Performance Analysis on DNA Alignment Workload with Intel SGX Multithreading

Lorenzo Brescia[1,*], Iacopo Colonnelli[1] and Marco Aldinucci[1]

[1]University of Turin, Computer Science Department, Alpha research group

Abstract

Data confidentiality is a critical issue in the digital age, impacting interactions between users and public services and between scientific computing organizations and Cloud and HPC providers. Performance in parallel computing is essential, yet techniques for establishing Trusted Execution Environments (TEEs) to ensure privacy in remote environments often negatively impact execution time. This paper aims to analyze the performance of a parallel bioinformatics workload for DNA alignment (Bowtie2) executed within the confidential enclaves of Intel SGX processors. The results provide encouraging insights regarding the feasibility of using SGX-based TEEs for parallel computing on large datasets. The findings indicate that, under conditions of high parallelization and with twice as many threads, workloads executed within SGX enclaves perform, on average, 15% faster than non-confidential execution. This empirical demonstration supports the potential of SGX-based TEEs to effectively balance the need for privacy with the demands of high-performance computing.

Keywords

Confidential computing, Parallel computing, Intel SGX, Gramine, Occlum

1. Introduction

In recent years, the awareness of the need for privacy has gained significant prominence. In the digital age, where information is predominantly stored and transmitted electronically, concerns regarding the protection of sensitive data have become increasingly prevalent. This confidential information can be extracted and reused without the knowledge or consent of the data owner, posing severe privacy risks. This issue is not confined to the interaction between individuals and digital services; It extends across various fields of scientific computing where data confidentiality is indispensable. Notable examples include bioinformatics, which processes DNA and genomic data; medical research that handles patient health records; epidemiology, particularly highlighted during the recent COVID-19 pandemic; and social sciences that address sensitive topics such as mental health, income levels, and political polarization. Economic considerations also drive the imperative to safeguard sensitive information. For instance, in economics, processing financial data for trading purposes necessitates stringent privacy measures. Similarly, in chemoinformatics, the discovery of drugs and molecular simulations, which possess significant commercial value, require robust data protection to prevent unauthorized access and exploitation.

For these reasons, it is imperative to adopt techniques that protect sensitive data at all stages. In scientific computing, private organizations often lack the computational power to perform their calculations. The simplest and most commonly used solution is outsourcing computation to a remote location by renting the necessary hardware resources. A typical example of this is cloud computing, where resources are allocated on demand, and an ecosystem exists to facilitate the execution of workloads seamlessly. Data protection is typically considered in two primary contexts: at rest (in storage) and in transit (during transmission over the network). However, it is less common to consider the vulnerability

CEUR-WS.org/Vol-3785/paper107.pdf

BigHPC2024: Special Track on Big Data and High-Performance Computing, co-located with the 3rd Italian Conference on Big Data and Data Science, ITADATA2024, September 17 – 19, 2024, Pisa, Italy.

*Corresponding author.

lorenzo.brescia@unito.it (L. Brescia); iacopo.colonnelli@unito.it (I. Colonnelli); marco.aldinucci@unito.it (M. Aldinucci)

https://alpha.di.unito.it/lorenzo-brescia/ (L. Brescia); https://alpha.di.unito.it/iacopo-colonnelli/ (I. Colonnelli); https://alpha.di.unito.it/marco-aldinucci/ (M. Aldinucci)

0009-0005-1147-496X (L. Brescia); 0000-0001-9290-2017 (I. Colonnelli); 0000-0001-8788-0829 (M. Aldinucci)

© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

of data during computation. Once a program starts executing on a remote machine, such as in cloud computing, there is often no control or protection over the data in the main memory. Confidential computing addresses this issue using trusted hardware to ensure data protection during execution. This approach breaks the chain of trust between the user and the external provider by introducing an additional entity in the trust process, the hardware manufacturer. This indirection step helps safeguard data while it is being processed, enhancing overall data security in outsourced computational environments. Figure 1 illustrates the entities involved and their relationships when a general user

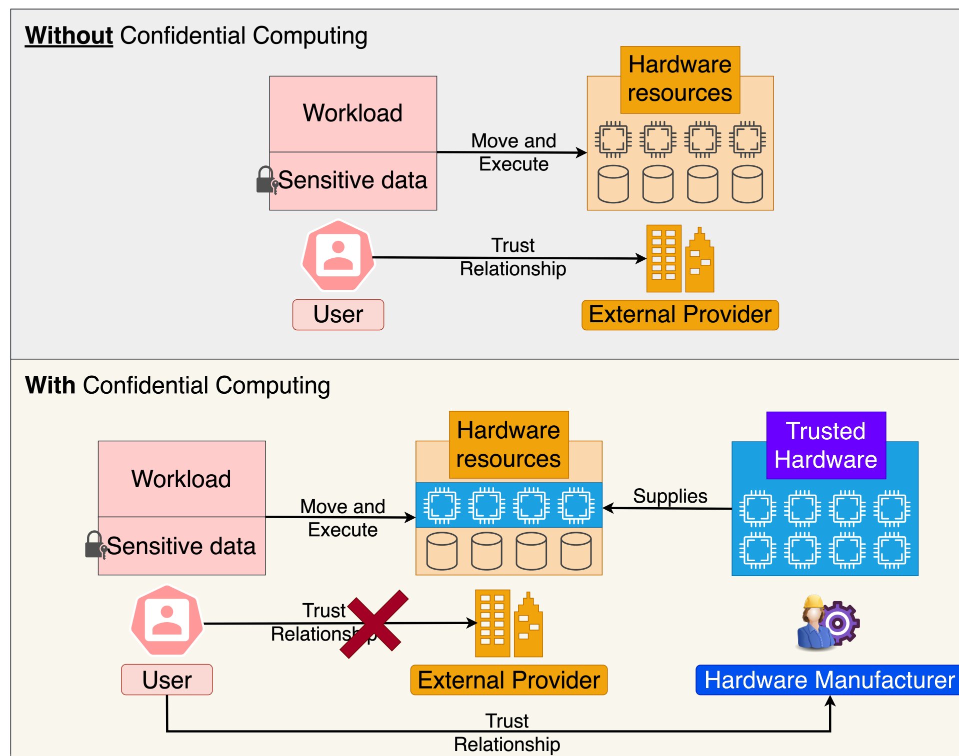

Figure 1: Remote computing scheme with or without confidential computing involved

utilizes a provider's remote resources. Without implementing confidential computing, the user transfers the computation to the provider. Even if the sensitive data is encrypted during transmission and on storage, it becomes vulnerable once it is decrypted for execution in the main memory. This exposure occurs because the data is no longer encrypted during processing, making it susceptible to risks in a multitenant environment, where potentially malicious workloads from other users may exist or if the provider is compromised or has malicious intent. In such scenarios, the user has no options; she has to unquestionably trust the provider, which is inherently untrusted. Confidential computing changes this dynamic by breaking the direct trust relationship between the user and the provider. Trusted hardware components, designed by the hardware manufacturer (e.g., CPU or GPU), incorporate specific features that ensure the confidentiality and integrity of the user's program during execution. This enables the user to establish an indirect trust relationship with the provider. Instead of trusting the provider directly, the user trusts the hardware manufacturer, which in turn supplies trusted components to the provider. This approach ensures that the user's data remains secure while being processed on the provider's infrastructure.

The purpose of this paper is to conduct a performance analysis on the use of Intel SGX processors as trusted hardware. The study is performed on an application called Bowtie2, which is a bioinformatics software. Section 2 explains all the necessary background: what Intel SGX CPUs are and how they can be exploited with Gramine and Occlum to facilitate their use. Furthermore, some reasons are given for the choice of Bowtie2 as workload to assess performance. Section 3 discusses related works considering other SGX frameworks besides Gramine and Occlum. In addition, an overview of previous

SGX performance studies in the High-Performance Computing (HPC) domain is provided. In Section 4, the configurations implemented to execute Bowtie2 in native and within SGX enclaves are explained. In Section 5, the results of the previously configured environment are illustrated, and finally, in Section 6, conclusions and possible future works are presented.

2. Background

2.1. Intel SGX

Intel Software Guard Extensions (SGX) [1] is a technology implemented in Intel processors designed to protect processes during execution by ensuring confidentiality and integrity of the main memory. Intel SGX extends the Instruction Set Architecture (ISA) with instructions that enable the creation of Trusted Execution Environments (TEEs) [2], referred as **enclaves** in Intel's terminology. These enclaves are secure memory regions that provide protection even against privileged system software, such as operating systems or hypervisors. Activating SGX features involves a non-trivial process. There are primarily two approaches to obtain this:

Rewriting application code involves modifying the application code using the libraries provided by Intel's Software Development Kit (SDK) [3] to manage enclaves. While this approach allows for granular control over what should be protected - down to the level of a single instruction - the effort required for the porting is considerable.

Using frameworks to execute existing applications aims to simplify application deployment by allowing them to run entirely within an enclave without significant rewriting. Several frameworks support this method, including Gramine and Occlum Library Operating System (LibOS), which facilitate the execution of legacy applications within enclaves.

Intel SGX has evolved, and the community recognizes two main versions: SGXv1 and SGXv2. These versions differ primarily in efficiency improvements and enclave size capacities, with SGXv2 supporting enclaves up to 512GB (against 128MB of SGXv1) and introducing Enclave Dynamic Memory Management (EDMM) [4]. EDMM allows dynamic allocation of enclave pages (EPCs) as needed, rather than requiring a predefined enclave size at startup time, although this feature can be complex and inefficient to implement. A notable capability of Intel SGX processors is the concurrent execution of the same enclave code using multiple threads. Each thread is associated with an EPC with type Thread Control Structure (TCS); this requires prior knowledge of the number of threads to ensure sufficient EPC allocation. Obviously, this requirement is alleviated when EDMM is enabled due to the capabilities of allocating EPC after the enclave's creation. Another key feature of Intel SGX is remote attestation, which allows a remote user to verify the correct instantiation of an enclave on an SGX processor. This is not the focus of our work; in short, the remote attestation process verifies the hash of the enclave and relies on Intel's certificates as the root of trust. There are principally two attestation schemes for SGX: Enhanced Privacy ID (EPID) [5] and Data Center Attestation Primitives (DCAP) [6].

2.2. Gramine

Gramine [7], known initially as Graphene [8], is a LibOS designed to enable unmodified Linux binaries to run within Intel SGX enclaves. The core purpose of a LibOS is to intercept system calls from an application and resolve them directly within user space whenever possible. Gramine extends this capability by integrating support for SGX, ensuring that the entire application, including the LibOS itself, operates within an SGX enclave transparently to the user. To execute an application with Gramine, the required effort is minimal and involves writing a manifest in a declarative manner. This manifest specifies all options necessary for the execution and customization of SGX features. Once the manifest is prepared, the workload can be executed using a set of commands from the Gramine toolchain. Although this LibOS was one of the first to support SGX, it remains highly competitive and continuously evolves to incorporate new SGX features, such as EDMM of SGXv2.

One of Gramine's most notable properties is its support for multiprocessing and related system calls, such as `fork`, `vfork`, `clone`, and `execve`. This support allows multiprocessing to be handled transparently, much like in non-SGX environments. For example, when a `fork` occurs, a second enclave is created, and the content is copied using message passing. Before this, a local attestation procedure is conducted between the enclaves, establishing a TLS secure channel for future communications. This method of handling multiprocessing is known as Enclave-Isolated Processes (EIP) (Figure 2a), where each enclave contains an instance of LibOS.

The EIP approach is inherently expensive in terms of execution time. Creating a process within an enclave is costly, and inter-enclave communication requires exchanging encrypted messages over a secure TLS channel. However, despite these disadvantages, the EIP method has significant advantages. The primary purpose of a LibOS with SGX integration is to facilitate the transition of workloads from an unsafe environment to an enclave. By supporting system calls like `fork` and adopting EIP for multiprocessing, Gramine allows applications that use multiple processes to be deployed quickly, with no additional effort than single-process applications. This ease of deployment is crucial for transitioning existing applications to secure SGX environments.

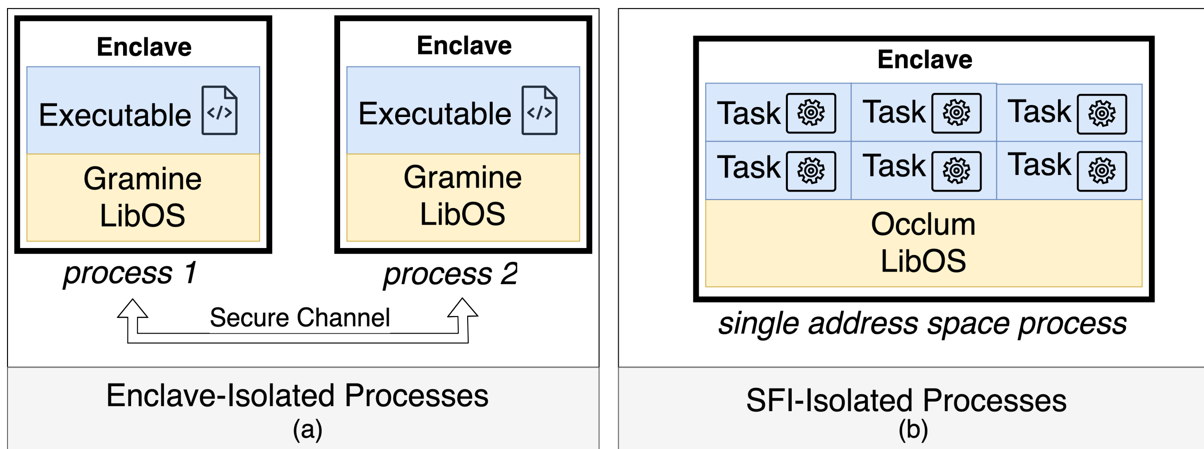

Figure 2: Different ways of handling parallelism using SGX enclaves. (a) Enclave-Isolated Processes (EIP), where each enclave is a separate process. (b) SFI-Isolated Processes (SIP), where a single enclave is used, and tasks are executed within a single address space.

2.3. Occlum

Occlum [9] is a toolchain that includes a LibOS designed to run applications inside SGX enclaves. To facilitate the transition of existing applications, the Occlum toolchain provides various utilities to prepare all necessary configurations for the building and running phases. Occlum aims to implement a LibOS that efficiently handles multitasking, a generic term referring to the parallel execution of multiple tasks. Occlum achieves this through a Software Fault Isolation (SFI) scheme called MPX-based, Multi-Domain SFI (MMDSFI). In the MMDSFI scheme, each process resides alongside the LibOS within the single address space of an enclave. This approach, known as SFI-Isolated Processes (SIPs) (Figure 2b), contrasts with the EIP scheme used by other LibOSes such as Gramine. The term "process" in the SIP scheme is somewhat misleading because the enclave maintains a single address space. Consequently, traditional process creation using the `fork` system call is not feasible, as it requires the child process to share the parent's address space. Instead, Occlum creates processes using the `spawn` system call, mapping each process to an SGX thread. This limitation means that applications relying on `fork`-like system calls cannot run within Occlum's LibOS without modification. However, the SIP scheme offers significant advantages, such as reducing the cost of setting up new enclaves (creation, local attestation, and duplication of the parent process state) and lowering the communication cost between enclaves. The primary disadvantage of the SIP scheme is the reduced portability of existing applications that utilize `fork`. To address this, intermediate work - potentially nontrivial, or even possible - may be

required to replace `fork` calls with `spawn`. This additional effort can be a barrier for some applications, but the overall benefits of the SIP scheme can make it a worthwhile trade-off for some use cases.

2.4. Bowtie2: DNA alignment

Bowtie2 [1] ([10], [11] and [12]) is a tool used for aligning sequencing reads to large genomes. During the alignment, the DNA sequences are compared to identify regions of similarity. This process is crucial for various applications, such as identifying genetic variations. Bowtie2 was selected as the performance evaluation workload in this paper for several logical considerations:

Memory-Intensive Application Bowtie2 is memory-intensive, making it an ideal candidate for evaluating the overhead associated with SGX, which aims to secure the main memory using encryption techniques.

Sensitive Data Analysis DNA sequence analysis involves highly susceptible data that must be protected, especially in remote environments like cloud providers. Using Bowtie2 helps assess the effectiveness of SGX in safeguarding this data.

Multithreading Performance Bowtie2's performance can be tuned through multithreading. While using multiple threads typically enhances performance, evaluating this in the context of SGX threads is particularly insightful, as the benefits may not be as straightforward due to the additional overhead and security constraints imposed by SGX.

3. Related work

3.1. Other SGX technologies

Besides Gramine and Occlum, there are other technologies whose purpose is to make it easy to run existing applications inside SGX enclaves:

- Haven [13] is one of the pioneering approaches to execute an entire LibOS within an SGX enclave, enabling the execution of unmodified Windows binaries securely.
- SCONE [14] ensures the confidentiality and integrity of containerized applications by leveraging SGX. Unlike LibOS, SCONE uses a thinner shielding layer to protect the application from the untrusted host OS. This means there is no entire LibOS within the enclave, but only some widely lighter shielding modules.
- Panoply [15] is another approach that tries to minimize the amount of code that needs to reside inside an SGX enclave. It introduces the concept of a micro-container, which encapsulates units of code and data isolated within SGX enclaves.
- SGX-LKL [16] enables Linux binaries to run inside SGX enclaves, similar to a LibOS approach but based on the Linux Kernel Library (LKL). It combines the flexibility of Linux with the security benefits of SGX, providing a lightweight solution for running Linux-based applications securely within enclaves.
- Ryoan [17] leverages SGX to process sensitive data securely in environments considered untrusted, both in terms of the application to run and the platform itself.

3.2. SGX performance analysis

Performance represents a significant concern in the realm of confidential computing. Although the goal is to achieve privacy, it is crucial not to compromise the execution time in chasing it. The study [18] conducted a performance evaluation using HPC benchmarks within SGX enclaves. The work included a comparison of performance between Gramine and Occlum, even if this comparison is inherently limited

[1] https://github.com/BenLangmead/bowtie2

due to Occlum's lack of support for multiprocessing, which is particularly relevant in HPC contexts. To address this limitation, our work focuses on evaluating a single real-world multithreaded workload rather than synthetic benchmarks. This approach ensures a fair comparison between Gramine and Occlum, providing valuable insights into their performance.

Another performance study [19] compares Intel SGX and AMD Secure Encrypted Virtualization (SEV) based-TEEs. Specifically, SCONE is employed to execute on SGX. HPC benchmarks have been used, encompassing traditional scientific computing, machine learning tasks, and graph analytics.

In our further recent work [20], the reference workload focused on the initial two steps of the Next Generation Sequencing (NGS) variant calling pipeline, which has been fully migrated to a cloud-based HPC environment [21]. Specifically, one of these steps involves the execution of Bowtie2 using Gramine.

4. Methods

This section outlines the setup of execution environments for the Bowtie2 DNA alignment bioinformatics workload. The configurations were designed to ensure fairness across different LibOSes environments (Gramine and Occlum). Only crucial aspects of the configuration files are presented for each setup. Both LibOSes were established using Dockerfiles, created based on the existing Docker images provided by the respective maintainers. A public GitHub repository[2] was established to provide insight into the configurations implemented for running in various environments. However, due to confidentiality concerns, it was not possible to publish the DNA reads input data.

4.1. Bare-metal

To use Bowtie2 on a native system, it is possible to easily utilize package managers such as Bioconda[3], which provides a distribution of bioinformatics software as a channel for the versatile Conda[4] package manager. However, in this study, the executables were built directly from the downloaded sources to facilitate fair comparisons between all execution environments (bare-metal, Gramine, and Occlum). In order to run Bowtie2, it is necessary to specify the basename of the index for the reference genome and the two files containing the paired-end reads (short DNA sequences). An example command for performing the alignment against the human hg38 genome is:

```
bowtie2 -S "out.sam" -x "Homo_sapiens_assembly38" \
        -1 "sample.r_1_val_1.fq.gz" -2 "sample.r_2_val_2.fq.gz" \
        -p num_of_threads
```

In this command, the `-x` option is used to specify the reference genome. The `-S` option designates the output file in `.sam` (Sequence Alignment/Map) format, and the `-1` and `-2` options are for the compressed paired-end reads in `.fq` (FASTQ) format. The `-p` option specifies the number of parallel threads to be used for searching; each thread runs on a different core, enabling all threads to find alignments in parallel.

4.2. Gramine

A manifest must be compiled to run an unmodified Linux binary inside an SGX enclave using Gramine. This manifest contains all the configuration information about the LibOS and the SGX enclave. In the Gramine toolchain, the `gramine-manifest` executable processes a manifest template, which can include Jinja[5] syntax for customization. Using this template simplifies the creation of the manifest and allows for more flexible configuration. To streamline the process of creating the manifest required to run Bowtie2 (`bow.manifest`), a Makefile was written that also includes the recipe below:

[2]https://github.com/lorenzobrescia/performance-SGX-Bowtie2
[3]https://bioconda.github.io
[4]https://docs.conda.io/en/latest/
[5]https://jinja.palletsprojects.com

```
bow.manifest: manifest.template
	gramine-manifest -Dthreads=num_of_threads $< >$@
```

As can it be observed from the previous recipe, a `manifest.template` must be prepared in order to generate `bow.manifest`. In the template file, all the arguments needed for execution are passed as environment variables in the following Gramine option:

```
loader.argv = ["/bowtie2-align-s", "-S", "/out.sam",
"-x", "/Homo_sapiens_assembly38", "-1", "/sample.r_1.fq.gz",
"-2", "/sample.r_2.fq.gz", "-p", "{{ threads }}"]
```

The options specified in the `manifest.template` are self-explanatory in relation to the bare-metal execution of Bowtie2. It is important to note that the `bowtie2-align-s` binary is run directly, rather than Bowtie2 itself. The latter is a Perl wrapper that selects the appropriate aligner to use. The wrapper is bypassed to simplify the process and ensure a smoother comparison with Occlum. For this reason, the `bowtie2-align-s` binary is executed directly. Consequently, `bowtie2-align-s` is set as the LibOS entry point in the `manifest.template`, meaning it is the code executed immediately after the enclave is ready:

```
libos.entrypoint = "/bowtie2-align-s"
```

For handling the EDMM feature, Jinja syntax was used, still within `manifest.template`. If the environment variable `edmm` is set to 1, the feature is enabled; otherwise, it is not. This configuration also allows specifying the size of the enclave and the number of threads available inside the enclave. The semantics of these configurations differ depending on whether the EDMM function is enabled. With EDMM enabled, `sgx.enclave_size` refers to the maximum size the enclave can reach, and `sgx.max_threads` represents the number of TCS EPCs allocated before execution. If more threads are required during execution, additional TCS pages will be created on demand. If EDMM is disabled, the options are straightforward: `sgx.enclave_size` sets the fixed size of the enclave, and `sgx.max_threads` specifies the total number of threads that can be used, both set at the time of enclave creation. The following snippet implements what has just been described:

```
{% if env.get('edmm', 0) == '1' %}
    sgx.edmm_enable = true
    sgx.enclave_size = "max_enclave_size"
    sgx.max_threads = number_of_preallocated_threads
{% else %}
    sgx.edmm_enable = false
    sgx.enclave_size = "enclave_size"
    sgx.max_threads = max_number_of_threads
{% endif %}
```

Once the `bow.manifest` is obtained from the Makefile, the SGX manifest (`bow.manifest.sgx`) is also created using the Gramine toolchain, and finally, the application is run simply with the command:

```
gramine-sgx bow
```

4.3. Occlum

To launch a Linux executable inside Occlum, it is necessary to create a workspace that includes the LibOS image that will host the executable inside the enclave. Occlum provides a comprehensive toolchain to facilitate the deployment of this instance. First, the workspace is created using the `occlum init` command. Subsequently, the file system inside the LibOS must be configured. This configuration is achieved using the `copy_bom` tool, where an input file `bow.yaml` specifies that the `bowtie2-align-s` executable is to be mounted inside the `/bin` folder. This process ensures the executable is correctly placed within the LibOS image for execution inside the SGX enclave. To achieve what has just been described, the file `bow.yaml` must contain the following configuration:

Table 1
Experiment configurations. Both LibOSes have been created starting from the authors' public Docker images: Gramine (gramineproject/gramine:stable-focal) and Occlum (occlum/occlum:0.30.0-ubuntu20.04). Some options are N/A because they cannot be specified in the referenced LibOS.

No EDMM	Gramine [Docker]	Occlum [Docker]
Enclave size	8 GB	8 GB
Number of threads	38	38
EDMM		
Init enclave size	N/A	512 MB
Max enclave size	32 GB	32 GB
Init number of threads	32	32
Max number of threads	N/A	38

```
targets:
  - target: /bin
    copy:
      - files:
        - bowtie2-align-s
```

Next, it is necessary to configure the `Occlum.json` file, which describes all the characteristics of the SGX enclave. This configuration includes essential information. In cases where EDMM is not active, it is possible to specify the enclave size and the maximum number of threads in this way:

```
"resource_limits": {
    "user_space_size": "enclave_size",
    "max_num_of_threads": num_max_of_threads
}
```

Instead, the following options should be additionally specified to configure EDMM:

```
"resource_limits": { ...
    "user_space_max_size": "enclave_max_size",
    "init_num_of_threads":  num_of_preallocated_threads
}
```

Thus, a single `Occlum.json` file can turn EDMM features on or off. Consequently, two different `.json` configuration files were created to delineate the desired features for the experiments. The `occlum build` command is used to construct the Occlum SGX enclave and generate its associated file system image according to the specifications in the `Occlum.json` configuration file. Finally, to run Bowtie2, the following command must be executed, specifying all the necessary options:

```
occlum run bowtie2-align-s -x "/Homo_sapiens_assembly38" \
    -1 "sample.r_1_val_1.fq.gz" -2 "sample.r_2_val_2.fq.gz" \
    -S "out.sam" -p num_of_threads
```

5. Results

To assess the performance of Bowtie2 across various environments, we utilized the configurations detailed in Table 1. The experimental setup involved a machine powered by an Intel Xeon Gold 6346 CPU operating at 3.10 GHz, with an available memory capacity of approximately 400 GB RAM.

Figure 3 illustrates the execution times of Bowtie2 under various configurations for both small and large input sizes. Each experiment was performed 10 times. Since no significant variance or outliers were observed, the mean value was considered representative of the configurations. A small input size refers to aligning approximately $10,000$ reads, while a large input size involves aligning about

3 million reads. As shown in Figure 3a, native execution completes rapidly within seconds for small workloads. However, both LibOSes exhibit poor performance in this scenario, although, as can be noticed, Occlum outperforms Gramine. Furthermore, there is a lack of scalability: increasing the number of threads does not significantly enhance execution time, even in the bare-metal configuration. Enabling EDMM generally leads to a stable and acceptable increase in execution times across most cases, except when Bowtie2 necessitates 32 threads on Gramine. The excessive overhead observed may result from the dynamic management of the TCS enclave pages. As indicated in Table 1, Gramine's EDMM configuration preallocates 32 threads. Although Bowtie2 operates with exactly 32 threads, Gramine requires at least three additional threads for managing inter-process communication (IPC), asynchronous tasks, and secure TLS communication within the LibOS, and the overhead likely arises from the effort needed to allocate these supplementary threads. These findings discourage the adoption of SGX technologies due to the unacceptable overhead compared to the native case and the absence of scalability. However, it is worth noting that scalability is also lacking in the native case. Consequently, the experiment was repeated with the same configurations detailed in Table 1 but applied to a much larger number of sequences, and the results are depicted in Figure 3b. Some patterns evident in the small input size scenario are also observed here. For instance, in the bare-metal environment, execution times are significantly faster compared to those in the LibOSes, and Gramine's dynamic thread management severely impacts performance when Bowtie2 uses 32 threads. However, unlike the small input size case, the plot indicates some scalability. All configurations exhibit good scaling, with Occlum performing slightly better than Gramine again.

The critical consideration is justifying using trusted hardware techniques such as Intel SGX. Although SGX provides privacy guarantees, it also significantly increases execution times. In the case illustrated in Figure 3a, the technology appears unfeasible due to the uneven trade-off between overhead and privacy. Conversely, 3b suggests that if the application is parallelizable, good scaling can be achieved even with SGX computations as the number of threads increases. In detail, empirical evidence indicates that running Bowtie2 on bare metal and then re-running the same workload on SGX with twice as many threads often increases performance. This effect is further highlighted in Figure 4, which presents

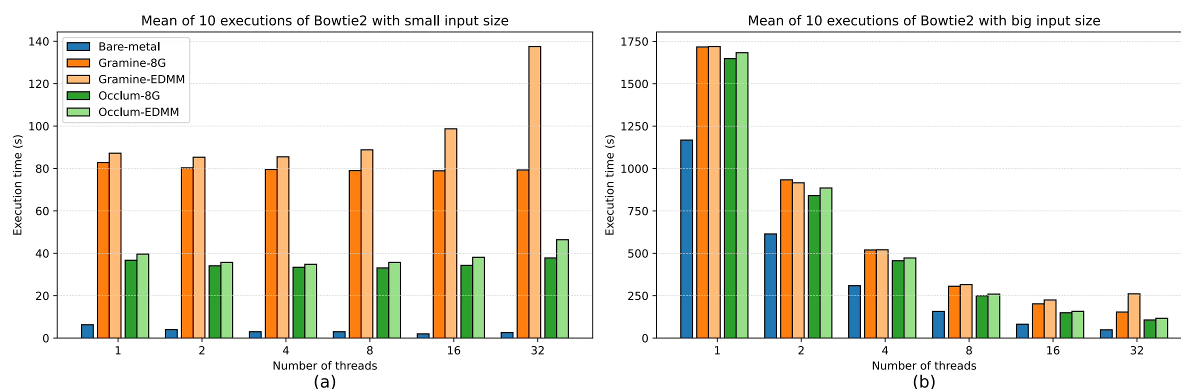

Figure 3: Comparison of execution times of Bowtie2 on different environments. The results are the mean of 10 executions. (a) The input for Bowtie2 consists of $9,997$ reads (small input size). (b) The input for Bowtie2 consists of $2,886,533$ reads (big input size)

scalability comparison plots. Figure 4a demonstrates the performance gains of bare-metal execution when the number of threads is doubled. Figures 4b and 4c provide comparisons between bare-metal and Gramine, and between bare-metal and Occlum, respectively, under the same conditions. As observed, using SGX doubling threads often results in a performance gain compared to the native case. In some instances, the gain can be substantial; for instance, Occlum shows a 38.96% performance increase when using two threads compared to single-thread in the bare-metal setup. Nevertheless, performance gains are not always achievable, particularly when approaching the scalability limits of the problem. For example, in this bioinformatics workload, the performance gain from 16 to 32 threads is marginal, even in the native case, yielding just a 67% improvement compared to the average 95% increase. Specifically,

when comparing 16 native threads to 32 threads in Gramine, there is a performance decrease of 46%, while Occlum shows a decrease of 23% in the same conditions. However, excluding the latter case, SGX with twice as many threads not only eliminates the overhead compared to non-confidential native execution but also achieves, on average, a 15% performance gain.

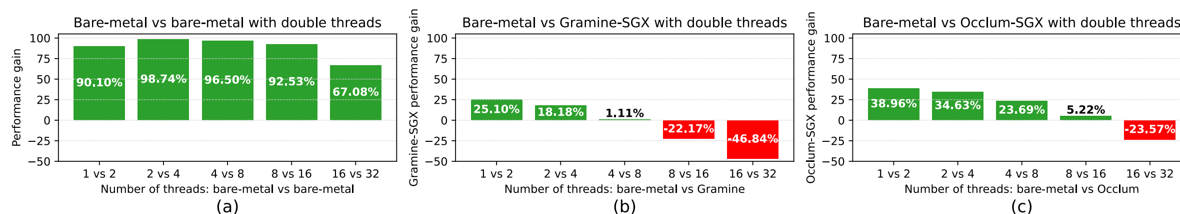

Figure 4: Bowtie2 with big input size: comparison of the scalability of different environments in respect of the bare-metal execution. (a) Performance gain in bare-metal VS bare-metal with double threads. (b) Gramine performance gain in bare-metal VS Gramine with double threads. (c) Occlum performance gain in bare-metal VS Occlum with double threads

A final consideration that emerges from the experiments is that Occlum generally outperformed Gramine in terms of execution time and scalability. However, it is essential to note that Gramine supports multiprocess applications, unlike Occlum, which makes Gramine particularly attractive for the portability of legacy workloads. A similar argument applies to the EDMM feature. Although, on average, EDMM increases execution time, it simplifies the configuration of LibOSes by eliminating the need to estimate the memory footprint, thus facilitating the portability of existing applications.

6. Conclusion and future work

This study provides a foundational analysis of Intel SGX's performance for parallel executions. Introductory empirical observations obtained in our study offer essential insights into the feasibility of employing SGX for this kind of execution. The results indicate that doubling the threads almost invariably improves performance compared to an environment without hardware encryption techniques. This scenario is entirely plausible in remote environments managed by external providers, as users typically offload computations to remote systems due to insufficient local computational resources. In addition, these findings suggest that SGX could effectively mitigate the inherent overhead associated with encryption, thereby preserving privacy at runtime.

Future work may expand this performance analysis in two directions. The first direction involves a deeper exploration of SGX technologies as highlighted in Section 3.1, and a broader examination of other types of hardware that enable the establishment of a TEE, such as AMD SEV or Intel Trust Domain Extensions (TDX). The second direction focuses on analyzing multiprocess applications extensively designed for HPC centers, extending beyond bioinformatics to encompass more general applications. By pursuing these two avenues, future research can provide a more comprehensive understanding of the capabilities and limitations of various hardware-based security technologies in different computational environments.

Acknowledgments

This work was supported by the Spoke 1 "FutureHPC & BigData" of ICSC - Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing, funded by European Union - NextGenerationEU.

References

[1] C. Victor, D. Srinivas, Intel SGX Explained, 2016. URL: https://eprint.iacr.org/2016/086, Accessed: 2024-07.

[2] M. Sabt, M. Achemlal, A. Bouabdallah, Trusted Execution Environment: What It is, and What It is Not, in: 2015 IEEE Trustcom/BigDataSE/ISPA, volume 1, IEEE, 2015, pp. 57–64. doi:10.1109/Trustcom.2015.357.

[3] Intel, Intel Software Guard Extensions (Intel SGX) SDK for Linux* OS, 2024. URL: https://download.01.org/intel-sgx/latest/linux-latest/docs, Accessed: 2024-07.

[4] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, C. Rozas, Intel software guard extensions (intel sgx) support for dynamic memory management inside an enclave, in: Proceedings of the Hardware and Architectural Support for Security and Privacy 2016, Association for Computing Machinery, 2016. doi:10.1145/2948618.2954331.

[5] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, F. Mckeen, et al., Intel software guard extensions: EPID provisioning and attestation services, 2016. URL: https://community.intel.com/legacyfs/online/drupal_files/managed/57/0e/ww10-2016-sgx-provisioning-and-attestation-final.pdf, Accessed: 2024-07.

[6] V. Scarlata, S. Johnson, J. Beaney, P. Zmijewski, Supporting third party attestation for intel sgx with intel data center attestation primitives, 2018. URL: https://www.intel.com/content/dam/develop/external/us/en/documents/intel-sgx-support-for-third-party-attestation-801017.pdf, Accessed: 2024-07.

[7] C.-C. Tsai, D. E. Porter, M. Vij, Graphene-SGX: a practical library OS for unmodified applications on SGX, in: Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference, USENIX Association, 2017, pp. 645–658.

[8] C.-C. Tsai, K. S. Arora, N. Bandi, B. Jain, W. Jannen, J. John, H. A. Kalodner, V. Kulkarni, D. Oliveira, D. E. Porter, Cooperation and security isolation of library OSes for multi-process applications, in: Proceedings of the Ninth European Conference on Computer Systems, Association for Computing Machinery, Amsterdam The Netherlands, 2014, pp. 1–14. doi:10.1145/2592798.2592812.

[9] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, S. Yan, Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX, in: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Association for Computing Machinery, 2020, pp. 955–970. doi:10.1145/3373376.3378469.

[10] B. Langmead, C. Trapnell, M. Pop, S. L. Salzberg, Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, Genome Biology 10 (2009) R25. doi:10.1186/gb-2009-10-3-r25.

[11] B. Langmead, S. L. Salzberg, Fast gapped-read alignment with Bowtie 2, Nature Methods 9 (2012) 357–359. doi:10.1038/nmeth.1923.

[12] B. Langmead, C. Wilks, V. Antonescu, R. Charles, Scaling read aligners to hundreds of threads on general-purpose processors, Bioinformatics 35 (2019) 421–432. doi:10.1093/bioinformatics/bty648.

[13] A. Baumann, M. Peinado, G. Hunt, Shielding Applications from an Untrusted Cloud with Haven, in: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, volume 33, Association for Computing Machinery, 2014, pp. 267–283. doi:10.1145/2799647.

[14] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, C. Fetzer, SCONE: Secure Linux Containers with Intel SGX, in: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, USENIX Association, 2016, pp. 689–703.

[15] S. Shweta, L. T. Dat, T. Shruti, S. Prateek, Panoply: Low-TCB Linux Applications With SGX Enclaves, NDSS Symposium (2017).

[16] C. Priebe, D. Muthukumaran, J. Lind, H. Zhu, S. Cui, V. A. Sartakov, P. Pietzuch, SGX-LKL: Securing the Host OS Interface for Trusted Execution, arXiv preprint arXiv:1908.11143 (2019).

[17] T. Hunt, Z. Zhu, Y. Xu, S. Peter, E. Witchel, Ryoan: A Distributed Sandbox for Untrusted Computa-

tion on Secret Data, ACM Trans. Comput. Syst. 35 (2018) 533–549. doi:10.1145/3231594.

[18] S. Miwa, S. Matsuo, Analyzing the Performance Impact of HPC Workloads with Gramine+SGX on 3rd Generation Xeon Scalable Processors, in: Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, SC-W '23, Association for Computing Machinery, 2023, pp. 1850–1858. doi:10.1145/3624062.3624267.

[19] A. Akram, A. Giannakou, V. Akella, J. Lowe-Power, S. Peisert, Performance Analysis of Scientific Computing Workloads on General Purpose TEEs, in: 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2021, pp. 1066–1076. doi:10.1109/IPDPS49936.2021.00115.

[20] L. Brescia, M. Aldinucci, Secure Generic Remote Workflow Execution with TEEs, in: Proceedings of the 2nd Workshop on Workflows in Distributed Environments, WiDE '24, Association for Computing Machinery, 2024, pp. 8–13. doi:10.1145/3642978.3652834.

[21] A. Mulone, S. Awad, D. Chiarugi, M. Aldinucci, Porting the Variant Calling Pipeline for NGS data in cloud-HPC environment, in: 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), 2023, pp. 1858–1863. doi:10.1109/COMPSAC57700.2023.00288.