GPU Accelerated Semantic Search Using Latent Semantic Analysis

Alexandru Iacob, Lucian Itu, Lucian Sasu, Florin Moldoveanu, Cosmin Ioan Nita, Ulrich Foerster, Constantin Suciu
Siemens Corporate Technology, Siemens SRL, Braşov, Romania
Siemens AG,
Transilvania University of Braşov, Braşov, Romania

Abstract — **During the last decade, the concept of general purpose computing on graphics processors has seen increased adoption. One candidate for leveraging a GPU for improved computational performance is the semantic text analysis. With increasing amounts of data and stringent response time requirements, improving the underlying implementation of semantic analysis becomes critical. We consider Latent Semantic Analysis (LSA), a numerical technique used to extract information from large collections of text documents. We present a parallel LSA implementation on the GPU, using CUDA and the cuSolver library for common matrix factorization and triangular solve routines for dense matrices. The performance is compared to that of a traditional CPU-based LSA implementation employing an optimized Basic Linear Algebra Subprograms library. The experimental results indicate that the GPU leads to significantly larger throughput compared to the CPU-only implementation: a speed-up of up to 21 is achieved.**

Keywords— semantic search, information retrieval, Latent Semantic Analysis, GPU, CUDA

I. INTRODUCTION

Semantic indexing is a popular technique used to access and organize large amounts of unstructured text data. Semantic search seeks to improve document retrieval accuracy by understanding searcher intent and the contextual meaning of terms as they appear in the searchable data space, whether on the web, or within a closed system. With the large amount of data being collected annually, automated methods are required for extracting valuable information from the data [1].

Generic text summarization approaches are divided into four classes. The first class of extraction methods employs simple techniques for scoring sentences, e.g. the sentence placement within the document or the occurrence of a word from the title in a sentence [2]. The next class includes approaches based on a document corpus (corpus-based methods) [3], e.g. TFIDF (Term Frequency-Inverse Document Frequency) [4]. The third class consists of methods which take a discourse structure into account, e.g. the lexical chains method which searches for chains of context words in the text [5]. The last group is called knowledge-rich approaches. It is the most advanced one, but can be applied only in certain domains. A relatively novel approach for text summarization employs Latent Semantic Analysis (LSA), belonging to the TFIDF document indexing methods.

Latent Semantic Analysis is a numerical technique used to extract information from large collections of text documents [6, 7]. These document collections often contain more than 10000 unique documents and / or 5000 unique terms. LSA is employed for these collections for finding relationships between various terms, sentences, and full documents. A large matrix representing term-document association data is used to construct a *semantic* space, wherein terms and documents that are closely associated are placed next to each other. Next, LSA applies singular value decomposition (SVD) to the matrix. Singular value decomposition allows for an arrangement of the space which reflects the major associative patterns in the data, and ignores the smaller, less important influences. As a result, terms that do not actually appear in a document may still end up close to the document, if that is consistent with the major patterns of association in the entire dataset. The core of the SVD algorithm requires an eigen decomposition of the matrix, which has been traditionally a computationally intensive operation [7, 8, 9]. This renders SVD a computationally expensive algorithm which makes it an excellent candidate for high performance computing approaches, e.g. parallelization. Previous research activities propose the utilization of programmable pipeline Graphics Processing Units (GPUs) for high speed string matching [10].

Graphics Processing Units (GPUs) are dedicated processors, designed originally as graphic accelerators. Before the development of specialized frameworks for general purpose GPU programming were introduced, a mapping to the graphical context was required. This mapping was often complex and suboptimal. Since the Compute Unified Device Architecture (CUDA) language was introduced in 2006 by NVIDIA as a graphic application programming interface, the GPU has been used increasingly in various areas of scientific computations due to its superior parallel performance and energy efficiency [11].

The GPU is viewed as a compute device which is able to run a very large number of threads in parallel inside a kernel (a function, written in C language, which is executed on the GPU and launched by the CPU). The threads of a kernel are organized at three levels: blocks of threads are organized in a three dimensional (3D) grid at the top level, threads are organized in 3D blocks at the mid level, and, at the lowest levels, threads are grouped into warps (groups of 32 threads formed by linearizing the 3D block structure along the *x*, *y* and *z* axes, respectively) [12].

The goal of the current work is to evaluate the viability of using GPUs to speed-up the Latent Semantic Analysis based string searching algorithm. Moreover, to increase the information retrieval accuracy, several preprocessing techniques have been considered. Stemming was used to reduce inflected or derived words to their word stem, base or root form. Also, because the purpose of the algorithm was to be able to operate with large amounts of any type of text data, especially unstructured sets, a tokenizing procedure was employed.

The paper is organized as follows. Section II performs a theoretical introduction of Latent Semantic Analysis search algorithm and introduces its GPU accelerated version. Section III displays the results obtained with the different kernel versions for different Kepler GPUs. Finally, section IV draws the conclusions.

II. METHODS

A. Latent Semantic Analysis

Latent Semantic Analysis is a theory and method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text [13]. The underlying idea is that the aggregate of all the word contexts in which a given word does and does not appear provides a set of mutual constraints that largely determines the similarity in meaning of words and sets of words to each other. LSA is a fully automatic mathematical / statistical technique for extracting and inferring relations of expected contextual usage of words in passages of discourse. It is not a traditional, natural language processing or artificial intelligence approach: it uses no humanly constructed dictionaries, knowledge bases, semantic networks, grammars, syntactic parsers, or morphologies, and takes as input only raw text parsed into words defined as unique character strings and separated into meaningful passages or samples such as sentences or paragraphs.

The first step is to represent the text as a matrix in which each row stands for a unique word and each column stands for a text passage or other context. Each cell contains the frequency with which the word of its row appears in the passage denoted by its column. Next, the cell entries can be subjected to a preliminary transformation, in which each cell frequency is weighted by a function that expresses both the importance of the word in the particular passage and the degree to which the word type carries information in the domain of discourse in general [13]. This corresponds to the creation of a term by document matrix $A = [A_1, A_2, \dots, A_n]$, vector A_i representing the weighted term-frequency vector of word i in the document under consideration. If there are a total of m terms and n documents in the collection, a matrix of $m \times n$ will be generated. Since words typically do not appear in each sentence, matrix A is sparse.

Next, LSA applies singular value decomposition to the matrix. This is a form of factor analysis, or more properly the mathematical generalization of which factor analysis is a special case. Given the $m \times n$ matrix, where without loss of generality $m \geq n$, the SVD of A is defined as:

$$A = U\Sigma V^T \tag{1}$$

where $U = [u_{ij}]$ is an $m \times n$ column-orthonormal matrix whose columns are called left singular vectors; $\Sigma = diag(\sigma_1, \sigma_2, \dots, \sigma_n)$ is an $n \times n$ diagonal matrix, whose diagonal elements are non-negative singular values sorted in descending order, and $V = [V_{ij}]$ is an $n \times n$ orthonormal matrix, whose columns are called right singular vectors. If $rank(A) = r$, then Σ satisfies:

$$\sigma_1 \geq \sigma_2 \dots \geq \sigma_r \geq \sigma_{r+1} = \cdots = \sigma_n = 0. \tag{2}$$

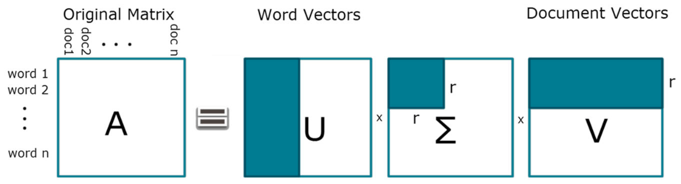

Figure 1. Singluar Value Decomposition.

Reducing the number of dimensions by applying minimal SVD decomposition significantly reduces the noise and the amount of data, memory and processing time required to obtain results with LSA. This process is called optimization dimensionality [14] and involves finding the k-th dimension (the columns that represent collections of documents) for the best k-dimensional approximation of the original matrix. Thus, the document collection is represented by a k-dimensional vector space derived through SVD.

The interpretation of applying the SVD to the terms by sentences matrix A can be performed from two different points of view. From a transformation point of view, the SVD derives a mapping between the m-dimensional space spawned by the weighted term-frequency vectors and the r-dimensional singular vector space. From a semantic point of view, the SVD derives the latent semantic structure from the document represented by matrix A. This operation reflects a breakdown of the original document into r linearly-independent base vectors or concepts, a mathematical technique also known as the Vector Space Model (VSM). Each term from the document is jointly indexed by these base vectors / concepts. A unique SVD feature is that it is capable of capturing and modeling interrelationships among terms, so that it can semantically cluster terms and sentences [15]. The idea of the VSM is to represent each document in a collection as a point in a space (a vector in a vector space). Points that are close together in this space are semantically similar, and points that are far apart are semantically distant. The user query is represented as a point in the same space as the documents. The documents are sorted in order of increasing distance (decreasing semantic similarity) from the query and then presented to the user. The most popular method for measuring the similarity of two frequency vectors (raw or weighted) is to use the *cosine similarity*. Thus, in order to rank the documents in relation to a query q, we use the cosine distance. In other words, we compute:

$$sim(q, d_i) = \frac{d_i \cdot q}{|d_i||q|}, \quad (3)$$

where q represents the vector associated to the query, and d_i represents the vector associated to the document i. In other words, the cosine of the angle between two vectors is the inner product of the vectors, after being normalized to unit length. If x and y are frequency vectors for words, a frequent word will have a long vector and a rare word will have a short vector, yet the words might be synonyms. Cosine captures the idea that the length of the vectors is irrelevant, and the important aspect is the angle between the vectors.

This step takes up a large portion of the time for computing the SVD and is the main focus of our parallel algorithm.

B. *GPU-accelerated Latent Semantic Analysis*

The main operation of the Latent Semantic Analysis, the generation of the vectorial representation of documents using the SVD technique, can be efficiently parallelized. The computational intensity can be exploited on a GPU, which contains several streaming multiprocessors, each of them containing several cores. The GPU (usually called device) has a certain amount of global memory to / from which the CPU or host thread can write / read, and which is accessible by all multiprocessors. Furthermore, each multiprocessor also contains shared memory and registers which are partitioned between the thread blocks and the threads, which run on the multiprocessor, respectively. The global memory bandwidth plays an important role since the performance of many kernels is bound by the peak global memory throughput: current GPUs have a bandwidth of up to 300 GB/s. The shared memory on the other side is a fast on-chip memory which can be accessed with similar throughput as the registers [11].

The Graphics Processing Unit (GPU) can run highly parallel algorithms or compute intensive operations more efficiently than the traditional CPU. Thus, a deployable system based on a GPU represents a much more efficient choice (in terms of cost / performance ratio) than a cluster. Due to the application-specific architecture of the GPU, harnessing its computational prowess for LSA is a challenge. Memory transfer from system memory to GPU memory (host to device and device to host) remains relatively slow and can often become a bottleneck in the applications. The effect of a slow memory transfer can be minimized by performing as many GPU based computations as possible between inter-device data transfers. Transferring large amounts of data at one time is typically faster than performing several small memory transfers.

In the following we introduce the overall workflow of the CPU-GPU based hybrid Latent Semantic Analysis implementation (fig. 2).

First, the text files, which are subsequently used to generate the associated term-document matrix, are read. Not all index terms are equally useful in representing document content. Hence, several preprocessing techniques are employed to increase the information retrieval accuracy. Because the purpose of the LSA-based information retrieval system was to be able to work with large amount of

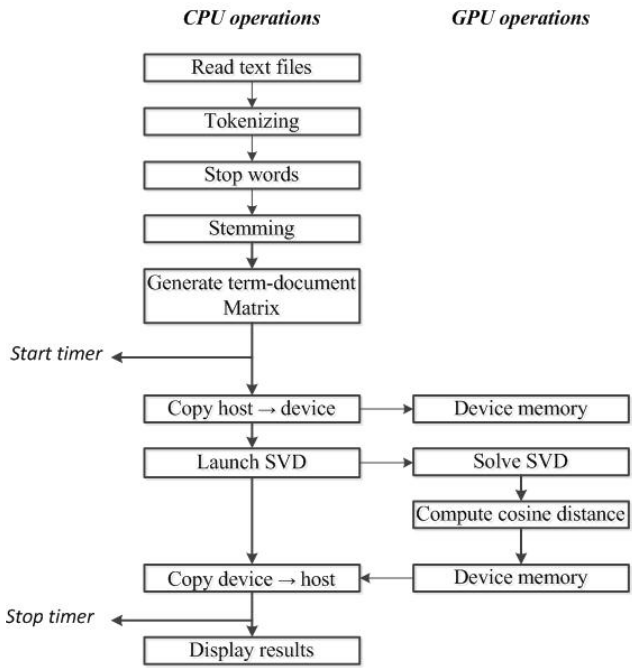

Figure 2. Workflow of the CPU-GPU LSA algorithm.

unstructured data, a tokenizing procedure was employed to remove unnecessary characters such as punctuation marks and spaces, and also to separate the stream of text into words. Furthermore, a stemming algorithm was applied to reduce all words with the same root to their base form. Finally, a predefined set of stop words [16] is removed from the input data: this reduces the amount of data to be processed, without any loss of information (since documents typically include these frequently used words, such a stop word has low or no semantic value).

Next, the host memory is being initialized with the preprocessed data, and a timer is started to determine the exact execution time of the program. If A represents the term-document matrix, only A needs to be stored in the device memory, along with a few vectors which occupy a fraction of the memory allocated for matrix A. This is advantageous, as the memory on a GPU card can become a limiting factor in large scale applications. After implementing the CPU and GPU based algorithm, we timed each implementation for various number of documents included in the analysis.

Our implementation is based on NVIDIA's cuSolver library for solving the computationally expensive operation of SVD. The cuSolver library is a high-level package based on the cuBLAS and cuSPARSE libraries. It combines three separate libraries under a single umbrella, each of which can be used independently or in concert with other toolkit libraries [17]. The intent of cuSolver is to provide useful LAPACK-like features, such as common matrix factorization and triangular solve routines for dense matrices, a sparse least-squares solver and an eigenvalue solver. In addition, cuSolver provides a new refactorization library useful for solving sequences of matrices with a shared sparsity pattern, a bidiagonalization routine, and support for SVD. The transfer of data is of consideration when using BLAS in general, and the performance of the GPU libraries specifically depends on the data placement. Initially, the input matrix A is generated on the CPU and is transferred to the GPU. CUBLAS [18] provides high performance matrix-vector, matrix-matrix multiplication, and norm computation functions. The blocking approach for bidiagonalization can be performed efficiently since CUBLAS provides efficient implementations for the matrix-vector and matrix-matrix multiplication operations even if one of the dimensions is small.

Next, the query results are copied back to the host memory. Finally, the documents are ranked based on their relevance relative to the query, and they are displayed to the user. The relevance rankings of documents in a keyword search is calculated based on the assumptions of document similarity theory [19], by comparing the deviation of angles between each vector associated to a document and the original query vector, whereas the query is represented using the same vector space model.

III. Experimental Results

To evaluate the performance of the Latent Semantic Analysis application, we used an Intel i7 8 core at 3.4 GHz, and a NVIDIA GTX Titan Black graphics card. To test the above mentioned developments, a publicly available benchmark dataset consisting of unstructured text files were used, collected from articles in various scientific domains such as bioinformatics, molecular physics, astronomy, etc. [20]. A preliminary analysis revealed that the average number of words per article is of around 330.

Next, we focus on the comparison of the CPU and the hybrid CPU-GPU based implementations of the Latent Semantic Analysis application. Computational results were perfectly identical for both versions. First, we analyze the execution time improvements for the parts that were ported to the GPU. All execution times reported below are averaged over 10 runs.

To allow for a fair comparison between the CPU and the GPU based implementations, the execution time measurements include the time required for data transfer.

Table I displays the execution time results for the SVD operation, when different number of files are used. One can observe that the GPU based implementation significantly outperforms the CPU based implementation: the results indicate a performance increase of 11 to 26 times. If more than 500 files are used, the differences in terms of executions times become even larger. This performance gap is given by the latency of the global memory operations.

Next, we analyzed the execution times for the overall workflow of the Latent Semantic Analysis implementation, when varying the number of files being processed (Table II).

For very small matrices (smaller than 600 x 600), which correspond to a very small number of text files being processed, the CPU version is faster than the GPU version. The reason for this is that the GPU based implementation requires significant constant overhead, which can comprise a large percentage of the execution time [21].

Table I. Average execution time and speed-up analysis for the single value decomposition operation, for varying number of documents.

Number of files	CPU execution time [ms]	GPU execution time [ms]	Speed up
2	14.9	1	14.9
5	162.2	6.9	23.507
10	527.8	20.1	26.258
50	11167.4	957.8	11.659
105	48047.6	3876.6	12.394
250	422106	24908.6	16.946
500	2403120	101930.4	23.576

Table II. Average execution time and speed-up analysis for the overall algorithm, for varying number of documents.

Number of files	CPU execution time [ms]	GPU execution time [ms]	Speed up
2	18,444	469,7	0,039
5	180,111	499,1	0,361
10	603	514,5	1,172
50	13685,444	1611,2	8,494
105	58265,875	4693,8	12,413
250	505496	25708,7	19,662
500	2278157,331	107019	21,287

Table III. Percentage of the total execution time occupied by the SVD operation.

Number of files	Percent SVD exec. time/Overall exec. time [%]
2	5,422
5	8,637
10	16,123
50	53,667
105	71,433
250	86,757
500	91,381

When the matrix size increases, the percentage of the total time occupied by the overhead operations gradually decreases. The overhead time does not change, but the amount of computations being performed significantly increases. As matrix sizes grow, the total time that SVD requires will also increase, rendering tge GPU version of the routine highly efficient, as can be observed in Table III.

IV. Conclusions

We have developed a parallel latent semantic analysis algorithm for a GPU based implementation. The results of our tests are very promising. The speed-up of the GPU based algorithm increases up to 21 for a number of documents of up to 500. The GPU based SVD algorithm efficiently exploits the parallelism of the GPU architecture and achieves high computing performance.

With the GPU being widespread in modern PCs, the algorithm is not limited to expensive custom ordered workstations. Most mid range computers currently come with a graphics card, including laptops. This makes it possible to perform GPU-based LSA on mobile computers. The bidiagonalization of the term-document matrix is performed entirely on the GPU, using the highly optimized cuSolver library for obtaining maximum performance.

Future work will focus on developing an LSA implementation for a multi-GPU hardware configuration. Currently, to use multiple GPUs the algorithm must explicitly state how to distribute the workload to each device. There is no automatic optimization allowing the use of multiple GPUs, and the implementation needs to be modified for each individual configuration / application. The use of multiple GPUs would be highly beneficial, as many computers are being released with multiple graphics cards built in.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 600932.

This work is supported by the program Partnerships in Priority Domains (PN II), financed by ANCS, CNDI - UEFISCDI, under the project nr. 130/2012.

REFERENCES

[1] N. Adams, G. Blunt, D. Hand, and M. Kelly, "Data mining for fun and profit", Statistical Science, 15(2):111-131, 2000.

[2] H. P. Edmundson, "New Methods in Automatic Extracting", Journal of the Association for Computing Machinery, 16(2):264-228.

[3] J. Kupiec, J. Pedersen, F. Chen, "A Trainable Document Summarizer", Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, United States 1995, pp. 68-73.

[4] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval", Information Processing & Management, 24(5):513-523, 1988.

[5] R. Barzilay, M. Elhadad, "Using Lexical Chains for Text Summarizer", Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, United States, 1995, pp. 68-73.

[6] S. Dumais, G. Furnas, T. Lanerwester, R. Harshmandauer, S. Deerwester, R. Harshman, "Using latent semantic analysis to improve access to textual information", Proceedings of the the SIGCHI conference on Human factors in computing systems, Washington, United States, 1988.

[7] M. Berry, "Large-scale sparse singluar value computations", The International Journal of Supercomputer Applications, 6(1):13-49, 1992.

[8] C. Lanczos, "An iteration method for the solution of th eeigenvalue problem of linear differential and integral operators", J. Res. Natl. Bureau Stand., 45(1):255-282, 1950.

[9] C. Paige, N. Parlett, H. V. der Vorst, "Approximate solutions and eigenvalue bounds from Krylov subspaces", Numerical Linear Algebra with applications, 2(2):115-134, 1995.

[10] A. Gee, "Research into GPU accelerated pattern matching for applications in computer security", Univ. of Canterbury, Christchurch, New Zealand, 1987.

[11] D. Kirk, and W.M. Hwu, "Programming Massively Parallel Processors: A Hands-on Approach", London: Elsevier, 2010.

[12] NVIDIA Corporation, "CUDA, Compute unified device architecture, Programming Guide v5.5", 2013.

[13] T. K. Landauer, P. W. Foltz, D. Laham, "Introduction to Latent Semantic Analyis", Discourse Processes, no. 25, pp. 259-284, 1998.

[14] T. K. Launder, S. T. Dumais, "Latent Semantic Analysis", Scholarpedia, vol 3(11), pp. 43-56, 2008.

[15] J. Steinberger, K. Jezek, "Using Latent Semantic Analysis in Text Summarization and Summary Evaluation", Proceedings of ISIM '04, pages 93-100, 2004.

[16] *** "List of English Stop Words", 2009. [Online]. Available: http://xpo6.com/list-of-english-stop-words//

[17] *** "cuSOLVER Library", 2015. [Online]. Available: https://developer.nvidia.com/cusolver/

[18] NVIDIA Corporation, "NVIDIA CUBLAS Library", 2007. [Online]. Available:https://developer.download.nvidia.com/compute/1_1/CUBLAS_Library_1.1.pdf.

[19] P. D. Turnley, P. Pantel, "From Frequency to Meaning: Vector Space Models of Semantics", Journal of artificial intelligence research, vol. 37, pp. 141–188, January 2010.

[20] *** "University of California, Irvine Machine Learning Repository", 2015. [Online]. Available: https:// archive.ics.uci.edu/ml/datasets.html.

[21] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. Lefohn, T. Purcell, "A survey of general-purpose computation on graphics hardware", Computer Graphics Forum. 26(1):80-113, 2007.