Using Graphics Processing Units for Accelerated Information Retrieval

Ştefan MOCANU, Ramona DIN, Daniela SARU, Cosmin POPA

Faculty of Automatic Control and Computer Science,
University Politehnica of Bucharest,
060042, Romania,
stefan.mocanu@upb.ro

Abstract: The Parallel computing platforms enable dramatic increases in computing performance by harnessing the power of Graphics Processing Units (GPUs). Their design, based on a high level of hardware parallelization achieved through a big number of processing cores, made GPUs serious competitors for CPU based processing architectures. This fact is most obvious when it comes to processing huge amount of data. Many recent studies aimed at the development of GPU based implementations for various fields such as: astronomy, medicine, image processing, data compression and others. However, very few of them aimed at achieving information retrieval improvement based on GPU. Considering this, along with the latest stage of content based information retrieval algorithms and their practical efficiency for a wide variety of applications, this paper focuses on emphasizing parallel GPGPU algorithms performances against their CPU equivalents.

Keywords: content based information retrieval, graphic processing unit, GPU, GPGPU, inverted file, parallel computing

1. Introduction

Due to continuous and massive growth of heterogeneous data collections, new information retrieval techniques must be developed or, at least, present techniques should be improved. Information retrieval applications are currently widespread and millions of people are relying on them to accomplish their professional, academic or personal targets. Some of traditional (relational) database searching techniques are being replaced with data mining techniques dedicated to various applications such as: text documents search; image, audio and video content search; web search engines. Several challenges of digital information retrieval and solutions are presented in [3].

While search is the central topic within the area of information retrieval, researchers and IT engineers pay attention to a wide variety of other interrelated problems as well. These cover storage and data manipulation, document routing, filtering, and selective dissemination (e.g. news aggregators, e-mail spam filters), text clustering and categorization, topic detection and tracking, information extraction, summarization, question answering systems, expert/knowledge search, and other wide studied fields [1].

The volume of digital data available to be processed online is staggering. Large information service providers have massed databases of petabytes, the biggest intranets contain over a million web pages, and even private document collections showed an exponential growth lately. Internet itself was proven to closely follow a “Moore like” growth law [2] As the amount of data grows, searching and indexing costs rise up altogether with penalties in response time. Parallel algorithms and distributed processing and storing techniques may represent an efficient solution for enhancing modern search systems.

Although parallel computing algorithms eventually produce same results as their sequential equivalents, they have some advantages over the latter: shorter processing time, capability of processing higher amounts of data or solving a more difficult problem. Though, parallel algorithms usually have a higher complexity which requires more complex or additional hardware architectures. For years, all these were challenges for the central processing unit (CPU) and the performance improvement was highly dependent on the number of cores, being, of course, limited by the Amdahl's law [27].

During the last years, processing units' design has split into two major directions. On one hand, we have *multi-core processors* (2-10 cores per chip, such as AMD Phenom II, Intel I3, I5, I7 or Xeon E7) which are mostly dedicated to general purpose application for which they offer improved performance due to increased working frequency and small degree of parallelization. On the other hand, we *have many-core microprocessors* (processors with huge number of cores, hundreds or even thousands, such as Nvidia GTX590 – 1024 cores or Radeon HD 7970 – 2048 cores) which were mainly developed by graphics card manufacturers in their attempt to improve video performance with the support of the highly parallelized GPU.

For the past years, in certain applications, GPGPUs have shown far better performances than multi-core microprocessors available on the market and, also, their performance improvements from generation to generation are more consistent than those of CPUs. For instance, medium and peak floating-point calculation throughput is about 10 times better in case of GPGPU as opposed to CPU while the operating bandwidths of GPGPU and CPU follow the same ratio. A practical implementation of a non optimum algorithm in [6] achieved performances of almost 20 times better whiles the GPU's manufacturer states [7] that certain algorithms may achieved performances up to 400 times better.

The high parallelism of the GPUs' hardware design recommends them not only for video or graphic applications but for general purpose applications that require massive data processing. Development extensions (such as CUDA or C++ AMP) allow programmers to interact directly with the GPU and write and run applications that are not related to video or graphical purposes. In fact, this is the reason for which particular GPUs are also known as GPGPUs (General Purpose GPUs) which will be addressed from now on in this paper.

All these can justify once more the interest for exploiting GPGPUs in other areas than video or graphic rendering especially when it comes to implementing applications with a considerable amount of parallel sections. Furthermore, parallel algorithms as a concern of programming paradigm, have as long of a tradition as the one of sequential ones [4] and, although future processor generations promise to come along with hundreds of cores per socket [5], an application can only benefit from this if it's designed for parallel execution.

As already stated, information retrieval is the foundation of any modern search process. Considering this along with the great potential of graphic processors, especially CUDA enabled platforms, and advanced researches in the field of parallel information retrieval algorithms, this paper aims to analyze and compare performances of GPGPU specific algorithms against their similar CPU's implementations. The results will be compared and presented against each other, and further used in the design process of a GPU search engine.

2. State of the Art

For long time now, the central data structure of any information retrieval system has been the inverted file, i.e. a mapping between terms and their occurrences [10]. Many specialized algorithms such as terms and phase searching, proximity ranking, or Boolean query processing were implemented using such techniques as galloping search, based on inverted indexes. Although many other data structures have been studied, all failed to provide more efficient support and more flexibility than inverted files. One of the most analyzed approaches and an important competitor for inverted indexes was the signature files based approach [11]. This method intended to speed-up the process by eliminating documents which didn't match the query. With excellent results at beginning, signature files started to lose in popularity. Good behaviour in low memory conditions is no longer a strong point in the context of today's hardware systems and, in addition, scaling from simple word-queries to phrase queries is not an easy task.

Moreover, for large data volume, the frequency of false positive matches was too high for these algorithms to perform well.

The most common retrieval methods aim to produce ranked results, ordering them according to the relevance they have against the initial query. Oftentimes, queries designed for this type of retrievals are referred to as term vectors, each term within the query (usually placed in between white spaces or other distinct separators) mapping to a correspondent vector component. Ranking algorithms usually pay attention to terms ordering, thus the name of this approach, *vector*, instead of *set*. As one base principle of ranking algorithms, not all terms in the query need to be comprised in it for a result to be ranked and returned as relevant, otherwise users would have been forced to create exhaustive queries and the results would have been highly coupled to specific data. Therefore, although many relevant results will contain some of the query's results, few will contain all. It is the ranked retrieval algorithm's role to decide which results should be returned and which is the impact of any missing data on the final ordering. One of the oldest and best known information retrieval models is the one designed by Gerald Salton, starting back in 1960s under the name of *vector space model* [10]. As its name states, the algorithm relies on a vector representation of both queries and documents to search in. It ranks the set of document vectors according to a given query vector by comparing the angle between vectors and deciding the similarity level between the input and result. The smaller the angle, the more similar the vectors are. On a wide variety of implementations, the vector space ranking implementations may vary from explicitly depending on term and/or inverted document frequencies to relying only on term proximity, in which case features like document frequency or length play no role at all. In recent years though, vector space model has been overshadowed by machine learning approaches and probabilistic models.

Among latest retrieval and ranking algorithm implementations and as part of probabilistic approaches, relevance and pseudo-relevance feedback models play also a major role. On one hand, the query terms might be easily re-weighted based upon their distribution among queries' results (terms which occur within relevant results gain increased weights for later results and, conversely for those within non-relevant results, which get decreased weights). On the other hand, feedback is often used for changing the terms in the query, so that relevant not-found results which do not contain initial query terms may be ranked as relevant. One step further, language modeling and methods related to it offer great support for message retrieval and ranking, language categorization, online adaptive spam filtering, fusion and meta-learning [10]. The field of information retrieval also covers the World Wide Web, users requesting and performing documents browsing, filtering, clustering based on their contents. In order to provide high throughput and fast response time and since not even the most sophisticated and clever algorithm optimizations alone are not sufficient, almost all current highly competitive systems within information retrieval area use large clusters consisting of thousands of servers, where each server is responsible for searching a subset of indexed data. According to Google, in 2000 the size of its search index reached a total size of about 1 billion documents [12]; in 2014, Google's search index covers about 50 billion webpages [28].

The World Wide Web architecture successfully distributes the heavy workload over many servers, therefore indexing and searching even over a small percentage became a great challenge for the computational power of a single machine. The problem revealed itself as a non-trivial one since all major information systems already use many performance optimizations including caching, index compression, galloping, and early termination. By the scale at which these systems apply, they fall under three classes: web search – where they have to overcome challenges as searching over billions of documents, distributed within the entire world web; enterprise, institutional, and domain-specific search – where retrieval applies to enterprises' internal documents, databases, or enterprise resource platforms, and personal information retrieval systems – which usually applies to personal devices, and include large ranges of document types, in the same time requiring fast computer startup and well-balanced disk usage without making it visible to the user.

From a practical perspective, as a starting point, this paper focuses on basic text processing aiming to analyze the performance factor between CPU, classical parallel, and GPGPU implementations of simple text processing algorithms. The text processing techniques are fairly simple, but their effects may reveal important aspects. Although, complex reasoning and computer text understanding are still challenges for modern information retrieval systems, none of these is vital. Search engines for instance, capture the meaning of text by using ranking algorithms and retrieving data based upon the number of occurrences. Within all texts, there are a few words with high frequencies, but many others have low frequencies. In English for instance, "the" and "of" cover about 10% of all words and the first 6 most used words account for 20% of occurrences, whilst the 50 most frequent already cover 40% of all text. In literature, Zipf's law (Figure 1) describes this distribution, stating that

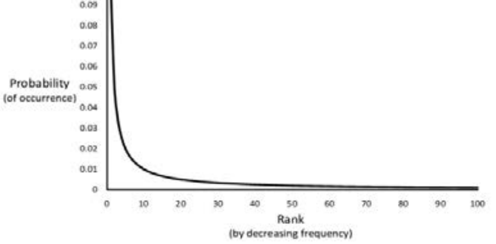

Figure 1. Rank vs. probability of words occurrence (Zipf's law)

the frequency of the *r*-th most common word is inversely proportional to *r* [13].

Phrases also follow Zipf's law. Moreover, combining the phrases and words in a relevant way, may result in better predictions for frequencies, even at low ranks as well [14].

There are many ways in which parallelism may help retrieval systems process information faster but existing studies are mostly limited to design systems appropriate for cluster running, comprising not more than a few dozen machines. Furthermore, although there are various parallel implementations for search-related algorithms, in order to achieve great performance increases, they must run against clusters with large number of machines.

During the past few years, general-purpose computing on GPU [8] has gained a lot of popularity in various yet very different areas such as chemistry, astronomy, applied mathematics and physics, not to mention the IT fields such as cryptography, database indexing and retrieval, neural networks and many others [15], [16]. Highly efficient processing based on a low cost hardware is a good reason to consider GPGPU computing a viable method for implementing algorithms with a considerable degree of parallelism [17].

Experiments revealed that pure algorithm optimizations may lead to impressive improvements over the single CPU-core version and, even more, the improvement factor rises along with a better coordination between GPGPU and CPU [18]. Unfortunately, this is not an universal truth, mobile GPUs, for example, being designed paying much more attention to power consumption rather than focusing on performance, thus lowering the bus traffic between GPU and other devices, (e.g. external memory, CPU) represent an exception [19].

Although there are many studies dedicated to information retrieval optimization of CPU based implementations [20], there are very few studies focused on using GPGPUs for implementation of information retrieval (parallel) algorithms [21].

The next section presents some information retrieval related algorithms, altogether with their GPGPU's implementation. They were meant to represent a starting point for further GPGPU image processing researches, in case of positive results.

The fourth section of the paper presents and compares the results after executing the code both on CPU and GPGPU. The last section presents conclusions and future plans for GPGPU programming.

3. Text based Information Retrieval Algorithms for GPUs

As applications which involve high modeling complexity and request much iteration, information retrieval algorithms are perfect candidates for GPUs implementation. For such a problem to be solved with parallel computing, it must be decomposed into sub-problems which can be safely computed at the same time, using a massive number of CUDA threads. Usually, the process of parallel programming can be divided into four steps: problem decomposition, algorithm selection, implementation, and performance tuning [9]. The starting key, when writing CUDA based applications, is to identify the work to be performed by each unit of parallel execution (a CUDA thread) so that both the problem parallelism and all hardware parallel units are well utilized. A typical CUDA application consists of several phases which can run either on CPU platform (host) or on GPGPU (device). Basically, any code which can run on device may be as well designed to run on host, but without guarantees about reaching the same performance level. Also, it is highly recommended that phases which exhibit little or no data parallelism to be implemented in the host code, otherwise the required communication time between host and device may lead to even lower performances than a single CPU implementation.

To analyze and compare the performance differences between CPU implementations and their GPGPU equivalent, the case study is based on implementation of two algorithms detailed in [29]. The first one is an information retrieval algorithm within an unsorted text file. The CPU implementation iterates over the input file and compares the target value against current iteration value; whenever all characters within the target string matches current value's characters in the exactly same order, the algorithm returns first character's position. Regarding the GPGPU execution, the implementation has to map software threads to as many hardware units as possible, in order to exploit GPGPU's parallelization. Considering that the input file consists of up to 2.5 million characters which have to be read and compared against one target string's characters, the algorithm's implementation spawns one thread for each character so that, each of them can determine if the characters starting at one specific index matches the target. Once a value has been found, its starting index is returned and a special flag with global GPGPU's memory is reset so that the other threads may be aware of this occurrence identification.

The second implemented algorithm computes the number of words in a text input. It identifies the characters at the beginning of a word by checking both that current character is an alphabetical one and that the one before it is not. The GPGPU implementation uses a specific CUDA library [7], known as *thrust* library. This application can be useful when implementing an automatic recommendation system based on searching certain keywords into big data collections such as documents or webpages.

4. Experimental Results

The hardware configuration has a strong impact on both CPU and GPGPU execution results, thus it is imperative to consider it during performance evaluation. The following experimental results are based on executing the algorithms on a mobile platform configuration with an AMD Turion64 X2 TL-62, running at 2.2GHz with 2 GB DDRAM400 of memory and one CUDA-enabled GTX470 graphics card with 1280 MB GDDR5 of global memory and 448 CUDA cores.

The text files used as input for both algorithms enclosed up to 2.5 million unsorted characters, while the target string used to search for was randomly generated with a fixed length of 3 characters. Within the scope of first approach, the input file has been entirely copied to GPGPU's global memory and the results were compared with the same algorithm implementation, but working only with limited blocks of data in memory at a time. Figure 2 presents the results gathered running the algorithm on CPU alone and using GPGPU as co-processor.

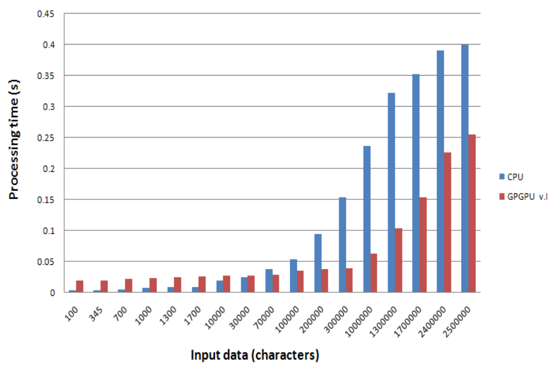

Figure 2. Search time on CPU vs. GPGPU

For larger data amounts a performance gain in favour of the GPGPU implementation can be observed, but for smaller input files, copying entire data set to GPGPU's memory proved to be an overhead, introducing significant delays. The crossing point between these two implementations was set around 45.000 data elements.

The same algorithm has been executed, but copying only segments of data into memory instead of copying the entire input set. For each copied segment, the implementation runs a call to one search method (known as *kernel* in GPGPU literature) which performs the actual information retrieval process inside of the input file. Next segment of data gets copied into memory only if no thread identified the target within the segment already in memory. This approach revealed even better processing times of the GPGPU's as opposed to its CPU similar implementation. This was a very small optimization in favour of GPGPU, not at the search algorithm level but at the data manipulation level. Since the purpose of this study was solely to analyze and compare the behaviours of the similar algorithm in CPU and GPGPU versions no other optimization was brought to the latter. Basically, even if there was no great concern for optimization, the results revealed a significant improvement when a large amount of data was involved in processing.

However, for a reduced set of input data, the CPU still performed better proving that there were noticeable delays introduced by communications between host and device in case of GPGPU. It can also be noticed a threshold after which the GPGPU processing time is increasing very slow as opposed to the CPU's behaviour. This observation is consistent with the Amdahl's law [27]. Another problem may come from different availability of GPGPU's resources at different but very close moments of time. Figure 3 illustrates the fact that GPGPU performance is nearly constant, as opposed to CPU who's processing time increases in proportion as input data set. A very similar behaviour was observed in a different study where the GPGPU was used in image processing applications [6].

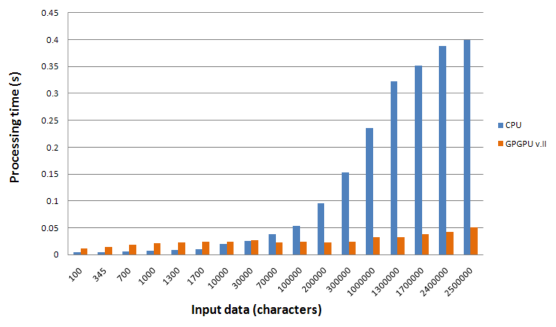

Figure 3. Search time on CPU vs. GPGPU improved implementation

The second case study revealed more dramatic performance differences between the CPU and GPGPU. The word counting algorithm has a different stop condition and a different return value than the search algorithm. It ends only when the entire input set was analyzed and returns the number of occurrences for the searched keyword. For this reason, fragmentation of the input set was not considered necessary nor implemented. However, this time, the difference was almost 150 times better on GPGPU platform. The results represented as a ratio between CPU and GPGPU execution time can be seen in Figure 4.

Once again, the input file has a major impact on final results, both on CPU and GPGPU versions. Just like in our previous experiments, GPGPU performed much better than CPU, this time even for small amounts of input data. The computed performance ratio reached a peak for 198.236 characters in the input file when the GPGPU was 144 times faster than CPU.

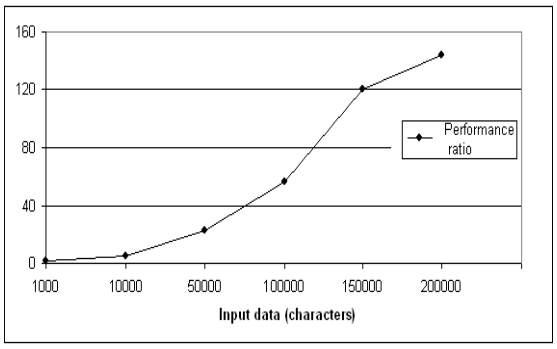

Figure 4. Performance ratio GPGPU vs. CPU

5. Conclusions and Future Work

This paper's main objective was to analyze the behaviour of two classic information retrieval algorithms' execution on CPU and GPGPU respectively as basis for future research.

The actual results have shown once again that low cost hardware can be a solution for supporting software applications that exhibit similar performances as ones that need far more expensive dedicated hardware platforms. Furthermore, for achieving superior but not optimum performance little specific training is required, medium parallel programming understanding and skills are sufficient.

Even though the software applications are portable over similar hardware platforms a significant dependence on them was noticed. For the presented case studies, a graphic card from NVIDIA along with CUDA Toolkit was used. Considering the results revealed a speed up of 144 times in case of detecting one word occurrences in a text without any advanced optimization, this has been marked as one strong argument for further research whether the GPGPUs may be efficiently exploited as an alternative to cloud computers for running enhanced search engines. If classic parallel information retrieval algorithms could be scaled to run on GPGPUs as hundreds of crawlers and for extremely large amounts of data, this may lead to excellent search results for World Wide Web or other massive data collections stored either local or distributed over many storage units.

Giving the results obtained so far it is expected that similar behaviours will be noticed in other areas where massive parallelization is involved. For instance, a rather cheap system with a GPGPU can be used instead of the more complex and expensive system presented in [30] with similar or better results.

A mid-term research goal is to use GPGPUs for improving medical images processing, classification and retrieval, exploiting them both for diagnoses and treatment procedures. Skin cancer, one of the most common form of cancer worldwide [22] and, also, the most aggressive one may benefit from faster and more precise computing offered by GPGPUs so that medical teams can promptly process and compare medical images and also develop new treatment plans and simulate biological processes. When treating various forms of cancer, oftentimes healthy tissue is unnecessarily affected. We intend to study new techniques of reducing patient exposure to treatment by faster processing of live images in order to improve precision of radiation therapy or allow fast and precise surgical interventions. Noticing the highly competitive and powerful general purpose graphic processors, researches have already begun conducting studies for including them in medical computations [23], [24], [25], [26] their results showing relevant performance improvements at low investment costs.

In addition to the performance gain offered by the parallel architecture of the GPGPUs, algorithms' optimization will be addressed in our future studies. First step may be represented by the implementation of a dynamic selection mechanism between the CPU implementation and GPGPU implementation which should choose a preferred implementation based on several parameters: size of input data, hardware configuration and performance and hardware availability.

Acknowledgements

This work was partially supported by the Romanian Ministry of Labor, Family and Social Protection, in contract POSDRU/89/1.5/S/62557.

REFERENCES

1. MANNING, C. D., P. RAGHAVAN, H. SCHUTZE, **Introduction to Information Retrieval**, Cambridge University Press, England, 2009.
2. http://www.physorg.com/news151162452.html, 2012
3. MITEA, A.-C., D. I. MORARIU, M. BREAZU, D. VOLOVICI, **Digital Information Retrieval**, Studies in Informatics and Control, vol. 19(2), 2010, pp. 185-192.
4. MOCANU, Ş., R. DOBRESCU, D. SARU, R. DIN, A. GRUMĂZESCU, **Arhitecturi complexe folosite în prelucrarea paralelă a imaginilor** (in Romanian), Revista Română de Informatică şi Automatică, vol. 20(1), 2010, pp. 97-105.
5. FEINBUBE, F., P. TROEGER, A. POLZE, **Joint Forces: From Multithreaded Programming to GPU Computing**, Software, IEEE, vol. 28(1), 2011, pp. 51-57.
6. DOBRESCU, R., Ş. MOCANU, D. SARU, A. GRUMĂZESCU, R. DIN, **Aplicaţii pentru procesarea de imagini pe platforma CUDA – studiu de caz** (in Romanian), Revista Română de Informatică şi Automatică, vol. 21(2), 2011, pp. 81-86.
7. *** NVIDIA – **GPU Computing**, http://www.nvidia.com/object/cuda_home_new.html
8. OWENS, J. D., M. HOUSTON, D. LUEBKE, S. GREEN, J. E. STONE, J. C. PHILLIPS, **GPU Computing**, Proc. IEEE, vol. 96(5), 2008, pp. 879-899.
9. KIRK, D. B., W. W. HWU, **Programming Massively Parallel Processors. A Hands-on Approach**, Elsevier, USA, 2010.
10. SALTON, G., A. WONG, C. S. YANG, **A Vector Space Model for Automatic Indexing**, Comm. ACM, vol. 18(11), 1975, pp. 613-6205.
11. BUETTCHER, S., C. CLARCKE, G. CORMACK, **Information Retrieval. Implementing and Evaluating Search Engines**, MIT Press, USA, 2010
12. http://googleblog.blogspot.ro/2008/07/ /we-knew-web-was-big.html, 2014
13. CROFT, B., D. METZLER, T. STROHMAN, **Search Engines: Information Retrieval in Practice,** Addison Wesley, First Edition, Feb 2009
14. QUAN, L. H., E. I. SICILIA-GARCIA, J. MING, F. J. SMITH, **Extension of Zipf's Law to Words and Phrases**, Proc. of the 19th Intl. Conf. on Comp. Linguistics, COLING-2002, pp. 315-320.
15. COHEN, J., **Novel Architectures: Solving Computational Problems with GPU Computing**, Computing in Science & Engineering, IEEE, vol. 11(5), Nvidia, Santa Clara, CA, USA, 2009, pp. 58-63.
16. DING, S., J. HE, H. YAN, S. TORSTEN, **Using Graphics Processors for High Performance IR Query Processing**, WWW '09, Proc. 18th Intl. Conf. on WWW, New York, USA, 2009, pp. 421-430.
17. SULLIVAN, T., H. NELSON, T. McBEE, M. ALVINO, **General-purpose Computing on Graphics Processing Units: GPU Processing of Protein**

Structure Comparisons, Technical report, 2007.

18. TEODORO, G., R. SACHETTO, O. SERTEL, M. N. GURCAN, W. MEIRA, U. CATALYUREK, R. FERREIRA, **Coordinating the Use of GPU and CPU for Improving Performance of Compute Intensive Applications**, Cluster Computing and Workshops, CLUSTER '09. IEEE Intl. Conf., New Orleans, LA, 2009, pp. 1-10.
19. AKENINE-MOLLER, T., J. STROM, **Graphics Processing Units for Handhelds**, in Proc. of the IEEE, vol. 96(5), 2008, pp. 779-789.
20. ZAKI, M. J., **Data Mining Parallel and Distributed Association Mining: A Survey**, IEEE Concurrency, vol. 7(4), 1999, pp. 14-25.
21. WENBIN, F., K. K. LAU, M. LU, X. XIANGYE, C. K. LAM, P. Y. YANG, B. HE, Q. LUO, P. V. SANDER, K. YANG, **Parallel Data Mining on Graphics Processors**, Technical Report HKUST-CS08-07, Oct 2008.
22. ***, **Death and DALY Estimates for 2004 by Cause for WHO Member States**, http://www.who.int/entity/healthinfo/global_burden_disease/gbddeathdalycountryestimates2004.xls, published 2009
23. SPOERK, J., C. GENDRIN, C. WEBER., M. FIGL, S. A. PAWIRO, H. FURTADO, D. FABRI, **High-performance GPU-based Rendering for Real-time, rigid 2D/3D Image Registration and Motion Prediction in Radiation Oncology**, Zeitschrift für Medizinische Physik, vol. 22(1), Elsevier GmbH., 2012, pp. 13-20.
24. BROWN, K., **UCSD Hospital Tests GPU-accelerated Cancer Treatment**, Nvidia Blog, 2012, http://blogs.nvidia.com/2012/04/ucsd-hospital-tests-gpu-accelerated-cancer-treatment/
25. *** NVIDIA, **Improving the Quality of Healthcare, Many Steps at a Time; Medical Imaging Technologies Running on GPU Produce better, Safer Results in Less Time**, http://www.nvidia.com/object/gcr-medical-imaging.html, 2012
26. HUANG, C. H., D. RACOCEANU, L. ROUX, T. C. PUTTI, **Bio-inspired Computer Visual System using GPU and Visual Pattern Assessment Language (ViPAL): Application on Breast Cancer Prognosis**, Neural Networks (IJCNN), The 2012 Intl. Joint Conf., Barcelona, Spain, July 2010, pp. 1-8
27. AMDHAHL, G. M., **Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities**, AFIPS Conf. Proceedings, 1967.
28. http://www.worldwidewebsize.com/, 2014
29. DIN, R., **Techniques for Medical Images Analysis using Parallel Processing**, PhD Thesis, UPB, 2012.
30. SIROMASCENKO, AL. - AL., I. LUNGU, **A Massive Multilevel-parallel Microscopic Traffic Simulator with Gridlock Detection and Solving**, Studies in Informatics and Control, vol. 22(3), 2013, pp. 279-288.