Performance Analysis of Scientific Computing Workloads on General Purpose TEEs

Ayaz Akram
UC Davis
yazakram@ucdavis.edu

Anna Giannakou
LBNL
agiannakou@lbl.gov

Venkatesh Akella
UC Davis
akella@ucdavis.edu

Jason Lowe-Power
UC Davis
jlowepower@ucdavis.edu

Sean Peisert
LBNL & UC Davis
sppeisert@lbl.gov

***Abstract*—Scientific computing sometimes involves computation on sensitive data. Depending on the data and the execution environment, the HPC (high-performance computing) user or data provider may require confidentiality and/or integrity guarantees. To study the applicability of hardware-based trusted execution environments (TEEs) to enable secure scientific computing, we deeply analyze the performance impact of general purpose TEEs, AMD SEV, and Intel SGX, for diverse HPC benchmarks including traditional scientific computing, machine learning, graph analytics, and emerging scientific computing workloads. We observe three main findings: 1) SEV requires careful memory placement on large scale NUMA machines (1×–3.4× slowdown without and 1×–1.15× slowdown with NUMA aware placement), 2) virtualization—a prerequisite for SEV—results in performance degradation for workloads with irregular memory accesses and large working sets (1×–4× slowdown compared to native execution for graph applications) and 3) SGX is inappropriate for HPC given its limited secure memory size and inflexible programming model (1.2×–126× slowdown over unsecure execution). Finally, we discuss forthcoming new TEE designs and their potential impact on scientific computing.**

***Index Terms*—HPC, Hardware Security, TEE, SGX, SEV.**

I. Introduction and Background

High performance computing (HPC) operators provide the computing hardware, storage, and networking infrastructure to help researchers solve large-scale computational problems. Increasingly, scientific computing involves data analysis, rather than traditional modeling and simulation. The data being analyzed is often provided by third parties to individual researchers or the HPC center for use by multiple users. The data may be fully open, or may be in some way sensitive e.g., subject to government regulations, considered proprietary, or pertain to public privacy concerns. This paper analyzes the performance impact of protecting this data with hardware-based trusted execution environment (TEE) on AMD and Intel based systems, one aspect of a larger secure system design [1].

The use of sensitive data at HPC centers raises security concerns from both external threats (the user of compute resources and the data provider might not trust other users sharing the compute resources) and internal threats (the user and the data provider might not trust the compute provider). Data owners (or compute providers) have to make a tradeoff between accepting the risk of hosting (or providing compute for) the sensitive data and declining the service. Currently, where secure HPC enclaves exist, such as those currently being used for analysis of sensitive health-related data pertaining to COVID-19 at universities and research institutes around the world, trust required by data providers of the compute facility, and liability to an organization for hosting such data are both very high. Moreover, traditionally, the environments created to protect sensitive data have significant usability challenges. For example, processing sensitive health data requires dramatically different environments compared to those typically used in NSF high-performance computing facilities. Processing capabilities are limited to only a handful of racks and access requires virtual private networks (VPNs) and/or remote desktops. These onerous usability requirements are particularly cumbersome for the scientific community which is mostly used to working in very open, collaborative, and distributed environments.

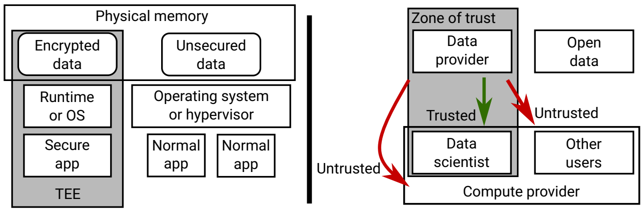


Fig. 1: Trusted execution environments and our threat model

The use of computing hardware supported by trusted execution environments (TEEs), which provide confidentiality and/or integrity guarantees for the sensitive data and compute, can act as a first step to enable secure scientific computing without severely impacting the usability of HPC systems by providing equivalent or greater layers of protection with fewer steps for the end-user. Data owners (or compute providers) are not required to accept a tradeoff of increased risk and usability (or openness) in a TEE-based secure computing environment.

Figure 1 shows how TEEs fit into the system architecture. TEEs can be used to isolate data from other users and the operating system or hypervisor that is controlled by the compute provider. Examples include Intel's SGX (Software Guard Extension) [2], ARM's TrustZone [3], AMD's SEV (Secure Encrypted Virtualization) [4], and emerging research platforms like RISC-V's Keystone [5]. In this paper, we investigate the performance of traditional and emerging HPC applications on TEEs based computing hardware. Researchers have evaluated the performance of TEEs in the context of client and cloud computing [6], [7]. However, to the best of our knowledge, a detailed analysis of performance overhead of TEEs on realistic HPC workloads has not been conducted in the past. The goal of this paper is to fill that gap. More specifically, we evaluate the performance of AMD's SEV and Intel's SGX platforms for a range of different HPC applications including traditional scientific computing (via the NAS Parallel Benchmarks [8]), graph processing (via the GAP Benchmark Suite [9]), and emerging potentially sensitive workloads, genetic analysis (BLAST [10]), traffic simulation (Mobiliti [11]), hydrodynamics stencil calculation (LULESH [12]), and a particle transport simulation (Kripke [13]) on realistic datasets.

The main contributions of this work are in the form of three major findings:

1) *SEV can be used for secure scientific computing without significant performance degradation for most workloads if it is configured correctly.* We found that when using the default configuration of SEV, applications suffer some performance degradation (up to 3.4×) from poor NUMA data placement as all data resides on a single NUMA node. However, most of this performance loss can be mitigated by simply configuring the system to use NUMA interleaving.

2) *Irregular workloads performance suffers due to virtualization when running under SEV.* SEV requires running applications in a virtualized environment (e.g., QEMU), which causes performance loss for highly-irregular workloads such as graph analytics and workloads with many I/O operations. Future architectures and systems should concentrate on reducing this overhead to improve the performance of HPC applications in secure environments.

3) *SGX is inappropriate for scientific computing.* We found that running unmodified HPC applications under SGX results in a significant performance degradation. We observed slowdowns from 1.2× to 126× compared to unsecure execution. This slowdown is mainly due to the limited secure memory capacity (about 100 MB) and the fact that HPC applications have large working sets and are difficult to partition into secure and normal components. Secondly, the HPC workloads under SGX exhibit poor multithreaded scaling (an important characteristic of HPC workloads). Furthermore, we found that the support for application development with SGX is also limited, which makes it inadequate (or at least unsuitable) for HPC environments, where domain scientists often rely on unmodified off-the-shelf third-party libraries such as Tensorflow.

II. Threat model

We assume that HPC system administrators are not trusted and that host operating systems and hypervisors are not trusted. However, the guest operating system of a virtual machine which is owned by the user is trusted. We assume very simple physical attacks are within scope, but that physical attacks that are more time consuming, such as opening a rack-mount HPC system and removing chips soldered on the board, are less important at this time because there are other means, such as video cameras pointed at the HPC systems, to monitor and mitigate such attacks. We assume HPC users themselves are trusted to not exfiltrate their own data, though we do not trust them to not attack others. Also, we focus on general-purpose computing hardware—FPGAs, GPUs, dedicated ASICs are not considered in this paper, mainly because no commercial TEEs yet exist for these hardware accelerators.

We assume that data providers trust the data users or that some other means (e.g., differential privacy) will ensure the sensitive data is not improperly exfiltrated by the scientific application developers and users. Figure 1 shows how TEEs fit into this threat model.

III. Background and Prior Work

A. Trusted Execution Environments

Trusted Execution Environments in hardware, at minimum, provide some degree of hardware-enforced separation from other users and processes, and the ability of end users to verify through cryptographic attestation that execution is taking place within the TEE. Some TEEs, including Intel's Software Guard Extensions (SGX) and AMD's Secure Encrypted Virtualization (SEV), also support encrypted memory. Both SGX and SEV protect against malicious system administrators and host operating systems. TEEs have their roots in earlier cryptographic hardware functions, including Trusted Platform Modules. In this work, we analyze the performance of AMD SEV [4] and Intel SGX [2]. We exclude the other major commercially available option ARM TrustZone [3] from this study as existing TrustZone based TEEs mainly target embedded and mobile devices, not general purpose compute devices.

B. Prior studies of TEE performance

There exist many prior studies on the performance impact of TEEs which either focus on a single TEE or a comparison of different TEEs [14]–[17]. These studies found that while the SGX overhead can be significant it can be mitigated for *small or partition-able workloads* and that the SEV overhead is negligible for small workloads. However, these studies are different from our work in three fundamental ways: (1) none of these works focus on scientific/high-performance computing, (2) they target micro-benchmarks or small benchmarks which are usually single-threaded, (3) with reference to SEV, none of these works evaluate large machines with multiple NUMA nodes which are common in HPC environments.

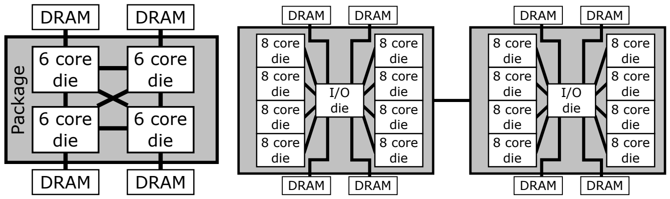
There has also been prior work focused on improving the software and hardware architecture of SGX to improve performance [6], [18], [19]. We do not evaluate these research proposals in this paper. Instead, we focus on the currently available hardware technologies.

IV. Methodology

We picked traditional scientific computing workloads as well as modern applications which fit the criteria of HPC application domain.

A. Traditional HPC Benchmarks/Kernels (NPB)

We evaluate workloads traditionally used to benchmark HPC systems i.e. NAS Parallel Benchmark suite (NPB) [8]. The NAS Parallel Benchmark suite, consisting of different



(a) AMD EPYC 7401P (Naples)

(b) AMD EPYC 7702 (Rome)

Fig. 2: Details of the non-uniform memory architecture for the two AMD systems evaluated in this paper.

TABLE I: System Configurations. See Figure 2 for details on the two EPYC systems.

Feature	AMD SEV 1	AMD SEV 2	AMD SEV 3	Intel SGX
CPU	EPYC 7401P	EPYC 7702	EPYC 7402P	Core i7-8700
Sockets	1	2	1	1
Cores	24	128	24	6
NUMA	4 Nodes	2 Nodes	1 Node	1 Node
RAM	64GB	1TB	64GB	32GB

kernels and pseudo applications, has been used to study HPC systems for a long time and is still getting updates. These benchmarks can be used with multiple input data sizes, thus different class names. In this work, we used NPB Class C for both SEV and SGX and NPB Class D for SEV only.

B. Modern and Emerging HPC Workloads

Apart from the traditional scientific computing kernels/workloads, we also focus on workloads which characterize modern HPC usage. We selected a set of graph workloads (GAPBS) [9] with an input of a graph of road networks in the US. As a proxy for general machine learning training we used a decision tree workload (LightGBM) [20] (characterized by irregular memory accesses) which is trained using Microsoft's Learning to Rank (MSLR) data set. Finally, we used modern HPC workloads as well, including Kripke [13] (a particle transport simulation), LULESH [12] (a hydrodynamics simulation), Mobiliti [11] (a transportation benchmark), and BLAST [10] (a genomics workload). Kripke [13] is a highly scalable code which acts as a proxy for 3D Sn (functional discrete-ordinates) particle transport. Livermore Unstructured Lagrange Explicit Shock Hydro (LULESH) [12] application solves a simple yet "full-featured" hydrodynamics simulation problem. Mobiliti [11] is a transportation system simulator (based on parallel discrete event simulation), designed to work on high performance computing systems. Basic Local Alignment Search Tool (BLAST) [10] is a famous bio-informatics tool, which is used to search sequence similarity of a given genome sequence compared to an existing database. We specifically use BLASTN in this work, which is a version of BLAST used to search a nucleotide sequence against a nucleotide database.

C. Hardware Platforms Used

Table I shows the configurations of the hardware platforms used for these experiments. For all of our evaluations, we evaluate *without* hyperthreading by limiting the number of threads to the number of cores on each platform. This allows to isolate main causes of slowdown of trusted execution by keeping things simple and reducing cache contention among threads for workloads with large working set sizes.

We used three server class AMD machines. Figure 2 shows the detailed NUMA configuration of the AMD EPYC 7401P (Naples architecture, Figure 2a) and the AMD EPYC 7702 (Rome architecture, Figure 2b). The Naples-based system has 24 CPU cores with 6 cores on each of four dies in a single multi-chip module. Although this system is a single socket platform, it has four NUMA nodes. A multi-chip module package has characteristics similar to a multi-socket system in terms of latency and bandwidth between separate dies. With its four NUMA nodes the Naples-based system has high variation in memory latency depending on if the data is in the local NUMA node or one of the remote NUMA nodes.

We also evaluated a recent Rome-based system since this design has a more uniform memory architecture. The Rome-based system has 64 cores with 8 cores on each of 8 dies in a multi-chip package, and it is a dual socket system for a total of 128 cores. The Rome system has more chips per package, but has a more uniform memory architecture since each die is equidistant from the I/O die with the memory controllers. In the Rome-based system we evaluated, there is only one NUMA node *per socket*. However, we used a dual socket system so our evaluations have two NUMA nodes. We also used an EPYC 7402P (Rome architecture, with one socket) system for validation of some results discussed in section V.

The forthcoming supercomputers Frontier and El Capitan are based on AMD microarchitecture [21], though these will likely be based on a future microarchitecture (e.g., Zen 3 or Zen 4). The specific memory architecture of these future devices are currently unknown publicly, but it is likely that they will support multiple sockets and thus will have at least as much non-uniformity as the Rome-based system. Google's confidential cloud computing initiative also relies on AMD SEV for trusted execution support [22].

We use a desktop-class processor with 6 cores and a single NUMA node to perform Intel SGX experiments, as there did not exist a server-class Intel processor with the support of SGX at the time of performing SGX experiments in this paper. Recently, Intel SGX is made available in one of the Intel Xeon parts (Xeon E3). However, the size of secure memory (doubled to be 256MB in total) is still significantly smaller than the working set of most of the workloads studied in this paper (only *ep* has a working set smaller than 256MB) and the conclusions drawn in this work (discussed in section V) should still hold true.

D. Software Tools/Frameworks

To execute unmodified applications under SGX, we make use of SCONE [23] framework container. Programs are compiled statically and linked against a modified standard C library in SCONE. SCONE runtime also makes use of threads outside the enclave to perform asynchronous execution of system calls. We evaluated other SGX interfaces and picked SCONE as it provided the most complete support for unmodified applications. These other SGX programming interfaces are discussed in section V (Finding 4.4). Rewriting HPC applications for SGX's programming model, by partitioning them into secure and un-secure components, is arduous but not impossible. However, in this work we focus on the use case of unmodified HPC applications. Also, the overhead of containerization like SCONE has been shown to be low. The original work [23] introducing SCONE showed that it has a 0.6–1.22× throughput compared to native execution for services like Apache, Redis, NGINX, and Memcached [23]. We also tested the performance of NAS parallel benchmarks in the "simulation mode" of SCONE. This mode uses all of the SCONE interfaces, but does not enable SGX. We found that the geometric mean of slowdown compared to native execution is 1.19×, which is insignificant compared to the slowdown of trusted execution (with SGX) in SCONE as shown in section V (Finding 4). Finally, we observed the performance of two memory intensive micro-benchmarks, partitioned into secure and un-secure parts directly using Intel SGX SDK, and found those numbers to be in line with our observations with SCONE as discussed in section V (Finding 4).

For SEV, we make use of the AMD provided scripts to set-up the SEV enabled host machine and the guest virtual machine (VM) managed by QEMU. We also evaluated using Kata [24] which is a containerized interface to the hardware virtualization support in Linux. However, we found that Kata's support for SEV was too preliminary to get consistent results. Kata or other virtualized container interfaces may provide an even simpler programming interface to SEV in the future, but they will likely have the same performance characteristics as QEMU since they both use hardware support for virtualization. When running with QEMU, we assign all of the host cores to the guest and allocate enough memory on the guest to fit the entire resident memory of the application. Moreover, we do not include the guest boot-up/initialization time when calculating the workload slowdown.

The documentation and scripts required to set-up and run the experiments discussed in this work are available publicly.[1]

V. Understanding the Performance of TEEs

In this section, we will present our findings on the performance impact of TEEs for scientific computing workloads and the reasons for the observed slowdowns. We make following main findings:

1) When the user configures the NUMA allocation policy correctly, SEV has small overhead for most workloads.
2) SEV relies on QEMU and hardware virtualization, which causes significant performance degradation for some irregular workloads, I/O intensive workloads, and workloads with high thread contention.
3) SEV initialization is slow and depends on the memory footprint of the application.
4) SGX has high performance overhead mostly due to its limited secure memory capacity and partially due to parallel scalability limitations and programming challenges.

A. *Finding 1: SEV can be used for secure scientific computing without significant performance degradation for most workloads if it is configured correctly.*

SEV requires nested page tables and is only available when running in a VM (virtual machine). Therefore, we compare three different cases: native (un-secure), QEMU (virtualized, but also no security gaurantees), and QEMU+SEV which provides security from the hypervisor and other users.[2]

Figures 3 and 4 show the performance of the NAS Parallel Benchmarks for the C and D class inputs relative to the "native" execution without any security guarantees. The solid bars on these figures show the performance of three cases: native execution, "QEMU" which is a KVM-based hypervisor running a virtual machine with the benchmark, and "QEMU+SEV" which has the SEV security extensions enabled (all relative to the performance of native execution). This shows that SEV results in significant performance degradation (shown in green solid bars) compared to the virtualized QEMU execution when default system configuration is used.

In this section, we will discuss how most of these slowdowns can be eliminated through careful NUMA data placement. We also present data from two different generations of AMD platforms to further investigate the overheads of SEV.

Finding 1.1: *Enabling SEV causes performance degradation beyond virtualization overheads.*

Although there is some overhead from virtualization for the NAS Parallel Benchmarks as shown in the orange bars of Figures 3 and 4, there is significantly more performance overhead when enabling SEV (green bars, up to 3× slowdown over the native execution).

Finding 1.2: *SEV overhead is because of NUMA placement.*

The reason QEMU+SEV suffers more performance overhead than QEMU is that when an SEV enabled virtual machine (VM) is launched, the memory pages allocated to the guest RAM are pinned by the hypervisor (QEMU) using *mlock* syscall. As a result, all data for the application is allocated on a single NUMA node and multi-threaded processes which expect performance improvements from running on large NUMA systems suffer from performance degradation under SEV. QEMU without SEV does not have this restriction.

Why SEV requires locking pages to physical addresses?: Figure 5 shows details of how SEV is implemented. This figure shows both the interaction with the nested page table translation used for hardware virtualization acceleration and the memory encryption engine. First, this figure shows how the guest virtual address is translated through a nested page table since it must translate first into the guest physical address space then into the host physical address space. Importantly,

[1] https://github.com/lbnl-cybersecurity/tee-hpc

[2] The initial implementation of SEV has many security vulnerabilities [25]–[27]. However, more recent implementations (e.g., Rome) fix many of the published vulnerabilities but still have similar performance characteristics to the systems we evaluate.

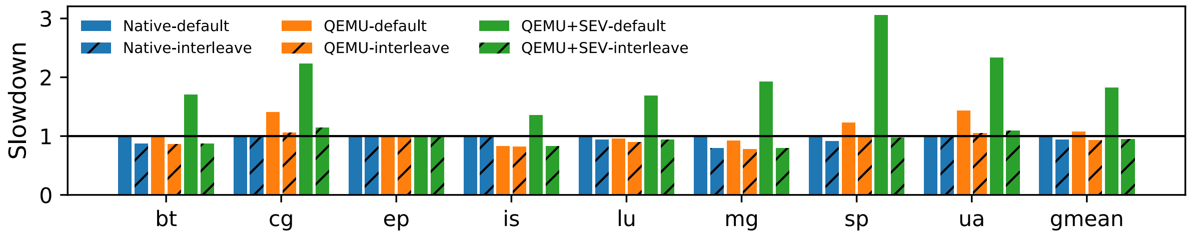


Fig. 3: Performance impact of SEV for NPB C Class on AMD Naples (24 Threads). The SEV performance overhead is mainly because of default NUMA memory allocation, most of which goes away with interleaved NUMA allocation.

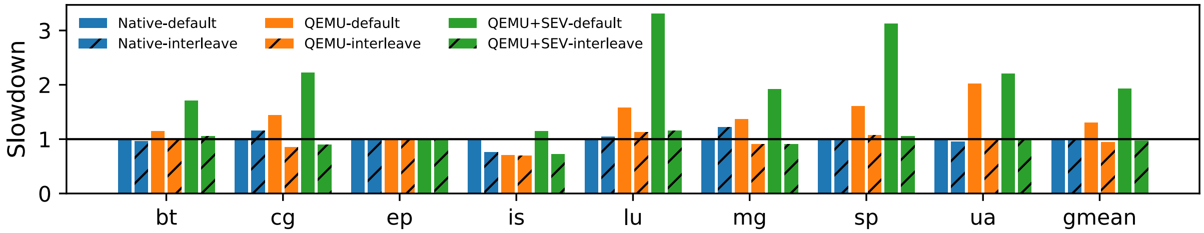


Fig. 4: Performance impact of SEV for NPB D Class on AMD Naples (24 Threads).

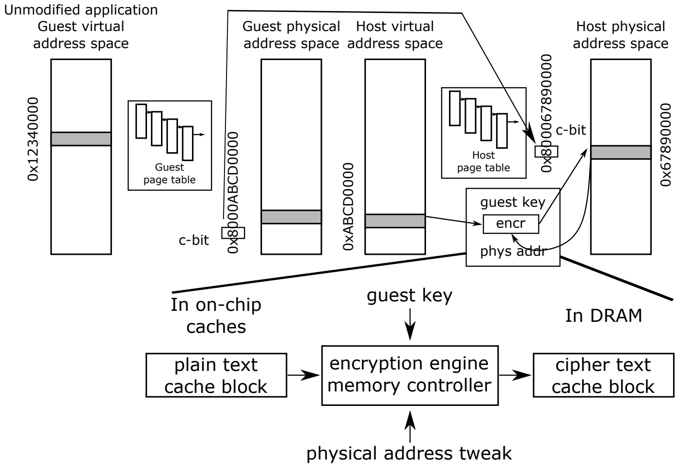
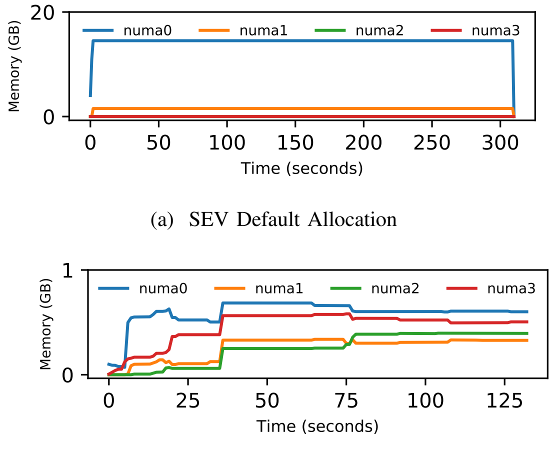


Fig. 5: Details of SEV encryption implementation.



(a) SEV Default Allocation

(b) No SEV Default Allocation

Fig. 6: Memory allocation over time using default policy.

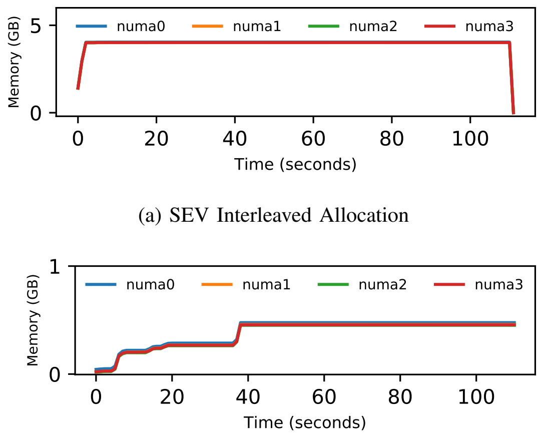
the “c-bit” or encrypted bit is removed from the guest physical address by hardware and replaced after the host page table translates the address to the host physical address space. By removing and replacing the c-bit, the hypervisor is unaware of which pages are encrypted or not.

Second, SEV must guarantee that two identical plaintext pages present at different physical addresses in the memory will have different cipher texts to protect against cipher text block move attacks. To make this possible, SEV uses a physical-address based tweak algorithm [25], [26] as shown in Figure 5 with the physical address of the cache block influencing the cipher text via an xor-encrypt-xor tweak [28]. Since the host-physical address is used to determine the cipher-text of a page, the hypervisor cannot move a page between two physical addresses once it is allocated to the secure VM.

This limitation causes two performance issues when using SEV. First, all data pages for the guest are *pinned* in physical memory by the hypervisor [29]. In fact, because the default NUMA policy on Linux is “first-touch”, all memory is allocated on a single NUMA node, which causes performance degradation for many of the scalable workloads evaluated in this work. Second, SEV-based guests can under-utilize the memory resources since they do not use on-demand paging.

Figures 6a and 6b visualize the memory allocation process when using QEMU and QEMU+SEV. These figures show the memory allocation over time on different NUMA nodes on a system with four NUMA nodes when a VM with 16 GB memory is launched to run (for example) *sp* benchmark. Figure 6a shows that under SEV all data is allocated at the time of the VM launch *on a single NUMA node* as opposed to the non-SEV case (Figure 6b) which follows on-demand paging scheme and spreads the data across all four nodes.

For additional evidence, we conducted an initial study on a single-socket AMD Rome based system (AMD EPYC 7402P, 24 core system, similar to Figure 2b but with a single



(a) SEV Interleaved Allocation

(b) No SEV Interleaved Allocation

Fig. 7: Memory allocation over time using an interleave policy.

package and four core dies) using NPB D class workloads. This system has a uniform memory architecture, and that is why the slowdowns due to NUMA placement issues (observed previously) do not exist in this case as shown in Figure 11.

Thus, we conclude the SEV-specific overhead is due to the NUMA allocation policy.

Finding 1.3: *Explicit interleaving of data across NUMA nodes using* numactl *recovers most of the performance loss.*

To mitigate the observed slowdown, we explicitly allocate memory pages across NUMA nodes rather than using the default NUMA memory allocation policy in the Linux kernel. We use `numactl` to allocate memory pages across NUMA nodes when the VM is launched under SEV. A visualization of the memory allocation using *interleaved* NUMA allocation policy is shown in Figure 7a. Under SEV, an equal amount of memory (4 GB on each node) is allocated across all nodes.

We observe that the interleaved memory allocation across all NUMA nodes results in significant performance improvements for SEV. In fact, the performance differences between QEMU and QEMU+SEV shrink as shown in Figure 3 and 4 when enabling NUMA interleaving (hatched bars). This is in contrast to prior work which evaluated server-based applications and found that using a single NUMA node results in the best performance for virtualized workloads [30]. Importantly, we also observe that for native execution the interleaved allocation results in better performance compared to the default allocation for most of the cases (prominent examples are *Kripke, Mobiliti,* and *cc* from GAPBS)

In addition to the HPC kernels in the NAS Parallel Benchmarks, we also studied modern HPC workloads. Figure 8a shows the execution time for native, QEMU and QEMU+SEV cases for GAPBS workloads when executed using a road network graph. Similar to NPB, NUMA interleaving reduces the difference between QEMU+SEV and SEV. Similar trends are found for other HPC workloads as shown in Figure 8b.

However, there are still some cases where QEMU and QEMU+SEV experience performance degradation compared to the native (unsecure) baseline. These differences can be attributed to *virtualization* overhead as discussed in Finding 2.

Finding 1.4: *NUMA placement still matters on new platforms with more uniform memory architecture.*

As discussed in Section IV, we studied the performance of these benchmarks on another modern server class AMD machine EPYC 7702 (Rome architecture), which contains two NUMA nodes instead of four (see Figure 2). Figure 9, Figure 10a and Figure 10b show the relative performance of native, QEMU, and QEMU+SEV for the Rome system. Similar to the Naples system, there are significant overheads when using SEV unless the data is explicitly interleaved between NUMA nodes. Thus, even for systems that have more "uniformity" in their memory architecture, data placement is important for performance when using SEV.

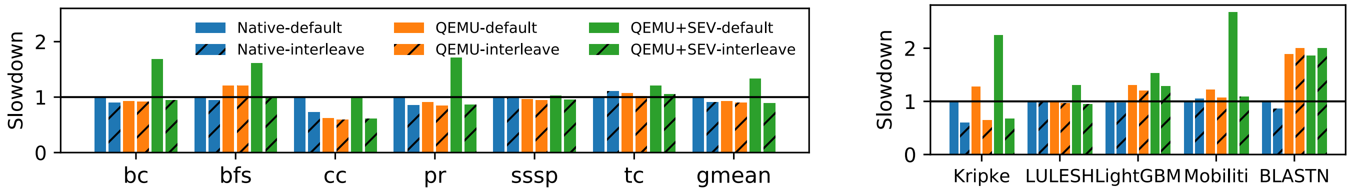
Finding 1 summary: When enabling SEV, there are additional overheads beyond just the virtualization platform overheads. These overheads are caused by the memory allocation restrictions of the SEV technology and persist even on the most recent architectures. However, we can overcome these SEV-specific overheads by explicitly interleaving data between NUMA nodes when the virtual machine is initialized.

B. ***Finding 2:*** *The remaining SEV performance differences are due to virtualization overheads.*

We find that in some cases there is performance degradation of the QEMU+SEV system compared to the baseline native execution even after applying our NUMA interleaving configuration change. These slowdowns come from the use of hardware virtualization and QEMU. For example, in Figure 10a, *sssp* with QEMU+SEV shows considerable slowdown compared to Native-default case irrespective of memory allocation policy (default or interleaved) on AMD Rome architecture. As visible in the Figure 10a, the performance of QEMU+SEV and QEMU match, indicating that the main cause of this slowdown is virtualization itself, not the SEV extension.

We observed that, when run with 128 threads (as in Figure 10a), *sssp* shows much higher number of kvm exits per second caused by the PAUSE instruction in comparison to the case when it is run with a smaller number of threads (e.g., 32). The PAUSE instruction is used to implement spinlocks and can cause KVM exits (i.e., a usermode to hypervisor switch) which has a higher latency than a normal context switch. In fact, when executed with only 32 threads, the virtualization slowdown of *sssp* improves to 1.7× (in contrast to 4× in Figure 10a). Similarly, the QEMU overhead for *bfs* reduces to 1.6× with 32 execution threads in contrast to 2.6× with 128 execution threads (Ding et al. made similar findings [31]). Thus, when using QEMU or QEMU+SEV it is important to use the appropriate number of execution threads for your workload and workloads with highly contended locks may result in significant performance degradation.

In Figure 8b, BLASTN also shows slowdown by virtualization on AMD Naples architecture. The nucleotide database which is used by BLASTN is approximately 245GB in size



(a) GAPBS (road network)

(b) Real world HPC workloads

Fig. 8: Performance impact of SEV for GAPBS and other real world HPC workloads on AMD Naples (24 Threads). Interleaved NUMA allocation works for graph and other HPC workloads as well except BLASTN which shows high overhead mainly because of virtualized disk I/O operations.

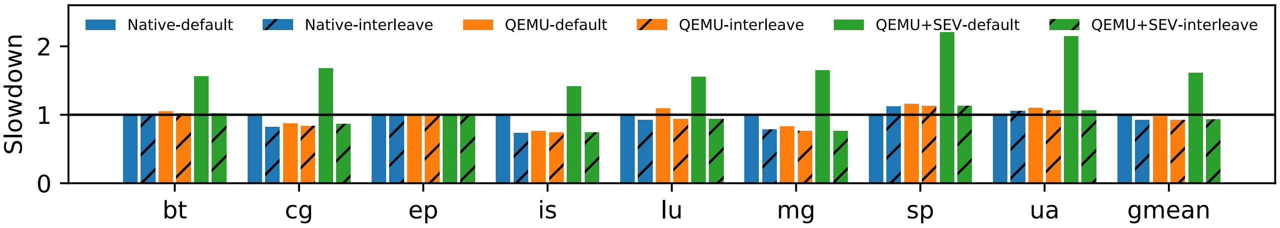
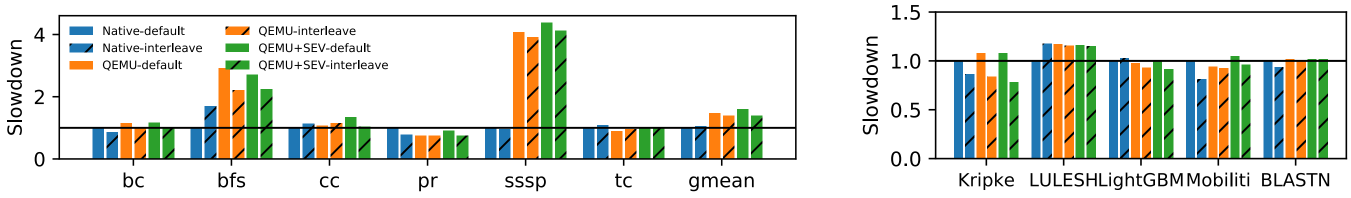


Fig. 9: Performance impact of SEV for NPB D Class on AMD Rome (128 Threads)



(a) GAPBS (road network)

(b) Real world HPC workloads

Fig. 10: Performance impact of SEV for GAPBS and other real world HPC workloads on AMD Rome (128 Threads). NUMA placement still matters on platforms with more uniform memory architecture. Two examples where main cause of overhead is virtualization are bfs and sssp.

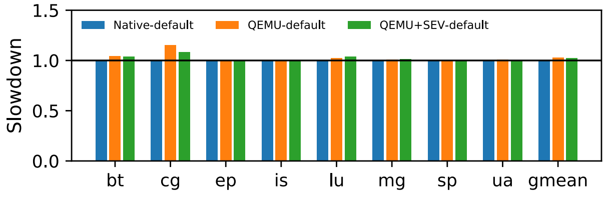


Fig. 11: NPB D Class on AMD EPYC 7402P (24 Threads)

(much larger than the memory size of 64 GB on our AMD Naples system), which leads to many disk I/O operations and thus slowdown under virtualization. On the other hand, when the same workload is executed on AMD Rome system (which has 1 TB of memory), there is not any noticeable virtualization overhead as shown in Figure 10b since the workload can fit in the available system memory.

There is significant prior work quantifying the impact of virtualization on the performance of HPC workloads [31]–[33]. These prior works mostly focus on overheads from TLB misses and nested page table walks. Similarly, our results show the virtualization overheads grow as the working set of the applications grow and are worse for workloads with irregular access patterns (e.g., graph workloads). Prior work has shown you can reduce this overhead by using huge pages or through changes to the hardware (e.g., Virtualized Direct Segments [33]). Additionally, the work of Ding et al. [31] presents possible strategies to mitigate the virtualization slowdown caused by multithreaded application scaling.

C. ***Finding 3:*** *SEV initialization is slow and depends on the memory footprint of the VM. Note: all previous data ignores the VM initialization time.*

We find that the time taken to initialize the workload is significant when using QEMU and increases when using QEMU+SEV. When using QEMU or QEMU+SEV, before running the workload the virtual machine guest operating system must complete the boot process. For QEMU this bootup time takes about one minute for our workloads.

However, when enabling QEMU+SEV, this boot time increases due to the hypervisor having to initialize the memory before handing it over to the guest OS. As shown in Figure 12,

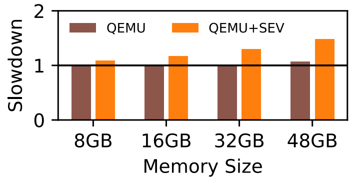


Fig. 12: Performance of VM boot (relative to QEMU-8GB)

SEV can cause a slowdown (relative to QEMU-8 GB) of 1.1×–1.47× depending on the size of the VM memory (from 8 GB to 48 GB). In addition to the memory initialization, QEMU+SEV also needs extra time for key management when launching a guest with QEMU+SEV. However, we believe that the main source of SEV slowdown is the fact that the entire VM memory has to be allocated at once in case of QEMU+SEV in contrast to on-demand allocation in case of QEMU (as discussed in section V-A), as evident by the increase in slowdown as the VM memory size is increased. This can specially become a bottleneck for the use cases where the user intend to launch their jobs in a new VM each time (e.g., when using Kata containers [24]).

*D. **Finding 4:** SGX is inappropriate for unmodified scientfic computing applications.*

We find a number of reasons that SGX is not an appropriate technology for securing HPC workloads. A primary design goal of SGX is to enable a small trusted compute base, and SGX was not designed to support large scale workloads. We find that running HPC workloads under SGX causes a (1×–126×) slowdown (mostly due to its limited secure memory capacity), workloads exhibit poor thread scalability under SGX, and it is difficult to adapt HPC code to work under the SGX programming model.

Finding 4.1: *Workloads with working sets larger than about 100 MB suffer large performance degradation under SGX.* Figure 13a shows the slowdown of HPC workloads under SGX compared to an un-secure baseline. For this experiment, we ran NPB with the "class C" inputs (blue in Figure 13a). We were limited to using the class C inputs, as most class D inputs were too large to run on the desktop systems that support SGX. However, we believe that running larger inputs under SGX would show at least as much performance overhead as the smaller inputs. We also show the relative performance of graph workloads and other modern HPC workloads in Figure 13a). We were not able to run BLASTN workload with SGX due its dependencies (discussed more in Finding 4.4).

Most of the performance degradation shown in Figure 13a can be explained by the overhead of moving data from un-secure memory into secure memory. SGX has a limited amount of secure memory, about 100 MB. Thus, any workload with a working set larger than 100 MB must use the secure memory as an *enclave page cache* (EPC). The EPC is managed by the SGX driver in software and has similar behavior to OS swapping and moving pages between normal and secure memory is a high latency event.

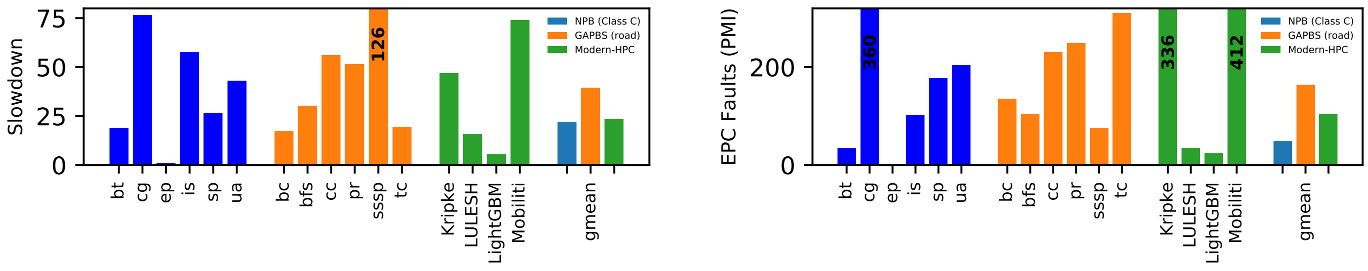
Figure 13b shows the number of EPC faults per million instructions for each of the workloads. This figure shows that most of the slowdown in Figure 13a can be explained by the EPC fault rate. The workloads with the highest rate of moving data between secure memory and normal memory (e.g., *cg* from NPB, Mobiliti, and Kripke) show very high slowdown. On the other hand, *ep* from NPB shows little performance overhead with SGX because it has a very small working set size (about 28 MB) which fits in the EPC and does not require data movement between secure and normal memory spaces.

Finding 4.2: *In some cases, SGX slowdown can be caused by system calls.* Applications under SGX exit the enclave to process a system call. This can become a problem for workloads with a large number of system calls. In the studied HPC workloads, the only case where we found system calls to be the dominant source of performance overhead is *sssp* benchmark from GAPBS. As shown in Figure 13b, the slowdown for *sssp* does not correlate with EPC fault rate. *sssp* shows significantly higher number of enclave exits (and system calls) and is the main contributor to its performance overhead compared to the un-secure execution. Most of these system calls were *write* and *futex* calls, which are needed due to the benchmark printing progress to the terminal. The *futex* calls are used for synchronization (of multiple threads) before printing the status messages using *write* calls. The effect on slowdown because of *futex* system calls can be understood by the difference in the observed slowdown for six thread execution (126×) and single thread execution (20×), which does not need any synchronization.

Finding 4.3: *Workloads exhibit poor multithreaded scaling under SGX.* Another factor that aggravates the slowdowns under SGX is explained with the help of Figure 14, which shows the workload performance when increasing the number of threads. Figure 14a shows that *cg* only achieves a speedup of 1.4× with six threads when using SGX compared to about 4× speedup normally. We hypothesize that the handling of EPC faults by the SGX kernel driver becomes the serializing factor because all logical processors executing an enclave's code are required to exit the enclave whenever an EPC page is deallocated [2]. Similar behavior is exhibited by most of the other workloads with high resident memory size. On the other hand, workloads with working set sizes that fit in the EPC (e.g., *ep*) scale under SGX as they would under normal execution as shown in Figure 14b.

Finding 4.4: *SGX's programming model is a poor fit for HPC applications.* Intel distributes an official SDK [34] for SGX which requires users to re-write their application and divide it into two pieces, secure code and non-secure code. Due to the complex nature of HPC codes, dependencies on external libraries, and frequent use of legacy codes (including a non-trivial number of them written in Fortran), we investigated several alternative interfaces to SGX which reduce the burden on the programmer.

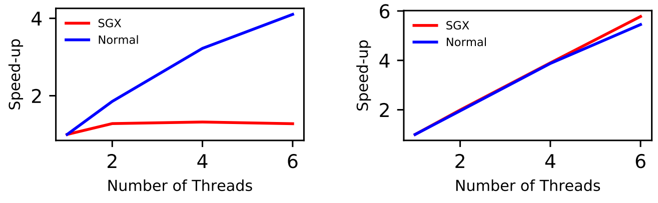
There are multiple third party solutions to run unmodified applications under SGX including SCONE [23], Graphene [35], Haven [36] and Asylo [37]. We did initial



(a) Relative performance running under SGX compared to native execution.

(b) EPC fault rate (per million instructions) when running under SGX

Fig. 13: Performance impact of SGX and its relation to EPC (Enclave Page Cache) faults



(a) cg

(b) ep

Fig. 14: Impact of multiple execution threads

experiments with both Graphene and SCONE as they were the best supported third party solutions at the time we ran our experiments. SCONE provides containerized environment and is easier to set-up and has a better support of running diverse workloads without any modifications, so we used SCONE for our experiments. Although we only evaluated SCONE, all SGX programming interfaces have similar limitations due to SGX's design which limits the TCB. Graphene [35] is not as convenient to use as SCONE and Google Asylo's [37] recently added support to run unmodified applications still lags behind SCONE in terms of the number of supported use-cases. Open Enclave SDK [38] is another SDK to build enclave applications and does not support unmodified applications.

We found that even with SCONE, which promises to run unmodified applications with SGX, it is fundamentally difficult to use SGX to run HPC applications. In order to keep the library OS simple, SCONE makes use of the musl libc library, instead of more traditional C library glibc, along-with some containerized services using the Linux Kernel Library (LKL). The use of musl libc instead of glibc means many applications are not portable to SCONE (e.g., BLASTN failed to compile inside SCONE and many common frameworks such as TensorFlow require glibc instead of musl libc). Moreover, SCONE does not support some system calls like fork, exec and clone mainly due to its user space threading model and the architectural limitations of SGX [23] which further limits its applicability to scalable applications.

Finding 4 summary: The current implementation of Intel's SGX limits the secure memory size which severely affects the performance of any workload that has a working set that does not fit in this cache. Additionally, there is currently no stable support to run unmodified workloads under SGX. The limited EPC capacity and application partitioning are a fundamental design constraints of SGX. Thus, we conclude that SGX is unsuitable for secure execution of HPC applications. In Section VII, we discuss possible future SGX implementations and how they may improve performance for some HPC workloads.

VI. Discussion

A. Beyond Single Node

Scientific computing workloads often scale across multiple machines (nodes). In this work, we only focus on a single node to isolate the performance impact of hardware TEEs. To understand the impact of communicating between TEEs on multiple nodes, we conducted a preliminary investigation of a multi-node system with support of SEV on CloudLab [39].

Current HPC systems mostly rely on high performance transport protocols like RDMA for communication among multiple nodes. However, RDMA does not provide any secure communication support although there is a recent research proposal for secure RDMA [40]. Therefore, we instead evaluated TCP for communication among machines using OSU MPI microbenchmarks. For point-to-point bandwidth benchmarks, we observed a 2× reduction in bandwidth of when comparing QEMU to QEMU+SEV, but the latency remains the same (approximately 1000 μs) for both QEMU and QEMU+SEV (ranging from 1 byte to 2MB packet sizes).

However, in this simple benchmark, the communication between nodes is insecure as there is no support for encrypted communication between multiple nodes in SEV automatically. A naive solution to make this communication secure is to use a VPN. We experimented with OpenVPN and found the slowdown of VPN based secure communication to be large. For example, for above microbenchmarks, the bandwidth number drops over 10× and latency increases by almost 20×.

Therefore, we conclude that there is a need to develop more performant architectures to enable TEEs across multiple nodes in a distributed memory. There are some existing software based solutions to enable encrypted communication across nodes, but they might not be sufficient for scientific computing scale workloads. For example, SCONE [23] provides a network shield which transparently protects the communication among multiple nodes (each with its SGX hardware). Asylo [37] (open-source framework for developing enclave applications) allows enclave based applications to be scaled across multiple machines using gRPCs (google remote procedure calls), while being agnostic to TEE implementation.

B. Observations on Security of TEEs

Although the focus of this paper is the performance analysis of TEEs for secure HPC, we provide a brief security analysis of TEEs discussed in this paper. SGX provides integrity guarantees while SEV lacks such support. However, the weaker guarantees of SEV (lack of integrity) are considered to be adequate by Google's confidential cloud computing initiative, and these guarantees are becoming stronger. AMD has introduced SEV-ES [41] that adds encryption of guest register state to provide additional protection against VM state related attacks, and SEV-SNP [41] provides integrity checks. These are encouraging developments from the security perspective, as they address some of the vulnerabilities and limitations of SEV. It should be noted that SEV-SNP [41] does not provide integrity guarantees using Merkle tree like data structures (as SGX does). Therefore, it is more scalable and can support larger secure memory sizes. Additionally, it seems Intel is also moving in the direction of full memory encryption and virtual machine-based trusted execution environments like AMD's SEV with total memory encryption (TME) and multi-key total memory encryption (MKTME) technologies [42], [43].

Finally, in this paper we have focused on just one aspect of the entire secure application workflow which may include other steps as well in addition to running it inside an SEV or SGX enclave. However, we believe that the execution of the workload itself in a secure enclave is the most important factor for performance analysis.

C. TEEs in Heterogeneous Systems

As computer systems become more heterogeneous and employ different hardware accelerators to optimize for specific applications, the attack vector expand in size. Thus, it is important to expand TEEs across all processing elements of such systems. We touch some aspects of heterogeneity i.e. through the use of NUMA systems in this paper. However, arguably the more interesting aspects of heterogeneity (accelerator based systems) are not in the scope of this paper. Research on TEEs in accelerator based systems is still at very early stage. Examples of research works to enable trusted execution in accelerator based systems include: Graviton [44] and HETEE [45]. Graviton [44] provides trusted execution capabilities on GPUs by providing isolation to sensitive kernels from other code executing on GPUs and the host. HETEE [45] relies on a centralized controller to manage trusted execution on all computing units including GPUs and accelerators. However, there is still a need to investigate what new features need to be added to existing practical TEE solutions to enable trusted execution in accelerator based systems.

D. Research Avenues

Next, we discuss few research avenues for software framework developers and computer architects to enable secure HPC using TEEs.

Security requirements of HPC workloads can be diverse. At the same time, the HPC platforms can be heterogeneous possibly composed of nodes from multiple vendors e.g., Intel and AMD, thus making both SGX and SEV available in the same environment. Therefore, we propose the idea of an intelligent job schedulers which can allocate applications to an appropriate node depending on their sensitivity and the expected slowdown from the secure environment (which can be predicted using pre-trained models).

Partitioning HPC applications into secure and non-secure parts can be difficult as they often rely on various third-party libraries. The support of automated application partitioning into secure and non-secure parts can help in mitigating the performance slowdowns by keeping the secure memory footprint to the minimum.

The limited secure memory size (as in SGX) is an issue that can have a severe impact on the performance of trusted execution of HPC workloads. Apart from exploring the ideas to design the more optimized integrity trees (which would enable larger secure memory sizes), techniques like prefetching (of sensitive pages into secure memory) can be adapted to overcome the performance slowdowns caused by EPC faults.

VII. Conclusion

In this paper, we studied the performance impact of two TEEs (SEV and SGX) for secure execution of scientific computing workloads. We showed that while Intel's SGX causes significant performance degradation for unmodified HPC workloads, when configured correctly AMD's SEV technology can provide protection against the most important HPC threats with minor performance overhead. Although there is no hardware available with Intel's MKTME technology, we believe the performance characteristics will be similar to the QEMU+SEV performance presented in this paper since these two technologies have closely related designs. The new developments in AMD SEV (SEV-ES [41] and SEV-SNP [41]) do not change the fact that pages are still pinned to a given NUMA node, so the issues discussed in section V still remain.

So, we believe, architectural innovations are necessary to mitigate the overheads reported in section V. Specifically, we are planning to harness the ideas in direct segments and coalesced TLBs [33], [46] to mitigate the overhead of virtualization in SEV. To reduce the overhead of SGX we are exploring opportunities for more intelligent page prefetching and eviction by learning the memory access pattens of applications and implementing it in hardware as part of the TCB.

Acknowledgement

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 and the National Science Foundation under Grant

No. CNS1850566. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors of this work. We would also like to thank anonymous reviewers, Marc Orr, and members of Davis Computer Architecture Research Group (DArchR) for their valuable feedback.

References

[1] S. Peisert, "Security in high-performance computing environments," *Communications of the ACM*, vol. 60, no. 9, pp. 72–80, 2017.

[2] V. Costan and S. Devadas, "Intel SGX Explained," Cryptology ePrint Archive, 2016, https://eprint.iacr.org/2016/086.

[3] T. Alves and D. Felton, "TrustZone: Integrated Hardware and Software Security," *Information Quarterly*, pp. 18–24, 2004.

[4] D. Kaplan, J. Powell, and T. Woller, "AMD MEMORY ENCRYPTION," 2016, white paper.

[5] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *European Conference on Computer Systems*, 2020, pp. 1–16.

[6] O. Weisse, V. Bertacco, and T. Austin, "Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves," in *International Symposium on Computer Architecture*, 2017, pp. 81–93.

[7] A. T. Gjerdrum, R. Pettersen, H. D. Johansen, and D. Johansen, "Performance of Trusted Computing in Cloud Infrastructures with Intel SGX," in *International Conference on Cloud Computing and Services Science*, 2017, pp. 668–675.

[8] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, "The NAS Parallel Benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[9] S. Beamer, K. Asanović, and D. Patterson, "The GAP Benchmark Suite," *arXiv preprint arXiv:1508.03619*, 2015.

[10] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic Local Alignment Search Tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[11] C. Chan, B. Wang, J. Bachan, and J. Macfarlane, "Mobiliti: Scalable Transportation Simulation Using High-Performance Parallel Computing," in *IEEE International Conference on Intelligent Transportation Systems*, 2018, pp. 634–641.

[12] "Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory," Tech. Rep. LLNL-TR-490254.

[13] A. J. Kunen, T. S. Bailey, and P. N. Brown, "KRIPKE - A Massively Parallel Transport Mini-App," Lawrence Livermore National Lab (LLNL), Livermore, CA, Tech. Rep., 2015.

[14] S. Mofrad, F. Zhang, S. Lu, and W. Shi, "A Comparison Study of Intel SGX and AMD Memory Encryption Technology," in *7th International Workshop on Hardware and Architectural Support for Security and Privacy*, 2018, pp. 1–8.

[15] T. Dinh Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and D. Hagimont, "Everything You Should Know About Intel SGX Performance on Virtualized Systems," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, pp. 1–21, 2019.

[16] C. Göttel, R. Pires, I. Rocha, S. Vaucher, P. Felber, M. Pasin, and V. Schiavoni, "Security, Performance and Energy Trade-offs of Hardware-Assisted Memory Protection Mechanisms," in *IEEE 37th Symposium on Reliable Distributed Systems*. IEEE, 2018, pp. 133–142.

[17] S. Brenner, M. Behlendorf, and R. Kapitza, "Trusted Execution, and the Impact of Security on Performance," in *Proceedings of the 3rd Workshop on System Software for Trusted Execution, SysTEX*, 2018, pp. 28–33.

[18] M. Taassori, A. Shafiee, and R. Balasubramonian, "VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures," in *Proceedings of the 23rd ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Williamsburg, VA, Mar. 2018, pp. 665–678.

[19] M. Orenbach, P. Lifshits, M. Minkin, and M. Silberstein, "Eleos: ExitLess OS Services for SGX Enclaves," in *Proceedings of the 12th ACM European Conference on Computer Systems*, 2017, pp. 238–253.

[20] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Proceedings of Advances in Neural Information Processing Systems*, Long Beach, CA, Dec. 2017, pp. 3146–3154.

[21] https://www.amd.com/en/products/exascale-era.

[22] https://cloud.google.com/blog/products/identity-security/expanding-google-clouds-confidential-computing-portfolio.

[23] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell *et al.*, "SCONE: Secure Linux Containers with Intel SGX," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, 2016, pp. 689–703.

[24] "Kata Containers," Available: https://katacontainers.io/.

[25] Z.-H. Du, Z. Ying, Z. Ma, Y. Mai, P. Wang, J. Liu, and J. Fang, "Secure Encrypted Virtualization is Unsecure," *arXiv preprint arXiv:1712.05090*.

[26] L. Wilke, J. Wichelmann, M. Morbitzer, and T. Eisenbarth, "SEVurity: No Security Without Integrity - Breaking Integrity-Free Memory Encryption with Minimal Assumptions," in *IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1431–1444.

[27] M. Morbitzer, M. Huber, J. Horsch, and S. Wessel, "Severed: Subverting AMD's Virtual Machine Encryption," in *Proceedings of the 11th European Workshop on Systems Security*, 2018, pp. 1–6.

[28] P. Rogaway, "Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC," in *Advances in Cryptology - ASIACRYPT 2004*. Springer Berlin Heidelberg, 2004, pp. 16–31.

[29] D. Kaplan, "AMD x86 Memory Encryption Technologies," in *Linux Security Summit*, 2017.

[30] U. Kamio and Y. Kinoshita, "KVM/QEMU tuning of NUMA and Memory," in *Open Infrastructure Summit*, 2017.

[31] X. Ding, P. B. Gibbons, and M. A. Kozuch, "A Hidden Cost of Virtualization when Scaling Multicore Applications," in *5th USENIX Workshop on Hot Topics in Cloud Computing*, 2013.

[32] A. Gupta, L. V. Kale, F. Gioachin, V. March, C. H. Suen, B.-S. Lee, P. Faraboschi, R. Kaufmann, and D. Milojicic, "The Who, What, Why, and How of High Performance Computing in the Cloud," in *IEEE 5th International Conference on Cloud Computing Technology and Science*, 2013, pp. 306–314.

[33] J. Gandhi, A. Basu, M. D. Hill, and M. M. Swift, "Efficient Memory Virtualization: Reducing Dimensionality of Nested Page Walks," in *47th International Symposium on Microarchitecture*, 2014, pp. 178–189.

[34] https://software.intel.com/en-us/sgx/sdk.

[35] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX," in *USENIX Annual Technical Conference (ATC)*, 2017.

[36] A. Baumann, M. Peinado, and G. Hunt, "Shielding Applications from an Untrusted Cloud with Haven," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, p. 8, 2015.

[37] "Asylo," Available: https://asylo.dev/.

[38] "Open Enclave SDK," Available: https://openenclave.io/sdk/.

[39] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Jul. 2019, pp. 1–14.

[40] K. Taranov, B. Rothenberger, A. Perrig, and T. Hoefler, "sRDMA–Efficient NIC-based Authentication and Encryption for Remote Direct Memory Access," in *USENIX ATC*, 2020, pp. 691–704.

[41] https://developer.amd.com/sev/.

[42] https://arstechnica.com/gadgets/2020/02/intel-promises-full-memory-encryption-in-upcoming-cpus/.

[43] "Intel Architecture Memory Encryption Technologies Specification," Intel Corporation, Tech. Rep. Rev. 1.2, April 2019.

[44] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on GPUs," in *13th USENIX Symposium on Operating Systems Design and Implementatio (OSDI 18)*, 2018, pp. 681–696.

[45] J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, B. Zhao, Z. Wang, Y. Zhang, J. Ying, L. Zhang, and D. Meng, "Enabling Rack-scale Confidential Computing Using Heterogeneous Trusted Execution Environment," in *IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 991–1006.

[46] N. Kunati and M. M. Swift, "Implementation of Direct Segments on a RISC-V Processor," *Proceedings of Second Workshop on Computer Architecture Research with RISC-V*, 2018.