The development of algorithms for parallel knowledge discovery using graphics accelerators

Paweł Zieliński[a,b,*] and Jan Mulawka[a]

[a] Institute of Electronic Systems, Warsaw University of Technology, Poland

[b] Interdisciplinary Centre for Mathematical and Computational Modeling, University of Warsaw, Poland

ABSTRACT

The paper broaches topics of selected knowledge discovery algorithms. Different implementations have been verified on parallel platforms, including graphics accelerators using CUDA technology, multi-core microprocessors using OpenMP and many graphics accelerators. Results of investigations have been compared in terms of performance and scalability. Different types of data representation were also tested. The possibilities of both platforms, using the classification algorithms: the k-nearest neighbors, support vector machines and logistic regression are discussed.

Keywords: CUDA, OpenMP, k-nearest neighbors, support vector machines, logistic regression

1. INTRODUCTION

In this paper multiple General Purpose Graphics Processing Units (GPGPUs) and many core CPU has been examined for usage in parallel processing. The primary objective of this work is not creating particular complex system, but rather analytical study on performance results of knowledge discovery algorithms on parallel platforms. Main used technologies for implementations are Compute Unified Device Architecture (CUDA)[1] and Open Multi-Processing (OpenMP).[2] They had been selected for the greatest portability, but extending for Message-Passing Interface (MPI)[3] standard is straightforward. We also used other tools and libraries to assess the performance of the CUDA platform. Created implementations were tested on multiple platforms and we present the best achieved results, because in some configurations differences in performance may be significant. Scalability will be also presented on various hardware configurations, multi-core architectures and many graphics accelerators.

GPGPU has been widely recognized platform for parallel scientific computation. We implemented a number of classification algorithms and checked them on GPU, and also on multi-core CPU. Data representation was examined, we checked performance result and scalability for dense and sparse data. Compressed row storage (CRS)[4] format was used for sparse data representation.

2. PERFORMANCE OF TESTING

In our work we are interested in group of algorithms that classifies a given data example by assigning a label. Classification, known as supervised learning, is dived in two phases: learning and testing. We present results for both phases. In learning phase model from dataset is created, in testing phase this model is applied to identifying new data to which class belongs. The UCI[5] repository was used to performance tests. It is commonly used benchmark for the knowledge discovery algorithms, we also use other well know data sets. The following datasets Adult, Forest Cover type, Web[6] and MNIST[7] are shown in Table 1. Forest P1 is Forest dataset randomly divided into training and testing datasets.

*E-mail: pzielins@elektron.elka.pw.edu.pl

Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2011, edited by Ryszard S. Romaniuk, Proc. of SPIE Vol. 8008, 80080P
 · doi: 10.1117/12.904125

Set	Adult	Web	MNIST	Forest	Forest P1
Size	32561	49749	60000	561012	280506
Features	123	300	780	54	54
Testing set size	16281	14951	10000	-	280506

Table 1. Datasets description

To speed-up calculation we use the coefficient

$$S(n,p) = \frac{T(n,1)}{T(n,p)} \tag{1}$$

where n is size of problem, p is number of process (threads) or devices for CUDA and T(n,p) is execution time. We also use relative acceleration coefficient defined as

$$S_{\%}(n,p) = \frac{S(n,p)}{p} \tag{2}$$

The programs have been executed on an Intel Core 2 Quad Q6600 2.40 GHz with 4GB of RAM and on GTX295 containing two independent chips. CUDA framework[1] has been applied. Test were automated by bash scripts. We use MS Windows as testing platform because system overhead for CUDA was much smaller than on Linux.

3. EXPERIMENTAL RESULTS

In this section we present selected data-mining techniques and performance results. Some implementation details are also provided.

3.1 The k-Nearest Neighbors

The k-Nearest Neighbors (kNN) is a method in with an object is classified by a majority vote of its neighbors. The object is assigned to the class most common among its k closest training examples in the feature space, like in Figure 1.

We implemented exhaustive kNN algorithm with $O(N^2D)$ complexity, where D is size of dimension. Our solution is comparable with presented by Garcia et al.,[8] execution times on GPU are close. We also can use multiple GPGPU in system by OpenMP technology, extending it to MPI is trivial. After applying CSR data representation we achieve speed up only for MNIST dataset, computation on other sets took about twice longer, so result for CSR are omitted.

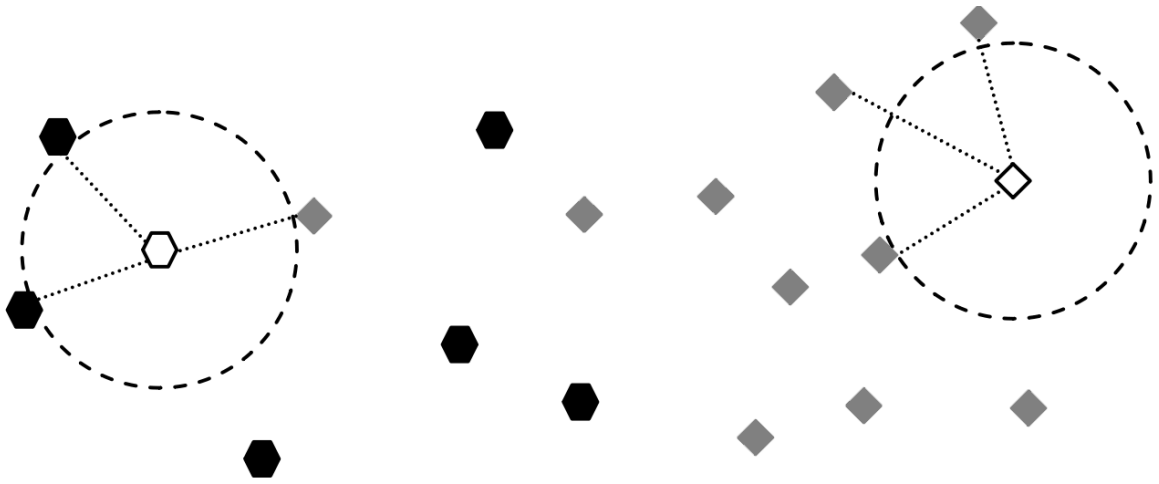

Figure 1. 2D example of classification of k-nearest neighbors with k equal 3.

We additionally supply highly optimized version on CPU using BLAS,[9] that is much faster than the ANN C++ library[10] for brute force version. Garcia[8] showed that the use of the NVIDIA CUDA API accelerates the kNN search by up to a factor of 400 compared to a brute force CPU-based implementation. Such huge acceleration is due to poor code optimization[11] of CPU version. Our implementation is only 5 times slower in worst case than GPU due to usage of Intel MKL library, that implements BLAS routines, for scalar product. Obtained results for time testing are presented in Table 2. Meaning for column notations are as follows:

kNNcpu$P$ - kNN version on multi-core CPU using OpenMP technology for parallelization, uses P threads. Sparse data representation is used, zero value are omitted.

kNNcpuDense$P$ - version similar to kNNcpu$P$, but uses dense data format. Additionally uses dgemm - BLAS routine for computing scalar product between bunch of examples.

kNNcuda$P$ - version of kNN on GPU, uses dense data format.

Set	kNNcpu1	kNNcpu4	kNNcpuDense1	kNNcpuDense4	kNNcuda1	kNNcuda2
Adult	115	29	31.6	9.9	3.3	1.9
Web	138	38	45.2	23.2	4.8	2.6
MNIST	1009	271	133	45.2	22	11.2
Forest P1	-	3037	-	1114	367	184

Table 2. Testing phase: execution time in seconds for kNN training, with k equals 16

Speed-up coefficients are shown in Table 3.

Method	Coefficient	Adult	Web	MNIST	Forest P1
kNNcpu$P$	$S(n,4)$	3.97	3.68	3.72	-
	$S_{\%}(n,4)$	99.1	92.0	93.1	-
kNNcpuDense$P$	$S(n,4)$	3.19	1.95	2.95	-
	$S_{\%}(n,4)$	79.8	48.7	73.6	-
kNNcuda$P$	$S(n,2)$	1.74	1.85	1.96	1.99
	$S_{\%}(n,2)$	86.8	92.3	98.2	99.7

Table 3. Speed-up on parallelization

3.2 Support Vector Machine

Support vector machine (SVM) is a non-probabilistic binary classifier and provides excellent generalization performance. The SVM training problem can be formulated as in Equation 3. Vapnik[12] showed how training a support vector machine leads to the quadric optimization problem (QP). Most popular strategy to solve this problem is decomposition, that splits the problem into a sequence of smaller QP subproblems to be efficiently solved.[6,13–17]

$$\min_{w,b} \frac{||\mathbf{w}||^2}{2} + C\sum_{i=1}^{n} \xi_i$$
$$\text{s.t.} \quad y_i(\mathrm{K}(\mathbf{x_i}, \mathbf{w}) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \tag{3}$$

Where $\mathbf{w}$ is a weight vector, b is a bias, C is a parameter, ξ_i is a slack variable, n is a number of training examples, y_i is target value and K is a Mercer kernel.

We use iterative algorithm to solve this optimization problem, called sequential minimal optimization (SMO).[6,13] Our approach is similar to that presented by Catanzaro,[18] additionally we use multiple GPGPUs and test different data representation.

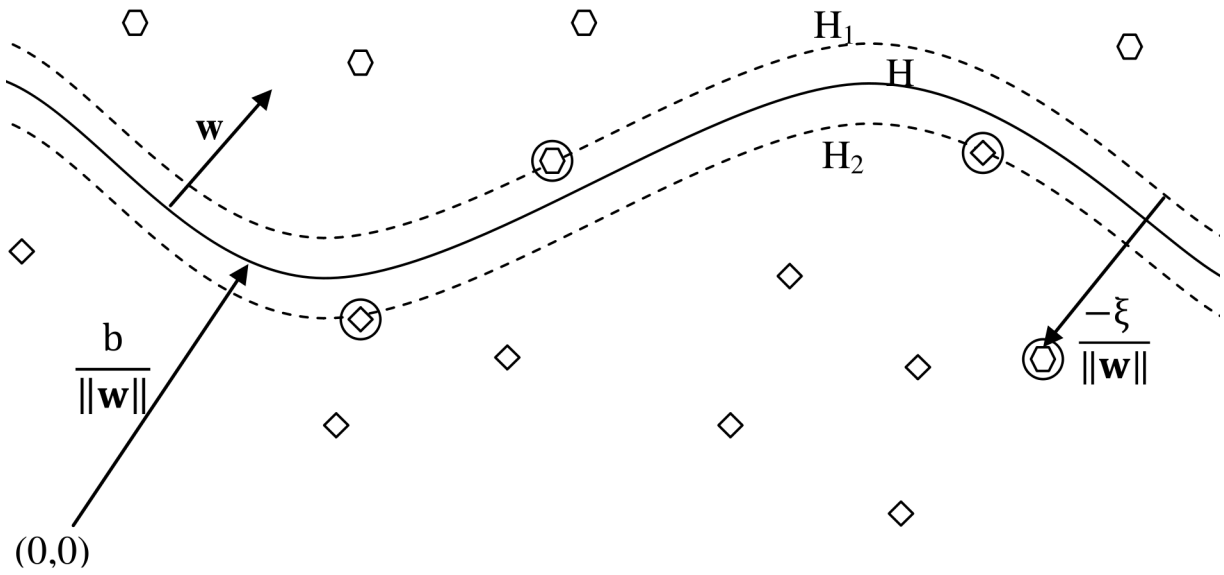

Figure 2. 2D example of classification of C-SVM

As follows from the above figure, maximum-margin hyperplane H for an SVM, samples on the margin are called the support vectors. Gaussian RBF kernel of the form $\mathrm{K}(\mathbf{x_i}, \mathbf{x_j}) = \exp -\gamma||\mathbf{x_i} - \mathbf{x_j}||^2$ was used in all experiments. Table 4 shows values used for C and γ for datasets.

Set	Adult	Web	MNIST	Forest	Forest P1
C	100	64	10	10	10
γ	0.5	7.8125	0.125	0.125	0.125

Table 4. C and γ values used for SVM training

Learning times are presented in Table 5. LibSVM[14] was used for comparison, because it is widely used implementation of SVM used for pattern recognition. The results are presented for the following different implementations:

PP - SMO algorithm on multi-core CPU using OpenMP technology for parallelization, uses P threads, uses cache for precomputing kernel function values between examples in dataset.

Cnocache - SMO algorithm for single CUDA device. It do not use cache, communication with CPU is reduced, learning occurs in epochs. All function are implemented for GPGPU device, CPU only checks if optimization ended. This version uses only single device.

Ccache$P$ - version similar to Cnocache, but contains cache. Parts of computation are made on CPU, mainly cache serving. This implementation uses dense date representation, but it makes into account a large number of zero values. Few GPU devices can be used.

CcacheCsr$P$ - an almost identical version to Ccache, but uses sparse data representation.

From Table 5 we can observe that, in each test problem usage of many devices or threads similarly as the sparse data representation reduces execution time. Method LibSVM and P1 are comparable in learning time, that is slow due to poor optimization[11] and wrong data representation for analyzed datasets. However, the method on the graphics accelerator is significantly faster than the version on the microprocessor. It may seem that the version without the cache should be much slower, but it is not. In the version without the cache exchange of information with the CPU was reduced to a minimum.

Set	LibSVM	P1	P4	Cnocache	Ccache1	Ccache2	CcacheCsr1	CcacheCsr2
Adult	460	618	166	52	20.2	17.6	14.3	13.1
Web	2325	2048	553	67	56	39	20.9	16
MNIST	18143	10862	2966	232	200	105	127	77
Forest	-	-	-	5854	2576	1417	1807	1102
Forest P1	-	-	-	1197	446	272	364	221

Table 5. Learning phase: execution time in seconds for SVM training

Presented results show significant performance gains, but there is still room for improvements. Many optimizations are possible for this problem such as cache shrinking or second order heuristic for choosing points for single QP optimization. Speed up coefficients are depicted in Table 6.

Method	Coefficient	Adult	Web	MNIST	Forest	Forest P1
PP	$S(n,4)$	3.72	3.70	3.66	-	-
	$S_{\%}(n,4)$	93.1	92.6	91.5	-	-
Ccache$P$	$S(n,2)$	1.15	1.44	1.9	1.82	1.64
	$S_{\%}(n,2)$	57.4	71.8	95.2	90.2	82
CcacheCsr$P$	$S(n,2)$	1.09	1.31	1.66	1.64	1.57
	$S_{\%}(n,2)$	54.5	65.3	83.0	82.0	78.3

Table 6. Speed-up on parallelization

Details from SVM training values of b (offset) and number of support vector are presented in Table 7 where Ccache* indicates all CUDA implementation using cache.

Set	Parameter	LibSVM	P1	P4	Cnocache	Ccache*
Adult	#SV	19059	18672	18672	18672	18518
Web		35231	35220	35220	35220	35150
MNIST		43754	43733	43733	43734	43673
Adult	b	-0.51021	-0.510144	-0.510144	-0.51018	-0.508418
Web		-0.94706	-0.946918	-0.946918	-0.94697	-0.946915
MNIST		0.249845	0.24973	0.249727	0.24971	0.249639

Table 7. Details from SVM training on provided implementations and LibSVM

Applying sparse data representation results in improving execution time. Achieved acceleration is presented in table 8. The best results can be seen for Web dataset, due to the largest reduction of row size.

Method	Adult	Web	MNIST	Forest	Forest P1
Ccache1/CcacheCsr1	1.6	2.7	1.6	1.4	1.3
Ccache2/CcacheCsr2	1.3	2.4	1.4	1.3	1.2

Table 8. Speed-up on data representation

The use of CSR has significantly accelerated execution time. It should be noted, however, that only with the large sizes of data and with a large number of attributes it is clearly visible. We do not present here time results for testing, because they are very similar to kNN.

3.3 Logistic regression

Logistic regression[19] is a generalized linear model, that is used for binomial regression. It can be applied when the target variable is a categorical variable with two categories. The approach is very similar to single-layer perceptron. Table 9 shows training time. In learning process, singular value decomposition (SVD) is used. Testing time were omitted, because they very short for analyzed datasets. Two version has been considered to test capabilities of parallel platform for generalized linear models:

LogitSVDP - version on CPU that implements iteratively reweighted least squares[20] algorithm.

LogitSVDcuda$P$ - version on GPGPU that make usage of CUBLAS.

Method	Adult	Web	MNIST	Forest	Forest P1
LogitSVD1	4.5	60	343	31	17.2
LogitSVD4	2.7	28	147	21	11.7
LogitSVDcuda1	1.26	9.7	57	9.8	5.3
LogitSVDcuda2	1.17	6.5	34	6.2	3.5

Table 9. Training time for logistic regression

Method	Coefficient	Adult	Web	MNIST	Forest	Forest P1
LogitSVDP	$S(n,4)$	1.7	2.1	2.3	1.5	1.5
	$S_{\%}(n,4)$	41.7	53.6	58.3	36.7	36.8
LogitSVDcuda$P$	$S(n,2)$	1.1	1.5	1.7	1.6	1.5
	$S_{\%}(n,2)$	53.8	74.6	83.1	79.0	75.7

Table 10. Result of speed-up on parallelization.

4. COMPARISON OF CLASSIFIERS

Further we apply hold-out technique for comparison of classifiers. Table 11 shows the classification accuracy where P is the number of GPGPU devices or CPUs. One may observe that there are no significant differences between the methods implemented on the processor architecture or graphics accelerator in all cases. The differences occur mainly in rounding floating point numbers and optimization of finding the minimum values.

Method	Adult	Web	MNIST	Forest P1
kNNcpu$P$	83.7	98.4	98.1	-
kNNcpuDense$P$	83.7	98.4	98.1	-
kNNcpuCsr$P$	83.7	98.4	98.1	79.1
kNNcuda$P$	83.7	98.3	98.1	79.1
kNNcudaCsr$P$	83.7	98.3	98.1	79.1
LibSVM	82.7	99.5	95.3	-
PP	82.7	99.5	95.3	-
Cnocache1	82.7	99.5	95.3	68.4
Ccache$P$	82.7	99.5	95.3	68.4
CcacheCsr$P$	82.7	99.5	95.3	68.4
LogitSVDCUDAP	85	98.7	90.1	70.1
LogitSVDP	85	98.7	90.1	70.1

Table 11. Accuracy in percents of classification on test datasets

The effectiveness of the classification, as well as previously received parameters of SVM proves that there is no significant differences between the implemented SMO algorithm and LibSVM. Due to inexact nature of optimization process, it is impossible to get equal results.

You can have a false impression that the method of k-nearest neighbors is better due to showed difference in effectiveness in Table 11. This is the result of badly chosen learning parameters: the cost of C and γ. However, values used in this work are commonly used to estimate the rate of learning SVM.

5. SUMMARY

We have investigated the possibility of some modern parallel architectures for selected knowledge discovery algorithms. As a result of this analysis, it can be noted that certain improvements in the process of computing have been achieved. In this respect, some new ideas concerning the modification of test algorithms for parallel computing are possible. By introducing some simplifications, has contributed to increasing their effectiveness.

In particular, the CUDA platform has been presented as a possible framework for graphics accelerators. The implementation in this environment is subject to many restrictions causing some difficulties to obtain full performance. However, allows a much better performance than using a single processor.

Classification algorithms such as the k-nearest neighbors, support vector machine and logistic regression were tested on multi-core processors and graphics accelerators. The possibility of usage few graphics accelerators was tested. Presented results of this work proves the superiority of graphics accelerators in the knowledge discovery algorithms. The presented results are not uniform, different values of speed-up coefficients were obtained between the acceleration platforms, which suggests that there are many ways for improvements. Notable impact on performance exhibit multiple graphics accelerators. Their application generally leads to a significant increase in performance.

Since graphics accelerators are an alternative to multi-core architectures, their use in some algorithms is necessary to obtain results in a acceptable time. Graphics accelerators and multi-core processors may create a hybrid form of high-performance computing unit allowing thus to solve complex computational problems in a shorter time.

Elaborated software was tested for data representation. It was verified which approach provides better performance on a given platform. In the case of graphics accelerators dense form of data is better where memory scanning has a significant contribution, as in the k-nearest neighbors. However the sparse data representation is preferable in support vector machine, where the utilization of memory bandwidth is masked by the numerical calculations.

REFERENCES

[1] *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide* (2007).

[2] Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., and Menon, R., [*Parallel programming in OpenMP*], Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001).

[3] Message Passing Interface Forum, "Mpi documents." `http://www.mpi-forum.org/docs/docs.html` (2009).

[4] Duff, I. S., Erisman, A. M., and Reid, J. K., [*Direct methods for sparse matrices*], Clarendon Press, New York, NY, USA (1989).

[5] Asuncion, A. and Newman, D. J., "UCI machine learning repository." `http://www.ics.uci.edu/~mlearn/Machine-Learning.html` (2007).

[6] Platt, J. C., [*Fast training of support vector machines using sequential minimal optimization*], 185–208, MIT Press, Cambridge, MA, USA (1999).

[7] LeCun, Y. and Cortes, C., "MNIST handwritten digit database." `http://yann.lecun.com/exdb/mnist/` (1998).

[8] Garcia, V., Debreuve, E., and Barlaud, M., "Fast k nearest neighbor search using gpu," in [*Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*], 1 –6 (june 2008).

[9] Foundation, N. S. and of Energy, D., "BLAS." `http://www.netlib.org/blas/` (2010).

[10] Mount, D. M. and Arya, S., "ANN: A Library for Approximate Nearest Neighbor Searching ." `http://www.cs.umd.edu/~mount/ANN/` (2010).

[11] Lee, V. W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., and Dubey, P., "Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu," *SIGARCH Comput. Archit. News* **38**, 451–460 (June 2010).

[12] Vapnik, V. N., [*The nature of statistical learning theory*], Springer-Verlag New York, Inc., New York, NY, USA (1995).

[13] Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., and Murthy, K. R. K., "Improvements to platt's smo algorithm for svm classifier design," *Neural Computation* **13**, 637–649 (March 2001).

[14] Chang, C.-C. and Lin, C.-J., "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology* **2**, 27:1–27:27 (2011). Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[15] Zanni, L., "An improved gradient projection-based decomposition technique for support vector machines," *Computational Management Science* **3**, 131–145 (2006). 10.1007/s10287-005-0004-6.

[16] Joachims, T., [*Making large-scale support vector machine learning practical*], 169–184, MIT Press, Cambridge, MA, USA (1999).

[17] Osuna, E., Freund, R., and Girosi, F., "An improved training algorithm for support vector machines," in [*Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop*], 276 –285 (sep 1997).

[18] Catanzaro, B., Sundaram, N., and Keutzer, K., "Fast support vector machine training and classification on graphics processors," in [*Proceedings of the 25th international conference on Machine learning*], *ICML '08*, 104–111, ACM, New York, NY, USA (2008).

[19] Hosmer, D. W. and Lemeshow, S., [*Applied logistic regression (Wiley Series in probability and statistics)*], Wiley-Interscience Publication, 2 ed. (2000).

[20] Hardin, J. W. and Hilbe, J., [*Generalized Linear Models and Extensions*], College Station, Texas: Stata Press (2001).