Scaling IDS Construction Based on Non-negative Matrix Factorization Using GPU computing

Jan Platoš, Pavel Krömer, Václav Snášel
Department of Computer Science
VŠB-Technical University of Ostrava
Ostrava, Czech Republic
{*jan.platos,pavel.kromer,vaclav.snasel*}*@vsb.cz*

Ajith Abraham
Machine Intelligence Research Labs (MIR Labs), USA
ajith.abraham@ieee.org

***Abstract*—Attacks on the computer infrastructures are becoming an increasingly serious problem. Whether it is banking, e-commerce businesses, health care, law enforcement, air transportation, or education, we are all becoming increasingly reliant upon the networked computers. The possibilities and opportunities are limitless; unfortunately, so too are the risks and chances of malicious intrusions. Intrusion detection is required as an additional wall for protecting systems despite of prevention techniques and is useful not only in detecting successful intrusions, but also in monitoring attempts to security, which provides important information for timely countermeasures. This paper presents some improvements to some of our previous approaches using a Non-negative Matrix factorization approach. To improve the performance (detection accuracy) and computational speed (scaling) a GPU implementation is detailed. Empirical results indicate that the speedup was up to $500\times$ for the training phase and up to $190\times$ for the testing phase.**

***Keywords*-non-negative matrix factorization, intrusion detection, GPU computing**

I. Introduction

Intrusion Detection Systems (IDS) were proposed to complement the prevention-based security measures. An intrusion is defined to be a violation of the security policy of the system; intrusion detection thus refers to the mechanisms that are developed to detect the violation of the system security policy. Intrusion detection is not introduced to replace the prevention-based techniques such as authentication and access control; instead, it is intended to be used along with the existing security measures and detect the actions that bypass the security control of the system. Thus, intrusion detection is usually considered as a second line of defense for computer and network systems.

Intrusion detection is defined to be the problem of identifying users or hosts or programs that are using a computer system without authorization and those who have legitimate access to the system but are abusing their privileges. Several types of intrusion detection systems are in use [1], [2], [3], [4], [5], [6]. Configuring IDS to detect internal attacks can be difficult. Part of the IDS challenge lies in creating a good recognition engines. The reason the recognition engine needs to be different is due to the fact that different network users require a different amount of access to different services, servers, and systems for their work. Once any attack patterns/behavior is identified, the system administrators will be able to identify any network users who pose a threat to network or system security.

Rest of the paper is organized as follows. In Section II, we present some theoretical background of non-negative matrix factorization approach. Section III and Section IV contains the summary of GPU computing and details of implementation of NMF on GPU. Experimental data are described in Section V and results are provided in Section VI followed by conclusions in the last Section.

II. Non-Negative Matrix Factorization (NMF)

The amount of audit data is huge even for small network but the IDS needs to evaluate them as quick as possible. The data may contain simple and complex relationships between features, which are difficult for humans to discover. Moreover, data may contain uncertainties and insignificant features. All these properties with usually high dimensionality of the data complicate the detection process.

This may be solved using several approaches. Data may be filtered and all uncertainties may be removed. Also, insignificant features may be removed too. Data can be grouped or clustered to reveal hidden patterns; by storing the characteristics of the clusters instead of the data, overhead can be reduced. All previously mentioned approaches may be done using dimension reduction techniques. Dimension reduction or matrix factorization or factor analysis is an important task helpful in the analysis of high dimensional real world data.

The group of dimension reduction techniques contains several methods. The most well known are Singular Value Decomposition (SVD) [7], [8], [9], [10], [11], [12], Semi-Discrete Decomposition (SDD), Principal Component Analysis (PCA) [13], [14] etc. Lars Eldén in the book [15] described methods for dimension reduction and theirs application to various problems such as Text Mining, Classification of Handwritten Digits, etc. Methods for dimensionality

reduction have already been successfully used to various problems many times. Application of dimension reduction techniques for designing of IDS is described in [1], [6], [16].

Non-negative matrix factorization [17], [18] is a class of decomposition, whose members are not necessarily closely related to each other [15], [19]. It was designed for data sets in which attribute values are not negative. A side-effect of this feature is that the mixing of components in decompositions can be additive only. A set of data S can be expressed as a $m \times n$ matrix A, where m is the number of attributes and n is the number of records in S. Each column A_j of A is an encoding of a particular record in S and every entry a_{ij} of vector A_j is the value of i-th term with regard to the semantics of A_j, where i ranges across attributes.

The NMF problem is defined as a search for an approximation of the matrix A with respect to some metric (e.g., the norm) by factoring A into the product $W \times H$ of two reduced matrices W and H. Each column of W is a basis vector which contains an encoding of a semantic space or concept from A and each column of H contains an encoding of the linear combination of the basis vectors that approximates corresponding column of A. Dimensions of W and H are $m \times k$ and $k \times n$, where k is the reduced rank. Usually, k is much smaller than n. Finding an appropriate value of k depends on application and it is also influenced by the nature of the collection itself. Common approaches to NMF obtain an approximation of A by computing a (W, H) pair to minimize the Frobenius norm of the difference $A - WH$. The matrices W and H are not unique. Usually H is initialized to zero and W to a randomly generated matrix where each $W_{ij} > 0$ and these initial values are improved with iterations of the algorithm.

NMF is computed in an iterative process. Minimization rules are applied in every iteration to minimize the difference between $W \times H$ and original matrix A. The first approach to solve this problem was based on multiplicative rules defined be Lee and Seung [18] and it can be described by the following steps:

1) Initialize matrix W and H with random numbers
2) For each iteration compute
   a) $H = H * \frac{W^T A}{W^T W H + \epsilon}$
   b) $W = W * \frac{A H^T}{W H H^T + \epsilon}$

where the ϵ constant is set to 10^{-9}, W^T and H^T represents the transposition of matrix W and H, symbol $*$ represents per element matrix multiplication and division (the fraction) is per element as well.

The NMF is a dimension reduction technique. To achieve decision making ability, another method must be used. In our previous works [16], [9], [20], we used the same and very simple technique. When the decomposition is processed, we take vectors of weights from matrix H and split them in two groups - clusters according their label, i.e. the first group contains all vectors which correspond to the normal traffic and the second group contains all vector which correspond to the attack. A center is computed in each group as an arithmetic mean of all vectors. These two center vectors and the basis vectors from matrix W are the knowledge extracted from the test data. When the testing phase is processed, each input vector is reduced into weight vector. Then its distance from two center vectors is computed and the closer one is selected as a winner. If the winner is the center of the first group, input vector is labeled as normal traffic; otherwise it is labeled as an attack.

III. GPU Computing

Modern graphics hardware plays an important role in the area of parallel computing. Graphics cards have been used to accelerate gaming and 3D graphics applications, but now, they are used to accelerate computations with relatively distant topics, e.g. remote sensing, environmental monitoring, business forecasting, medical applications or physical simulations etc. Architecture of GPUs (Graphics Processing Unit) is suitable for vector and matrix algebra operations, which lead to the wide usage of GPUs in the area of information retrieval, data mining, image processing, data compression, etc. Nowadays, one does not need to be an expert in graphics hardware because of existence of various APIs (Application Programming Interface), which help programmers to implement their software faster. Nevertheless, it will be always necessary to follow basic rules of GPU programming to write more effective codes.

The four main APIs exists today. The first two are vendor specific, e.g. they were developed by two main GPU producers - AMD/ATI and nVidia. The API developed by AMD/ATI is called ATI Stream and the API developed by nVidia is called nVidia CUDA (Compute Unified Device Architecture). Both APIs are similar. The rest two APIs are universal. The first one was designed by Khronos Group and it is called OpenCL (Open Computing Language) and the second was designed by Microsoft as a part of DirectX and it is called Direct Compute. All APIs are a general purpose parallel computing architectures that leverages the parallel compute engine in graphics processing units.

The main advantage of GPU is its structure. Standard CPUs (central processing units) contain usually 1-4 complex computational cores, registers and large cache memory. GPUs contain op to several hundreds of simplified execution cores grouped into so-called multiprocessors. Each SIMD (Single Instruction Multiple Data) multiprocessor drives eight arithmetic logic units (ALU) which process the data, thus each ALU of a multiprocessor executes the same operations on different data, lying in the registers. In contrast to standard CPUs which can reschedule operations (out-of-order execution), the selected GPU is an in-order architecture. This drawback is overcome by using multiple threads as described by Wellein et al. [21]. Current general purposes CPUs with clock rates of 3 GHz outrun a single

ALU of the multiprocessors with its rather slow 1.3 GHz. The huge number of parallel processors on a single chip compensates this drawback.

The GPU computing was used in many areas. Andrecut [22] described computing based on CUDA on two variants of Principal Component Analysis (PCA). The usage of parallel computing improved efficiency of the algorithm more than 12 times in comparison with CPU. Preis et al. [23] applied GPU on methods of fluctuation analysis, which includes determination of scaling behavior of a particular stochastic process and equilibrium autocorrelation function in financial markets. The speed up was more than 80 times than the previous version running on CPU. Patnaik et al. [24] used GPU in the area of temporal data mining in neuroscience. They analyzed spike train data with the aid of a novel frequent episode discovery algorithm. Achievement of more than 430x speedup is described in mentioned paper.

IV. Computation of NMF on GPU

The utilization of GPU in NMF is not so difficult, especially in case of basic version of NMF with multiplicative rules, because these rules are defined as the series of matrix multiplication and per element division. Matrix multiplication represents the task, for which GPUs were optimized since their creation. Per element operations are also often used in graphics rendering, because they are used for post processing effects such as motion-blur, bloom, heat-shimmering, etc.

Our algorithm was implemented using nVidia CUDA technology. The realization of matrix multiplication was simplified, because nVIDIA CUDA contains specially optimized implementation of BLAS library. This library contains functions for Vector-Vector, Vector-Matrix, and finally Matrix-Matrix operations. Our implementation use only *Sgemm* operation for matrix multiplication. This operation is able not only multiply matrices but is able to transpose first, second or both matrices. Other necessary functions, especially per element division and Frobienus norm were implemented manually in C for CUDA. Because of the fact, that CUDA is extension to standard C programming language, reimplementing of methods is simple. The implementation of per-element division was made in simple way. We store matrices in column-major form as a one-dimensional array. This format is necessary for BLAS operations. The per-element division was very simple - we write a method which take exactly one element from this array for both input matrices and store its division into third matrix. The computation of Frobenius norm is also per-element operation, but its implementation was more complicated that the per-element division. In the first step, the difference between both matrices is computed. In the second step, the squares of the differences is calculated. In third step, a parallel reduction technique is used for computation of the sum of all differences. Finally the square root is computed from the final value. The calculation of Frobenius norm is not called so often as the per-element division.

For comparison purposes, CPU based NMF computation was also implemented. This implementation was written in C++ and it was optimized for maximal performance.

V. Experimental data

The data for the experiments was prepared by the 1998 DARPA intrusion detection evaluation program by MIT Lincoln Labs [25]. The original data contains 744 MB data with 4,940,000 records. The data set has 41 attributes for each connection record plus one class label. Some features are derived features, which are useful in distinguishing normal connection from attacks. These features are either nominal or numeric. Description of these data may be found in [6], [16], [25], [26]. For our experiments, the 10% sample of these data is used [26].

The data was preprocessed; all nominal values were converted to the numeric representation and all features were normalized. The data was divided into 4 classes, according the attack classes defined in [26]. Each class contains all records signed as normal activity and all records which belong to specified attack class. For each class, a division into two collections - training data and testing data was performed. The training data contains 40% of randomly selected data and the testing data contains the rest of them. Finally, another one class was defined - this class contains all records from all 4 attack classes and the normal activity records. This class represented the main functionality of the IDS - distinguish attacks from the normal activity.

The data collections are called according the name of the attack class, which it contains. The five used collection are called *DoS* (Denial of Services), *U2R* (User to Root), *R2L* (Remote to User), *Probe* (Probing) and the last one which contains all records is called *Attack*, because its purpose is to test algorithm to distinguish any attack.

The statistics of the data collection is shown in Table I. As may be seen, the number of records differs in each collection. The class *DoS* contains the largest amount of records and the class *U2R* contains the lowest number of records. The amount of records, which belongs to the normal traffic is the same in each collection - approximately 90 000. This will compare the efficiency of the proposed algorithm in detecting events, which are rare (*U2R*) and events which are very frequent in the data (*DoS*).

Table I
STATISTICS OF THE DATA COLLECTIONS

Collection	training records	Testing records	Total records
DoS	195 494	293 242	488 736
U2R	38 931	58 399	97 330
R2L	39 361	59 043	98 404
Probe	40 553	60 832	101 385
Attack	197 608	296 413	494 021

Table II
ACCURACY FOR CLASS 1 (DOS) [%]

Iters.	10		50		100		500	
K	Teach	Test	Teach	Test	Teach	Test	Teach	Test
5	94.53	97.44	98.13	97.54	98.55	98.22	97.88	97.69
12	97.95	98.44	97.56	97.22	98.03	97.94	98.63	98.45
17	96.22	96.34	98.27	98.11	98.30	98.09	98.15	97.57
25	98.53	98.85	98.29	97.67	98.26	98.20	98.47	98.28
33	98.39	98.66	98.23	98.18	98.46	98.31	98.74	98.25
41	98.00	98.49	98.19	98.14	98.40	98.33	98.39	98.31

Table III
ACCURACY FOR CLASS 2 (U2R) [%]

Iters.	10		50		100		500	
K	Teach	Test	Teach	Test	Teach	Test	Teach	Test
5	77.69	74.46	78.93	81.31	78.78	84.37	81.67	83.44
12	87.46	82.41	87.98	90.93	92.58	95.74	95.09	97.16
17	87.86	82.35	92.34	95.65	94.30	96.69	96.06	98.10
25	92.29	89.20	96.68	98.82	97.08	98.54	96.07	99.02
33	91.21	87.27	97.29	99.18	96.51	98.88	98.24	99.57
41	92.51	89.58	97.54	99.28	97.50	99.73	98.55	99.87

Table IV
ACCURACY FOR CLASS 3 (R2L) [%]

Iters.	10		50		100		500	
K	Teach	Test	Teach	Test	Teach	Test	Teach	Test
5	74.88	75.38	76.71	78.18	77.17	79.36	78.93	78.71
12	81.36	79.27	90.01	92.37	88.15	90.31	92.77	95.31
17	84.23	79.89	89.35	90.80	90.92	92.84	97.05	98.29
25	86.49	85.19	96.10	98.14	97.13	98.33	98.81	98.80
33	86.36	84.16	96.05	97.69	98.17	98.78	98.90	98.80
41	86.72	83.66	97.38	98.54	98.57	98.93	98.82	98.80

Table V
ACCURACY FOR CLASS 4 (PROBE) [%]

Iters.	10		50		100		500	
K	Teach	Test	Teach	Test	Teach	Test	Teach	Test
5	85.31	84.61	88.10	88.08	90.27	90.67	87.94	87.88
12	90.57	89.99	88.60	88.79	92.66	92.03	93.70	93.79
17	90.84	90.13	90.44	90.77	90.22	90.24	94.72	93.36
25	88.91	88.33	98.13	96.97	94.93	94.96	95.67	95.33
33	91.45	90.93	96.15	95.72	95.64	94.57	97.30	96.76
41	92.85	94.00	97.16	95.06	95.61	94.83	98.05	98.05

VI. EXPERIMENTAL RESULTS

The experiment has two phases. In the first phase we present the efficiency of the NMF method as IDS for large data. In the second phase, comparison of the speed of CPU and GPU implementation of IDS will be presented. The main task in using of dimension reduction techniques is the setting of the reduction coefficient k. In our experiments, this coefficient was set to 5, 12, 17, 25, 33 and 41. The number of iteration used in training phase was set to 10, 50, 100 and 500. The number of iteration in testing phase was always 10. This is because the basis vectors must be computed as precise as possible in training phase, but computing weights from existing basis vectors is much simpler in testing phase. Since the matrices W and H were randomly initialized all experiments were repeated 3 times.

Table VI
ACCURACY FOR ATTACK COLLECTION [%]

Iters.	10		50		100		500	
K	Teach	Test	Teach	Test	Teach	Test	Teach	Test
5	92.88	90.96	95.98	95.80	97.44	97.31	97.76	97.31
12	97.13	97.41	96.90	96.68	97.91	97.94	97.69	97.76
17	97.47	97.64	97.18	96.91	98.12	98.00	98.06	97.89
25	97.24	97.58	98.24	98.05	97.68	97.43	97.95	97.71
33	97.59	97.88	98.19	98.03	97.83	97.60	98.12	97.57
41	97.64	98.05	98.23	98.06	98.24	98.18	98.08	97.57

Table VII
TIME CONSUMPTION FOR CLASS 2 [S]

Iters.	50				100			
K	Teach		Test		Teach		Test	
	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
5	18.4	0.9	3.0	0.1	38.0	1.6	3.2	0.1
12	49.0	1.1	7.9	0.1	99.8	1.9	8.0	0.1
17	72.6	1.2	12.2	0.1	173.3	2.2	12.1	0.1
25	124.1	1.4	21.1	0.2	267.6	2.5	20.7	0.2
33	205.3	1.6	32.8	0.2	417.5	2.9	31.9	0.2
41	285.7	1.6	43.1	0.3	526.8	3.0	43.4	0.3

Table VIII
TIME CONSUMPTION FOR ATTACK COLLECTION [S]

Iters.	50				100			
K	Teach		Test		Teach		Test	
	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
5	206.6	3.8	16.1	0.3	425.5	7.3	16.7	0.3
12	526.0	4.8	42.8	0.5	1 073.2	9.2	41.9	0.5
17	784.8	5.6	67.2	0.6	1 583.2	10.7	62.9	0.6
25	1 299.9	6.4	109.4	0.8	2 682.7	12.1	107.7	0.8
33	2 404.1	7.5	172.0	1.1	4 890.1	14.1	165.0	1.0
41	3 298.5	7.7	227.5	1.3	6 605.8	14.4	218.6	1.3

The basic results of the experiments for all five data collections are shown in Tables II-VI. As evident, the worst results were achieved for $k = 5$ and 10 training iterations. The algorithm is not able to calculate basis vectors precise enough. The accuracy of results increases with increasing values of k and/or number of iterations. The results achieved for $k \geq 25$ and number of iterations 50, 100 or 500 are very similar and are very good for all collections. The accuracy is higher than 95% for all collections and mostly it is higher than 97%. When we compare the results for Class 1 and Attack collection, which contain more attack records than normal traffic, with other collections, where the ratio between attack and normal traffic is contrary, then we sees that the results are very good for both types of collection. The NMF method works well without dependency on the ratio between event frequencies.

The second phase, time consumption of the CPU and GPU was compared. The results will be shown only for smallest and largest collections. Because of the change of parameter k, time consumption of the algorithm will be shown for both - small and large data. The results for 50 and 100 iterations are shown in Tables VII and VIII.

As evident, the testing times are almost equal, because

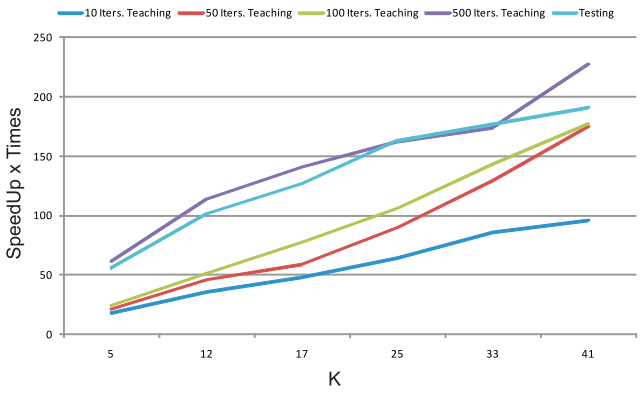

Figure 1. SpeedUp for the Class 2

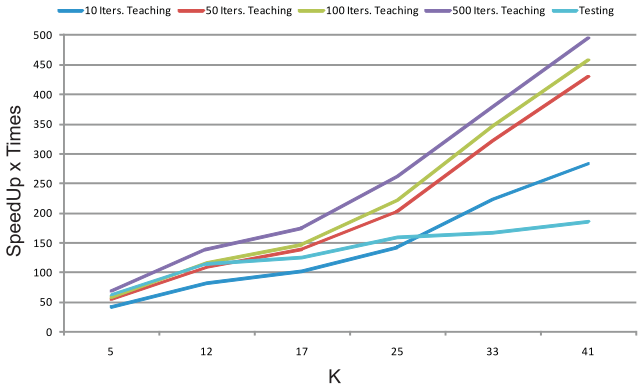

Figure 2. SpeedUp for the Attack Collection

only 10 iterations are used in any testing phase. The differences in time consumption increase with the size of the data. In practice, the most important aspect is the testing time, because the training phase must be done only once per time period, but the decision-making process must be applied on each record. But, because of the very good speed of the training process using GPU implementation, the training phase may be used more often than other methods. The testing times from Table VIII illustrate that the NMF based decision method is able to process almost 300 000 records in less than one second which is more than $100\times$ better than using conventional CPU implementation.

The speed up of the GPU version is depicted in Figures 1 and 2. As evident, the speed up of training phase depend on the size of the data (number of records and parameter k) and it is between $20\times$ and $230\times$ for Class 2 collections and between $40\times$ and $500\times$ for Attack Collection. The speed up in testing phase is different, because only 10 iterations are used. The results are between $50\times$ and $190\times$ for both collections.

As ideal setting may be choose 100 iterations and number of basis vectors equal to the 25. With this setting, the ideal ration between time consumption and accuracy is achieved.

VII. Conclusions

This paper presented a Non-negative Matrix factorization approach to detect different types of network intrusions. Moreover, a GPU implementation of this approach was described. The using of GPU in computational expensive tasks like NMF is very useful. The speedup was up to $500\times$ in training phase and up to $190\times$ in testing phase. This illustrate that the specific hardware used in IDS systems may be substituted by GPUs with preservation of open architecture and simple adaptability.

Acknowledgement

This work was supported by the Ministry of Industry and Trade of the Czech Republic, under the grant no. FR-TI1/420.

References

[1] S. Chebrolu, A. Abraham, and J. P. Thomas, "Hybrid feature selection for modeling intrusion detection systems," in *ICONIP*, ser. Lecture Notes in Computer Science, N. R. Pal, N. Kasabov, R. K. Mudi, S. Pal, and S. K. Parui, Eds., vol. 3316. Springer, 2004, pp. 1020–1025.

[2] S. Chavan, K. Shah, N. Dave, S. Mukherjee, A. Abraham, and S. Sanyal, "Adaptive neuro-fuzzy intrusion detection systems," in *ITCC (1)*. IEEE Computer Society, 2004, pp. 70–74.

[3] S. Mukkamala, A. H. Sung, and A. Abraham, "Modeling intrusion detection systems using linear genetic programming approach," in *IEA/AIE*, ser. Lecture Notes in Computer Science, R. Orchard, C. Yang, and M. Ali, Eds., vol. 3029. Springer, 2004, pp. 633–642.

[4] S. Mukkamala, A. Sung, and A. Abraham, "Intrusion detection using ensemble of soft computing paradigms," in *Third International Conference on Intelligent Systems Design and Applications, Intelligent Systems Design and Applications, Advances in Soft Computing*. Springer Verlag, Germany, 2003, pp. 239–248.

[5] S. Mukkamala, A. H. Sung, A. Abraham, and V. Ramos, "Intrusion detection systems using adaptive regression splines," in *ICEIS (3)*, 2004, pp. 26–33.

[6] V. Snasel, J. Platos, P. Kromer, and A. Abraham, "Matrix factorization approach for feature deduction and design of intrusion detection systems," in *IAS 2008 PROCEEDINGS*, M. Rak, A. Abraham, and V. Casola, Eds., 2008, pp. 172–179.

[7] M. Berry, S. Dumais, and T. Letsche, "Computational Methods for Intelligent Information Access." in *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, San Diego, California, USA, 1995.

[8] V. Snasel, P. Kromer, and J. Platos, "Evolutionary approaches to linear ordering problem," in *DEXA 2008 PROCEEDINGS*, 2008, pp. 566–570.

[9] V. Snasel, P. Kromer, J. Platos, and D. Husek, "On the implementation of boolean matrix factorization," in *DEXA 2008 PROCEEDINGS*, 2008, pp. 554–558.

[10] V. Castelli, A. Thomasian, and C.-S. Li, "Csvd: Clustering and singular value decomposition for approximate similarity search in high-dimensional spaces," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, pp. 671–685, 2003.

[11] S. Lahabar and P. J. Narayanan, "Singular value decomposition on gpu using cuda," in *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–10.

[12] P. Moravec, P. Gajdos, V. Snasel, and K. Saeed, "Normalization impact on svd-based iris recognition," june 2009, pp. 60 –64.

[13] I. K. Fodor, "A survey of dimension reduction techniques," Lawrence Livermore National Laboratory, Tech. Rep., 2002.

[14] M. Kurucz, A. Benczur, and A. Pereszlenyi, "Large-scale principal component analysis on livejournal friends network," in *Proceedings of SNAKDD 2008*, 2008.

[15] L. Eldén, *Matrix Methods in Data Mining and Pattern Recognition (Fundamentals of Algorithms)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007.

[16] J. Platoš, V. Snášel, P. Krömer, and A. Abraham, *Socioeconomic and Legal Implications of Electronic Intrusion*, 1st ed. Information Science Reference, April 2009, ch. Designing Light Weight Intrusion Detection Systems: Non-negative Matrix Factorization Approach, pp. 216–229.

[17] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, p. 788791, 1999.

[18] D. D. Lee and S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds. MIT Press, 2001, pp. 556–562.

[19] D. Skillicorn, *Understanding Complex Datasets: Data Mining using Matrix Decompositions*. Chapman & Hall/CRC, 2007.

[20] J. Platoš, V. Snásel, P. Krömer, and A. Abraham, "Detecting insider attacks using non-negative matrix factorization," in *IAS*. IEEE Computer Society, 2009, pp. 693–696.

[21] G. Hager, T. Zeiser, and G. Wellein, "Data access optimizations for highly threaded multi-core cpus with multiple memory controllers," in *IPDPS*. IEEE, 2008, pp. 1–7.

[22] M. Andrecut, "Parallel gpu implementation of iterative pca algorithms," *Journal of Computational Biology*, vol. 16, no. 11, pp. 1593–1599, November 2009.

[23] T. Preis, P. Virnau, W. Paul, and J. J. Schneider, "Accelerated fluctuation analysis by graphic cards and complex pattern formation in financial markets," *New Journal of Physics*, vol. 11, no. 9, p. 093024 (21pp), 2009. [Online]. Available: http://stacks.iop.org/1367-2630/11/093024

[24] D. Patnaik, S. P. Ponce, Y. Cao, and N. Ramakrishnan, "Accelerator-oriented algorithm transformation for temporal data mining," *CoRR*, vol. abs/0905.2203, 2009. [Online]. Available: http://arxiv.org/abs/0905.2203

[25] Lincoln Laboratory, "Darpa intrusion detection evaluation," February 2010. [Online]. Available: http://www.ll.mit.edu/mission/communications/ist/corpora /ideval/index.html

[26] The UCI KDD Archive, "Kdd cup data," February 2010. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html