Construção de Árvores Aleatórias em Paralelo usando Conjuntos Textuais

Julio C. B. Pires[1], **Wellington S. Martins**[2]

[1]Instituto Federal de Educação, Ciência e Tecnologia Goiano (IF Goiano)
Campus Avançado Hidrolândia – 75.340-000 – Hidrolândia – GO – Brazil

[2]Instituto de Informática – Universidade Federal de Goiás (UFG)
Campus Samambaia – 74.690-900 – Goiânia – GO – Brazil

julio.pires@ifgoiano.edu.br, wellington@inf.ufg.br

Abstract. *The large amount of information available arouses interest in learning important data patterns, which are mostly in text format and are unstructured. Thus, new challenges arise to deal with this type of data, such as the computational load associated with the high dimensionality and noise associated with the nature of the data. Thus, in the present work the parallelization of a learning algorithm is addressed, a combination of bagging of random trees with boosting for automatic text classification. The goal is to speed up the construction of the trees on the GPU.*

Resumo. *A grande quantidade de informação disponível desperta o interesse para o aprendizado de padrões importantes dos dados, que em sua maioria estão em formato de texto e são desestruturados. Desse modo, surgem novos desafios para lidar com esse tipo de dado, como a carga computacional ligada à alta dimensionalidade e ruídos atrelados à natureza dos dados. Assim, no presente trabalho é abordada a paralelização de um algoritmo de aprendizado, uma combinação de bagging de árvores aleatórias com boosting para a classificação automática de texto. O objetivo é acelerar a construção das árvores na GPU.*

1. Introdução

A quantidade de informação disponível atualmente desperta o interesse no aprendizado de padrões úteis e importantes dos dados. Se de um lado existe uma massa de dados para serem usados, de outro, surgem algumas dificuldades, como custo computacional e robustez para tratar dados ruidosos. O uso de processadores massivamente paralelos como Unidades de Processamento Gráfico (*GPUs*) pode diminuir o esforço do programa. Já a combinação de *Bagging* de Árvores Extremamente Aleatórias com *Boosting* [Campos et al. 2017], pode ser uma solução poderosa em face a ruídos nos dados textuais. No presente texto é apresentada uma paralelização desse algoritmo, onde é feita a aceleração do processo de construção das árvores usando algumas estratégias paralelas.

Basicamente, o algoritmo cria várias florestas de forma iterativa. No início são atribuídos pesos para cada documento de exemplo. Cada nova iteração altera o valor desses pesos, que refletem se o documento foi corretamente classificado. Tais valores servem para fazer novas amostragens, ou seja, exemplos mais difíceis de classificar têm

mais chance de serem escolhidos. Depois da amostragem (*bagging*) as árvores são construídas na placa gráfica. Esse método recente demonstrou bons resultados em comparação à outras classificadores de texto e à sua versão sequencial.

2. Árvores Aleatórias

A Classificação — uma das subáreas do Aprendizado de Máquina — constrói modelos a partir de conjuntos formados de muitos exemplos [Han et al. 2012]. Cada exemplo é caracterizado por atributos e rótulo de classe, assim, conjuntos textuais são compostos de documentos descritos pela frequência de termos. Geralmente os conjuntos que usam essa representação possuem grande dimensionalidade e esparsidade. Modelos de classificação podem ser representados na forma de regras ou Árvores de Decisão, formadas por nós internos (testes) que separam documentos de acordo com o valor dos termos. As folhas armazenam a categoria do documento [Tan et al. 2013], como esporte, viagem, etc.

Uma combinação popular de árvores é o Florestas Aleatórias [Breiman 2001], que cria árvores através da replicação aleatória do conjunto de dados original (*Bagging*) e seleção de atributos para divisão dos nós. Já as Árvores Extremamente Aleatórias [Geurts et al. 2006] usam o conjunto de dados original e fazem cortes aleatórios nos valores dos atributos. Em [Campos et al. 2017] é apresentado outro agrupamento de árvores, que faz amostragem com *bagging* [Breiman 1996] e cria modelos simples de forma iterativa usando *boosting* [Freund and Schapire 1997] e penaliza exemplos mal classificados a cada rodada. De acordo com os autores a estratégia está entre os melhores classificadores testados. O uso de aleatoriedade torna o algoritmo mais robusto à presença dos ruídos nos dados.

3. O Algoritmo

É altamente desejável construir algoritmos computacionalmente eficientes para lidar com grandes quantidades de dados. Nessa perspectiva está o uso do alto poder computacional oferecido pelas placas gráficas. As implementações de árvores de decisão paralelas em *GPU* encontradas na literatura se diferenciam pelas estratégias de partição do problema e formas de divisão dos dados. Em muitos trabalhos o foco está na construção dos classificadores e a grande maioria usa o modelo de programação *CUDA*. Por exemplo, em [Jansson et al. 2014] cada nível das árvores é criado em paralelo, em que cada bloco equivale a um nó e cada *thread* a um atributo ou exemplo. Esse modelo é a base de inspiração do presente trabalho.

Tudo começa após a alocação dos recursos necessários, o primeiro passo é inicializar os pesos dos documentos. Depois disso, começa um processo iterativo que faz nova amostragem dos dados e constrói as árvores. A construção da floresta é feita em largura, com um bloco por nó e uma *thread* por termo, onde é lançado um *kernel* por árvore para a construção de todos os nós do nível atual. Depois de escolher os termos, as *threads* deslizam nos documentos para geração do ponto de corte aleatório. Após a construção da floresta é feita uma predição para cálculo do erro do classificador, os pesos são atualizados e normalizados. No final de cada iteração a floresta é passada para a memória principal. Como exemplo simples, o código *CUDA* da Figura 1 faz a lançamento dos *kernels* para a construção de 3 níveis de uma única árvore. Nas linhas 7-8 são criados os vetores na memória principal (*CPU*) e na memória global (*GPU*) que representam uma

árvore binária perfeita de altura 2. Depois disso, na linha 9 um laço de repetição lança o *kernel* da linha 10 para construir os nós de cada nível. Na linha 11 os dados são devolvidos para a memória principal (*CPU*) e o programa termina após a liberação de memória na linha 12. Na Figura 2 é representado o processamento do segundo nível da árvore.

```
1  #include <stdio.h>
2  #include <math.h>
3  __global__ void kernel( int* arv ) { arv[ ... ]; }
4
5  int main( void ) {
6    int tamanho = ( pow( 2, 2 + 1 ) - 1 ) * sizeof( int );
7    int* h_arv = ( int* ) malloc( tamanho );
8    int* d_arv; cudaMalloc( ( void** ) &d_arv, tamanho );
9    for( int i = 0, nos = 1; i <= altura; i++, nos *= 2 )
10       kernel<<< nos, 128 >>>( d_arv );
11   cudaMemcpy( d_arv, h_arv, tamanho, cudaMemcpyDeviceToHost );
12   cudaFree( d_arv ); free( h_arv );
13 }
```

Figura 1. Construção de uma árvore.

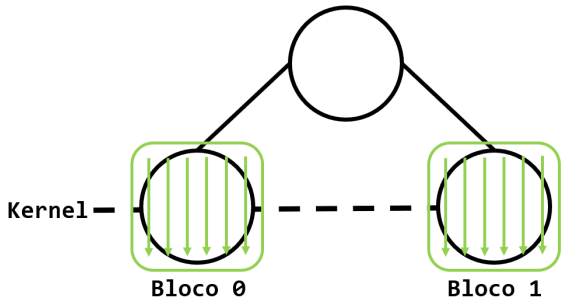

Figura 2. Construção do segundo nível da árvore.

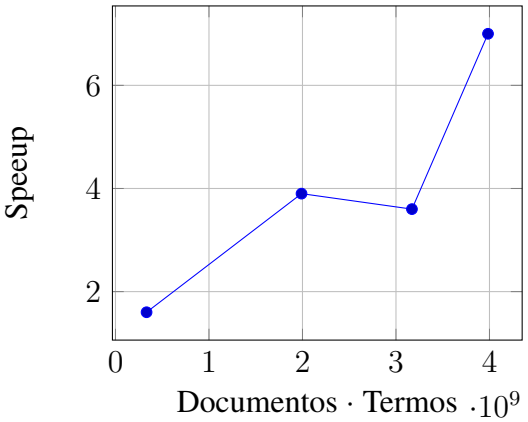

Figura 3. Speedup e dimensões.

4. Resultados Preliminares

Os experimentos foram realizados em um ambiente *Linux Ubuntu 18.04 LTS*, *driver CUDA* versão 10, processador *Intel i7-7700* de 4 núcleos com *4,20 GHz* e uma placa de vídeo *NVIDIA GeForce 1070* de 1920 núcleos com *1771 MHz* para a construção de 8 árvores profundas. Os conjuntos textuais utilizados são compostos por documentos descritos por termos (*TF-IDF*) e possuem dimensionalidades diversas. A performance dos algoritmos foi avaliada de acordo com execuções sequenciais e paralelas. O algoritmo sequencial foi executado na *CPU* usando apenas um núcleo. Os algoritmos rodaram 5 vezes com cada conjunto de dados para obtenção de uma média aritmética. Os resultados usando 4 conjuntos são apresentados no gráfico da Figura 3 com os seguintes *speedups*: 4UNI (1,6x), ACM (3,9x), 20NG (3,6x) e REUTERS90 (7x).

Em comparação com o trabalho anterior [Pires and Martins 2019], que utilizou árvores rasas, a proposta descrita neste trabalho constrói árvores profundas; totalmente

desenvolvidas, até o tamanho máximo. Quando se usa árvores rasas pode-se usar uma representação de árvore binária perfeita — a cada nível o número de nós dobra, o que é fácil de controlar. Além disso, uma árvore de altura (h) 4, usada no trabalho anterior [Pires and Martins 2019], ocupa um vetor de tamanho pequeno, 31 ($2^{h+1} - 1$). Para árvores profundas essa representação demandaria muita memória, e desperdiçaria grande parte dela visto que as árvores não são completas. Por isso neste trabalho usamos uma representação com um vetor de bits, atribuindo 1 aos nós internos e 0 às folhas. Mesmo com esta representação, a quantidade de memória é considerável e o gerenciamento dos dados, feito usando operações bit a bit, torna-se complexo.

5. Considerações

A tarefa da classificação de textos lida com conjuntos de dados ruidosos e de alta dimensionalidade, o que torna a construção de classificadores uma tarefa custosa. Neste sentido, foi apresentado neste trabalho a primeira implementação paralela que se tem conhecimento sobre a construção de árvores extremamente aleatórias com altura máxima no contexto do *boosting*. Dependendo da aplicação e do tamanho dos dados, usar algoritmos paralelos pode significar a redução do tempo de execução de dias para horas. Mesmo assim, ainda é necessário analisar novas formas de paralelismo, explorar o uso eficiente da memória da placa gráfica e realizar mais testes.

Referências

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

Campos, R., Canuto, S., Salles, T., de Sá, C. C., and Gonçalves, M. A. (2017). Stacking bagged and boosted forests for effective automated classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 105–114, New York, NY, USA. ACM.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42.

Han, J., Kamber, M., and Pei, J. (2012). Data mining concepts and techniques, third edition.

Jansson, K., Sundell, H., and Boström, H. (2014). gpurf and gpuert: Efficient and scalable gpu algorithms for decision tree ensembles. In *2014 IEEE International Parallel Distributed Processing Symposium Workshops*, pages 1612–1621.

Pires, J. and Martins, W. (2019). Impulsionando Árvores extremamente aleatórias em paralelo para a classificação de dados textuais. In *Anais Principais do XX Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 312–323, Porto Alegre, RS, Brasil. SBC.

Tan, P.-N., Steinbach, M., and Kumar, V. (2013). *Introduction to Data Mining: Pearson New International Edition (English Edition)*. Pearson Education Limited, Harlow, ESX, UK.