Brute-Force Password Cracking

Gustavo Antonio Martini

[1]UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
COLEGIADO DE CIÊNCIA DA COMPUTAÇÃO

gutamen@live.com

Resumo. *Abordagem para resolução de um problema de paralelização apresentado na maratona de programação paralela de 2019.*

1. Introdução

O problema foi proposto na 14ª edição da maratona de programação paralela, o nome dele é *Brute-Force Password Cracking*, responsável por descobrir a senha, que está codificada em *hash* MD5, por força bruta. O código original faz o cálculo da *String* e itera caractere a caractere do alfabeto que para checar se o *hash* resultante é igual, originalmente uma função recursiva, era definido o tamanho da palavra chave para a parada sem sucesso.

2. Descrição do problema

O programa faz o uso de computação intensiva para gerar todas as *Strings* possíveis e, a partir das palavras, gerar códigos *hash* MD5 para comparar entre o *hash* gerado e o quer ser encontrado. A função de geração realiza operações em modo recursivo, cada caractere adicional na palavra é um nível de recursão que é adentrado para geração. Caso encontre o resultado uma *tag* de parada é acionada e todas as funções recursivas param sucessivamente.

Lê o hash do arquivo
Gera string com N letras
Para cada N letra é entrado um nível de recursão
Cria hash a partir da string gerada
Compara os dois hashs
Caso diferente volta para gerar string
Caso igual imprime a palavra que tem o hash certo e o hash

O principal problema é conseguir evitar a função recursiva, que tem um custo computacional alto em C, o que dificulta a paralelização do programa.

3. Descrição da paralelização

Principal diferença foi a mudança da função de recursiva para uma função iterativa, além disso, por questões de compatibilidade, a função de cálculo *hash* foi incorporada ao código, ao invés de ser importada de uma biblioteca. Na função iterativa o comportamento do avanço na *String* de comparação ocorre a partir de um número com base na quantidade de caracteres utilizados para forma as palavras, assim é possível iterar a partir

do número baseado no alfabeto utilizado e acessar todas a palavras possíveis.
Para paralelização a abordagem seguiu o principio em dividir a carga entre os processadores de modo em que cada núcleo trabalhasse com um conjunto de posições na primeira letra da palavra chave, essa divisão é feita utilizando OpenMP, a iteração é incrementada com base no número de núcleos operando para resolver a criptografia. Assim, cada processador sempre repete quais caracteres ele combinará até chegar na iteração de limite, essa que é definida pelo tamanho máximo da palavra chave que será testada.

4. Análise de desempenho

4.1. Máquina Utilizada

Nos testes foi utilizado um computador com os seguintes componentes: Processador Ryzen 5 7600X, 32 GB RAM DDR5 5600 MHz, utilizando Windows 11 e o compilador MinGW-w64, o mesmo tem disponível OpenMP para execução do código.

4.2. Testes

Os testes foram realizados a partir do código *hash* da palavra **PARALEL**, a versão recursiva com um núcleo e iterativa com um, seis e doze núcleos são comparadas, cada uma gerou respectivamente os seguintes resultados: 10625,164 segundos, 8080,477 segundos, 1708,512 segundos e 1262,85 segundos, como na figura 1. Houve uma melhora significativa de desempenho, mesmo com um núcleo único, quando comparado ao desempenho original, demonstrado na figura 2.
Quando é comparado o custo computacional e o desempenho aumentado, como esperado, é perceptível o desperdício de recursos, gráfico de eficiência na figura 3. O código contou com uma execução quase que por completa em multi núcleos o que fez com que a conta da lei de Amdahl devolvesse uma resposta de quase um para um de melhora alcançável, resultado disponível na figura 4.

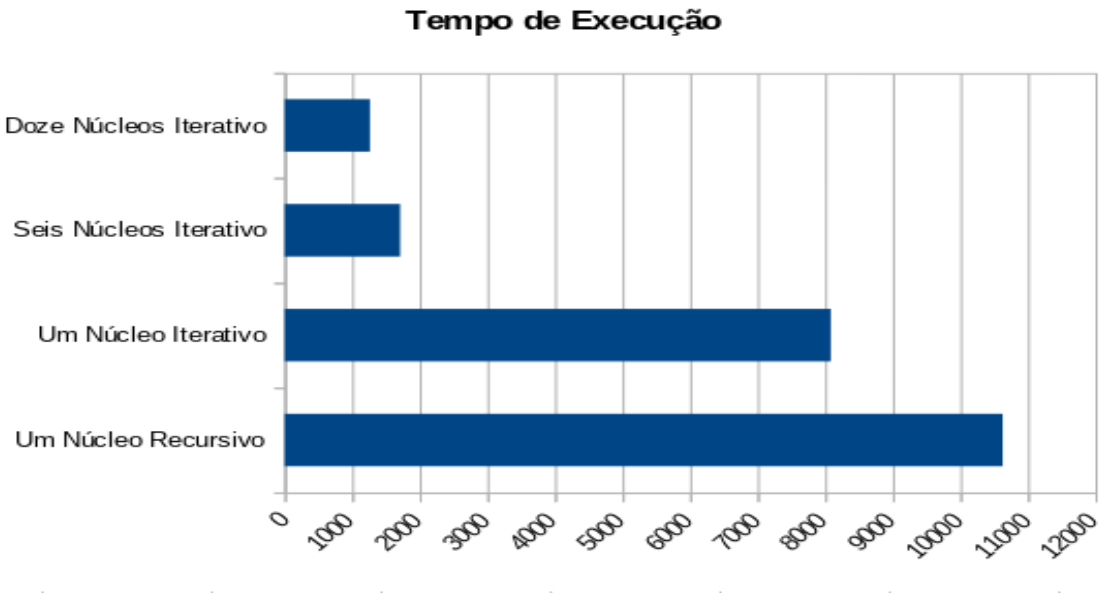

Figura 1. Tempo de execução para cada configuração

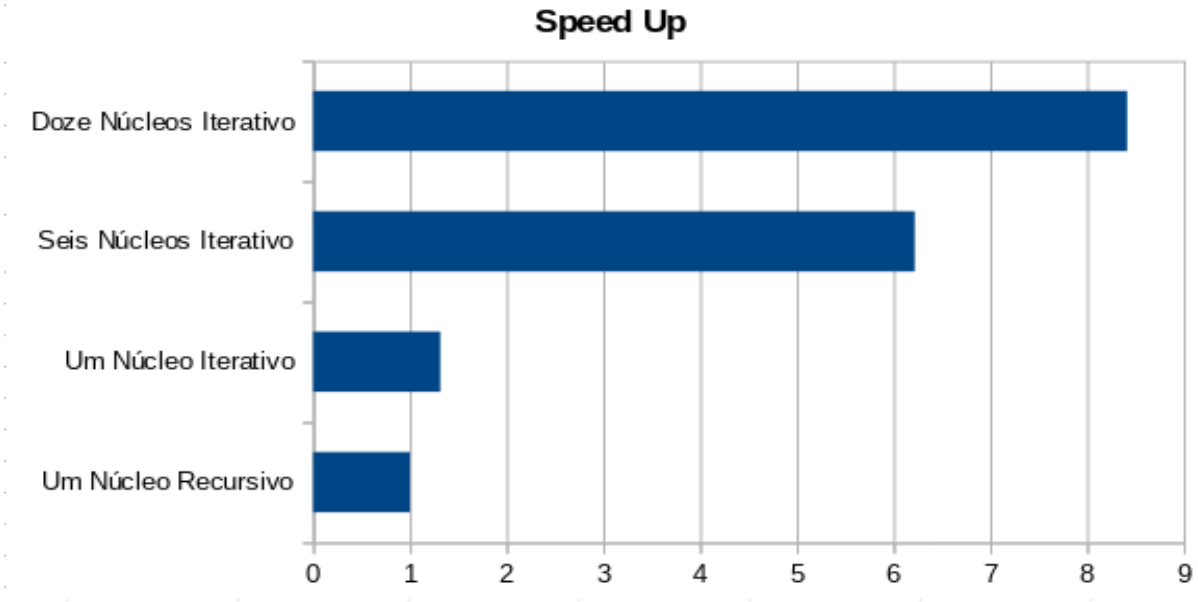

Figura 2. Comparação da velocidade de execução

5. Conclusão

Houve uma melhora do desempenho do código, quando comparado entre a versão de um núcleo e as versões que utilizavam mais núcleos. Com o *Speed-up* de xxxx com doze núcleos, porém o código retém um problema, o limitador de máximo de núcleos é atribuído pelo tamanho do alfabeto utilizado na geração de *Strings* e o desempenho pode ser afetado dependendo de qual quantidade de núcleos está sendo utilizada, pois existem casos em que um núcleo faz iterações a menos para avançar na geração. Mesmo com os problemas o programa melhorou substancialmente e foi possível quebrar a senha com menos tempo do que era necessário.

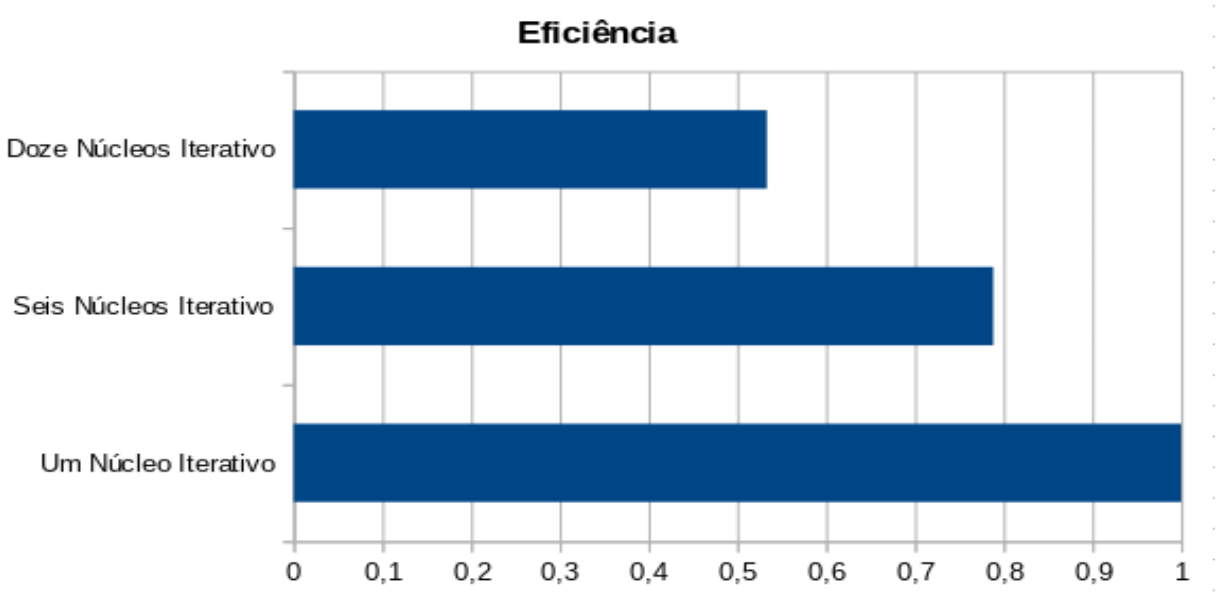

Figura 3. Eficiência na melhora, comparando somente os iterativos

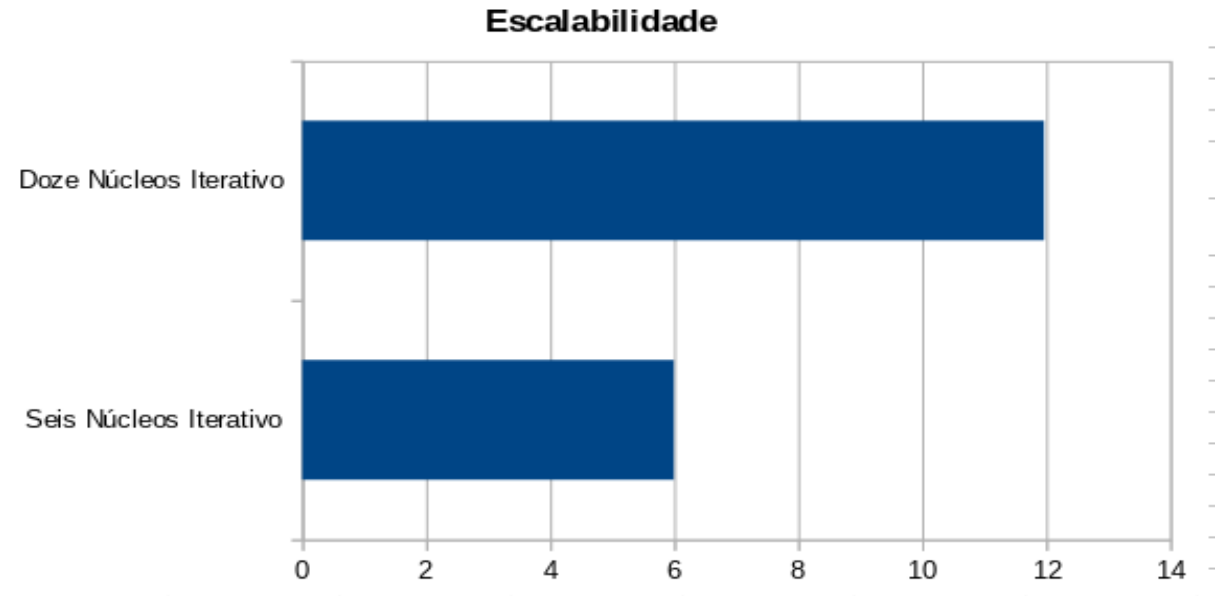

Figura 4. Escalabilidade possível, código *single thread* ocupa baixíssimo tempo da execução