessary background information on shared caches and SRAM bank architectures. Section III details the DoS attacks we evaluate in this paper, including our proposed Cache Bank-Aware DoS attack. Section IV then evaluates the DoS attacks on two popular embedded multicore platforms, and Section V explores possible strategies for mitigating our proposed attack. We discuss related work in Section VI and conclude in Section VII.

II. Background

In this section, we provide necessary background on microarchitectural DoS attacks and cache banking.

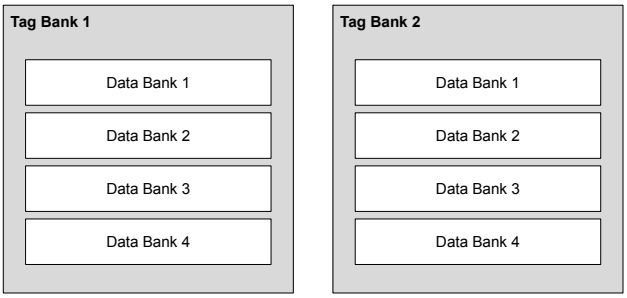
A. Microarchitectural Denial-of-Service Attacks

On modern multicore platforms, the processors are designed such that parallel accesses can be made throughout the entire memory hierarchy. At the shared cache level (e.g. LLC), this is often achieved through the use of non-blocking caches that allow for multiple concurrent misses. However, these non-blocking caches can only support a finite number of misses, which is dictated by the available hardware resources (e.g. MSHRs). If this limit is reached, the cache blocks any further accesses to it, which is known as *cache blocking*. Likewise, at the main memory level, DRAM chips are composed of multiple banks that can be accessed in parallel. While each individual bank can only service one memory request at a given time, the employment of multiple banks allow for parallel accesses to be made to all banks simultaneously. On the other hand, multiple memory accesses to the same bank can instead lead to reduced memory performance due to serialization, which is called *DRAM bank contention*.

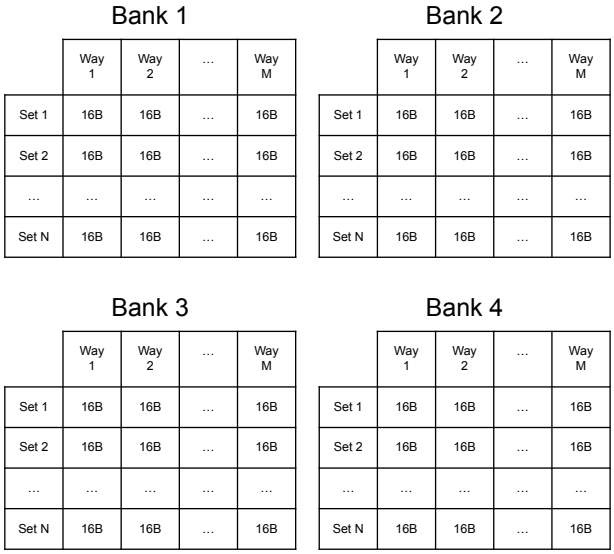
Prior works have demonstrated that cache (LLC) blocking can be exploited by malicious actors to perform DoS attacks [8], [41]. In these works, the attacks generate large amounts of cache misses in order to induce LLC blocking. In doing so, they prevent any important real-time tasks from accessing the LLC. Taking it one step further, the authors of [5] were able to induce both LLC blocking and DRAM bank contention by carefully forcing LLC misses to stress a certain DRAM bank. This resulted in their DoS attack being significantly more effective in delaying a cross-core victim that had its own dedicated cache partition. These prior works also showed that cache space partitioning is not effective in protecting victim tasks from potential DoS attacks, but that DRAM bandwidth throttling is an effective mitigation method [8], [41]. In this work, however, we find that DRAM bandwidth throttling might not be sufficient to protect against attacks that generate shared LLC bank contention.

B. Shared LLC and Cache Banking

In modern CPUs, shared caches employ multiple banks to allow for more parallel accesses. Each bank is independently addressable and has its own data lines. For example, both the ARM Cortex-A57 and Cortex-A72 processors, which we use in this paper, employ a shared LLC that is partitioned into two tag banks to enable up to two simultaneous access to the L2 cache [1]. Each tag bank is further partitioned into four data banks, which enables streaming accesses to those data banks. Figure 1a gives a depiction of the bank layout used in the shared LLC for both the A57 and A72 [1], [2].



(a) Top-level layout



(b) Detailed data-bank layout of a tag-bank

Fig. 1: L2 bank layout in ARM Cortex-A57 and Cortex-A72.

With regards to bank accesses, the mapping schemes used to allocate and access those banks are publicly available in the reference manuals for both cores [1], [2]. Namely, physical address bit 6 is used to select the tag bank, while bits 4 and 5 are used to select which of the four data banks should be accessed. Note that bits 4 and 5 are part of the offset address within a cache-line (64 bytes). In other words, a single cache line is split across four different LLC banks, with each

bank containing one fourth of the cache-line data (16 bytes). Figure 1b depicts four of such LLC data banks. On the other hand, bit 6 is outside of the cache-line granularity. As such, two adjacent cache-lines are mapped to different tag banks, which can be accessed simultaneously. However, if two or more accesses are made to the same tag bank, they will suffer from cache bank contention due to inevitable serialization. This property can be exploited by attackers as we discuss in the following section.

III. Cache Bank-aware Denial-of-Service Attacks

In this section, we present the threat model, baseline microarchitectural DoS attacks and our proposed Cache Bank-Aware DoS attack.

A. Threat Model

We assume that a victim task and one or more attacker tasks are co-located on a multicore platform, which has a shared last-level cache (LLC) and main memory (DRAM). We assume that the victim and the attackers are partitioned to run on dedicated CPU cores and LLC cache spaces. In addition, we assume that the attackers have non-privileged access on the target platforms and can only execute code from the userspace. Lastly, we assume that the cache bank address mapping information is known beforehand. Note that such mappings are often publicly available for ARM cores [1], [2]. We consider our assumptions realistic in many current and future multicore embedded systems deployment scenarios (e.g., connected ground/air vehicles and edge/IoT devices).

B. Baseline DoS Attacks

In this work, we use two previously proposed DoS attacks as baselines. Namely, we use the *bandwidth* and *latency-mlp* benchmarks from the IsolBench suite [41], which are engineered to generate continuous memory accesses so as to cause LLC or memory contention. The main difference between the two is in their memory access patterns: bandwidth performs sequential accesses over a 1D array, whereas latency-mlp performs pointer chasing operations over multiple randomly shuffled parallel linked lists (PLL). Both benchmarks can be further configured to perform either read or write accesses, and their working set size (WSS) can be adjusted to fit in any level of the memory hierarchy. We configure them to perform write operations, which have been shown to be more effective than reads [8], and adjust their WSS to fit in either the LLC (2X size of L1 data cache) or DRAM (2X size of LLC). We refer to instances of the bandwidth DoS attacks as *BwWrite(LLC)* and *BwWrite(DRAM)*. Likewise, we call instances of the latency-mlp DoS attacks as *PLLWrite(LLC)* and *PLLWrite(DRAM)*, respectively. Figure 2 shows the code listings of the baseline DoS attacks. Note that the memory level parallelism (MLP) of PLLWrite is experimentally determined, based on the method described in [41] (mlp=6 on both A72 and A57 core).

```
1  for (int64_t i = 0; i<mem_size; i += LINE_SIZE)
2  {
3      ptr[i] = 0xff;
4  }
```

(a) *BwWrite*

```
1  static int* list[MAX_MLP];
2  static int next[MAX_MLP];
3
4  for (int64_t i = 0; i < iter; i++) {
5      switch (mlp) {
6      case MAX_MLP:
7      .
8      .
9      case 2:
10         list[1][next[1]+1] = 0xff;
11         next[1] = list[1][next[1]];
12         /* fall-through */
13     case 1:
14         list[0][next[0]+1] = 0xff;
15         next[0] = list[0][next[0]];
16     }
17 }
```

(b) *PLLWrite*

Fig. 2: Baseline DoS attacks: *BwWrite* and *PLLWrite* perform sequential and random memory updates, respectively.

C. Cache Bank-Aware DoS Attack

Our proposed Cache Bank-Aware attack is based on the PLLWrite(LLC) attack described above, but is modified to target a specific cache bank in the LLC so as to generate maximum contention on that cache bank. The attack creates multiple randomly shuffled linked lists over a memory block. When creating the linked-lists, it calculates which cache bank each entry would be assigned to, and only uses addresses that would access the user specified target bank.

```
1  #define bit(addr,x)  ((addr >> (x)) & 0x1)
2  int paddr_to_sram_bank(unsigned long addr)
3  {
4      return ( (bit(addr, 6) << 2) |
5               (bit(addr, 5) << 1) |
6               bit(addr, 4) );
7  }
```

Fig. 3: Cache bank mapping function for Cortex-A57 and A72

Figure 3 shows a code listing for the bank address mapping function used in creating the linked lists. For a given physical address, this function returns a value between 0 and 7 that represents the cache bank the address would access. As a result, this allows us to manually pick physical addresses that will access a target SRAM bank. Note that the function uses low address bits 4, 5, and 6, all of which are within a 4KB virtual memory page boundary. This means that no special privilege is needed to control these address bits in order to target a specific cache bank. As such, this attack is unlike prior work that targets DRAM banks [5], which required privileged

system access or HugePage support to control DRAM bank address bits because they are generally outside of a 4KB page boundary. We set the WSS of our proposed attack to be LLC-fitting in order to maximize the amount LLC hits generated. The intuition here is that continuous LLC hits would generate more cache bank contention than LLC misses due to the faster access rate. As such, we refer to instances of our cache bank-aware attack as *BkPLLWrite(LLC)*.

IV. Evaluation

In this section, we evaluate the effectiveness of our proposed Cache Bank-Aware DoS attack on two out-of-order ARM-based multicore platforms using both synthetic and real-world victim workloads.

A. Multicore Platforms

We deploy all DoS attacks on two ARM-based embedded multicore platforms: a Raspberry Pi 4 Model B and an Nvidia Jetson Nano. The Raspberry Pi 4 equips a quad-core cluster of Cortex-A72 cores, while the Jetson Nano equips a quad-core cluster of Cortex-A57 cores. For both platforms, the L2 cache bank mapping information is publicly available in the reference manuals for their respective ARM cores, with the same addressing scheme being used for both of them [1], [2]. Table I shows the basic characteristics of the two platforms.

Platform	Raspberry Pi 4 (B)	Nvidia Jetson Nano
SoC	BCM2711	Tegra X1
CPU	4x Cortex-A72 out-of-order 1.5GHz	4x Cortex-A57 out-of-order 1.43GHz
Private L1 Cache Shared L2 Cache	48KB(I)/32KB(D) 1MB (16-way)	48KB(I)/32KB(D) 2MB (16-way)
L2 (LLC) Bank Bits	4, 5, 6	4, 5, 6
Memory	4GB LPDDR4	4GB LPDDR4

TABLE I: Compared embedded multicore platforms.

For the operating systems, the Pi 4 runs Raspberry Pi OS 64-bit with Linux kernel 5.15, and the Nano runs Ubuntu 18.04 with Linux kernel 4.9. On both platforms, we employ a page coloring technique to perform LLC set partitioning so that the DoS attacks can not directly evict any victim task data from the LLC. To that end, we patch the Linux kernel on both tested platforms with the PALLOC memory allocator [48], which exploits virtual memory page translations to enforce page allocations to specific page colors. With PALLOC, we partition the LLC into four equally sized partitions (colors) and perform a 2/2 split of those partitions such that the victim task gets half of the LLC space, while the attackers collectively share the other half.

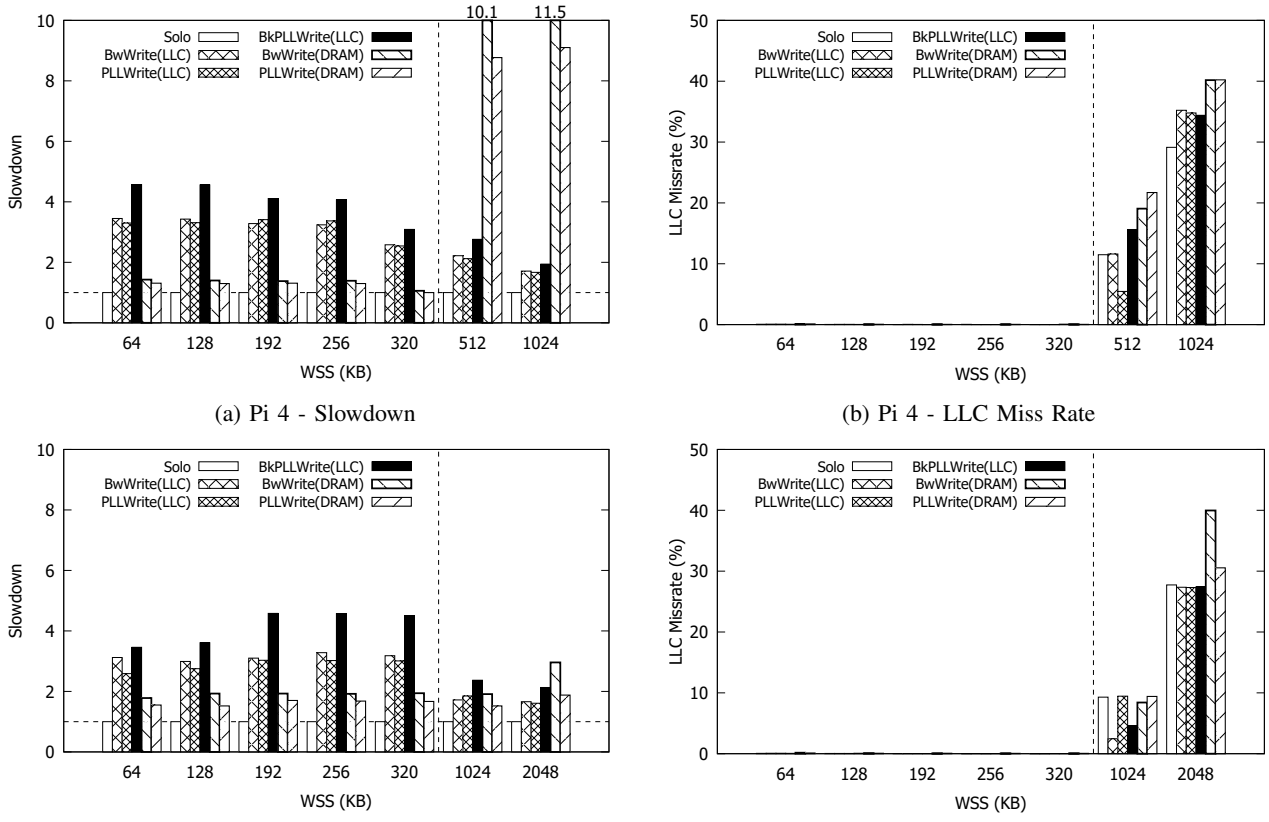
B. Impact on Synthetic Workloads

In this experiment, we evaluate the impacts of our proposed Cache Bank-Aware DoS attack to a synthetic victim task and compare its performance against the baseline DoS attacks. The experiment setup is as follows: We run one instance of the victim task alone on a single core (Core 0), and profile it to obtain its solo performance. We then repeat the experiment but this time with three instances of DoS attacker tasks on the other cores (Cores 1-3). We compare the victim's performance under the DoS attacks with that of the solo case to quantify the impacts of the DoS attackers. For the victim, we use the *bandwidth* benchmark, which is configured to perform read accesses. We vary the victim task's WSS from 64KB to 320KB, in increments of 64KB, all of which fit within the dedicated cache partition (half of the LLC) and thus cannot suffer from cache evictions by the DoS attackers. We also include two larger WSS that do not fit into the victim's cache partition and thus may suffer from self evictions and DRAM-level interference.

Figure 4 shows the results of all DoS attacks on both the Pi 4 and the Nano. Note that the vertical dotted line in all of the graphs represents the dedicated LLC space for the victim task, with WSS left of the line being LLC-fitting and WSS right of the line being DRAM-fitting. We use this same notation for all subsequent experiments. First, we find that when the WSS of the victim task fits in its cache partition (left of the dotted vertical lines), the proposed Cache Bank-Aware attack, BkPLLWrite(LLC), is noticeably more effective in slowing down the victim's execution time than the bank-unaware baseline DoS attacks. On both platforms, we see up to 4.6X slowdown from the BkPLLWrite(LLC) attackers, whereas the baseline DoS attacks only generate up to 3.5X and 3.3X slowdown on the Pi 4 and Nano, respectively. It is important to note that none of these slowdowns are caused by cache evictions, as the LLC miss rates in Figures 4b and 4d clearly show. Because the cache is partitioned, the attacker tasks are unable to evict the victim's cache lines. As such, the observed slowdowns are due to cache bank contention, with our BkPLLWrite(LLC) attack generating more contention compared to cache-bank oblivious DoS attacks.

On the other hand, when the WSS of the victim task is large enough (right of the dotted vertical lines) such that it experiences significant LLC misses (due to self-evictions), we find that the baseline BwWrite(DRAM) and PLLWrite(DRAM) attackers start to outperform our BkPLL-Write(LLC) attackers. This was especially the case on the Pi 4, as the BwWrite(DRAM) attack generated 11.5X slowdown against the 1MB WSS victim, while our BkPLLWrite(LLC) attacks remained at $\sim$2.8X slowdown. The same also happened on the Nano but to a lesser degree, with the BwWrite(DRAM) attackers causing 3X slowdown against the 2MB WSS victim. This is because the victim tasks in these cases (DRAM-fitting WSS) are sensitive to both LLC and DRAM level interference. As such, the baseline DRAM-fitting DoS attackers are able to interfere with the victim at both the LLC and in DRAM. Our BkPLLWrite(LLC) attackers are instead specifically designed to only interfere at the LLC level, which is why they are less effective against DRAM-sensitive victim tasks.

From these results, we make two key findings. First, our proposed Cache Bank-Aware DoS attacks outperform baseline attacks against LLC-sensitive victim workloads. Second, we find that *Cache Bank-Aware DoS attacks effectively bypass LLC set partitioning mechanisms*. This is because cache bank



(a) Pi 4 - Slowdown

(b) Pi 4 - LLC Miss Rate

(c) Nano - Slowdown

(d) Nano - LLC Miss Rate

Fig. 4: Impacts of DoS attackers to a synthetic victim task running on a dedicated core and cache partition (Y-axis: victim's execution time slowdown ratio for 4a and 4c, and victim task LLC miss rate for 4b and 4d. X-axis: victim's WSS. The dotted vertical lines indicate the cache partition boundary.)

allocations happen at a finer cache line granularity that can not be accounted for with LLC set partitioning.

C. Impact of Cache Bank Partitioning

In this experiment, we evaluate the impact that cache bank partitioning has in mitigating LLC bank contention. Much like prior works [5], [8], [41], we found that LLC set partitioning was unable to protect against DoS attacks, including our Cache Bank-Aware DoS attack, because other hardware structures in the LLC are still shared. In our case, LLC banks were still accessible by all cores meaning that all LLC-fitting DoS attacks could still effectively slowdown victim performance. As such, we next want to determine whether partitioning the LLC banks could help protect the victim tasks.

Note that cache bank partitioning cannot be realized via OS-level page coloring because the page size granularity is too coarse grained for cache bank control. Also, there is no hardware-level support for partitioning the cache at a cache bank granularity. Instead, we perform a simple experiment to simulate a bank partitioned LLC. Concretely, we employ a BkPLLRead(LLC) instance as a victim task and allocate all of its memory addresses to cache data bank 0. We then run it alongside three other BkPLLWrite(LLC) instances as the attackers, and vary which data bank the attackers access. In total, this gives us eight data points, one for each data bank in the LLC.

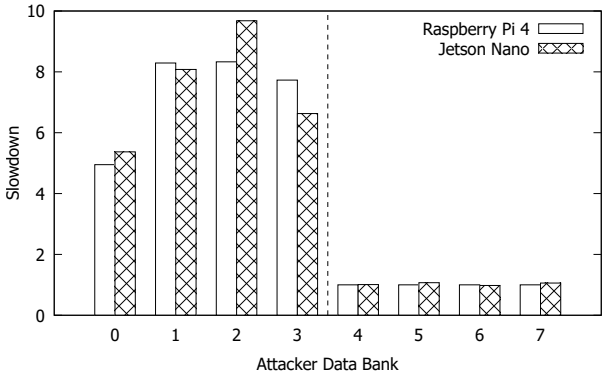


Fig. 5: Impact of LLC bank partitioning on DoS attacker effectiveness. The victim is allocated to data bank 0 while the attacker's allocated bank varies from 0 to 7 (data bank 0-3 $\in$ tag bank 0; data bank 4-7 $\in$ tag bank 1).

Figure 5 shows the results on the Pi 4 and Nano. First, we find that the victim task experiences significant contention (up to $\sim$9.7X slowdown) when it is assigned to the same LLC

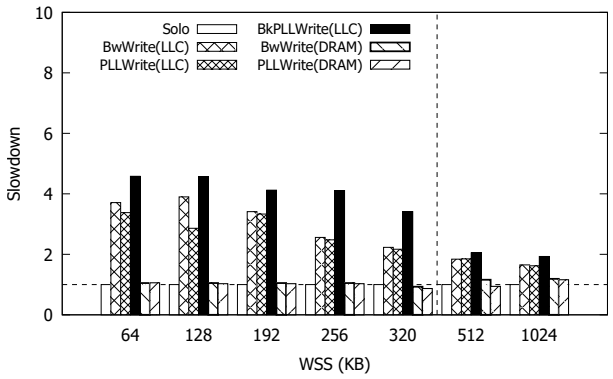
tag bank (data banks 0-3) as the attacker tasks. On the other hand, when the attackers are allocated on a different LLC tag bank (data bank 4-7), the victim achieves near perfect isolation, indicating that the slowdowns caused by LLC-fitting DoS attackers are mainly due to LLC tag bank contention. Interestingly, when both the attackers and the victim are accessing the same tag and data bank (data bank 0), the victim task's observed slowdown is considerably less than when they access the same tag bank but different data banks (data bank 1-3). We speculate that this is due to differences in how sub cache-line accesses are processed at the cache controller.

D. Impact of DRAM Bandwidth Throttling

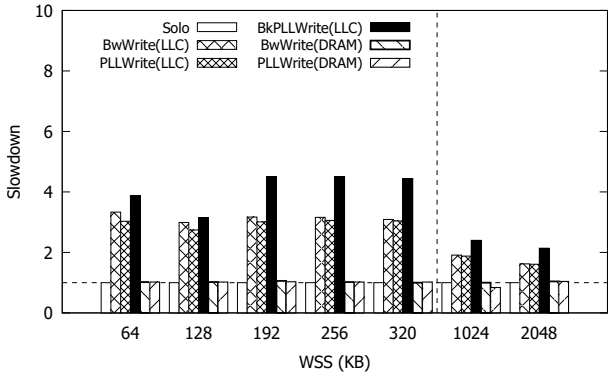
In this experiment, we test the impact of memory bandwidth throttling in mitigating DoS attacks. Note that prior work suggested memory throttling to be an effective solution for mitigating DoS attacks [8]. As in [8], we used a modified MemGuard tool [49], which uses hardware performance counters to regulate memory bandwidth on a per-core basis. That is, each core is given a set bandwidth budget, which it is allowed to consume over the course of a period (e.g. 1 ms). If any core empties its allotted budget before the end of the period, the core is throttled by the OS until the beginning of the next period. In our case, we configure MemGuard so that all three attacker cores only receive a memory bandwidth budget of 100 MB/s each. The victim core then has unlimited access to the memory bandwidth (i.e. it is never throttled).

Figure 6 shows the victim task slowdowns on the Pi 4 and Nano. First and foremost, we find that memory bandwidth throttling is extremely effective at mitigating the DRAM-fitting baseline DoS attacks on both platforms. In the worst case, we only see slowdowns of 6% to LLC-fitting victims on both the Pi 4 and Nano. However, for the LLC-fitting attackers we find their performance to be unaffected, with the BkPLLWrite(LLC) attacks still achieving worst case slowdowns of ∼4.5X on both platforms. This is because LLC-fitting attackers are specifically intended to only access the LLC (w/o generating any LLC misses). This means that the attacker cores are never throttled as no memory accesses are made and, by proxy, no memory bandwidth is consumed. In the following experiments, we keep MemGuard enabled and do not evaluate the baseline DRAM-fitting DoS attacks as they can be effectively mitigated via MemGuard's memory bandwidth throttling. We instead focus on evaluating LLC-fitting DoS attacks.

These results show that our *Cache Bank-Aware DoS attacks are able to effectively bypass both DRAM bandwidth throttling and LLC set partitioning mechanisms*. Since these two mechanisms are the two of the most commonly employed software-based solutions for combating temporal interference on multicore platforms, we posit that newer solutions, either software or hardware-based, are necessary to mitigate such DoS attacks. We discuss such possibilities further in Section V.



(a) Raspberry Pi 4



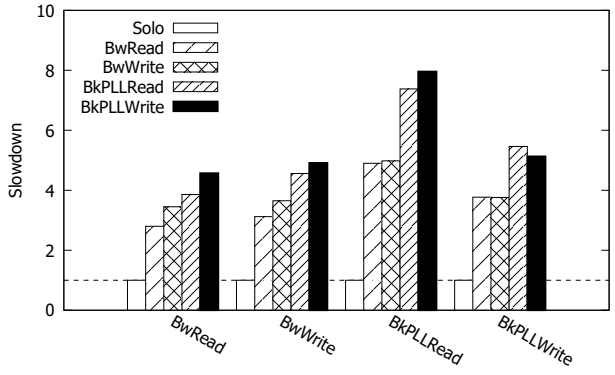
(b) Jetson Nano

Fig. 6: Impact of DRAM bandwidth throttling on DoS attacker effectiveness.

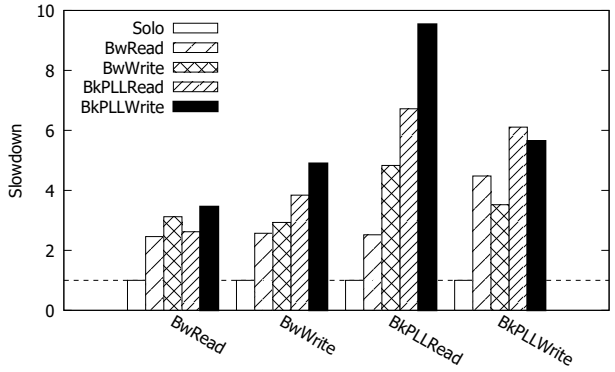
E. Impact of Memory Access Pattern

In this experiment, we explore the effect of memory access patterns. Specifically, we vary both the victim and attacker tasks to be either read or write tasks in a given scenario (e.g. read victim vs. read attackers, read victim vs. write attackers, etc.). For both the victim and attacker tasks, we use the BwRead, BwWrite, BkPLLRead, and BkPLLWrite tasks, and fix all of their WSSs to 64KB. Note that both cache partitioning and MemGuard are enabled in this experiment (and all subsequent ones) to ensure neither cache evictions nor DRAM bandwidth contention can affect the results.

Figure 7 shows the results on the Pi 4 and Nano. Note first that, between read and write attackers, the latter are generally more effective in delaying the victim task in most cases. For example, BkPLLWrite attackers (vs. a BkPLLRead victim) generate up to 9.5X slowdown on Nano, whereas the BkPLLRead attackers only generate up to 6.7X slowdown against the same victim. However, the differences between read and write DoS attackers are somewhat less pronounced in Pi 4. Second, between read and write victim tasks, the results are mixed as the BkPLLRead victim is more susceptible to the DoS attacks than the BkPLLWrite victim whereas the BwWrite victim is more susceptible than the BwRead victim on both platforms.

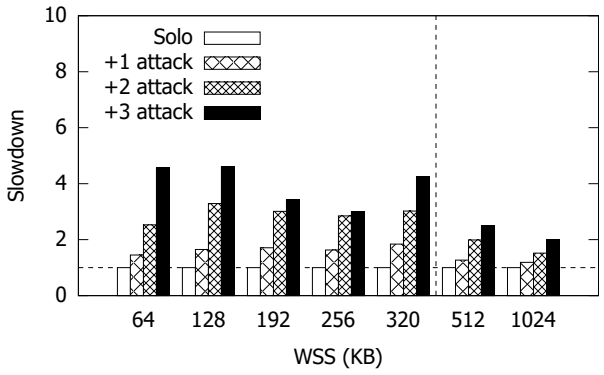


(a) Pi 4

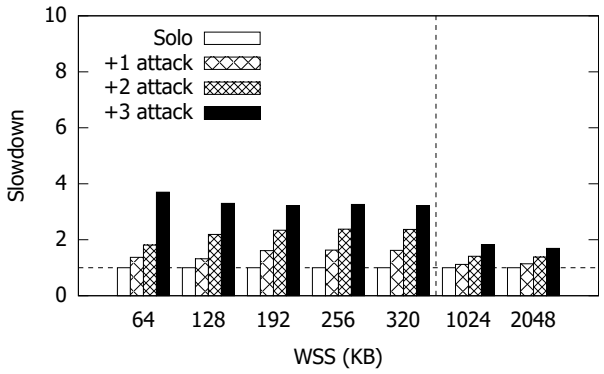


(b) Nano

Fig. 7: Impact of memory access patterns in victim and attacker combinations. Note that the x-axis shows the victim tasks used, while the legends (except Solo) indicate different DoS attacker tasks used to delay the victims.



(a) Raspberry Pi 4



(b) Jetson Nano

Fig. 8: Impact of varying the number of concurrent BkPLLkWrite(LLC) attackers.

F. Impact of Varying the Number of Attackers

In this experiment, we evaluate the impact that the number of concurrent DoS attackers has on victim performance. To this point, we have allowed the attackers to occupy all remaining CPU cores not utilized by the victim task. But this may not always be a realistic assumption for some use-case scenarios where the attackers may have access to only a portion of the available hardware resources. For example, in a cloud-based or virtualized environment, the attacker may have access to only a handful of CPU cores. As such, we want to determine if our Cache Bank-Aware DoS attacks could potentially work in such settings. To test this, we run the same synthetic victim task experiments as before, but vary the number of attacker tasks from 1 to 3. Note that we only consider *BwRead(LLC)* for the victim task, and *BkPLLWrite(LLC)* as DoS attackers in this experiment.

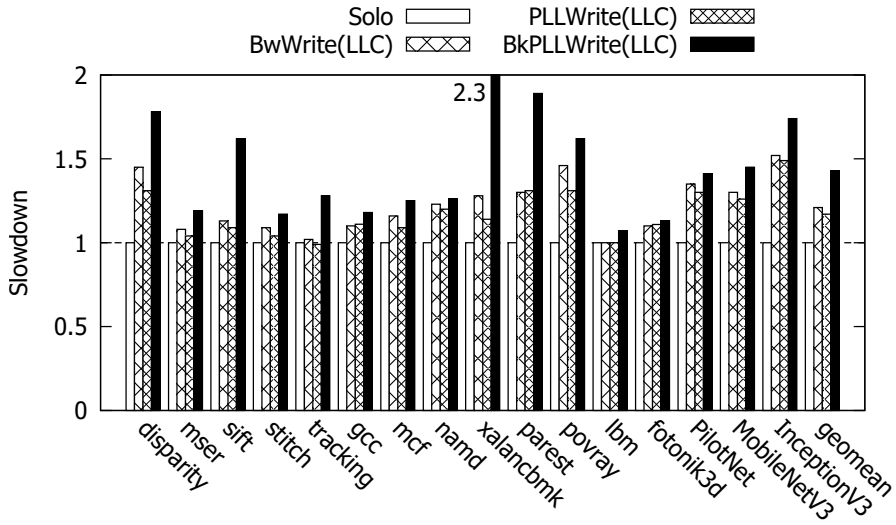
Figure 8 shows the slowdown results for the Pi 4 and Nano. As expected, we find that fewer DoS attackers do have less impact to victim performance. Largely speaking, the amount of slowdown experienced by the victim task increases linearly with the number of attackers present. On the Pi 4 the victim task sees worst case slowdowns of 1.8X, 3.3X and 4.6X with 1, 2 and 3 attackers, respectively. Likewise, on the Nano the victim task see worst case slowdowns of 1.6X, 2.4X and 3.7X as the number of attackers increases from 1 to 3. From this, we find that our Cache Bank-Aware attacks would likely still work in settings with restrictions on CPU core access, but with noticeably less impact than if all physical CPU cores were available.

G. Impact to Real-World Workloads

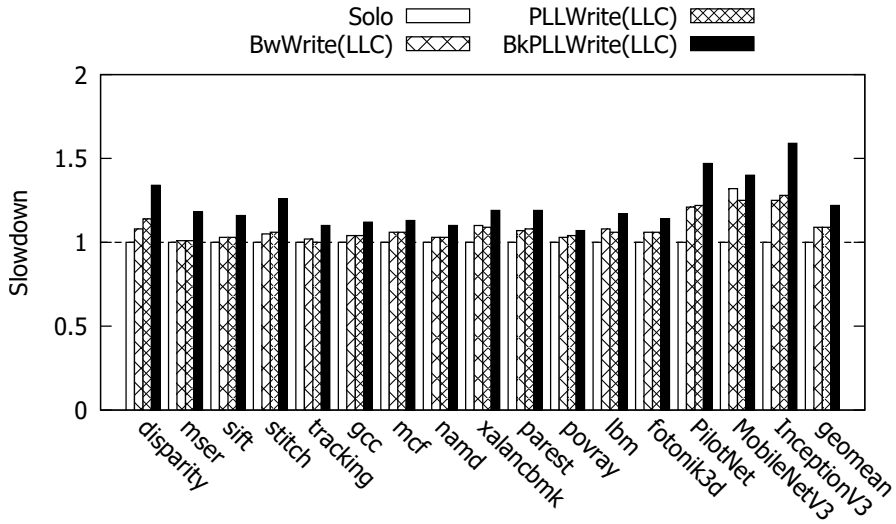
In this experiment, we evaluate the impact that DoS attackers have on representative real-world workloads. For this, we employ the following benchmarks:

- Five benchmarks from the SD-VBS suite [42], each using the *fullhd* input size.
- Eight benchmarks from the SPEC2017 suite [37], each using the default *ref* size.
- Three well-known DNN models of varying sizes: PilotNet [11], MobileNetV3 [17] and InceptionV3 [38]. For these workloads, we measure their average inference latencies across 1000 input frames.

The experiment setup is then the same as in the synthetic victim experiments. We assign the victim benchmark to Core 0 and initially run them alone. We then run each benchmark again alongside three DoS attacker instances and calculate the performance slowdowns the victim task experiences.



(a) Raspberry Pi 4



(b) Jetson Nano

Fig. 9: Impact of DoS attackers on representative real-world benchmarks. Note that both LLC partitioning and DRAM bandwidth throttling are enabled.

Figure 9 shows the results on the Pi 4 and Nano. In general, we see behavior similar to that in Figures 4 and 6. Most benchmarks are significantly impacted by the LLC-fitting DoS attacks, with our cache bank-aware attack, BkPLLWrite(LLC), being the most effective (>2X on average than the bank-oblivious ones). We generally see higher slowdowns on the Pi 4, up to 2.3X, than the Nano, up to 1.6X. Note that Pi 4's Cortex-A72 cores have a more advanced and faster design than Nano's Cortex-A57 cores. This includes improvements in the L2 cache for high bandwidth workloads [18], suggesting that advanced processors could be more vulnerable to our attacks. Moreover, note that these slowdowns are observed despite the fact that both LLC partitioning and memory bandwidth throttling are enabled to protect the victims.

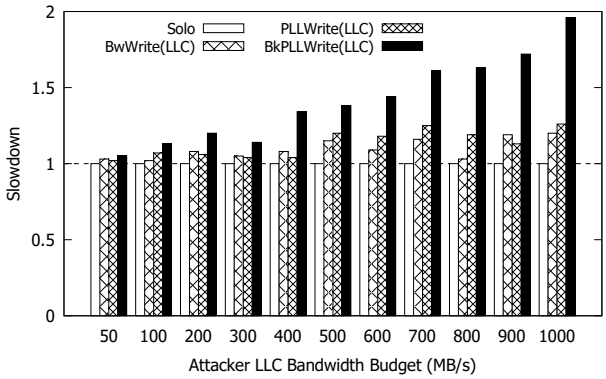
In summary, our attacks are effective in slowing down real-world workloads and thus are dangerous in real-time systems because (1) small changes to execution timings can have major impacts to system safety, and (2) they can bypass existing defense mechanisms, including memory bandwidth throttling. To address this second point, we next explore possible mitigation strategies to defend against LLC-fitting cache bank-aware DoS attacks.

V. Mitigation

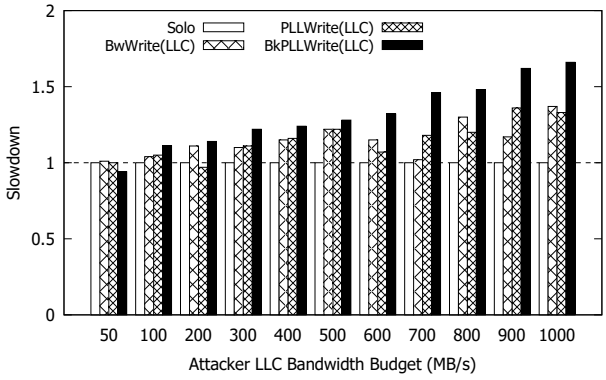
In this section, we explore a possible software-based solution and other hardware-based solutions that could be implemented to potentially protect against our attack.

A. LLC Bandwidth Throttling

As discussed in Section IV-D, DRAM bandwidth throttling is unable to mitigate our Cache Bank-Aware DoS attack because the attackers themselves do not access memory, meaning that they never get throttled. Based on this insight, we explore the potential for *LLC bandwidth throttling*. In theory, because the DoS attackers continually access the LLC, throttling LLC traffic should help to mitigate their impacts. To achieve this, we modify the vanilla MemGuard module to track L1 data cache misses instead of LLC misses. In doing so, we can instead limit the amount of LLC accesses that each CPU core is allowed to make in a given period. Using this modified version of MemGuard, which we call *LLCGuard*, we perform the same experiments as in Figure 6. In this case, though, we fix the victim WSS at 64KB and instead vary the LLC budget that each attacking core receives per period. In total, we test eleven different LLC bandwidth budgets, going from 50 MB/s up to 1000 MB/s.



(a) Raspberry Pi 4



(b) Jetson Nano

Fig. 10: Impact of LLC bandwidth throttling (LLCGuard) on DoS attacker effectiveness.

Figure 10 shows the impact of LLC bandwidth throttling on both the Pi 4 and the Nano platforms. As expected, we find that LLC bandwidth throttling can effectively protect against all LLC-fitting DoS attacks, including our Cache Bank-Aware DoS attack, when the LLC budget is set low enough. For example, at 50 MB/s the attackers only achieve worst case slowdowns of 5% and 1% on the Pi 4 and Nano, respectively. As we increase the budget, however, we again see non-trivial slowdowns to victim performance. Even at 1000 MB/s, or 1/15th of the measured peak LLC bandwidth, our cache bank-aware attacks can slowdown the victim by 2X and 1.7X on the Pi 4 and Nano, respectively.

Moreover, we find that utilizing LLCGuard could incur an unacceptably high performance cost to the throttled best-effort cores. For example, if we assume a tolerable slowdown threshold of 10% for the victim, the attacker cores' LLC bandwidth would have to be throttled to 50 MB/s on both platforms to defend against our cache bank-aware DoS attacks. Compared to the 15 GB/s peak LLC bandwidth we observe without throttling, this means that all throttled best-effort cores could suffer up to ∼300X slowdown. For reference, to achieve the same 10% slowdown against DRAM-fitting DoS attacks, memory bandwidth throttling by MemGuard could cause up to ∼40X slowdown to the throttled best-effort cores.

	Peak BW (MB/s)	Throttled BW (MB/s)	Max Slowdown
DRAM Attackers	4,000	100	40X
LLC Attackers	15,000	50	**300X**

TABLE II: Comparison of best-effort slowdowns incurred by DRAM and LLC bandwidth throttling mechanisms in order to achieve <10% real-time slowdown.

Table II compares the slowdowns incurred by both forms of bandwidth throttling. In order to avoid such a massive performance hit from throttling, we next discuss potential hardware-based solutions to protect against LLC-fitting DoS attacks.

B. Hardware-Based Solutions

To protect against LLC-fitting DoS attacks, there are several possible hardware-based solutions. First, a prime candidate is hardware-supported LLC bank partitioning. We already evaluated the impact of cache bank partitioning in IV-C, which was able to effectively protect against our Cache Bank-Aware DoS attack. However, cache bank partitioning cannot be achieved in software (at the OS level) on current ARM-based multicore platforms. This is because the physical address bits used for determining the LLC bank also fit inside the offset field for the cache address, so it is impossible for the OS to have any control over LLC bank allocations. As such, cache bank partitioning can only be effectively implemented with hardware support.

Second, our proposed software solution, LLCGuard, in V-A has shortcomings related to its software-based implementation. Namely, because it operates at a 1 ms period, it can still allow for large amounts of LLC accesses to happen in a short time. As a result, LLC bank contention can still be induced during these short bursts of accesses. Alternatively, a hardware-based throttling solution could operate at a finer granularity that instead spreads LLC accesses more evenly across the entire throttle period, as in BRU [13] and Intel RDT [19], [35], thereby potentially reducing the negative performance impact

of throttling best-effort cores. We would like to explore this possibility in the future.

Lastly, another potential solution is the use of complex address mapping schemes to map physical addresses to cache banks. In this paper, we focused on ARM-based platforms as their LLC bank mapping schemes are publicly available. In addition, when address mapping schemes are simple, as in our tested platforms, they could be reverse-engineered with little effort even without public information. As such, another mitigation strategy would be to employ a complex addressing scheme, such as those used in other CPU architectures. For example, Intel CPUs often use complex mappings for their LLC bank and slice allocations [15], which makes it difficult to reverse-engineer the mappings. However, it is important to note that a motivated adversary may still be able to reverse engineer such complex mappings. Once the mapping of a certain processor is decoded, all systems based that processor would then be vulnerable to our attacks.

VI. Related Work

Microarchitectural DoS attacks have been studied for several different types of shared resources in multicore systems. Moscibroda et al. demonstrated DoS attacks on DRAM controllers [29], and the FR-FCFS [31] scheduling algorithm. This then led to the proposals of many DRAM controller scheduling algorithms that focus on fairness [26], [30]. Keramidas et al. studied DoS attacks on cache space and proposed a cache replacement policy that allowed less space for attackers [23]. Woo et al. investigated DoS attacks on bus bandwidth and shared cache space in a simulated environment [43]. More recently, Valsan et al. and Bechtel et. al showed that cache internal hardware buffers are susceptible to DoS attacks and have demonstrated their severity [7], [8], [41]. Iorga et al. studied the DoS attacks from [8] and presented a statistical approach for evaluating DoS attacks on embedded multicore platforms [21]. Several researchers have also studied various interference channels on GPUs. Yandrofski et al, systematically investigated interference channels on Nvidia's discrete GPUs by generating various "enemy" programs [45]. Bechtel et al., studied potential DoS attacks on Intel iGPUs [6], which share a LLC with the CPU.

Even in a normal scenario, applications on a multicore platform can contend for shared hardware resources, which in turn affect their timing behaviors. Consequently, there is a large body of work in the real-time systems research community to provide stronger isolation in multicore. This includes various software and hardware mechanisms to manage the shared resources [14], [24], [25], [28], [32], [33], [36], [44], [47], [49]. This desire for greater management of shared hardware resources in multicore has been seen by major industry players as well. Intel introduced their Resource Director Technology (RDT) [20] set of tools, which can be used to manage both shared cache space and memory bandwidth with lower overheads. Likewise, ARM introduced a similar framework for their processors called Memory System Resource Partitioning and Monitoring (MPAM) [3]. Despite these efforts, it has been shown that cache space partitioning techniques are unable to mitigate DoS attacks as other hardware structures are still shared among all cores [5], [8], [41]. On the other hand, DRAM bandwidth throttling has shown much greater ability in defending against such DoS attacks [8]. In this work, however, we proposed a new DoS attack that makes both cache partitioning and DRAM bandwidth throttling ineffective on commercial off-the-shelf (COTS) ARM multicore processors by inducing contention on a cache bank.

Contention on memory (DRAM) banks is a well-known issue and has been extensively studied for many years. As early as 1985, Bailey [4] identified memory bank contention as a bottleneck for supercomputers and discussed possible solutions to mitigate performance loss. Blelloch et al. [10] evaluated the performance loss that could be caused by memory bank contention on high-bandwidth multiprocessors. As multicore architectures became the norm and the memory wall became a significant bottleneck, many researchers in both academia and industry investigated the DRAM bank contention problem [27], [39], [50]. Cache bank contention, in contrast, has received much less attention. Evenblij et al. [12] evaluated the performance impacts of bank contention in both SRAM and STT-MRAM based caches using the Gem5 simulator [9]. They found that SRAM caches were unaffected by contention when eight or more banks were present, only having a worst-case slowdown of 5%. While there are some works that exploit cache bank contention in private L1 caches as side-channels to steal secret data [22], [46], they do not consider bank contention on shared L2/L3 caches and their performance impacts to cross-core victims. In contrast, we show the impact of cache bank contention on shared L2 caches of real COTS ARM multicore processors and demonstrate significant real-world performance impacts to cross-core victims that cannot be mitigated by conventional isolation techniques, such as cache partitioning or memory bandwidth throttling.

VII. Conclusion

In this paper, we showed that deliberately generating bank contention in the shared last-level cache (LLC) is an effective way to delay cross-core victims on modern multicore platforms. We developed a new attack, called a Cache Bank-Aware DoS attack, that generates a large number of parallel requests to a specific LLC bank to induce maximum contention on the bank. From extensive experiments on two popular ARM multicore platforms, we showed that our attack significantly outperforms prior DoS attacks in delaying LLC-sensitive cross-core victim tasks. Furthermore, we showed that our attack effectively bypasses two commonly employed defense mechanisms: LLC set partitioning and DRAM bandwidth throttling. We explored both software- and hardware-based defense mechanisms and discussed their limitations in mitigating our DoS attack. For future work, we plan to evaluate our attack on diverse hardware platforms. We also plan to explore more effective mitigation techniques.

ACKNOWLEDGEMENTS

This research is supported in part by NSF grant CNS-1815959, CPS-2038923 and NSA Science of Security initiative contract no. H98230-18-D-0009.

REFERENCES

[1] ARM. *Cortex™-A57 Technical Reference Manual, Rev: r1p3*, 2016.
[2] ARM. *Cortex™-A72 Technical Reference Manual, Rev: r0p3*, 2016.
[3] ARM. *Arm Architecture Reference Manual Supplement: Memory System Resource Partitioning and Monitoring (MPAM), DDI:0598B.b*, 2020.
[4] D. H. Bailey. Vector Computer Memory Bank Contention. Technical report, NASA, 1985.
[5] M. Bechtel and H. Yun. Memory-Aware Denial-of-Service Attacks on Shared Cache in Multicore Real-Time Systems. *Transactions on Computers*, 2021.
[6] M. Bechtel and H. Yun. Denial-of-Service Attacks on Shared Resources in Intel's Integrated CPU-GPU Platforms. In *ISORC*, 2022.
[7] M. G. Bechtel, E. McEllhiney, M. Kim, and H. Yun. DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car. In *RTCSA*, 2018.
[8] M. G. Bechtel and H. Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. In *RTAS*, 2019.
[9] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The Gem5 simulator. *ACM SIGARCH Computer Architecture News*, 2011.
[10] G. E. Blelloch, P. B. Gibbons, Y. Matias, and M. Zagha. Accounting for Memory Bank Contention and Delay in High-Bandwidth Multiprocessors. In *SPAA*, 1995.
[11] M. Bojarski et al. End-to-End Learning for Self-Driving Cars. *arXiv*, 2016.
[12] T. Evenblij, M. Perumkunnil, F. Catthoor, S. Sakhare, P. Debacker, G. Kar, A. Furnemont, N. Bueno, J. I. Gómez-Pérez, and C. Tenllado. A Comparative Analysis on the Impact of Bank Contention in STT-MRAM and SRAM Based LLCs. In *ICCD*, 2019.
[13] F. Farshchi, Q. Huang, and H. Yun. BRU: Bandwidth Regulation Unit for Real-Time Multicore Processors. In *RTAS*, 2020.
[14] F. Farshchi, P. K. Valsan, R. Mancuso, and H. Yun. Deterministic Memory Abstraction and Supporting Multicore System Architecture. In *ECRTS*, 2018.
[15] A. Farshin, A. Roozbeh, G. Q. Maguire Jr, and D. Kostić. Make the Most Out of Last Level Cache in Intel Processors. In *EuroSys*, 2019.
[16] A. Hamann. Industrial Challenges: Moving From Classical to High Performance Real-Time Systems. In *WATERS*, 2018.
[17] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al. Searching for MobileNetV3. In *ICCV*, 2019.
[18] M. Humrick. ARM Cortex-A72 Architecture Deep Dive . https://www.tomshardware.com/reviews/arm-cortex-a72-architecture,4424.html.
[19] Intel. Intel® Resource Director Technology (Intel® RDT) Framework. https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html.
[20] Intel. *Intel 64 and IA-32 Architectures Software Developer Manuals, Vol 3b*, May 2020.
[21] D. Iorga, T. Sorensen, J. Wickerson, and A. F. Donaldson. Slow and Steady: Measuring and Tuning Multicore Interference. In *RTAS*, 2020.
[22] Z. H. Jiang and Y. Fei. A Novel Cache Bank Timing Attack. In *ICCAD*, 2017.
[23] G. Keramidas, P. Petoumenos, S. Kaxiras, A. Antonopoulos, and D. Serpanos. Preventing denial-of-service attacks in shared cmp caches. In *SAMOS*, 2006.
[24] H. Kim, A. Kandhalu, and R. Rajkumar. A Coordinated Approach for Practical OS-Level Cache Management in Multi-core Real-Time Systems. In *ECRTS*, 2013.
[25] N. Kim, B. C. Ward, M. Chisholm, J. H. Anderson, and F. D. Smith. Attacking the One-Out-of-M Multicore Problem by Combining Hardware Management with Mixed-Criticality Provisioning. *Real-Time Systems*, 2017.
[26] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *MICRO*, 2010.
[27] S. Li and B. Jacob. Statistical DRAM modeling. In *MEMSYS*, 2019.
[28] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni. Real-Time Cache Management Framework for Multi-core Architectures. In *RTAS*, 2013.
[29] T. Moscibroda and O. Mutlu. Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems. In *USENIX Security Symposium*, 2007.
[30] O. Mutlu and T. Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *MICRO*, 2007.
[31] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. Owens. Memory Access Scheduling. In *ACM SIGARCH Computer Architecture News*, 2000.
[32] S. Roozkhosh and R. Mancuso. The Potential of Programmable Logic in the Middle: Cache Bleaching. In *RTAS*, 2020.
[33] A. Saeed, D. Dasari, D. Ziegenbein, V. Rajasekaran, F. Rehm, M. Pressler, A. Hamann, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann. Memory Utilization-Based Dynamic Bandwidth Regulation for Temporal Isolation in Multi-Cores. In *RTAS*, 2022.
[34] J. P. Shen and M. H. Lipasti. *Modern Processor Design: Fundamentals of Superscalar Processors*. Waveland Press, 2013.
[35] P. Sohal, M. Bechtel, R. Mancuso, H. Yun, and O. Krieger. A Closer Look at Intel Resource Director Technology (RDT). In *RTNS*, pages 127–139, 2022.
[36] P. Sohal, R. Tabish, U. Drepper, and R. Mancuso. E-WarP: a System-wide Framework for Memory Bandwidth Profiling and Management. In *RTSS*, 2020.
[37] SPEC CPU2017. https://www.spec.org/cpu2017.
[38] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *CVPR*, 2016.
[39] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi. Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores. In *ISCA*, 2010.
[40] P. K. Valsan and H. Yun. Medusa: a predictable and high-performance dram controller for multicore based embedded systems. In *CPSNA*, 2015.
[41] P. K. Valsan, H. Yun, and F. Farshchi. Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems. In *RTAS*, 2016.
[42] 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 *IISWC*, 2009.
[43] D. H. Woo and H. Lee. Analyzing performance vulnerability due to resource denial of service attack on chip multiprocessors. In *CMP-MSI*, 2007.
[44] M. Xu, L. T. X. Phan, H.-Y. Choi, Y. Lin, H. Li, C. Lu, and I. Lee. Holistic Resource Allocation for Multicore Real-Time Systems. In *RTAS*, 2019.
[45] T. Yandrofski, J. Chen, N. Otterness, J. H. Anderson, and F. Smith. Making Powerful Enemies on NVIDIA GPUs. In *RTSS*, 2022.
[46] Y. Yarom, D. Genkin, and N. Heninger. CacheBleed: A Timing Attack on OpenSSL Constant-Time RSA. *Journal of Cryptographic Engineering*, 2017.
[47] Y. Ye, R. West, Z. Cheng, and Y. Li. Coloris: a Dynamic Cache Partitioning System Using Page Coloring. In *PACT*, 2014.
[48] H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni. PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms. In *RTAS*, 2014.
[49] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms. In *RTAS*, 2013.
[50] Z. Zhu and Z. Zhang. A Performance Comparison of DRAM Memory System Optimizations for SMT Processors. In *HPCA*, 2005.