Parallel KNN and Neighborhood Classification Implementations on GPU for Network Intrusion Detection

Phuangpaka Kuttranont, Kobkun Boonprakob, Comdet Phaudphut, Songyut Permpol, Phet Aimtongkhamand, Urachart KoKaew, Boonsup Waikham and Chakchai So-In
Applied Network Technology (ANT), Department of Computer Science,
Khon Kaen University, Khon Kaen, Thailand.
chakso@kku.ac.th

***Abstract*—With a rapid growth of Internet community making a practical usage of numbers of application used in many areas, i.e., research, commercial, industry, and even in military, there are millions of reports on attacks and attempts to invade the system online; and that phenomenon has led the essential of intrusion detection system (IDS). Data mining is one of the promising approaches to deal with large scale dataset including attack detection and recognition based on attack traces as an example from KDD CUP 1999. However, one of its key limitations is the computational complexity, and thus, this research investigates the possibility to integrate parallel processing to enhance the detection speed-up implemented on NVIDIA CUDA GPU. Several proposals have focused on k-Nearest Neighbour (KNN) as one of the promising approaches due to its key advantage of simplicity and high precision; however, in addition to KNN evaluation, this research also proposes the integration of a simplified neighborhood classification (Neighborhood) using the percentage instead of group ranking resulting in higher accuracy gain with in-significantly increase of computational complexity trade-off.**

***Index Terms*—Data Mining; GPU; Graphics Processing Unit; Intrusion Detection; k-Nearest-Neighbour; KDD CUP; Neighborhood; Network Security.**

I. INTRODUCTION

With era of Internet of Thing [1], not only people but also any instance is able to join the Internet; and that phenomenon rapidly increases the number of end systems as well as huge amount of data traffic; these massive systems and information will obviously lead to the concern of security; and this brings to the awareness of Intrusion Detection System (IDS) [2-3], especially when the Internet has no more limitation on just for education and non-profit organization.

Presently, IDS is one of the intuitive components for any organization to at least state a basic level of protection or prevention of the system. Based on the study in 2014 provided by Symantec and McAfee [4-5], there are beyond 556 million privacy breaches; and Cybercrime is keep growing and cost the global economy more than $400 million. IDS can be classified in terms of detection behavior into misuse–based and anomaly–based detections [6]. The first approach is based on signature matching while the second is to detect the anomaly behavior from the network. Each has its own strength in high detection precision and speed including its complexity as trade-off.

Apart from various attack recognition techniques [7], data mining is one of the efficient pattern classifications for misuse detection [8]. Traditionally, several probable classification algorithms have been well investigated to solve science and engineering problems including IDS [8-9]. It is worth noting that each approach has its own strength, especially the detection precision depending upon the data distribution (signature or behavior). However, above all, the main limitation is still on the computational time complexity, in particular, while applying into the real-time or online classification [10-11].

Especially, considering the application to IDS, k-Nearest Neighbour (KNN) [12] is one of the candidates for IDS classification as in the group of data mining approaches based on its key advantage of simplicity with high detection precision [13-14]. It should be noted that there are several proposals applying KNN as the solver for engineering problems [15]. In general, KNN will try to group or class a whole dataset using the nearest concept (distance between each attribute) in K group. To determine the testing data, the distance computation, i.e., Euclidean, will be performed with ranking concept; and then the final decision will be issued based on the majority vote.

Although KNN can be considered as one of the promising approaches for attack classification, one of our two contributions, here, is to propose another candidate by integrating a simplified neighborhood concept, i.e., Neighborhood [16], using the percentage instead of (K) group ranking for the purpose of higher precision gain. It should be noted that again although either KNN or our proposal can yield high detection precision, its key limitation is still on the computational complexity for real-time recognition system.

In the recent years, to move beyond the traditional serial computation, the practicality of parallel processing has stepped up with the invention of Graphics Processing Unit (GPU) [17]. GPU will normally co-function with a traditional CPU. However, high computational tasks will be placed to GPU but with aids of CPU instruction. GPU has its own strength, i.e., thousands core (processing units) and fast memory-cache; which then supports multi-thread computation as cost effective approach. Recently, the accessibility to program GPU becomes probable, e.g., with Compute Unified Device Architecture (CUDA) framework provided by NVIDIA with C/C++/C## programming [16]. Note that there are a number of application adopting this parallel-computation advantage, such as image processing, security and encryption, and simulation and modelling including data recognition [19-21].

Thus, in this research, the contribution lays on two-folds: noting that despite the attempts to apply data mining for IDS classification, again, one of which is KNN with key advantage of simplicity and precision gain; first, to enhance its efficiency using parallel processing, our focus is on the evaluation of KNN in GPU implementation on NVIDIA CUDA platform. Second, this research also proposes another candidate of IDS classification by integrating a simplified neighborhood technique (Neighborhood) using the percentage instead of group clustering. In addition, its speed-up is then enhanced with GPU implementation.

This research article is organized as follows. In Section 2, we briefly survey recent researches and proposals regarding the performance evaluation of data mining techniques on IDSs, especially KNN, Then, in Section 3, the overview of our methodology will be discussed including KNN and Neighborhood implementation on GPU. Section 4 provides the detailed analysis including the discussion of the comparative performance. Finally, the conclusions and future work are drawn in Section 5.

II. Literature REview

In this section, the main consideration will be on the focus of two approaches either applying KNN to solve a particular problem, especially implementing on parallel processing platform or the classification for IDS.

As previously stated, KNN can be considered as one of the most efficient classification algorithms proposed by many researchers. For example, in 2008, Garcia et al. [22] reported the evaluation of KNN efficiency using a statistical dataset on GPU using CUDA platform to illustrate the speed-up. Similarly, a year later, Kuang and Zhao [23] implemented the data segmentation algorithm on the focus of distance computation using CUDA platform on GPU in the comparison with CPU to show the speed-up. Recently in 2014, Nikam and Meshram [24] provided the performance evaluation results of KNN speed-up using GPU and OpenCL based on UCI datasets.

Considering data mining techniques applying to IDS classifications, Julisch and Helali [8-9] reviewed various data mining techniques to classify intrusion behavior. Patel and Sondhi [25] also reviewed the machine learning approaches applying to IDS, especially in terms of classification accuracy. In 2009, Jian et al. [26] showed the comparative performance of Apriori, KNN, and K-mean using KDD CUP 1999 dataset. Although K-mean resulted the fastest recognition, the detection accuracy is still in investigation. Recently, in 2014, So-In et al. [14] provided a detailed performance comparison survey over soft-computing approaches on KDD CUP datasets and reported that KNN is one of the most promising approaches so as to gain high precision for attack classification.

III. Empirical Study

In order to evaluate the performance of parallel classification techniques, there are three main components to perform the investigation, i.e., Data Preparation, Classification Model, and Parallel Processing as follows.

A. Data Preparation

In this research, the network intrusion detection traces was selected from KDD CUP 1999 [27] which is one of the biggest dataset for well-known intrusion detection systems. The dataset is suitable for the evaluation process since there exists the attacking status for testing purposes, i.e., either normal or attack. To lessen the experimental period, here, only 20% of the entire dataset was evaluated, and then, each half will be used as either training or testing set as stated in Table 1. In general, KDD CUP consists of four main attacks; namely, DoS (Denial of Service), PROBE, U2R (User to Root), and R2L (Remote to User).

Table 1
KDD CUP 1999 dataset

Type	Class	Total Records	20% of Records
NORMAL		261908	52832
DoS (178694)	Back	3720	744
	Land	69	14
	Neptune	160361	32072
	Pod	773	155
	Smurf	10291	2058
	Teardrop	3480	696
PROBE (45325)	Ipsweep	14006	2801
	Nmap	5784	1157
	Portsweep	11390	2278
	Satan	14145	2829
U2R (202)	Buffer_overflow	116	23
	Loadmodule	34	7
	Perl	12	2
	Rootkit	40	8
R2L (3871)	Phf	15	3
	ftp_write	31	6
	Guess_passwd	201	40
	Imap	44	9
	Multihop	28	6
	Spy	8	2
	Warez_client	3466	693
	Warezmaster	78	15
		490000	98000

DoS: This attack can freeze the system operation including activity by acquiring all resources, and so, the system cannot provide any services, one of each is the attack based on flooding schemes.

PROBE: This attack is generally used during a preparation stage to be ready for other attacking schemes in order to gain valuable information, such as enabled/disabled ports and active services as well as Internet name and address information.

U2R: This attack performs a specific operation so that the system is vulnerable to be penetrated as a hole or leak, such as Buffer Overflow attacks.

R2L: The attack is used to take advantages of related users' safety information and/or configuration, such as SQL Injection.

Table 2
KDD CUP dataset: record example.

Type	Record
NORMAL	0,tcp,http,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,1.00,1.00,1.00,0.00,0.00,2,233,1.00,0.00,0.50,0.10,0.00,0.00,1.00,0.98,normal
ATTACK	0,udp,private,SF,105,147,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,snmpgetattack

Note that the KDD CUP dataset is ZIP format consisting of a variety of attributes (41 in totals) to represent the attack records or traces. Each attribute will be separated by "," and the final attribute states the attacking status as examples shown in Table 2.

B. Classification Models

After the preparation stage, there are also three main sub-states, especially for the evaluation, i.e., Data Pre-processing, Distance Computation, and Classification Selection.

Data Pre-processing: Since the record of KDD CUP dataset also composes of characters, not just numeric, the data transformation will be required to convert all attributes into digits as example shown in Algorithm 1. Here, the protocol field will be converted accordingly, i.e., *0* is *tcp* and *1* is *udp*.

Algorithm 1: String to Digit Conversion

```
1  int preProcessProtocol(string protocol){
2    if (protocolMap.size() == 0) {
3                    protocolMap["tcp"] = 0;
4                    protocolMap["udp"] = 1;
5                    protocolMap["icmp"] = 2;
6  }
7        return protocolMap[protocol];
8  }
```

Distance Computation: Once all attributes are in numeric formats, the distance will be computed between the known dataset (aka training) (q) and the unknown one (aka testing) (p) based on the equation below.

$$Distance = \sqrt{\sum_{k=1}^{n} (p_k - q_k)^2} \quad (1)$$

Figure 1 shows an example of distance computation such that given the data set 1 (Test) at the first record, the attribute 1 (p_1) will be performed the subtraction operation with the first attribute (q_2) from data set 2 (Train), then making a square; this operation will be continued until the end of data set 2 leading to the final distance of the first record.

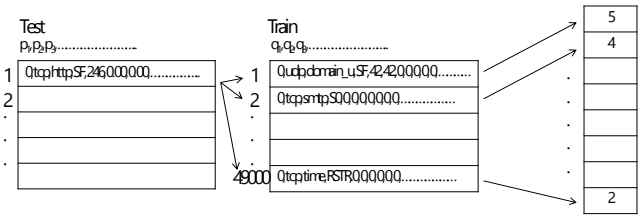

Figure 1: Example of distance computation

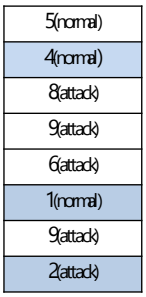

Figure 2: Example of classification based on distance computation

Classification Selection: This state is used to perform the actual classification based on either KNN or Neighborhood. Given the result from the first state (as examples shown in Figure 2), for KNN (K=3), the least distance will be selected in a group of K or 3 in this example. Here, the distance of 1 (normal), 2 (attack), and 4 (normal) are in the group of three, and with the majority concept, "normal" will be finally representing for the final decision.

However, with Neighborhood, instead of using K groups, the percentage out of the maximum distance will be then used. For instance, with 50% Neighborhood, the maximum distance is 10, so the selection of distance will be less than 5, and so, the distance of 1 (normal), 2 (attack), 3 (normal), and 4 (attack and normal) will be then selected in an interest group. Finally, "normal" will be the decision based on the majority concept.

C. Parallel Processing

Figure 3 shows an overview of parallel processing with GPU integrating with CPU. Here, a share memory architecture was used. There are three steps of the processing as follows: (1) Data Transfer (the computed data will be replicated for GPU processing from shared memory), (2) Data Instruction (the key commands will be issued from CPU to GPU), and (3) Parallel Processing (the main operation, i.e., high computational tasks, will be performed at GPU cores).

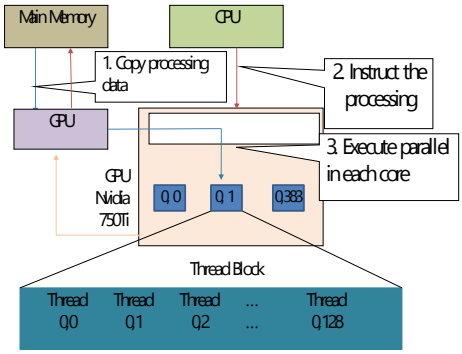

Figure 3: Example of parallel processing

Algorithm 2: KNN Implementation in GPU (K = 3)

```
1  Start_knn();
2  void knn(vector<networkTraffic> t,
   vector<networkTraffic> testData, double k)
3  double* d_knnMinDistances;
4  cudaMalloc(&d_knnMinDistances, k *
   sizeof(double));
5  double* d_knnLabels;
6  cudaMalloc(&d_knnLabels, k * sizeof(double));
7  __global__ void cuComputeDist
   networkTraffic*t, networkTraffic *testData,
   double *distance, double *label, double
   *knnMindistances,double *knnLabels,double
   *knnGuesses, int j,int size,int k,int max,int
   maxClass) {
8  unsigned int i = blockIdx.x * blockDim.x +
   threadIdx.x;
9  if(i<size) {
10   double sum = pow((t[i].duration
     testData[j].duration), 2) + ... +
     pow((t[i].dst_host_srv_rerror_rate
     testData[j].dst_host_srv_rerror_rate), 2);
11   distance[i] = sum;
12   label[i] = t[i].label;
13 }
14 __syncthreads();
15 If (i<k) {
16    knnMindistances[i] = distance [i];
17    knnLabels[i] = label[i];
18 }
```

Algorithm 2 also shows detailed operations and implementations of KNN in GPU as follows: after the initialization and function declaration (lines 1-2), line 4 shows the main function of KNN; lines 3 to 6 state the variable declaration; line 7 indicates the parallel function operation; line 8 shows the thread indication based on

blocking concept; given the size, lines 9 to 13 show the main computation tasks (the summation of each attribute stated in line 10 which is 41 in total, and here the notation is +…+); line 14 is used to start thread operations; lines 15 to 18 show the minimum distance computational process.

It should be noted that the parallel algorithm for Neighborhood is similar to that of KNN. However, the derivation of distance will select the ratio (percentage) instead of group *k*, modified in lines 15 to 18.

IV. Performance Evaluation

In this section, the evaluation processes were performed. In general, there are two main scenarios to illustrate the classification precision and computational time performance in both CPU and GPU.

A. Empirical Setup

To intensively validate the empirical results, *K*-Fold Cross Validation [28] was selected with folds and confusion matrix [29]. The first validation was based on the dataset with division of *K* sets equally (*K* folds). In each round, a single set was chosen from *K* to be a testing set and the other *K–1* as a training set to perform the actual evaluation. Then, the subsequence set will be performed accordingly as the testing and the training for the others, *K* rounds, in total, and here, for simplicity, two was chosen for this evaluation as *K*.

For the sake of simplicity, the evaluated dataset is 20% out of the 490000 records due to the computational time constraint. The evaluation system was on Windows 7 Core i5; 4 GB DDR-SDRM and 1 TB 5400 rpm DISK with NVIDIA 750Ti (as a graphic card) [30].

Two main metrics were used in this setup; namely, classification precision (accuracy) and computational time (seconds). The first metric was based on confusion matrix given predicted and actual values in terms of TP (True Positive), FN (False Negative), FP (False Positive), and TN (True Negative), to illustrate the classification accuracy as stated in equation 2 and Table 3.

There are two main scenarios for the purpose of comparative evaluation on the classification performance of KNN and Neighborhood. To state the comparative precision, for KNN, *K* was varied in range (odd number) of *3* to *9*, respectively. However, with Neighborhood, instead, the percentage was varied from 10% to 90% with the increment of 20% each. Both classification techniques will be evaluated in both CPU and GPU to state the comparative speed-up.

$$\%Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100 \quad (2)$$

Table 3
Confusion matrix.

Class	C1 (Predicted)	C2 (Actual)
C1 (Predicted)	True Positive (TP)	False Negative (FN)
C2 (Actual)	False Positive (FP)	True Negative (TN)

B. Empirical Results

Table 4 shows the performance evaluation results from the first scenario in that considering the effect of *K*, the least *K* gains the highest performance, i.e., almost 98% in comparison of just 94% with *K = 9*. It should be noted that the classification precision will be the same either CPU or GPU for our algorithm justification. In terms of computational time complexity, varying *K*s has no significantly effect on computational time complexity, i.e., around 32 seconds for GPU and 1040 seconds for CPU. It is worth noting that with GPU implementation, the speed-up is on the factor of three.

Table 4
CPU vs. GPU performance of KNN (K = 3 to 9).

CPU/ GPU	Metric	K=3	K=5	K=7	K=9
GPU	Accuracy (%)	97.82	96.44	95.13	93.88
	Time (sec.)	31.89	32.22	32.35	32.67
CPU	Accuracy (%)	97.82	96.44	95.13	93.88
	Time (sec.)	1030.38	1048.38	1058.07	1062.66

Table 5 shows the second scenario results. In general, the performance precision has no significantly impact when varying the percentage for Neighborhood classification, i.e., around 99.10% to 99.30%. Similarly, the computational time has no significantly effect on the percentages, i.e., around 33 to 37 seconds for GPU and 1112 to 1116 seconds for CPU. However, again, the speed-up of GPU is over the factor of three.

Table 5
CPU vs. GPU performance of Neighborhood (10% to 90%)

CPU/ GPU	Metric	10%	30%	50%	70%	90%
GPU	Accuracy (%)	99. 26	99.28	99.30	99.27	99.10
	Time (sec.)	33.85	33.89	33.80	36.00	37.27
CPU	Accuracy (%)	99.26	99.28	99.30	99.27	99.10
	Time (sec.)	1116.28	1112.36	1112.30	1114.81	1114.78

It was noticed that when comparing Tables 4 and 5, the precision gain of Neighborhood is higher than that of KNN, i.e., around 99% vs. 95% for KNN as in average. The computational time complexity trade-off has in-significant effected, i.e., 34 seconds vs. 32 seconds with GPU. However, with CPU, Neighborhood can result in higher computational time, i.e., more than 1110 seconds vs. just 1050 seconds; however, again, Neighborhood with GPU implementation still maintains the outstanding result, and can be used as the candidate of IDS classifications.

V. Conclusion and Future Work

Among various data mining techniques using for classifications, especially applying for detection the networking attack based on IDS (KDD CUP 199), k–Nearest–Neighbour (KNN) is one of the promising approaches. However, with a very large scale trace, one of the key limitations of traditional serial computation (CPU) is reached, and so, this research then investigates an alternate approach by integrating the parallel computation using GPU based on NVIDIA CUDA framework.

In addition to KNN and its computational enhancement, the other candidate was also investigated, i.e., Neighborhood, and then, again, with the improvement of its traditional computation with parallel processing which turns to the outstanding parallel classification algorithm of Neighborhood, i.e., the precision gain is at 99% with only around 34 seconds for computational time.

It should be noted that the comparative results discussed in this paper can be used as the baseline for further investigation. However, more analyses and classification selections should be well investigated, i.e., various datasets including recent attacks with heterogeneous numbers of traffic patterns, and advanced classification techniques, and these are left for future work.

REFERENCES

[1] Atzori, L. Iera, A. and Morabito, G. 2010. The Internet of Things: A survey. *Computer Networks*. 54(15): 2787–2805.

[2] Mukherjee, B. Heberlein, L.T. and Levitt, K.N. 1994. Network intrusion detection. *IEEE Network*. 8(3): 26–41.

[3] Chen, P.Y. Cheng, S.M. and Chen, K.C. 2014. Information Fusion to Defend Intentional Attack in Internet of Things. *IEEE Internet of Things Journal*. 1(4): 337–348.

[4] Symantec Corporation. 2014. INTERNET SECURITY THREAT REPORT 2014. 19. [www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf]

[5] McAfee. 2014. Net Losses: Estimating the Global Cost of Cybercrime. [www.mcafee.com/mx/resources/reports/rp-economic-impact-cybercrime2.pdf]

[6] Butun, I. Morgera, S.D. and Sankar, R. 2014. A Survey of Intrusion Detection Systems in Wireless Sensor Networks. *IEEE Communication Surveys & Tutorials*. 16(1): 266–282.

[7] Liao, H.J. Lin, C.H.R. Lin, Y.C. and Tung, K.Y. 2013. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*. 36(1): 16–24.

[8] Julisch, K. 2002. Data Mining for Intrusion Detection. *Application of Data Mining in Computer Security, Advances in Information Security*. 6: 33–62.

[9] Helali, R.G.M. 2010. Data Mining Based Network Intrusion Detection System: A Survey. *Novel Algorithms and Techniques in Telecommunications and Networking*. 501–505.

[10] Vaarandi, R. 2009. Real-time classification of IDS alerts with data mining techniques. *Proc. IEEE International Conference on Military Communications*. 1–7.

[11] Gianfelici, F. Turchetti, C. and Crippa, P. 2007. Efficient Classification of Chaotic Signals with Application to Secure Communications. *Proc. IEEE International Conference on on Acoustics, Speech, and Signal Processing*. 1073–1076.

[12] Wu, X. Kumar, V. Quinlan, J.R. Ghosh, J. Yang, Q. Motoda, H. McLachlan, G.J. Ng, A. Liu, B. Yu, P.S. Zhou, Z. Steinbach, M. Hand, D.J. and Steinberg, D. 2008. Top 10 algorithms in data mining. *Knowledge and Information Systems Journal*. 14(1): 1–37.

[13] Wagh, S.K. Pachghare, V.K. and Kolhe, S.R. 2013. Survey on Intrusion Detection System using Machine Learning Techniques. *International Journal of Computer Applications*, 78(16): 30–37.

[14] So-In, C. Mongkonchai, N. Aimtongkham, P. Wijitsopon, K. and Rujirakul, K. 2014. An Evaluation of Data Mining Classification Models for Network Intrusion Detection. *Proc. International Conference on Digital Information and Communication Technology and its Applications*. 90–94.

[15] Bhatia, N. and Vandana. 2010. Survey of Nearest Neighbor Techniques. *International Journal of Computer Science and Information Security*. 8(2): 302–305.

[16] Hu, Q. Yu, D. and Xie, Z. 2008. Neighborhood classifiers. *Expert Systems with Applications*. 34(2): 886–876.

[17] Kirk, D.B. and Hwu, W.W. 2010. Programming Massively Parallel Processors: A Hands-on Approach. *Morgan Kaufmann*. 280 pp.

[18] Wilt, N. 2013. CUDA Handbook: A Comprehensive Guide to GPU Programming. *Addison-Wesley Professional*. 528 pp.

[19] Navarro, C.A. Hitschfeld-kahler, N. and Mateu, L. 2014. A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures. *Communications in Computational Physics*. 15(2): 285–329.

[20] Shi, L. Liu, W. Zhang, H. Xie, Y. and Wang, D. 2012. A survey of GPU-based medical image computing techniques. *Quantitative Imaging in Medicine and Surgery*. 2(3): 188–206.

[21] So-In, C. Poolsanguan, S. Poonriboon, C. Rujirakul, K. and Phaudphut, C. 2013. Performance Evaluation of Parallel AES Implementations over CUDA GPU Framework. *International Journal of Digital Content Technology and its Applications*. 7(5): 501–511.

[22] Garcia, V. Debreuve, E. and Barlaud, M. 2008. Fast k nearest neighbor search using GPU. Proc. *IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 1–6.

[23] Kuang, Q. and Zhao, L. 2009. A Practical GPU Based KNN Algorithm. *Proc. Symposium International Computer Science and Computational Technology*. 151–155.

[24] Kikam, V.B. and Meshram, B.B. 2014. PARALLEL kNN ON GPU ARCHITECTURE USING OpenCL. *International Journal of Research in Engineering and Technology*. 3(10): 367–372.

[25] Patel, S. and Sondhi, J. 2014. A Review of Intrusion Detection Technique using Various Technique of Machine Learning and Feature Optimization Technique. *International Journal of Computer Applications*. 93(14): 43–47.

[26] Jian, L. Wang, C. Liu, Y. Liang, S. Yi, W. and Shi, Y. 2013. Parallel data mining techniques on Graphics Processing Unit with Compute Unified Device Architecture (CUDA). *The Journal of Supercomputing*. 64(3): 942–967.

[27] KDD CUP 1999 Data. [kdd.ics.uci.edu/databases/kddcup99/kddcup99.html].

[28] J. Schneider, "Cross Validation". [www.cs.cmu.edu/~schneide/tut5/node42.html].

[29] H. Hamilton, "Confusion Matrix". [www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html].

[30] Geforce GTX 750 TI Dataset. [www.nvidia.com/gtx-700-graphics-cards/gtx-750ti/].