Łukasz KORDUŁA, Maciej WIELGOSZ, Michał KARWATOWSKI, Marcin PIETROŃ, Dominik ŻUREK, Kazimierz WIATR
AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY
30 Mickiewicza Av., 30-059 Cracow, Poland

Assessment of various GPU acceleration strategies in text categorization processing flow

Abstract

Automatic text categorization presents many difficulties. Modern algorithms are getting better in extracting meaningful information from human language. However, they often significantly increase complexity of computations. This increased demand for computational capabilities can be facilitated by the usage of hardware accelerators like general purpose graphic cards. In this paper we present a full processing flow for document categorization system. Gram-Schmidt process signatures calculation up to 12 fold decrease in computing time of system components.

Keywords: GPU, NLP, text categorization, OpenCL.

1. Introduction

Document categorization is considered to be one of the most common applications in a field of Natural Language Processing (NLP). NLP is interdisciplinary field connected with computer science, machine learning and artificial intelligence. The main aim of NLP is to communicate with machines in human natural language and how computers could retrieve data without human supervision. This paper will be considering the part of NLP connected with natural language understanding, especially automatic document categorization. Categorization using human resources can be very time consuming or even impossible in a reasonable time, as it can incorporate complicated calculations [1].

Significant factor facilitating presented approach is the fact that we are only interested in general topic of given documents. We are not interested in grammar and sophisticated words, therefore we can simplify the text to contain only raw information. Text simplification is one of basics techniques to improve language processing applications. Natural languages are very complex and it introduces many obstacles for machines. One of simplification techniques is lemmatization. It brings different forms of the same word to the canonical form of a word. This operation is safe, as we do not lose much information comparing to original words, while decreasing complexity of a sentence. In natural languages there are also words that do not provide much or any information. Such words can be omitted without observable impact on the meaning of the sentence. These words are called stop-words or stop-list.

2. Documents categorization

Finding documents belonging to the same category requires usage of a similarity metric such as cosine similarity. It does not take into consideration lengths of vectors but only angle between them. Cosine similarity is equal to cosine of the angle between vectors. If cosine is equal to one then the vectors are the most similar, more specifically there are the same. For cosine equal to zero they are orthogonal which, taking into consideration the character of the data, means that they do not contain any common words, therefore are not similar. Authors have successfully accelerated cosine similarity measure in Field-Programmable Gate Arrays (FPGAs) as a part of a document comparison system in a previous work [5]. To compensate unequal importance of words in documents Term Frequency - Inverse Document Frequency (TF-IDF) weighting scheme can be applied. If a selected word appears frequently in a document if may be closely related to documents topic. However, if this word also appears frequently in a whole dataset it can be a commonly used word without significant impact on the document meaning.

Vector Space Model (VSM) is way of representation documents in application memory, which is necessary to compare them with each other. Using VSM we can calculate similarity using one of the metrics like cosine similarity. Besides all the benefits, vector space model brings also some problems. Adding new words to corpus cause the rise of dimensionality of vectors. With dimensionality, complexity of computing rises exponentially. This phenomena is called curse of dimensionality [2]. It affects computing in fields like distance functions, nearest neighbor search, machine learning and many more.

Described method is valid for small dimensional datasets. For higher dimensions it becomes less effective. At this point approximate methods, like Locality Sensitive Hashing (LSH), come with help [3]. Because there is significant amount of data to investigate and we desire to do it in real time we agree on some imperfections. LSH is different than cryptography hashing. In cryptography we are trying to have an algorithm that even for the smallest change in file generates completely different hash of the file. With LSH however, for similar input data it should produce similar hash. Locality sensitive hashing is a technique to decrease dimensionality in a given dataset. It can be parallelized what can brings notable speed-up of whole program. We can also treat vectors from vector space model as points of the end of the vector because all vectors have their begin in the middle of coordinate system. To calculate hashes we divide the hyperspace with random hyper-surfaces. Next, for every vector we calculate signatures. Signatures describe in which side of hyper-surface the vector has its end. We can do this in easy way by using vector normal to the surface and calculate cosine between surface normal and document vector. If cosine is positive the signature is positive, if the cosine is negative the signature is negative. At this point every document vector is described as vector of boolean values. It can indicate where the vector lie in hyperspace. Next step is to calculate hash of the signatures. It is done by summary multiples of very big prime number. Random hyper-surfaces are generated using Gram-Schmidt process, they are making projection of a vector on all previous vectors, and subtract this projection from the vector. We repeat it for every vector from the set. In this way we get set of vectors that are orthogonal to each other. Moreover in case when we have more vectors than dimensions we divide vectors in groups not bigger than dimensionality. As described in [4] to make the algorithm more friendly for parallel computing we can change order of vector calculations. Original version of the algorithm calculates the vectors serially, one vector at the time. To make is easier to implement in GPU the algorithm has to be parallelized. To do this we can calculate in one step one projection on all vectors. And with each step we get new vector necessary to calculate next step. Equation (1) shows the calculation of k^{th} vector.

$$U_k = V_k - \sum_{j=1}^{k-1} proj_{U_j}(V_k) \tag{1}$$

where:

$$proj_U(V) = \frac{\langle V,U\rangle}{\langle U,U\rangle} U \tag{2}$$

where $<V, U>$ is the inner product of vectors V and U.

3. Processing flow

First step of whole operation is to load corpus to memory. For experiments we used texts downloaded from news portal interia.pl as a corpus. The next step is to prepare the data for further processing. For this purpose we have to simplify texts in corpus. We used stop-list to delete words that are common but do not give any useful information. Then we use lemmatization to eliminate any grammatical forms and bring words to their basic form. Next, not to favor bigger documents we use TF-IDF. Afterwards we convert prepared corpus to Vector Space Model. Following step is orthogonalization of vectors. To do so, we use Gram-Schmidt process. Next the orthogonalized vectors are used to calculate signatures of vectors from corpus. In this case orthogonalized vectors act as random hyperplanes in random projection method. When two documents are similar (and have similar vectors) then there is high probability that they will have similar signature. The last step in data preparation is to compute hashes of signatures. Figure 1 presents schematic of Gram-Schmidt process describer in chapter 2. OpenCL implementation of the process is presented in Figure 2.

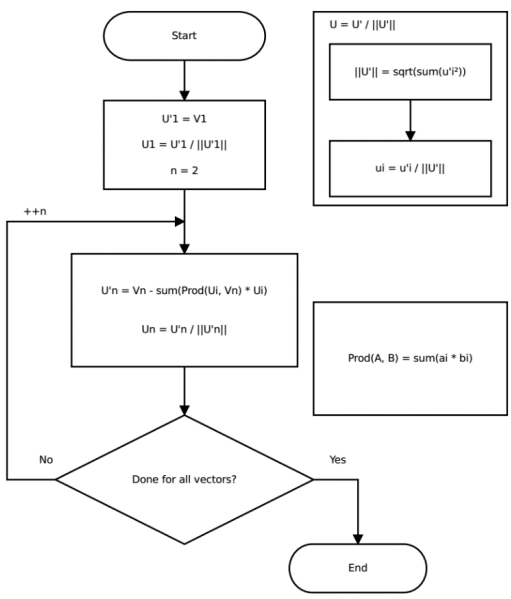


Fig. 1. Gram-Schmidt process diagram

```
1  for(int i = 0; i < number_of_vectors; i++){
2     if(vec_num > i){
3        //multiply elements (v*u)
4        global_temp[thread_num] = u[(vec_num + i) * dimensions + cell_num]
5                                  * hyperplanes[thread_num];
6        //sum(v*u)
7        if(cell_num == 0){
8           vec_len[vec_num] =
9                        sum(&global_temp[vec_num*dimensions], dimensions);
10       }
11       barrier(CLK_GLOBAL_MEM_FENCE);
12       // sum(v*u) * u
13       global_temp2[thread_num] = u[(vec_num + i) * dimensions + cell_num]
14                                  * vec_len[vec_num];
15       //multiply elements (u*u)
16       global_temp[thread_num] = u[(vec_num + i) * dimensions + cell_num]
17                                  * u[(vec_num + i) * dimensions + cell_num];
18       //sum(u*u)
19       if(cell_num == 0){
20          vec_len[vec_num] =
21                       sum(&global_temp[vec_num*dimensions], dimensions);
22       }
23       barrier(CLK_GLOBAL_MEM_FENCE);
24       // v projection substracted from u  (sum(v*u)/sum(u*u))*u
25       u[thread_num] -= global_temp2[thread_num] / vec_len[vec_num];
26       barrier(CLK_GLOBAL_MEM_FENCE);
27    }
28 }
```

Fig. 2. Gram-Schmidt process implementation in GPU

4. Experiments

As the main coding language we used Python. In GPU computations there are two major computing platforms, CUDA and OpenCL. CUDA seems to be better developed but works only on Nvidia graphic cards, while OpenCL programs can run on graphic cards from main manufacturers, as well as on some CPUs, therefore OpenCL was chosen to implement the kernels. Tests were made on a few platforms: Intel i5-2450M CPU, AMD X6 FX-6300 CPU, Nvidia NVS 4200M GPU, Radeon R9 280 GPU.

Tab. 1. Computation times comparison for CPU and GPU for orthogonalization and orthogonalization and signatures

	Matrix size	CPU time, s	GPU time, s	Time ratio
Orthogonalization	10	0.000541	0.028492	0.018988
	30	0.008936	0.027512	0.324804
	45	**0.029990**	**0.030566**	**0.981156**
	46	**0.030805**	**0.030083**	**1.024000**
	70	0.098122	0.038902	2.522287
	100	0.283812	0.063737	4.452861
	150	0.959562	0.162320	5.911545
	200	2.325558	0.431423	5.390436
	249	4.407996	0.938575	4.696477
Orthogonalization and signatures	10	0.000375	0.025302	0.014820
	30	0.006151	0.026044	0.236177
	51	**0.027167**	**0.028031**	**0.969176**
	52	**0.029109**	**0.026757**	**1.087902**
	70	0.076966	0.031947	2.409177
	100	0.178712	0.041078	4.350552
	150	0.595096	0.062597	9.506781
	200	1.461008	0.121431	12.03158
	249	2.940604	0.254221	11.56711

Table 1 presents computation times for experiments performing Gram-Schmidt process and signatures calculation. To depict performance growth we performed calculations for a range of matrix sizes. Figure 3 shows how much time takes the orthogonalization using CPU and GPU for different size of matrix, plot is magnified on matrices smaller than 100. It is visible that with increase of the size of the matrix the time grows in a non-linear manner. The phenomena is clear for CPU computation where increase of matrix brings huge increase of calculation time. For GPU it is less visible but still it is non-linear dependence.

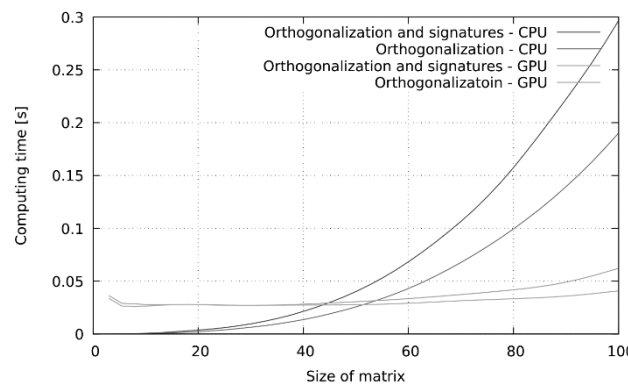


Fig. 3. Orthogonalization time, zoom on small matrix sizes

There are visible both advantages and drawbacks in both CPU and GPU cases. Time of CPU calculations starts from close to zero while GPU has some constant offset. This is connected with time needed to transfer the data to GPU memory and transfer the results back to host memory. It creates a time offset that cannot be avoided in GPU computations. However, time of CPU computing grow faster than GPU. It is connected with the fact that amount of computation grows non-linearly with growth of data to compute. Slower time growth in GPU can be achieved by parallelization of certain calculations. For matrices with size of 45-46, the time of GPU and CPU orthogonalization intersects. This is point when parallelization brings profits despite the time offset necessary to move the data to

and from graphical memory. When signatures are additionally calculated this situation happens for matrices of size 51-52.

Figure 4 shows computation time ratio between CPU and GPU orthogonalization. You can see that for small matrices the ratio is lower than one which means that CPU is faster than GPU, for nigget matrices however GPU is faster. The benefit is not limitless, for a certain size of matrices, speed-up stops to grow and is around constant. This is caused by fact that program cannot be parallelized to infinity. Data used to plot graphs in Figures 3 and 4 is also collected in Table 1. Points with ratio closest to one are bolded.

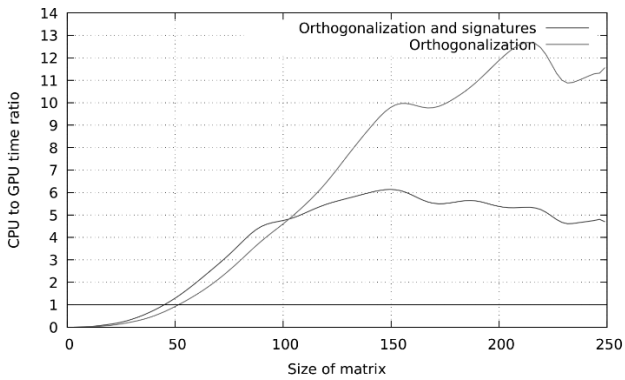


Fig. 4. CPU to GPU computation time ratio

5. Conclusions

Conducted experiments present advantages and drawbacks of using GPUs to accelerate computations. While GPU calculations can be massively parallelized and therefore speed up, the effect may not be significant if operations are more memory bandwidth demanding than computationally demanding. Also for small amounts of data, offset required for starting GPU computation may be significant. In experiments we showed that text categorization process can be accelerated up to 12 times when using GPU to compute orthogonalization and signatures. Presented method allows for significant reduction in response time of categorization process. However, still many elements of processing flow can be further accelerated, TF-IDF and cosine similarity can be implemented in hardware coprocessors. Also presented kernels can be further adjusted.

This research is supported by statutory founds of AGH UST, Department of Computer Science, Electronics and Telecommunication. No. 11.11.230.017.

6. References

[1] Sebastiani Fabrizio: Machine learning in automated text categorization. ACM computing surveys (CSUR) 34.1 (2002): 1-47.
[2] Keogh Eamonn, Abdullah Mueen: Curse of dimensionality. Encyclopedia of Machine Learning. Springer US, 2011. 257-258.
[3] Datar M., Immorlica N., Indyk P., & Mirrokni V. S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the twentieth annual symposium on Computational geometry, June 2004, pp. 253-262. ACM.
[4] Milde Benjamin, Schneider Michael: Parallel implementation of classical Gram-Schmidt orthogonalization on CUDA graphics cards. TU Darmstadt Fachbereich Informatik Kryptographie und Computeralgebra, 2009.
[5] Karwatowski M., Russek P., Wielgosz M., Koryciak S., Wiatr K.: Energy Efficient Calculations of Text Similarity Measure on FPGA-Accelerated Computing Platforms. In: International Conference on Parallel Processing and Applied Mathematics, Sept. 2015, pp. 31-40. Springer International Publishing.

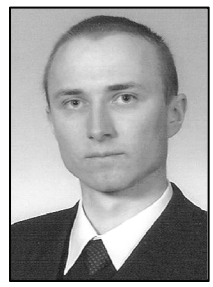
Received: 09.03.2017 Paper reviewed Accepted: 03.05.2017

Łukasz KORDUŁA, BSc, eng.

Received the BSc eng. degree in Electronics and Telecommunication from the Silesian University of Technology, Gliwice, Poland. Currently he is MSc student of the same field of study at the AGH University of the Science and Technology in Cracow. He is interested in both low level microcontroller programming and high level natural language processing.

e-mail: lukaszkordula@outlook.com

Maciej WIELGOSZ, PhD



He received MSc and PhD degrees in electronic engineering in 2005 and 2010 respectively from the AGH University of Science and Technology, Cracow, Poland. He currently works in Academic Computing Centre CYFRONET AGH and University of Science and Technology. His research interests include hardware acceleration, text mining and hardware architectures for artificial intelligence.

e-mail: wielgosz@agh.edu.pl

Michał KARWATOWSKI, MSc, eng.

He received the BSc Eng. and MSc degrees in electronic engineering in 2013 and 2014 respectively from the AGH University of Science and Technology, Cracow, Poland. Currently he is a student on PhD studies. His research interests include usage of hardware accelerators, especially FPGAs, in complex computations and energy efficient systems. Mainly for the needs of machine learning and natural language processing.

e-mail: mkarwat@agh.edu.pl

Marcin PIETROŃ, PhD

He received MSc degree in electronic engineering and in computer science in 2003 and PhD in 2013 from the AGH University of Science and Technology, Cracow, Poland. He currently works in Academic Computing Centre CYFRONET AGH and University of Science and Technology. His research interests include parallel computing, automatic parallelization and data mining.

e-mail: pietron@agh.edu.pl

Dominik ŻUREK, MSc, eng.

He received the BSc eng. and MSc degrees in electronic engineering in 2011 and 2012 respectively from the AGH University of Science and Technology, Cracow, Poland. Currently PhD student. His research interests include parallel computing, automatic parallelization and data mining.

e-mail: dzurek@agh.edu.pl

Prof. Kazimierz WIATR, DSc, eng.

He received the MSc and PhD degrees in electrical engineering from the AGH University of Science and Technology, Cracow, Poland, in 1980 and 1987, respectively, and the habilitation degree in electronics from the University of Technology of Łódź in 1999. Received the professor title in 2002. His research interests include design and performance of dedicated hardware structures and reconfigurable processors employing FPGAs for acceleration computing.

e-mail: wiatr@agh.edu.pl