WebCrawler - MagazineLuiza

Desempenhos variados conforme execução sequencial ou paralela e número de threads

Gabriel do Vale Rios, Engenharia de Computação. Insper-2018.

Introdução

Hoje em dia é muito comum a pesquisa de diversos produtos em websites. Diversas lojas e empresas possuem uma boa logística de *e-commerce,* o que torna rápido e fácil o acesso e compra de itens variados. Porém, muitas vezes os sites apresentam diversos produtos, e a comparação de preços se torna lenta e tediosa, fazendo com que o consumidor, às vezes, não opte pela melhor opção possível de acordo com suas preferências. Assim surgiram os sites comparadores de preços, onde o consumidor não mais precisa manualmente fazer a comparação de diversos preços, checar informações e etc. Tais comparadores utilizam de uma técnica chamada "*Crawling"*, onde um ou diversos computadores pesquisam e analisam os produtos de acordo com um filtro, este fornecido pelo consumidor.

O trabalho citado neste relatório se inspirou em um destes analisadores de produtos automático, ou ainda "*WebCrawlers*". O objetivo é programar 2 algoritmos que realizam as tarefas de download de produtos e análise de suas informações, cada algoritmo realizando a mesma tarefa com estratégias diferentes:

- O primeiro algoritmo analisará os produtos de uma certa categoria de modo sequencial,
- O segundo algoritmo analisará os produtos de uma certa categoria de modo paralelo, utilizando um número variado de threads de acordo com o desejo do usuário do código.

Por fim, serão comparados os desempenhos dos tempos entre as estratégias, sequencial e paralelo, junto da variação do número de threads.

Implementação

Todo código citado neste documento pode ser baixado e executado a partir do endereço abaixo:

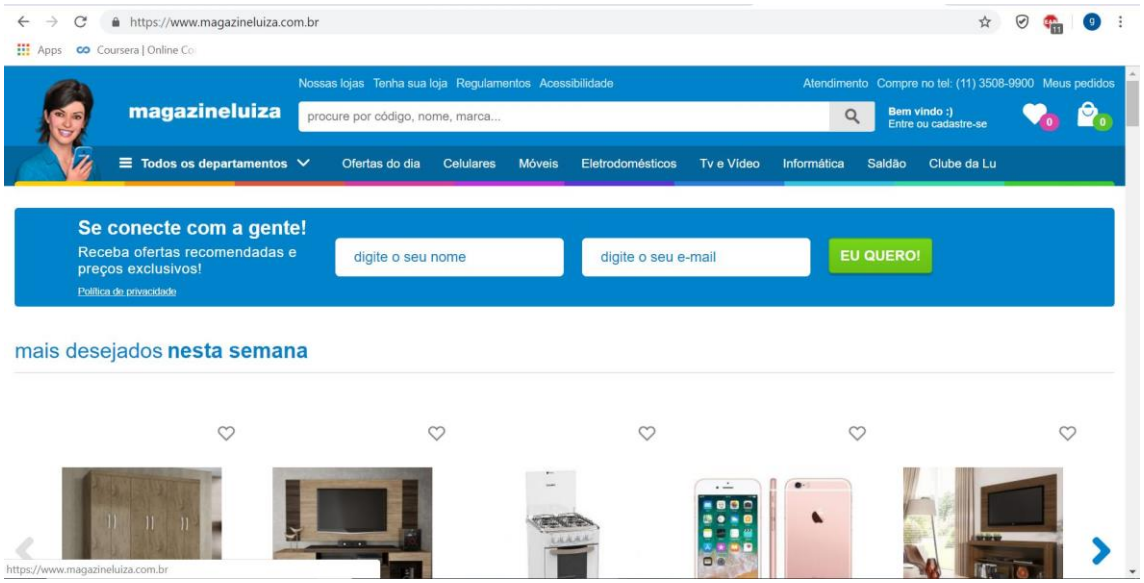
https://github.com/GabrielValeRios/superComp2018/tree/master/projeto2

Para a implementação e gestão de todo o código feito para esta simulação, se utilizou a linguagem C++, python e Cmake. O hardware utilizado para compilação e execução das simulações segue abaixo, junto de suas informações:

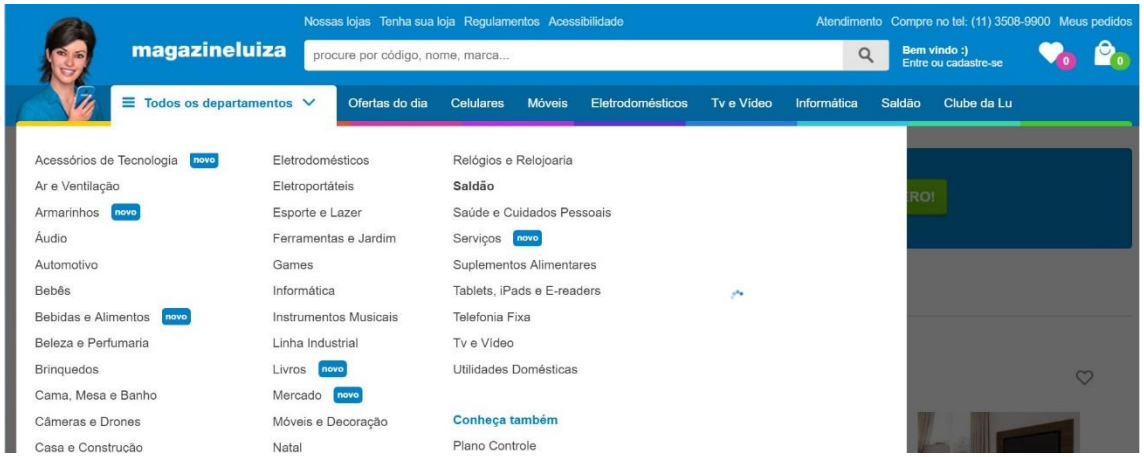
- Processador: Intel® Core™ i7-4500U CPU @ 1.80GHz 2.40GHz
- Memória (RAM): 16.0 GB
- Tipo de Sistema: Sistema Operacional de 64 bits, processador com base em x64, 4 cores
- Sistema Operacional utilizado: Distribuição Linux (Ubuntu 18.04 LTS)

O código busca simular um "*WebCrawler*", onde informações de produtos de uma certa categoria serão baixadas e processadas. Para a realização dos testes, a empresa escolhida foi a MagazineLuiza, e abaixo segue um passo-a-passo de como navegar pelo seu site em busca de uma URL válida, ou seja, uma URL que mostra os produtos de uma certa categoria:

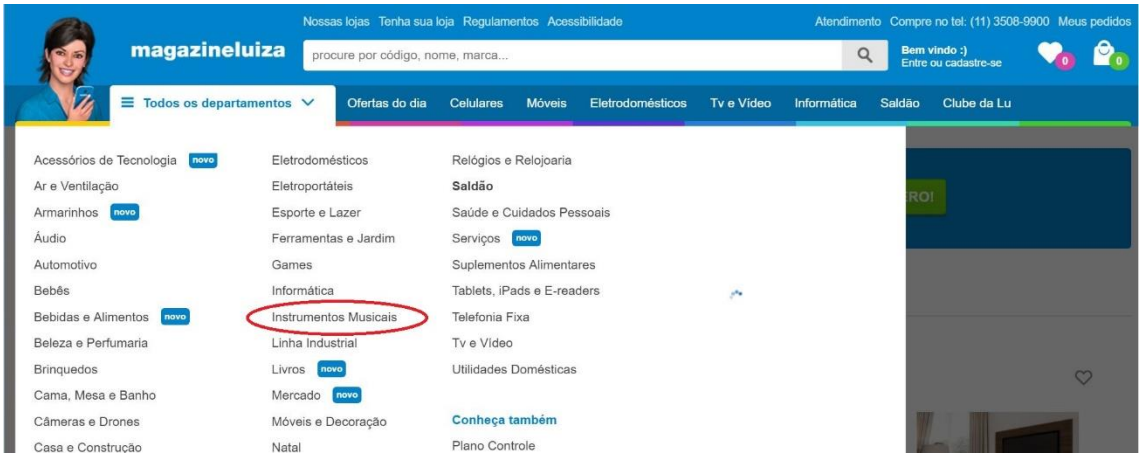
1. Entrar em www.magazineluiza.com.br



2. Nesta página, selecionar a aba "Todos os departamentos"

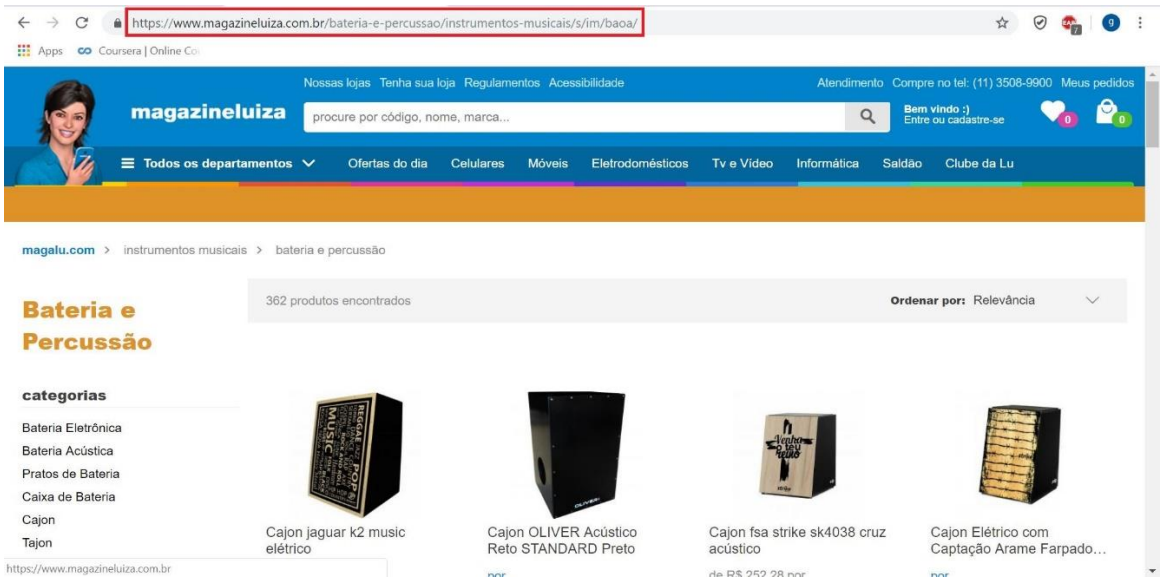


3. Na aba selecionada, para fins de replicar os testes feitos, selecione a aba “instrumentos musicais”.



4. Uma vez que se selecionou esta aba, selecione “Bateria e Percussão”

5. Por fim, o link a ser utilizado para o “crawling” é o indicado na região abaixo



Link: https://www.magazineluiza.com.br/bateria-e-percussao/instrumentos-musicais/s/im/baoa/

- Arquivos em C++

Os arquivos deste projeto foram separados entre os que lidavam com a programação sequencial e paralela. Assim:

- Para gerar os testes com programação sequencial, foram utilizados os arquivos mainS.cpp, crawler_seq.cpp e produto.cpp,
- Para gerar os testes com programação paralela, foram utilizados os arquivos mainP.cpp, crawler_par.cpp e produto.cpp.

O arquivo produto.cpp é responsável por formatar as informações de cada produto em JSON. Para isso, cada produto analisado é um objeto e existe o método “to.Json()” que realiza essa tarefa.

Os arquivos mainS.cpp e mainP.cpp são responsáveis por chamarem uma primeira função que irá desencadear uma série de outras chamadas de funções, que serão responsáveis por realizar o crawling.

Por fim, crawler_seq.cpp e crawler_par.cpp são os responsáveis por realizar o crawling. No caso sequencial, o algoritmo crawler_seq irá baixar a página de um produto e processar suas informações, e assim por diante, um produto de cada vez.

Já no caso paralelo, será adotado o modelo “Produtor-Consumidor”. Neste modelo, usam-se threads para realizar tarefas simultâneas, onde algumas destas tarefas serão responsáveis por encontrar os produtos a serem analisados nas páginas, e outras tarefas irão processar suas informações, ou seja, nome, preço, descrição, etc. Deste modo, espera-se um ganho de desempenho em virtude de múltiplas tarefas estarem sendo feitas ao mesmo tempo.

Assim, é usado um buffer (vetor) compartilhado onde, a todo momento, URL's de produtos estão sendo armazenadas para processamento, e consumidas para visualizar as informações de um determinado produto. Para que não haja resultados inesperados, utiliza-se uma estratégia de controle chamada semáforo, para que, por exemplo, nenhuma tarefa leia uma URL de um produto mais de uma vez, ou ainda deixe de ler algum produto porque sua URL foi sobrescrita.

- Arquivos em Python

Para que a simulação dos testes fosse feita de maneira menos tediosa, existem 3 arquivos em python que ajudam a realizar os testes:

- buildFiles.py, responsável por gerar o MakeFile através do CMake, e executá-lo, também gerando os executáveis.
- sequentialRun.py, responsável por executar o crawler sequencial para uma certa categoria
- parallelRun.py, responsável por executar o crawler paralelo para uma certa categoria

NOTA: Mais informações de como realizar os testes e usar seus resultados para interpretação podem ser encontradas em README, no link:

https://github.com/GabrielValeRios/superComp2018/tree/master/projeto2

- CMake

Para cada simulação, paralela e sequencial, foi necessária a criação de um executável diferente. Além disso, o uso de threads e expressões regulares (regex) exigem flags de compilação próprias.

Para satisfazer tais exigências, o arquivo CMakeLists.txt possui todas as configurações necessárias, tanto para o sequencial quanto para o paralelo.

Assim, as flags a serem usadas foram:

- Para o executável do crawler paralelo - "-lpthread -lboost_regex"
- Para o executável do crawler sequencial - "-lboost_regex"

Os executáveis gerados são:

- crawler_sequential
- crawler_parallel

Simulação e Comparação

Para a simulação e análise de resultados, foi escolhida a categoria "Bateria e Percussão", de instrumentos musicais. O link para acesso é:

https://www.magazineluiza.com.br/bateria-e-percussao/instrumentos-musicais/s/im/baoa/

Assim, para cada simulação de um executável, são gerados também 3 arquivos texto ao final da execução. São eles:

- AverageTimePerProductParallel/Sequential, responsável por dizer, no fim da simulação, o tempo médio gasto por produto, ou seja, o tempo total de execução do programa dividido pelo total de produtos analisados,

- parallel/sequentialTimes, responsável por dizer, no fim da simulação, o tempo total gasto no download e análise de cada produto,

- WaitTimeParallel/Sequential, responsável por dizer, no fim da simulação, o tempo total ocioso, ou seja, o tempo que o algoritmo espera os downloads das páginas dos produtos.

Para os testes paralelos, foram usados 6 diferentes números de threads. Para cada simulação, sequencial e as 6 diferentes simulações paralelas, os dados de tempo foram guardados, e posteriormente analisados em forma de gráfico. Seguem abaixo os gráficos de análise dos resultados.

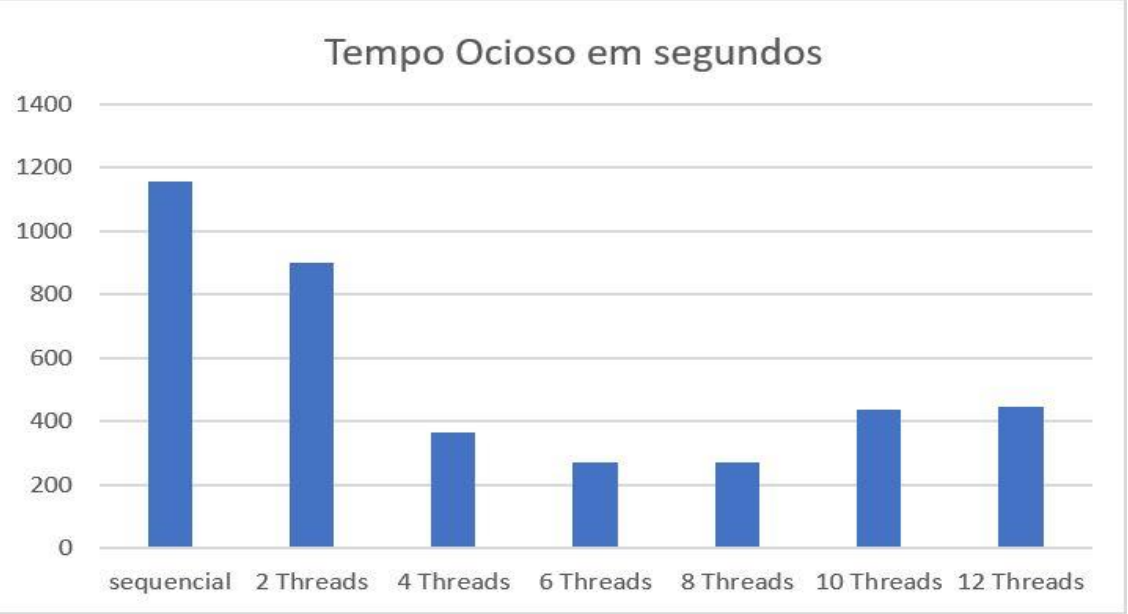


Figura 1 tempo ocioso

Neste gráfico, para cada simulação, foi medido o tempo de espera total do algoritmo. O tempo de espera é igual a soma de todos os tempos de download das páginas dos produtos

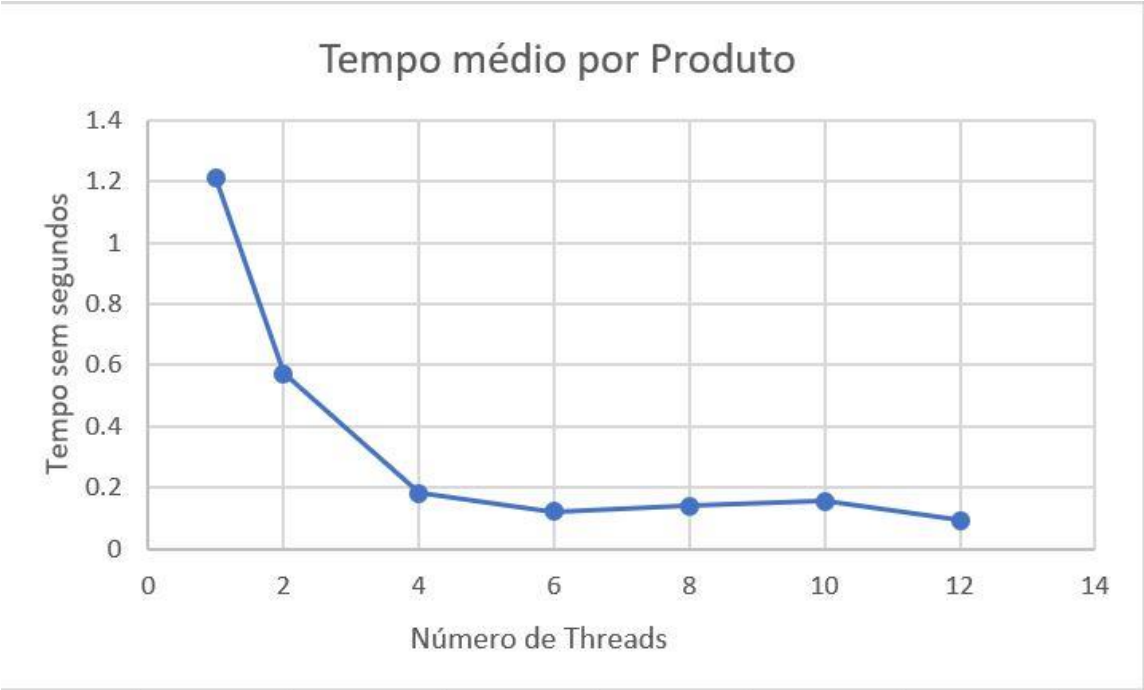


Figura 2 tempo médio por produto

Neste gráfico, foram medidos os tempos de download e análise de cada produto, ou seja, quanto tempo aquele produto demorava desde sua busca até o processamento de suas informações. O primeiro ponto (número de Threads = 1) representa a versão sequencial do algoritmo.

A fim de se obter mais informações a respeito das diferenças entre as simulações, foi medido o uso de memória. Segue abaixo os gráficos com as informações de uso de memória para cada simulação.

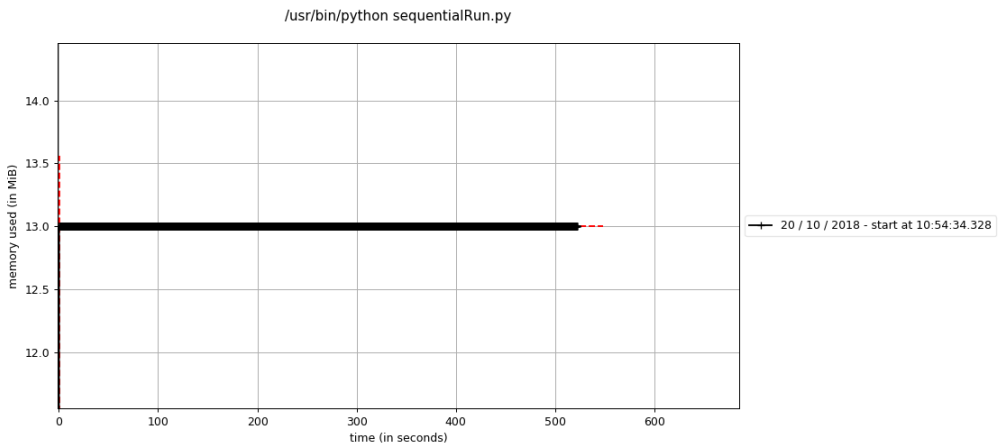


Figura 3 Uso de memória para sequencial

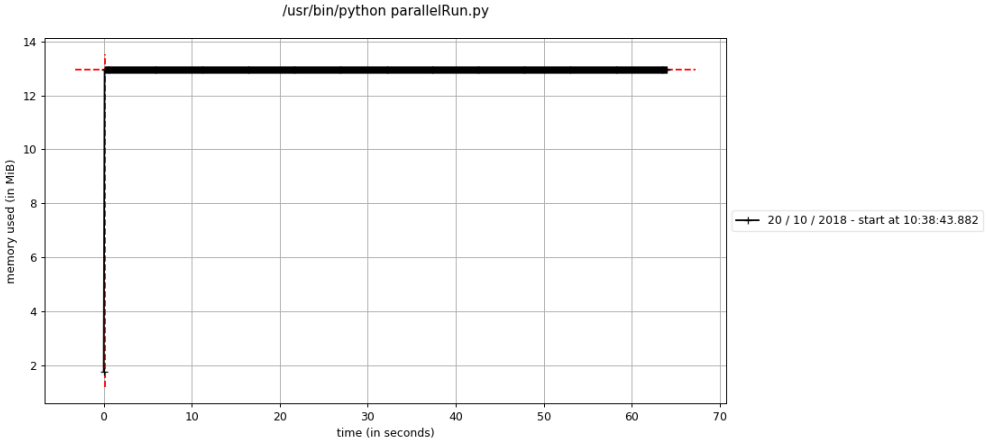


Figura 4 Uso de memória para paralelo 2 threads

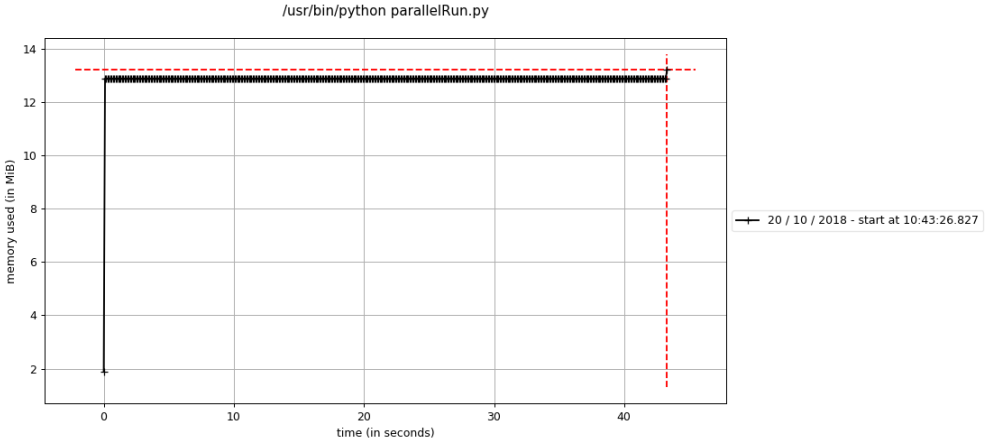


Figura 5 Uso de memória para paralelo 4 threads

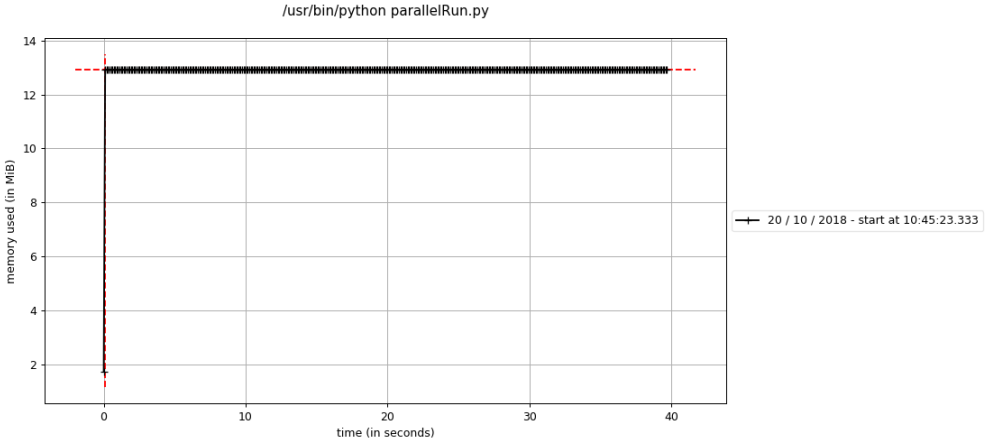


Figura 6 Uso de memória para paralelo 6 threads

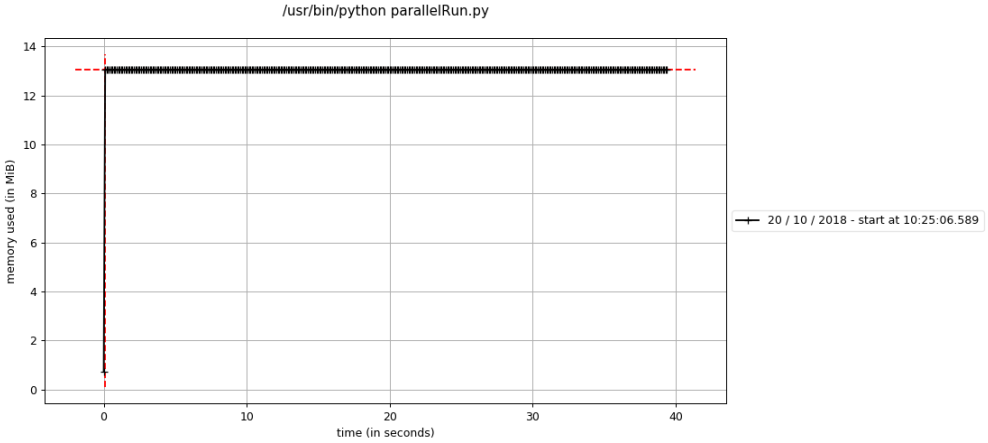


Figura 7 Uso de memória para paralelo 8 threads

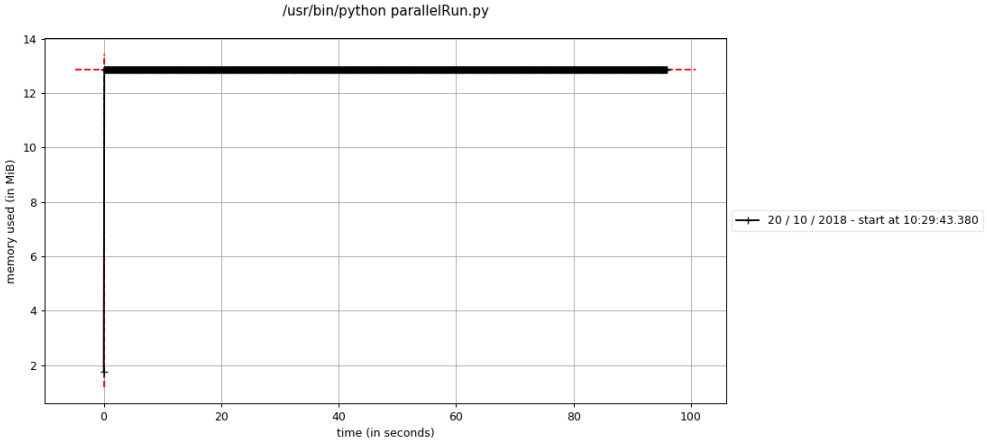


Figura 8 Uso de memória para paralelo 10 threads

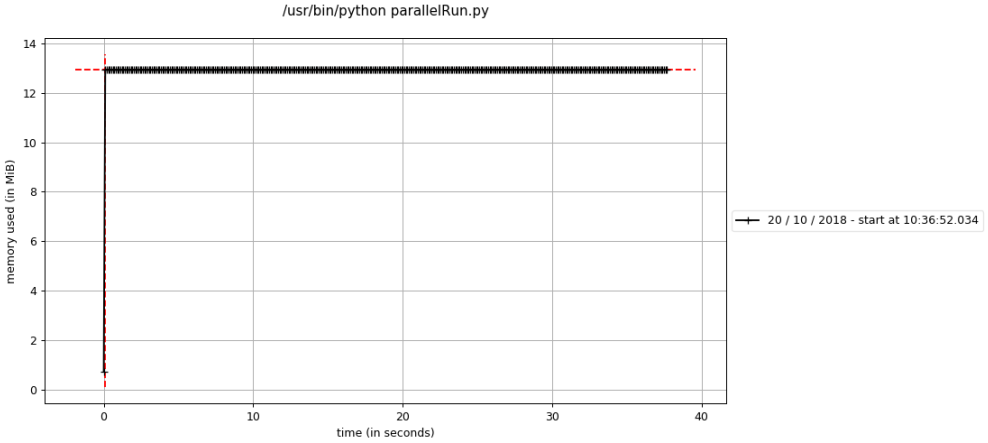


Figura 9 Uso de memória para paralelo 12 threads

Análise e Conclusão

Em um primeiro momento, pode-se observar que com o aumento de threads no código paralelo, existe menos tempo ocioso no geral (figura 1). Isso significa que existe menos perda de tempo no geral, pois enquanto o algoritmo está esperando o download de uma página de produto, ele também busca processar dados já baixados em um buffer. Porém, a partir de 6 threads, o tempo ocioso do sistema volta a subir. Isso pode ser explicado pelo fato de que o número de threads que o computador tenta disparar é maior do que ele pode processar de maneira ótima, assim começando a ter um aumento no tempo ocioso pois existem mais threads que não estão executando tarefas.

Já na figura 2, pode-se ver uma grande queda no tempo médio de produtos do sequencial para o paralelo. Conforme o número de threads aumenta, o tempo diminui, até estabilizar por volta de 6 threads. Isso ocorre pois, em paralelo, assim que a página de um produto já foi baixada, existe uma thread consumidora pronta para o processamento de dados, assim diminuindo o intervalo de tempo de análise de cada produto.

A partir da figura 3, pode-se ver que o uso de memória é o mesmo para todas as simulações. Porém, esses dados geram duvidas, pois com o aumento do número de threads seria esperado uma mudança no consumo de memória. Portanto, apesar das medições, é necessária uma reavaliação de como as medições foram realizadas e não se pode concluir, pelo menos neste primeiro estudo, que o uso de memória é constante.

Conclui-se, portanto, que a execução de um crawler em paralelo é vantajosa em relação a uma estratégia sequencial. Porém, é necessário estar ciente dos recursos das máquinas onde o processamento será realizado, pois múltiplasthreads podem ser ineficazes ou ainda podem prejudicar o desempenho de um algoritmo se mal utilizadas.