Geração de Matrizes de Coocorrência na GPU com Aplicações em Bases Textuais

Chayner Cordeiro Barros, Wellington Santos Martins

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Alameda Palmeiras, Quadra D, Campus Samambaia – 74.690-900 – Goiânia – GO – Brazil

{chaynercordeiro,wellington}@inf.ufg.br

Abstract. *Co-occurrence matrices are computational artifacts used in applications such as the representation models, in typical NLP (Natural Language Processing) tasks such as machine translation, text mining, document classification. Its construction is quite computationally expensive, because it requires the analysis of the co-occurrence relationship between the terms existing in a corpus, and may require a lot of memory to store these relationships, if the vocabulary, the context window and the corpus used are very large. To overcome these limitations, our solution was based on a hash-like data structure, capable of efficiently storing an inherently sparse matrix of large dimensions. This solution adapts easily to different architectures, having been implemented and tested in applications with single thread, multicore and manycore architectures.*

Resumo. *As matrizes de coocorrência são artefatos computacionais utilizados em aplicações como a construção de modelos de representação, em tarefas típicas de NLP (*Natural Language Processing*) como a tradução por máquina, na mineração de texto, na classificação de documentos. Sua construção é bastante onerosa computacionalmente, porque demanda a análise da relação de coocorrência entre os termos existentes num* corpus*, podendo exigir bastante memória para armazenar essa relação, caso o vocabulário, a janela de contexto e o* corpus *empregados sejam muito grandes. Para contornar essas limitações, nossa solução é baseada numa estrutura de dados do tipo* hash*, capaz de armazenar de forma eficiente uma matriz inerentemente esparsa de grandes dimensões. Esta solução se adapta facilmente a diferentes arquiteturas, tendo sido implementada e testada em aplicações com arquiteturas* single thread*,* multicore *e* manycore*.*

1. Introdução

Matrizes de coocorrência são estruturas de dados que permitem o armazenamento da relação de coocorrência entre pares de objetos. Dependendo da aplicação, esses objetos podem ser pixels, letras, palavras ou até mesmo documentos inteiros. A aplicabilidade dessas matrizes se mostra muito abrangente, em áreas como reconhecimento de padrões, processamento digital de imagens, mineração de textos e processamento natural de linguagem (NLP).

Na mineração de textos e em NLP, essas matrizes são aplicadas, em geral, para extrair relações semânticas entre termos existentes em um *corpus* analisado. Com essas

relações podemos construir representações vetoriais, que permitem um tratamento computacional mais eficiente das informações contidas no texto, e que podem ser usadas em aplicações mais especializadas como sistemas de pergunta/resposta, de recomendação, de classificação de documentos, entre outros.

Nossa pesquisa evidencia essa flexibilidade da matriz de coocorrência, e compara o desempenho da solução proposta com algoritmos capazes de extrair essa relação semântica entre termos, assim como representar esses termos num espaço vetorial com vetores densos [Pennington et al. 2014] [Mikolov et al. 2013]. Essa flexibilidade está relacionada, inclusive ao menor consumo de memória alcançado pelo uso de tabelas *hash*, permitindo que bases maiores sejam processadas em equipamentos com menos recursos.

2. Trabalhos Relacionados e Solução Proposta

O modelo de representação vetorial GloVe [Pennington et al. 2014] fatora uma matriz de coocorrência para construir os vetores que representam os termos enumerados no vocabulário, e obtém dela a relação semântica entre esses termos. Os autores conseguiram acelerar o cálculo da matriz de coocorrência de palavra-palavra com uma solução que busca encontrar regiões de alta densidade na matriz, para então calculá-las de forma separada. Uma outra abordagem [Lin 2009] propõe o uso da estratégia paralela de MapReduce para acelerar o cálculo dessas matrizes de coocorrência. Contudo, não é de nosso conhecimento, soluções anteriores que aproveitam o poder computacional que as GPUs oferecem para produzir matrizes de coocorrência em bases textuais.

A nossa solução explora o paralelismo das GPUs (e CPUs) com uma estrutura de dados do tipo *hash* que propusemos anteriormente [Barros and Martins 2019], e que utilizamos para representar em memória o vocabulário, os *embeddings*, e a própria matriz de coocorrência. Além de ser bastante eficiente para o armazenamento e recuperação das informações, esta solução se mostrou bastante adaptável às arquiteturas *multicore* (CPU) e *manycore* (GPU), e obteve desempenho superior quando comparada com um algoritmo estado da arte.

3. Experimentos

Para a realização dos experimentos foi utilizado um computador com a seguinte configuração: uma GPU NVIDIA GTX 1050 Ti, processador AMD A10-7850K, 4 cores @ 3.7 GHz, 16 GB de memória DDR3, e um SSD NVMe Samsung 970 EVO como dispositivo de armazenamento.

O primeiro experimento consiste na geração da própria matriz de coocorrência a partir de uma base textual. A base de dados utilizada nesse experimento foi a *1 Billion Words Benchmark* [Chelba et al. 2013]. Este conjunto de dados consiste em pouco menos de 1B de tokens (cerca de 719M) e é comumente usado para avaliar modelos de linguagem devido à grande quantidade de termos únicos, incluindo vários ruídos.

Foram construídas três implementações equivalentes, uma para cada algoritmo e arquitetura específica. As implementações paralelas foram comparadas com o algoritmo sequencial proposto e com os métodos otimizados de extração de vocabulário e criação da matriz de coocorrência implementados no GloVe [Pennington et al. 2014], executando sequencialmente. Utilizou-se janelas de contexto simétricas de tamanho 2 em todas as

medições, inclusive as realizadas com o GloVe em nossos experimentos. O algoritmo proposto não define limites para os principais parâmetros, como o tamanho da janela e do vocabulário. Contudo, para efeitos de comparação, adotou-se a janela de tamanho 2 para que fosse possível efetuar os experimentos no hardware disponível, sem que fosse necessário diminuir o tamanho do vocabulário.

A Tabela 1 apresenta os tempos gastos (em segundos) na execução das versões sequencial, *multicore*, *manycore* e os tempos de execução dos módulos do GloVe que extraem o vocabulário e geram a matriz de coocorrência. Esses valores foram computados a partir da média de 10 execuções utilizando a ferramenta `time` do Linux. O *speed-up* foi computado, comparando-se as três versões produzidas com a versão sequencial do *GloVe*; e comparando-se as versões paralelas (*multicore* e *manycore*) com a versão sequencial proposta.

Na versão *manycore* foram utilizados *streams* CUDA para permitir a execução assíncrona dos *kernels*, intercalada com a transferência de dados entre CPU/GPU, para obter um ganho de performance.

	Sequencial	Multicore	Manycore	GloVe
Vocabulário + Matriz	**245,397**	**85,130**	**15,210**	**447,046**
Total (incluindo I/O)	**266,575**	**108,613**	**21,829**	**459,563**
Desvio-Padrão	**2,164**	**1,228**	**0,312**	**2,316**
Streams	*N/A*	*N/A*	**2**	*N/A*
Speed-up × Seq.	*N/A*	**2,454**	**12,212**	-
Speed-up × GloVe	**1,724**	**4,232**	**21,053**	*N/A*

Tabela 1. Tempos de execução (em segundos) das versões propostas e dos módulos de vocabulário e matriz de coocorrência do GloVe, juntamente com o *speed-up* obtido.

O segundo experimento consiste na construção de *embeddings* utilizando nossa proposta de implementação paralela do algoritmo *Random Indexing* [Sahlgren 2005]. Para este experimento foi utilizado como *corpus* um extrato da Wikipedia (em inglês) de 2014 com aproximadamente 8.5 GiB, com um vocabulário de 5.021.032 termos e um total de 1.546.379.363 *tokens*. A Tabela 2 apresenta os valores de acurácia medidos em um dos testes realizados com os *embeddings* produzidos. Nesse teste utilizamos o *Google Analogy Test Set* e comparamos os resultados com os resultados obtidos através do mesmo conjunto de testes, usando os *embeddings* gerados pelo *GloVe* e pelo *Word2Vec*, treinados no mesmo *corpus* e vocabulário.

	RI 300 P1	RI 1024 P1	GloVe 300 P14	Word2Vec 300 P5
Questões vistas/total	**100,00%** (19544/19544)	**100,00%** (19544/19544)	82,03% (16032/19544)	82,03% (16032/19544)
Acurácia Semântica	19,65% (1743/8869)	**21,19%** (1879/8869)	0,94% (53/5657)	8,68% (491/5657)
Acurácia Sintática	8,67% (926/10675)	9,26% (989/10675)	1,81% (188/10375)	**58,70%** (6090/10375)
Acurácia Total	13,66% (2669/19544)	14,67% (2868/19544)	1,50% (241/16032)	**41,05%** (6581/16032)

Tabela 2. Acurácia dos vetores produzidos com os algoritmos *Random Indexing*, *GloVe* e *Word2Vec*

As colunas representam, em ordem da esquerda para a direita, os valores obtidos com os vetores gerados pelo algoritmo *Random Indexing* com 300 e 1024 dimensões, respectivamente, produzidos em uma única época (P1); os vetores gerados pelo *GloVe* com 300 dimensões e treinados em 14 épocas, sendo esses vetores obtidos na época com

menor custo de um total de 25 épocas; e os vetores gerados pelo *Word2Vec* com 300 dimensões e treinados em 5 épocas.

Como pode ser observado na Tabela 2, o algoritmo *Random Indexing*, utilizando nossa implementação de tabela *hash*, produz os vetores de 100% dos tokens do vocabulário. Isso permite, por exemplo, que tarefas de NLP tenham maior acurácia, ao diminuir a probabilidade de um termo importante não ser encontrado, tendo assim, que ser substituído por um termo "coringa".

O teste realizado consiste numa tarefa de analogia desenvolvida pelo Google, que objetiva avaliar a analogia como um grau de similaridade existente nos *embeddings*. Desta forma, para cada três palavras existentes no *dataset*, uma quarta é solicitada ao modelo a partir do vocabulário disponível. Com os resultados, pode-se observar uma qualidade dos *embeddings*, como a acurácia sintática, relacionada com a capacidade dos vetores representarem uma similaridade estrutural, como no caso de plural; ou a acurácia semântica, onde os vetores conseguem representar associações de sentido, como sinonímia.

4. Trabalhos Futuros

Os experimentos demonstram a vantagem de se acelerar a construção de matrizes de coocorrência, a partir de estruturas de dados do tipo *hash* em arquiteturas *manycore*, assim como o uso de tais matrizes na realização de tarefas de mineração de texto e representação de palavras.

Tendo em vista os resultados obtidos, planeja-se direcionar o trabalho para a análise e experimentação de algoritmos de classificação automática e *clusterização* de documentos, que fazem uso das matrizes de coocorrência, com o intuito de contribuir com a aceleração de tais algoritmos.

Referências

Barros, C. C. and Martins, W. (2019). Acelerando a construção de vocabulário e matriz de co-ocorrência em bases textuais. In *Anais Principais do XX Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 418–429. SBC.

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. (2013). One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

Lin, J. (2009). 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*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Sahlgren, M. (2005). An introduction to random indexing. In *Methods and applications of semantic indexing workshop at the 7th international conference on terminology and knowledge engineering*.