Denial-of-Service Attacks on Shared Resources in Intel's Integrated CPU-GPU Platforms

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

***Abstract*—In this paper, we study the effectiveness of denial-of-service (DoS) attacks on Intel's heterogeneous multicore system-on-chips with integrated GPU (iGPU) in which the last level cache (LLC) and the main memory subsystem are shared between the multicore CPU and the iGPU. Using two Intel processors with iGPU, we evaluate four different DoS attacks, three CPU based and one iGPU based, and show they can induce very high degree of shared resource contention and thus dramatically slowdown the victim's execution time. We further evaluate the effectiveness of Intel's recent hardware based shared resource isolation mechanisms, namely Intel Cache Allocation Technology (CAT) and Graphics Technology Class of Service (GT COS), which provide shared LLC partitioning capability for the CPU cores and the iGPU, respectively, in defending against these DoS attacks. Using both synthetic and real-world benchmarks, we find that hardware based LLC partitioning mechanisms does provide spatial LLC space isolation but does not necessarily provide temporal isolation.**

***Index Terms*—Denial-of-Service Attack, Multicore, Integrated GPU, Way-based Cache Partitioning**

I. INTRODUCTION

Heterogeneous system-on-a-chip processors (SoCs), which integrate multiple CPU cores, GPU and other accelerators into a single chip, are increasingly adopted in all areas of computing. This includes many intelligent cyber-physical systems applications such as self-driving cars, drones, and smart manufacturing, which require high performance for timely processing of real-time data while meeting size, weight and power constraints [13], [16].

However, contention in shared hardware resources among the heterogeneous computing elements remains a major challenge as it can significantly impact task execution timing. Furthermore, such contention can be intentionally induced by malicious actors to perform Denial-of-Service (DoS) attacks that specifically target these shared resources. For example, it has been shown that DoS attacks targeting various internal structures in a shared LLC can dramatically impact the performance of real-time tasks on several contemporary embedded multicore platforms [6], [37].

The problem of shared resource contention in multicore is well-known and extensively studied in the real-time embedded systems community, but still with limited success, especially on complex and powerful processors. The problem is only exacerbated on heterogeneous systems as more and varied computing resources compete for the shared resources. Recent studies on NVIDIA's Jetson embedded computing platforms, which have ARM cores and an integrated GPU, show that they can suffer significant timing variations due to contention in the shared DRAM among the CPU cores and the integrated GPU (iGPU) [7], [8].

As the demand for computing increases, thanks in part due to the emergence of AI/ML workloads, powerful x86 architecture-based heterogeneous SoCs are increasingly considered in many embedded applications. For example, Tesla's Model X and Model S are known to use AMD's embedded platform with integrated graphics (Zen CPU core plus Vega iGPU) to drive display and graphics of the vehicles [9]. However, powerful x86 processors from both Intel and AMD are known to be quite complex and difficult to analyze their temporal behaviors in part because they tend to have a large amount of shared hardware resources. For instance, most Intel processors for consumer and embedded markets are already equipped with an integrated GPU, which shares not only a common DRAM memory subsystem but also a large shared last-level cache (LLC) with the CPU.

In this paper, we study the feasibility and effectiveness of micro-architectural DoS attacks on two Intel computing platforms with integrated GPUs, a Coffee Lake micro-architecture based consumer grade desktop and a Tiger Lake micro-architecture based Intel UP3 computing platform. Note that the Tiger Lake platform is specially tailored for embedded applications and has several hardware features that are potentially beneficial for embedded/real-time applications. Namely, it has Intel Cache Allocation Technology (CAT) and Graphics Technology Class of Service (GT COS) [15], which support hardware assisted way-based cache partitioning capabilities for both CPU and iGPU, respectively. In both platforms, CPU and the iGPU share not only the main memory (DRAM) but also a common shared LLC, as in all other Intel CPUs with an iGPU [17]. On these platforms, we intend to answer the following questions: (1) How effective are existing DoS attacks? (2) Can the iGPU be used as an effective DoS attack vector? (3) Are Intel's CAT and GT COS mechanisms effective in preventing such DoS attacks?

This paper makes the following contributions:

- We provide extensive evaluation results on the effectiveness of various DoS attacks on two different Intel processor architectures with integrated GPUs, namely a Coffee Lake-based desktop processor and a newer Tiger

978-1-6654-0627-7/22/$31.00 ©2022 IEEE

Lake-based embedded application processor [1].

- We find that DoS attacks are generally much more effective on the Coffee Lake processor (up to 70X slowdown), while they are also effective, to a less degree, on the newer Tiger Lake processor (up to 12X slowdown).
- We find that Intel's hardware assisted LLC way partitioning techniques, namely CAT and GT COS, prevent unwanted LLC evictions but do not always translate to improved temporal isolation, especially in the case of iGPU-based DoS attacks, suggesting the limitations of these technologies. To the best of our knowledge, we are the first to evaluate the effectiveness of these features on a TigerLake embedded application processor.

The rest of this paper is organized as follows: Section II provides necessary background information on Intel iGPUs and RDT. Section III details the threat model we assume for this paper. Section IV describes the DoS attacks we used in this paper. Section V provides evaluation results of the effectiveness of DoS attacks on two Intel platforms and the isolation effects of Intel CAT and GT COS technologies. We discuss related work in SectionVI and conclude in Section VII.

II. Background

A. Integrated Graphic Processing Unit

In this paper, we focus on Intel's integrated GPU (iGPU) based computing platforms. The most basic computational node in Intel iGPUs is the *Execution Unit (EU)*, which is capable of executing seven concurrent threads. Each thread is composed of 128 general-purpose registers and two ALUs that can each execute up to eight 16-bit or four 32-bit operations. EUs are grouped into sets of *SubSlices*, each of which have 64KB of Shared Local Memory (SLM) that can be used to improve memory access times. Collectively, all subslices form a single *Slice* and share an L3 cache [17]. Figure 1 shows a high level depiction of the Slice architecture found in a Tiger Lake's Iris Xe Graphics iGPU. It is composed of 6 sub-slices and a total of 96 EUs, which enable a total of up to 5,376 concurrent 32-bit floating point operations. Intel iGPUs share access to a common shared LLC[2] with the CPU, so misses at the GPU's private L3 cache generate accesses to the LLC [19]. It is important to note that this differs from Nvidia's iGPUs, which share the DRAM with the CPU but not the LLC. In this paper, we are interested in studying the interference impacts of the LLC sharing between the CPU and the GPU.

B. Resource Directory Technology

To address the problem of shared resource contention, Intel has released a number of tools and technologies, collectively known as Intel Resource Directory Technology (RDT) [18], to monitor and control LLC space and DRAM bandwidth resources. For LLC management, the Cache Monitoring Tool (CMT) supports per-core LLC usage monitoring capability, and the **Cache Allocation Technology (CAT)** provides a way-based LLC partitioning capability among the CPU cores. For DRAM bandwidth management, Memory Bandwidth Monitoring (MBM) and Memory Bandwidth Allocation (MBA) support per-core DRAM bandwidth monitoring and throttling capabilities, respectively. More recently, Intel introduced a tool for controlling the LLC ways that their iGPUs can access, called **Graphics Technology Class of Service (GT COS)** [15], which works identically to the aforementioned CAT technology but for the iGPU—that is, it can limit the LLC ways used by the iGPU to reduce its impact to the CPU.

At a lower level, the RDT technologies all operate in a similar fashion. Rather than directly assign resource partitions directly to cores, Intel uses an additional layer of abstraction known as a Class-of-Service (CLOS) to which cores or groups of cores can be assigned. The resource partitions are then assigned on a per-CLOS basis. This allocation is done by assigning bitmasks to per-CLOS model-specific registers (MSRs). For example, the user can modify CAT specific MSRs to dictate which LLC ways the cores in that CLOS can access. Note that this same process applies to GT COS. In this work, we primarily focus on shared last-level cache management techniques, namely CAT and GT COS, and investigate whether these technologies are effective at mitigating interference on shared memory resources.

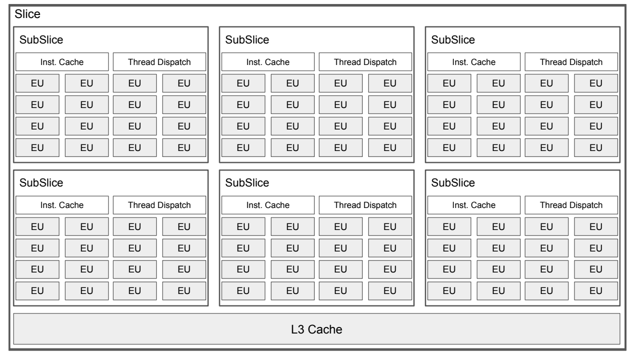


Fig. 1: High level overview of the Slice in a Tiger Lake Iris Xe Graphics iGPU. Adapted from [19].

III. Threat Model

We assume that a victim task and one or more attacker tasks are co-located on a heterogeneous multicore platform, which consists of multiple CPU cores and an integrated GPU. The CPU cores and the integrated GPU may have private per-core caches but they all share a common last-level cache (LLC) and the main memory (DRAM). We assume that the platform can provide hardware-level LLC way-partitioning (e.g., Intel CAT [18]), and that the runtime environment can partition CPU, iGPU and the LLC ways to minimize inference by ensuring these resources are not shared between the victim and the attacker tasks. We allow the attacker(s) to only execute unprivileged code on CPU cores and/or the iGPU if they are not used by the victim task. In this setting, the goal of the

[1]We provide our evaluation framework as open-source at https://github.com/mbechtel2/GTCOS-DoS.

[2]The shared LLC is level 4 from the GPU's perspective while it is level 3 from the CPU's perspective

attacker is to increase the victim's execution time, whether it runs on the CPU or iGPU, by running DoS attacks.

IV. Denial-of-Service Attacks on Shared Memory Resources

In this section, we present the denial-of-service (DoS) attacks we used in this work to stress the shared memory resources in a heterogeneous multicore platform.

A. CPU-based DoS Attacks

In this work, we use three CPU-based DoS attackers. They are specially engineered to stress shared hardware resources in the memory hierarchy in order to induce maximum delays to the victim task that shares the platform with the attackers.

```
for (i = 0; i<mem_size; i += LINE_SIZE)
{
    ptr[i] = 0xff;
}
```

(a) *BwWrite*

```
static int* list[MAX_MLP];
static int next[MAX_MLP];

for (int64_t i = 0; i < iter; i++) {
    switch (mlp) {
    case MAX_MLP:
    .
    .
    case 2:
        list[1][next[1]+1] = 0xff;
        next[1] = list[1][next[1]];
        /* fall-through */
    case 1:
        list[0][next[0]+1] = 0xff;
        next[0] = list[0][next[0]];
    }
}
```

(b) *MlpWrite*

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

The first two attackers are based on the *bandwidth* and *latency-mlp* from the IsolBench suite [37] and perform sequential and random memory accesses, respectively, with adjustable working set size (WSS) to fit in any level of the memory hierarchy. Furthermore, we configure them to only perform memory write operations, which are known to cause longer delays than reads [5], [6], [37]. We henceforth refer to these attackers as *BwWrite* and *MlpWrite*, respectively. Figure 2 shows the code listings for both attacks. Note that we choose the *latency-mlp* benchmark as it can better stress the memory hierarchy than the *latency* benchmark, which only performs a single memory access at a time. This is because *latency-mlp* can be configured to access multiple linked lists concurrently, while *latency* is limited to a single linked list. In addition, we use a **MAX_MLP** value of 12, which we found using the same method from [37].

```
#define bit(addr,x)  ((addr >> (x)) & 0x1)
int paddr_to_bank(unsigned long addr)
{
    return ((bit(addr, 6)^bit(addr,13))<<3|
            (bit(addr,14)^bit(addr,17))<<2|
            (bit(addr,15)^bit(addr,18))<<1|
            (bit(addr,16)^bit(addr,19)));
}
```

Fig. 3: DRAM bank mapping function for *MemWrite*, which performs bank-aware random memory updates.

The third CPU-based attacker is a memory-aware (DRAM bank-aware) DoS attack from [5]. This attack is identical to the MlpWrite attacker above, except that it only accesses a subset of array entries whose physical addresses are mapped to the same memory bank, thus creating bank contention. As such, we refer to this attacker as *MemWrite*. To achieve this attack, we first reverse engineer our Intel platforms' memory address mappings using the state-of-the-art DRAMA tool [11], [29] and use them to select the array entries that will be accessed. Figure 3 shows the code we use to convert a given physical address to a DRAM bank on the Coffee Lake platform we use in our evaluations.

B. GPU-based DoS Attacks

Since Intel iGPUs share the LLC and the DRAM with the CPU, we create a simple GPU kernel, which we call *GpuWrite*, using the OpenCL framework, that performs parallelized *memset* operations to stress both LLC and DRAM. We found the memset operation to be the most effective GPU kernel in generating contention on our tested platforms, which is consistent with the finding in [8]. Figure 4 shows the code listing of the *GpuWrite* kernel.

```
kernel void write(global TYPE * restrict a)
{
    const size_t i = get_global_id(0);
    a[i] = i;
}
```

Fig. 4: GPU DoS attack: *GpuWrite*

V. Evaluation

In this section, we investigate the impact of DoS attacks on two Intel platforms where the CPU and iGPU share and can simultaneously access the LLC.

A. Hardware Platforms

For our experiments, we use two Intel-based heterogeneous hardware platforms: a PC equipped with an Intel i7-8700K (Coffee Lake) processor and an Intel UP3 embedded platform equipped with an Intel i7-1185G7E (Tiger Lake) processor. The i7-8700K includes six physical cores and twelve hardware threads, while the i7-1185G7E includes four physical cores and eight hardware threads. On both processors, each core has private L1 and L2 caches and all cores share an L3 cache that is also the LLC. Both processors equip iGPUs and the CPU and iGPU on both platforms share the LLC and the main memory subsystem. On the software side, both systems run Ubuntu 20.04 as their operating system. However, the Coffee Lake system runs Linux v5.13 and the Tiger Lake UP3 runs Linux v5.8.

Platform	Coffee Lake PC	Tiger Lake UP3
CPU (Shared)	Intel i7-8700K 6 cores / 12 threads 3.7GHz base 4.7GHz boost 32KB(I/D) L1 256KB L2 (4-way) 12MB L3 (16-way)	Intel i7-1175G7E 4 cores / 8 threads 2.8GHz base 4.4GHz boost 48KB(I)/32KB(D) L1 1280KB L2 (16-way) 12MB L3 (12-way)
iGPU	Intel UHD 630 24 EUs 350MHz base 1.2GHz boost	Intel Iris Xe 96 EUs 350MHz base 1.35GHz boost
Memory	8GB (1 DIMM) 1 rank, 16 banks	8GB (1 DIMM) 1 rank, 16 banks
Bank functions	6⊕13, 14⊕17, 15⊕18, 16⊕19	N/A
RDT	Not enabled	L3 CAT GT COS

TABLE I: Intel Platform System Specs

The Tiger Lake UP3 platform supports Intel RDT technologies, namely CAT and GT COS, which provide hardware supported LLC way-partitioning between the CPU cores (CAT) as well as between the CPU and iGPU (GT COS) (see Section II-B). Unfortunately, however, the MSR registers used for controlling GT COS allocations are not publicly available. As such, we reverse engineer these MSRs (see Appendix A for details). Note that the LLC on the TigerLake platform has 12 ways with each way being 1MB in size. As such, both CAT and GT COS MSRs can be assigned anywhere from 1 to 12 ways.

We disable turbo boost and simultaneous multithreading (SMT) and always pin tasks, both attackers and the victim, on dedicated cores to improve the repeatably of the results.

From this point, we refer to the platforms as Coffee Lake and Tiger Lake, respectively, based on their CPU architectures. Additional platform specifications can be found in Table I.

B. *Effect of DoS Attacks*

In the first set of experiments, we evaluate the effect of DoS attacks on a CPU victim task. The basic experiment setup is as follows: We run a victim task alone on CPU Core 0, first in isolation and then together with DoS attackers on all other available cores on the CPU or the iGPU. In each case, we measure the performance of the victim and calculate the slowdown ratios caused by the attackers.

1) Synthetic CPU victim: In this experiment, we use *bandwidth* from the IsoBench suite as a synthetic victim task. We configure the task to perform read accesses, vary its WSS from 2MB to 64MB, and use HugePage allocation to reduce LLC set conflicts. We henceforce refer to the task as *BwRead*. For the DoS attackers, we use all three CPU-based attacks (*BwWrite*, *MlpWrite*, *MemWrite*) and the GPU-based attack (*GpuWrite*) described in Section IV on the Coffee Lake platform while we omit MemWrite on Tiger Lake as we were unable to determine its full memory address mapping scheme.

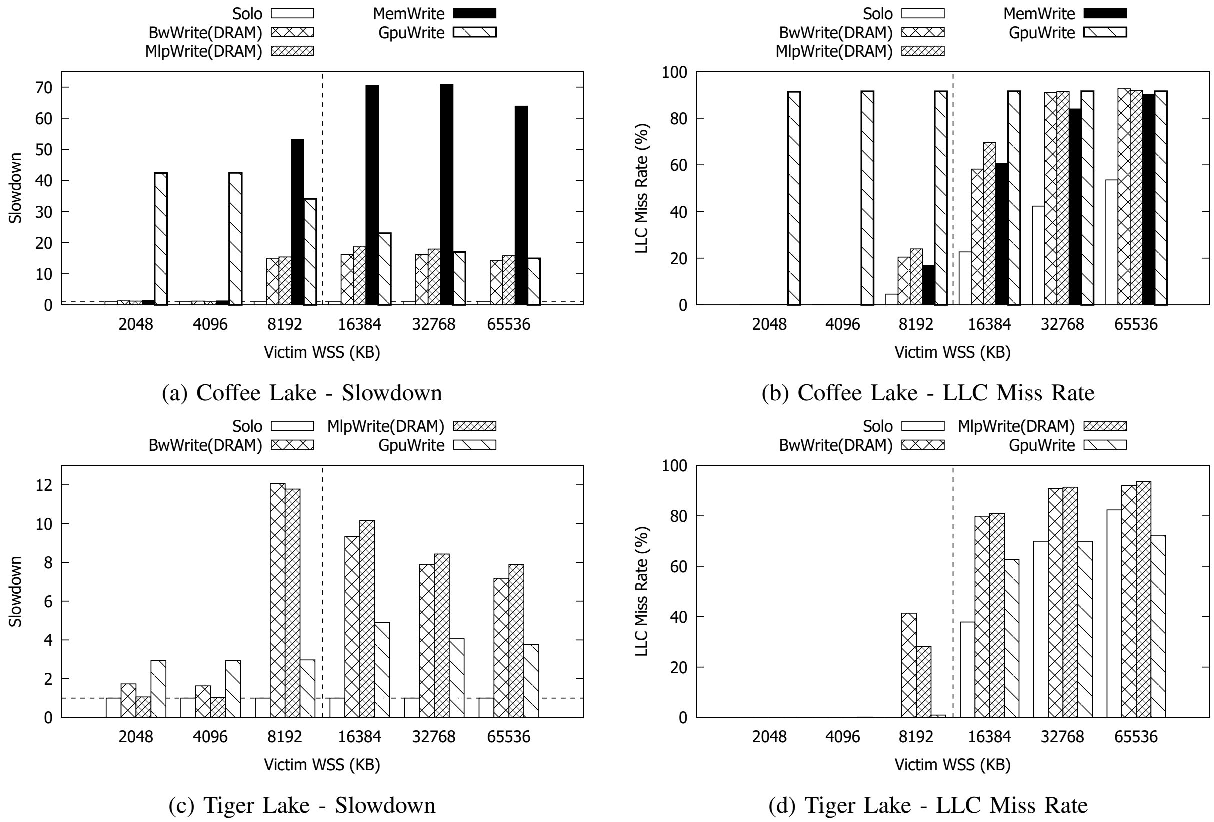
Figure 5 shows the results for the BwRead victim. On Coffee Lake, when the victim is 4MB or smaller, the GPU-based attack, GpuWrite, is the most effective, achieving up to ∼42X slowdown at 4MB WSS. On the other hand, the memory-aware attack, MemWrite, is the most effective when the victim's WSS is 8MB or larger, up to ∼71X slowdown at 16MB WSS, which is due to the additional impacts of memory bank contention (e.g. prolonged LLC blocking times). Interestingly, all CPU based attacks—BwWrite, MlpWrite, and MemWrite—are ineffective when the victim's WSS is less than 4MB, in which cases the victim experiences close to zero LLC miss rates, despite the fact that the LLC is fully shared by both the victim and the DoS attacker. On the other hand, the GPU attacker, GpuWrite, is able to effectively evict the victim's cache-lines, regardless of the victim's WSS, as evidenced by the victim's high LLC miss rates, around ∼90%, in Figure 5b. These results suggest that the GPU's memory accesses may be prioritized over the CPU's memory accesses on Coffee Lake.

On Tiger Lake, on the other hand, the first general observation is that all DoS attacks are relatively less effective than on Coffee Lake. Again, when the victim's WSS is small, less than 4MB, GpuWrite is more effective as on Coffee Lake, although the degree of which is much smaller. As the victim's WSS increases, however, the CPU based attacks, both BwWrite and MlpWrite, become more effective as they cause up to ∼12.1X slowdown, while the GPU-based attack, GpuWrite, only causes ∼4.1X in the worst case.

Lastly, on both platforms, we find that attacker's memory patterns, sequential or random, do not have significant impact to the victim. We also varied the victim's memory access pattern and found marginal differences in terms of observed slowdowns and LLC miss rates on both platforms. This is somewhat surprising as prior work on interference in NVIDIA Jetson platforms [7] and memory performance attacks [24] suggested significant impact depending on the memory access patterns. However, we did not observe such noticeable differences on the two Intel platforms we tested and thus omit the results due to space limitation.

2) SPEC2017 victim on CPU: We next evaluate the effectiveness of the DoS attacks on 23 real-world benchmarks from the SPEC2017 suite [35]. The basic experiment setup is the same as the prior experiment, except that we use one of the SPEC2017 benchmark as the victim.

Figure 6 shows the results. Note that the benchmarks are organized in ascending order of their memory bandwidth usages measured in isolation on their evaluated platforms. On both platforms, the slowdowns experienced by the SPEC2017 benchmarks generally increase along with their DRAM bandwidth usage, up to ∼32.3X against the *roms* benchmark on Coffee Lake and ∼7.6X against the *omnetpp* benchmark on Tiger Lake. This is likely because DRAM contention also becomes a bottleneck, in addition to LLC contention, so there



(a) Coffee Lake - Slowdown

(b) Coffee Lake - LLC Miss Rate

(c) Tiger Lake - Slowdown

(d) Tiger Lake - LLC Miss Rate

Fig. 5: Impact of DoS attackers on the execution time and LLC miss rate of BwRead victim tasks. Sizes left of the vertical line are LLC-fitting and the sizes to the right are DRAM-fitting.

are more avenues for the attackers to generate slowdown. We still see slowdown for the LLC sensitive benchmarks on both platforms, but to a far lesser degree. On Coffee Lake, the memory-aware DoS attack, MemWrite, is highly effective at delaying the victims, despite the memory-aware attackers consuming less memory bandwidth than memory-unaware attackers, suggesting that contention is not necessarily in overall DRAM bandwidth only. Instead, the memory-aware attack is more effective because it targets a specific DRAM bank, which causes excessive bank conflicts and thus slows down all memory performance. The GPU attacker, GpuWrite, is also very effective, especially compared to the memory-unaware CPU DoS attacks, with a geometric mean slowdown of 5X across all tested SPEC2017 victims. On Tiger Lake, on the other hand, the memory-unaware CPU attackers are generally more effective than the GPU attacker, which is expected from the prior synthetic experiment results.

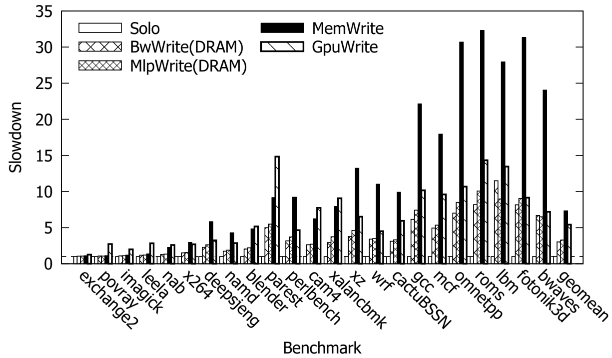
3) Parboil victim on iGPU: To this point, all of our victims have run on the CPU. However, victim kernels run on the iGPU may also be susceptible to DoS attacks. To investigate this, we employ 11 real-world GPU benchmarks from the Parboil suite [36] and perform the same DoS experiments as before. In this case, though, we only use CPU-based DoS attacks as the GPU-based attack cannot run at the same time as the Parboil kernels.

Figure 7 shows the Parboil benchmark results on Coffee Lake and Tiger Lake. In general, we observe that GPU kernel victims are less susceptible to DoS attacks from the CPU on both platforms. However, we find that memory-aware CPU DoS attacks are still noticeably effective on Coffee Lake. In the worst case, we see ∼10X and ∼2.6X slowdown against the *lbm* benchmark on Coffee Lake and Tiger Lake, respectively. From this, we find that DoS attacks do have the potential to negatively affect real-world applications on the iGPU.

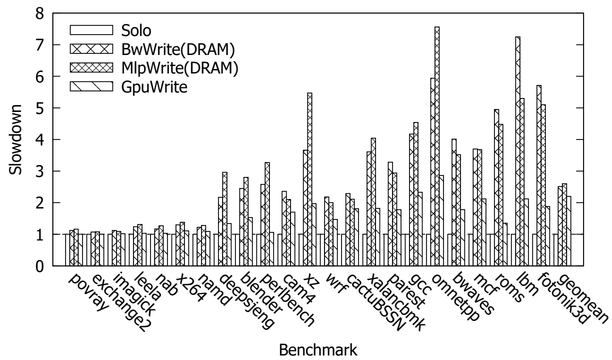
C. Isolation Effect of Intel CAT/GT COS

In this section, we seek to determine if hardware-based LLC partitioning on Intel platforms, namely CAT and GT COS technologies (see Section II-B) can help mitigate the impacts of DoS attacks. To do this, we first rerun the BwRead victim tests on the Tiger Lake platform, but with CAT and GT COS partitioning enabled. For the partition allocations, we assign the victim core to its own CLOS (CLOS 0) and the attacker cores to another CLOS (CLOS 1). We then assign all but one of the LLC ways to the victim and the last way to all attackers, including the GPU-based attacker. In other words, we give the victim sole access to 11MB (11 ways) of the LLC while the attackers collectively only have 1MB.

Figure 8 shows the results with partitioning enabled. As expected, when the victim fits in their given LLC space we see virtually no LLC misses. In particular, we highlight the

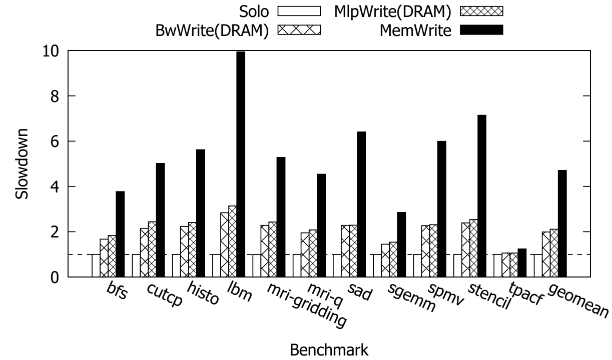


(a) Coffee Lake

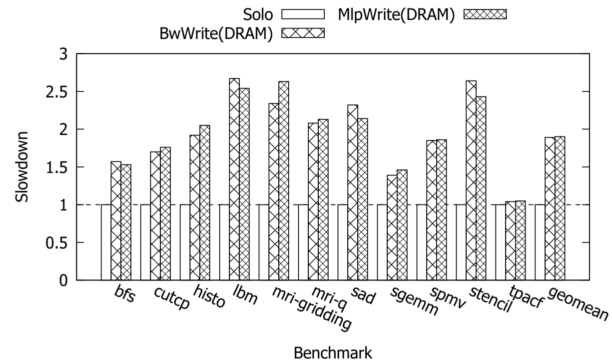


(b) Tiger Lake

Fig. 6: Impact of DoS attacks on SPEC2017.



(a) Coffee Lake



(b) Tiger Lake

Fig. 7: Impact of DoS attacks on Parboil kernel performance.

8MB victim case as we see a notable reduction in its LLC miss rate with partitioning enabled. To be more specific, the 8MB victim goes from a ∼41% worst case miss rate without partitioning down to 0% with partitioning. From this, we find that both CAT and GT COS work as intended as they do provide spatial isolation for the LLC. Furthermore, we find that this also translates to improved temporal isolation against CPU-based attackers. For example, the 8MB victim goes from ∼12.1X slowdown against the MlpWrite attackers down to 9% with partitioning. On the other hand, we find that LLC partitioning does not provide effective temporal isolation against GPU-based DoS attacks as the victim still sees ∼3X slowdown regardless. From this, we find that (1) slowdowns from CPU-based attacks are primarily due to LLC evictions, and (2) the slowdown from GPU-based attacks are not due to LLC evictions and are more likely due to other factors (e.g. LLC blocking [6], [37]), thus rendering LLC partitioning ineffective.

We also test how well LLC space partitioning can protect the performance of the SPEC2017 and Parboil benchmarks. We run the same experiments as in Figures 6b and 7b, but with CAT and GT COS enabled. Note that for the Parboil benchmarks, we change the GT COS allocation such that the iGPU, and hence the victim kernel, has access to 11 LLC ways separate from the attackers' single way.

Figure 9 shows the SPEC2017 benchmarks' performance on Tiger Lake. As we expected, enabling LLC partitioning did help to improve the performance of the victim tasks but did not provide complete temporal isolation. Collectively, the benchmarks went from a worst case geometric mean of ∼2.6X slowdown without partitioning down to ∼2X. However, we note that the performance improvements were mostly seen when the attackers were CPU-based. Similar to the synthetic victims, LLC partitioning had little impact against GPU-based attackers as the benchmarks only improved from a geometric mean of 50% slowdown down to 43% slowdown.

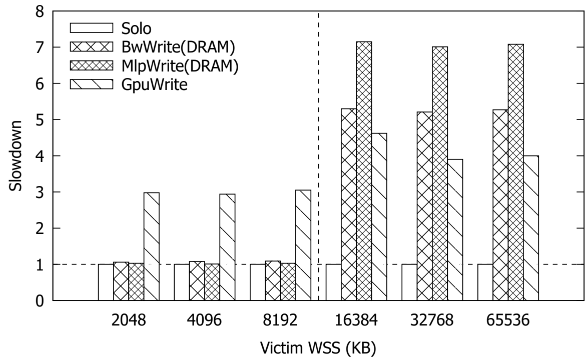
Figure 10 then shows the results for the Parboil benchmarks. We again see that LLC partitioning helps to improve temporal performance through spatial isolation, but does not completely protect real-time performance. On average, the benchmarks go from a worst case geometric mean of 90% without partitioning to 49% with partitioning.

From this, we find that additional mechanisms should be used in conjunction with LLC partitioning in order to better guarantee system predictability.

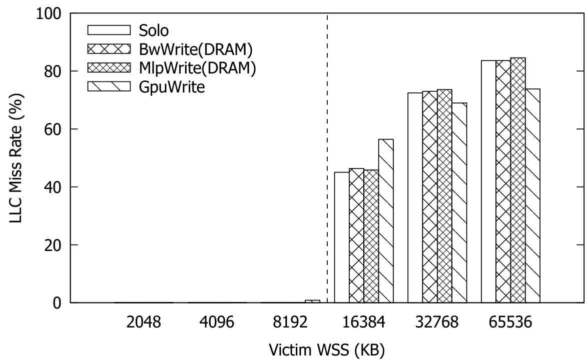
VI. Related Work

A. GPUs in Real-Time Systems

Because GPUs offer significant performance improvements for highly data-parallel applications, much research effort has been focused on the predictable use of GPUs for real-time applications. Unlike CPU tasks, GPU kernels are often non-preemptable or incur high preemption overhead, which can



(a) Slowdown



(b) LLC Miss Rates

Fig. 8: Impact of Intel CAT/GT COS way-based partitioning in protecting a BwRead CPU victim. (The sizes left to the bar are LLC-fitting and the sizes to the right are DRAM-fitting.)

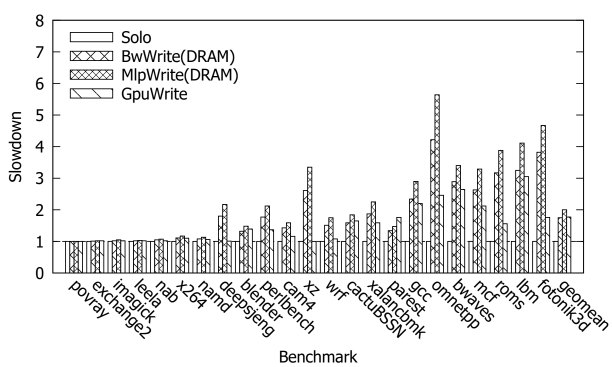


Fig. 9: Impact of Intel CAT/GT COS way-based partitioning in protecting SPEC2017 CPU benchmarks.

cause undesired priority inversions. To address this, there have been many efforts to account for GPU execution in WCET estimations and real-time scheduling decisions. These include kernel slicing [38], [42], [46], spatial and temporal partitioning [2], [32]. Much of these efforts have focused on NVIDIA's discrete GPUs due to their popularity and maturity, though some recent works also investigated the use of AMD's discrete GPUs for real-time systems [26], [27].

Recently, some researchers have investigated the use of

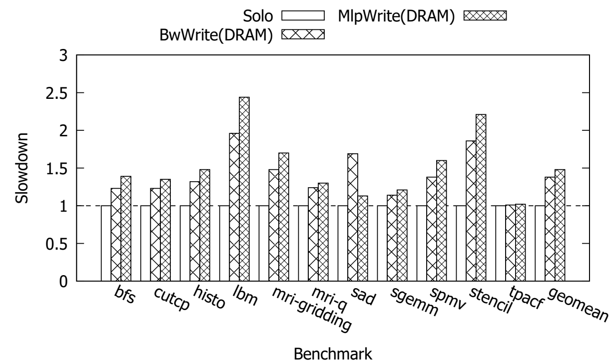


Fig. 10: Impact of Intel CAT/GT COS way-based partitioning in protecting Parboil iGPU benchmarks.

integrated CPU-GPU platforms for real-time applications [1], [4], [7], [8], [25], [28], [41]. Compared to discrete GPUs, integrated GPUs are more challenging to provide real-time guarantees because they share the main memory and other resources with the CPUs. Closely related to this work, Cavicchioli et al. evaluated the impacts of shared resource contention on Nvidia Jetson embedded platforms and an Intel Skylake micro-architecture based PC [7], [8]. However, their work only used synthetic tasks and did not evaluate the impact of shared resource isolation techniques. In contrast, our work evaluated the impact of real-world applications and the effect (and the limitations) of Intel's recent hardware assisted isolation mechanisms.

B. Denial-of-Service Attacks

Many denial-of-service attacks on shared resources have been studied on multicore systems. Moscibroda et al. demonstrated the feasibility of DoS attacks on DRAM [24]. These attacks exploit the characteristics of memory controller's FR-FCFS scheduling algorithm [30] that favors sequential access over random ones. More recently, DoS attacks that target internal hardware structures of the shared LLC to induce cache blocking have been proposed [5], [6], [37]. Leveraging the attacks from [6], Iorga et al. presented a statistical testing method to evaluate shared resource interference on a number of embedded multicore platforms [20].

Even without malicious attackers, normal applications can cause interference with one another. Therefore, there has been much work in the real-time community to provide stronger isolation for shared LLC and DRAM on multicore systems. Many researchers have proposed various software and hardware mechanisms and policies to manage these resources [10], [21]–[23], [31], [33], [34], [40], [43], [45], [47]. Common software based techniques include page coloring based cache or DRAM bank partitioning [21]–[23], [43], [44] and performance counter based bandwidth throttling [45]. The move for greater shared resource management has also been seen in industry, such as Intel's introduction of the CAT and CMT technologies [14]. ARM also introduced a similar technology called Memory System Resource Partitioning and Monitoring

(MPAM) [3] in addition to their existing hardware QoS mechanisms [33]. Recent works leveraged these hardware features to provide stronger real-time guarantees. Xu et al. proposed a joint shared cache space and memory bandwidth partitioning technique to provide stronger isolation in multicore [39], [40] utilizing Intel CAT and MemGuard [45]. Gifford et al. have since extended this technique to introduce task phases and perform phase-aware resource allocations [12]. Serrano-Cases et al. and Zini et al. have explored the use of ARM QoS mechanism in Xilinx MPSoCs to regulate the contention at the system bus level [33], [47].

Our work is different from prior DoS studies in that we focus on recent Intel processors with integrated GPU to explore the feasibility of iGPU based DoS attacks, and evaluate the effectiveness of Intel's new hardware feature that supports cache way partitioning for iGPU.

VII. Conclusion

In this paper, we investigated the potential for DoS attacks on Intel CPUs with integrated GPU (iGPU) in which not only the DRAM but also the LLC are shared between the CPU and the iGPU. From extensive experiments, we find that both computational nodes could effectively impact the performance of the other for both synthetic and real-world representative victim tasks. We then evaluated the CAT and GT COS technologies introduced by Intel for providing CPU and iGPU LLC way partitioning. In doing so, we find that both are able to improve spatial isolation and reduce LLC evictions but are unable to provide temporal isolation in all cases, suggesting the presence of additional sources of interference that delay the execution of the affected victims. As future work, we plan to investigate more sophisticated iGPU-based DoS attacks that take advantage of memory bank mapping and other available system information. We also plan to investigate hardware-based bandwidth throttling mechanisms for Intel iGPUs.

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. The Tiger Lake UP3 platform used for this research was donated by the Intel Corporation. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF, NSA or Intel.

References

[1] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith. GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed. In *RTSS*, 2017.

[2] T. Amert, Z. Tong, S. Voronov, J. Bakita, F. D. Smith, and J. H. Anderson. Timewall: Enabling Time Partitioning for Real-Time Multicore+ Accelerator Platforms. In *RTSS*, 2021.

[3] ARM. ARM Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A. https://developer.arm.com/documentation/ddi0598/latest.

[4] S. Bateni, Z. Wang, Y. Zhu, Y. Hu, and C. Liu. Co-Optimizing Performance and Memory Footprint Via Integrated CPU/GPU Memory Management, an Implementation on Autonomous Driving Platform. In *RTAS*, 2020.

[5] M. Bechtel and H. Yun. Memory-Aware Denial-of-Service Attacks on Shared Cache in Multicore Real-Time Systems. *ToC*, 2021.

[6] M. G. Bechtel and H. Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. In *RTAS*, 2019.

[7] N. Capodieci, R. Cavicchioli, I. S. Olmedo, M. Solieri, and M. Bertogna. Contending Memory in Heterogeneous SoCs: Evolution in NVIDIA Tegra Embedded Platforms. In *RTCSA*, 2020.

[8] R. Cavicchioli, N. Capodieci, and M. Bertogna. Memory Interference Characterization Between CPU Cores and Integrated GPUs in Mixed-Criticality Platforms. In *ETFA*, 2017.

[9] I. Cutress. AMD Expanding Into Tesla Model 3 and Model Y. https://www.anandtech.com/show/17198/amd-expanding-into-tesla-model-3-and-model-y.

[10] F. Farshchi, P. K. Valsan, R. Mancuso, and H. Yun. Deterministic Memory Abstraction and Supporting Multicore System Architecture. In *ECRTS*, 2018.

[11] P. Frigo, E. Vannacc, H. Hassan, V. Van Der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi. TRRespass: Exploiting the Many Sides of Target Row Refresh. In *IEEE SP*, 2020.

[12] R. Gifford, N. Gandhi, L. T. X. Phan, and A. Haeberlen. DNA: Dynamic Resource Allocation for Soft Real-Time Multicore Systems. In *RTAS*, 2021.

[13] A. Hamann. Industrial Challenges: Moving From Classical to High Performance Real-Time Systems. In *WATERS*, July 2018.

[14] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, and R. Iyer. Cache QoS: From Concept to Reality in the Intel® Xeon® Processor E5-2600 v3 Product Family. In *HPCA*, 2016.

[15] Intel.

[16] Intel. Hard Iron Meets Artificial Intelligence. https://www.intel.com/content/www/us/en/customer-spotlight/stories/audi-automated-factory.html.

[17] Intel. Intel® Processors with Integrated Graphics. https://www.intel.com/content/www/us/en/develop/documentation/oneapi-gpu-optimization-guide/top/gen-arch.html.

[18] Intel. Intel® Resource Director Technology (Intel® RDT) Framework. https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html.

[19] Intel. Software Optimization for Intel® GPUs (NEW). https://www.intel.com/content/www/us/en/develop/documentation/vtune-cookbook/top/methodologies/software-optimization-for-intel-gpus.html.

[20] D. Iorga, T. Sorensen, J. Wickerson, and A. F. Donaldson. Slow and Steady: Measuring and Tuning Multicore Interference. In *RTAS*, 2020.

[21] 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.

[22] 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.

[23] 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.

[24] T. Moscibroda and O. Mutlu. Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems. In *USENIX Security Symposium*, 2007.

[25] I. S. Olmedo, N. Capodieci, J. L. Martinez, A. Marongiu, and M. Bertogna. Dissecting the CUDA Scheduling Hierarchy: a Performance and Predictability Perspective. In *RTAS*, 2020.

[26] N. Otterness and J. H. Anderson. AMD GPUs as an Alternative to NVIDIA for Supporting Real-Time Workloads. In *ECRTS*, 2020.

[27] N. Otterness and J. H. Anderson. Exploring AMD GPU Scheduling Details by Experimenting With "Worst Practices". In *RTNS*, 2021.

[28] N. Otterness, M. Yang, S. Rust, E. Park, J. H. Anderson, F. D. Smith, A. Berg, and S. Wang. An Evaluation of the NVIDIA TX1 for Supporting Real-Time Computer-Vision Workloads. In *RTAS*, 2017.

[29] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In *USENIX Security Symposium*, 2016.

[30] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. Owens. Memory Access Scheduling. In *ACM SIGARCH Computer Architecture News*, 2000.

[31] S. Roozkhosh and R. Mancuso. The Potential of Programmable Logic in the Middle: Cache Bleaching. In *RTAS*, 2020.

[32] S. K. Saha, Y. Xiang, and H. Kim. STGM: Spatio-Temporal GPU Management for Real-Time Tasks. In *RTCSA*, 2019.
[33] A. Serrano-Cases, J. M. Reina, J. Abella, E. Mezzetti, and F. J. Cazorla. Leveraging Hardware QoS to Control Contention in the Xilinx Zynq UltraScale+ MPSoC. In *ECRTS*, 2021.
[34] P. Sohal, R. Tabish, U. Drepper, and R. Mancuso. E-WarP: a System-wide Framework for Memory Bandwidth Profiling and Management. In *RTSS*, 2020.
[35] SPEC CPU2017. https://www.spec.org/cpu2017.
[36] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu. Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing. *Center for Reliable and High-Performance Computing*, 2012.
[37] P. K. Valsan, H. Yun, and F. Farshchi. Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems. In *RTAS*, 2016.
[38] Y. Xiang and H. Kim. Pipelined Data-Parallel CPU/GPU Scheduling for Multi-DNN Real-Time Inference. In *RTSS*, 2019.
[39] M. Xu, R. Gifford, and L. T. X. Phan. Holistic Multi-Resource Allocation for Multicore Real-Time Virtualization. In *DAC*, 2019.
[40] 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.
[41] M. Yang, N. Otterness, T. Amert, J. Bakita, J. H. Anderson, and F. D. Smith. Avoiding Pitfalls When Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems. In *ECRTS*, 2018.
[42] M. Yang, S. Wang, J. Bakita, T. Vu, F. D. Smith, J. H. Anderson, and J.-M. Frahm. Re-thinking CNN Frameworks for Time-Sensitive Autonomous-Driving Applications: Addressing an Industrial Challenge. In *RTAS*, 2019.
[43] Y. Ye, R. West, Z. Cheng, and Y. Li. Coloris: a Dynamic Cache Partitioning System Using Page Coloring. In *PACT*, 2014.
[44] 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.
[45] 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.
[46] H. Zhou, S. Bateni, and C. Liu. Sˆ3DNN: Supervised Streaming and Scheduling for GPU-Accelerated Real-Time DNN Workloads. In *RTAS*, 2018.
[47] M. Zini, G. Cicero, D. Casini, and A. Biondi. Profiling and Controlling I/O-Related Memory Contention in COTS Heterogeneous Platforms. *Software: Practice and Experience*, 2021.

APPENDIX

A. Reverse Engineering the GT COS Model Specific Register

As previously discussed, the GT COS technology operates in a manner similar to Intel's standard CAT solution. The GT COS in Tiger Lake is controlled by four MSR registers, which support four logical class-of-service (CLOS) partitions for integrated GPU. Since the addresses of GT COS MSR registers are not publicly available, we reverse engineer the MSRs as follows. First, we learned that the values of all four GT COS MSRs are changed to preset values when enabling or disabling relevant settings in the BIOS, according to [15]. With these known sets of values, we then searched all available MSR registers that match with the known values. In this way, we determined that MSR registers *0x18b0-0x18b3* are associated with GT COS's CLOS 0-3 respectively.

To validate these MSRs, we performed an experiment using a BwRead(LLC) victim task on the CPU alongside a GpuWrite(DRAM) attacker on the iGPU with and without LLC partitioning. For GT COS, we set the victim to have a WSS of 9.5MB and assign 11MB of the LLC (i.e., 11 ways) to the victim's partition, while the remaining 1MB being given to the iGPU's LLC partition using the identified GT COS MSRs.

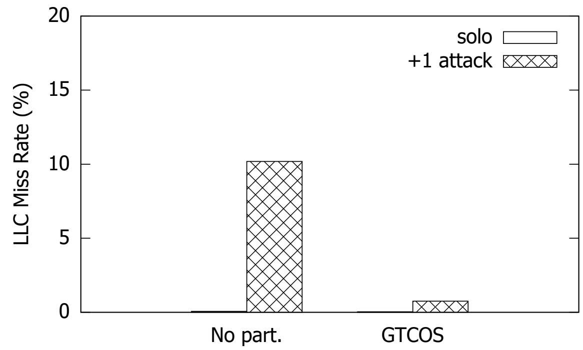


Fig. 11: Impact of GTCOS partitioning on the LLC miss rate of a BwRead victim task.

Figure 11 shows the LLC miss rates of the victim tasks in both scenarios. Without partitioning, we do see a notable increase in the victim's miss rate, ∼10%, indicating that the iGPU on Tiger Lake can evict LLC lines when the victim task is sufficiently large. When we enable GT COS partitioning, we see a meaningful drop in the LLC miss rate, down to ∼0.7%. From the results, we conclude that (1) the MSRs found in our reverse engineering are valid, and (2) GT-COS works as intended in providing spatial isolation between the CPU and the iGPU in accessing the shared LLC.