Aceleração da Classificação de Lavouras de Milho com MPI e Estratégias de Paralelismo

Anthony Vanzan[1], **Gabriel Rustick Fim**[1], **Greice Aline Welter**[1], **Dalvan Griebler**[1,2]

[1] Laboratório de Pesquisas Avançadas para Computação em Nuvem (LARCC), Faculdade Três de Maio (SETREM), Três de Maio, Brasil

[2]Escola Politécnica, Grupo de Modelagem de Aplicações Paralelas (GMAP), Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre, Brasil

{vanzan.anthony,gabifimtm,greicewelter8,dalvangriebler}@gmail.com

Resumo. *Este trabalho visou acelerar a execução de um algoritmo de classificação de lavouras em imagens áreas. Para isso, foram implementadas diferentes versões paralelas usando a biblioteca MPI na linguagem Python. A avaliação foi conduzida em dois ambientes computacionais. Conclui-se que é possível reduzir o tempo de execução a medida que mais recursos paralelos são usados e a estratégia de distribuição de trabalho dinâmica é mais eficiente.*

1. Introdução

A implementação de novas tecnologias no agronegócio, que aprimorem as técnicas já desenvolvidas ou até mesmo criem novas formas de realizar tarefas, otimizando o tempo e possibilitando aferir os dados com maior precisão, são de suma importância para o desenvolvimento da área. Cada dia mais os avanços tecnológicos permitem a implementação de tecnologias de inteligência artificial (IA) juntamente com imagens aéreas providas por veículos aéreos não tripulados (VANTs). A análise destas imagens a partir de IA requer alto poder computacional e a aplicação de técnicas de computação paralela para obter os resultados em tempo hábil.

Um programa para classificação de lavouras de milho em linguagem Python, chamado de classificador [Vanzan et al. 2020], foi desenvolvido no escopo do projeto AgroComputação[1], onde o objetivo é desenvolver soluções que auxiliam na medição da densidade da plantação através da análise de imagens capturadas a partir de VANTs e aplicando algoritmos de aprendizado profundo (*Deep Learning*). Neste artigo, o foco é implementar e avaliar diferentes versões paralelas do classificador, com o objetivo de acelerar o processo de classificação de áreas de lavouras, especificamente na cultura do milho. Além de contribuir com a aceleração deste processo, o artigo visa avaliar diferentes padrões paralelos, estratégias de distribuição de trabalho e ambientes computacionais.

Na literatura, a paralelização de programas escritos em Python tem sido realizada usando a biblioteca MPI, já bastante utilizada e consolidada nas linguagens C/C++ e Fortran. Nota-se que os estudos que avaliam o uso para processamento paralelo apresentaram desempenho muito similar entre as linguagens [Dalcin et al. 2011]. Mais especificamente, no contexto da paralelização de aplicações de aprendizado profundo, o foco tem sido a parte do treinamento [Ma et al. 2017]. No caso deste trabalho, o classificador é um programa que usa um modelo já treinado. Este artigo está estruturado da seguinte forma. Na seção 2 é descrito o classificador e como ele foi paralelizado. Na seção 3 são discutidos os resultados obtidos em dois ambientes computacionais.

[1]ACST - Projeto AgroComputação com apoio da SETREM e TECNICON: http://acst.setrem.com.br/

2. Paralelização do Classificador

A Figura 1(a) ilustra as etapas do programa classificador em sua versão sequencial. Ele começa realizando o recorte de uma ortofoto em imagem de 32x32 Pixeis.Esse é o tamanho da entrada suportado pela LeNet5 [Y. Lecun and Haffner 1998], sendo esta a arquitetura de rede neural utilizada pelo classificador. Essas imagens recortadas e as coordenadas de onde estas foram retiradas são anexadas a uma lista como uma tupla. Quando toda a ortofoto tiver sido recortada, o algoritmo realizará a classificação das imagens. Após ter transformado o recorte em *grayscale*, é chamado um modelo de rede neural LeNet5 pré-treinado para classificar em 'Milho' ou 'Não Milho', caso o recorte seja classificado em 'Não Milho' o mesmo é devolvido em *grayscale*, estas imagens são adicionadas novamente a uma lista para posterior remontagem da ortofoto uma vez que todas as imagens foram processadas, como é mostrado na Figura 1.

Para realizar a paralelização foram utilizados dois padrões paralelos, um utilizando a estratégia em lotes e outro em *stream*, usando a arquitetura `Master/Worker` e `Farm` respectivimente. Para cada um dos padrões, duas versões foram criadas, uma implementando a distribuição de trabalho `round-robin` e outro implementando a distribuição de trabalho `on-demand`. Essas opções são usadas recorrentemente na literatura e obtiveram bom desempenho em outros estudos [McCool et al. 2012, Andrade et al. 2014].

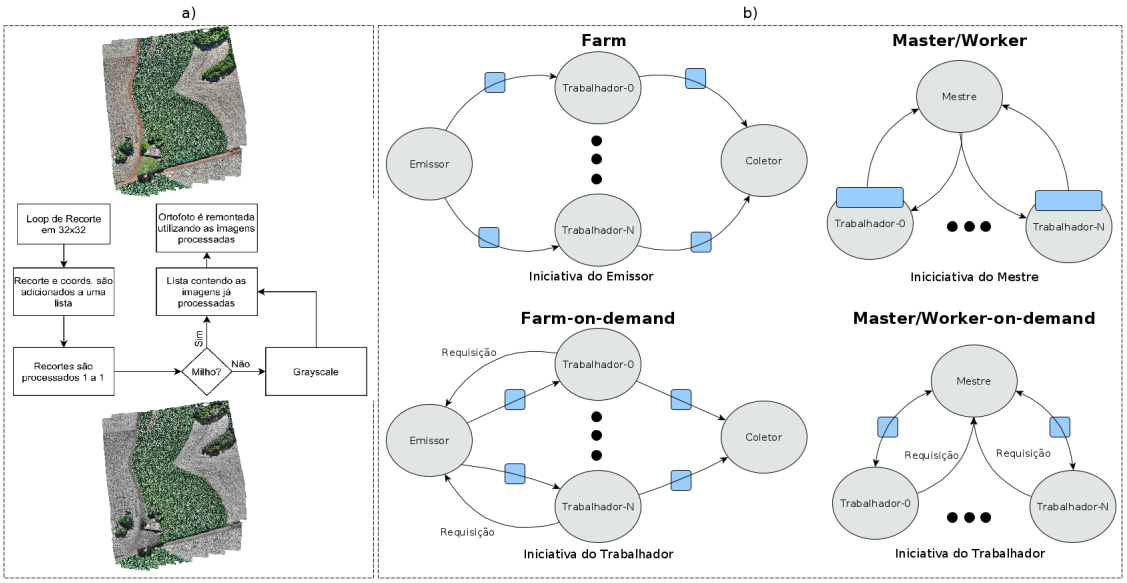

Figura 1. Representação gráfica das versões do classificador.

Na Figura 1(b), as versões `Master/Worker` utilizaram o mesmo tipo de fila implementado no algoritmo `Serial`. Na distribuição de trabalho `round-robin`, o Mestre realiza a divisão e envio dos recortes da ortofoto para os Trabalhadores realizarem o processamento e devolverem estes ao Mestre para montagem da ortofoto processada. Já no `Farm`, a lista não se faz necessária pois o Emissor realiza o recorte de 32x32 pixeis da ortofoto e o envia imediatamente a um Trabalhador, que uma vez concluído o processamento, a envia ao Coletor para montagem da ortofoto processada. Nas distribuições de trabalho `on-demand`, a iniciativa parte dos Trabalhadores. Estes requisitam ao Mestre ou Emissor um trabalho quando ociosos. Todas as versões foram paralelizadas com a biblioteca OpenMPI em Python3 para a troca de mensagens, mais especificamente através de comunicação bloqueante.

3. Resultados

Os experimentos foram executados na infraestrutura do LARCC[2] em dois ambientes computacionais. O primeiro ambiente, apelidado de `StormBreaker`, possui como sistema operacional o Ubuntu Server 20.04.1 LTS (Focal Fossa) (Kernel 5.4.0-62) e softwares Python 3.7.7, NumPy 1.19.2, OpenCV 3.4.2, PyTorch 1.7.1, mpi4py 3.0.3 e OpenMPI 4.0.3. Seu hardware é composto por um processador Intel i9-7900x (20 *threads*), 32GB de RAM, discos de armazenamento 4x 2TB Sata III 6.0GB/s em raid 5.

O segundo ambiente apelidado de `Nebula-Cloud` é construído de duas máquinas HP Proliant DL385 G6, onde cada uma delas possuí um processador AMD Opteron 2425 2100 MHz *Six-Core* (totalizando 24 threads), 32 GB de memória RAM DDR3 e 8 discos de 146 GB de armazenamento em RAID 5. Cada máquina possui 5 interfaces de rede Gigabit. A Nuvem configurada utiliza-se da ferramenta OpenNebula 5.12.0.3 com o virtualizador KVM 5.4.0-1031. Para o cenário de testes, foram criados 6 máquinas virtuais de mesma configuração (4 vCPUS, 6GB de memória RAM e 20GB de disco), as quais foram configuradas como um *cluster Beowulf* com NFS para compartilhamento de arquivos. Essa configuração do ambiente de nuvem visou evitar o uso de *hyper-threading*, uma vez que foi observado nenhum benefício ao executar no ambiente da `Stormbreaker` (Figura 2). As máquinas virtuais, onde as versões do classificador foram testadas, possuem as mesmas versões de software: Sistema operacional Ubuntu Server 20.04.1 LTS (Focal Fossa) (5.10.10-051010-generic) com Python 3.8.5, NumPy 1.19.5, OpenCV 4.2.0, PyTorch 1.7.1, mpi4py 3.0.3 e OpenMPI 4.0.3 instalados.

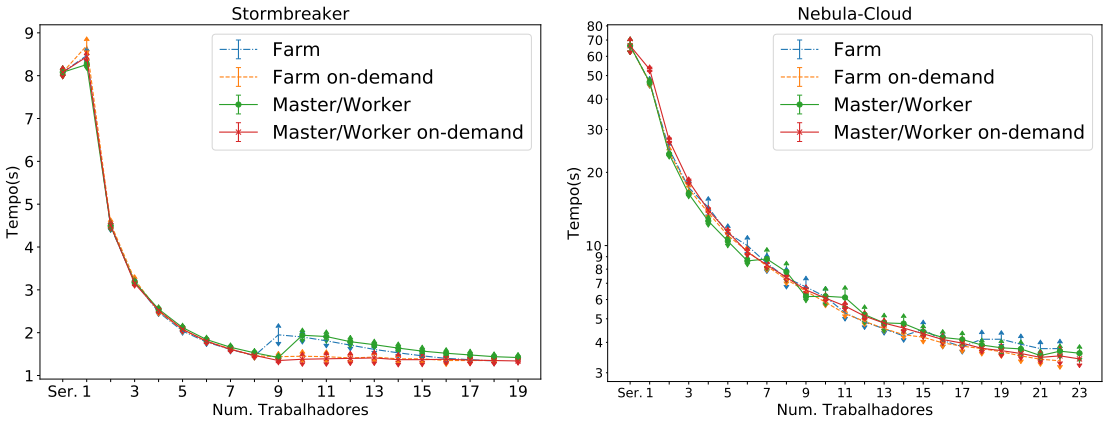

Figura 2. Resultados dos testes nos ambientes.

Para avaliação das versões paralelas, foi utilizada uma ortofoto de 3417x4095 pixeis, que consequentemente produz 13664 imagens de 32x32 pixeis a serem classificados e remontados pelo classificador. A ortofoto apresenta um exemplo real de uma imagem que seria classificada pelo algoritmo, englobando um ou mais milharais. Na Figura 2 são apresentados os gráficos dos testes, repetindo intercaladamente 30 vezes cada configuração de trabalhadores. Os gráficos apresentam o tempo de execução em segundos (eixo y) em relação a quantidade de trabalhadores (eixo x). Cada linha do gráfico é uma versão paralela diferente do classificador. Junto dos pontos nas linhas, são plotados o desvio padrão das 30 execuções, onde para a maioria das configurações é quase imperceptível. É possível observar que ambos os padrões implementados usando a estratégia de distribuição de trabalho `on-demand` se sobressaíram. Com isso, é possível concluir que a distribuição

[2]Laboratório de Pesquisas Avançadas para Computação em Nuvem: http://larcc.setrem.com.br/

de carga tem impacto no desempenho. Nota-se também que na `Stormbreaker`, que possui um processador mais novo, o tempo de execução da versão sequencial é muito menor se comparado ao *cluster* montado na nuvem privada `Nebula-Cloud`, conforme era esperado. Ainda na `Stormbreaker`, o tempo de execução das versões paralelas não diminui mais quando entra na zona de uso do *hyper-threading* para as versões `on-demand`. Ao invés disso, nota-se que versões `round-robin` apresentaram um aumento no tempo de execução com 9 trabalhadores no padrão `Farm` e 10 trabalhadores no padrão `Master/Worker`. Isso é porque o padrão `Farm` possui também dois processos concorrentes (Emissor e Coletor) enquanto que o `Master/Worker` possui apenas um (o Mestre). Além disso, é visível que esta concorrência por recursos gera um desbalanceamento de carga entre os trabalhadores. Embora a estratégia `on-demand` demande um número maior de mensagens, ela acaba sendo mais eficaz neste aspecto.

No ambiente `Nebula-Cloud`, plotado em escala logarítmica, observa-se que a versão `Master/Worker` obteve uma leve superioridade nos resultado até 6 Trabalhadores e após isso, oscilou bastante e tornou-se menos eficiente que as demais versões. Nota-se também uma oscilação para a versão `Farm`, atingindo seus melhores resultado entre 11 e 14 trabalhadores. Concluí-se que este comportamento em ambas as versões é devido ao impacto do balanceamento de carga, uma vez nas versões `on-demand`, o comportamento se mostrou estável. `Farm on-demand` foi a versão que melhor se saiu a partir de 10 Trabalhadores. Por fim, nota-se que o tempo de execução foi diminuindo quase que logaritmicamente, pois não foi usada propositalmente a tecnologia *hyper-threading*.

4. Conclusão

Concluiu-se que é possível reduzir o tempo de execução a medida que mais recursos paralelos são usados e que a estratégia de distribuição de trabalho `on-demand` seria a mais eficiente para o objetivo que os algoritmos buscam alcançar. Como estudo futuro, planeja-se realizar testes comparativos com outros padrões paralelos e a aplicação de métricas como *SpeedUp* dentro dos algoritmos desenvolvidos, visando diminuir ainda mais o tempo necessário para o processamento e classificação das áreas em uma imagem.

Agradecimentos Os autores agradecem ao Laboratório de Pesquisa Avançada em Computação em Nuvem (LARCC/SETREM, Brasil) por fornecer recursos de computação.

Referências

Andrade, H. C., Gedik, B., and Turaga, D. S. (2014). *Fundamentals of stream processing: application design, systems, and analytics*. Cambridge University Press.

Dalcin, L. D., Paz, R. R., Kler, P. A., and Cosimo, A. (2011). Parallel distributed computing using Python. *Advances in Water Resources*, 34(9):1124 – 1139.

Ma, H., Mao, F., and Taylor, G. W. (2017). Theano-MPI: A Theano-Based Distributed Training Framework. In *Euro-Par 2016: Parallel Processing Workshops*, pages 800–813, Cham. Springer.

McCool, M., Reinders, J., and Robison, A. (2012). *Structured parallel programming: patterns for efficient computation*. Elsevier.

Vanzan, A., Fim, G. R., Welter, G. A., Sausen, M. C., and Griebler, D. (2020). Algoritmo de Deep Learning para Classificação de Áreas de Lavaoura com VANTs. In *22 Salão de Pesquisa Setrem (SAPS)*, page 5, Três de Maio, RS, Brazil. Sociedade Educacional Três de Maio.

Y. Lecun, L. Bottou, Y. B. and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324*.