Impulsionando Árvores Extremamente Aleatórias Ensacadas em Paralelo para Classificação de Textos

Julio C. B. Pires[1], Wellington S. Martins[2], Daniel X. de Sousa[3]

[1]Instituto Federal Goiano (IF Goiano) – Campus Urutaí
Urutaí – GO – Brazil

[2]Instituto de Informática (INF) – Universidade Federal de Goiás (UFG)
Goiânia – GO – Brazil

[3]Instituto Federal de Goiás (IFG) – Campus Anápolis
Anápolis – GO – Brazil

julio.pires@ifgoiano.edu.br, wellington@inf.ufg.br,
daniel.sousa@ifg.edu.br

Abstract. *The present work proposes the parallelization of BERT, an algorithm that combines boosting with bagging of extremely randomized trees to make automatic classification of textual datasets. Using high dimensionality sets can make the construction of classifiers an onerous task. Parallelism combined with graphics cards can overcome this challenge, since they offer a high processing power, the execution time can decrease considerably.*

Resumo. *O presente trabalho tem como proposta a paralelização do BERT, um algoritmo que combina boosting com bagging de árvores extremamente aleatórias para fazer classificação automática de conjuntos de dados textuais. Usar conjuntos de alta dimensionalidade pode tornar a construção dos classificadores uma tarefa onerosa. O paralelismo aliado às placas gráficas pode contornar esse desafio, uma vez que elas oferecem um alto poder de processamento, o tempo de execução pode diminuir consideravelmente.*

1. Introdução

Com a grande quantidade de informação disponível na Web e o seu rápido crescimento, surge a necessidade da organização e extração de padrões úteis dessa massa de dados [Campos et al. 2017]. Neste sentido, estão as estratégias do aprendizado de máquina, como a classificação, que constrói modelos através de vários exemplos para predizer rótulos de classe para novos exemplos. Construir classificadores para execução sequencial na era do *Big Data* pode se tornar inviável, uma vez que a quantidade massiva de dados precisa de um grande esforço de processamento. Usar programação de alto desempenho em unidades de processamento gráfico (*GPUs*) pode acelerar o processo de aprendizado, ou seja, a computação dos classificadores se torna menos onerosa.

A maior parte da informação da web está em linguagem natural, classificar estes textos é uma forma de compreender melhor esses dados. Assim, o presente trabalho aborda a classificação automática de textos, que na sua maioria, são conjuntos de dados massivos e ruidosos. Um dos algoritmos de maior sucesso para resolver problemas de classificação é o Florestas Aleatórias (FAs), porém sua performance não é a mesma em

face à ruídos nos dados [Campos et al. 2017]. Para resolver esse problema pode ser usado o algoritmo Árvores Extremamente Aleatórias (AEAs), uma solução baseada no FAs, que adiciona um pouco mais de aleatoriedade na parte da construção das árvores de decisão. Levando tudo isso em consideração, a proposta está em paralelizar o algoritmo **BERT** (*Boosted Extremely Randomized Trees*) [Campos et al. 2017] para melhoria na velocidade de execução.

O algoritmo **BERT** usa *boosting* [Mitchell e Frank 2017], um procedimento que cria vários classificadores de forma iterativa (Figura 1). São atribuídos pesos para as amostras do conjunto de dados, e a cada iteração (treino, modelo e teste) esses pesos são modificados. Assim, as amostras mais difíceis de classificar (erros) recebem um peso maior e consequentemente são o foco da iteração subsequente. Cada iteração constrói um grupo de AEAs (modelo). Além disso, as árvores de cada grupo são construídas usando outro procedimento, o *bagging*, onde acontece uma amostragem com substituição do conjunto de treinamento original (Figura 3). Cada classificador é representado por uma árvore binária, que é construída de cima para baixo através da divisão dos dados de treinamento. Outra estratégia do **BERT** está em atualizar apenas os pesos das amostras que não são usadas na construção (*out-of-bag*) [Salles et al. 2015] da árvore. Ao termino das iterações, todos os grupos de cada iteração formam o classificador final (predição).

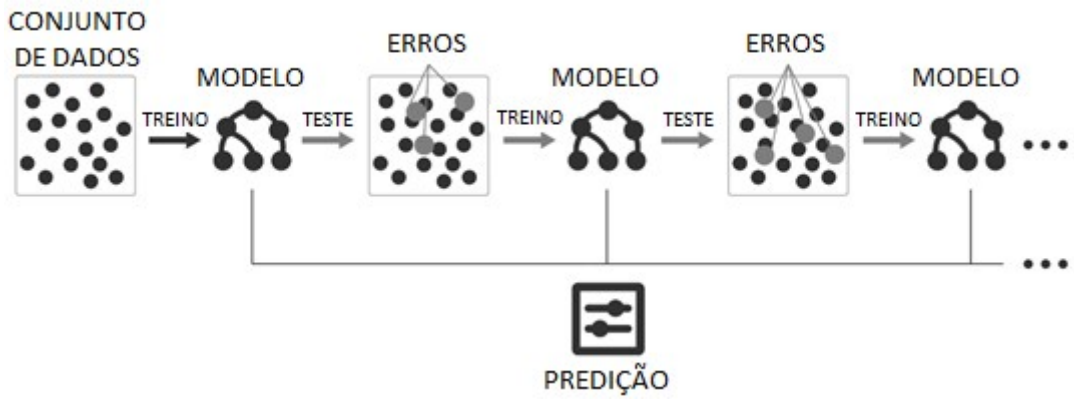

Figura 1. Esquema de *boosting*. Adaptado de [Alonso 2017].

2. Proposta

O volume massivo de dados usado no treinamento demanda computação de alto desempenho para deixar o processamento mais rápido. Placas gráficas são uma alternativa barata em comparação à clusters ou servidores para ganho de desempenho, mas não se deve ignorar a alta complexidade de codificação e projeto cuidadoso para maximizar a eficiência e ocupação de memória. Assim, maximizar a eficiência de execução na *GPU* é um tanto quanto difícil e requer um projeto refinado, onde o programador precisa conhecer a arquitetura de comunicação das memórias e os fluxos de execução das tarefas de uma *GPU*. Esses hardwares possuem memórias organizadas de forma hierárquica e oferecem paralelismo na forma da arquitetura *SIMD*, onde uma única instrução é executada sobre muitos dados diferentes. Essas placas são otimizadas para o uso do paralelismo massivo em uma grande quantidade de dados (*throughput*) para esconder a latência. Existem duas plataformas principais de programação para *GPUs*, a *OpenCL* (*Khronos*) e a *CUDA* (*NVIDIA*). A última é mais utilizada e já fornece várias bibliotecas otimizadas e por isso é a escolhida para este projeto. Os programas executados nessas placas são chamados de *kernel*. Eles são processados em grades de blocos de *threads*

(Figura 2), ou seja, *threads* são agrupadas em blocos e blocos são agrupados em grades. *Threads* são executadas em grupos de 32 (*warp*) [Jansson et al. 2014].

Existem algumas soluções paralelas que usam árvores de decisão, FAs e AEAs [Cano 2018], mas ainda não se conhece uma paralelização do algoritmo ***BERT***. Várias estratégias foram usadas para construir algoritmos baseados em árvores na GPU, como a construção do FAs em nível de árvore [Grahn et al. 2011], implementação do FAs e o AEAs em nível de nó [Jansson et al. 2014] e construção de nível por nível [Marron et al. 2014]. A proposta deste trabalho consiste em usar soluções paralelas eficientes para resolver problemas de classificação de textos. Essa solução melhorada poderá ser usada em problemas de classificação em geral, análise de sentimentos e categorização de tópicos, por exemplo. Fazer um projeto cuidadoso para maximizar o desempenho do algoritmo pode reduzir o tempo e escalar a execução do algoritmo.

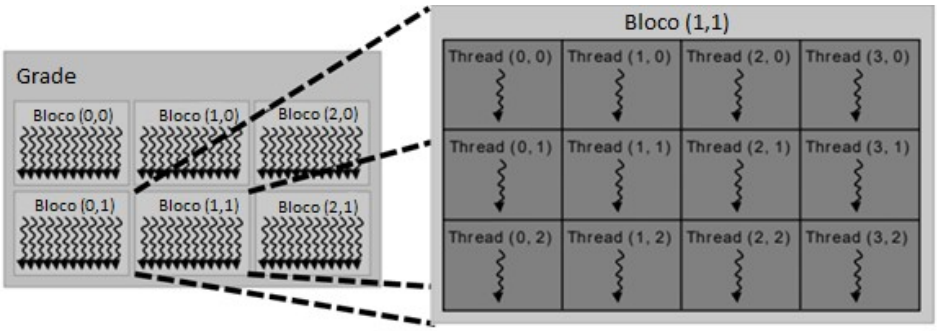

Figura 2. Uma Grade de Blocos de *Threads.* Adaptado de [NVIDIA 2018].

3. Metodologia

A metodologia deste trabalho comporta as etapas de fundamentação teórica, construção do algoritmo sequencial, projeto e implementação do algoritmo paralelo, análise dos resultados, otimização do algoritmo e documentação. Para o referencial teórico foi feita uma pesquisa sobre classificação, árvores de decisão, e os algoritmos FAs e AEAs. Além disso, foi feito um estudo dos procedimentos de *bagging* e de *boosting* (Figura 3). Além de utilizar alguns livros base, a maioria dos artigos foram encontrados em diversos repositórios científicos. Depois do estudo dos trabalhos relacionados ao tema da pesquisa foi feita a construção do algoritmo sequencial.

A próxima etapa consiste da implementação completa da versão paralela. Essa versão será executada em uma *GPU* para comparação com a versão sequencial. Para isso será usada a métrica de *speedup*, que é a razão do tempo do algoritmo sequencial pelo tempo do algoritmo paralelo [Navarro et al. 2014]. Depois da execução dos algoritmos sequencial e paralelo, será feita a colheita dos resultados para avaliação de desempenho. Depois o código será analisado em busca de erros e maiores oportunidades de ganho de tempo. Por fim será a documentação do trabalho na forma de um artigo.

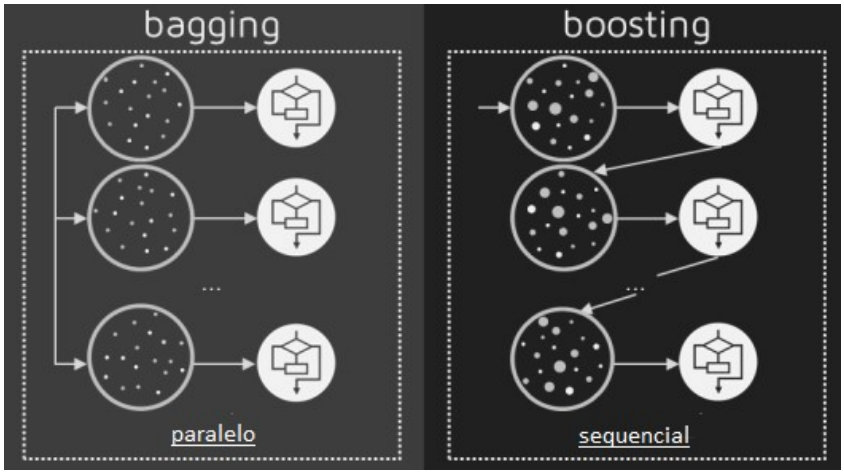

Figura 3. Diferença entre *bagging* e *boosting*. Adaptado de [Garrido 2016].

4. Resultados Preliminares

Após o levantamento bibliográfico, análise de códigos sequencias, estudo de estratégias de códigos paralelos, entendimento de algoritmos descritos em vários artigos e implementação própria do algoritmo sequencial, a próxima etapa foi a implementação de partes do algoritmo paralelo. Várias versões paralelas foram projetadas e implementadas, algumas não alcançaram os resultados desejados e outras apresentaram problemas de instabilidade. Atualmente, o foco está no afinamento e complemento do código paralelo, uma vez que a parte do *bagging* já está pronta. Os experimentos foram conduzidos na preparação de 8 amostragens em 200 iterações. Os resultados iniciais com conjuntos relativamente medianos (4uni e acm) mostraram um bom *speedup* (Tabela 1) e validam uma parte da proposta. A próxima etapa consiste em criar de fato todas as árvores.

Tabela 1. Comparação entre as execuções sequencial e paralela

Conjunto de Dados	Atributos	Amostras	Classes	Tempo Sequencial	Tempo Paralelo	Speedup
4uni	4.196	6.624	7	86,2368 s	1,3986 s	61,8 x
acm	59.991	26.546	10	1384,2556 s	20,3990 s	67,9 x

5. Considerações

A classificação de textos é utilizada para organizar informações e tem algumas aplicações, como análise de sentimentos e categorização de tópicos. Essa tarefa lida com conjuntos de dados ruidosos e de alta dimensionalidade, o que pode diminuir a eficácia de um classificador e tornar o custo computacional oneroso. A utilização de conjuntos de treino grandes demanda execução de forma rápida, o que é indispensável na era do *Big Data*. Desse modo, neste trabalho foi proposto a paralelização do algoritmo **BERT** utilizando placas gráficas visando ganhos de velocidade. Por enquanto, fazendo apenas o *bagging* das árvores com conjuntos medianos a abordagem melhora consideravelmente o tempo de execução. Isso é importante, uma vez que os dados usados para classificação de texto podem ser grandes. Além disso, mais e mais informação nova fica disponível a cada dia na imensidão da Web. Crucial é fazer uso dessa informação para construir soluções

inteligentes que executem em tempo hábil. O próximo passo pode estar na implementação paralela do **BERT** usando o esquema das Florestas Aleatórias Profundas [Zhou and Feng 2017]. Cada floresta é organizada em cascata e as entradas da próxima floresta dependem da floresta anterior.

Referências

Alonso, M. J. (2017) "Introduction to Boosted Trees", https://blog.bigml.com/2017/03/14/introduction-to-boosted-trees, September.

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.

Cano, A. (2018). A survey on graphic processing unit computing for large-scale data mining. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 8(1).

Garrido, A. P. (2016) "What is the difference between Bagging and Boosting?", https://quantdare.com/what-is-the-difference-between-bagging-and-boosting, September.

Grahn, H., Lavesson, N., Lapajne, M. H., Slat, D. (2011). CudaRF: A CUDA-based implementation of random forests. In *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications*, AICCSA. 95-101.

Jansson, K., Sundell, H., and Boström, H. (2014). gpurf and gpuert: Efficient and scalable gpu algorithms for decision tree ensembles. In *Proceedings of the 2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, IPDPSW '14, pages 1612–1621, Washington, DC, USA. IEEE Computer Society.

Marron, D., Bifet, A., and Morales, G. D. F. (2014). Random forests of very fast decision trees on gpu for mining evolving big data streams. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, ECAI'14, pages 615–620, Amsterdam,The Netherlands, The Netherlands. IOS Press.

Mitchell, R. and Frank, E. (2017). Accelerating the xgboost algorithm using gpu computing. *PeerJ Computer Science*, 3:e127.

Salles, T., Gonçalves, M., Rodrigues, V., and Rocha, L. (2015). Broof: Exploiting out-of-bag errors, boosting and random forests for effective automated classification. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 353–362, New York, NY, USA.ACM.

Navarro, C., Hitschfeld-Kahler, N., and Mateu, L. (2014). A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures. Communications in Computational Physics, 15(2), 285-329.

NVIDIA. (2018) "Cuda C Programming Guide", https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, September.

Zhou, Z.-H. and Feng, J. (2017). Deep forest: Towards an alternative to deep neural networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, IJCAI-17, pages 3553–3559.