Algoritmo RSFK para busca de similaridade em GPU

Bruno Henrique Meyer[1], **Wagner M. Nunan Zola**[1], **Aurora Trinidad Ramirez Pozo**[1]

[1]Departamento de Informática
Universidade Federal do Paraná (UFPR) – Curitiba , PR – Brazil

{bhmeyer,wagner,aurora}@inf.ufpr.br

Resumo. *Este trabalho apresenta um algoritmo chamado RSFK para computar a estrutura KNN Graph utilizando GPU. Foi possível observar que o RSFK obteve melhores resultados em comparação às outras opções observadas na literatura, considerando o tradeoff entre tempo e acurácia do resultado.*

1. Introdução e Trabalhos Relacionados

A estrutura que representa os K vizinhos mais próximos para cada elemento de um conjunto de pontos é definida como K-*Nearest Neighbors Graph* (KNN *Graph*), e é utilizada em diversos cenários, incluindo algoritmos de aprendizado de máquina [Tang et al. 2016]. Devido à natureza da complexidade quadrática do tempo necessário para computar o KNN *Graph*, diversos algoritmos foram propostos para aproximar a busca dos vizinhos [Yan et al. 2019, Johnson et al. 2019]. No algoritmo *Random Projection Forest* KNN (RPF-KNN) [Yan et al. 2019], as partições são criadas por meio da projeção dos pontos em um hiperplano aleatório. Após a projeção, os pontos estarão ordenados e poderão ser separados em duas partições de tamanho equivalente. A partir delas o mesmo processo poderá ser realizado recursivamente, o que pode ser representado por uma árvore binária. Neste trabalho, apresentamos uma variante do RPF-KNN chamada *Random Sample Forest* KNN, que iremos abreviar como RSFK. No RSFK, a mesma estratégia do RPF-KNN é utilizada, com a diferença de que a criação de um hiperplano aleatório e a projeção dos pontos não são etapas necessárias. Na nossa abordagem, são gerados hiperplanos equidistantes a dois pontos da partição, que são selecionados aleatoriamente. As primitivas *cuRAND* da linguagem CUDA são utilizadas para gerar números pseudoaleatórios. Dessa forma, o lado em que cada ponto ficará em relação ao hiperplano será usado como critério para definir quais serão as novas partições.

Na biblioteca ANNOY, que implementa o RPF-KNN, a estratégia para paralelizar o algoritmo consiste em criar uma *thread* da CPU para computar cada árvore, o que é adequado apenas quando poucos núcleos de processamento serão utilizados, o que não é o caso de arquiteturas de GPU. A nossa abordagem permite realizar a paralelização criando uma árvore por vez, porém de forma paralelizada inteiramente em GPU ao criar uma *thread* da GPU para cada ponto. Para implementar esta abordagem, é necessário utilizar operações atômicas para garantir a sincronização de todas as *threads*. Para implementar o RSFK, também possibilitamos o uso da técnica *Nearest Neighbor Exploring* (NN *Exploring*) [Tang et al. 2016], uma abordagem para o pós-processamento dos dados para melhorar a acurácia da aproximação.

2. Materiais e Métodos

Testamos nossa implementação do RSFK em GPU com processador i5-4460 3.20GHz (4 núcleos) com GPU NVIDIA RTX 2070 e CUDA 10.1. Nos experimentos foram utilizadas as bases de dados MNIST e ImageNet. O KNN *Graph* de cada base foi computado

usando as bibliotecas FAISS [Johnson et al. 2019], e ANNOY, e a nossa implementação RSFK. Cada técnica foi executada diversas vezes alterando os parâmetros que controlam o *tradeoff* entre tempo e acurácia O RSFK teve 2 execuções adicionais para avaliar a estratégia NN *Exploring*. A acurácia do RSFK e ANNOY é controlada pelo número de árvores geradas, que são combinadas (*ensemble*) para obter um KNN *Graph* mais preciso. Em nossos experimentos, variamos o número de árvores entre 1 e 896 para o RSFK, e entre 1 e 21 para o ANNOY. Cada árvore pode ser considerada uma execução do algoritmo (cada árvore usa uma semente diferente para os geradores de números pseudoaleatórios). Na biblioteca FAISS, podemos controlar a acurácia do método por meio do parâmetro *nprobe*, que foi variado entre 1 e 20. Para cada execução, consideramos o número de vizinhos (K) igual a 32. Para cada técnica, medimos o número médio de pontos processados por segundo e a acurácia da aproximação do KNN-Graph em comparação com o seu resultado exato (porcentagem de vizinhos corretos).

3. Resultados e Conclusões

Na Figura 1 identificamos que o RSFK obteve o melhor *tradeoff* dentre as técnicas comparadas. Verificamos que o uso da estatégia NN *Exploring* apresentou uma melhora apenas quando é necessário criar o KNN *Graph* com acurácias próximas de 100%. Concluímos que as evidências apresentadas neste trabalho indicam que o algoritmo RSFK pode trazer vantagens em acurácia e/ou tempo em relação à biblioteca ANNOY e ao método IVFLAT da biblioteca FAISS. Em trabalhos futuros pretendemos aplicar o algoritmo no contexto de aplicações em Aprendizado de Máquina, por exemplo na redução de dimensionalidade [Meyer et al. 2020] para visualização e interpretação de grandes *datasets*.

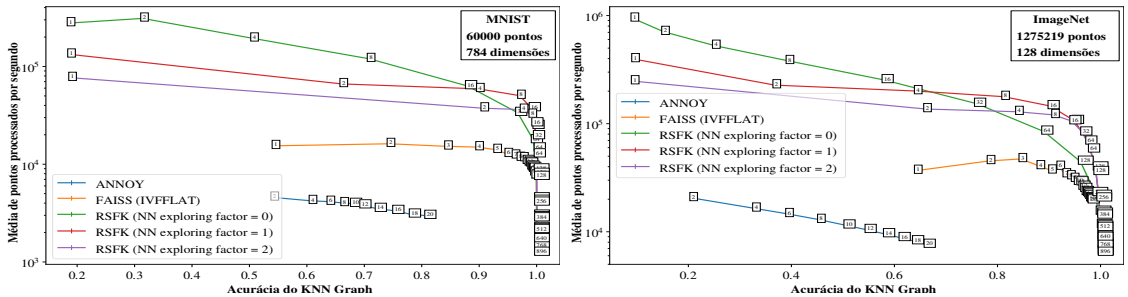

Figura 1. Comparação de algoritmos . Cada ponto representa a execução de um algoritmo com uma variação de parâmetro, *nprobe* para o FAISS (IVFFLAT) e número de árvores para ANNOY e RSFK.

Referências

Johnson, J., Douze, M., and Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*.

Meyer, B. H., Pozo, A. T. R., and Zola, W. M. N. (2020). Improving Barnes-Hut t-SNE scalability in GPU with efficient memory access strategies. In *2020 International Joint Conference on Neural Networks (IJCNN)*.

Tang, J., Liu, J., Zhang, M., and Mei, Q. (2016). Visualizing large-scale and high-dimensional data. *25th International World Wide Web Conference, WWW 2016*.

Yan, D., Wang, Y., Wang, J., Wang, H., and Li, Z. (2019). K-nearest Neighbors Search by Random Projection Forests. *IEEE Transactions on Big Data*, 7790.