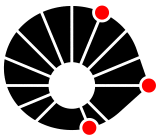

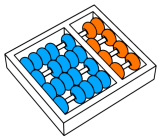

# Análise de desempenho do uso de Enclaves com Intel SGX

*Lucas Andrade* *Edson Borin*

Relatório Técnico - IC-PFG-21-15
Projeto Final de Graduação
2021 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

# Análise de desempenho do uso de Enclaves com Intel® SGX

Lucas B. Andrade Edson Borin*

**Resumo**

Este relatório técnico apresenta uma análise de desempenho de aplicações executadas de forma segura sobre a plataforma Intel SGX, presente em alguns processadores da fabricante, utilizando a biblioteca Open Enclave. Realizou-se experimentos com aplicações do pacote Madagascar para obter dados a fim de se construir hipóteses e então utilizou-se *benchmarks* para validar as hipóteses e caracterizar melhor as diferenças no desempenho das aplicações executadas com ou sem a plataforma. Concluiu-se que a operação cujo desempenho foi mais afetado ao rodar sobre a plataforma de segurança foi a busca de páginas de memória que não estavam na *cache* do processador e que as operações que não fazem acesso à memória não sofrem impacto em seu desempenho.

## 1 Introdução

Com as recentes mudanças nas políticas relacionadas ao tratamento de dados digitais, como a criação da Lei Geral de Proteção de Dados (LGPD), além da forte popularização dos serviços de computação em nuvem, vê-se ainda mais forte a necessidade de tecnologias que permitem realizar computação de forma segura. Nesse contexto, os Ambientes de Execução Confiáveis, ou TEEs[1], também chamados de Enclaves, se tornam um ponto de atenção, pois podem tornar a execução de um código mais segura, além de proteger os dados que estão sendo processados.

O *Intel Software Guard Extension* (Intel® SGX)[2] é uma tecnologia proprietária de *hardware* da Intel, projetada para proteger os dados e o código de aplicações, evitando modificações e acesso indevido a ambos. Desenvolvedores podem criar áreas de execução na memória que têm seu código e dados criptografados, os quais são decodificados em tempo de execução. Provendo, assim, uma interface de programação para criar um Enclave, implementada em diversos processadores da fabricante. A biblioteca *Open Enclave*[3] procura prover uma interface de programação única e simplificada para tirar proveito de diferentes plataformas de TEE, como o Intel SGX e o ARM TrustZone.

O objetivo deste trabalho foi investigar o impacto que o uso destas tecnologias de segurança traz ao desempenho de algumas aplicações e quais as características de execução delas que sofrem mais impacto quando executadas utilizando TEEs.

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

[1] Do inglês: *Trusted Execution Environments*.

[2] `https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html`

[3] `https://openenclave.io/sdk/`

## 2 Intel SGX e Open Enclave SDK

O Intel SGX, introduzido em 2015, adicionou um conjunto de instruções à arquitetura dos processadores da família *Core*[4], de codinome *Skylake*, com o objetivo de habilitar a programação de Enclaves, protegendo-os contra possíveis ataques. Por ser uma tecnologia de *hardware*, consegue proteger até mesmo contra o sistema operacional.

O modelo de programação de Enclaves é dividido em duas partes, uma parte confiável e uma não confiável. A parte não confiável pode realizar comunicação com todo o sistema, além de criar e configurar a execução de Enclaves, os quais constituem a parte confiável do sistema.

A comunicação entre as partes confiável e não confiável se dá através das chamadas de funções denominadas ECALL e OCALL, sendo que ECALL é o nome das chamadas da parte não confiável para a parte confiável, enquanto OCALL nomeia o inverso. Desta forma, a segurança da aplicação como um todo também depende dessa interface. A interface é definida utilizando da *Enclave Definition Language* (EDL). Essa interface de comunicação é ilustrada na Figura 1.

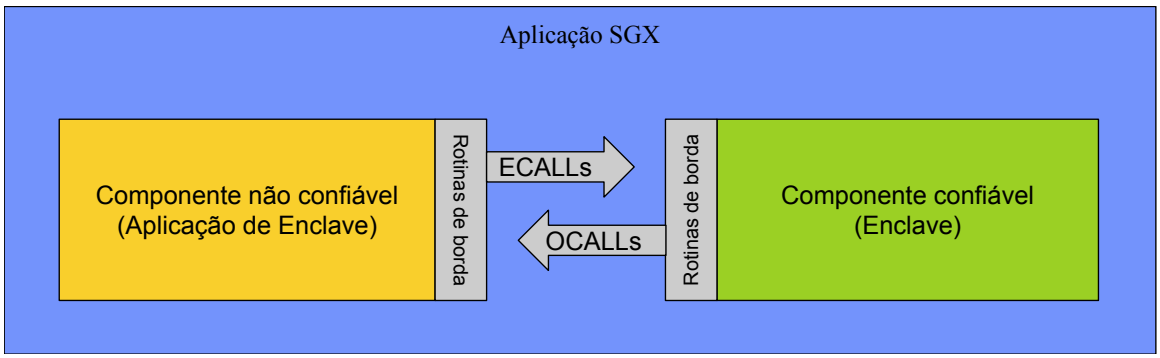


Figura 1: Modelo de programação de Enclaves [2].

A parte confiável, que executa em um Enclave, tem seu espaço de memória reservado no qual a memória é criptografada e decodificada em tempo de execução pelo processador. O espaço de memória em que as Enclaves são executadas é chamado de *Enclave Page Cache* (EPC), que não deve ser confundido com a memória *cache* do processador. A EPC faz parte da memória reservada do processador (PRM) [3]. A organização de memória é ilustrada na Figura 2

Uma aplicação que deseja utilizar Enclaves, desta maneira, precisa alocar um Enclave e então utilizá-lo para realizar as chamadas de ECALLs. A alocação (ou criação) de um Enclave no *Open Enclave* é feita utilizando a função `oe_create_enclave`. Durante a criação de um Enclave, deve-se determinar um número inicial de páginas EPC, as quais são alocadas utilizando instruções EADD e EEXTEND. Caso o Enclave necessite de mais memória, as mesmas instruções são chamadas para a alocação de novas páginas EPC. Assim, o tempo de criação de um Enclave deve ser proporcional à memória alocada.

[4] `https://www.intel.com.br/content/www/br/pt/products/details/processors/core.html`

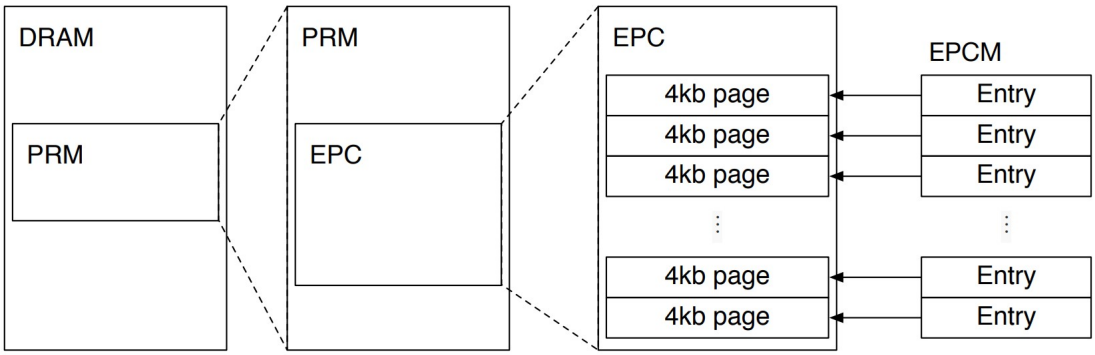


Figura 2: A memória dos Enclaves se localiza nas EPCs, as quais fazem parte da PRM e tem o tamanho de 4KB [3].

O tempo de execução de uma aplicação que utiliza do SGX então é composto pelo tempo de construção do Enclave, somado ao tempo de execução da lógica da aplicação e ao tempo gasto na transição de contexto entre a Enclave e a aplicação.

## 3 Metodologia

Com o objetivo de investigar o impacto no desempenho de uma aplicação ao executá-la em um TEE, foram escolhidas aplicações do pacote *Madagascar*[5], para serem executadas utilizando do Intel SGX e da biblioteca *Open Enclave SDK*. Para isso, comparou-se o tempo de execução das aplicações com e sem o uso destas tecnologias de segurança.

Para executar as aplicações utilizando a biblioteca *Open Enclave*, a interface definida é que tudo o que era executado na função `main` da aplicação foi transferido para uma outra função que passa a ser chamada pela `main`, em todos os casos foi dado o nome `run_app` por conveniência. Assim `run_app` é a única ECALL e as OCALLs são as funções da *stdlibc*, definidas e implementadas pela biblioteca. Os códigos fonte utilizados e instruções de como utilizar encontram-se no repositório do LMCAD[6]. Estes foram construídos seguindo o exemplo disponibilizado pela biblioteca *Open Enclave*[7].

Antes de avaliar o desempenho, é necessário verificar a corretude dos códigos sendo executados dentro do Enclave. Tendo os resultados corretos pode-se iniciar a verificação do desempenho dessas aplicações ao executá-las normalmente e também no Enclave. Para avaliar o desempenho das aplicações, o código foi modificado para incluir o cálculo do tempo de execução e a saída deste tempo em um arquivo.

Para a medição de tempo foi utilizada a chamada de sistema `gettimeofday` do sistema operacional Linux, a qual possui precisão de microssegundos. Assim foram medidos: o

[5] `https://reproducibility.org/wiki/Main_Page`
[6] `https://github.com/lmcad-unicamp/SGX-Performance-Experiments`
[7] `https://github.com/openenclave/openenclave/tree/master/samples/helloworld`

tempo de execução da aplicação (chamada a `run_app`) em execução normal, o tempo de criação do Enclave, o tempo de execução de `run_app` no Enclave já criado e o tempo total de execução da aplicação que utiliza Enclave, o que inclui a criação do Enclave, a geração de chaves para a ela, a execução da ECALL `run_app`, e a finalização do Enclave.

As aplicações do pacote *Madagascar* foram analisadas para identificar as características de execução que sofriam maior impacto no desempenho quando executadas no TEE. Buscou-se analisar as aplicações cuja razão entre o tempo de execução no Enclave e o tempo de execução normal eram as maiores ou as menores dentre as aplicações executadas. Estas apontariam mais facilmente quais características impactam mais fortemente e quais tem o menor impacto no desempenho da aplicação.

Além das aplicações do pacote *Madagascar* foram construídos *benchmarks*, os quais têm características de execução mais específicas, a fim de se validar as hipóteses elaboradas ao analisar-se a execução das aplicações acima.

A Tabela 1 descreve as configurações do computador utilizado nos experimentos.

| | |
|---|---|
| Sistema Operacional | Ubuntu 18.04 |
| Processador | Intel Core i7 7700HQ |
| Memória RAM | DDR4 16HGB (2x8GB) @ 2400MHz |
| Armazenamento | 256GB M.2 SSD |
| BIOS | X580VD.317 |
| Driver Intel SGX (apt)[8] | libsgx-enclave-common 2.13.103.1-bionic1<br>libsgx-enclave-common-dev 2.13.103.1-bionic1 |
| Versão do SGX-DCAP[9] | 1.10 |
| Open Enclave (apt) | open-enclave 0.15.0 |
| Madagascar[10] | 3.1.1 |
| Compilador utilizado | gcc 7.5.0 |

Tabela 1: Configurações do computador utilizado para realizar os experimentos

Para reproduzir os experimentos deve-se, primeiramente, instalar os pré-requisitos citados na Tabela 1, seguindo o tutorial do *Open Enclave*[11] e o tutorial de instalação do *Madagascar*[12] e então fazer o clone do repositório. O tempo de execução dos testes realizados é obtido executando-se o comando `bash run.sh` em cada um dos diretórios de interesse, sendo esses: `Teste-fora`, `Teste-dentro-enclave`, `Benchmarks/fora` e `Benchmarks/dentro`. O *script* realiza a compilação, as executa 10 vezes armazenando os tempos de execução num arquivo chamado `times.txt`, localizado dentro do diretório de cada aplicação pertinente. Os gráficos podem ser obtidos ao substituir os dados presentes nos arquivos `*.ipynb` no diretório `data` pelos novos dados obtidos e executando os *scripts* destes arquivos novamente.

[8] `https://download.01.org/intel-sgx/sgx_repo/ubuntu` bionic main

[9] `https://download.01.org/intel-sgx/sgx-dcap/1.10/linux/distro/ubuntu18.04-server/`

[10] `https://sourceforge.net/projects/rsf/files/madagascar/madagascar-3.1/`

[11] `https://github.com/openenclave/openenclave/blob/master/docs/GettingStartedDocs/install_oe_sdk-Ubuntu_18.04.md`

[12] `https://reproducibility.org/wiki/Installation`

# 4 Resultados

## 4.1 Aplicações do pacote *Madagascar*

As seguintes aplicações do pacote Madagascar foram escolhidas para os testes: *sfwi, sfeikonal, sfhelicon, sfkirmod, sfmf, sfmig2, sfpwd, sfricker1, sfsigmoid, sfslice, sfsnr2 e sfunif2.*

A partir da execução dos passos mencionados na Seção 3, as aplicações acima foram modificadas para serem executadas dentro do Enclave SGX. As saídas foram verificadas e apenas a aplicação *sfwi* não obteve êxito após a modificação. O motivo desta aplicação não ser executada corretamente não foi investigado. Tendo as saídas devidamente verificadas, cada aplicação foi executada 10 vezes, obtendo-se o tempo de execução que estas tinham com o código sem modificações e o tempo de execução após as modificações. Os resultados obtidos para o tempo de inicialização e para o tempo de execução da lógica das aplicações encontram-se nas figuras 3 e 4, respectivamente.

### 4.1.1 Tempo de inicialização

Cada aplicação possui um número fixo de páginas de memória EPC alocadas para a inicialização da sua Enclave. Conhecendo esse valor de antemão, mediu-se o tempo de inicialização da Enclave para cada uma das aplicações testadas e gerou-se o gráfico da Figura 3.

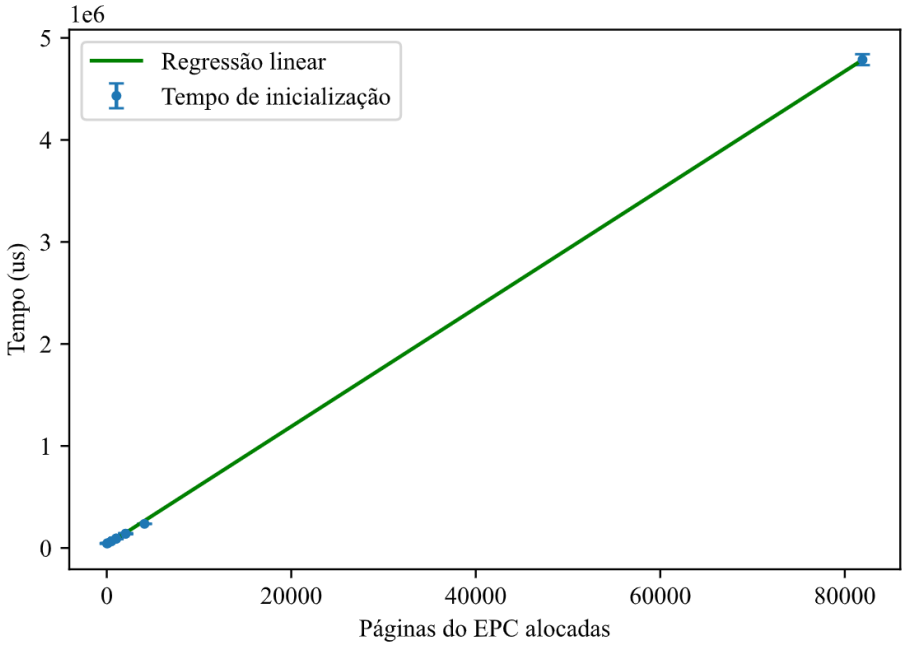


Figura 3: Tempo de inicialização em função do número de páginas de memória alocadas.

Observa-se comportamento linear na relação entre o número de páginas de memória alocadas e o tempo de inicialização da aplicação. A documentação do SGX [2] cita que na inicialização são executadas instruções do tipo EADD e EEXTEND a fim de se alocar

as páginas requisitadas, o que provavelmente está gerando esse comportamento linear. A partir da equação obtida pela regressão linear, pôde-se obter o tempo de inicialização de um Enclave sem memória alocada inicialmente de aproximadamente $29,75ms$ com taxa de crescimento de cerca de $58\mu s/página\ alocada$.

### 4.1.2 Tempo de execução

Além do tempo de inicialização das aplicações foi obtido o tempo de execução da lógica da aplicação. Os resultados obtidos são apresentados na Figura 4.

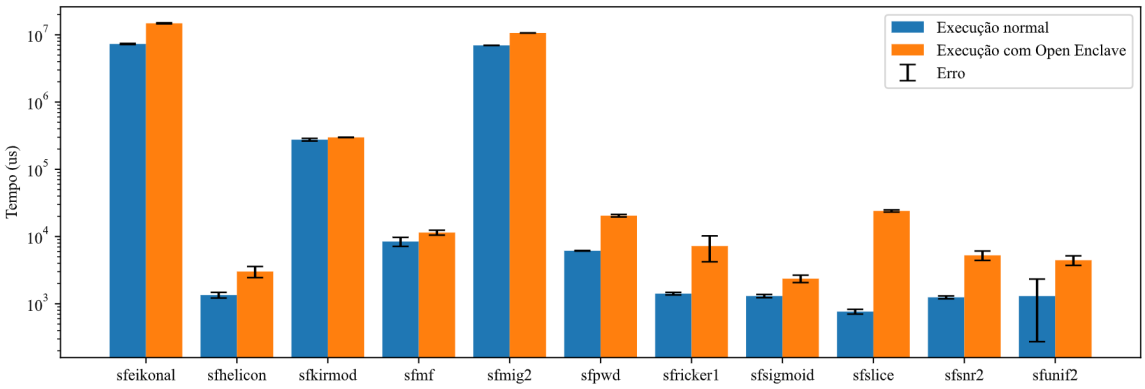


Figura 4: Tempo de execução médio das aplicações.

Pode-se observar que em todos os casos o tempo de execução com *Open Enclave* é maior, como é esperado, porém existem alguns casos em que a razão entre os tempos de execução é muito maior do que outros. Percebe-se também que houve um caso que foi pouco afetado pela execução com *Open Enclave*. A Figura 5 ilustra melhor a diferença entre o tempo de execução dos dois casos ao mostrar a razão entre os valores.

A aplicação que teve maior impacto foi a *sfslice* e a que teve o menor foi a *sfkirmod*. Ao verificar o código da aplicação *sfslice* pode-se ver que o laço principal da aplicação realiza operações de leitura e escrita em disco com poucas operações aritméticas, logo o tempo de execução deve ser dominado pelo desempenho das operações do disco.

No caso da aplicação *sfkirmod*, no laço principal faz-se a filtragem de um sinal utilizando da *Fast Fourier Transform*, que realiza diversas operações de leitura e escrita em memória junto a operações aritméticas e de controle de fluxo. Outra aplicação que teve o desempenho bastante impactado foi *sfricker1*, sendo 5,1 vezes mais lenta ao executar na Enclave. No laço principal desta aplicação também se faz a filtragem de um sinal, porém o resultado é bem diferente do que se esperava, dada a característica aparentemente similar à de *sfkirmod*.

## 4.2 *Benchmarks*

A fim de obter mais informações sobre características que causam maior impacto no desempenho das aplicações foram construídos *benchmarks*, tendo características de execução

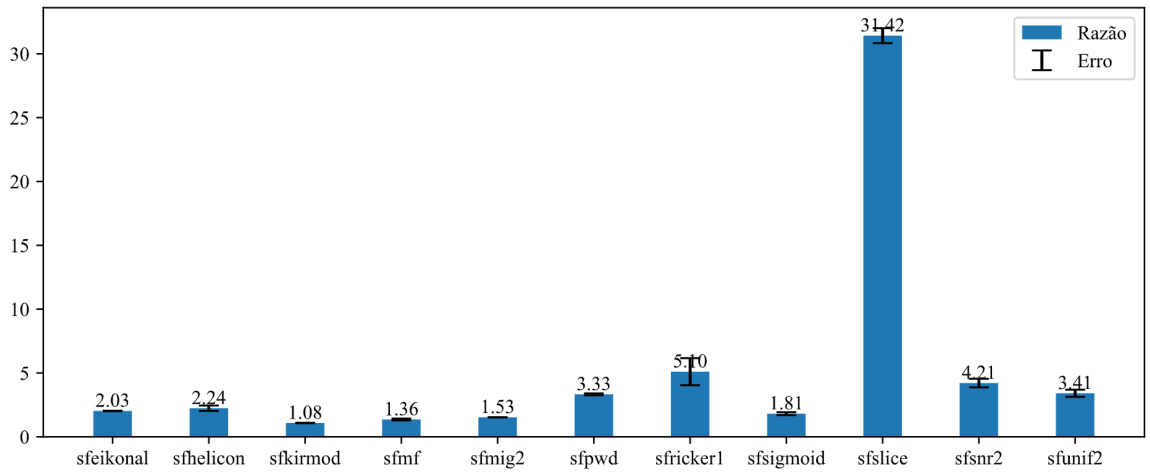


Figura 5: Razão entre o tempo de execução das aplicações com e sem *Open Enclave*.

mais bem definidas. Os primeiros *benchmarks* apresentados medem o impacto no desempenho das operações de leitura e escrita em disco, de cópia de memória dentro do Enclave, de rotinas com forte recursão, e de operações que dependiam apenas do desempenho do processador.

Para medir o impacto no desempenho das operações de leitura e escrita em disco foi medido o tempo de leitura de um arquivo de 6,9MB utilizando a função *fread*, e o tempo de escrita, copiando os dados lidos para outro arquivo, utilizando a função *fwrite*. Para medir o impacto no desempenho das operações de cópia de memória, foi medido o tempo que se leva para copiar os dados de um vetor de 64MB para outro vetor de mesmo tamanho. Para medir o impacto no desempenho de rotinas com forte recursão foi utilizado o algoritmo recursivo para obtenção do quadragésimo número da sequência de Fibonacci. Finalmente, para medir o tempo de execução das operações que dependiam apenas do processador, foi utilizado um algoritmo que encontra o número de primos menores que 100 mil[13].

Os gráficos das figuras 6 e 7 mostram o tempo médio de execução dos *benchmarks* com e sem Open Enclave e a razão entre estes tempos. Pode-se ver que as operações que dependem apenas do desempenho do processador, como operações aritméticas com registradores e operações de controle de fluxo, não sofrem redução do desempenho ao serem executadas com a tecnologia SGX.

O teste que teve o desempenho mais afetado foi o de cópia de memória, o que era esperado de acordo com a documentação em [2]. Esta cita que faltas na *cache* causam uma sobrecarga de desempenho adicional quando comparadas à sistemas que não estão operando com o SGX, pois toda a memória que está fora da *cache* do processador é protegida e a decodificação é feita em tempo de execução ao se buscar por esse endereço de memória. O caso de teste do *benchmark* realiza a cópia de uma porção de memória para outra, logo todos os acessos à memória, com exceção dos endereços que se encontram em uma mesma página, geram uma falta na *cache*. Desta forma a leitura/escrita de memória sequencial foi

[13]Modificado de `https://www.cplusplus.com/reference/ctime/clock`

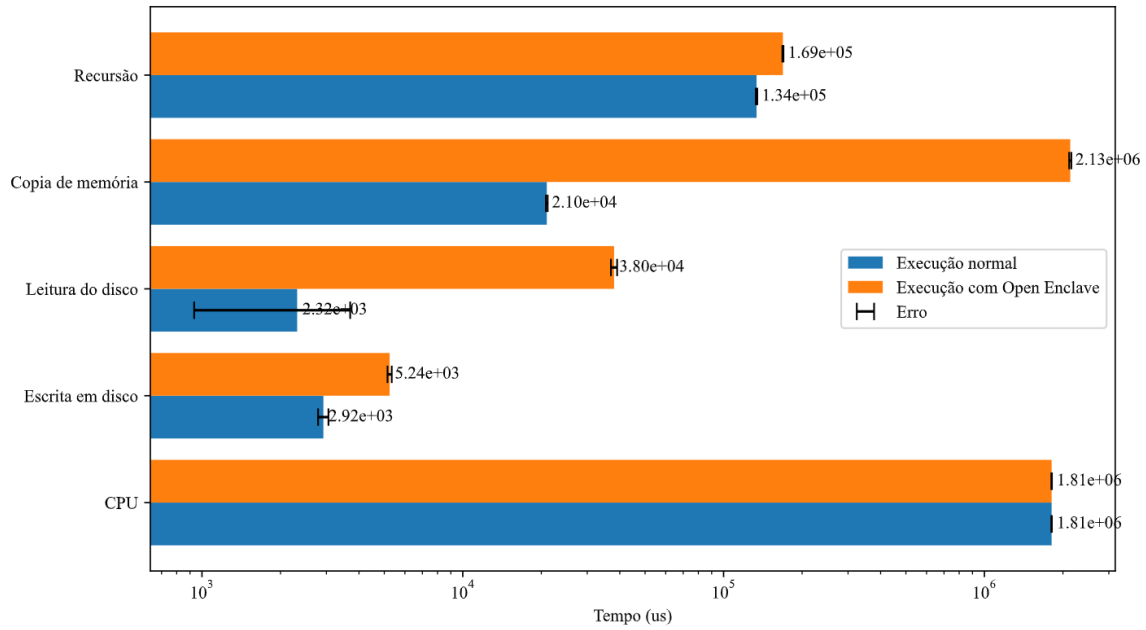


Figura 6: Tempo de execução médio dos *benchmarks* com e sem Open Enclave.

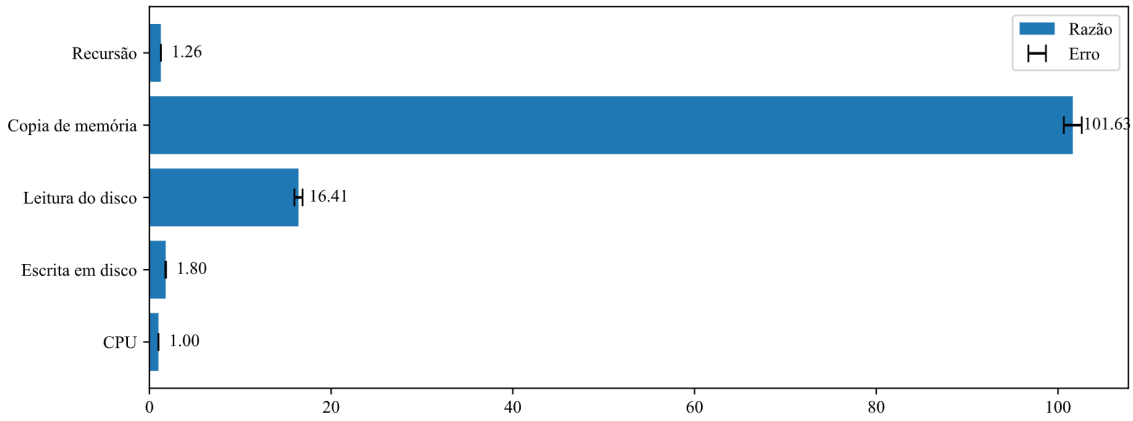


Figura 7: Razão entre os tempos de execução dos *benchmarks* com e sem *Open Enclave*.

impactada em aproximadamente 100 vezes quando executada com o SGX.

Apesar da cópia de memória ter sofrido um grande impacto, o teste de recursão, que também faz uso intensivo da memória, foi pouco impactado quando executada no Enclave. Por se tratar de memória as quais uma se localiza na pilha enquanto a outra se localiza na *heap*, não podia ser descartada a hipótese de que o uso da memória da pilha poderia ter desempenho melhor que o uso de memória da *heap*, neste caso a degradação de desempenho deveria estar vindo da pilha estar excedendo o tamanho previamente alocado, gerando consecutivas alocações de páginas para a pilha. Além disso, pode-se verificar o impacto das

faltas de *cache*, já que algoritmos recursivos se beneficiam de localidade temporal, ao passo que os dados e as posições da pilha são constantemente reutilizados.

A fim de verificar esta hipótese foram executados mais testes, desta vez iterativamente, variando o número da sequência de Fibonacci a ser encontrado de 32 até 42 e variando o tamanho do *buffer* de memória copiado em potências de 2, variando-se este valor de 2 até 32 MB. Os gráficos das figuras 8 e 9 apresentam estes resultados. Observa-se que não há diferença significativa, dados os intervalos de confiança de cada dado, entre as razões de tempo para os diferentes números de Fibonacci. Portanto como a alocação de memória tem comportamento linear, esta não está impactando no desempenho deste *benchmark*. Logo, resta a hipótese de que o algoritmo, ao se beneficiar de localidade temporal, causa menos faltas na *cache* do processador, evitando a sobrecarga de desempenho causada pela leitura e/ou escrita de dados na memória protegida.

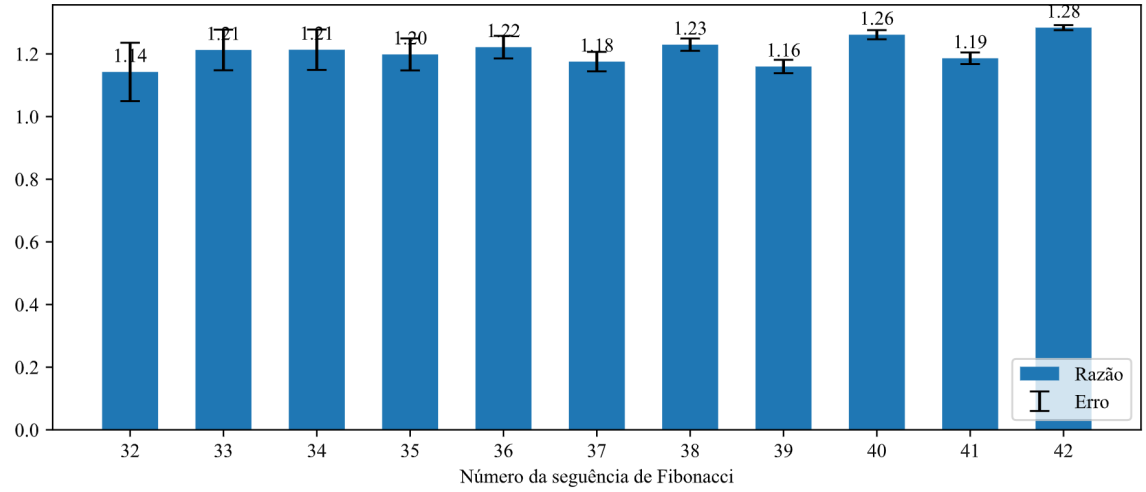


Figura 8: Razão entre o tempo de execução usando *Open Enclave* e o tempo de execução normal para o algoritmo do número de Fibonacci recursivo.

Além do caso do teste de recursão, as operações de disco apresentaram comportamento bem diferentes ao comparar o impacto gerado no tempo de leitura com o impacto no tempo de escrita. O arquivo lido era de apenas 6,9MB, logo grande parte dele poderia ser armazenado na *cache* L3 do processador utilizado, o qual possui 6MB de cache L3, já que o código lê o arquivo do disco e escreve uma cópia dele logo em seguida. Para validar a hipótese de que o aproveitamento da memória *cache* estaria fazendo a escrita em disco obter um melhor desempenho quando comparado à leitura, foi feito um experimento aumentando-se o tamanho do arquivo copiado iterativamente. Os resultados obtidos encontram-se nas figuras 10 e 11.

O desempenho não teve alterações consideráveis conforme o tamanho dos dados lidos se altera, o que indica que a presença de memória *cache* não auxilia no desempenho desse tipo de operação. Vale ressaltar que parte do tempo gasto na leitura dos arquivos também vem da cópia de memória do espaço de memória do disco para a memória do Enclave.

O caso da escrita em disco também não teve diferenças consideráveis, logo a hipótese

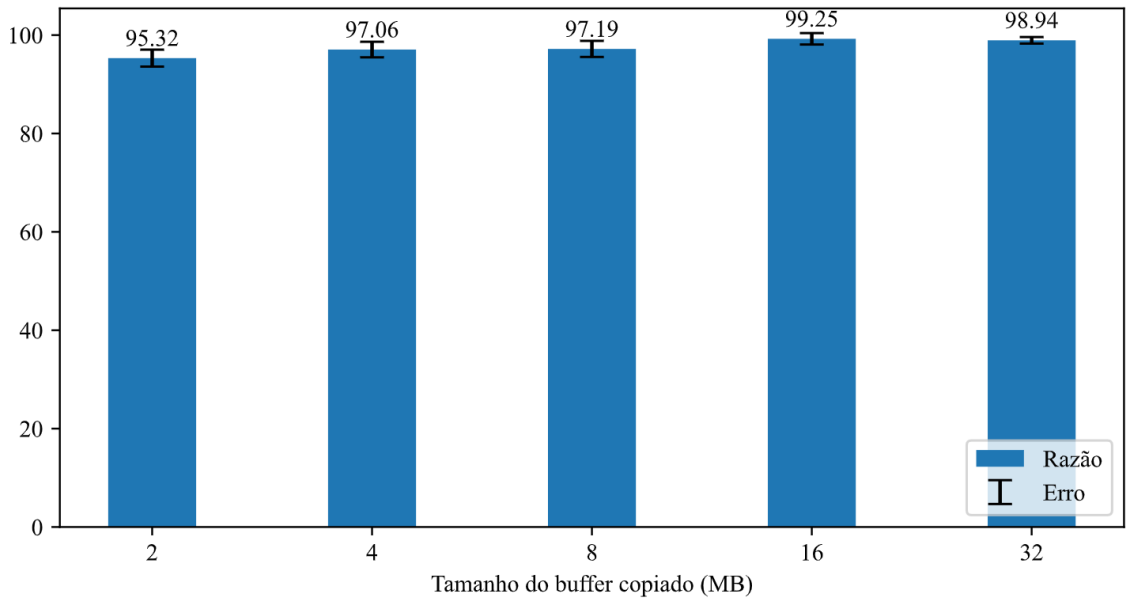


Figura 9: Razão entre o tempo de execução usando *Open Enclave* e o tempo de execução normal para a cópia de memória variando o tamanho do *buffer* copiado.

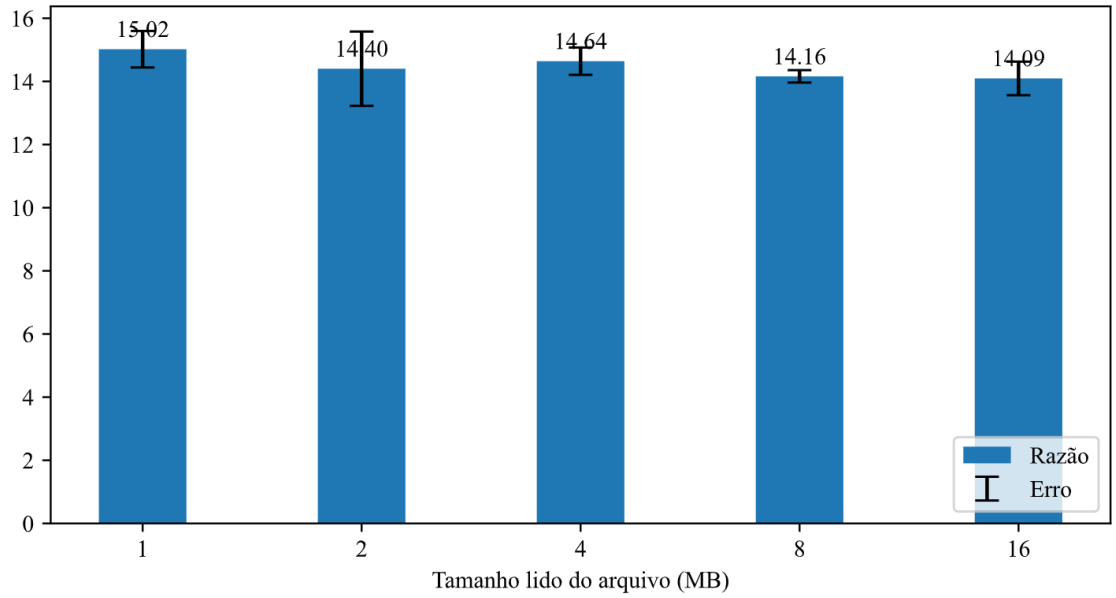


Figura 10: Razão entre o tempo de execução usando *Open Enclave* e o tempo de execução normal para a leitura de um arquivo em disco variando o tamanho lido.

de que a memória *cache* do processador estaria desempenhando a favor do caso de teste de escrita se tornou falsa.

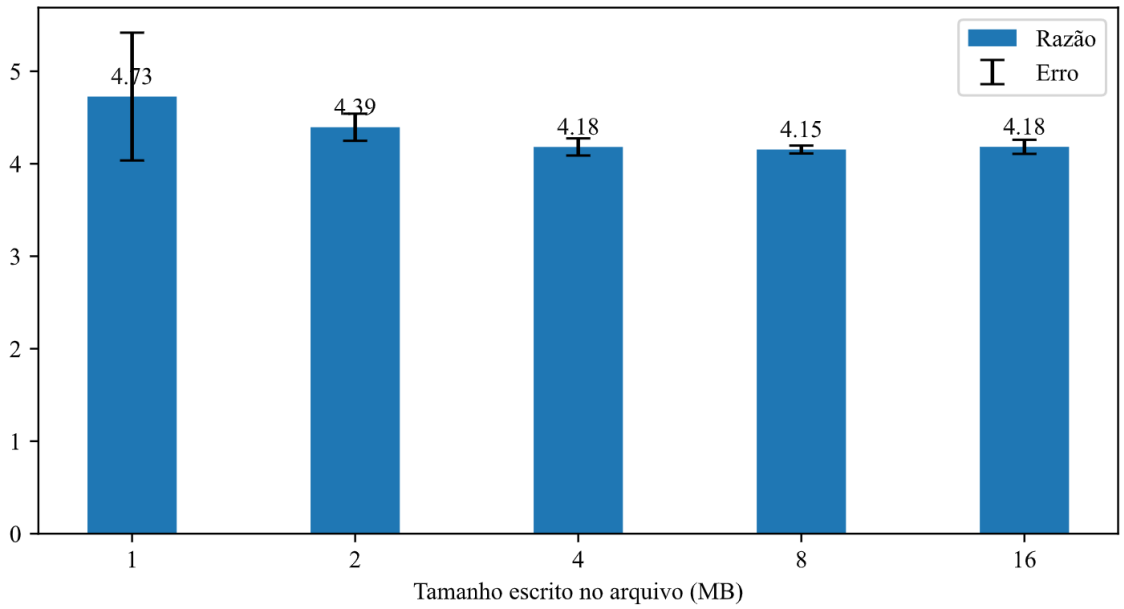


Figura 11: Razão entre o tempo de execução usando *Open Enclave* e o tempo de execução normal para a escrita em um arquivo em disco variando o tamanho escrito.

## 5 Conclusões

A operação que mais tem o desempenho impactado ao ser executada em uma Enclave do Intel SGX foi a operação de cópia de memória, segundo os *benchmarks*, mas foi visto também que operações de memória que se beneficiam da memória *cache* do processador são pouco afetadas. Mostrando que uma das principais fontes de sobrecarga de desempenho da Enclave é o acesso à memória quando ocorre uma falta de *cache*.

Os testes realizados foram feitos com leituras sequenciais do disco e da memória, o que possui localidade espacial, portanto a leitura aleatória pode ser ainda mais impactada, já que a proporção de faltas na *cache* do processador a cada acesso pode aumentar.

Não foi possível averiguar se o impacto no desempenho da leitura e escrita em disco está relacionado à codificação e decodificação em tempo real da memória da Enclave. Para verificar essa possibilidade propõe-se fazer os testes em diferentes tipos de dispositivos de armazenamento (SSD e HDD), determinando-se assim, se o maior limitante no caso do Enclave é o desempenho da memória ou do disco.

As operações que não fazem acesso à memória não tiveram seu desempenho impactado consideravelmente, mostrando que a arquitetura mantém o desempenho do processador quando operando somente com os registradores.

Além disso, observou-se que o tempo de inicialização da Enclave cresce linearmente de acordo com a quantidade de memória EPC alocada inicialmente. Quando a memória inicial alocada é zero, o tempo de inicialização é da ordem de milissegundos.

Outra característica a se investigar é o tempo levado na transição de contexto da Enclave

quando se fazem chamadas ECALL e OCALL.

Desta forma para obter o melhor desempenho de aplicações utilizando a biblioteca Open Enclave e a plataforma Intel SGX faz-se as seguintes recomendações. O programador deve evitar fazer a criação de Enclaves com frequência, procurando utilizar uma mesma instância da Enclave para fazer diferentes chamadas ECALL. O programador, sabendo de antemão, a memória máxima utilizada pela Enclave, deve alocar inicialmente o mínimo de páginas de memória possível, diminuindo o tempo de inicialização. E para a lógica das ECALLs implementadas, deve-se procurar fazer proveito ao máximo de propriedades de localidade, evitando assim a ocorrências frequentes de faltas na *cache* do processador durante a execução da aplicação.

## Referências

[1] S. Mohammed, A. Mohammed and B. Abdelmadjid, *Trusted Execution Environment: What It is, and What It is Not,* 2015 IEEE Trustcom/BigDataSE/ISPA, 57–64 (2015).

[2] Intel Corporation, *Intel® Software Guard Extensions Developer Guide,* [Online]. Disponível em: `https://download.01.org/intel-sgx/linux02.2/docs/Intel_SGX_Developer_Guide.pdf` [Acessado em Junho 2021].

[3] V. Costan e S. Devadas, *Intel SGX Explained,* (2015) [Online]. Disponível em: `https://eprint.iacr.org/2016/086.pdf` [Acessado em Julho 2021].