Processador: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz

APLICATIVOS UTILIZADOS

Foi escolhido quatro aplicativos que fazem a mesma coisa, porém escritos em diferentes linguagens: C, Java, Javascript e Python. Cada um deles foi escrito de forma mais parecida possível (no entendimento da linguagem humana), também foi utilizada a menor quantidade possível de bibliotecas externas, a fim de evitar comparações injustas.

O objetivo do algoritmo implementado é simples: tentar formar uma string que corresponda a outra string de tamanho 6, sendo que ela pode ser formada por qualquer letra minúscula do alfabeto ou número, ou seja, 36 caracteres distintos ou não. Portanto, tal algoritmo funciona para descobrir, por força bruta, a senha de alguém.

Para simplificar o trabalho, já que não é o objetivo dele, foi assumido que uma senha sempre terá 6 caracteres. Além disso, poderia ter sido implementado a conversão de cada string gerada para o padrão hash SHA-256, já que normalmente as senhas salvas nos bancos de dados estão nesse padrão.

A análise de desempenho para diferentes linguagens pode obter resultados completamente diversos, dependendo do algoritmo utilizado, pois cada linguagem pode ser melhor em algumas instruções em relação a outras, portanto este relatório analisa a melhor linguagem para comparar diversas vezes muitos caracteres.

Por fim, caso tenha interesse em ver como foi estruturado os algoritmos em cada linguagem (teste.c, teste.js, teste.py e Teste.java), pode-se acessá-los no link abaixo:

https://github.com/thassis/desempenho-c-java-python-javascript

EVENTOS MONITORADOS

Foram monitorados os seguintes eventos: cpu-clock, cache-misses, cpu-cycles e instructions. Justificativa de cada evento:

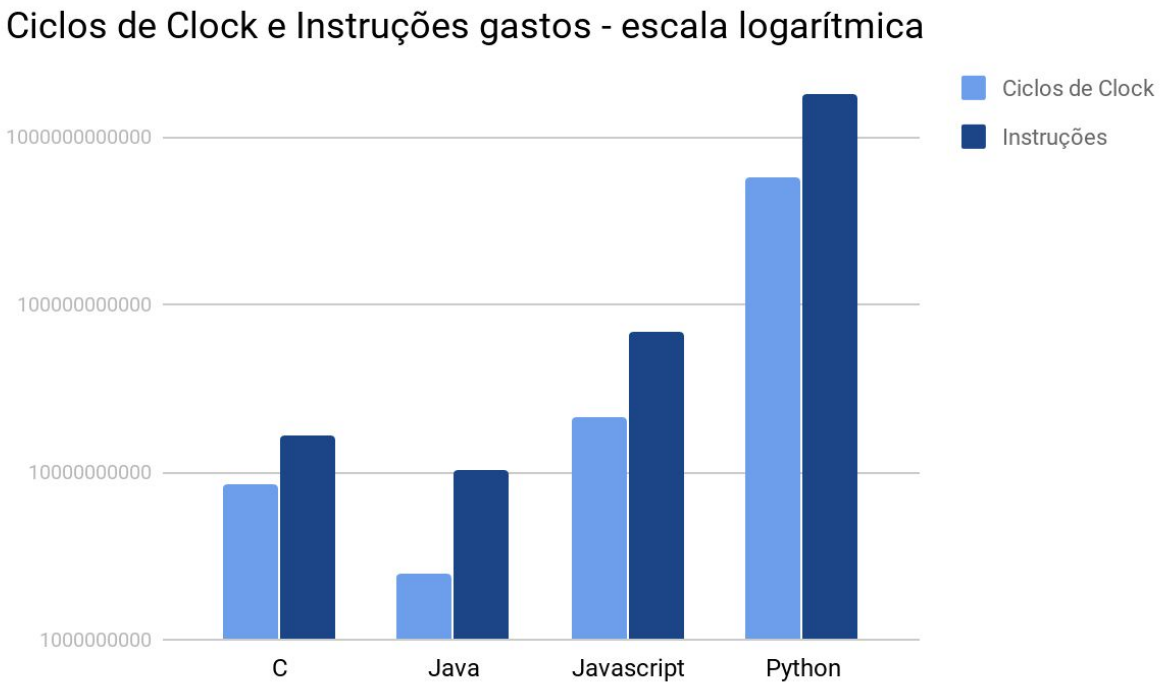
- cpu-clock: verificar o tempo gasto total na CPU, esse evento é importante também porque ele retorna a quantidade usada da CPU, sendo que todos os aplicativos tiveram cerca de 100% de uso da CPU, ou seja, processaram com os mesmos recursos.
- cpu-cycles: quantidade de ciclos de clock necessários para executar o programa. Esse é um monitoramento importante, já que podemos ver como diferentes linguagens precisam de bem mais ciclos de clock para fazerem a mesma coisa.

- instructions: quantidade de instruções executadas pelo processador. Funciona de forma parecida com o cpu-cycles, aquele programa que executa menos instruções para o mesmo código, é o melhor.
- cache-misses: tentativa falha de ler ou gravar um dado no cache, o que resulta numa latência muito maior ao acesso da memória, impactando negativamente no tempo de execução do programa.

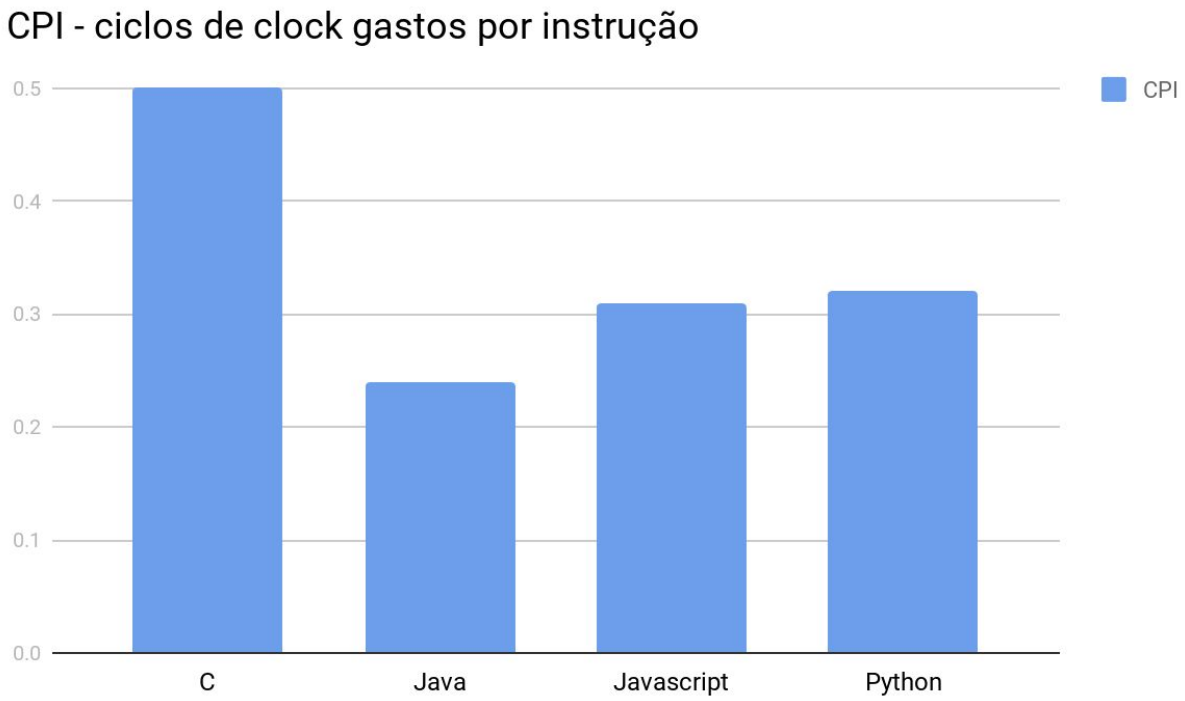
RESULTADOS OBTIDOS

	C	Java	Javascript	Python
CPU - Clock (msec)	**2.579,38**	**800,06**	**6.442,96**	**244.848,99**
Cache misses	**69.551**	**1.534.873**	**1.939.226**	**80.737.700**
Quantidade de Ciclos de Clock	**8.427.557.868**	**2.497.022.307**	**21.350.066.312**	**581.990.248.188**
Instruções	**16.771.957.354**	**10.288.109.339**	**68.301.063.395**	**1.814.652.345.830**

ANÁLISE DOS DADOS



Como esperado, os aplicativos que precisam de menos instruções e menos ciclos de clock tiveram também um menor tempo gasto na execução da CPU, nesse caso o algoritmo implementado em Java foi o melhor.



Ao analisar isoladamente o CPI, naturalmente deduz-se que os que possuírem o menor valor será melhor, já que precisa de poucos ciclos para executar uma instrução. Contudo, o programa em C teve o maior CPI (~0.50), enquanto o programa em Python teve um bem menor (~0.32), mas o tempo de execução do C foi bem menor, pois ele teve bem menos instruções e ciclos gastos no total (em Python foi executado cerca de 108 vezes mais instruções e cerca de 69 vezes mais instruções).

CACHE MISSES

Analisando o Cache Misses, percebe-se que o programa em C teve o melhor desempenho e um dos menores tempos de execução, já que de fato é uma das melhores linguagens (entre as escolhidas) para se trabalhar diretamente com a memória, ou seja, quanto menos dados perdidos entre o cache, menor também será o tempo gasto na execução do programa, diminuindo a perda de tempo entre a procura de dados para leitura/escrita nas camadas de memória cache.

Importante ressaltar que esse evento não determina sozinho o tempo que será gasto na CPU, já que o programa em Java teve cerca de 22 vezes mais Cache Misses do que o programa em C, mas no fim teve um tempo bem menor.

CONCLUSÃO

Considerando o menor tempo gasto para definir a melhor linguagem para o algoritmo analisado, conclui-se que Java teve o melhor desempenho, seguido por: C; Javascript; Python.

Os dados foram um pouco diferente do esperado, pois eu pensei que C seria bem menor, enquanto Python poderia ser a pior, mas não com um valor tão superior às outras linguagens.

Tal ordem pode ser justificada claramente pelos dados obtidos pelo *perf*, conforme análise já feita. Mas justificando também a nível de software, linguagens interpretadas, como Javascript (Js) e Python, tendem a ser mais lentas. Contudo, o compilador e interpretador utilizado para Js foi o Node.js (desenvolvido em C++), que já foi feito para ter um alto desempenho, enquanto o Python foi puramente interpretado.

Além disso, uma justificativa para que Java tenha tido menos ciclos de clock e menos instruções executados é que o compilador dessa linguagem tenha feito otimizações bem melhores, mas se fosse analisado o tempo total gasto (desde a compilação até a execução), pode ser que Java precise de um tempo maior que C, já que Java envolve mais processos para compilar e interpretar sua linguagem na JVM.