UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Relatório - EP1

Aluno: Luís Henrique Gundes Valim.
Matrícula: 2015207401
Disciplina: Processamento Paralelo - Semestre 2019/1.

Introdução

Configurações do computador utilizado para testes

- Sistema operacional: Linux Mint 19.1 Cinnamon (versão 4.0.10 do Cinnamon)
- Processador: Intel© Core™ i7-8550U CPU @ 1.80GHz x 4
- Memória: 8 GB RAM
- Versão do g++: 7.4.0

Execução

- Linha de comando: g++ busca_senhas_$(sequencial ou binária).cpp -O3 -pthread -o programa
- ./programa numThread arqSenhasVazadas.txt arqSenhasVerificar.txt
- Foi executado, tanto a busca binária quanto a sequencial, com 5 quantidades de Threads diferentes (1,2,3,4 e 8). Para cada quantidade de Thread, o programa foi executado 10 vezes. Esse processo foi aplicado para cada quantidade de senhas a serem verificadas (1,2,100,500,1000 e 5000).

Busca Sequencial e Paralela

Em ambas as buscas, o que muda no código é o tipo de busca em si, todo o restante é comum a ambos códigos. De uma forma simplificada, é possível descrever o algoritmo da seguinte forma:

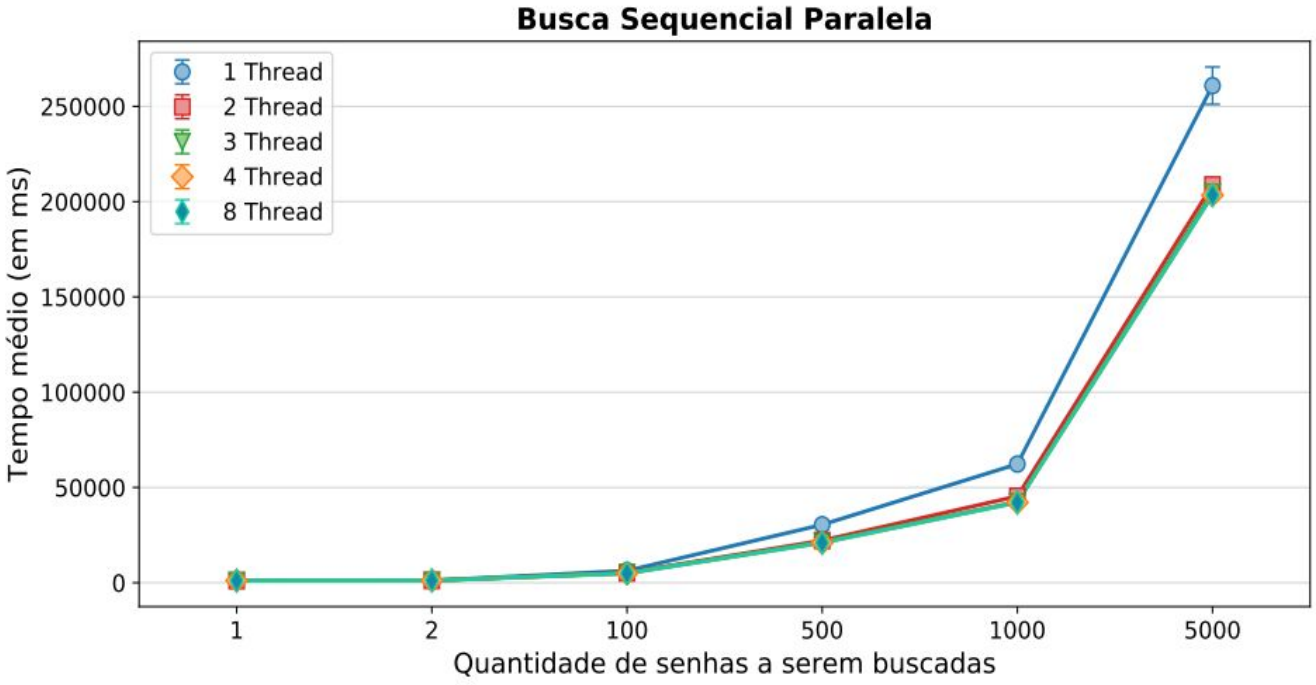
1. Lê os arquivos, das senhas vazadas e a serem verificadas.
    1.1. Verifica a quantidade de linhas existentes em cada para posterior utilização.
    1.2. Armazena as senhas em dois vector's respectivamente.
2. Fecha os arquivos, pois as senhas já estão em vetores na memória e não é mais necessário a utilização dos arquivos.
3. Cria um vetor de Threads e um vetor de TAD's argumentos.
4. Conforme o número de Threads que foram passadas como argumentos, é criado a(s) Thread(s) e essas são executadas. É passado os parâmetros da função de busca para a TAD.
    4.1. Cada Thread busca as senhas a serem verificadas dentro do espaço de busca atribuído a ela. Por exemplo, se foram vazadas 100 senhas e 10 precisam ser buscadas, além disso, é utilizado 2 Threads, a 1ª Thread irá buscar no arquivo de senhas vazadas apenas entre os índices 1 e 50 enquanto a 2ª Thread buscará entre os índices 51 e 100. Em ambos os casos, as 10 senhas seriam buscadas pelas Threads. Ou seja, o espaço de busca no arquivo de senhas vazadas é dividido entre a quantidade de Threads.
    4.2. Para cada senha a ser verificada é feito uma busca no arquivo de senhas vazadas. Se existir, um contador é incrementado. Após todas as verificações na Thread atual, o valor do contador é adicionado em uma variável global que armazena a quantidade total de senhas que foram verificadas e que estavam presentes no arquivo de vazamento. Nessa parte, é utilizado um mutex para controle de acesso a variável global.

5. Após execução das Threads, elas são encerradas e o programa é finalizado.

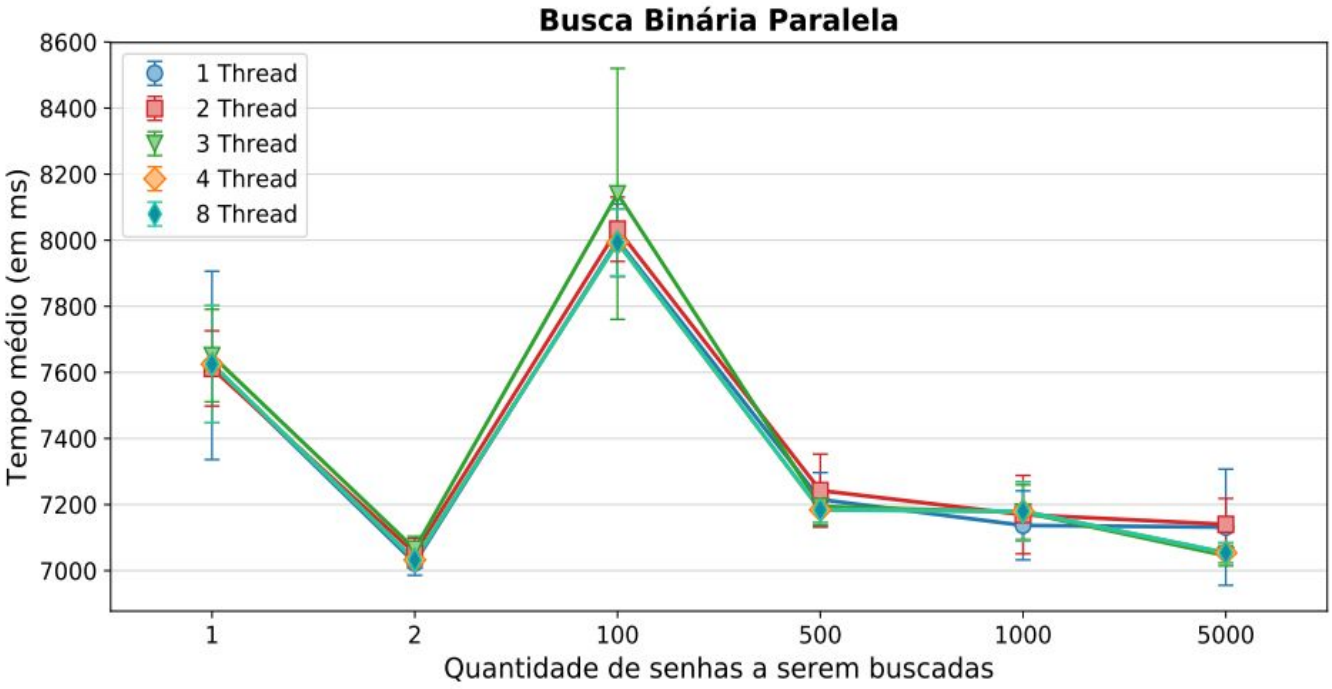
Resultados

A seguir é mostrado os gráficos que foram obtidos em ambas buscas. Neles é feito uma análise do tempo em relação a quantidade de buscas a serem buscadas, e ao tipo de busca sendo utilizada.

Busca Sequencial



Busca Binária



Análise dos resultados

Busca Sequencial

Nesse tipo de busca, conforme a quantidade de senhas a serem buscadas cresceu, maior foi o tempo, algo esperado. O tempo gasto foi bem alto em comparação com a binária, com algumas otimizações no código possivelmente esse tempo pode ser diminuído. Em relação a quantidade de threads, vê-se que o desempenho é consideravelmente pior apenas quando utilizado uma única thread. Com as outras quantidades, o desempenho é similar.

Busca Binária

Na busca binária houve alguns resultados interessantes. Para poucas senhas a serem buscadas, o tempo gasto foi parecido com a busca sequencial, em alguns casos teve até um desempenho pior. Porém, conforme a quantidade de senhas a serem verificadas aumentava, o tempo gasto não houve aumento considerável. Na verdade, com exceção da busca com 100 senhas a serem verificadas, o tempo decaiu. Com 100 senhas a serem buscadas, pode aparentar um desempenho estranho quando visto o resultado no gráfico. Porém ao analisarmos a variância e a diferença para as outras quantidades, vê-se que não há muita diferença. Talvez esse comportamento se deva aos dados contidos no arquivo de 100 senhas.

Em relação a quantidade de Threads, quando utilizado 3 Threads, o arquivo com 100 senhas apresentou uma variância grande em cada execução. A mesma observação, em menor escala, pode ser vista quando utilizada 1 Thread para os arquivos com 1 e 5000 senhas. No mais, houve variações no tempo em relação a quantidade de Threads. As quantidades que apresentaram uma maior estabilidade foram a execução com 4 e 8 Threads.

Conclusões

Para poucas senhas a serem buscadas, 1 ou 2, não existe uma grande diferença no tempo ao utilizar a busca sequencial ou binária. Conforme o número de senhas a serem buscadas aumenta, é preferível a utilização da busca binária, independente da quantidade de threads.

Teoricamente, a busca binária deveria apresentar melhores resultados que a busca sequencial sempre, porém quando a quantidade de senhas a serem buscadas é pequena, o processamento da sequencial é rápido e essa diferença em relação a busca binária não é muito impactante. Como na busca binária há o cálculo para dividir o vetor, o tempo desse

processamento impactou e fez a busca sequencial apresentar melhores resultados para 1 e 2 senhas a serem buscadas. Com quantidades maiores, esse cálculo da busca binária foi irrelevante quando comparado a quantidade de varreduras nos elementos que é feito na busca sequencial, fazendo assim a busca binária apresentar resultados melhores quando procurado uma quantidade grande de senhas. Em determinadas situações, como em 5000 senhas, a busca binária foi até aproximadamentes 30 vezes mais rápida que a sequencial.