Explorando Paralelismo em Python para Múltiplas Execuções de Modelos de Culturas Agrícolas

Lucas F. da Silva[1], **Andrea S. Charão**[1], **Romulo P. Benedetti**[3], **Nereu A. Streck**[3]

[1] Curso de Bacharelado em Ciência da Computação

[2]Departamento de Linguagens e Sistemas de Computação

[3]Departamento de Fitotecnia

Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brasil

{lferreira, andrea, rbenedetti}@inf.ufsm.br

Resumo. *Neste trabalho, explorou-se um dos recursos de paralelismo disponíveis para a linguagem Python, o multiprocessing, no contexto da paralelização de um script que automatiza execuções de modelos agrícolas. Como resultado, obteve-se uma melhoria de desempenho em relação ao script original, reduzindo o tempo necessário para obter os dados de saída desejados.*

1. Introdução

A linguagem Python tornou-se popular em uma ampla gama de aplicações, prestando-se desde à iniciação à programação até à computação científica. Trata-se de uma linguagem multi-paradigma, interpretada, projetada para favorecer a legibilidade do código, priorizando-a em relação à velocidade de execução. Embora inicialmente distante da computação de alto desempenho, onde predominam linguagens como Fortran e C/C++, a linguagem Python tem despertado o interesse da comunidade dessa área. Exemplos típicos nesta linha são pacotes para multiprocessamento ou ferramentas voltadas à programação em Python com aceleração em GPUs.

Na Universidade Federal de Santa Maria, uma colaboração entre o Departamento de Fitotecnia e o Departamento de Linguagens e Sistemas de Computação têm propiciado o desenvolvimento de diferentes tipos de software visando a modelagem de culturas agrícolas. Nesta colaboração, modelos que efetuam cálculos em Fortran são acoplados a interfaces gráficas em Java ou Javascript, produzindo ferramentas disponíveis ao público, como por exemplo SimulArroz, Simanihot e Phenoglad, respectivamente para culturas de arroz, mandioca e gladíolo [CropModelsUFSM 2017].

Em pesquisas que utilizam esses modelos agrícolas, é comum a necessidade de coleta de grandes quantidades de dados sobre as interações genótipo-ambiente de cada cultura. Assim, faz-se necessária a execução de modelos diversas vezes, alternando-se seus parâmetros, para que todos os fatores a serem analisados sejam atendidos. Neste trabalho, explora-se uma alternativa de programação paralela na linguagem Python para acelerar conjuntos de execuções desses modelos.

2. Fundamentação

Existem distintos pacotes para multiprocessamento disponíveis para Python, porém o uso da linguagem para computação de alto desempenho ainda é pouco frequente. Uma razão

para isso pode estar na forma de como o multiprocessamento é realizado em CPython, implementação principal e mais usada da linguagem, pois o interpretador do CPython conta com o *Global Interpreter Lock* (GIL) em cada um de seus processos, o que faz com que as subrotinas concorrentes sejam literalmente desativadas, segundo [PythonDocs 2017].

Mesmo com as limitações relacionadas ao processamento paralelo em Python, opções como o módulo *threading* e o pacote multiprocessing podem trazer bons resultados dependendo da finalidade do algoritmo em questão. O módulo *threading*, baseado na interface de *threads* da linguagem Java, oferece uma solução de mais alto nível para manipulação de *threads*, abstraindo a complexidade da implementação de funcionalidades mais refinadas no módulo Thread, seu antecessor. Já o pacote multiprocessing oferece simultaneidade local e remota, pois são utilizados subprocessos em vez de *threads*, evitando-se assim o bloqueio gerado pelo GIL no interpretador. Isso possibilita uma melhor utilização dos processadores de uma mesma máquina, pois passa a ser possível a distribuição dos vários subprocessos para os processadores de forma mais homogênea, o que melhora significativamente o desempenho do algoritmo que utiliza multiprocessing.

2.1. Modelos agrícolas

Os modelos agrícolas são softwares que simulam o crescimento e o desenvolvimento das plantas. Os dados sobre o clima, o solo e o manejo das culturas são processados para prever o rendimento, data de maturidade, eficiência dos fertilizantes e outros elementos da produção agrícola. Os cálculos nos modelos de culturas baseiam-se no conhecimento existente da física, fisiologia e ecologia das respostas das culturas ao meio ambiente ao qual estão inseridas [Dourado-Neto et al. 1998].

Para sua execução, um modelo necessita da entradas de alguns dados que servirão de parâmetros para as equações matemáticas descritas no algoritmo e que representam algum fator de influência no processo de desenvolvimento da planta. Nos modelos desenvolvidos na UFSM, mais especificamente no modelo PhenoGlad, é necessário que se informe como entrada a localidade (onde se desenvolverá a cultura) e seus respectivos dados de temperatura (valores máximos e mínimos). Além disso, em tempo de execução, o modelo requer informações referentes ao dia e ano que se deseja realizar a simulação e também algumas características sobre a cultivar e seu ciclo de desenvolvimento. Ao final da execução, o modelo entrega como saída um arquivo de texto com dados tabulados representando o desenvolvimento diário da planta durante todo período simulado.

Para execuções do modelo PhenoGlad com uma única configuração de dados de entrada, o tempo de execução não é um fator problemático. Contudo quando se deseja realizar um processamento que consiga abranger várias datas, cultivares e regiões, passam a ser necessários o uso de *scripts* para a automação desse processo, os quais acabam por demandar um considerável tempo de execução, pois realizam diversas simulações intercalando diversas opções de entrada, alcançando um tempo total de processamento na unidade de horas.

3. Paralelização

A solução proposta baseia-se na melhoria do desempenho do *script* Python que faz a automação das múltiplas execuções dos modelos agrícolas, alternando os dados de entrada referentes ao local de plantio, dia de início da simulação e ano de início da simulação. Os

dados resultantes das múltiplas simulações são utilizados para realizar o zoneamento da cultivar de gladíolo para as inúmeras regiões aptas ao seu cultivo no Rio Grande do Sul. Sendo assim, o número de execuções do modelo Phenoglad segue o seguinte padrão: para cada um dos 55 anos de dados meteorológicos, executa-se uma simulação para cada um dos 365 deste ano, multiplicado pelo número total de regiões.

Ao observar o tempo gasto, na escala de horas, para a realização das múltiplas execuções do modelo PhenoGlad, pode-se perceber a necessidade da otimização de tal processo. Com a implementação do *script* utilizada até então, não era possível o aproveitamento de todo o recurso de processamento da máquina, pois o *script* era executado em um único processo, saturando apenas um dos *cores* do processador, deixando os demais ociosos. Pensando nisso, encontrou-se no pacote multiprocessing do Python uma alternativa para a melhoria do desempenho do *script* por meio do paralelismo de processos.

O *script* pode ser dividido em 3 partes, são elas: chamada de execução de simulações, filtragem dos dados dos arquivos de saída de cada simulação e contagem de danos sofridos pela planta. Ao realizar uma análise do comportamento do *script*, percebe-se que a maior carga de trabalho concentra-se da parte que faz as chamadas de execução do modelo, consumindo em média 98,23% do tempo de execução total do *script*. A porcentagem de tempo gasto com a filtragem e a contagem é de 1,76% e 0,0033% respectivamente.

A implementação de uma versão paralela do *script* Python, utilizando o pacote multiprocessing, somente foi possível graças à independência existente entre cada execução do modelo agrícola. O processo de paralelização baseou-se na distribuição das múltiplas execuções entre processos distintos, por meio do objeto Pool do multiprocessing, processando assim mais de uma simulação simultaneamente, não havendo memória a ser compartilhada e nem regiões críticas entre os processos, pois cada execução do modelo escreve os resultados em um arquivo diferente.

4. Experimentos e Resultados

Foram realizados testes variando os dados de entrada referentes ao dia e ano de início da simulação e limitando a execução para apenas uma das regiões produtoras de gladíolo. Assim, o *script* automatizou execuções do modelo para cada um dos 365 dias de cada ano dos 55 anos de dados meteorológicos disponíveis, totalizando 20.075 (55 x 365) execuções. Para validar os tempos de execução dos testes com diferentes números de processos, foram realizadas 10 execuções do *script* para cada uma das configurações distintas e, posteriormente, feita a média dos tempos obtidos.

O *hardware* utilizado para a execução dos testes é um computador *desktop*, onde geralmente as simulações e execuções de *scripts* que envolvem o modelo são realizadas. A máquina é equipada com um processador Intel Core i5-2410M de 2.30GHz com 2 núcleos e 4 *threads*, 6 GB de RAM DDR3 e sistema operacional Debian GNU/Linux 9.

Os resultados para os testes com 2, 4, 6 e 8 processos por *pool* do multiprocessing, podem ser vistos de forma sintetizada na Figura 1. Com os tempos de saída para cada um dos testes pode se observar que a implementação paralela se mostrou bastante eficiente e, desde a execução do *script* paralelo com 2 processos no pool do multiprocessing, pode ser notada uma redução de mais da metade do tempo de execução. Porém, é possível

observar também que a diminuição dos tempos de execução alcança seu limite ao passo que o número de processos se aproxima do número de *cores* do processador.

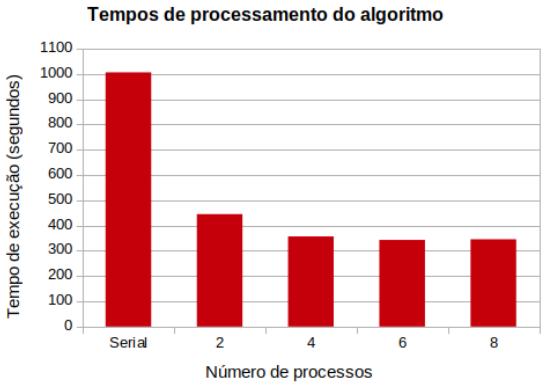

Figura 1. Tempos de execução com o número de processos por *pool*

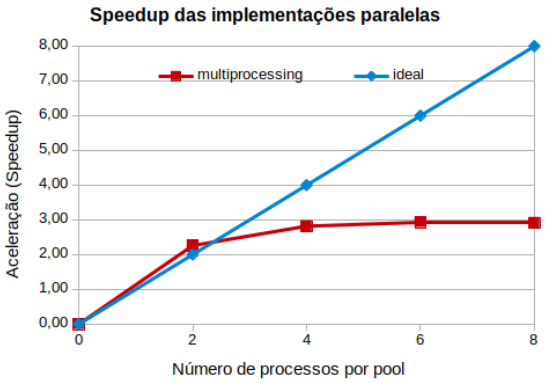

Figura 2. *Speedup* obtido com as implementações paralelas

Com os testes, é possível notar também uma das limitações do paralelismo com processos, que é a sobrecarga que pode ser gerada na fila de escalonamento do processador quando muitos processos, mais que o números de *cores*, são demandados para execução. Esse fator pode ser observado na Figura 2, que apresenta o gráfico do *speedup* atingido. Quando o número de processos chega a 8 (dobro do número de *cores* disponíveis), há uma ligeira queda no *speedup* com relação aos testes em que o número de processos não extrapola o número de cores do processador.

5. Considerações Finais

Os resultados obtidos se mostraram satisfatórios e o paralelismo com processos, utilizando o pacote multiprocessing do Python, se revelou uma solução viável para a otimização dos *scripts* que automatizam as múltiplas execuções dos modelos agrícolas. Algumas limitações do paralelismo com processos foram notadas. Entretanto, para a utilização do *script* para a coleta de dados, elas podem ser contornadas com uma administração do número de processos de cada *pool*. Como trabalhos futuros, pretende-se avaliar a escalabilidade do *script* paralelizado, realizando testes mais aprofundados em uma máquina de processamento de alto desempenho, bem como explorar outras aplicações as quais o paralelismo de processos possa ser implementado.

Referências

CropModelsUFSM (2017). Modelos matemáticos de culturas agrícolas da ufsm. `http://www.cropmodels.ufsm.br/sobre/`. Acesso em: 10 Ago. 2017.

Dourado-Neto, D., Teruel, D. A., Reichardt, K., Nielsen, D., Frizzone, J. A., e Bacchi, O. (1998). Principles of crop modeling and simulation: I. uses of mathematical models in agricultural science. *Scientia Agricola*, 55:46 – 50.

PythonDocs (2017). Python 3 documentation. `https://docs.python.org/3/glossary.html#term-global-interpreter-lock`. Acesso em: 10 Ago. 2017.