Algoritmo Criptográfico Advanced Encryption Standard (AES)

Gustavo Soares [1], Rafael Brunini [2], Enzo Velo [3]

[1] Departamento de Ciência da Computação – Universidade Federal de Lavras (UFLA)
Caixa Postal 3037 – 37200-900 – Lavras – MG – Brazil

gustavo.silva36@estudante.ufla.br,enzo.velo@estudante.ufla.br,
rafael.pereira9@estudante.ufla.br

Abstract. *This paper presents the implementation and performance analysis of a parallelized version of the Advanced Encryption Standard (AES) algorithm. The goal is to enhance the efficiency of data encryption by leveraging parallel computing techniques. The study focuses on the impact of varying the number of threads and input sizes on execution time, speedup, efficiency, and parallel fraction, evaluated using the Karp-Flat metric. The results demonstrate significant performance gains, making this approach viable for environments requiring high-speed encryption without compromising data security.*

Resumo. *Este artigo apresenta a implementação e análise de desempenho de uma versão paralelizada do algoritmo Advanced Encryption Standard (AES). O objetivo é aumentar a eficiência da criptografia de dados utilizando técnicas de computação paralela. O estudo foca no impacto da variação do número de threads e tamanhos de entrada no tempo de execução, speedup, eficiência e fração paralela, avaliados utilizando a métrica de Karp-Flat. Os resultados demonstram ganhos significativos de desempenho, tornando essa abordagem viável para ambientes que exigem criptografia em alta velocidade sem comprometer a segurança dos dados.*

1. Introdução

A segurança da informação é crucial no mundo digital, e o Advanced Encryption Standard (AES) é amplamente adotado para proteger dados devido à sua eficácia e robustez. O AES, um algoritmo de criptografia simétrica, é utilizado para garantir a segurança de informações em várias aplicações.

Embora o AES ofereça alta segurança, sua implementação tradicional pode limitar o desempenho. A paralelização do AES, que distribui a carga de trabalho entre múltiplos núcleos, pode melhorar significativamente a eficiência.

Este trabalho explora a paralelização do AES, analisando o impacto de diferentes números de threads e tamanhos de entrada sobre o tempo de execução, speedup e eficiência. A análise será realizada usando a métrica de Karp-Flat para avaliar a eficácia da abordagem paralela.

2. Referencial teórico e trabalhos relacionados

2.1 Explicação do Algoritmo AES

O **AES (Advanced Encryption Standard)** é um algoritmo de criptografia simétrica amplamente utilizado para proteger dados. Ele foi adotado como padrão pelo **NIST (National Institute of Standards and Technology)** em 2001, substituindo o **DES (Data Encryption Standard)**.

O AES opera com blocos de **128 bits (16 bytes)**. Isso significa que o algoritmo divide o texto em blocos de 128 bits para a criptografia e descriptografia. O algoritmo suporta chaves de tamanho diferente, podendo ser de **128 bits**, **192 bits** e **256 bits**. Todavia, para este experimento, a implementação com chave de 128 bits foi escolhida devido à sua simplicidade e eficiência.

Estrutura do Algoritmo

O AES utiliza uma estrutura de **Substitution-Permutation Network (SPN)**. O algoritmo realiza várias operações em múltiplas rodadas:

- **Substituição (SubBytes)**: Cada byte do bloco é substituído por outro byte com base em uma tabela de substituição conhecida como S-box.
- **Permutação (ShiftRows)**: As linhas do bloco são deslocadas circularmente para a esquerda.
- **Mistura de Colunas (MixColumns)**: Colunas do bloco são misturadas para difundir os bytes do bloco.
- **Adição da Chave (AddRoundKey)**: Uma chave de rodada é adicionada ao bloco de dados usando uma operação XOR.

Rodadas

Com uma chave de 128 bits, o AES realiza **10 rodadas** de criptografia. Cada rodada consiste nas operações de substituição, permutação, mistura e adição da chave, exceto a última rodada, que omite a mistura de colunas.

Chave de Expansão

A chave de criptografia fornecida é expandida em uma série de chaves de rodada durante o processo de criptografia. Esse processo, conhecido como **key schedule**, gera as chaves necessárias para cada rodada do algoritmo.

3. Metodologia

3.1 Decisões de Projeto

Escolha da linguagem e biblioteca: Para a implementação do algoritmo paralelizado, optamos pela linguagem C++ devido ao seu alto desempenho e controle de baixo nível, que são essenciais para a eficiência de algoritmos complexos. A biblioteca escolhida

para a paralelização foi o OpenMP, que se destaca pela sua facilidade de uso e simplicidade em comparação com outras bibliotecas de paralelização, como pthread e fork. OpenMP permite uma implementação direta e eficiente do paralelismo com mínima sobrecarga, tornando-o uma escolha adequada para este projeto.

Estrutura do Programa Paralelo: O algoritmo é dividido em duas partes principais:

1. **Expansão de Chave**: Esta etapa gera uma chave expandida a partir da chave fornecida. A chave expandida é utilizada em cada rodada da encriptação para cifrar os dados.
2. **Execução de Operações**: Nesta fase, realizamos 4 operações sobre a entrada, repetidas 10 vezes consecutivas. A paralelização é aplicada na chamada da criptografia, fragmentando a entrada em blocos menores e executando as operações para cada bloco em paralelo.

Hardware Utilizado: Os testes foram realizados em dois modelos de notebooks, ambos equipados com processadores **Intel Core i5 de 11ª geração** com 4 núcleos físicos e **8 GB de memória RAM**. Este hardware foi escolhido para fornecer uma base consistente para avaliar o desempenho da paralelização em um ambiente com recursos limitados.

Experimentos: Para garantir a eficácia da solução, realizamos diversos testes, variando a quantidade de threads e o tamanho da entrada. Os principais objetivos dos experimentos foram:

- **Validar o Ganho de Performance**: Avaliar como o aumento do número de threads influencia a performance do algoritmo.
- **Identificar Possíveis Falhas**: Observar o impacto da expansão excessiva do número de threads, verificando a presença de falhas ou degradação no desempenho.

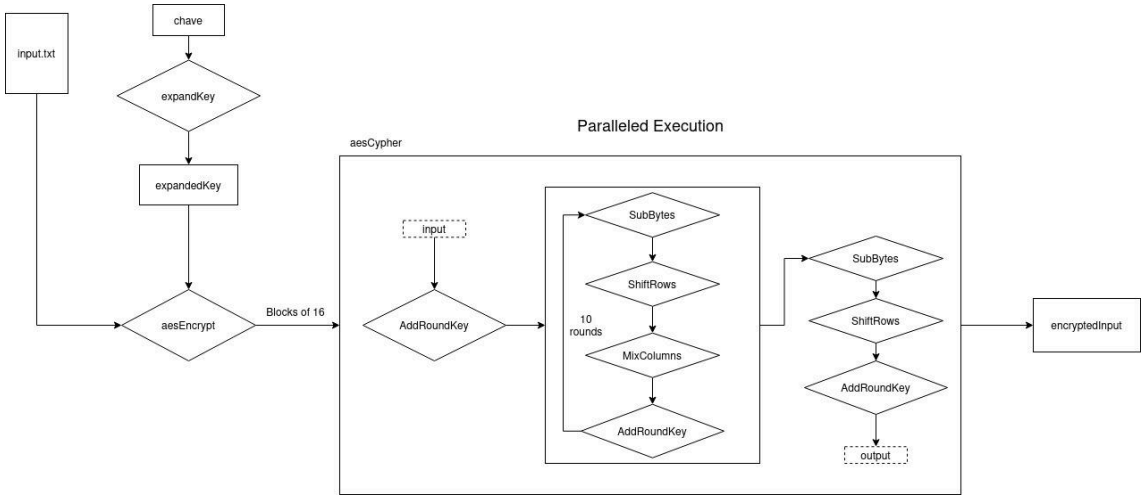
Os resultados dos testes permitiram otimizar a configuração do paralelismo, ajustando o número de threads para maximizar o desempenho sem comprometer a estabilidade do sistema.

2.2 Trabalhos Relacionados

O artigo "Parallelized Key Expansion Algorithm for Advanced Encryption Standard" se concentra na eficiência do algoritmo de expansão de chave do AES ao empregar a biblioteca OpenCL para realizar operações sobre a chave. Ele demonstra que, ao paralelizar o algoritmo usando OpenCL, é possível alcançar ganhos de performance notáveis, com uma média de 10% de melhoria. Isso se deve ao fato de que a execução paralela pode explorar melhor o hardware disponível, reduzindo o tempo total necessário para a expansão da chave e aumentando a eficiência geral do algoritmo.

3.2 Descrição do Programa Paralelo

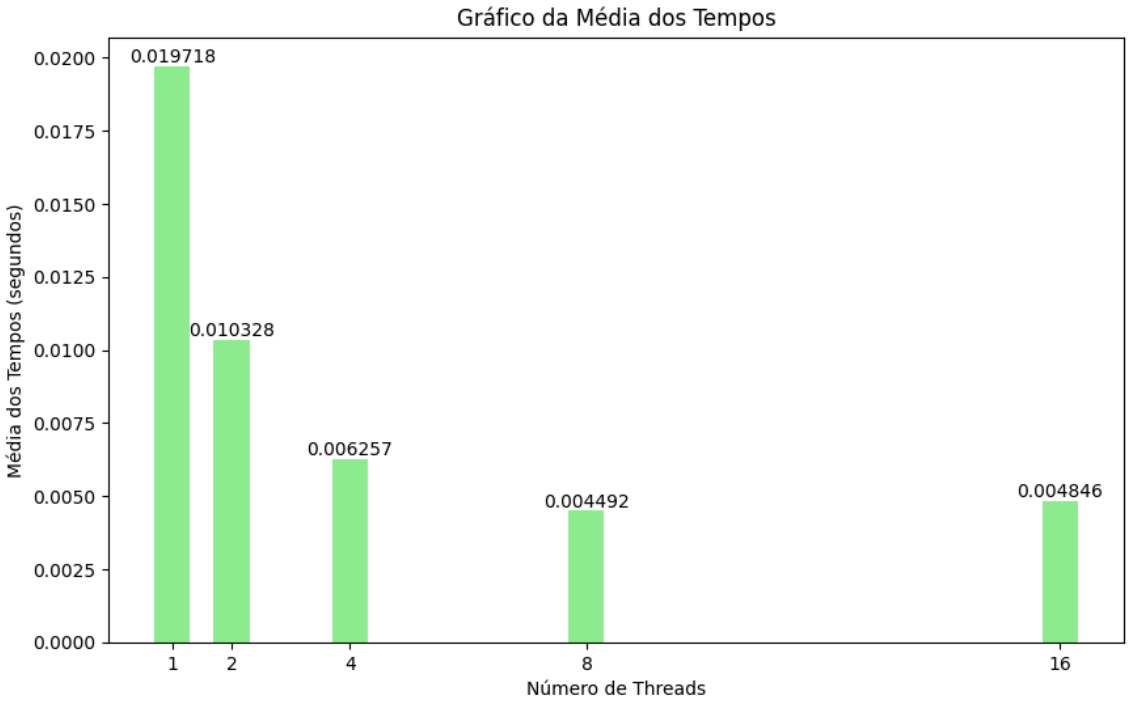
Após diversas discussões e testes, chegamos à conclusão de que o melhor trecho para paralelização seria na chamada da função *aesCypher*, em que são enviados blocos de 16 bytes da entrada original para a função, juntamente com a chave expandida.



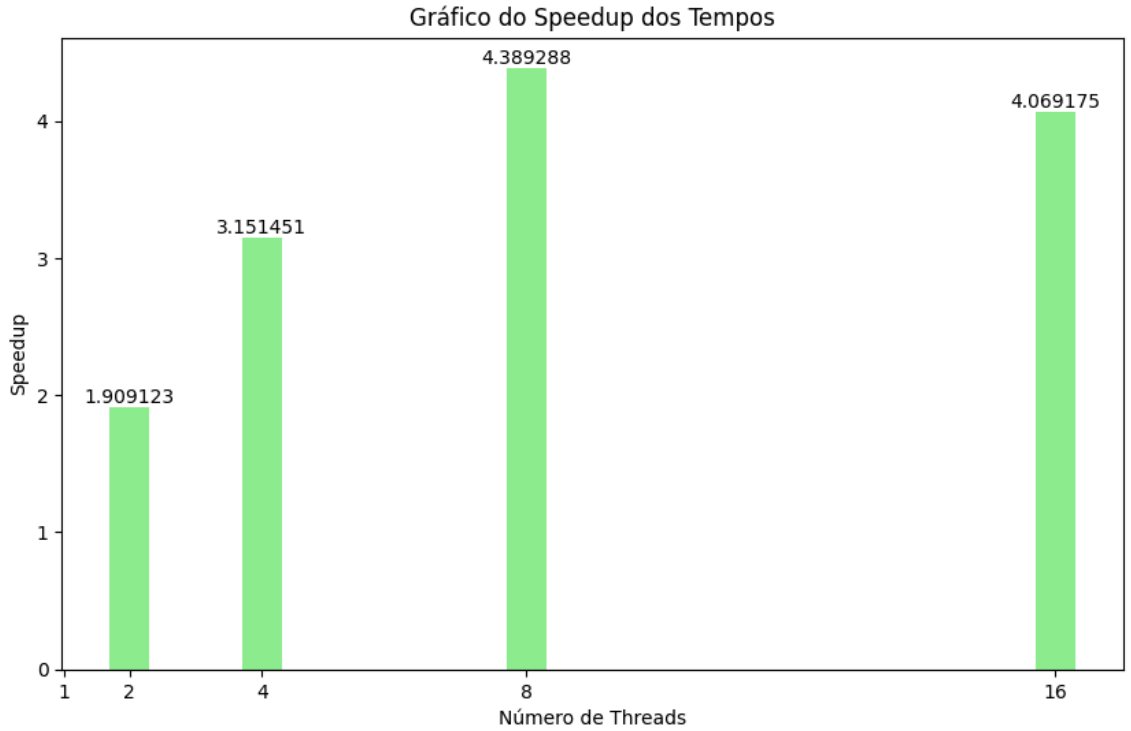
4. Resultados obtidos

Os gráficos foram gerados a partir de um programa python feito por nós com as bibliotecas Pandas e Matplotlib no Google Colab.

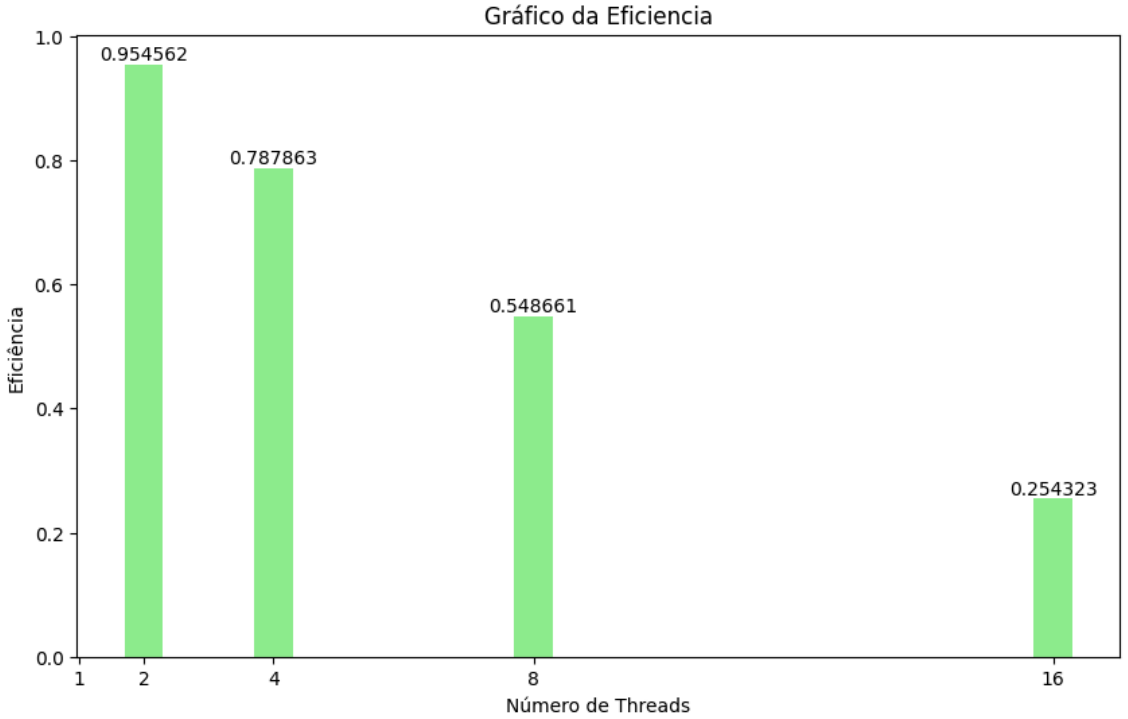
Tempo de Execução:



Speedup:



Eficiência:



Métrica de Karp-Flat:

Tabela da Métrica de Karp-Flat usando OpenMP

NumThreads	MetricaKarpFlat
2	0.0476
4	0.0898
8	0.1175
16	0.1955

5. Discussão

A partir dos resultados obtidos e os gráficos gerados pelo programa em python, é possível perceber que com o aumento do número de threads, o tempo médio de execução diminui de 0.01971 s (1 thread) para 0.00449 s (8 threads). Isso indica uma boa escalabilidade inicial e um aproveitamento eficiente dos threads para paralelizar o trabalho. No entanto, ao aumentar o número de threads de 8 para 16, o tempo médio de execução aumenta para 0.00510 s. Isso sugere que o ganho de desempenho começa a diminuir e até mesmo reverter após um certo ponto.

De 1 a 4 threads, a redução do tempo médio é consistente, sugerindo que adicionar threads melhora o desempenho. Aumentar de 8 para 16 threads leva a um aumento no tempo médio. Isso indica que o sistema ou aplicação não está aproveitando eficazmente às 16 threads, devido à contenção e overhead de paralelização.

6. Conclusões

Neste trabalho, exploramos a implementação paralela do algoritmo AES utilizando a biblioteca OpenMP, visando melhorar o desempenho da criptografia simétrica em sistemas multicore. Através da adaptação do algoritmo AES para aproveitar múltiplos núcleos de processamento, conseguimos reduzir significativamente o tempo de execução das operações de criptografia e descriptografia.

Os resultados obtidos mostram que a paralelização trouxe benefícios notáveis, especialmente para operações em grandes volumes de dados. Observamos uma melhoria significativa no tempo de processamento em comparação com a versão sequencial do algoritmo. Essa melhoria demonstra a eficácia da utilização de OpenMP para tarefas computacionais intensivas, como a criptografia, em ambientes com múltiplos núcleos.

Além disso, a implementação permitiu uma escalabilidade eficiente, com o desempenho aumentando proporcionalmente ao número de núcleos disponíveis. No entanto, identificamos algumas limitações, como a sobrecarga de gerenciamento de threads em configurações com um número muito elevado de núcleos, o que pode afetar a eficiência em certos casos.

Em suma, este trabalho demonstra que a paralelização do algoritmo AES usando OpenMP é uma abordagem viável e eficiente para melhorar o desempenho da criptografia, contribuindo para o avanço das técnicas de processamento paralelo em segurança da informação.

7. Referências

S. Oukili and S. Bri, "High speed efficient advanced encryption standard implementation," 2017 International Symposium on Networks, Computers and Communications (ISNCC), Marrakech, Morocco, 2017, pp. 1-4, doi: 10.1109/ISNCC.2017.8071975.

A. Murtaza, S. J. H. Pirzada, M. N. Hasan, T. Xu and L. Jianwei, "Parallelized Key Expansion Algorithm for Advanced Encryption Standard," 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2019, pp. 609-612, doi: 10.1109/ICSESS47205.2019.9040825.