Performance improvement of data mining in Weka through GPU acceleration

Tiago Augusto Engel¹, Andrea Schwertner Charão², Manuele Kirsch-Pinheiro², Luiz-Angelo Steffenel³

¹Universidade Federal de Santa Maria, Santa Maria, Brazil
²Université Paris 1 Panthéon-Sorbonne, Paris, France
³Université de Reims Champagne-Ardenne, Reims, France

Abstract

Data mining tools may be computationally demanding, so there is an increasing interest on parallel computing strategies to improve their performance. The popularization of Graphics Processing Units (GPUs) increased the computing power of current desktop computers, but desktop-based data mining tools do not usually take full advantage of these architectures. This paper exploits an approach to improve the performance of Weka, a popular data mining tool, through parallelization on GPU-accelerated machines. From the profiling of Weka object-oriented code, we chose to parallelize a matrix multiplication method using state-of-the-art tools. The implementation was merged into Weka so that we could analyze the impact of parallel execution on its performance. The results show a significant speedup on the target parallel architectures, compared to the original, sequential Weka code.

© 2014 The Authors. Published by Elsevier B.V.
Selection and peer-review under responsibility of Elhadi M. Shakshuki.

Keywords: data mining tools; parallel computing; GPU

1. Introduction

In the new era of data-intensive science, data mining techniques receive more and more attention. Nowadays, data analysts can rely on a broad spectrum of tools, ranging in functionality, scope and target computer architectures. In the open source scenario, we find, at one side, some recently appeared tools like Mahout and Vowpal Wabbit, which can perform big data analytics in large computer clusters. On the other hand, we have desktop-based tools like R, Weka and RapidMiner, which are usually employed in smaller yet important problems.

Considering the computational complexity of some data mining algorithms, data analytics may take hours to complete. Different alternatives can be considered in order to improve data analysis performance. While some users will rely on cloud-based solutions, heterogeneous environments based on GPU architectures appear as a valuable solution to improve performance with significant cost saving. As pointed out by Schadt et al., using these environments, data can be stored and analyzed locally without requiring a specialized structure. This represents an interesting asset
when considering access control and privacy problem that may concern analyzed data. GPU architectures represent
then a complementary solution face to cloud environments. Nevertheless, several authors underline the complexity
of using such architectures that often require significant expertise on GPU programming and related technologies.
Such complexity may become an important limitation due to the growing popularity of data analytics for non-technical
users (for Business Intelligence or for live sciences, for instance).

Currently, desktop-based tools do not usually take full advantage of parallel architectures to speed up data analysis,
even though multi-core processors and GPUs are increasingly common on desktops. Most approaches to parallel data
mining focus on distributed execution of multiple experiments, or specific parallel algorithms that are not integrated
into popular data mining tools. Indeed, computational power offered by new GPU architectures remains almost
unexplored in most popular desktop data mining tools.

In order to demonstrate the interest of GPU architectures for improving data analysis performance on desktop
data mining tools, we propose, in this paper, an exploratory work in which we adapt a popular tool named Weka
(Waikato Environment for Knowledge Analysis) to a GPU use. Weka is a popular open source tool for data mining,
and it comprises a collection of data mining and machine learning algorithms packed together in a feature-rich
toolbox, which includes preprocessing and analysis tools. Weka provides users with a Graphical User Interface and
a Java-based Application Programming Interface. The Weka API reveals a number of packages containing abstract
and concrete classes. It implements algorithms for regression, classification, clustering, association rule mining and
attribute selection. With all these features, Weka is widely used in business, research and education.

Parallel and distributed computing was not a concern in the design of Weka, but it has been addressed by a few
Weka-related projects, as for example Weka-Parallel and Grid Weka. Furthermore, the development branch of
Weka (3.7) includes a package called WekaServer, which enables multiple servlet-driven process instances that can
run multiple tasks in parallel. Even so, parallel processing with GPUs is still underexploited by Weka.

In this paper, we explore further opportunities for parallel execution within Weka. Our main goal is to improve
the performance and reduce the end-user response time of Weka on a single machine, taking advantage of GPU
acceleration. To do so, we carried out an in depth study of Weka source code and analyzed its performance using
Java profilers. After that, we focused on the parallelization of one of its hotspots – matrix multiplication – using
state-of-the-art tools for GPU programming in Java.

The rest of the paper is organized as follows: Section 2 presents related work on parallel execution of Weka
algorithms. Section 3 discusses the profiling of Weka and presents some hotspots found on Weka classifiers. Section
4 describes our solution for replacing the sequential Weka matrix multiplication code by a parallel, GPU-accelerated
code. Section 5 presents our experiments and results with this parallel implementation. Finally, in Section 6 we draw
some conclusions and describe our next steps of research.

2. Related Work

Several initiatives to improve performance of data mining software can be found on the literature. Often such
works focus on distributed architectures, such as cluster and grids. Unfortunately, such architectures involve sig-
nificant equipment and maintenance costs. They may also be concerned by data transfer problems and issues related
to data access. GPU architectures appear then as a complementary solution offering improved cost performance
ratio without requiring any specialized infrastructure. Multiple works in the literature propose improving perfor-
mance of data analysis thanks to GPU architectures. Among them, several works propose programming interface,
modules or frameworks for programming dedicated application for data mining. Kumar et al., for instance, propose
a framework based on R for building data mining tools over heterogeneous architectures. Ma and Agrawal propose
a framework for translating data intensive application to GPU cluster. Similarly, Jiang and Agrawal proposes a
framework focusing particularly on building map reduce application over GPU architectures. On those works, data
analysis is performed by dedicated code particularly designed for these frameworks.

Indeed, only a few initiatives involve improving the performance of final users data mining tools by considering
GPU architectures. Weka is a good example of this kind of tools and of this issue. Most of works improving
Weka performance have focused on distributed architectures. GridWeka7 leverages the power of grid computing to accelerate data mining tasks. The two main components of this system are Weka server and Weka client. The server is based on the original Weka. Each machine participating in a Weka grid runs the server component. The Weka client is responsible for accepting a learning task and input data from a user and distributing the work on the grid. In this system, a set of data mining tasks can be distributed across several machines in an ad-hoc environment. Tasks that can be executed using GridWeka include: building a classifier on a remote machine, labeling a dataset using a previously built classifier, testing a classifier on a dataset, and cross-validation.

Weka4WS13 is an application that extends Weka to perform data mining tasks on WSRF enabled grids. The rst prototype of Weka4WS has been developed using the Java Web Services Resource Framework (WSRF) library provided by GT4. The goal of Weka4WS is to support remote execution of data mining algorithms in such a way that distributed data mining tasks can be concurrently executed on decentralized nodes on the grid, exploiting data distribution and improving performance. Each task is managed by a single thread and therefore a user can start multiple tasks in parallel, taking full advantage of the grid environment.

WekaG8 is another application that performs data mining tasks on a grid. It extends the data mining toolkit Weka. WekaG implements a vertical architecture called Data Mining Grid Architecture (DMGA), which is based on the data mining phases: preprocessing, data mining and post-processing. The application implements a client/server architecture. The server side is responsible for a set of grid services that implement the different data mining algorithms and data mining phases. The client side interacts with the server and provides a user interface which is integrated in the Weka interface (without modifying Weka itself).

Using another approach, Senger9 presents and evaluate Inhambu, a distributed object-oriented system that supports the execution of data mining applications on clusters of PCs and workstations. This system provides a resource management layer, built on the top of Java RMI, which supports the execution of Weka. The results are compared with those achieved by another initiative called Weka-Parallel6. Inhambu provides additional advantages such as application checkpointing, support for dynamic aggregation of hosts to the cluster, automatic restarting of failed tasks, and a more effective usage of the cluster.

In the multithreading scenario, Fang14 has introduced GPUMiner, a parallel data mining system that utilizes new-generation graphics processing units (GPUs). The software was designed focusing on three main components: a GPU-CPU buffer manager, a co-processing parallel mining module and a GPU-based viewing module. GPUMiner implemented a k-means clustering and the Apriori frequent pattern mining algorithms. The preliminary results have shown significant speedups over state-of-the-art CPU implementations.

In spite of these efforts to distribute Weka across a cluster or a grid, no real advance was made to effectively improve the usage accelerators on a single machine. Indeed, the current version of Weka only includes the possibility to launch a different task for each core, through the use of a client-server model (WekaServer package). While this "batch" mode allows the deployment of tasks over a multi-core machine or even a cluster of machines, each single task remains bounded to a single computing core. In our work, we try to improve the performance of Weka tasks by providing GPU acceleration on key elements of Weka. We believe that this effort contributed to the overall performance of Weka, as both single tasks and multiple experiments in parallel shall benefit from this acceleration.

3. Profiling Weka

Profilers are tools that allow the gathering and analysis of execution characteristics of a program. One of the main uses of profilers is to evaluate the different software elements and to detect which parts of the code are more computing intensive or introduce slowdowns. This can be helpful as a starting point to software parallelization.

To identify the most computing intensive procedures on Weka – "hotspots" – we focused on a subset of algorithms that are common to several Weka applications. We selected the MSP15 algorithm, which is responsible for the creation of decision trees that will be used in regression models. This algorithm was suggested by experts in data mining, which had used the MSP in a previous job16.

Several profilers for Java are available, with different strengths and strategies to collect information about a running program. For this reason, the results of different profilers may vary 17. It also worth note that the use of profilers is an intrusive approach that slows-down considerably the execution of a program, diluting the impact of external factors like I/O and network accesses. Having that in mind, this work combined the analysis from different profiling tools,
merging the results and focusing on the methods that most appeared. The tools we used are VisualVM\textsuperscript{18}, JProfiler\textsuperscript{19}, JProbe\textsuperscript{20} as well as Java own integrated profiler (jprof).

As expected, the performance and results from different profilers varied a lot. Indeed, VisualVM was exceptionally intrusive and slow, preventing the collection of any useful data. JProfiler had a better behavior than VisualVM, showing a lower overhead and allowing the detection of some hotspots as illustrated in Fig. 1. JProbe had the best behavior concerning the profiling overhead, and it highlighted several hotspots pointed by JProfiler. Finally, the Java profiler showed useful, even though it does not show a lot of detailed information.

![Fig. 1. Example of hotspot detection provided by JProfiler](image)

In addition to the overhead caused by the profiling applications, another important factor that must be considered relates to the choice of M5P parameters. Indeed, this algorithm allows different parameter combinations when analyzing the dataset, and we noticed differences between the profiling results when these parameters were changed. Therefore, the methods that caused most impact in the performance were chosen by crossing the results of the different parameters with the output from different profilers, resulting in the list illustrated in Table 1.

| Hotspot                                      | Inherent time | Average Time | Invocations |
|----------------------------------------------|---------------|--------------|-------------|
| weka.classifiers.functions.LinearRegression   | 5 159 s (21 %) | 150 ms       | 34 240      |
| weka.core.DenseInstance                      | 4 883 s (20 %) | 2 µs         | 2 147 483 647 |
| weka.core.matrix.Matrix                      | 4 366 s (18 %) | 4 µs         | 1 058 482 902 |
| weka.core.matrix.LinearRegression            | 3 860 s (16 %) | 72 773 µs    | 53 052      |
| weka.core.AbstractInstance                   | 2 916 s (12 %) | 2 µs         | 1 201 282 738 |
| weka.core.Matrix                             | 2 164 s (9 %)  | 1 µs         | 1 337 513 053 |
| weka.core.matrix.LUDecomposition             | 154 s (0 %)    | 2 914 µs     | 53 052      |

Table 1. Hotspots identified in Weka

Most of these methods are well-known to the parallel computing community and can be ported to GPUs. In this work we decided to evaluate the impact of GPUs through the parallelization of a single method, the Matrix Multiplication (weka.core.matrix.Matrix.times), as this method was presents a high number of calls with a non-negligible impact on the time they consume. Also, this matrix multiplication method is not only used by M5P but also by other classifier algorithms within Weka, so its parallelization may improve the overall Weka’s performance.

4. GPU implementation

Since Weka is written entirely in Java, and since Java does not support GPU devices, we had to find ways to do so. The Java policy of ”Write once, run everywhere” makes difficult to implement such feature due to several difficulties such as hardware detection and bytecode adaptation at runtime. Therefore, the simplest way is to use Java’s native interface to call CUDA (Compute Unified Device Architecture) functions in C. There are several initiatives enabling Java access to GPU devices. We can cite Aparapi\textsuperscript{21}, JCudaMP\textsuperscript{22}, JaBEE\textsuperscript{23} and JCuda\textsuperscript{24}.

The toolkit we chose was JCuda\textsuperscript{24}. JCuda uses the Java Native Interface (JNI), creating an interface for Java programmers to access the CUDA programming toolkit. The syntax is very similar to the original CUDA\textsuperscript{25} and, as in CUDA, the programmer writes a serial program that calls parallel kernels, which may be simple functions or full programs. Java native bindings are used to launch the kernel, which executes in parallel across a set of parallel threads.

JCuda is a stable software suite that provides an extension to CUDA Basic Linear Algebra Subroutines (CUBLAS), called JCublas\textsuperscript{26}, which we found useful to adapt the matrix multiplication (sequential) to a GPU-enabled one. Indeed,
JCublas provides a GPUs method cublasDgemm, which calculates matrix multiplications using double precision according to the framework presented on Algorithm 1.

**Algorithm 1:** The basic framework of DGEMM routines

| Step | Description |
|------|-------------|
| 1    | acc[0...rx][0...ry] = 0 |
| 2    | load one bm × bk block of A into smA[bk][bm] |
| 3    | load one bk × bn block of B into smB[bk][bn] |
| 4    | synch |
| 5    | while \((k/bk) > 0\) do |
| 6    | for \(ki = 0\) to \(bk\) do |
| 7    | load one column of A in smA into \(rA[0...rx]\) |
| 8    | load one row of B in smB into \(rB[0...ry]\) |
| 9    | \(acc[0...rx][0...ry] = rA[0...rx] \times rB[0...ry]\) |
| 10   | end |
| 11   | load one bm × bk block of A into \(smA[bk][bm]\) |
| 12   | load one bk × bn block of B into \(smB[bk][bn]\) |
| 13   | synch |
| 14   | end |
| 15   | Merge \(acc[0...rx][0...ry]\) with \(bm \times bn\) block of C. |

Please note that not all matrix multiplications are fit for GPU acceleration. Hence, we found that M5P uses mostly rectangular matrices of different sizes, and sometimes these matrices are too small to benefit from GPUs due to the overhead of data transfer to and from the GPU. Having that in mind, we created a mechanism to identify when the matrix size would benefit from GPU execution, i.e., when there is enough work to be done in parallel without suffering from the data transfer overhead. This mechanism identifies the size of the matrix and, according to its size, calls the parallel matrix multiplication or the sequential multiplication.

5. Experiments

To conduct these experiments, we executed the sequential and the GPU-improved version of Weka on three different machines. These machines contain different processors and NVIDIA GPUs, which allow us to infer on the contribution of both CPUs and GPUs to the overall performance of Weka. In all experiments and developments, we used Weka 3-7-7 developer version with Java(TM) SE Runtime Environment (build 1.7.0-09), and the machines run Debian 2.6.32. Please note that the ROMEO node is part of the ROMEO cluster†, a new platform that entered at the 151\textsuperscript{st} position of the last TOP500‡ ranking (and 5\textsuperscript{th} position at the Green500§ ranking). All the characteristics are presented in Table 2.

The dataset used in the experiments is entitled Relative location of CT slices on axial axis, and can be obtained at UCI Data Mining Repository\textsuperscript{28}. In\textsuperscript{29}, this dataset was used to predict the relative location of CT (Computed Tomography) slices on the axial axis using k-nearest neighbor search. Also\textsuperscript{30} used the data to apply weighted combinations of image features for the localization of small sub volumes in CT scans. The dataset has 386 continuous attributes and 53500 instances. It is a relatively large dataset for desktop-based data mining tools.

In order to evaluate the impact of GPUs on Weka, our experiments considered three different approaches:

---

† https://romeo.univ-reims.fr
‡ http://www.top500.org
§ http://www.green500.org
Table 2. Characteristics of the nodes used in our experiments

| GPU Configuration | LSC4          | DoubleKITT    | ROMEO         |
|-------------------|---------------|---------------|---------------|
| GPU Model         | Tesla M2050   | 2 x Quadro K2000 | 2 x Tesla K20Xm |
| CUDA Driver Version | 4.2          | 5.5            | 5.5           |
| CUDA Capability   | 2.0           | 3.0            | 3.5           |
| GPU Clock rate    | 1.15 GHz      | 0.95 GHz       | 0.73 GHz      |
| CUDA Cores (thread processors) | 448         | 384            | 2688          |
| Maximum number of threads per multiprocessor | 1536       | 2048           | 2048          |
| Maximum number of threads per block     | 1024         | 1024           | 1024          |
| Memory Clock rate | 1546 MHz      | 2000 MHz       | 2600 MHz      |
| Memory Bus Width  | 384-bit       | 128-bit        | 384-bit       |
| Total amount of global memory            | 3072 MBytes  | 2048 MBytes    | 6144 MBytes   |

| CPU Configuration | LSC4          | DoubleKITT    | ROMEO         |
|-------------------|---------------|---------------|---------------|
| Processor Model    | Intel Xeon E5620 | 2 x Intel Xeon E5-2620 | 2x Intel Ivy Bridge |
| Number of cores    | 4             | 2xe           | 2x8           |
| CPU Clock Rate     | 2.4 GHz       | 2.1 GHz       | 2.6 GHz       |

1. Sequential execution (Weka original behavior)
2. GPU-only (using JCublas)
3. A mixed code that selects the best approach (GPU or sequential) according to the matrices sizes.

Table 3 presents execution times for the three scenarios. To perform the tests, the number of folds was fixed in two and we compared different values for the number of attributes per leaf (-M). This parameter orients how the model tree (a regression tree) is built, by setting the minimum number of instances to allow at a leaf node. A small value of M produces a larger tree, favoring parallelization, while a large value of M produces a more compact tree, better adapted to sequential computation. By default, Weka sets the value of M to 4.

Table 3. Execution times of M5P on LSC4 with different approaches and values of M

| Parameter | Sequential | GPU | Mixed  |
|-----------|------------|-----|--------|
| -M 4      | 734.82s    | 289.01s | 276.15s |
| -M 8      | 728.31s    | 286.40s | 274.82s |
| -M 16     | 651.18s    | 274.52s | 262.74s |
| -M 32     | 603.17s    | 266.84s | 256.61s |
| -M 64     | 451.29s    | 240.47s | 233.26s |
| -M 128    | 338.54s    | 215.46s | 210.69s |

The GPU implementation improved considerably the M5P’s performance on LSC4, as can be seen in Fig. 2. As less the number of attributes per leaf (-M), more of them are needed, augmenting the height of the tree. When the tree is higher, the matrix multiplication algorithm is called more times and the speedup is more significant. On the other hand, when the tree is shorter, the accelerated matrix multiplication method was not called so many times, and its impact in the overall performance was less significant but yet considerable. The mixed approach showed itself the fastest approach, as the use of GPU-only brings an unnecessary overhead when the matrices are too small.

While the performance of DoubleKITT can be compared to that of LSC4, we observe that in ROMEO the improvement is much more important (Fig. 3). Although this node presents a better sequential performance than the other machines as it uses a new generation of CPUs, the GPU accelerated Weka presents a speedup of at least 89%. This improvement (compared to 49% on DoubleKITT or 60% on LSC4) indicates that the next generations of GPUs shall keep improving their efficiency, and therefore bringing a massive computing power to the user.

6. Conclusion

The popularization of GPUs represents new opportunities for software parallelization and performance improvement accessible to the final user on its own desktop. Data mining is one of the application fields that can benefit from
these advances. Indeed, some factors like data security and data availability may prevent a user to rely on third-party platforms from cloud computing providers.

In this paper, we presented the results of a first experience to improve the performance of Weka data mining software, a well-known data mining tool. Thanks to the use of Java profilers, we identified a set of operations that are time-consuming and that can easily be adapted to GPUs. In this first experiment we parallelized the Matrix Multiplication method, adapting the work method to take advantage of CPUs or GPUs according to the size of the matrices. As a result, we observed speedup levels of at least 49%, drastically decreasing the time the algorithm consumes to handle a dataset.

Future works shall continue the parallelization of other Weka algorithms on GPUs, but also improve the efficiency of CPUs on Weka, which for the moment is not able to take all the advantages of multi-core architectures.

The last version of our GPU-based Weka implementation can be found at https://github.com/tiiago11/MGPWeka.

Acknowledgments

The authors would like to thank the LSC Laboratory and the ROMEO Computing Center for the access to their resources. This project is partially financed by the STIC-AmSud PER-MARE project† (project number 13STIC07), an international collaboration program supported by CAPES/MAEE/ANII agencies.

† http://cosy.univ-reims.fr/PER-MARE
References

1. Schadt, E.E., Linderman, M.D., Sorenson, J., Lee, L., Nolan, G.P.. Computational solutions to large-scale data management and analysis. *Nature Reviews Genetics* 2010;11(9):647–657. URL: http://www.nature.com/nrg/journal/v11/n9/abs/nrg2857.html. doi:10.1038/nrg2857.

2. Wu, R., Zhang, B., Hsu, M.. Gpu-accelerated large scale analytics. Tech. Rep. HPL-2009-38; HP Labs; 2009. URL: http://www.hpl.hp.com/techreports/2009/HPL-2009-38.html.

3. Ma, W., Agrawal, G.. Auto-gc: automatic translation of data mining applications to gpu clusters. In: 24th IEEE International Symposium on Parallel and Distributed Processing - Workshop Proceedings. IEEE Computer Society; 2010. p. 1–8.

4. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.. Weka: practical machine learning tools and techniques with java implementations. In: *Proceedings of the ICONIP/ANZIIS`99 Workshop on Emerging Knowledge Engineering and Connectionist-Based Information Systems*. 1999, p. 192–196.

5. Celis, S., Musicant, D.R.. Weka-parallel: Machine learning in parallel. Tech. Rep.; Carleton College, CS TR; 2002.

6. Khussainov, R; Zuo, X.; K.N.. Grid-enabled weka: A toolkit for machine learning on the grid. *ERCIM News* n 59 2004;URL: http://www.ercim.eu/publication/Ercim\_News/en59/khussainov.html.

7. Perez, M.S., Sanchez, A., Herrero, P., Robles, V., Peña, J.M.. Adapting the weka data mining toolkit to a grid based environment. In: *Advances in Web Intelligence (AWIC)*; vol. 3528 of Lecture Notes in Computer Science. ISBN: 0302-9743. Lodz, Polonia: Springer; 2005, p. 492–497. URL: http://www.gmx.de/Publications/2005/PSHRP05.

8. Senger, H., Hruschka, E.R., Silva, F.A.B., Sato, I.M., Bianchini, C.P., Jerosch, B.F.. Exploiting idle cycles to execute data mining applications on clusters of pcs. *J Syst Softw* 2007;80(5):778–790. URL: http://dx.doi.org/10.1016/j.jss.2006.05.035.

9. Ghoting, A., Kambadur, P., Pednault, E., Kannan, R.. Nimble: A toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce. In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*; KDD `11. New York, NY, USA: ACM. ISBN 978-1-4503-0813-7; 2011, p. 334–342. URL: http://doi.acm.org/10.1145/2020408.2020464. doi:10.1145/2020408.2020464.

10. Kumar, P., Ozisik, Y., Memik, G., Choudhary, A.. High performance data mining using r on heterogeneous platforms. In: *Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW), 2011 IEEE International Symposium on*. 2011, p. 1720–1729.

11. Jiang, W., Agrawal, G.. Mate-cg: A map reduce-like framework for accelerating data-intensive computations on heterogeneous clusters. In: *Parallel Distributed Processing Symposium (IPDPS)*, IEEE 26th International. 2012, p. 644–655. doi:10.1109/IPDPS.2012.65.

12. Talia, D., Trunfio, P., Vorta, O.. Weka4ws: a wrenabled weka toolkit for distributed data mining on grids. In: *Proc. of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases* (PKDD 2005). Springer-Verlag; 2005, p. 309–320.

13. Fang, W., Lau, K.K., Lu, M., Xiao, X., Lam, C.K., Yang, P.Y., et al.. Parallel data mining on graphics processors. Tech. Rep.; Department of Computer Science and Engineering, Hong Kong University of Science and Technology; 2008.

14. Wang, Y., Witten, I.H.. Induction of model trees for predicting continuous classes. In: *Poster papers of the 9th European Conference on Machine Learning*. Springer; 1997,.

15. Machado, K.S., Winck, A.T., Ruiz, D.D., Norberto de Souza, O.. Mining flexible-receptor docking experiments to select promising protein receptor snapshots. *BMC Genomics* 2010;11(5):1–13.

16. Mytkowicz, T., Diwan, A., Hauswirth, M., Sweeney, P.F.. Evaluating the accuracy of java profilers. In: *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation*; PLDI `10. New York, NY, USA: ACM. ISBN 978-1-4503-0019-3; 2010, p. 187–197. URL: http://doi.acm.org/10.1145/1806596.1806618. doi:10.1145/1806596.1806618.

17. Apapari team. 2013. URL: http://visualvm.java.net/.

18. Jprofiler: Java profiler. 2013. URL: http://www.ej-technologies.com/products/jprofiler/overview.html.

19. Jprofiler: Java profiler. 2013. URL: http://www.ej-technologies.com/products/jprofiler/overview.html.

20. Apapari team. 2013. URL: https://code.google.com/p/apapari/.

21. Dottler, G., Veldema, R., Klemm, M.. Jcudamp: Openmp/java on cuda. In: *Proceedings of the 3rd International Workshop on Multicore Software Engineering*; IWMSE `10. New York, NY, USA: ACM. ISBN 978-1-60558-964-0; 2010, p. 10–17. URL: http://doi.acm.org/10.1145/1808954.1808959. doi:10.1145/1808954.1808959.

22. Zaremba, W., Lin, Y., Grover, V., Iabe: framework for object-oriented java bytecode compilation and execution on graphics processor units. In: *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units*; GPGPU-5. New York, NY, USA: ACM. ISBN 978-1-4503-1233-2; 2012, p. 74–83.

23. Jcuda: Java bindings for cuda. 2013. URL: http://www.jcuda.org/.

24. Nickolls, J., Buck, I., Garland, M., Skadron, K.. Scalable parallel programming with cuda. *Queue* 2008;6(2):40–53. http://doi.acm.org/10.1145/1365490.1365500. doi:10.1145/1365490.1365500.

25. Java bindings for cublas. 2013. URL: http://www.jcuda.org/jcuda/jcublas/JCublas.html.

26. Tan, G., Li, L., Triechle, S., Phillips, E., Mao, Y., Sun, N.. Fast implementation of dgemm on fermi gpu. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*; SC `11. New York, NY, USA: ACM. ISBN 978-1-4503-0771-0; 2011, p. 35:1–35:11. URL: http://doi.acm.org/10.1145/2063584.2063631.

27. Bache, K., Lichman, M.. UCI machine learning repository 2013. URL: http://archive.ics.uci.edu/ml.

28. Graf, F., Kriegel, H.P., Schubert, M., Pölstler, S., Cavallaro, A.. 2D image registration in ct images using radial image descriptors. In: *MICCAI* (2). 2011, p. 607–614.

29. Graf, F., Kriegel, H.P., Pölstler, S., Schubert, M., Cavallaro, A.. Position prediction in ct volume scans. In: *Proceedings of the 28th International Conference on Machine Learning (ICML) Workshop on Learning for Global Challenges*. Bellevue, Washington, WA, 2011,.