SGXGauge: A Comprehensive Benchmark Suite for Intel SGX

Sandeep Kumar
School of Information Technology
Indian Institute of Technology, Delhi
New Delhi, India
Email: sandeep.kumar@cse.iitd.ac.in

Abhisek Panda
Department of Computer Science
Indian Institute of Technology, Delhi
New Delhi, India
Email: abhisek.panda@cse.iitd.ac.in

Smruti R. Sarangi
Department of Computer Science
Indian Institute of Technology, Delhi
New Delhi, India
Email: srsarangi@cse.iitd.ac.in

***Abstract*—Trusted execution environments (TEEs) such as Intel SGX facilitate the secure execution of an application on untrusted machines. A plethora of work focuses on improving the performance of such environments necessitating the need for a standard, widely accepted benchmark suite. We present *SGXGauge*, a benchmark suite for SGX containing a diverse set of workloads from different domains. We also thoroughly characterize the behavior of the benchmark suite on a native platform and on a platform that uses a library OS-based shim layer (GrapheneSGX).**

***Index Terms*—Intel SGX, benchmark, EPC, library operating system**

I. Introduction

Intel Secure Guard eXtension or Intel SGX [1], [2] has gained popularity in recent years as a way to securely execute an application on a remote, untrusted machine. The security of the application and data within SGX, i.e., confidentiality, integrity, and freshness are guaranteed by the hardware.

However, this protection comes at a cost in terms of certain restrictions on the applications running within SGX, such as the lack of support for system calls since the operating system is not a part of the trusted framework of SGX [2], and a limited amount of secure memory called the *enclave page cache* or EPC. Applications allocating more memory than the EPC incur a significant amount of performance overhead [2], [3].

Researchers have focused on alleviating this problem by proposing different mechanisms and workarounds to reduce the overheads [4], [5], [6], [7], [8], [3]. To show the benefits of their methods, researchers have resorted to manual porting of applications to Intel SGX [9], [10]. However, porting an application requires significant expertise and development effort [9]. Also, the decision of which application to port is generally motivated by the ease of porting, and not necessarily by the gains accrued by doing so. Hence, there is no accepted, standard method for benchmarking SGX-based systems primarily due to the ad hoc nature of workload creation.

A benchmark suite needs to thoroughly evaluate all the critical components of Intel SGX, and enable performance comparison by setting a common denominator across different proposals – this is missing in prior work [9], [11]. We present *SGXGauge* – a comprehensive benchmark suite for Intel SGX. SGXGauge contains 10 real-world and synthetic benchmarks from different domains that thoroughly evaluate all the critical components of Intel SGX.

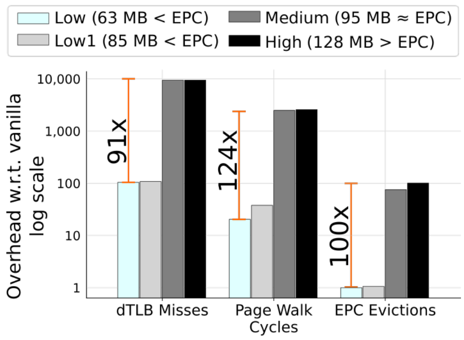

Fig. 1: Allocating beyond the EPC size increases the overhead. The baseline is a *Vanilla* (non-SGX) setting with the same input size. For EPC evictions the baseline is the Low setting.

II. Motivation

Limited work has been done in this area, mainly due to the limitations of the Intel SGX framework and the engineering effort required to port an application to SGX. Hasan et al. [9] and Fu et al. [11] ported LMbench and Nbench to SGX: LMbench-SGX and Nbench-SGX, respectively. LMbench-SGX mainly focuses on the memory bandwidth and the system call latencies. Nbench-SGX contains CPU-intensive workloads and is designed to check the efficiency of integer and floating-point operations on a CPU.

Impact of the EPC: The limited amount of EPC memory is one of the biggest challenges in SGX [3], [2]. Many applications' working-set is greater than the EPC, which forces SGX to move the pages to untrusted memory (after securing them). In case of an access to an evicted page (an EPC fault), SGX brings back the page back to the EPC. These EPC faults are common and incur performance overheads. As shown in Figure 1, on crossing the EPC boundary the number of dTLB misses increases by 91×, page table walk cycles by more than 124×, and EPC evictions by 100× as compared to when the memory footprint is less than the EPC size. *Hence, analyzing*

TABLE I: Terminology

Execution Modes	
Vanilla	An application executing without Intel SGX support.
Native	A ported application executing within Intel SGX.
LibOS	An application executing with GrapheneSGX (a library OS) [15].
Input Settings (I/P)	
Low: memory $<$ EPC, **Medium**: memory $\approx$ EPC, **High**: memory $>$ EPC	

TABLE II: Description of the workloads in SGXGauge along with the specific settings used in the paper. (Thr: Threads)

Workload	*Native*	*LibOS*	I/P: **Low**, **Medium**, **High**	Thr
Blockchain [18]	✓	✓	Blocks 3, 5, 8	12
OpenSSL [13]	✓	✓	File 76 MB, 88 MB, 151 MB	1
B-Tree [19]	✓	✓	Elements 1 M, 1.5 M, 2 M	1
HashJoin [20]	✓	✓	Table 61 MB, 91 MB, 122 MB	1
BFS [21]	✓	✓	Nodes 70 K, 100 K, 150K Edges 909 K, 1.3 M, 1.9 M	1
PageRank [21]	✓	✓	Nodes 4,500, 4,750, 5,000 Edges 10.1 M, 11.2, 12.5	1
Memcached [22]	✗	✓	Records: 50 K, 100 K, 200 K	1
XSBench [23]	✗	✓	Points: 53 K, 88 K, 768 K	1
Lighttpd [24]	✗	✓	Requests: 50 K, 60 K, 70 K.	16
SVM [25]	✗	✓	Data 4 K, 6 K, 10 K Features 128	1

the impact of the EPC size on the performance is crucial – a fact ignored by LMbench-SGX [9] and Nbench-SGX [10].

Real-world Benchmarks: Real-world applications exhibit different phases during their execution. A typical pattern is that an application will read some data from the file system, process it, and then store the results. Micro-benchmarks such as Nbench [12] lack this phase change behavior and thus do not represent a real-world scenario.

III. SGXGauge Benchmark Suite

To select the benchmarks for SGXGauge[1] (see Table II), we selected workloads that have been used by highly cited works using SGX in the recent past. Our main aim was to ensure that every component of SGX is stressed and evaluated by SGXGauge. There are three main sources of overhead in Intel SGX: encryption/decryption, enclave transitions, and EPC faults. First, we selected some of the most commonly used workloads such as OpenSSL [13], [14] and Lighttpd [15], [8], [16] that stress the enclave transition mechanism. To stress the CPU, we selected the Blockchain workload, which is a CPU-intensive and multi-threaded workload. However, while it stresses the CPU, it does not use a lot of memory. To ensure both the components are stressed, we opted for an HPC workload XSBench [17]. In order to exclusively stress the EPC, we selected the following from prior work: B-Tree, BFS, HashJoin, and PageRank. Each of them has different data access patterns.

IV. Evaluation

Here, we discuss the performance of workloads in SGXGauge under different execution modes and with different

[1]https://github.com/srsarangi/SGXGauge/

TABLE III: The overhead in performance (run time) and other system events. Avg. value of EPC evictions is reported when compared with the *Vanilla* mode.

	Over-head	dTLB misses	Walk Cycles	Stall cycles	LLC misses	EPC Evicts
Native Mode w.r.t *Vanilla* (6 workloads)						
Low	2.0×	8.38×	29.7×	2.5×	1.8×	21.5 K
Medium	3.0×	14.6×	57.0×	5.3×	2.0×	49.6 K
High	3.4×	17.48×	59.1×	6.4×	3.0×	79.6 K
LibOS Mode w.r.t *Vanilla* (10 workloads)						
Low	2.03×	40.6×	517×	114×	24×	796 K
Medium	3.13×	59.7×	724×	146×	18.5×	1,792 K
High	3.7×	44.0×	113×	12.7×	15.5×	2,255 K

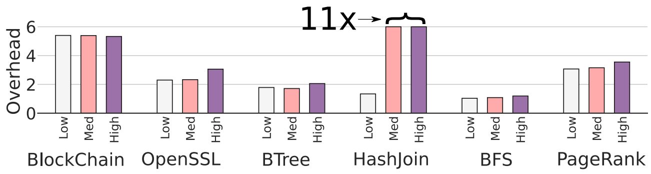

Fig. 2: Performance impact of SGX on applications in the *Native* mode for different input sizes.

input settings (see Table I). Table III shows an overview of the evaluation results. Our test system uses a single-socket Intel Xeon E-218G CPU with 32 GB of DRAM. The size of the usable secure memory (EPC) is 92 MB. A *LibOS* allows the execution of an unmodified binary on SGX; thus, saving on the high cost and effort of porting the application [9]. We use GrapheneSGX [15] for our experiments in the *LibOS* mode.

***Native* Mode Performance:** As shown in Figure 2, the performance overhead increases by an average of 50% as we go from the Low to the Medium setting, and by an average of 13% from the Medium to the High setting because of the following reasons. We see an unusually high performance overhead for HashJoin as it implements an "equi-join" logic that needs multiple iterations over the entire hash table. The total number of EPC evictions increases by an average of 130% when the input size is increased from Low to Medium. On further increasing the input from Medium to High, the total number of EPC evictions increases by an average of 60%. As we increase the size from Low to Medium, dTLB misses increase by an average of 74%, and then by 19% as we go from Medium to High mainly due to EPC faults. Consequently, the total stall cycles increase by an average of 110% (Low to Medium), and by 20% while going from Medium to High.

***LibOS* Mode Performance:** The performance overhead increases by an average of 54% while going from Low to Medium, and by up to 18% while going from Medium to High (not shown). The total number of dTLB misses, walk cycles, and stall cycles increase by an average of 47%, 40%, and 28% as we go from Low to Medium, respectively. However, the same metrics drop by 26%, 84%, and 91% as we go from Medium to High, respectively. This is because in this setting, the overhead of the *LibOS* is somewhat hidden due to the long execution time of the benchmarks. The performance still drops in this setting (Medium to High) because of an increase in the total number of EPC faults (25% on an average).

V. Conclusion

We introduced SGXGauge, a benchmark suite for Intel SGX that captures a holistic view of the performance of applications running in such TEEs – this includes the impact of the EPC memory. SGXGauge contains diverse benchmarks that affect different components of SGX. We also performed an evaluation of the performance of SGX in *LibOS* mode and showed that there is a marked difference in behavior as the memory footprint crosses the EPC size limit.

Acknowledgement

We would like to thank the anonymous reviewers for their insightful comments. This work was funded in part by the Semiconductor Research Corporation (SRC) via grant IR-3053.

References

[1] "Academic Research — Intel Software Guard Extensions — Intel Software," https://software.intel.com/en-us/sgx/documentation/academic-research, 2019, (Accessed on 11/18/2019).

[2] V. Costan and S. Devadas, "Intel SGX Explained," *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016.

[3] X. Liu, W. Wang, L. Wang, X. Gong, Z. Zhao, and P.-C. Yew, "Regaining Lost Seconds: Efficient Page Preloading for SGX Enclaves," in *Proceedings of the 21st International Middleware Conference*, ser. Middleware '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 326–340. [Online]. Available: https://doi.org/10.1145/3423211.3425673

[4] M. Taassori and A. Sha, "VAULT : Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures," *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18)*, pp. 665–678, 2018.

[5] Sergei Arnautov and Bohdan Trach and Franz Gregor and Thomas Knauth and Andre Martin and Christian Priebe and Joshua Lind and Divya Muthukumaran and Dan O'Keeffe and Mark L. Stillwell and David Goltzsche and Dave Eyers and Rüdiger Kapitza and Peter Pietzuch and Christof Fetzer, "SCONE: Secure linux containers with intel SGX," in *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 689–703. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov

[6] H. Tian, Q. Zhang, S. Yan, A. Rudnitsky, L. Shacham, R. Yariv, and N. Milshten, "Switchless Calls Made Practical in Intel SGX," in *Proceedings of the 3rd Workshop on System Software for Trusted Execution*, ser. SysTEX '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 22–27. [Online]. Available: https://doi.org/10.1145/3268935.3268942

[7] M. Orenbach, P. Lifshits, M. Minkin, and M. Silberstein, "Eleos: ExitLess OS Services for SGX Enclaves," ser. EuroSys '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3064176.3064219

[8] O. Weisse, V. Bertacco, and T. Austin, "Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves," ser. ISCA '17. ACM, 2017, pp. 81–93.

[9] A. Hasan, R. Riley, and D. Ponomarev, "Port or Shim? Stress Testing Application Performance on Intel SGX," in *2020 IEEE International Symposium on Workload Characterization (IISWC)*, 2020, pp. 123–133.

[10] Y. Fu, E. Bauman, R. Quinonez, and Z. Lin, "Sgx-Lapd: Thwarting Controlled Side Channel Attacks via Enclave Verifiable Page Faults," in *RAID*, 2017.

[11] ——, "utds3lab/sgx-nbench: The nbench benchmark ported to SGX." https://github.com/utds3lab/sgx-nbench, 2019, (Accessed on 09/23/2019).

[12] BYTE, "Nbench," https://www.math.utah.edu/~mayer/linux/bmark.html, 1995, (Accessed on 09/23/2019).

[13] OpenSSL, "Openssl," https://www.openssl.org/, 2019, (Accessed on 12/07/2019).

[14] Intel, "intel/intel-sgx-ssl: Intel® Software Guard Extensions SSL," https://github.com/intel/intel-sgx-ssl, (Accessed on 06/23/2021).

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

[16] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, and S. Yan, "Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX," ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020.

[17] S. Yadalam, V. Ganapathy, and A. Basu, "SGXL: Security and Performance for Enclaves Using Large Pages," *ACM Trans. Archit. Code Optim.*, vol. 18, pp. 12:1–12:25, 2021.

[18] S. Mohapatra, "mohaps/libcatena: a blockchain written in C++ for learning purposes," https://github.com/mohaps/libcatena, 2019, (Accessed on 09/23/2019).

[19] Reto Achermann, "mitosis-project/mitosis-workload-btree: The BTree workload used for evaluation." https://github.com/mitosis-project/mitosis-workload-btree, September 2020, (Accessed on 10/03/2020).

[20] ——, "mitosis-project/mitosis-workload-hashjoin: The HashJoin workload used for evaluation." https://github.com/mitosis-project/mitosis-workload-hashjoin, September 2020, (Accessed on 10/03/2020).

[21] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," *SIGPLAN Not.*, vol. 48, no. 8, p. 135–146, Feb. 2013. [Online]. Available: https://doi.org/10.1145/2517327.2442530

[22] "memcached - a distributed memory object caching system," https://memcached.org/, 2019, (Accessed on 11/18/2019).

[23] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz, "XSBench - The Development and Verification of a Performance Abstraction for Monte Carlo Reactor Analysis," in *PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future*, Kyoto, 2014. [Online]. Available: https://www.mcs.anl.gov/papers/P5064-0114.pdf

[24] A. Bogus, *Lighttpd installing, compiling, configuring, optimizing, and securing this lightning-fast Web Server / Andre Bogus.* Birmingham, UK: Packt Publishing, 2008.

[25] M. A. Hearst, "Support vector machines," *IEEE Intelligent Systems*, vol. 13, no. 4, p. 18–28, Jul. 1998. [Online]. Available: https://doi.org/10.1109/5254.708428