Mestrado em Engenharia Informática

PROGRAMAÇÃO CONCORRENTE E DISTRIBUÍDA

Trabalho Prático de Programação Concorrente

a15060 - Luís Castro

a15061 - Filipe Ribeiro

Junho de 2018

Introdução e Objetivos

Com este trabalho pretende-se implementar um programa em JAVA de “força bruta” para encontrar palavras que correspondem a funções de hash SHA-512.

O primeiro passo foi desenvolver uma versão sequencial, tendo como exemplo um projeto[1] aconselhado pelo docente, cujo objetivo foi encontrar uma solução que resolvesse o problema e posteriormente optimiza-la o mais possível. De seguida, tirando partido da programação concorrente implementar um programa paralelo com o mesmo propósito, tendo em atenção que seja sempre produzido o mesmo resultado, sendo o mais eficiente possível.

Por fim serão analisados alguns tempos de execução nas diferentes versões para as palavras apresentadas. Na análise dos tempos, o dispositivo utilizado tem como características: 2,5 GHz Intel Core i5 com 8GB Memória RAM.

[1] https://github.com/cdhop/BruteCrack

Implementação do Problema

1. Apresente o valor de hash SHA-512 em hexadecimal das seguintes palavras: “zoom”, “zigzag”, “zirconium”:

Com o auxílio da função para o cálculo da SHA-512 disponibilizada pelo docente, obtivemos os seguintes valores:

```
> Word to SHA-512 HexHash?
$ zoom
DD8D98401D41639578A18327DC41B706E2F22C5B44648353D4343EDFB26DA8A81E730D115D1E21BE1E1D7EB
30935C0F48D88307E1D64EB89E87B534D8CB671F3
```

Fig. 1 - Hash SHA-512 ‘zoom’

```
> Word to SHA-512 HexHash?
$ zigzag
53DB64256C9C326A42517ACA4AF6A272484AC9EDE9E04AA8D79880B03F2C37164D23C2983625F20069C880B
BFC8A7CDAD5A2C0FD07BC43ADCA27C5A3AA854D5A
```

Fig. 2 - Hash SHA-512 ‘zigzag’

```
> Word to SHA-512 HexHash?
$ zirconium
B099007D9652ED81799EA79E463F86014CDF26CAB79B831CD87C5843990F7599DF9FC35563A20FF4CC91CA2
6A76EC54D3DB65EDB6EC14F4D6F5BB9C673940DE7
```

Fig. 3 - Hash SHA-512 ‘zirconium’’

2. Desenvolva uma aplicação sequencial (i.e., sem concorrência) que recebe como entrada o valor de hash a procurar assim como o tamanho máximo em caracteres das palavras a gerar, e apresenta como resultado a palavra origem que gera esse mesmo hash, o tempo de execução e quantas palavras foram testadas.

Com base no projeto disponibilizado, foi implementado e otimizado uma versão sequencial em que dada uma hash SHA-512 encontra a palavra que a originou. O alfabeto possível contém apenas caracteres de a-z sem caracteres especiais, fazendo um total de 26 caracteres.

A implementação consiste em iniciar o teste com o primeiro caractere do alfabeto, e caso a hash gerada não seja igual à que é passada por parâmetro incrementa a palavra a testar até encontrar a palavra correspondente.

Para as palavras de teste, foram analisadas:

```
> SHA-512 Hash to crack?
$ DD8D98401D41639578A18327DC41B706E2F22C5B44648353D4343EDFB26DA8A81E730D115D1E21BE1E1D7EB30935C0F48D88307E1D64EB89E87B534D8CB671F3
> Maximum of characters?
$ 4

Start Crack...
Word Cracked: zoom
Tested words: 467519
time (secs): 0.847997103
```

Fig. 4 - Versão sequencial zoom

```
> SHA-512 Hash to crack?
$ 53DB64256C9C326A42517ACA4AF6A272484AC9EDE9E04AA8D79880B03F2C37164D23C2983625F20069C880BBFC8A7CDAD5A2C0FD07BC43ADCA27C5A3AA854D5A
> Maximum of characters?
$ 6

Start Crack...
Word Cracked: zigzag
Tested words: 313169201
time (secs): 400.4018595
```

Fig. 5 - Versão sequencial zigzag

Não foi possível testar a palavra ‘zirconium’ devido ao elevado tempo na sua execução.

3. Apresente uma versão concorrente desta aplicação, sendo que o programa deverá receber como entrada e produzir como resultado a mesma informação do programa sequencial. Adicionalmente, esta versão concorrente deve permitir também especificar como parâmetro de entrada o número de cálculos em simultâneo a fazer (i.e., no de threads). De referir que o cálculo do valor de hash é independente para duas palavras distintas, i.e., pode ser concorrente.

Para implementar uma solução paralela, o maior desafio é encontrar uma forma de dividir o processamento pelo número de threads de forma a que todas elas executem todas as operações sem repetir o teste em nenhuma palavra.

A solução implementada consiste em separar o alfabeto pelas threads, ou seja, cada thread é responsável por analisar as palavras que comecem por um conjunto de letras que lhe é atribuído.

Na Fig. 6 no caso da 'thread 0', irá analisar 'a', 'b', até 'h' e de seguida aumenta o número de caracteres, de 'aa' até 'hz' e aumenta novamente. Podemos concluir que a 'thread 0' irá encontrar a palavra caso esta comece entre 'a' e 'h'.

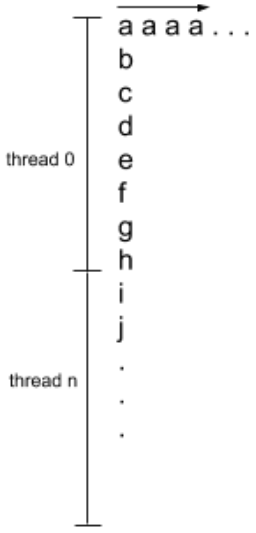


Fig. 6 - Abordagem para a versão paralela

Para uma melhor organização foram criados diversos ficheiros que constituem a solução implementada.

A class "main" cria um novo objeto da classe "Variables" que será passado para cada threads por parâmetro, contento todas as variáveis comuns. Cria também um novo objeto "Result", sendo este partilhado e acedido por todas as threads para guardar a informação e o resultado que alguma thread encontrou a palavra que originou a hash, fazendo com que todas as outras threads terminem a sua execução.
A figura abaixo demonstra toda a arquitetura da solução.

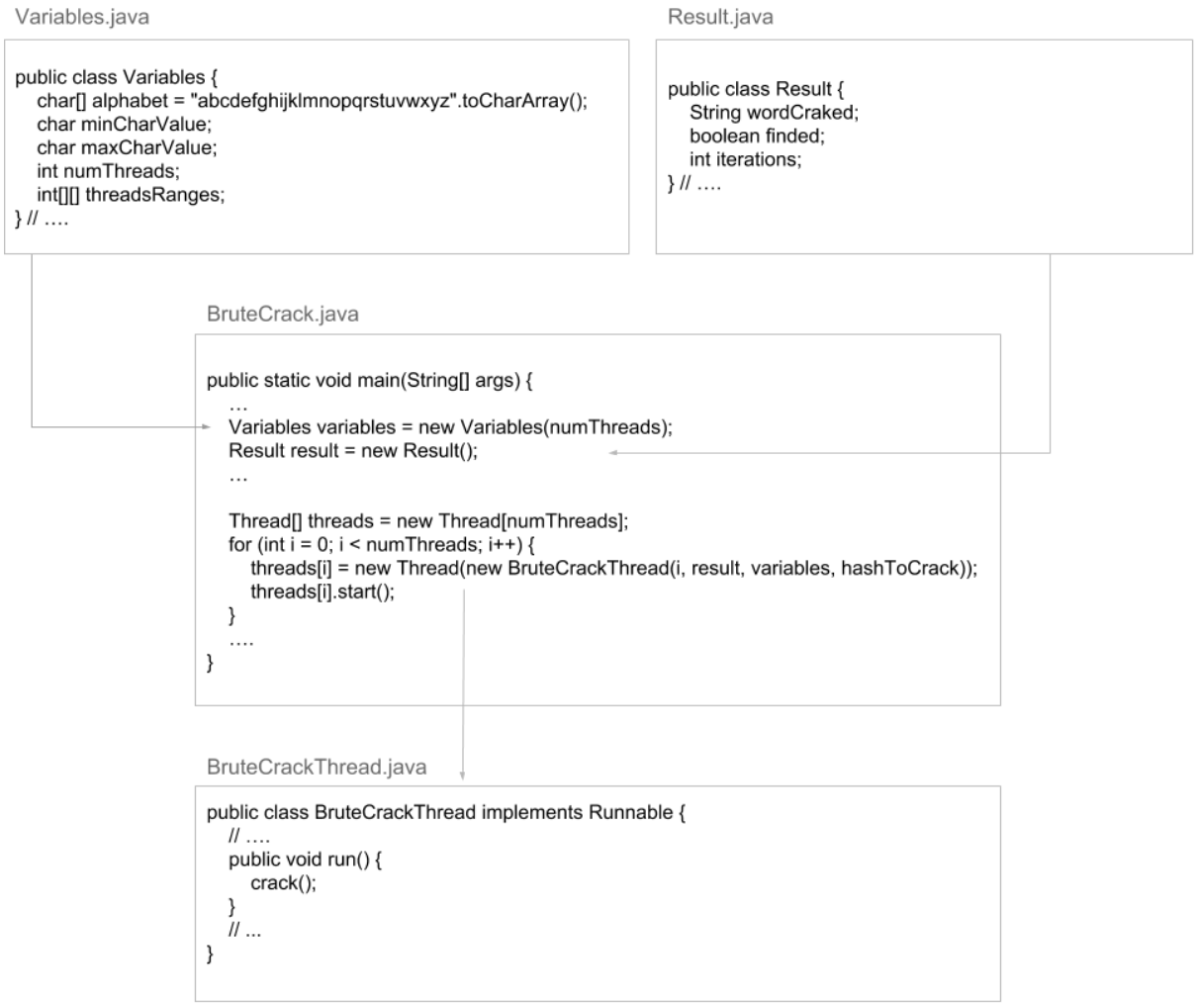


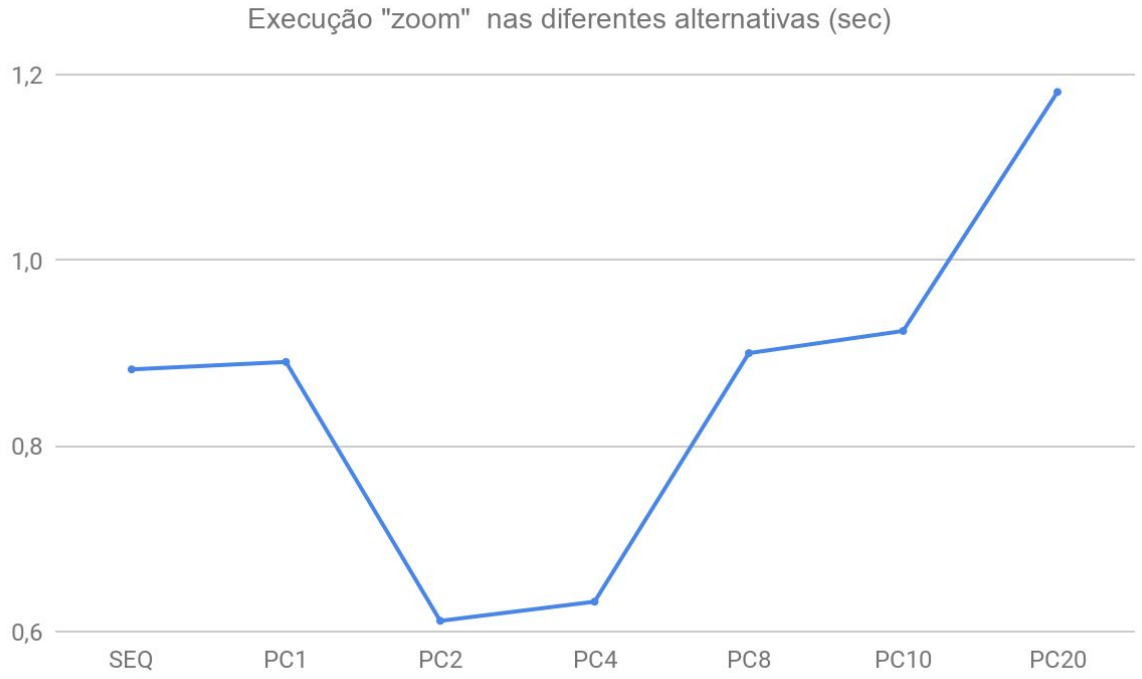
Fig. 7 - Arquitetura da solução paralela desenvolvida

De realçar que cada thread implementa um objeto MessageDigest, e em cada iteração faz reset ao mesmo de modo a garantir uma maior eficácia na sua execução.

Análise de Desempenho

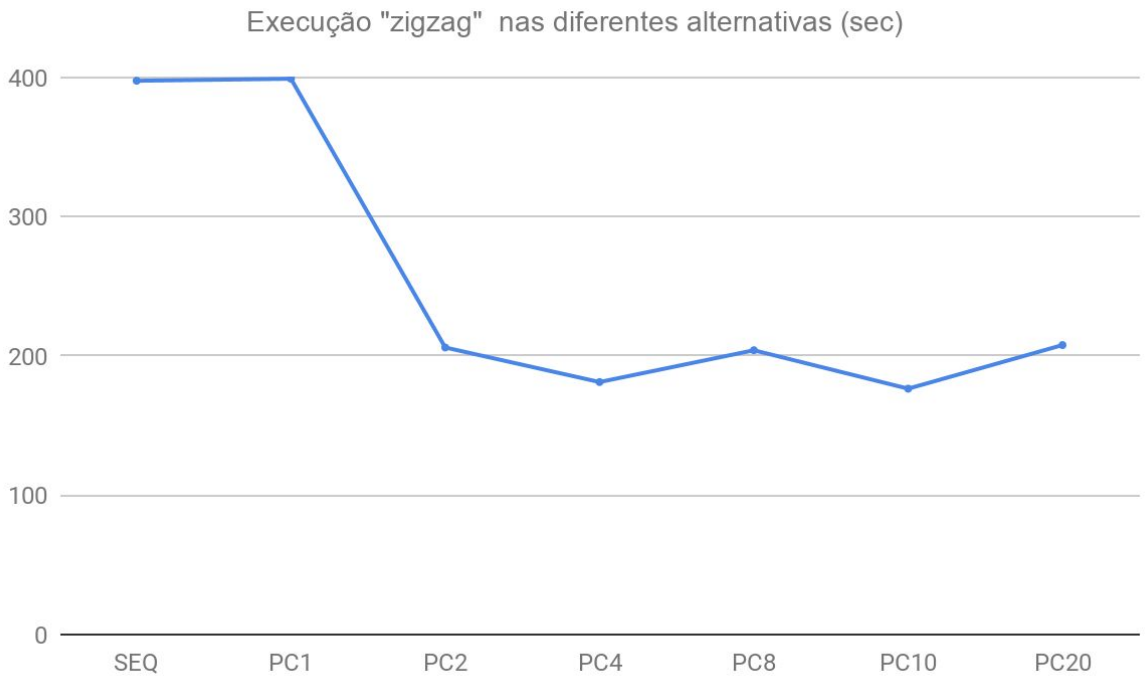
1. Com base nos programas desenvolvidos nos pontos anteriores, construa um gráfico onde representa no eixo dos YY o tempo de execução do programa, e no eixo dos XX o no de threads (assuma como exemplo: sequencial, PC1, PC2, PC4, PC8, PC10, PC20).

Tendo em conta a implementação paralela apresentada anteriormente, foram analisados os tempos de execução das palavras "zoom" e "zigzag". Para cada uma será apresentado um gráfico e os respectivos valores recolhidos nas execuções, utilizando para representação do gráfico a média dos valores recolhidos. Para obter os tempos de execução, foi utilizada a função *System.nanoTime().*



0,838926833	0,877369806	0,5615386	0,6080619	0,879005559	0,988549803	1,470636824
0,876066463	0,886646161	0,649948063	0,561436249	0,715206017	0,840186434	1,286708062
0,861655829	0,915959085	0,581866562	0,680196531	0,816072547	0,909283885	1,090930012
0,887930033	0,875051018	0,702594811	0,658249158	0,803581463	0,82645985	1,422464832
0,86614351	0,878467496	0,595981392	0,613176262	1,065954451	0,7500999	1,165815431
0,86322844	0,934822801	0,657989066	0,576584807	0,851524438	0,905534298	0,959438748
0,951088937	0,896678136	0,644565164	0,636660308	1,010043694	1,256528947	1,312197681
0,965978359	0,881183121	0,553900986	0,624021448	0,836279024	1,144269153	1,014516653
0,864576176	0,874210825	0,596472893	0,673931272	1,087351728	0,747905802	1,022620209
0,847997103	0,883402338	0,570197091	0,689117567	0,933569836	0,868923581	1,06592324

0,882359168	0,890379079	0,611505463	0,632143555	0,899858876	0,923774165	1,181125169
SEQ	PC1	PC2	PC4	PC8	PC10	PC20



392,5269095	419,8137315	211,4385282	184,3710617	214,6590694	172,0741261	210,7658865
396,4681696	396,0885994	219,7969935	178,377615	202,1322101	175,3160206	200,9293536
394,9723377	393,0327769	217,4448514	179,85176	197,2707318	169,0115529	212,2713661
388,4581803	397,7601358	204,9746153	182,2272952	201,9122341	170,8549335	206,7316355
390,4645866	399,5173207	198,6614233	182,0361761	203,5635567	175,8682809	206,8995133
417,3757871	391,6388051	196,9984222	180,8806412	206,7160592	178,3506432	210,4515923
395,5270123	391,633045	198,0863476	181,8835762	200,8551895	172,5613458	200,1280807
393,0703064	396,9750274	199,0437435	182,4206894	206,1417853	180,1371895	205,3465812
407,2171052	410,4606851	204,3926154	181,9366607	199,8567139	191,9072575	211,4581844
400,4018595	393,8042401	210,2461176	178,9834597	208,1174937	179,4190102	213,5541537
397,6482254	399,0724367	206,1083658	181,2968935	204,1225044	176,550036	207,8536347
SEQ	PC1	PC2	PC4	PC8	PC10	PC20

2. Compare o desempenho da execução concorrente com a execução sequencial, calculando o valor de Alfa (percentagem de código sequencial do programa) de acordo com a Lei de Amdahl, tomando como referência os ganhos de desempenho obtidos. Indique o tempo de execução medido para os casos sequencial e concorrente, assim como o número de núcleos do processador. Faça o cálculo para todas as versões concorrente (no de threads).

Com base na versão sequencial, foi calculado recorrendo à Lei de Amdahl a percentagem código sequencial de cada versão. Nesta análise foi apenas utilizado a execução da palavra “zigzag”.

$$Speedup \leqslant \frac{1}{F + \frac{1-F}{N}}$$

Fig. 8 - Lei de Amdahl

Os valores utilizados estão representados na alínea anterior, utilizados na representação do gráfico.

- 1 Thread (N=1)

No caso de ser utilizado apenas uma thread temos um programa idêntico a uma execução sequencial.

- 2 Threads (N=2)

$$\frac{397.6482254}{206.1083658} \leqslant \frac{1}{F + \frac{1-F}{2}} <=> 1.92932 \leqslant \frac{1}{F + \frac{1-F}{2}} <=> F \leqslant 0.0366$$

Com duas threads temos um speedup de 1.93, tendo um valor de F ≤ 3.6%

- 4 Threads (N=4)

$$\frac{397.6482254}{181.2968935} \leqslant \frac{1}{F + \frac{1-F}{4}} <=> 2.19335 \leqslant \frac{1}{F + \frac{1-F}{4}} <=> F \leqslant 0.2746$$

Com quatro threads temos um speedup de 2.19, mais de metade da versão sequencial, tendo um F ≤ 27,5%

- 8 Threads (N=8)

$$\frac{397.6482254}{204.1225044} \leqslant \frac{1}{F + \frac{1-F}{8}} \iff 1.94809 \leqslant \frac{1}{F + \frac{1-F}{8}} \iff F \leqslant 0.4438$$

Com oito threads temos um speedup de 1.95, tendo um F ≤ 44.4%

- 10 Threads

$$\frac{397.6482254}{176.550036} \leqslant \frac{1}{F + \frac{1-F}{10}} \iff 2.25233 \leqslant \frac{1}{F + \frac{1-F}{10}} \iff F \leqslant 0.3822$$

Com dez threads temos um speedup de 2.25, tendo um F ≤ 38.2%

- 20 Threads

$$\frac{397.6482254}{207.8536347} \leqslant \frac{1}{F + \frac{1-F}{20}} \iff 1.91312 \leqslant \frac{1}{F + \frac{1-F}{20}} \iff F \leqslant 0.4976$$

Com vinte threads temos um speedup de 1.91, tendo um F ≤ 49.8%

3. Comente os resultados dos tempos de execução para as várias experiências, apresentando justificações para os valores obtidos.

Para a análise de desempenho, foi apenas utilizado a execução da palavra “zigzag”.

Ao observar os tempos obtidos, a versão sequencial e a versão paralela com apenas uma thread são bastante semelhantes, como seria de esperar, mesmo ao analisar a lei de Amdahl chegamos à mesma conclusão.

O tempo de execução vai diminuindo até à utilização de 4 threads, sendo o número de processadores presente na máquina utilizada. Ao aumentar o número de threads aumenta também o tempo de execução devido ao custo de criar mais threads do que processadores disponíveis. Com oito threads o valor diminui novamente, isto pode ter acontecido devido a divisão do alfabeto pelo número de threads, quando a divisão não é um valor inteiro, a última thread fica com mais carga, sendo que neste caso, como as palavras analisadas começam pela letra ‘z’, são encontradas pela última thread e o tempo pode não ser o esperado.

Com base nos resultados obtidos na alínea anterior, podemos concluir que para a palavra ’zigzag’, utilizando oito threads conseguimos o melhor valor do speedup, com 2.25 em comparação à versão sequencial e ao utilizar apenas duas threads, a menor percentagem de código sequencial, com cerca de 3.6%, sendo possível porque a divisão do alfabeto é uniforme com apenas duas threads.

Conclusão

Concluído este projeto, notamos que, a maior dificuldade foi pensar de forma paralela, de que forma o processamento iria ser dividido entre as threads, e de que forma estas se comunicam entre si para caso uma encontre a palavra, as outras terminam a sua execução.

Para as palavras para análise, apenas foi possível executar a “zoom” e a “zigzag”. No caso da “zirconium”, devido ao elevado tempo de execução não foi possível concluir a sua execução, quer na versão sequencial quer na paralela esteve a executar durante cerca de 6h e não foi concluído.