ACELERAÇÃO DA CONSTRUÇÃO DE MATRIZES DE COOCORRÊNCIA PALAVRA-PALAVRA EM TEXTOS USANDO GPU

Misael Mateus Oliveira de Moraes[1], **Wellington Santos Martins**[2]

Instituto de Informática
Universidade Federal de Goiás
email: misaelmateus12@gmail.com[1], wellington@inf.ufg.br[2]

Resumo: Neste trabalho implementamos algoritmos sequenciais e paralelos para o problema de construção de matrizes de coocorrências na GPU

Palavras-chave: Texto, Processamento de Linguagem Natura, Word Embeddings, Paralelismo, GPU.

1 INTRODUÇÃO

A cada ano a quantidade da dados produzido no mundo digital cresce significantemente. Neste contexto, a Mineração de Texto (MT) tem um papel importante ao processar essa grande quantidade de texto. A MT, refere-se ao processo de extrair informações úteis em documentos no formato textual não-estruturado através da identificação de conhecimento e exploração de padrões.

Muitos algoritmos de processamento de linguagem natural(PNL) necessitam de um pré-processamento pesado. Enquanto os pesquisadores tem direcionado muito esforço para aumentar a eficiência dos algoritmos de PNL , pouco esforço se tem direcionado para o grande custo do pré-processamento.

Dentre os algoritmos de PNL, os que transformam palavras de uma base de textos em representações vetoriais(word embedding) estão em crescente uso. Esses métodos permitem extrair significados para palavras baseado nos seus contextos. Por exemplo, se vec(w) representa o word embedding da palavra w, então vec("Berlin") - vec("Germany") terá uma valor parecido de vec("Paris") - vec("France"). Muitos desses algoritmos tem como entrada matrizes largas, como a matriz de Coocorrência palavra-palavra em contexto ou a Pointwise Mutual Information Matrix(PMI). Por exemplo, o famoso algoritmo GloVe de Pennington et al.'s (2014) para a construção de word-embeddings de palavras, tem como entrada uma matriz de coocorrência M onde Mij é uma estatística relacionada a quantas vezes as palavras i e j apareceram perto uma da outra no documento. Essa matriz tem alto custo computacional para construção, e considerando que muitas vezes o corpo de textos é dinâmico, com muitas mudanças ao longo do tempo, esse custo pode se tornar um grande problema na aplicação de algoritmos de PNL.

Pennington et al.'s (2014), na implementação do GloVe, propôs uma solução que busca achar regiões de alta densidade na matriz pra calculá-las de forma separada, enquanto as outras regiões são calculadas usando estruturas esparças. Porém, essa solução não aproveita o poder computacional que as GPU's oferecem.

Nesse trabalho será proposta uma solução baseada na arquitetura paralela GPU, usando a API de programação paralela CUDA, para a construção da matriz de Coocorrência palavra-palavra no contexto. O que abre caminho para o cálculo de suas variações como a Pointwise Mutual Information Matrix(PMI) ou Positive Pointwise Mutual Information Matrix(PPMI).

[1]Bolsista
[2]Orientador

2 SOLUÇÃO PARALELA

Apresentaremos algoritmos, que foram implementados em um ambiente paralelo com a API Cuda.

2.1 Construção de Matrizes de Coocorrência

Focaremos no cálculo da matriz de coocorrência palavra-palavra dentro de uma janela de contexto.

Primeiro vamos abordar como seria a construção da matriz de coocorrência se houvesse memória infinita na GPU para então abordar como resolver o problema de memória insuficiente.

Como representação esparça para a matriz usaremos uma lista de tuplas (w1, w2, c) onde w1 e w2 são os id's das palavras e c é o contador do número de vezes que o par (w1, w2) apareceu em uma mesma janela de contexto.

2.1.1 Construção de Matriz de Coocorrência com Memória infinita na GPU

O cálculo da matriz palavra-palavra coocorrência pode ser dividido nos seguintes passos:

1. Transformar todas palavras do texto em um id numérico de 1 a T conforme dado pelo vocabulário. Onde T é o tamanho do vocabulário.

2. Construir um array de tuplas V com todas tuplas de palavras em uma mesma janela no texto. Por exemplo, se o texto é [w1 w2 w3 w4 ...] e o tamanho da janela é 2, então V = {(w1, w2), (w1, w3), (w2, w1), (w2, w3), (w2, w4), (w3, w1) ....}

   Nota-se que w1 e w4 não formam uma tupla, pois a janela tem tamanho 2.

3. Ordernar array V

4. Criar um array V_unique com todos elementos únicos de V, ou seja, sem repetição.

5. Adicionar um terceiro elemento em cada tupla de V_unique, representando o número de vezes que essa tupla apareceu em V. Isso pode ser feito através de uma busca binária em V.

Pra etapa 1 foi feito um função kernel para executar em paralelo. As etapas 2, 3 e 4 foram feitas usando as funções sort, unique e upper_bound da biblioteca Thrust do Cuda.

Se o texto tem N palavras e a janela tamanho d. A complexidade do uso de memória é O(N * d). O uso de memória ultrapassou 25 Gb pra datasets com 1 bilhão de palavras. O que pode ser inviável para GPU na maioria dos casos.

No próximo tópico, apresentamos uma solução para resolver o problema de falta de memória na GPU.

2.1.2 Solução pra falta de memória

Para resolver esse problema, o algoritmo divide a base textual em vários segmentos, de forma que seja possível resolver o problema pra cada segmento de forma independente, pra então fazer uma união desses segmentos na CPU e obter um resultado final.

O algoritmo executa as seguintes etapas:

1. Divisão do texto em K segmentos.

2. Execução do algoritmo da seção 3.2.1 para cada segmento. O i^{o} segmento resultará em uma lista A[i] de tuplas ordenadas representando o resultado final pra esse segmento.

3. Unir as listas de tuplas em uma única lista com o resultado final através de um algoritmo que utiliza uma estrutura de fila de prioridades para indentificar a qualquer momento a menor tupla dentre cada primeira tupla da lista A[i], para cada i.

Ao fim da execução o resultado será uma única lista de tuplas representando os pares de palavras e suas coocorrências.

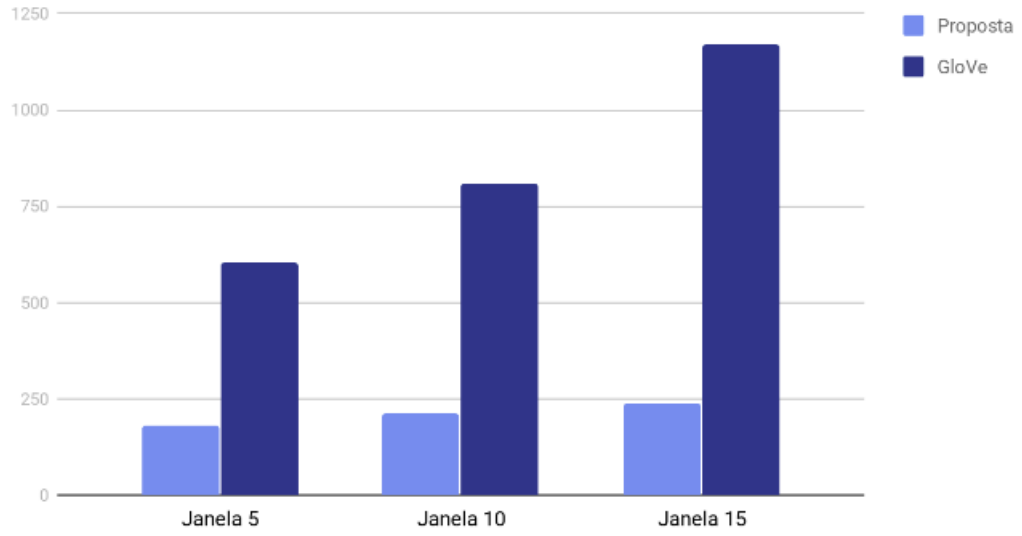
Caso ocorre limitação de memória RAM, cada lista A[i] pode ser armazenada em disco.

3 Resultados Computacionais

Os programas foram testados em um computador com placa de vídeo GeForce GTX TITAN Black, 24 cores de CPU modelo Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz.

Foi usado uma base textual com 800 milhões de palavras, com documentos de origens diversas, como sites de notícias da internet.

Seguem os gráficos com tempo de execução total da nossa proposta para construção da matriz de coocorrência e a proposta estado da arte utilizada por Pennington et al.'s (2014).



Como mostrado pelo gráfico a nossa proposta obteve speedup de 3.36x para janela de tamanho 5, 3.85x para janela de tamanho 10, 4.87x para janela de tamanho 15.

4 CONCLUSÃO

Esse trabalho apresentou uma proposta paralela para geração da matriz de coocorrência palavra-palavra, obtendo um speedup entre 3x-5x na comparação com o estado da arte de Pennington et al.'s (2014).

Essa solução pode se tornar uma alternativa no cálculo da matriz de coocorrência de algoritmos, como o GloVe, de aprendizado não supervisionado em linguagem natural, diminuindo o alto custo de execução desses algoritmos.

Referências

[1] Arora, Sanjeev, Ge, Rong, Halpern, Yonatan, Mimno, David, Moitra, Ankur, Sontag, David, Wu, Yichen, and Zhu, Michael. A practical algorithm for topic modeling with provable guarantees. In Proceedings of The 30th International Conference on Machine Learning, pp. 280–288, (2013).

[2] J. Gantz and D. Reinsel, *'The digital universe decade-are you ready*, *Proc. White Paper, IDC*(2010).

[3] Lee, Moontae, Mimno, David, and Bindel, David. Robust spectral inference for joint stochastic matrix factorization. In Advances in neural information processing systems, (2015).

[4] Lin, Jimmy. Scalable language processing algorithms for the masses: A case study in computing word cooccurrence matrices with MapReduce. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP '08, pp. 419–428, Stroudsburg, PA, USA, (2008)

[5] Levy, Omer and Goldberg, Yoav. Neural word embedding as implicit matrix factorization. In Advances in Neural Information Processing Systems, pp. 2177–2185, (2014).

[6] Mikolov, Tomas, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, (2013).

[7] Pennington, Jeffrey, Socher, Richard, and Manning, Christopher D. Glove: Global vectors for word representation. Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014), 12:1532–1543, (2014)

[8] TAN, A.-H, *Text mining: the state of the art and the challenges* , KDAD, Beijing. China. PAKDD, p. 71-76., (1999).