GPU-Accelerated Non-negative Matrix Factorization for Text Mining

Volodymyr Kysenko[1], Karl Rupp[2,3], Oleksandr Marchenko[1], Siegfried Selberherr[2], and Anatoly Anisimov[1]

[1] Faculty of Cybernetics, Taras Shevchenko National University of Kyiv, Ukraine
[2] Institute for Microelectronics, TU Wien, Austria
[3] Institute for Analysis and Scientific Computing, TU Wien, Austria

Abstract. An implementation of the non-negative matrix factorization algorithm for the purpose of text mining on graphics processing units is presented. Performance gains of more than one order of magnitude are obtained.

1 Introduction

The automatic extraction of high-quality information from text, typically referred to as text mining, has received a lot of attention in many areas. Due to the vast amount of text to be processed, there is a virtually insatiable need for computational power. We address this demand by an implementation of the non-negative matrix factorization (NMF) algorithm [1] and its application to document clustering on graphics processing units (GPUs).

In contrast to traditional central processing units (CPUs), the computational power of modern GPUs already ranges into the tera-floating point operations per second (TFLOPs) regime. The higher flexibility of modern GPU architectures also allows for the use of GPU-hardware for non-graphics application. As a consequence, such general purpose computations on graphics processing units (GPGPUs) have gained a lot of popularity recently. Significant performance gains within scientific applications have been reported in many different applications, e.g. [2].

The high computational power of GPUs can only be accessed, if the underlying algorithm provides a sufficiently high degree of fine-grained parallelism in order to occupy thousands of light-weight threads simultaneously. Since this constitutes a considerable paradigm-shift compared to single-threaded implementations, existing codes often need to be rewritten in order to benefit from GPU acceleration. This problem is at least partially addressed by software libraries, which provide basic functionalities via a high-level interface. In the context of the NMF algorithm, basic linear algebra operations are required, which are well-studied and well-developed for GPUs [5]. A number of different libraries exists and most of them rely on CUDA [3] technology, for example, CUBLAS and MAGMA. In our work the C++ OpenCL-based [4] Vienna Computing Library (ViennaCL) [6,7] is used. It offers a convenient means to run custom compute

G. Bouma et al. (Eds.): NLDB 2012, LNCS 7337, pp. 158–163, 2012.

kernels not provided with the library and allows for a comparison on a broader range of GPUs from different vendors.

Aspects of document clustering and their link to matrix decompositions are discussed in Sec. 2. A formal description of the NMF algorithm is given in Sec. 3, whereas implementation details are discussed in Sec. 4. Results are discussed in Sec. 5 and a conclusion is drawn in Sec. 6.

2 Document Clustering

Clustering is a process of collecting a set of objects into subsets (clusters) such that objects inside a cluster are in a certain predefined sense more similar to each other than to objects from other clusters.

In this work we consider the application of clustering for grouping documents by topics. More formally, for a set of m text documents, the goal of clustering by topics is to find a division of documents into groups such that documents from one group share common topics. It is crucial to note that the list of topics is typically not known prior to the clustering process. Therefore, such a clusterization can also be interpreted as an automatic document categorization and topics extraction process.

Traditional methods of document clustering are often based on the vector space model (VSM) [9]. In this model every document is represented as a term-frequency vector. Therefore, VSM uses words as a measure for similarity between different documents. Various metrics can be defined for the vectors of terms, for example a cosine distance or an Euclidian distance. Due to this representation of documents in terms of linear algebra, traditional clustering techniques can be applied.

One of the major disadvantages of the VSM model is that terms are assumed to be statistically independent from each other. However, this is often not the case in real-world texts, where topics, concepts, and semantics are key features of each document. In order to extract these key features from the texts, special techniques referred to as *feature extraction* have been developed. The goal of such methods is an extraction of core concepts from the individual texts and a representation of documents as a combination of these. Singular value decomposition [10] or NMF [12] are often used as a basis for feature extraction methods. Methods of this type are also known as low-rank matrix approximation or latent semantic analysis. Further note that applications of these methods are not limited to document clustering. They has been successfully used for solving a wide range of natural language processing problems such as cross-language retrieval, information indexing [10], or selectional preference induction [11].

For the remainder of this work only NMF is considered. Generally, NMF is a decomposition of a matrix V into a product of two matrices W and H, where the additional constraint of non-negativity of the entries in the matrices V, W, H is imposed. It is important to note that such a decomposition is not necessarily unique and neither W nor H need to be orthogonal. NMF factorization became popular with the publication of Lee and Seung [1], where two algorithms for computing non-negative matrix factorizations are proposed.

Other methods such as the projected gradient method [13], alternating non-negative least squares [14], or sparse encoding [15] were proposed later. For this paper, the original Lee-Seung method, which is also known as *multiplicative update rules algorithm*, with the Frobenius norm as a cost function is considered.

In text mining NMF is usually applied to the term-document matrix (TD matrix). Every row in the TD matrix corresponds to one document, while every column of the matrix corresponds to one term. For m documents and a total number of n terms, the TD matrix V consequently is of size $m \times n$. NMF is used to decompose V into two matrices W and H with sizes $m \times k$ and $k \times n$ respectively, where typically $k \ll \min(m, n)$ is set to the expected number of clusters. In the following we will refer to the parameter k as the *number of features*. Intuitively, k can be interpreted as follows: The TD matrix V ('documents' $\times$ 'terms') is decomposed into a product of two matrices. The matrix W relates documents and features, while H relates features and terms. As a result of matrix product properties, every document is thus presented as a linear combination of the extracted features. Traditional clustering techniques[8] can therefore be applied to the rows of W.

NMF as a tool for natural language processing possesses several advantages over other feature extraction methods. First, the matrices W and H have non-negative entries only, which results in an easier interpretation in terms of text mining. Second, the columns of W do not need to be orthogonal. Hence, the extracted topics are allowed to share common senses, which seems to be quite usual for real-world documents.

3 The Non-negative Matrix Factorization Algorithm

In the following, the NMF algorithm proposed by Lee and Seung [1] using the Frobenius norm is described. The objective function is

$$\min_{W,H} ||V - WH||_F^2 \ , \tag{1}$$

where the entries of W and H need to be non-negative. Note that the minimum in (1) is typically non-zero. Given arbitrary initial matrices W_0 and H_0, the NMF algorithm consists of an iterative application of the following two steps:

$$(H_k)_{i,j} = (H_{k-1})_{i,j} \times \frac{(W_{k-1}^T V)_{i,j}}{(W_{k-1}^T W_{k-1} H_{k-1})_{i,j}} \ , \tag{2}$$

$$(W_k)_{i,j} = (W_{k-1})_{i,j} \times \frac{(V H_{k-1}^T)_{i,j}}{(W_{k-1} H_{k-1} H_{k-1}^T)_{i,j}} \ . \tag{3}$$

Here $(\cdot)_{i,j}$ refers to the entry in row i and column j of the matrix in parentheses.

In practise, (2) and (3) are repeated until either a stationary point or a maximum number of iterations is reached. Lee and Seung proved two main properties of this algorithm. First, the objective function (1) is non-increasing with k. Second, W and H become constant, if and only if they represent a stationary point.

4 Implementation

The iterative NMF algorithm (2) and (3) can almost directly be implemented with the features provided by ViennaCL. Two types of matrix operations are required: matrix-matrix-multiplications and element-wise manipulations. While the former are provided by ViennaCL directly, the latter requires a simple custom OpenCL kernel for setting up the matrix indicated by parentheses in (2) and (3).

In the typical case, where k is small compared to m and n, the computationally most expensive operations are the matrix-matrix products with V. This is the case, because only V is of size $m \times n$, while at least one of the dimensions of all other matrices are given by k.

In practical applications it is often observed that the TD matrix V is sparse. As a consequence, computational efficiency can be improved substantially and memory requirements can be reduced significantly by exploiting the structural information. Memory consumption is particularly a concern when using GPUs due to the typically limited amount of memory on GPU adapters compared to main memory on the host machine. Thus, by using a sparse matrix storage format such as the compressed sparse row format (CSR)[16], we are able to process a much bigger set of documents than in the case of a dense matrix type.

It is important to note that (2) and (3) in principle require products with V from the left and from the right. However, the CSR-format used for the storage of V allows for an efficient implementation of the matrix-matrix product only, if V is multiplied from the left, which is not the case for the operation $W^T V$. There are two possible remedies in this case: The first is to store V in a compressed column format in addition. Besides additional memory, this requires a dense-matrix-sparse-matrix multiplication kernel. The second option is to set up and store V^T in CSR format in addition. Rewriting $W^T V = ((W^T V)^T)^T = (V^T W)^T$ enables the reuse of the multiplication kernel, but another entry-wise manipulation kernel must be provided. Since the entry-wise manipulation kernel is simpler, we stick with the second option.

5 Benchmark Results

For our experiments we have used recent mid- to high-end consumer hardware: An Intel Core i7 960 CPU with 3.2 GHz clock freqency, an NVIDIA GeForce 470 GTX GPU, and an AMD Radeon 6970 HD GPU.

For a comparison of our GPU-accelerated implementation we developed a purely CPU-based version of the NMF algorithm. The goal of this version is not only performance numbers, but also validation of numerical accuracy. The open-source linear algebra library Eigen [17] was used for this purpose. The CPU version was compiled with maximal optimization level and with support for SIMD instructions (SSE2) and OpenMP enabled.

Our benchmarks include both real-world and artificially generated matrices. As a benchmark based on real-world data, the *Newsgroups (NG)* dataset [18] is used. Texts in this dataset are messages gathered from 20 different newsgroups

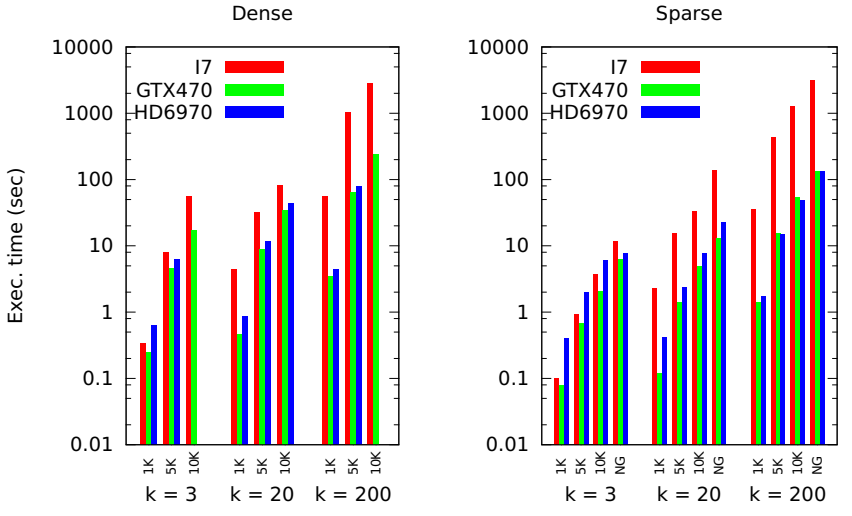

Fig. 1. Execution times for all test matrices on different platforms

with a total number of 18827 documents. Due to the nature of the newsgroup messages, which are usually quite short, the TD matrix V for this dataset is very sparse. The resulting size of the term-document matrix is $18\,827 \times 87\,014$, while the number of non-zero elements is 1 553 867. Note that a dense matrix of the same dimensions would require the storage of more than one billion entries, which may already exceed the memory available in current desktop computers. As an artificial benchmark we generated a set of random matrices, with sizes varying from 1000×1000 (1K) to 10000×10000 (10K). All matrices from this set have about 1% of non-zero elements, which allows for the use in both dense and sparse tests. In both cases values of 3, 20, and 200 are used for the number of features k.

Benchmark results of our experiments with all matrices are shown in Fig. 1. It can be seen that the use of a sparse matrix type instead of a dense matrix type leads to a performance gain on both CPU and GPU. While the gain of using a sparse matrix type is about one order of magnitude for $k = 3$ on the CPU, the difference is only a factor of two for $k = 200$. On GPUs the performance difference is around five to ten for $k = 3$ and $k = 20$, and around a factor of three to five on both the NVIDIA and the AMD GPU. In the latter case, the dense matrix-matrix multiplication on GPUs achieves higher performance due to the more uniform size of the matrices involved, thus the difference is smaller. A comparison of execution times for CPU and GPUs on all test matrices reveals a difference in execution times by a factor of up to 25 for the largest matrices and $k = 200$. For $k = 3$ a performance gain from the use of GPUs of only a factor of up to two is obtained, whereas $k = 20$ leads to a gain of almost one order of magnitude.

6 Conclusion

Our investigations of the acceleration of the NMF algorithm by GPUs for the use in text mining tasks show that a performance gain of more than one order of magnitude can be obtained. Consequently, modern GPU hardware must not be ignored for such purposes, whenever performance is of importance.

Large parts of the NMF algorithm are ported to GPUs by reusing high-level functionality already provided with common GPU libraries such as ViennaCL. Therefore, the entry-barrier to GPU computing is lower than it may appear at first sight. Most linear algebra functionality is ready to be reused within other algorithms commonly employed in the field of text mining.

Our implementation of the NMF algorithm presented in this work is to be made freely available with ViennaCL 1.3.0.

References

1. Lee, D.D., Seung, H.S.: Algorithms for Non-Negative Matrix Factorization. In: NIPS (2000)
2. Hwu, W.W. (ed.): GPU Computing Gems. MKP (2011)
3. NVIDIA CUDA, http://www.nvidia.com/
4. Khronos Group. OpenCL, http://www.khronos.org/opencl/
5. Agullo, E., et al.: Numerical Linear Algebra on Emerging Architectures: The PLASMA and MAGMA projects. J. Phys.: Conf. Ser. 180(1) (2009)
6. ViennaCL, http://viennacl.sourceforge.net/
7. Rupp, K., et al.: ViennaCL - A High Level Linear Algebra Library for GPUs and Multi-Core CPUs. In: Proc. Intl. Workshop on GPUs and Scientific Applications (GPUScA 2010), pp. 51–56 (2010)
8. Xu, R., Wunsch II, D.: Survey of Clustering Algorithms. IEEE Trans. on Neural Networks 16(3), 645–678 (2005)
9. Salton, G., et al.: A Vector Space Model for Automatic Indexing. Communications of the ACM 18(11), 613–620 (1975)
10. Deerwester, S., et al.: Indexing by Latent Semantic Analysis. Journal of the American Society for Information Science 41(6), 391–407 (1990)
11. Van de Cruys, T.: A non-negative tensor factorization model for selectional preference induction. In: Proc. Workshop on Multiword Expressions, pp. 83–90 (2009)
12. Xu, W., et al.: Document Clustering Based on Non-Negative Matrix Factorization. In: Proc. 26th Intl. Conf. Research and Development in Information Retrieval, pp. 267–273 (2003)
13. Pauca, V., et al.: Text Mining Using Non-Negative Matrix Factorizations. In: Proc. 4th SIAM Intl. Conf. Data Mining (2004)
14. Amy, L., Carl, M.: ALS Algorithms Nonnegative Matrix Factorization Text Mining. SAS NMF Day (2005)
15. Hoyer, P.: Non-Negative Sparse Coding. In: Proc. IEEE Workshop on Neural Networks for Signal Processing (2002)
16. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM (2003)
17. An Overview of Eigen. First Plafrim Scientific Day. Bordeaux (May 31, 2011), http://eigen.tuxfamily.org/
18. 20 Newsgroups, http://people.csail.mit.edu/jrennie/20Newsgroups/