SVD BASED LATENT SEMANTIC INDEXING WITH USE OF THE GPU COMPUTATIONS

Raczyński Damian and Stanisławski Włodzimierz

University of Applied Science in Nysa, Poland

ABSTRACT

The purpose of this article is to determine the usefulness of the Graphics Processing Unit (GPU) calculations used to implement the Latent Semantic Indexing (LSI) reduction of the TERM-BY-DOCUMENT matrix. Considered reduction of the matrix is based on the use of the SVD (Singular Value Decomposition) decomposition. A high computational complexity of the SVD decomposition - $O(n^3)$, causes that a reduction of a large indexing structure is a difficult task. In this article there is a comparison of the time complexity and accuracy of the algorithms implemented for two different environments. The first environment is associated with the CPU and MATLAB R2011a. The second environment is related to graphics processors and the CULA library. The calculations were carried out on generally available benchmark matrices, which were combined to achieve the resulting matrix of high size. For both considered environments computations were performed for double and single precision data.

KEYWORDS

LSI, GPU, reduction, parallel computing, SVD

1. INTRODUCTION

Nowadays, there is a tremendous increase in the number of text resources associated with various themes. The most obvious example of this phenomenon is still growing the World Wide Web. With the increase in the number of text documents placed in various databases increasingly important are the methods of automatic documents indexing. The LSI method, introduced in 1990, uses the theory of linear algebra to automate the process of indexing and retrieval [1]. This method is widely used for the purpose of information retrieval systems, for example:

- in [1] for indexing of medical (1033 documents) and information science (1460 documents) abstracts,
- in [2] for indexing collection of 382845 documents divided into 9 subcollections,
- in [3] for indexing patent documents,
- in [4] for cross language information retrieval (CLIR), more precisely for the Malay-English CLIR system,
- in [5] for the Greek-English CLIR system.

Furthermore, application of the LSI can also be found in the field of ontology mapping [6] or even genomic studies [7]. The LSI offers up to 30% better performance than traditional lexical techniques [8].
Despite a number of advantages, the LSI method is computationally problematic - implementation of the algorithm requires the use of equipment with significant computing power. In particular, this drawback is visible in a case of databases of text documents with a huge

number of positions (expressed in thousands), where there is a need for continuous updating. In relation to such large structures, realization of search system, based on the LSI method is a difficult task to perform. The time analysis of the LSI method, applied to a large number of documents, can be found in a relatively small number of papers. Referring to [9], application of the LSI method for a matrix of size 11,152×3891 (43392432 elements) required calculations that lasted $2.3*10^5$ seconds. We used in this article a matrix with a similar number of elements (matrix of size 5896×7095 – 41832120 elements). We also used the Matlab environment to determine the computation time for the CPU environment. The results are compared to computations using the GPU environment. The authors would like to point out that in literature known to us nobody made a similar comparison. Taking into account the computational power of modern graphics cards, such a comparison is desirable. Usually the computing power of the commercial GPUs is much higher for the single precision data than for the double precision. In the literature known to us there is not a study to determine the usefulness of the LSI method used for the single precision data. Therefore the authors carried out a comparison of the time complexity and correctness of the results obtained using the single precision data.

The results are promising. The use of the GPU in place of the CPU from the similar market segment has allowed to reduce the computation time almost by half. Additionally, changing the data format from the double precision to the single precision allowed to further reduce of the computation time. Moreover, regardless of the used environment and the data format, for the considered case, precision of the search system remained at a similar level. Despite the fact that the metodology proposed in this article allows several times to reduce the computation time, we realize that the use of the GPUs to LSI method has its limitations. Application of LSI for indexing of huge documents collection may be achievable with the combine of decompositotinal techniques and parallel computing using the computer cluster with the GPUs.

2. VECTOR SPACE MODEL

The Vector Space Model in the information retrieval is a widely used technique [10,11], in which both documents and queries are represented by vectors. Each element of the vector represents the weight of a given expression in a given document or query.

The first step of building the Vector Space Model is to identify occurrences of keywords in the entire collection of documents. Selection of the appropriate list significantly affect the performance of a retrieval system. In the next step the index structure is being built. Each keyword is mapped in a multidimensional space in such a way to make correlation of all documents containing that phrase with the same phrase. The result is the TERM-BY-DOCUMENT matrix, which contain weighted connections between the documents and the concepts.

There are several methods of selection factors lie at the intersection of the columns and rows of the TERM-BY-DOCUMENT matrix. The most commonly used are:

• Boolean model – the occurrence of concepts in the document is marked by one, the lack of concepts is marked by zero,
• Term Frequency model (TF) – the values contained in the matrix specifies the number of occurrences of concept in a document,
• Term Frequency – Inverse Document Frequency model (TF-IDF) – the model takes into account both – the frequency of occurrence of a concept in a document and its power of discrimination (measure of the prevalence of concept).

In the space of concepts (term space) each document creates a vector while each concept (term) represents dimension. In the documents space, terms are represented by vectors located between the axes defined by the documents.

	d1	d2	d3
t1	0	4	0
t2	0	3	2
t3	6	0	0

Figure 1. The sample TERM-BY-DOCUMENT matrix

The idea of both spaces, for example TERM-BY-DOCUMENT matrix in fig. 1, is shown in fig. 2.

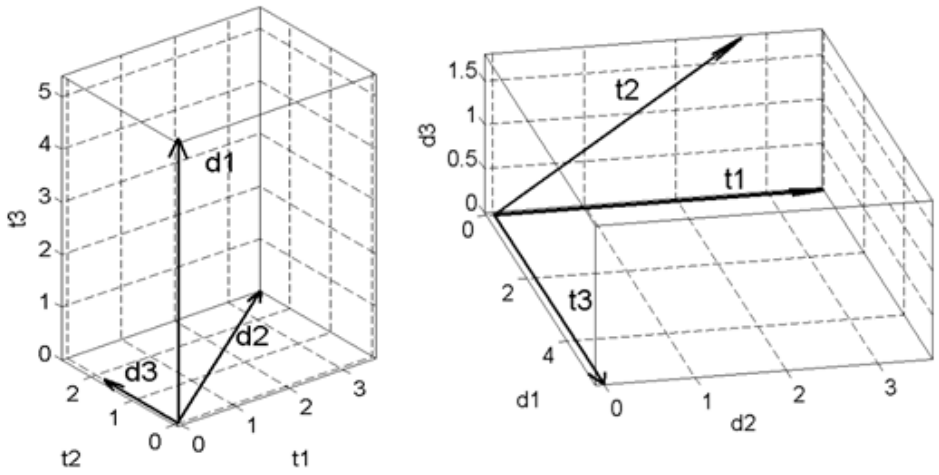

Figure 2. Terms' space (left part) and Document's space (right part) of the sample TERM-BY-DOCUMENT matrix

The two most commonly used measures of similarity between two vectors are the scalar product (1) and the cosine distance (2).

$$\mathbf{sim}\,(\mathbf{a},\mathbf{b}) = \mathbf{a}^{\mathbf{T}}\mathbf{b} = \sum_{\mathbf{i}=1}^{\mathbf{m}} \mathbf{a}_{\mathbf{i}}\mathbf{b}_{\mathbf{i}} \tag{1}$$

$$\mathbf{sim}\,(\mathbf{a},\mathbf{b}) = \frac{\mathbf{a}^{\mathbf{T}}\mathbf{b}}{|\mathbf{a}|\cdot|\mathbf{b}|} = \frac{\sum_{\mathbf{i}=1}^{\mathbf{m}} \mathbf{a}_{\mathbf{i}}\mathbf{b}_{\mathbf{i}}}{\sqrt{\sum_{\mathbf{i}=1}^{\mathbf{m}} \mathbf{a}_{\mathbf{i}}^{2}} \cdot \sqrt{\sum_{\mathbf{i}=1}^{\mathbf{m}} \mathbf{b}_{\mathbf{i}}^{2}}} \tag{2}$$

In practice, frequently used is a cosine distance. The smaller angle between two vectors means closer similarity. The value of cosine distance varies between -1 (180 degrees – vectors not similar) to 1 (0 or 360 degrees – a hundred percent similarity).

3. SVD DECOMPOSITION AND REDUCTION

The high-order matrix is decomposed using SVD transformation into product of three matrices (3).

$$\mathbf{A}_{\mathbf{mn}} = \mathbf{U}_{\mathbf{mm}}\,\mathbf{S}_{\mathbf{mn}}\,\mathbf{V}^{\mathbf{T}}_{\mathbf{nn}} \tag{3}$$

where:

The columns of the matrix $\mathbf{U}_{mm}$ are orthonormal eigenvectors of the matrix product $\mathbf{A}_{mn}\mathbf{A}_{mn}^{T}$. The columns of the matrix $\mathbf{V}_{nn}$ are orthonormal eigenvectors of the matrix product $\mathbf{A}_{mn}^{T}\mathbf{A}_{mn}$. The matrix $\mathbf{S}_{mn}$ is a diagonal matrix containing the square roots of the eigenvalues (singular values) of the product of matrices $\mathbf{A}_{mn}\mathbf{A}_{mn}^{T}$ and $\mathbf{A}_{mn}^{T}\mathbf{A}_{mn}$. The diagonal values of $\mathbf{S}_{mn}$ are sorted in non-increasing order.

In order to determine the low rank approximation of the original TERM-BY-DOCUMENT matrix, it is possible to use equation (4).

$$\mathbf{A}_{mn} \approx \hat{\mathbf{A}}_{mn} = \mathbf{U}_{mk}\,\mathbf{S}_{kk}\,\mathbf{V}_{nk}^{T} \tag{4}$$

The low rank representation of TERM-BY-DOCUMENT matrix is obtained by removing the last (m-k) and (n-k) columns of a matrices $\mathbf{U}_{mm}$ and $\mathbf{V}_{nn}$ respectively and by retain only the k largest singular values of the matrix $\mathbf{S}_{mn}$. As shown in [8], the matrix $\hat{\mathbf{A}}_{mn}$ is the best approximation of the matrix $\mathbf{A}_{mn}$ with respect to the Frobenius norm (5).

$$\left\|\mathbf{A}_{mn}\right\|_{F} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^{2}} \tag{5}$$

The approximation error can be determined from (6).

$$\left\|\mathbf{A}_{mn} - \hat{\mathbf{A}}_{mn}\right\|_{F} = \sqrt{\sigma_{k+1}^{2} + \ldots + \sigma_{\min(m,\,n)}^{2}} \tag{6}$$

where σ denotes the removed singular values.

In order to reduce size of the original TERM-BY-DOCUMENT matrix, it must be mapped into the space of a smaller number of dimensions. The linear transformation (7)

$$\begin{aligned} \mathbf{L}: \quad \mathbf{R}^{m} &\rightarrow \mathbf{R}^{k} \\ \mathbf{x} &\mapsto \mathbf{S}_{kk}^{-1}\mathbf{U}_{mk}^{T}\mathbf{x} \end{aligned} \tag{7}$$

is used to transform $\mathbf{A}_{mn}$ into (8).

$$\begin{aligned} \mathbf{A}_{mn} &\mapsto \mathbf{S}_{kk}^{-1}\mathbf{U}_{mk}^{T}\mathbf{A}_{mn} \\ &= \mathbf{S}_{kk}^{-1}\mathbf{U}_{mk}^{T}\mathbf{U}_{mm}\mathbf{S}_{mn}\mathbf{V}_{nn}^{T} \\ &= \mathbf{S}_{kk}^{-1}\left[\mathbf{I}_{kk} \mid \varnothing_{k(m-k)}\right]\mathbf{S}_{mn}\mathbf{V}_{nn}^{T} \\ &= \mathbf{S}_{kk}^{-1}\left[\mathbf{S}_{kk} \mid \varnothing_{k(n-k)}\right]\mathbf{V}_{nn}^{T} \\ &= \left[\mathbf{I}_{kk} \mid \varnothing_{k(n-k)}\right]\mathbf{V}_{nn}^{T} \\ &= \mathbf{V}_{nk}^{T} \end{aligned} \tag{8}$$

After the reduction of multidimensional space, linked documents are closer to each other. The relative distance between points in the reduced space represents the semantic similarity between documents.

Mapping of the user questions to the reduced vector space is done by (9)

$$\mathbf{q} \mapsto \mathbf{S}_{kk}^{-1}\mathbf{U}_{mk}^{T}\mathbf{q} \tag{9}$$

where **q** denotes pseudo-document (keywords entered by the user build the vector in the original space).

4. THE EXPERIMENT

This chapter provides information about the data and equipment used to perform the calculations. The results of the study are also included here. The chapter also includes the results of the experiment.

4.1. Data

The study, which is presented in this article, was made on generally available benchmark including connected sets of documents:

- CACM: 3204 documents,
- CISI: 1460 documents,
- CRAN: 1398 documents,
- MED: 1033 documents.

The TERM-BY-DOCUMENT matrix is organized in such a way that its columns represent the documents from collection while rows represent words. The size of the matrix is 5896 rows per 7095 columns. The structure of the original TERM-BY-DOCUMENT matrix is shown in the fig. 3.

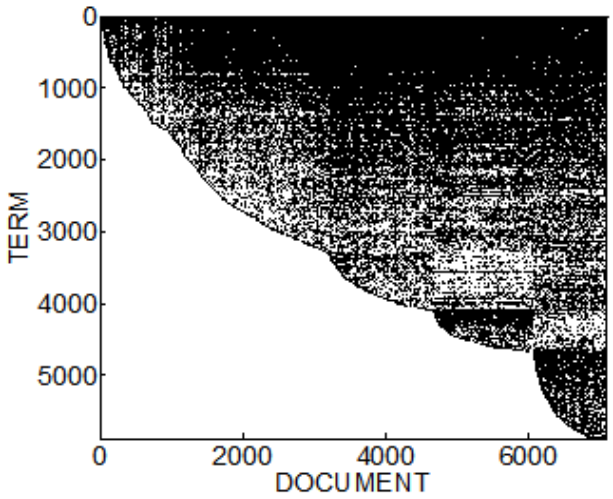

Figure 3. The structure of the TERM-BY-DOCUMENT matrix
This is a sparse TF matrix with 247,157 elements different form zero.

4.2. Implementation and hardware

The LSI algorithm has been developed for two parallel computing environments to compare the time and the accuracy of the calculations. The first environment – related to CPU – MATLAB R2011a, which use the Intel Math Library version 10.2, which supports multi-core and data parallelism [12]. The second environment – related to GPU – developed programs in C++ language with use of CUDA, CULA [13] and CULYA [14] libraries.

The CULYA library (described in details in [15]) uses the CULA, CUBLAS and CUDA libraries to provide basic linear algebra operations for vectors and matrices. The library includes 115 methods and overloaded operators (e.g. +,-,*,^), which makes operations on matrices more intuitive. The table 1 presents the description of the selected methods of the CulyaMatrix class, which is the main class of the library.

Table 1. The selected methods of the CulyaMatrix class

Method	**Description**
to_card	copies the matrix from the main memory to the graphics card memory
from_card	copies the matrix from the graphics card memory to the main memory
del_dev	removes the matrix from the graphics card memory
del_host	removes the matrix from the main memory
operator *	matrices product
operator +	matrices addition
operator -	matrices subtraction
operator ++(int)	returns matrix transpose
operator ++	transposes matrix, overwriting the source data
dup	returns a copy of the matrix
inv	matrix inversion
qr_full	the QR decomposition of the matrix. The method returns an object of class Tqr containing matrices R and Q of the decomposition
eig_full(char vector)	the eigenvalues and the eigenvectors of the matrix. The method returns an object of class Teig containing the eigenvalues vector, the right and the left eigenvectors
svd(char param)	the SVD decomposition. The method returns an object of class TSvd containing the singular values vector, the U and VT matrices.

The difference in the architecture of conventional processors and GPUs mainly based on the number of cores and cache size (fig. 4).

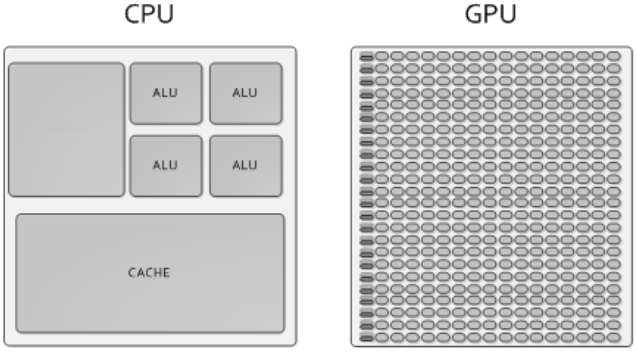

Figure 4. Comparison of CPU and GPU architectures

The GPUs have large number of cores, performing basic arithmetic operations and small cache. This architecture is very effective to process the matrix/block data. The basic unit of code that is responsible for performing computations using GPU is the CUDA kernel. The structure of a simple program that performs addition of two vectors, using the computing power of the graphics card, is presented below:

```
1.  __global__ void add(double*a, double*b, double*c)
2.  {
3.    int tid=threadIdx.x+blockIdx.x*blockDim.x;
4.    if(tid<N)
5.         c[tid]=a[tid]+b[tid];
6.  }
7.
8.  …
9.
10. double vector1[N];
11. double vector2[N];
12. double vector3[N];
13. double *vect1;
14. double *vect2;
15. double *vect3;
16. for (long int i=0; i<N; i++)
17. {
18.   vector1[i]=i;
19.   vector2[i]=N-i;
20. }
21. cudaMalloc((void**)&vect1, N*sizeof(double));
22. cudaMalloc((void**)&vect2, N*sizeof(double));
23. cudaMalloc((void**)&vect3, N*sizeof(double));
24. cudaMemcpy(vect1,      vector1,      N*sizeof(double),
    cudaMemcpyHostToDevice);
25. cudaMemcpy(vect2,      vector2,      N*sizeof(double),
    cudaMemcpyHostToDevice);
26. add<<<(N+127)/128, 128>>>(vect1, vect2, vect3);
    cudaMemcpy(vector3,            vect3,N*sizeof(double),
27. cudaMemcpyDeviceToHost);
28. cudaFree(vect1);
29. cudaFree(vect2);
30. cudaFree(vect3);
```

In the lines 1 – 6 there is the CUDA kernel code, which will be run on the GPU. The declaration of the function is similar to the C language, however, it must be preceded by the __global__ directive. The function cannot return a value via the stack (void). Despite that the CUDA kernel will perform the operation of addition of two vectors, the code does not contain any software loop that would occur with a classic solution to this problem. The CUDA computing environment automatically allocates the GPU computing resources, for each data element, calling kernel code many times simultaneously.

After starting the application on the GPU environment, execution of the CUDA kernel is equivalent to running multiple threads simultaneously, each of which performs the same operation (SIMT architecture – Single Instruction Multiple Thread). The threads are combined into groups called threads blocks. The threads blocks are grouped in grids, each grid contains

only blocks belonging to the same CUDA kernel. Each block and each thread has its identifier that uniquely identifies their location respectively in the block and grid. Each running thread has a structure that determine the coordinates of its block in the grid – blockIdx, and coordinates of the thread in the block – threadIdx. The CUDA kernel code refers to the specific coordinates by field componens x,y,z. Threads have also ability to determine the dimension of the block and the grid with use of the structure blockDim and gridDim.

The application developer has a direct impact on the division of computational problem for a certain number of threads and blocks. This is done by specifying two parameters contained in brackets <<<…>>> in place of the CUDA kernel invocation. The first value determines the segmentation of computational problem into a certain number of blocks. The second value determines the number of threads running in each block. In the fig. 5 there is an example of division of computational problem into 4 blocks of 5 threads each.

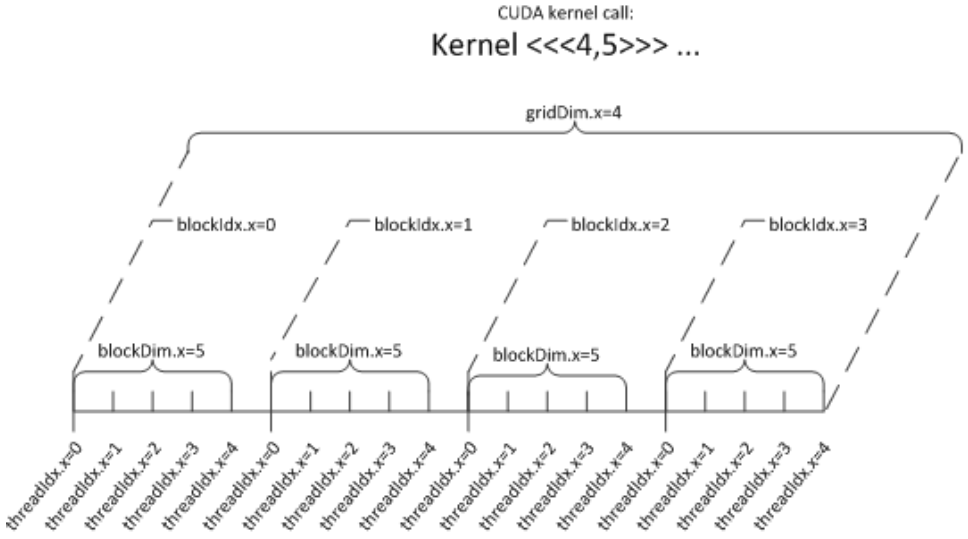

Figure 5. Example of division of computational problem

It should also be mentioned, that the modern x86 processors have a whole range of solutions that affect the calculations time. The first important solution accelerating the calculations is the multi-core architecture. However the number of cores in the CPU is not as large as the number of cores in the GPU (for example, the unit used in our case – Core i7 3770 has 4 cores). Another element worthy of mention are vector extensions supporting matrix computations [16] (also used by MATLAB R2011a). The comparison of the used equipment is summarized in table 2.

Table 2. Comparison of used equipment

UNIT	**CPU**	**GPU**
MODEL	Intel Core i7 3770	Nvidia GTX580
NUMBER OF CORES	4 + AVX extensions	512 CUDA cores
MEMORY	16GB DDR3 (host memory)	4GB GDDR5

4.3. The reduction error

As a reduction error, a difference between the cosine distance of vectors of the original and the reduced TERM-BY-DOCUMENT matrix is assumed. The sample error maps for the case of reduction to k=100 rows, for double precision calculations, for the MATLAB and GPU environments, are shown in fig. 6.

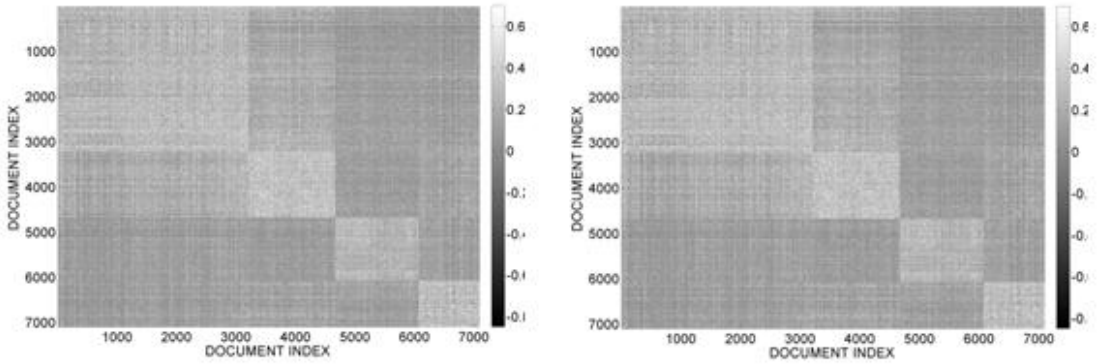

Figure 6. Reduction error maps for k=100 (MATLAB – left part, GPU – right part)

In order to present reduction error for the all studied cases, the mean square error was used (10).

$$MSE = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(COS\_ORG_{ij} - COS\_RED_{ij}\right)^2 \qquad (10)$$

where:

n=7095, COS_ORG_{ij} - the cosine distance between the i'th and j'th documents in original TERM-BY-DOCUMENT matrix, COS_RED_{ij} - the cosine distance between the i'th and j'th documents in reduced TERM-BY-DOCUMENT matrix. The MSE errors are shown in fig. 7.

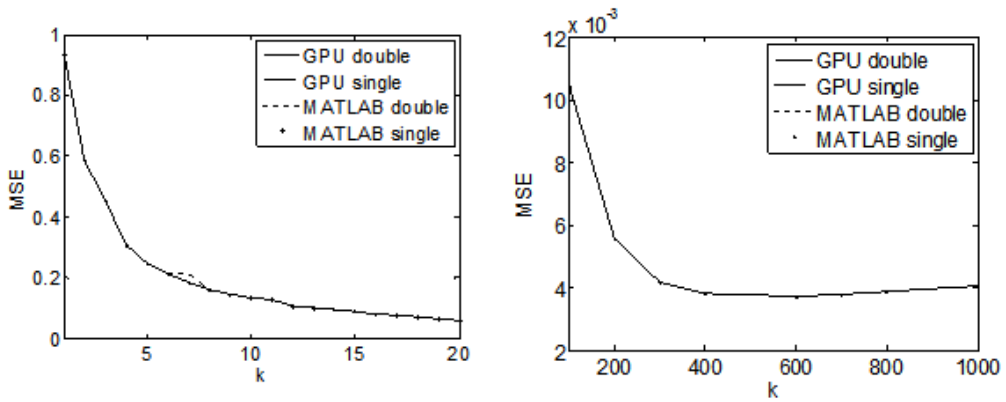

Figure 7. The MSE errors of reduction

The results are consistent with initial expectations. The error decreases with increase of the size of the reduced TERM-BY-DOCUMENT matrix. It should be noted, however, that the results determine only the similarity of the reduced matrix relative to the original. This measure does not take into account the positive effects of the application of the LSI method. The LSI method is able to detect the existing relationship between the semantic content of various documents.

Search systems based on the LSI are able to return a list of documents related to terms of a user's query, despite the difference in keywords used by him.

4.4. Reduction time

The reduction times, depend on environment and precision, are shown in fig. 8. Due to the fact that the main operation, which affect the computation time is the SVD decomposition (which takes the same time in each case), the results are given for the reduction for k=100 rows.

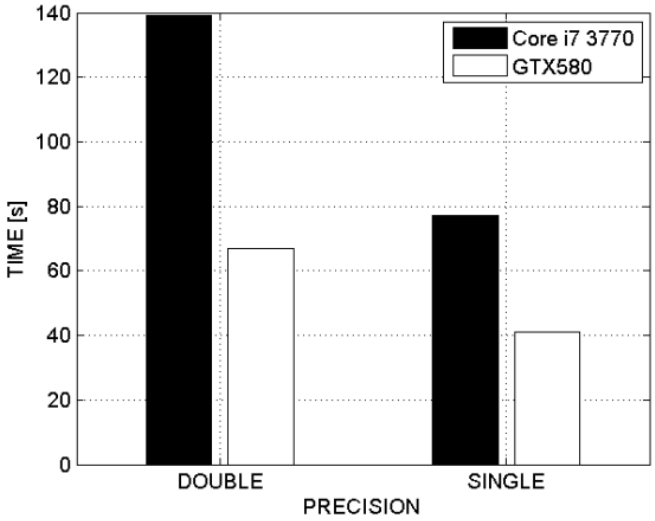

Figure 8. The reduction time depending on the environment and precision

The times of search the most similar document, according to the cosine distance, depending on the size of the TERM-BY-DOCUMENT matrix are shown in fig. 9.

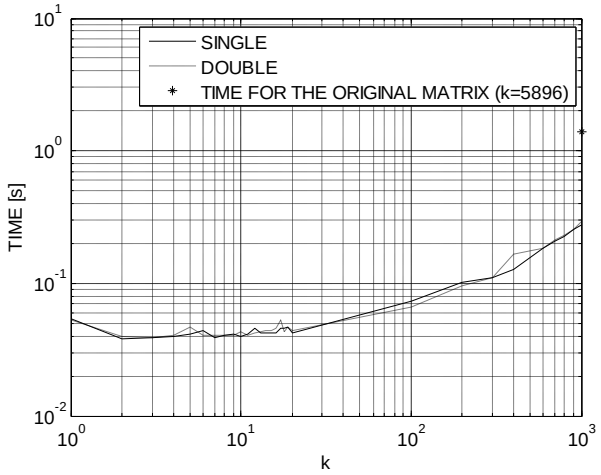

Figure 9. The times of search the most similar document, depending on the rank of the TERM-BY-DOCUMENT matrix

Summarizing the results, the calculation of the reduced TERM-BY-DOCUMENT matrix using the GPU environment brings a reduction in calculation time by almost half compared to the time obtained for the CPU environment (fig. 8). In addition, the search system using the reduced TERM-BY-DOCUMENT matrix in place of the original one is able to find the closest match document (considering the cosine distance) in time which is only a fraction of the original searching time (fig. 9).

4.5. Search engine precision

The final evaluation of the used environments has been determined in the following way. For the 100 randomly generated user's query vectors (the number of non-zero elements in the vector equals 5, the value of each non-zero element varies from 1 to maximum value of the original TERM-BY-DOCUMENT matrix) the most matched document (based on the cosine distance) are determined from the original TERM-BY-DOCUMENTS matrix. Then, for the same query vector, p (where p=1, 10, 20, 30 …, 100) the most matched documents in the reduced TERM-BY-DOCUMENT matrix are determined. The precision is computed as a percentage of the same documents in both collections. The obtained results are shown in fig. 10-12.

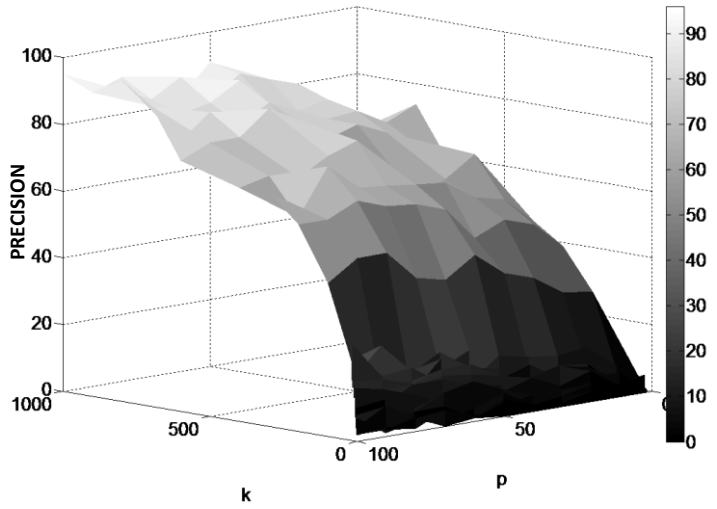

Fig. 10. Precision for reduced matrix obtained in MATLAB environment (double precision)

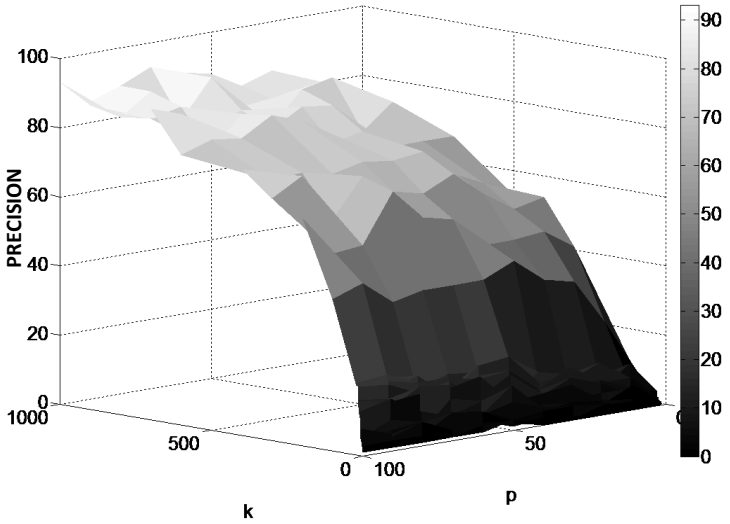

Fig.11. Precision for reduced matrix obtained in GPU environment (double precision)

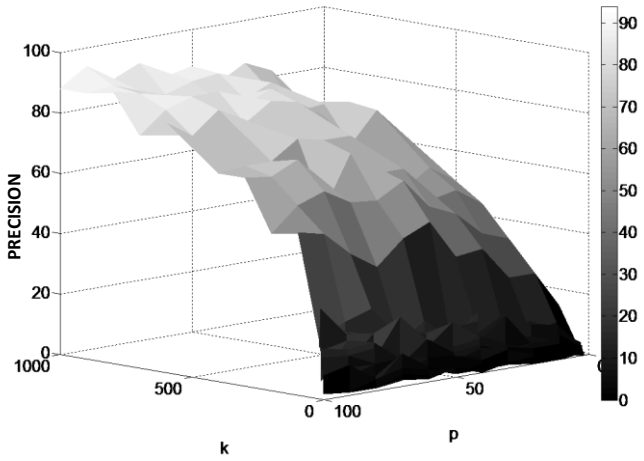

Fig. 12. Precision for reduced matrix obtained in GPU environment (single precision)

The charts show that regardless of the used computational environment, the precision associated with searching for the most similar documents (relative to the set obtained for the original TERM-BY-DOCUMENT matrix) is at a comparable level. What is interesting, the graphs show also, that change the arithmetic form the double precision to single precision (for calculations on the GPU environment) does not make a drastic deterioration of the results. Considering the fact that the calculation time for the single precision arithmetic is faster, this suggests the possibility of using only calculations using the single precision values in a target retrieval system.

5. CONCLUSIONS

The LSI method based on the SVD decomposition is an effective way to reduce the size of the original TERM-BY-DOCUMENT matrix. Referring to the literature, the method is frequently applied to a collection of homogeneous documents. Applying it to large collections of heterogeneous documents is problematic mainly due to long time of computation. This is caused by the high computational complexity of the SVD decomposition - $O(n^3)$. One step towards solving this problem could be the use of parallel computing, in this article, the authors focused on the application of the graphics processor. In the literature, up to now, there was no comparison of the efficiency and precision of the two possible computing environments – associated with the CPU and GPU, for the purpose of the LSI method.

The article shows that the use of GPU can reduce time needed to reduce TERM-BY-DOCUMENT matrix – application of the presented equipment allowed reducing computational time by almost half (comparing the time obtained in the GPU environment to the time for the CPU). The precision obtained in both environments was similar. We should also clearly indicate that despite the fact that we do not use the most modern equipment available on the market (mostly for financial reason), both devices were released at the similar time (Core i7 3770 – Q2'12, GTX580 – Q4'2010) and the two devices belong to the similar segment of the market.
The presented methodology allows almost by half to reduce the computation time, but it does not give the possibility of reducing a huge index structures, where the number of documents is counted in the hundreds of thousands. Our future experiments will focus on the different decomposition methods of TERM-BY-DOCUMENT matrix before the reduction (like K-means and Epsilon decomposition) and the use of the Krylov methods to replace the original SVD decomposition. This will allow for multiple increase of the size of considered indexing structures. The decomposition of one large computational problem into a number of smaller has a number of advantages. The first is associated with a reduction of the computational complexity. Assuming that the original TERM-BY-DOCUMENT matrix will be divided into k parts of equal

size, we get the computational complexity equals k*O((n/k)^3) from the original equals O(n^3). Moreover, the obtained matrices from the decomposition process, could be reduced simultaneously by using parallel computer cluster equipped with the GPU's.

REFERENCES

[1] Deerwester S., Dumais S. T., Furnas G. W., Landeuer T. K. & Harshman R., (1990) „Indexing by latent semantic analysis", Journal of the American Society for Infomation Science Vol. 41, No. 6, pp. 391-407.

[2] Dumais S. T., (1992) „LSI meets TREC: A status report", In: D. Harman (Ed.), The First Text Retrieval Conference (TREC1), National Institute of Standards and Technology Special Publication, pp. 137-152.

[3] Moldovan A., Boţ R. I. & Wanka G., (2005) „Latent Semantic Indexing for patent documents", International Journal of Applied Mathematics and Computer Science, Vol. 15, No. 4, pp. 551-560.

[4] Abdullah M. T., Ahmad F., Mahmod R. & Sembor T. M., (2003) „Application of latent semantic indexing on Malay-English cross language information retrieval," DigitalLibraries: Technology And Management Of Indigenous Knowledge For Global Access Lecture Notes In Computer Science, 2911, pp. 663-665.

[5] Berry M. W. & Young P. G., (1995) „Using latent semantic indexing for multilanguage information retrieval", Computers and the Humanities, Vol. 29, No. 6, pp. 413-429.

[6] Kotis K., Vouros G. A. & Stergiou K., (2004) "Capturing semantics towards automatic coordination of domain ontologies", Lecture Notes in Computer Science, 3192, pp. 22-32.

[7] Homayouni R., Heinrich K., Wei L. & Berry M. W., (2005) „Gene clustering by Latent Semantic Indexing of MEDLINE abstracts", Bioinformatics, Vol. 21, No. 1, pp. 104-115.

[8] Dumais S. T., (1991) „Improving the retrieval of information from external sources", Behavior Reaserch Methods Instruments&Computers, Vol. 23, No. 2, pp. 229-236.

[9] Gao J. & Zhang J., (2005) „Clustered SVD strategies in latent semantic indexing", Information Processing and Managament 41, pp. 1051-1063.

[10] Berry M. W., Drmac Z. & Jessup E. R., (1999) „Matrices, Vector Spaces, and Information Retrieval", SIAM Review, Vol. 41, No. 2, pp. 335-362.

[11] Manning C. D. & Schütze H., (1999) „Topics in In formation Retrieval", Foundations of Statistical Natural Language Processing, Massachusetts, The MIT Press, pp. 529-574.

[12] Intel Corporation, (2009) "Intel Math Kernel Library (Intel MKL) 10.2 In-Depth", Retrieved from http://software.intel.com/sites/products/collateral/hpc/mkl/mkl_indepth.pdf.

[13] Raczyński D., (2014) „Matrix computations using GPU", Contemporary Approaches to Design and Evalutionary of Information Systems, Wrocław, 2014.

[14] Raczyński D. & Stanisławski W., (2012) „Controllability and observability gramians parallel computation using GPU," Journal of Theoretical and Applied Computer Science, Vol. 6, No. 1, pp. 47-66.

[15] Raczyński D., (2014) „Redukcja modeli obiektów sterowania z zastosowaniem obliczeń równoległych na przykładzie modelu kotła energetycznego [Model order reduction of control

systems with use fo the parallel computing on example of the power boiler]", PhD thesis, Faculty of Electrical Engineering, Automatic Control and Informatics, Opole University of Technology, Opole.

[16] Stanisławski W. & Raczyński D., (2009) "Programowanie jednostek wektorowych procesorów rodziny x86 [Programming of the vector units of the x86 processors family]", Oficyna Wydawnicza Politechniki Opolskiej, Opole.

Authors

Damian Raczyński - Academic Lecturer at the University of Applied Science in Nysa, Poland. The PhD degree in technical sciences received in year 2014 from Faculty of Electrical Engineering, Automatic Control and Informatics, Opole University of Technology, Poland. His main research interests are parallel and distributed computing, reduction of complex models using the SVD methods.

Włodzimierz Stanisławski - Professor in the discipline of automatics controlAnd robotics. Academic Lecturer at the University of Applied Science in Nysa, Poland. His main research interests are modelling and computer simulation.