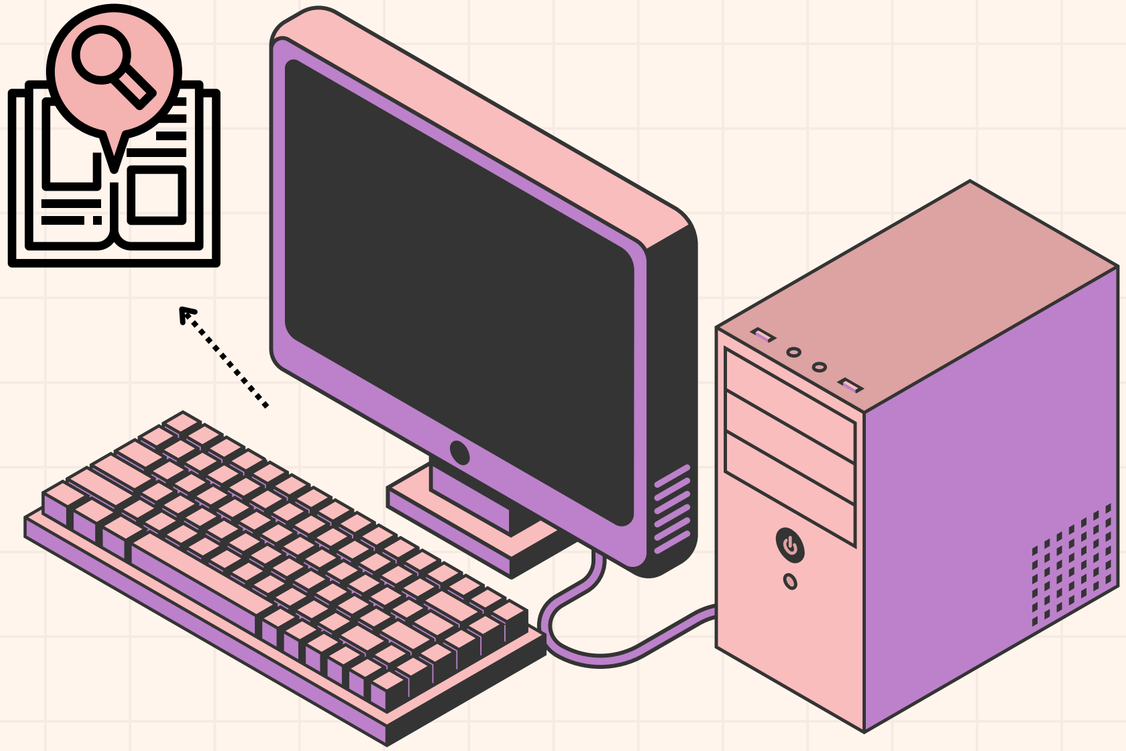
PYTHON MULTIPROCESSING - ATAQUE DE DICIONÁRIO

ORGANIZAÇÃO DE COMPUTADORES



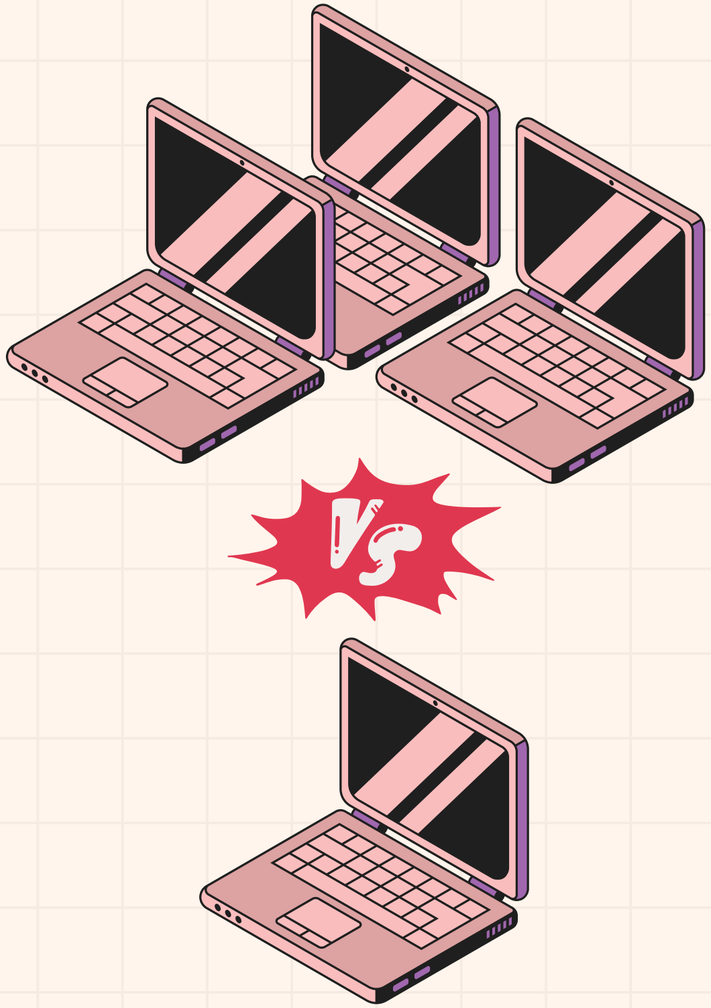
JOÃO EDUARDO, VICTOR HUGO, VINÍCIUS GABOARDI E VICTOR SILVEIRA

INTRODUÇÃO

Nesta apresentação, nós analisaremos o possível ganho de desempenho de um algoritmo de ataque de dicionário por meio da exploração de paralelismo.

Para tal, desenvolvemos um algoritmo em Python utilizando Visual Studio Code para realizar o ataque de forma sequencial e paralela a fim de retornar os dados a serem avaliados comparando seus resultados.

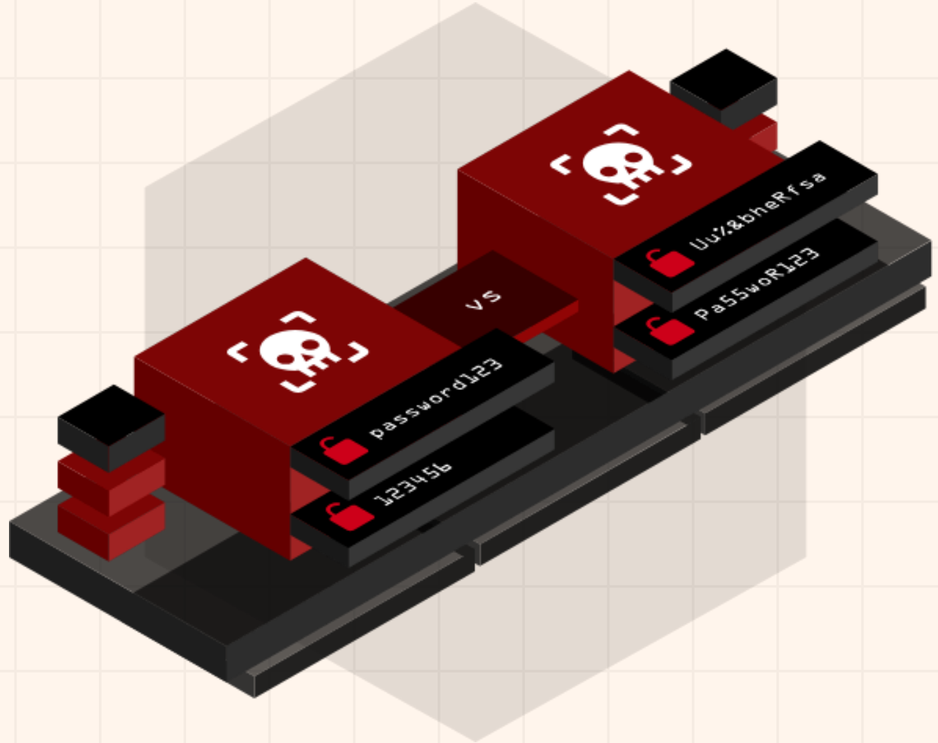
Também foram utilizadas máquinas com contagens de cores e threads variadas para compreender como (e até onde) um número maior de núcleos físicos e lógicos pode trazer vantagens na demanda específica se a estrutura for bem explorada pelo algoritmo.



ALGORITMO
ATAQUE DE DICIONÁRIO

Ataque de dicionário é uma forma de **ataque de força bruta** ou **busca excessiva de chave**, i.e.: processo que envolve violar o login de uma conta através de tentativa e erro.

No caso do ataque de dicionário, o algoritmo faz uso de listas ou arquivos com vários termos pré-definidos, o **“dicionário”**. Essas palavras geralmente incluem termos relacionados ao alvo ou senhas comuns, e o algoritmo as utiliza sistematicamente para tentar adivinhar o nome de usuário e/ou a senha.



HARDWARE & SOFTWARE

COMPUTADOR 1

SISTEMA OPERACIONAL: MICROSOFT WINDOWS 11 HOME
VERSÃO 10.0.22631 COMPILAÇÃO 22631;

TIPO DO SISTEMA: PC BASEADO EM X64;

PROCESSADOR: AMD RYZEN 7 5700X3D 8-CORE PROCESSOR, 3001 MHZ, 8 NÚCLEO(S), 16 PROCESSADOR(ES) LÓGICO(S);

MOBO: PRIME A520M-E;

MEMÓRIA FÍSICA (RAM) INSTALADA 16,0 GB

COMPUTADOR 2

SISTEMA OPERACIONAL: MICROSOFT WINDOWS 10 PRO
VERSÃO 10.0.19045 COMPILAÇÃO 19045;

TIPO DO SISTEMA: PC BASEADO EM X64;

PROCESSADOR: INTEL(R) CORE(TM) I5-9400F CPU @ 2.90GHZ, 2904 MHZ, 6 NÚCLEO(S), 6 PROCESSADOR(ES) LÓGICO(S);

MOBO: TUF H310M-PLUS GAMING/BR;

MEMÓRIA FÍSICA (RAM) INSTALADA 16,0 GB

COMPUTADOR 3

SISTEMA OPERACIONAL: MICROSOFT WINDOWS 10 HOME
VERSÃO 10.0.19045 COMPILAÇÃO 19045;

TIPO DO SISTEMA: PC BASEADO EM X64;

PROCESSADOR: AMD RYZEN 5 5600 6-CORE PROCESSOR, 3501 MHZ, 6 NÚCLEO(S), 12 PROCESSADOR(ES) LÓGICO(S);

MOBO: PRIME B450M-GAMING/BR;

MEMÓRIA FÍSICA (RAM) INSTALADA 16,0 GB

COMPUTADOR 4

SISTEMA OPERACIONAL: MICROSOFT WINDOWS 11 HOME SINGLE LANGUAGE
VERSÃO 10.0.22631 COMPILAÇÃO 22631;

TIPO DO SISTEMA: PC BASEADO EM X64;

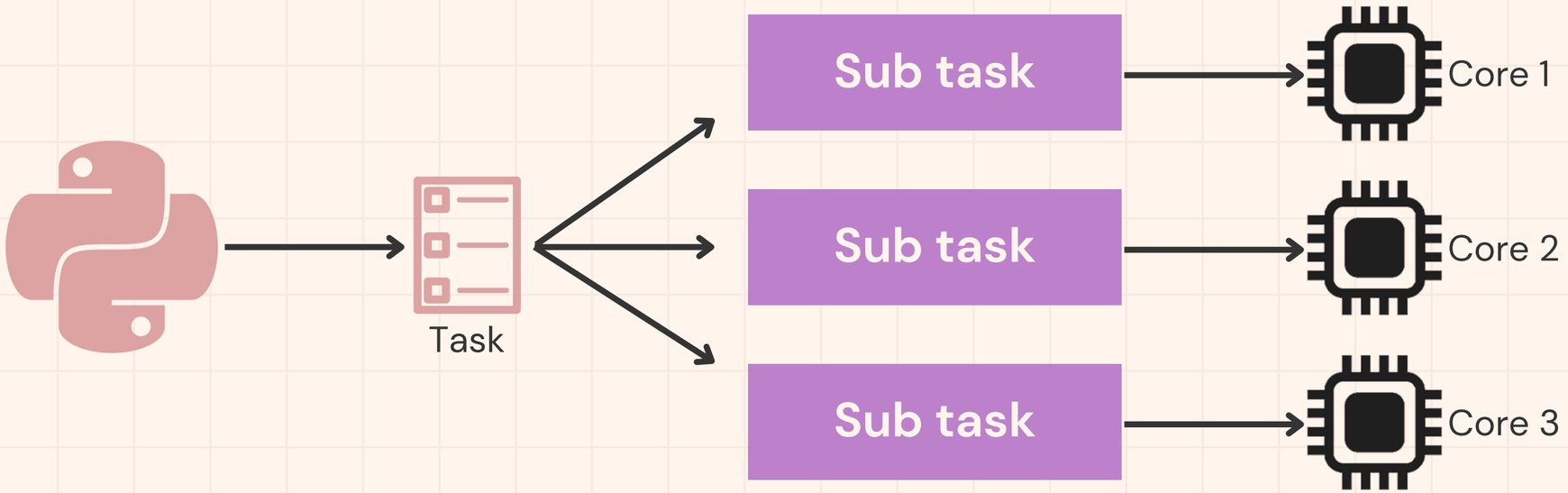
PROCESSADOR: INTEL(R) CORE(TM) I5-1035G1 CPU @ 1.00GHZ, 1190 MHZ, 4 NÚCLEO(S), 8 PROCESSADOR(ES) LÓGICO(S);

MOBO: LNVNB161216;

MEMÓRIA FÍSICA (RAM) INSTALADA 8,00 GB

PYTHON MULTIPROCESSING

- Criação de Processos:
  - A **multiprocessing.Pool** é usada para gerenciar um grupo de processos que executam uma função em paralelo, distribuindo tarefas automaticamente.
- Distribuição de Trabalho:
  - O arquivo de senhas é dividido em pedaços menores (**chunks**) para que cada processo lide com uma parte específica, aumentando a eficiência.



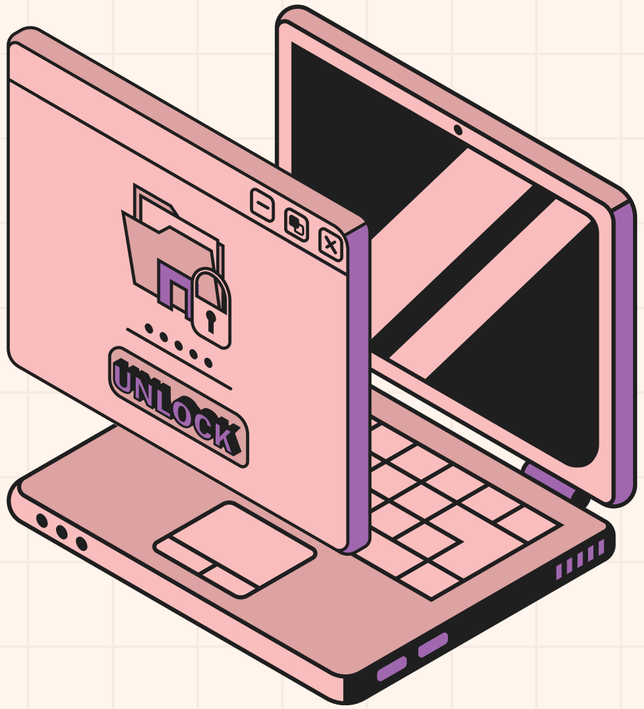
PYTHON MULTIPROCESSING

- Escalabilidade:
  - O uso de **multiprocessing.cpu_count()** determina automaticamente o número máximo de processos possíveis com base nos núcleos do processador.
- Compartilhamento de Dados:
  - Embora cada processo seja independente e possua seu próprio espaço de memória, os dados são organizados para serem manipulados de forma eficiente.
- Gerenciamento de Processos:
  - O **Pool** cuida da criação e do encerramento automático dos processos. Garantindo que os recursos sejam liberados adequadamente.

PYTHON
HASHLIB & TIME

A comparação das strings poderia ser feita diretamente. Contudo, fazemos o uso da biblioteca hashlib para aplicar algoritmos de hash SHA-256 para simular ambientes de verificação de senhas em um banco de dados, onde as senhas são salvas não como são escritas, mas criptografadas.

A biblioteca time do Python fornece funções para trabalhar com o tempo, incluindo a manipulação de datas e horários, bem como a medição de tempo de execução.



VISÃO GERAL
IMPLEMENTAÇÃO

- As condições do algoritmo remetem à verificação de 100 milhões de senhas armazenadas em um arquivo chamado “senhas100M.txt”. Dessa maneira, elas são geradas randomicamente considerando caracteres alfanuméricos, usando letras maiúsculas e minúsculas, totalizando 62 dígitos para uma senha de 5 elementos;

- Utilizamos a biblioteca Time para cronometrar o tempo dos métodos sequencial e paralelo;

- Reunimos os dados retornados: tempo sequencial, paralelo, overhead, speedup e a eficiência.

62^3 = 238.328 combinações

62^4 = 14.776.336 combinações

62^5 = 916.132.832 combinações

62^6 = 56.800.235.584 combinações

62^7 = 3.521.614.606.208 combinações

VISÃO GERAL
IMPLEMENTAÇÃO

Os cálculos de Overhead, SpeedUp e eficiência:

OVERHEAD

$$\text{Overhead} = \frac{T_{\text{paralelo}} - T_{\text{sequencial}}}{T_{\text{paralelo}}}$$

SPEEDUP

$$S(p) = \frac{\text{tempo de execução sequencial}}{\text{tempo de execução paralela}} = \frac{t_s}{t_p}$$

EFICIÊNCIA

$$\text{Eficiência} = \frac{\text{Speedup}}{N}$$

SEGMENTOS PARALELIZADOS

Etapa	Ação	Objetivo
Divisão do arquivo	O arquivo de senhas é dividido em pedaços (chunks) para que cada processo trate de um subconjunto de linhas.	Garantir que o trabalho seja dividido de forma balanceada.
Função process_chunk	Processa cada pedaço de tentativas, verificando se alguma senha corresponde ao hash da senha-alvo.	Executar as verificações de forma independente entre os processos.
Execução com multiprocessing	Cada processo no pool executa a função process_chunk, recebendo um pedaço do dicionário e o hash da senha-alvo.	Aproveitar múltiplos núcleos para paralelizar a execução e reduzir o tempo total do ataque.
Coleta de resultados	Os resultados dos processos paralelos são reunidos, e o algoritmo retorna a primeira senha válida encontrada ou None.	Consolidar os resultados paralelos para determinar o sucesso ou o fracasso do ataque.

```python
# Função para verificar senhas em um intervalo de linhas do arquivo
def process_chunk(chunk, password_hash):
    for attempt in chunk:
        if check_password(password_hash, attempt):
            return attempt
    return None
```

```python
# Função para dividir o arquivo em pedaços e realizar o ataque de dicionário
def parallel_dictionary_attack(file_path, password, num_processes):
    password_hash = sha256_hash(password)
    with open(file_path, 'r', encoding='latin-1') as f:
        attempts = f.readlines()

    # Divide as tentativas em pedaços para processamento paralelo
    chunk_size = len(attempts) // num_processes
    chunks = [attempts[i:i + chunk_size] for i in range(0, len(attempts), chunk_size)]

    # Processa os pedaços em paralelo
    with multiprocessing.Pool(num_processes) as pool:
        results = pool.starmap(process_chunk, [(chunk, password_hash) for chunk in chunks])

    # Retorna a primeira senha encontrada ou None
    return next((res for res in results if res), None)
```

RESULTADOS
COMPUTADOR 1

Ryzen 7 5700x3d							
Posição em linha no arquivo .txt	**Senha**	**Sem Paralelo (s)**	**Com Paralelo (s)**	**Overhead do Paralelismo (s)**	**Speedup**	**Eficiência**	**Variação do speedup**
1	AwHCW	0.02	38.71	38.71	0	0.00	
10.000.000	6C6YG	15.72	36.28	35.30	0,43	0.03	0,43
20.000.000	0E6l0	31.98	35.69	33.69	0,9	0.06	0,47
30.000.000	UkMC5	46.74	35.87	32.95	1,3	0.08	0,4
40.000.000	LYms3	61.67	35.57	31.72	1,73	0.11	0,43
50.000.000	tguKO	76.34	36.40	31.63	2,1	0.13	0,37
60.000.000	hRv9n	93.24	36.24	30.41	2,57	0.16	0,47
70.000.000	ErKA9	109.15	36.42	29.60	3	0.19	0,43
80.000.000	A3OYi	124.81	36.92	29.12	3,38	0.21	0,38
90.000.000	959v5	139.94	36.73	27.99	3,81	0.24	0,43
100.000.000	q8hF3	155.73	36.18	26.45	4,3	0.27	0,49
						Média da variação do speedup	0,43

RESULTADOS
COMPUTADOR 2

Intel i5-9400f							
Posição em linha no arquivo .txt	**Senha**	**Sem Paralelo (s)**	**Com Paralelo (s)**	**Overhead do Paralelismo (s)**	**Speedup**	**Eficiência**	**Variação do speedup**
1	AwHCW	0.00	81.55	81.55	0	0.00	
10.000.000	6C6YG	22.22	67.74	64.04	0,33	0.05	0,33
20.000.000	0E6l0	47.11	67.45	59.60	0,7	0.12	0,37
30.000.000	UkMC5	69.38	66.99	55.43	1,04	0.17	0,34
40.000.000	LYms3	91.55	64.91	49.65	1,41	0.24	0,37
50.000.000	tguKO	113.77	67.60	48.63	1,68	0.28	0,27
60.000.000	hRv9n	137.60	65.61	42.67	2,1	0.35	0,42
70.000.000	ErKA9	156.81	61.85	35.72	2,54	0.42	0,44
80.000.000	A3OYi	177.31	65.09	35.54	2,72	0.45	0,18
90.000.000	959v5	206.99	57.42	22.92	3,6	0.60	0,88
100.000.000	q8hF3	225.97	63.46	25.80	3,56	0.59	-0,04
						Média da variação do speedup	0,36

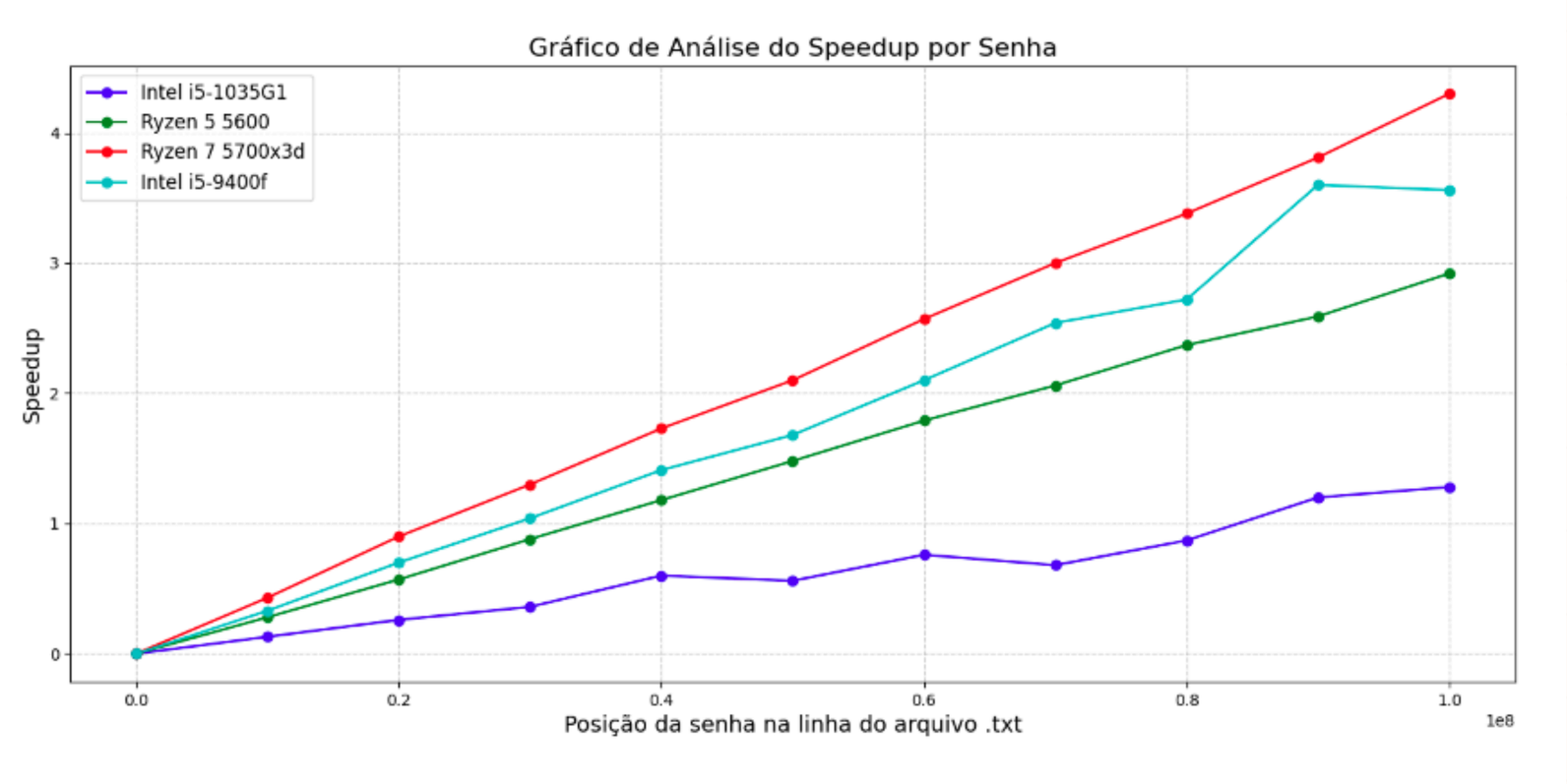
RESULTADOS
COMPUTADOR 3

Ryzen 5 5600							
Posição em linha no arquivo .txt	**Senha**	**Sem Paralelo (s)**	**Com Paralelo (s)**	**Overhead do Paralelismo (s)**	**Speedup**	**Eficiência**	**Variação do speedup**
1	AwHCW	0.00	53.87	53.87	0	0.00	
10.000.000	6C6YG	14.61	51.45	50.23	0,28	0.02	0,28
20.000.000	0E6l0	29.15	51.26	48.83	0,57	0.05	0,29
30.000.000	UkMC5	42.94	48.86	45.29	0,88	0.07	0,31
40.000.000	LYms3	58.35	49.43	44.57	1,18	0.10	0,3
50.000.000	tguKO	72.38	49.05	43.02	1,48	0.12	0,3
60.000.000	hRv9n	87.87	48.97	41.65	1,79	0.15	0,31
70.000.000	ErKA9	100.18	48.64	40.29	2,06	0.17	0,27
80.000.000	A3OYi	115.37	48.62	39.01	2,37	0.20	0,31
90.000.000	959v5	128.29	49.46	38.77	2,59	0.22	0,22
100.000.000	q8hF3	143.74	49.19	37.21	2,92	0.24	0,33
						Média da variação do speedup	0,29

RESULTADOS
COMPUTADOR 4

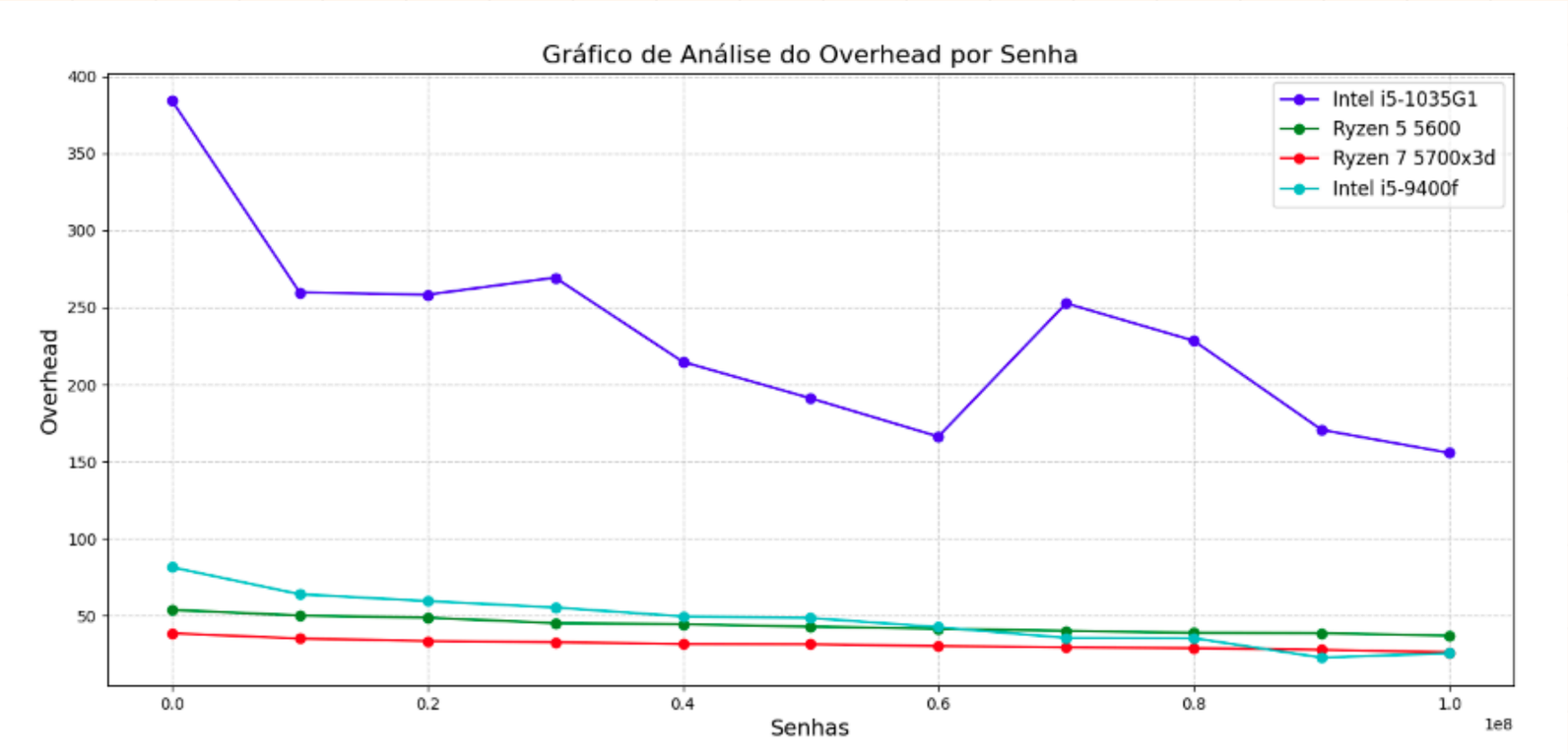
Intel i5-1035G1							
Posição em linha no arquivo .txt	**Senha**	**Sem Paralelo (s)**	**Com Paralelo (s)**	**Overhead do Paralelismo (s)**	**Speedup**	**Eficiência**	**Variação do speedup**
1	AwHCW	0.00	384.15	384.15	0	0.00	
10.000.000	6C6YG	35.04	264.18	259.80	0,13	0.02	0,13
20.000.000	0E6l0	69.19	266.93	258.28	0,26	0.03	0,13
30.000.000	UkMC5	102.24	282.16	269.38	0,36	0.05	0,1
40.000.000	LYms3	138.43	232.04	214.74	0,6	0.07	0,24
50.000.000	tguKO	115.13	205.49	191.10	0,56	0.07	-0,04
60.000.000	hRv9n	139.06	183.63	166.25	0,76	0.09	0,2
70.000.000	ErKA9	186.64	276.03	252.70	0,68	0.08	-0,08
80.000.000	A3OYi	221.79	256.14	228.41	0,87	0.11	0,19
90.000.000	959v5	241.71	200.87	170.65	1,2	0.15	0,33
100.000.000	q8hF3	237.62	185.36	155.66	1,28	0.16	0,08
						Média da variação do speedup	0,13

RESULTADOS
SPEEDUP



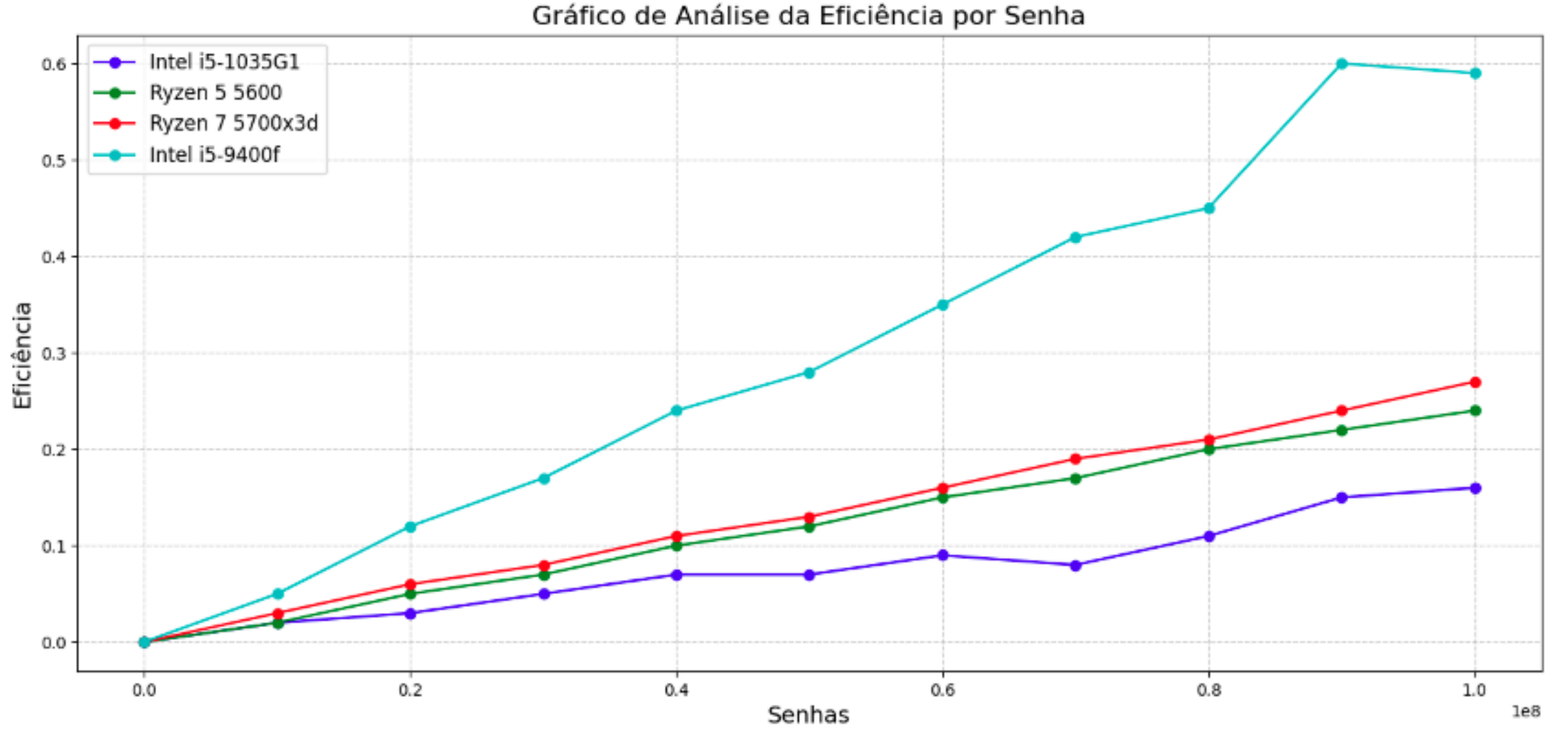
- QUANTO MAIOR, MELHOR.

RESULTADOS
OVERHEAD



- QUANTO MENOR, MELHOR.

RESULTADOS
EFICIÊNCIA



EFICIÊNCIA

$$\text{Eficiência} = \frac{\text{Speedup}}{N}$$

```
clock / número de processadores lógicos

Computador 2 (Intel i5-9400f) -> 2,9 / 6 = 0,48
Computador 1 (Ryzen 7 5700x3d) -> 3 / 16 = 0,18
Computador 3 (Ryzen 5 5600) -> 3,5 / 12 = 0,29
Computador 4 (Intel i5-1035G1) -> 1,1 / 8 = 0,13
```

- QUANTO MAIOR, MELHOR.

TESTES EM SALA DE AULA

```python
import hashlib
import multiprocessing
import time

# Função para calcular o hash de uma senha
def sha256_hash(password):
    return hashlib.sha256(password.encode()).hexdigest()

# Função para verificar se uma tentativa corresponde à senha alvo
def check_password(password_hash, attempt):
    return sha256_hash(attempt.strip()) == password_hash

# Função para verificar senhas em um intervalo de linhas do arquivo
def process_chunk(chunk, password_hash):
    for attempt in chunk:
        if check_password(password_hash, attempt):
            return attempt
    return None
```

```python
# Função para dividir o arquivo em pedaços e realizar o ataque de dicionário
def parallel_dictionary_attack(file_path, password, num_processes):
    password_hash = sha256_hash(password)
    with open(file_path, 'r', encoding='latin-1') as f:
        attempts = f.readlines()

    # Divide as tentativas em pedaços para processamento paralelo
    chunk_size = len(attempts) // num_processes
    chunks = [attempts[i:i + chunk_size] for i in range(0, len(attempts), chunk_size)]

    # Processa os pedaços em paralelo
    with multiprocessing.Pool(num_processes) as pool:
        results = pool.starmap(process_chunk, [(chunk, password_hash) for chunk in chunks])

    # Retorna a primeira senha encontrada ou None
    return next((res for res in results if res), None)
```

TESTES EM SALA DE AULA

```python
if __name__ == "__main__":
    file_path = "senhas100M.txt"  # Caminho do arquivo de senhas

    num_processes = multiprocessing.cpu_count()  # Usa o número máximo de processos disponíveis

    print("senha; sem paralelo; com paralelo; Overhead do paralelismo; Speedup; Eficiência")

    for i in ['AwHCW','6C6YG','0E6l0','UkMC5','LYms3','tguKO','hRv9n','ErKA9','A3OYi','959v5','q8hF3']:
        #'AwHCW','6C6YG','0E6l0','UkMC5','LYms3','tguKO','hRv9n','ErKA9','A3OYi','959v5','q8hF3'
        target_password = i
        start_seq = time.time()
        with open(file_path, 'r', encoding='latin-1') as f:
            for line in f:
                if check_password(sha256_hash(target_password), line):
                    found_password_seq = line.strip()
                    break
            else:
                found_password_seq = None
```

```python
end_seq = time.time()

resultado = found_password_seq
tempo_sequencial = end_seq - start_seq

start_par = time.time()
found_password_par = parallel_dictionary_attack(file_path, target_password, num_processes)
end_par = time.time()

tempo_paralelo = end_par - start_par

overhead = tempo_paralelo - (tempo_sequencial / num_processes)
speedup = tempo_sequencial / tempo_paralelo
eficiencia = speedup / num_processes

print(f"{resultado}; {tempo_sequencial:.2f}; {tempo_paralelo:.2f}; {overhead:.2f}; {speedup:.2f}; {eficiencia:.2f}")
```

REFERÊNCIAS

https://docs.python.org/3/library/multiprocessing.html

https://sites.icmc.usp.br/lefraso/parallel.html

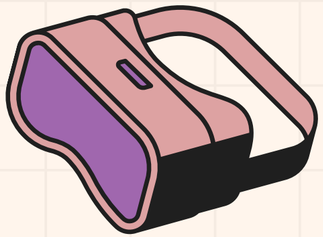
http://profs.ic.uff.br/~simone/labprogparal/contaulas/metricas.pdf

https://www.inf.pucrs.br/emoreno/undergraduate/CC/orgarqii/class_files/Aula08.pdf

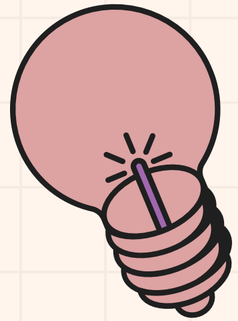
https://medium.com/@arianehorbach/the-basics-of-the-basic-dictionary-attack-2234b72a4fee

https://nobug.com.br/glossario/o-que-e-hash-de-senha/

https://awari.com.br/python-multiprocessing-aumente-a-eficiencia-do-seu-codigo-com-processamento-paralelo/



OBRIGADO !



E VISITE NOSSO GIT:

HTTPS://GITHUB.COM/VICTORHUGOCHRISOSTHEMOS/COMPUTACAO_PARALELA