University of Peradeniya

Sri Lanka

Proceedings of the Fourth Engineering Students' Conference at Peradeniya (ESCaPe) 2016

Department of Computer Engineering
Faculty of Engineering, University of Peradeniya
14 September 2016

Technical Program Committee

Dr. J. Alawatugoda
Dr. A. Bandaranayake
Dr. S.D. Dewasurendra
Mr. D.S. Deegalla
Dr. D. Elkaduwe
Mr. H. Gamaarachchi
Ms. D. Herath
Dr. N. Karunarathne
Dr. R. Krishanthmohan
Mr. Z. Maraikar
Dr. I. Nawinne
Dr. S. Radhakrishnan
Dr. R. G. Ragel (Chairperson)
Dr. K. Samarakoon
Dr. M. Sandirigama

In charge of Posters

Dr. N. Karunarathne

Proceedings Layout and Arrangement

Dr. J. Alawatugoda

Cover-page Design

Mr. A.R.S.P. Rodrigo

Contents

Message from the Dean, Faculty of Engineering…….……iv
Message from the Head, Department of Computer Engineering……..…v
Message from the President, Association of Computer Engineering Students (ACES)……..vi

Section I: Data Mining and Machine Learning

Cricket Sabermetrics: A Data Mining Analysis of Cricket……..……..1
Comparative Analysis of Machine Learning Methods in Anomaly-based Intrusion Detection……5

Section II: Communication and Networking

Implementation of Stateful SDN Firewall……..…….11
SDN Flow Caching……...…15

Section III: Internet of Things

Affordable Environmental Monitoring and Controlling System for Greenhouses……..20
Smart Campus Phase One: Smart Parking Sensor Network……25

Section IV: High Performance Computing

Accelerating k-NN Classification Algorithm Using Graphics Processing Unit (GPU)……30
Accelerating Mutual Information Analysis Based Power Analysis Attacks using the GPU…….34
A Faster GPU Implementation of Breadth-First Traversal for EDA Circuit Graphs……..…..….40
Accelerating losses compression with GPU……..……46

Message from the Dean, Faculty of Engineering

It's my great pleasure to write this message for the Students' Symposium of the Department of Computer Engineering, Faculty of Engineering, University of Peradeniya.

At a time when the local engineering ad IT industries expect higher education institutions to produce graduates who possess attributes of critical thinking, analytical skills and sufficient hands on exposure, to employ fresh graduates directly as engineers with minimum training on the job, the academic programmes are indirectly made responsible to fill the void of providing suitable hands on training. In addition, providing opportunities to the students to use skills and creativity, individually as well as in groups, is a part of the requirements of the Washington accord accreditation that recognize our degree on global platforms.

In the face of the above change of approach in the industry and in the curricula design, the students' research project component in the third and final year curricula, with a significant involvement on the part of the students, requires utilization of knowledge acquired in the academic programme, develop a strong sense of creativity and team work. This help students, to a greater extent, to acquire certain degree of hands on exposure, and to prepare for the current demands in the industry through the exercise.

With such a background, the Computer Engineering department of the Faculty conducts its annual students' symposium, ESCaPe 2016, in this year too. The symposium is aimed at recognizing undergraduate research and consolidating a research culture among undergraduates in the Faculty, while paving way for national contribution through research. I believe that the research publication will serve as a good record of the outcomes of students' research projects and also will be a testimony of their teamwork.

I would like to take this opportunity to thank the Department of Computer Engineering for making this effort to continue the symposium annually and thank the staff of the Department for their hard work in facilitating and encouraging the students to conduct useful projects that finally make valuable contribution to the IT and Computer engineering industries.

Prof. Leelananda Rajapaksha

Dean, Faculty of Engineering
University of Peradeniya

Message from the Head, Department of Computer Engineering

It gives me great pleasure to pen this message to ESCaPe16 – Engineering Student Conference at Peradeniya, which is ironically held in Colombo. The conference showcases final year projects of the Undergraduates, who are following the Computer Engineering Specialization at the Department of Computer Engineering, Faculty of Engineering, University of Peradeniya.

In 2015, the Department took a strategical decision to move the conference, which was previously held in Peradeniya, to Colombo for the obvious reason of bringing the conference closer to where the industry is based. So here we are in Colombo for the second consecutive year.

Research plays a significant role in our Undergraduate curriculum. Every student is expected to complete a research project in their final year. This is in addition to four other development projects they would do in their second and third years. An illustrative example of our inclination towards research is the NVIDIA research laboratory at the Department Computer Engineering – the only one of its kind in Sri Lanka.

It takes courage to do cutting edge research. When you are attempting what others have not, you are bound to fail; hours and even days of hard work might have to be thrown out, and you start again. I take this opportunity to say kudos to those students and the staff who put in lots of effort to bring the projects that would witness today to their current level.

In addition to the usual pressure of doing a research project, students were faced with another burden this year. To cover-up the time lost due to the recent non-academic staff trade union action, the Faculty decided to extend the timetable till 7:00 pm. Regular lecture and laboratory classes ran till 7:00 pm eating into the free time of students if they had any. I want to put on record the mammoth effort of both the staff and the students to bring these projects to the stage they are at today.

Last but not least, I would like to thank all those who are present today. Your participation today means a lot to us and is an immense encouragement to the budding Engineers.

Dr. Dhammika Elkaduwe

Head, Department of Computer Engineering
University of Peradeniya

Message from the President, Association of Computer Engineering Students (ACES)

How in heaven's name can a nation with a $1 trillion surplus threaten so much scientific research so vital to its future?
“I believe in innovation and that the way you get innovation is you fund research and you learn the basic facts. ” -Bill Gates

“By welcoming eager, talented workers, we expand America's potential for growth, and our competitive culture of invention and possibility” - Ron Conway

It gives me great pleasure to write this message on behalf of Association of Computer Engineering Students for ‘ESCaPe 2016’, the annual Project Symposium of Department of Computer Engineering, Faculty of Engineering, University of Peradeniya.

As Ron Conway says, The potential growth of America depends on new inventions, So why can’t we? Sri Lanka too have many hidden talents, whom are capable of doing some remarkable inventions.

As ACES, the student body of Department of Computer Engineering, we thought it was our responsibility to showcase, what our undergraduates are capable of to the Industry revolving in the nation's city of Colombo, in order to use them for the betterment of the country. ‘ESCaPe 2016’ is the result of that objective.

ESCaPe 2016 show cases several researches carried out by the final year undergraduates of Department of Computer Engineering. So what is presented today highlight not only the new invention they found, it also emphasizes the eager and commitment they put up to do these projects a success despite of their busy academic schedule.

As the president of ACES, I make this opportunity to congratulate all my colleagues for their success on their researches, and my heartfelt thank goes to all the lectures of Department of Computer Engineering, for guiding the students achieve this today, and to everyone in my team who worked day and night to make this event a success.

Mr. Supun Athukorala

President
ACES

Cricket Sabermetrics

A Data Mining Analysis of Cricket

P.A. Gregory, D.H.M.S.N. Herath, D.S.L. Karunasekara, S. Deegalla, A. Bandaranayake
Department of Computer Engineering
Faculty of Engineering, University of Peradeniya
Peradeniya 20400 Sri Lanka

***Abstract*— Where there is sports there is statistics and cricket is no exception to this. The game of cricket has a wide wealth of complex statistical data associated with the game. Analysts and fans throughout history have crunched the numbers, trying to make meaning out of this vast pool of data. The aim of this project is to attempt the same using data mining techniques to hopefully unearth key underlying data patterns that can be used to accurately model in-game performances.**

Keywords—Cricket; Data Mining; Statistics; Indian Premier League;

I. INTRODUCTION

Whether it is the sprinter who finished first or the team that scored more points, it's usually easy to determine who won a sporting event. But finding the statistics that explain why an athlete or a team wins is more difficult and major figures at the intersection of sports and numbers are determined to crack this problem.

One such attempt to make meaning out of statistics was seen in baseball. The aim was to objectively analyze the game of baseball by deviating from traditional performance measures. This approach now commonly known as Sabermetrics, often questioned traditional measures of baseball as they did not provide an accurate representation of in game performance [1]. Instead, Sabermetric researches used statistical analysis to determine performance metrics that actually contribute towards a team win.

Sabermetrics has transformed baseball. The question is, can it do the same for cricket? Cricket is a far more diverse sport spread across three major formats and played across the globe. But like baseball it has a wealth of statistics associated with the game. These traditional statistical measures of cricket are well established performance metrics commonly used to evaluate player performances. But like traditional baseball statistics, we believe they do not provide an accurate representation of in game performance.

This project aims to explore the statistical aspect of cricket in a selected domain, and realize key performance metrics that contribute towards the outcome of a cricket match.

II. LITERATURE REVIEW

Player classification using performance metrics always topped the priority list of researches irrespective of the sport under consideration. However in cricket, there hasn't been extensive research on performance-based player classification apart from the traditional measures such as averages, strike rates and economy rates. These existing player evaluation metrics in cricket are believed to be fundamentally flawed [2]. Alternative performance measure have been proposed, an example being a classification scheme developed for batmen using performance data of One Day International (ODI) matches and Test cricket [3][4]. The proposed method uses a single measure derived from a batsman's average, strike rate and batting consistency to evaluate performances.

The latest addition to cricket is the Twenty-20 format. The fast paced game of T20 has given rise to many lucrative domestic T20 competitions, such as the Indian Premier League (IPL). The IPL has emerged as a focal point for many different disciplines, from Economics and Finance to Statistics and Decision Science. With wide spread growth of T20 cricket, many new attempts to model player performances have been made. Reference [5] discusses the performance of players in the first T20 World Cup. All-rounder performance from the first edition of the IPL was evaluated using the terms batting all-rounder, bowling all-rounder and ideal all-rounder [6]. Even fantasy cricket has had its own share of player performance modelling where existing measures of player performances were used in a binary integer programming model to select players [7]. The idea is to use both static prediction and dynamic prediction to develop a dynamic prediction algorithm based on an aggregate of static predictions.

In 2010 Venky Mysore, was bought on board the IPL team Kolkata Knight Riders to aid the selection process during the player auction. Influenced by MoneyBall: The Art of Winning an Unfair Game, his strategy was to buy wins rather than players [8]. With various predictive analysis capabilities, KKR put together the winning team of IPL 2014 [9].

WASP - "Winning And Score Predictor" is a calculation tool used in cricket to predict scores and possible results of a limited overs match [10]. Cricket Australia's new statistical approach and "Key to Success" statistical analysis method are few examples of them. Since these examples are commercial products, the methods they used in developing these applications are not a matter of public record.

The main challenge lies in identifying performance metrics that actually matter. Past attempts at solving the same, have relied on expert domain knowledge which can introduce a potential bias to the final results. To eliminate this, our

approach uses data mining and machine learning techniques to identify patterns in data that could potentially provide us with an indication on key performance metrics that could be used for player evaluation.

III. METHODOLOGY

A. The Domain

As stated previously, cricket is a diverse sport. It is played globally across three different formats and as a result analyzing the game as a single entity is a difficult if not impossible task to accomplish. Therefore, the domain of choice for our problem is the Indian Premier League (IPL). The IPL being eight editions old, provides us with a decent sample set of data to perform our analysis.

B. Data Set

An exhaustive set of up-to-date statistical data for the IPL domain was obtained. The dataset contained complete statistical details of 501 instances of IPL matches. The data was parsed and stored in a database using an object to relational mapping framework. Classes were created following Object Oriented principles to manipulate the data as needed during feature construction.

C. Methodology Outline

An iterative approach was followed for the analysis. The analysis looped between developing the feature set and improving the data mining model. Two different approaches were used to improve the overall accuracy of the analysis. An outline of the approach is given below.

- Feature Set Development
- Feature Selection
- Modelling and Analysis

D. Feature Set

A main component of the analysis is the feature set. The analysis is only as good as the feature set that's fed into the model. Potentially any attribute that can be associated with a team innings can be used in the feature set. Initially a basic set of attributes was used, and as the analysis progressed a more complex set of features were developed to improve the accuracy of the model. Each attribute was built using various combinations of the basic units of information for an innings, namely runs, balls and wickets.

- Number of Wickets Lost (1)
- Four Hitting Frequency (2)
- Six Hitting Frequency (3)
- Boundary Run Percentage (4)
- Dot Ball Percentage (5)
- Dot Ball to Runs Ratio (6)
- Run Rate (7)
- Average Partnership Score (8)
- Number of Batting Segments (9)
- Batting Segment to Wicket Ratio (10)
- Average Runs in a Batting Segment (11)
- Average Pressure Factor (12)
- Pressure of Wickets (13)
- Final Score (14)

The attribute values were calculated for each match across the entire IPL domain. The resulting set of data contained 501 instances of the above attribute set (i.e.: for every match played in the IPL).

E. Feature Selection and Modelling Analysis

The idea behind the analysis was to predict the outcome of a match based on the feature set. The reasoning behind this was, if the prediction accuracy is high, the input feature set can be recognized as an accurate representation of in-game performance metrics.

Feature selection can be carried out using either Filter methods or Wrapper methods. The former is independent of a classifier and ranks the features according to a specific mathematical model. The latter on the other hand, generates subsets of the feature set and calculates the accuracies of each subset against a classification algorithm. Due to computational costs and almost similar results from both approaches, our analysis was carried out using Filter methods.

Three different attribute selection algorithms were tested on the feature set. The resulting subset of features were then fed into a classification model running the J48 decision tree algorithm using ten-fold cross validation. To improve the accuracy various attribute combinations of a given subset were also tested. The subset that provided the highest accuracy was selected as the ideal subset of attributes.

F. Innings Segmentation

Up until now, a single innings was treated as one complete segment. While this allowed us to identify feature impact throughout an innings, it did not provide us with information on the impact at different stages of an innings.

To address this, we proceeded to segment the innings into three main segments:

- Powerplay (1-6 overs)
- Middle (7-15 overs)
- Death (16- 20 overs)

The complete feature set was then calculated for each segment separately. That is, for each of the 501 instances in the dataset, the feature set was evaluated for the three segments. Initially, the individual predictive accuracy of each feature was recorded for each segment. (i.e.: Using a single attribute at a time on the classification model). The feature selection process described under subsection E was then

carried out to identify the optimal subset of features for each segment.

Finally, a combination of all the features (segments and complete innings) were analyzed using Wrapper methods.

IV. RESULTS

A. Feature Selection for Complete Innings

The classification model for the analysis used the J48 decision tree classifier. The feature selection process was carried out using three attribute selection algorithms.

- *CfsSubsetEval* – Selects attributes with high correlation with the class and low inter-correlation
- *InfoGainAttributeEval* – Selects attributes ranked according to information gain
- *ReliefFAttributeEval* – Selects attributes by repeated sampling

The results of feature selection carried out for the complete innings is shown in Table I and Table II. The accuracy improvements are shown in Table III and Table IV.

TABLE I. FIRST INNINGS FEATURE SELECTION

Algorithm	Optimum Feature Subset	Accuracy (%)
CfsSubsetEval	8,12,5,6,2	70.4591
InfoGainAttributeEval	6,7	70.2595
ReliefFAttributeEval	5,6,1	70.4591

TABLE II. SECOND INNINGS FEATURE SELECTION

Algorithm	Optimum Feature Subset	Accuracy (%)
CfsSubsetEval	11, 8, 1	88.4232
InfoGainAttributeEval	13	88.8224
ReliefFAttributeEval	8,10,9,1,13	88.0240

TABLE III. FIRST INNINGS PREDICTIVE ACCURACY

Attribute Selection	Accuracy(max)
Without feature selection	67.6647
With feature selection	70.4591

TABLE IV. SECOND INNINGS PREDICTIVE ACCURACY

Attribute Selection	Accuracy(max)
Without feature selection	86.0279
With feature selection	88.8224

Optimum subset of attributes:

First Innings: Dot Ball to Runs Ratio, Dot Ball Percentage, Number of Wickets Lost
Second Innings: Pressure of Losing Wickets

B. Innings Segmentation

Innings segmentation was done to analyze the feature impact at different stages of the match. This was carried out for the first innings. The innings was divided into three segments, namely Powerplay, Middle and Death. Table V shows the highest individual predictive accuracy of each attribute across all segments.

TABLE V. MAXIMUM PREDICTION ACCURACY OF INDIVIDUAL FEATURES

Attribute	Segment with Highest Accuracy	Accuracy (%)
Number of Wickets Lost	Powerplay	60.4790
Four Hitting Frequency	Complete	60.8782
Six Hitting Frequency	Complete	65.2695
Boundary Run Percentage	Complete	53.6926
Dot Ball Percentage	Complete	61.2774
Dot Ball to Runs Ratio	Complete	69.0619
Run Rate	Complete	71.0579
Average Partnership Score	Complete	64.0719
Number of Batting Segments	Complete	53.6926
Bat Segment to Wicket Ratio	Powerplay	60.0798
Avg Runs in Bat Segment	Complete	66.6667
Avg Pressure Factor	Complete	67.6647
Pressure of Wickets	Powerplay	58.6826
Final Score	Complete	71.4571

The optimum subset of features for each segment was then evaluated. This was done using both Filter methods and Wrapper methods with the J48 classification algorithm. The results are shown in Table VI.

TABLE VI. SEGMENT FEATURE SELECTION

Segment	Optimum Feature Subset	Accuracy (%)
Powerplay	1	60.4790
Middle	5,7	64.0719
Death	1,4,5,6,12	65.4691

The combined arff file containing all features (segment + complete) was analyzed using *WrapperSubsetEval*, a selection algorithm that generates random subsets and tests the accuracy against a classification algorithm returning the subset that gave the maximum accuracy. The results are shown in Table VI.

TABLE VII. COMBINED FEATURE SELECTION

Segment	Optimum Feature Subset
Complete	2,9,14
Powerplay	10,3
Middle	4,11,12,5
Death	-

The above subset of features showed a prediction accuracy of 71.6567 using the J48 decision tree algorithm.

Optimum subset of attributes:

First Innings: Four Frequency, Number of Batting Segments, Final Score, Batting Segment to Wicket Ratio (PP), Six Hitting Frequency (PP), Boundary Run Percentage (Middle), Average Runs in Batting Segment (Middle), Average Pressure Factor (Middle), Dot Ball Percentage (Middle), Run Rate (Middle)

I. CONCLUSION AND FUTURE WORK

The initial analysis provided some important distinctions between the first and second innings of a match. The feature selection process identified a different subset of attributes for the two innings. It can be concluded that there are different dynamics to a chase. This is obvious when the resultant subset

of features are analyzed. While wicket loss plays a major part in the result of a chase, it is not as important while setting targets. Similarly, dot ball percentages seem to play a bigger role during the first phase of the match when compared to the second.

The segmentation process resulted in a different subset of features for each segment. For example, according to the above analysis, the most important feature during the powerplays is wickets lost. For the middle overs, dot balls and runs scored has a greater impact. During the death overs though, there is no clear feature that stands out. Rather a combination of different features seem to decide the outcome of a match. This proves that the importance of an attribute varies throughout an innings.

Finally, the combination of features from all segments provided us with the highest predictive accuracy. The combined subset of features contains attributes from the complete analysis as well as the segmentation analysis.

This analysis is by no means complete at this point. Future work would involve exploring new options to improve the model accuracy. One such option is using ensembling, a process that attempts to increase the predictive accuracy of a model by combining different models together. Attention should also be paid to the feature set, since the model is only as good as the feature set. Therefore, future development work will follow an iterative approach between developing the feature set and improving the model.

REFERENCES

[1] Wikipedia, Sabermetrics (2016, August) [Online]. Available: https://en.wikipedia.org/wiki/Sabermetrics

[2] Lewis, A J (2005). "Towards Fairer Measures of Player Performance in One-Day Cricket," The Journal of the Operational Research Society, 56(7), 804- 815.

[3] Lemmer, H (2004). "A Measure for the Batting Performance of Cricket Players," South African Journal for Research in Sport, Physical Education and Recreation, 26(1), 55-64.

[4] [4] Lemmer, H (2006). "A Measure of the Current Bowling Performance in Cricket," South African Journal for Research in Sport, Physical Education and Recreation, 28(2), 91-103.

[5] Lemmer, H (2008). "An Analysis of Players' Performances in the First Cricket Twenty20 World Cup Series," South African Journal for Research in Sport, Physical Education and Recreation, 30(2), 71-77.

[6] Van Staden, P J (2008). "Comparison of Bowlers, Batsmen and All-rounders in Cricket Using Graphical Display," Technical Report 08/01, Department of Statistics, University of Pretoria, South Africa.

[7] Brettenny, W (2010). Integer Optimization for the Selection of a Fantasy League Cricket Team, Unpublished M.Sc Dissertation, Faculty of Science, Nelson Mandela Metropolitan University, South Africa.

[8] How KKR got it right – starsports.com. (2016, August). [Online]. Available: http:/www.starsports.com/cricket/articles/article=how-kkr-got-right/index.html

[9] How SAP Helped KKR Win Pepsi IPL 2014 | cioandleader.com. (2016). [online] Cioandleader.com. Available at: http://www.cioandleader.com/articles/40155/how-saphelped-kkr-win-pepsi-ipl-2014 [Accessed 1 Apr. 2016].

[10] WASP (cricket calculation tool). (2016). [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/WASP_(cricket_calculation_tool) [Accessed 1 Apr. 2016].

Comparative Analysis of Machine Learning Methods in Anomaly-based Intrusion Detection

W.D.Y.N. Piyasinghe, K.E.G.A.P. Kadurugasyaya, K.P.D.H.M. Karunatissa,
S.L.P. Yasakethu, R.G. Ragel
Faculty of Engineering
University of Peradeniya
Peradeniya,Sri Lanka
{ yniluka, anuradhe.prabhath, harsha.mshan, lasithkethu }@gmail.com, roshanr@pdn.ac.lk

***Abstract* - Intrusions can put computer systems to a vulnerable state by compromising overall security. A lot of research studies have been done in anomaly-based intrusion detection methods using machine learning based techniques. Most of such methods can only detect known attacks while some of these algorithms have the capability of identifying novel attacks. But still there is an imbalance between the extensive amount of research on this domain and operational deployments of such systems due to the high false positive rate. In our study, we propose a comparative performance analysis of supervised, semi-supervised and unsupervised learning methods for the first time in intrusion detection domain by using several learning algorithms.**

Keywords – Anomaly-based intrusion detection, Supervised Learning, Semi-Supervised Learning, Unsupervised Learning

I. INTRODUCTION

Cyber intrusion is any unauthorized access to a network, which makes connected computers vulnerable by compromising network security and stability. Intrusion detection techniques are classified according to detection principles as misuse/signature detection, anomaly detection, and hybrid detection [1]. Signature detection compares similarities between events and signatures of known attacks. Therefore, it has lower false positive rate when detecting known attacks but fails in detecting new attacks. Anomaly detection builds a model of usage patterns describing the typical behavior of the data at the initial phase. Then new events are compared with this model, and deviations are detected [2] and flagged as attacks. This approach has the advantage of detecting previously unknown attacks but suffers from a high rate of false alarms. Hybrid detection is a combination of anomaly-based and signature-based detection to overcome drawbacks of the above mentioned methods.

Machine learning techniques are essential in building classifier models and detecting intrusions in anomaly detection systems. Supervised machine learning methods need labeled training data set for the above task, but this kind of data is difficult to obtain in real network environments since it is a time-consuming procedure. Also, there is no possible way to guarantee that the labeled data covers all possible attacks in network environment due to the dynamic behavior of network traffic [1]. Unsupervised learning methods can overcome drawbacks of supervised learning methods since there is no need for a labeled training dataset. Previous research works show that the supervised learning has high false positive rate if unknown attacks are present in the testing dataset. Unsupervised methods do not show a significant difference between known and unknown attacks but have low performance [3]. To overcome the limitations of the above methods, the recent intrusion detection research is more focused towards semi-supervised learning methods. Semi-supervised learning uses a small amount of labeled data with a large number of unlabeled data for building classifiers [4] [5].

The objective of this study is to compare the performance of supervised, semi-supervised and unsupervised machine learning techniques in anomaly based intrusion detection. For this study, we will use algorithms which have shown promising results in previous research. Further, we will select the most efficient algorithm out of the selected algorithms considering the detection accuracy and complexity. Our experiment is based on KDD Cup 1999 dataset [6]. Receiver operator characteristic (ROC) curves are used for comparing results after algorithms have been tested in a common testing framework.

II. RELATED WORK

Laskov et al. [1] have done an evaluation of several supervised and unsupervised algorithms using KDD Cup 1999 dataset. The experiment was done in two scenarios. In the first scenario, both training and testing data come from the same data distribution. Therefore only known attacks were included in the testing dataset. In the second scenario some of the attacks in the testing data are not presented in training dataset (unknown attacks for testing). Results showed that the supervised learning methods outperform unsupervised learning methods if testing data contains known attacks. The accuracy of supervised methods has a lower value if testing data includes unknown attacks. Unsupervised methods have similar performance when detecting both known/unknown

attacks. Accuracy values were alike to the performance of detecting unknown attacks in supervised learning methods. Also, non-linear methods (such as Support Vector Machines (SVM) and rule-based methods) have the best performance among selected supervised learning methods.

Zhang et al. [7] have done a study using random forest algorithm to detect intrusions without using attack-free data in the training phase. They built a framework to analyze the performance of random forest in misuse, anomaly, and hybrid detection systems and especially used outlier detection of random forest algorithm in anomaly detection framework. Sampling techniques such as cross-validation are used to increase the detection rate. The results showed that their approach achieved higher detection rate when the false positive rate is low, compared to previous unsupervised anomaly detection methods. The results also demonstrated a decrease in the detection performance when the amount of attack data is increased in the testing dataset.

Another research [8] with more focus towards comparing performances of supervised methods in intrusion detection is done against KDD Cup 1999 dataset. Support vector machine (SVM) and Neural Network (NN) algorithms are used for the experiment and performance were compared using different measures such as accuracy and false positive rate in the optimum level of those algorithms. Since data is not distributed evenly in the KDD dataset, they applied different normal data proportions for training and testing phases and obtained one average value for comparison. It has shown that SVM has superior results than NN for accuracy and false positive rate.

But there is no previous study done in comparative analyze of the performance of supervised, semi-supervised and unsupervised machine learning techniques in anomaly based intrusion detection. We focus on addressing this in our research.

III. BACKGROUND: MACHINE LEARNING METHODS

Machine learning algorithms used for our experiment are described in the following section.

A. Supervised Learning Algorithms

Support Vector Machine (SVM)

A Support Vector Machine can be used for both classification and regression purposes. The goal of SVM is to find a hyperplane that would leave the maximum distance between data points from two classes. As shown in Fig 1 hyperplane can be written as the set of points x satisfying $w^Tx + b = 0$, where the vector w is a normal vector perpendicular to the hyperplane and b is the offset of the hyperplane $w^Tx + b = 0$ from the original point along the direction of w.

Both nonlinear and linear SVM map the original feature space to a higher-dimensional feature space where the training set is separable by using kernel functions. Selecting a kernel function and tuning parameters for SVM are still trial-and-error procedure.

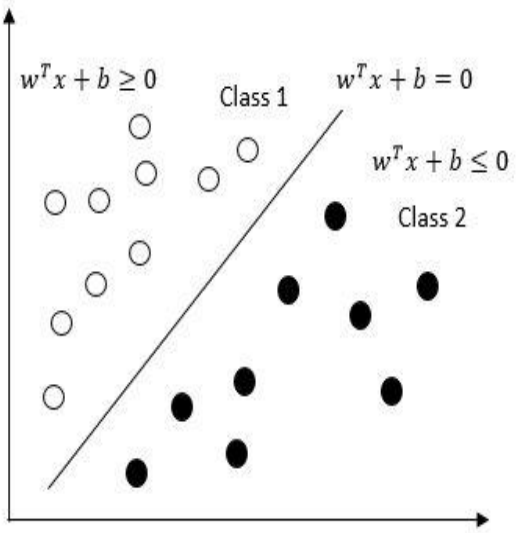

Fig. 1 Hyperplane through linearly separable classes

K-Nearest Neighbors (KNN)

K-Nearest Neighbours algorithm is a non-parametric method, which can be used for both classification and regression. Input for the algorithm is k closet training data points in the feature space. In classification, the output is a class membership, which is obtained by a majority vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbours. Different distance functions are used to calculate the distance from a particular data point to its neighbours. The optimal value of k depends upon the characteristics of data. In general, larger values of k reduce the effect of noise on the classification, but make boundaries between classes less distinct. Heuristic techniques are used to find an adequate k value.

B. Semi-Supervised Learning Algorithms

One-class Support Vector Machine

One-class SVM needs to have data related to only one class for building a strong decision boundary which differentiates a particular data class from other data type. This algorithm obtains a spherical boundary, in feature space, around the data. The volume of this hypersphere is minimized, to minimize the effect of incorporating outliers in the solution. The resulting hypersphere is characterized by a center and a radius R>0 as distance from the center to (any support vector on) the boundary, of which the volume will be minimized. As shown in Fig 2, this hypersphere is used as classification

boundary for identify new data as similar or different to the training set.

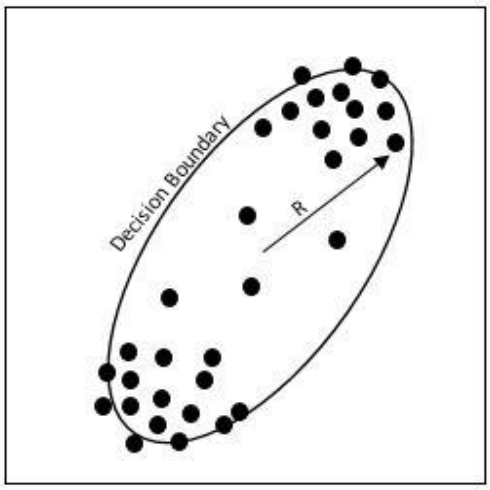

Fig. 2 Hypersphere defines the decision boundary for outliers

YATSI

YATSI classifier uses both labeled and unlabeled data in a two-stage setup for training. In the first stage, a standard classifier is trained on the available labeled training data. In the second stage, the model generated from the learning data is used to pre label all the unlabeled training data. Then these pre-labeled data are used together with labeled original training data in a weighted nearest neighbor algorithm for constantly improvement of the classifier.

C. Unsupervised Learning Algorithms

k-Means

k-Means clustering partitions the given data points into k clusters, in which each data point is more similar to its cluster centroid than to the other cluster centroids. k-Means initially creates k clusters from a set of objects so that the members of a group are more similar than in between groups. Then new objects are assigned to the cluster which is closer to the centroid using different distance metrics. The process is repeated until cluster centroids do not change. Accuracy is depended on distance metrics and cluster number k for partitioning. No evaluation method can guarantee optimal value for k.

Expectation Maximization (EM)

The EM algorithm is an iterative procedure to compute the Maximum Likelihood (ML) estimate in the presence of missing or hidden data. Each iteration of the EM algorithm consists of two processes, The E-step and the M-step. In E-step, the missing data are estimated given the observed data and current estimate of the model parameters. This is achieved using the conditional expectation, explaining the choice of terminology. In the M-step, the likelihood function is maximized under the assumption that the missing data are known. The estimate of the missing data from the E-step is used for the actual missing data. Steps are repeated until convergence.

IV. INTRUSION DATASET AND DATA PREPROCESSING

Our experiment is based on KDD Cup 1999 dataset [6], which was built using data captured in DARPA'98 IDS evaluation programme. In this dataset, a total of nine weeks raw Transmission Control Protocol (TCP) dump data was split into a seven-week training data set and remaining two weeks were turned into test data [9]. It consists of nearly 4.9 million single connection vectors which contain 41 features and labelled as normal or as an attack by specifying attack name [10]. Attacks fall into one of following categories: DoS (Denial of service), R2L (Remote to Local) – unauthorized access to a local machine from a remote system, U2R (User to Root) – Root level privileges for a normal user and Probing. There is a total of 24 attack types contain in the dataset [11]. Due to difficulties in processing such a large amount of data, we have used a kddcup_data_10_percent dataset [6] which only consists of nearly 0.49 million records.

Some characteristic of this dataset eventually leads to problems in applying machine learning algorithms for training. Certain attributes have nominal values which have to be converted into numeric representation before feeding into some algorithms. First, each nominal feature which has n possible classes was converted into binary representation by mapping i^{th} class value of the feature to j^{th} component as 1 and other positions as 0. Then the binary representation was converted into corresponding decimal value.

$$b(i) = (0,..,1,..,j,.); i \in n, j = \begin{cases} 1; if\ i^{th} class = j^{th} component \\ 0; if\ j^{th} class \neq j^{th} component \end{cases} \quad (01)$$

The dataset also has to be normalized for making all attributes proportion with one another. This will eliminate the influence of one feature over another since most of the features in the dataset have uneven distribution [9]. We used standard score formula for data normalization.

$$n(x_i) = \frac{x_i - \mu}{\sigma}\ ;\ \mu = Mean, \sigma = StandardDeviation \quad (02)$$

The dataset also has an irregular class distribution [9], which means an imbalance between normal and attack data. If classification algorithm is not robust enough for supervised learning, this imbalanced behavior leads classifier's more bias towards the majority class [12]. Therefore, the overall intrusion detection accuracy might be compromised.

Also, previous works suggest that all 41 attributes are important for identifying attacks and therefore reducing dataset based on features will result in low probability of recognizing certain attack types [13].

By considering above mentioned criteria, we build 4 types of datasets using KDD data for our experiment.

1. Training dataset for supervised learning
2. Imbalanced training dataset for unsupervised learning
3. Training dataset for semi-supervised learning
4. Testing datasets

Testing datasets were prepared to analyze the detection accuracy of previously known attacks and unknown attacks. The known attack dataset contained all the 14 different attack types which are already contained in the training datasets whereas unknown dataset contained 8 more novel attacks.

IV. Implementation

Weka machine learning environment [14] was used with some external Java based libraries for the implementation purposes.Weka has built in capability to generate receiver operating characteristic (ROC) curves for defined threshold values. Data pre-processing methods were implemented using the Java programming language.

A. Classifier models in supervised and semi-supervised learning

Separate datasets were prepared for training supervised algorithms and semi-supervised algorithms due to different requirement in Weka for the above two learning methods. Classifier models obtained by optimizing parameters of each algorithm for highest classification accuracy. 10 fold cross-validation method also used to avoid over-fitting of the classifier to training dataset.

B. Labeled clusters in unsupervised learning

Since unlabeled data is fed into algorithms in unsupervised learning, there is no explicit method to determine whether resulting clusters belongs to normal or attacks unless imbalanced dataset is used for training [18]. Thus, we labeled clusters implicitly by using the imbalanced dataset in the training phase and then testing data categorized according to the shortest Euclidian distance to any labeled cluster.

V. Results and Evaluation.

Precision (03), Recall (04) and F-Score (05) values for intrusion detection were formulated using the confusion matrix shown in Table I.

TABLE I
CONFUSION MATRIX FOR INTRUSION DETECTION

	Classified as Normal	Classified as Attack
Actual Normal	TP	FP
Actual Attack	FN	TN

TP – Classified as Normal while they actually were Normal

TN – Classified as Attack while they actually were Attack

FP – Classified as Attack while they actually were Normal

FN – Classified as Normal while they actually were Attack

$$Precision = \frac{TP}{TP + FP} \quad (03)$$

$$Recall = \frac{TP}{TP + FN} \quad (04)$$

$$F - Score = 2.\frac{Precision.Recall}{Precision + Recall} \quad (05)$$

Also performance of each machine learning method was evaluated by plotting the receiver operating characteristic (ROC) curve. It draws true positive rate of detection (TPR) against the false positive rate of detection (FPR) for various threshold values. Total area under the curve summarizes the overall detection accuracy of the intrusion detector [19]. Hence it can be used as a performance measuring parameter.

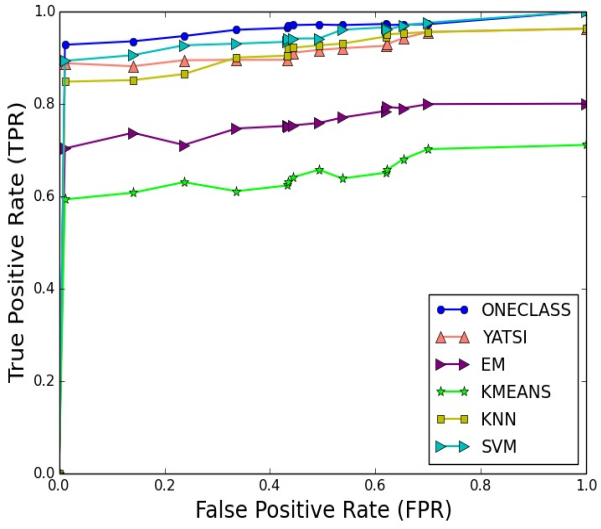

Fig. 3 ROC-curves obtained with the different machine learning methods on testing datasets with known attacks.

As shown in Fig 3, One-class SVM has the highest accuracy followed by SVM, YATSI and KNN algorithms

when detecting known attacks. But EM and k-Means algorithms have significant lower variation of detection accuracy. It depicts that both supervised (SVM, KNN), and semi-supervised (One-class SVM, YATSI) learning methods show very high accuracy than unsupervised methods (k-Means, EM) for detecting known attacks. Table II shows Precision, Recall and F-Score values for detecting known attacks by different algorithms.

TABLE II
PERFORMANCE METRICS FOR DETECTING KNOWN ATTACKS

Algorithm	Precision (%)	Recall (%)	F-Score (%)
SVM	92.8	90.7	91.7
KNN	96.2	91.7	93.9
EM	89.4	53.9	67.3
KMeans	92.9	50.1	65.1
one-class SVM	92.8	99.8	96.3
YATSI	95.0	96.9	95.9

According to Fig 4 unsupervised learning methods (k-Means, EM) show the highest accuracy for detecting unknown attacks followed by semi-supervised methods (One-class SVM, YATSI) in the second. Supervised learning methods (SVM, KNN) are not suitable in this scenario due to very low accuracy rates. The main advantage of unsupervised learning is that there is no need of labelled training data to build clusters. Variations in attributes distribution in the dataset can be used to build different clusters without using the class label. Hence, there is no significant accuracy variation for known or unknown testing data. But the quality of the training dataset directly affects the accuracy. Table III shows Precision, Recall and F-Score values for detecting unknown attacks by different algorithms.

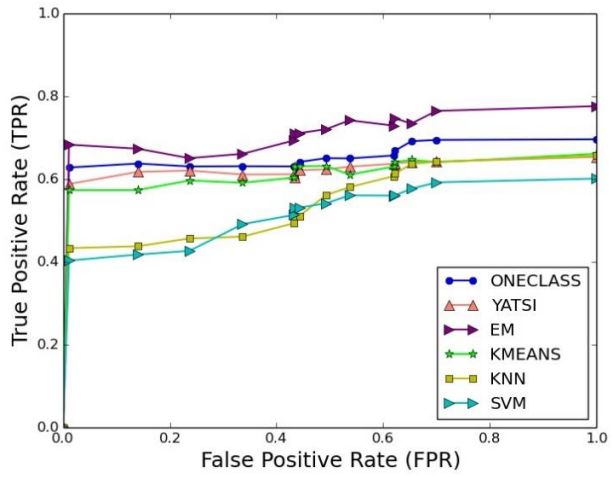

Fig. 4 ROC-curves obtained with the different machine learning methods on testing datasets with unknown attacks.

TABLE III
PERFORMANCE METRICS FOR DETECTING UNKNOWN ATTACKS

Algorithm	Precision (%)	Recall (%)	F-Score (%)
SVM	79.1	44.9	57.3
KNN	68.2	39.0	49.6
EM	81.1	69.7	75.0
KMeans	91.8	53.6	67.7
1-class SVM	89.3	59.6	71.5
YATSI	84.5	51.0	63.6

Table IV shows total detection accuracy of algorithms for known and unknown attacks. It shows that One-class SVM algorithm (semi-supervised) has highest ROC area when detecting known attacks while having a reasonable value for detecting unknown attacks. One-class SVM algorithm is trained using a dataset contain only one label. In this scenario, a dataset only containing normal data was used to build a classifier model. One-class SVM has the capability to build strong boundaries about standard data which will be used to classify other class types as anomalies.

TABLE IV
ROC AREA OF DIFFERENT ALGORITHMS

Testing Dataset	Algorithm	ROC Area(Accuracy) (%)
Known Data	SVM	95.1
	KNN	93.1
	EM	78.4
	KMeans	68.0
	One-class SVM	97.9
	YATSI	97.0
Unknown Data	SVM	54.1
	KNN	58.3
	EM	71.1
	KMeans	62.4
	One-class SVM	65.1
	YATSI	63.7

In supervised learning (SVM, KNN), the classification model is built based on different class labels of the dataset. If classifier's parameters are tuned into an optimum level, then the model has the capability of classifying known testing data with very high accuracy using previous knowledge. But accuracy rate drastically reduces for unknown test data since the supervised model does not have a prior knowledge about untrained attack types. Therefore supervised learning algorithms show high accuracy in detecting known attacks while have very low accuracy in detecting novel attacks.

V. Conclusions

In this study, we have presented an evaluation of different supervised, semi-supervised and unsupervised learning methods in intrusion detection. It demonstrates that semi-supervised and supervised learning methods have higher performance than unsupervised methods when test data does not contain unknown attacks. The detection rate of supervised methods significantly drops when testing for unknown attacks. The accuracy of unsupervised learning is not reduced by unknown attacks and it has very similar performance to semi-supervised learning methods. Semi-supervised learning methods show promising results for any type of test data. Experiment suggests that the semi-supervised learning methods show promising results in intrusion detection domain than both supervised and unsupervised learning methods. Also, One-class SVM semi-supervised algorithm has the highest overall accuracy of the experimented machine learning methods.

References

[1] Pavel Laskov, Patrick Dussel, Christin Schafer and Konrad Rieck, "Learning intrusion detection:supervised or unsupervised?," Image Analysis and Processing – ICIAP 2005, vol. 3617, pp. 50 - 57, 2005.

[2] Gustavo Nascimento, Miguel Correia, "Anomaly-based Intrusion Detection in Software as a Service," DSNW '11 Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops, pp. 19-24, 2011.

[3] C.Chen,Yunchao Gong,Yingjie Tian, "Semi-supervised learning methods for network intrusion detection," in Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on , Singapore, 2008.

[4] Jyoti Haweliya,Bhawna Nigam, "Network Intrusion Detection using Semi Supervised Support Vector Machine," International Journal of Computer Applications , vol. 85, 2014.

[5] X. Zhu, "Semi Supervised Learning Literature Survey," Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.

[6] "KDD Cup 1999 Data," 28 October 1999. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. [Accessed 01 April 2016].

[7] Jiong Zhang, Mohammad Zulkernine, Anwar Haque, "Random-Forests-Based Network Intrusion Detection Systems," IEEE Transactions on Systems, Man, and Cybernetics, vol. 38, no. 5, pp. 649 - 659 , 2008.

[8] Hua TANG,Zhuolin CAO, "Machine Learning-based Intrusion Detection Algorithms," Journal of Computational Information Systems, vol. 5, no. 6, pp. 1825-1831, 2009.

[9] Bertrand Portier, Jerome Froment-Curtil, "Data Mining Techniques for Intrusion Detection," in the University of Texas, Austin, 2000.

[10] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu and Ali A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," in Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium, Ottawa, 2009.

[11] L. Portnoy, "Intrusion detection with unlabeled data using clustering," Columbia University.

[12] Sumeet Dua,Xian Du, "Modeling Data with Skewed Class Distributions to Handle Rare Event Detection," in Data Mining and Machine Learning in Cybersecurity, Auerbach, 2011, p. 52.

[13] H. Gunes Kayacik, A. Nur Zincir-Heywood, Malcolm I. Heywood, "Selecting Features for Intrusion Detection:A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets," in In Proceedings of the third annual conference on privacy, security and trust, 2005.

[14] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten, "The WEKA Data Mining Software: An Update," SIGKDD Explorations, vol. 11, no. 1, 2009.

[15] T. Subbulakshmi1,A. Ramamoorthi,Dr. S. Mercy Shalinie, "Ensemble design for intrusion detection systems," International Journal of Computer science & Information Technology(IJCSIT), vol. 1, no. 1, 2009.

[16] R. Kohavi, "Wrappers for Performance Enhancement and Oblivious Decision Graphs," Thesis (Ph. D.) - Stanford University, 1995.

[17] Kurt Driessens, Peter Reutemann, Bernhard Pfahringer, Claire Leschi, "Using weighted nearest neighbor to benefit from unlabeled data," Advances in Knowledge Discovery and Data Mining, vol. 3918, pp. 60 - 69, 2006.

[18] Sumeet Dua, Xian Du, "Clustering-Based Anomaly Detection," in Data Mining and Machine Learning in Cybersecurity, Auerbach, 2011, p. 102.

[19] R.A Maxion, R.R Roberts, "Proper Use of ROC Curves in Intrusion/Anomaly Detection," 2004.

[20] Y Wang, J. Wong, A. Miner, "Anomaly intrusion detection using one class SVM," Information Assurance Workshop, pp. 358-364, 2004.

Implementation of Stateful SDN Firewall

Madusha Jayawardhana, Ninada Perera, Priyankara Bandara, Suneth Namal and Sampath Deegalla
Department of Computer Engineering
University of Peradeniya, Sri Lanka
{e11487, e11297, e11040}@gmail.com, {namal, dsdeegalla}@pdn.ac.lk

***Abstract*— SDN Firewalls have become very prominent in modern networking security. Since Software Defined Networks separate control logic from network devices and act as a centralize controller, most of the traditional networking issues could be solved. Thus major disadvantages of traditional firewalls are overcome by SDN Firewalls. Moreover, SDN firewalls are more advanced in packet handling.**

Objective of this project is to implement a stateful firewall that runs on the Floodlight runtime. We develop a distributed flow-based firewall prototype and test it on a simulated network through Mininet. Also a data mining model was integrated to the firewall to automate the decision making process to identify network intrusion detection.

Keywords—SDN, Firewall, Data Mining, Network Security

I. INTRODUCTION

SDN service chaining uses software to insert services virtually into the flow of network traffic. A centralized controller can connect multiple network and security services in a series-or chain-across network devices. With SDN service chaining, networks can be reconfigured on the fly, allowing them to dynamically respond to the needs of the business. For both enterprises and service providers, this means that SDN service chaining will dramatically reduce the time, cost, and risk for them to design, test, and deliver new network and security services. Security service chaining for SDN has many benefits, such as the ability to elastically scale stateful firewalls that run as virtual machines. It is all based on need and dynamically adjustable as instances of services come and go. Scrub traffic, protecting various elements of the SDN, authenticate and ensure authorization of changes to the dynamic network, optimizing traffic flows, correlate network changes with security changes and Creating and associating new service classes are couple of requirements the uses may come across in daily basis.

The progression of secure SDN will entail taking out the stateless Access Control Lists (ACLs) and providing better protection through stateful inspection and all the other goodies that come with purpose-built virtualization security—including compliance, introspection, intrusion detection, etc. It will make for a more solid solution by ensuring that no one can disable the controller.

If every Ethernet switch in the environment could perform like a traditional firewall, it would change the way security policy is implemented in a networked environment. Assuming that every Ethernet switch was a multi-port firewall, and then firewall policies could be implemented throughout the network at every ingress switch port and on every link between switches. There would be firewalls for every server, desktop, every link. The policies for these firewalls would be implemented by a controller that maintains the global view of the current application traffic. Having security policies enforced throughout the environment would mean the complete erosion of the security perimeters. Having that many security policies implemented and maintained manually would be an administrative nightmare. However, with centralized controller architecture, the policy would be created once and then pushed down to every network device for enforcement.

The key concept to the feasibility of using an SDN-enabled switch as a firewall is the state that it would maintain of the application traffic flows. ACLs are not stateful and do not have awareness of when the connection started or ended. However, SDN switches as firewalls allow to create stateful firewalls. A firewall does not handle packets, but the flows. That is, the flow between a client and a server from start to the end of a Session. The firewall need to handle all flows in a session with stateful packet inspection. This allows to take the control over the packets being transmitted. Thereby the network will be able to self-manage the resources once traffic is configured at the start.

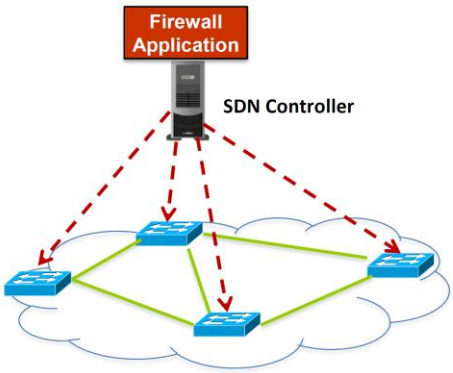

Figure 1.1. SDN firewall architecture

II. METHODOLOGY

The solution was to provide better security for the networks through SDN firewall by using stateful inspection and all the other good features that come with purpose-built virtualization security and thereby try to automate the process of SDN firewall decision making, done at the SDN controller using data mining.

The proposed solution was focusing on implementing a firewall module and run it on the SDN controller of the simulated network and thereby collect a set of sample data from the firewall information logs for further studies. The training data set obtained was to be used in data mining process in order to come up with an algorithm to automate the process of firewall configuration at the SDN controller.

First the stateful firewall should be implemented. There are several SDN controllers available such as POX, NOX, Beacon, Floodlight, Ryu, Trema and etc. NOX is the first SDN controller and POX which is an extended version of NOX is a python based controller[1]. The firewall application is coded with the same a programming language used for coding the controller. Floodlight[2] is a Java based SDN controller with built in stateless firewall application. Available SDN firewalls were only supporting rule based stateless firewall implementation. Our design was to create a stateful firewall on SDN controller. Figure 2.1 shows the overall architecture of the solution.

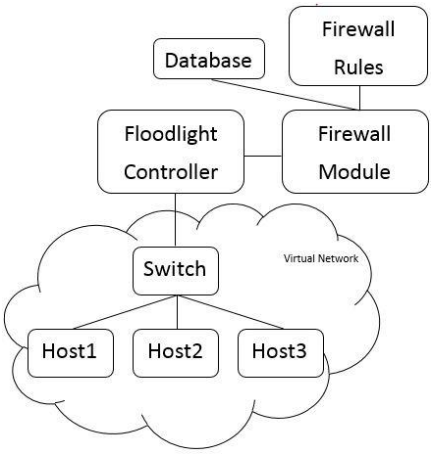

Figure 2.1 Overall architecture of the solution

The network which consists of switches running Openflow protocol is controlled by a centralized SDN controller. The stateful firewall module implemented, is residing in SDN controller, providing security for the network. SDN testbed was created using Mininet which was a system that provided scalable, virtual software defined networks for rapid prototyping on a single computer. OpenFlow was used as the standardized SDN protocol which defined a way for the controller to communicate with switches. OpenFlow provides the capabilities of centralized control of multiple switches from a single controller, dynamic forwarding information update and analysis of traffic statistics at switches [3].

FLoodlight controller was selected as the SDN controller due to its recognition for the features and performance. It is a java based controller which supports the Openflow versions from 1.0 to 1.4. Floodlight is considered as a robust controller which is easy to use. These were some reasons behind choosing Floodlight as the SDN controller.

Finally a simulated network is built on mininet network simulator and random network traffic is generated from hosts to the servers. Then some attributes of all the connections were stored to a database. Connection details stored on the database were then used to train & test a data mining model which is integrated to the firewall later. After the integration the firewall is able to identify any suspicious activity & alert the concerned parties.

III. Implementation

Before building the firewall, a virtual network should be simulated. Mininet is used to simulate the virtual network and Floodlight is used to create the controller. Then the firewall module can be integrated in to the firewall Foodlight controller. Next different network topologies were created using mininet. An example is given below. The network was created with 81 hosts & 40 OpenFlow switches connecting in tree topology which had a depth of 4 & branching factor of 3. And 5 of the nodes were configured as http servers.

```
sudo mn --topo tree,depth=4,fanout=3 --
controller=remote,ip=192.168.246.128,port=6653
--switch=ovsk,protocols=OpenFlow13
```

In basic packet filtering the firewalls does not keep track of the state of the ongoing connection sessions. This may easily lead to DoS attacks where external users can flood internal hosts with malicious packets. These firewalls are called stateless firewalls. But stateful firewalls address the problems of stateless firewalls by keeping track of ongoing connection sessions. Stateful TCP packet filtering is easier due to the presence of TCP flags (SYN, ACK, FIN, PSH, URG, and FIN). But tracking of UDP session state is a complicated process[4]. Because UDP is a connectionless transport protocol which does not contain any sequence numbers or flags like TCP. Therefore, we planned to implement the stateful firewall only for TCP, in the limited time frame we had.

The existing rule based stateless firewall module of Floodlight was used and modified according to the need. The firewall enforces Firewall rules which are stored in a text file in comma separated format. For example, the rules stored in the following format as source IP address, destination IP address, protocol allows the packets to pass through the network.

10.0.0.1,10.0.0.4,ARP, ALLOW

10.0.0.4,10.0.0.1,ARP, ALLOW

The connection state of ongoing TCP connections was stored in a hash map in the memory. A combination of source and destination IP addresses together with their TCP ports were used to uniquely identify a connection flow. TCP Flags in TCP packet headers were checked in order to determine the state of these connections.

The firewall inserts a new connection flow into the Hash map and sets its state to SYN_SENT, when a SYN packet is received. And once the two other remaining packets of the three-way handshake process are received, the TCP connection state transits to the ESTABLISHED state. Therefore, that flow can easily travel through the firewall. When the TCP connection is terminated with the reception of FYN packets, the firewall removes the connection flow entry from the hash map & blocks the connection.

The flow chart represented in figure 3.1 shows the flow of work in stateful firewall module. Incoming packets to the switches are sent to the controller if there is no matching flow in the switch. Packets sent to the controller are checked against the predefined firewall rules in the controller. Packets which are not allowed are dropped and packets which are allowed are further processed. Allowed ARP, UDP and ICMP packets are forwarded through the network. Allowed TCP packets are examined in order to maintain the state of the connections. Then the packets are allowed to pass through the network. But the packets which try to violate the connection flows are dropped. A retransmission counter is used to prevent syn flood attacks. If syn packets belonging to a certain connection flow are transmitted more than a threshold value, those packets are blocked. Furthermore, TCP packet transmissions among hosts without establishing a TCP connection are also prevented.

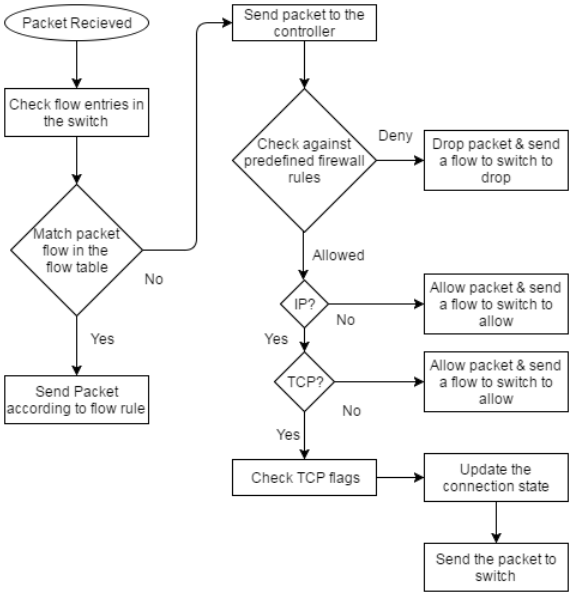

Figure 3.1. Flow chart

Random TCP traffic was generated from the hosts to servers using IPERF tool with different configurations. Then important details about every connection, were stored in a database for further studies. Information like connection duration, source ip, source port, destination ip, destination port, no. of bytes transferred in a connection, connection flag, number of packets of a connection, number of retransmissions of syn packets were stored. This was basically done for the future use, to obtain a data set in order to apply data mining techniques.

IV. Results And Discussion

Stateful firewall module that maintains connection states of TCP connections was built on floodlight controller. Once the connections were completed the details about connections was stored in a database for future use.

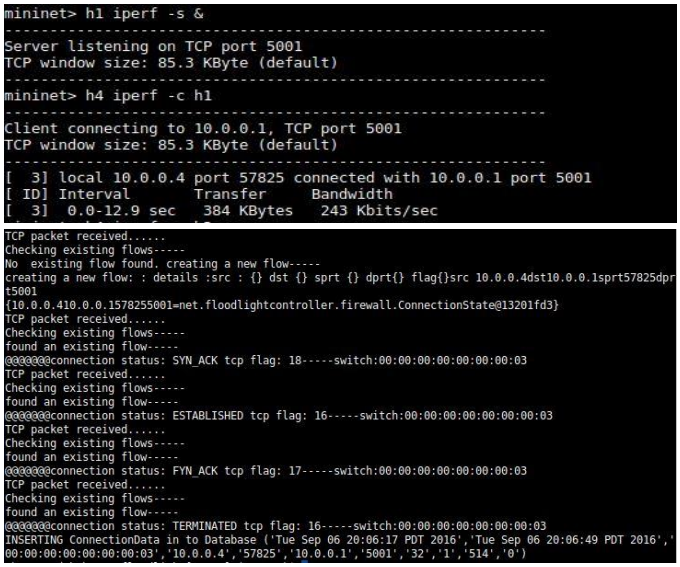

Figure 4.1. connection state maintained for a successful TCP connection

The connection state maintained for a TCP connection between host1 and host4 in virtual mininet network is shown in figure 4.1. Host1 was configured as TCP sever and host4 as TCP client. Connection state of the connection changes with the TCP flags exchanged between hosts. And once the connection is terminated details of the connection was stored in to the database. Packet blocking using SYN retransmission counter feature was not be able to test using the virtual network environment due the inability of simulating that network behaviour.

As further improvements the data collected using the firewall can be used in data mining and thereby come up with a data mining model to detect unusual network behaviour that can occur in the network. With these improvements we can automate the security process at the SDN controller.

V. Conclusion

SDN firewalls have become a prominent role in modern day security. By integrating stateful features to an SDN firewall makes the firewall more intelligent. The Stateful SDN firewall that we created by maintaining the state of the TCP connections prevents SYN flood attacks and DoS attacks that come from impersonated attackers. The controller has been made intelligent to analyze the network behavior and act more like distributed firewall. SDN is still evolving in so many areas and further research should be done to optimize the network security by covering all the aspects.

References

[1] Shieha, Alaauddin, "Application layer firewall using Openflow" (2014). *Interdisciplinary Telecommunications Graduate Theses & Dissertations.* Paper 1. http://scholar.colorado.edu/tlen_gradetds/1

[2] Floodlight: Open SDN Controller. [Online]. Available: http://www.projectfloodlight.org

[3] Raphael Eweka, Sangeetha Elango. "Implementation of address learning/packet forwarding, firewall and load balancing in Floodlight controller for sdn network management."

[4] Zouheir Trabelsi. "Teaching stateless and stateful firewall packet filtering: a hands-on approach"

SDN Flow Caching

N.B.U.S. Nanayakkara, R.M.L.S. Bandara, N.B. Weerasinghe, S,N, Karunarathna
Department of Computer Engineering, Faculty of Engineeirng
University of Peradeniya, Sri Lanka

Abstract—**Software-Defined Networking (SDN) allows control applications to install fine-grained forwarding policies in the underlying switches, using a standard API like OpenFlow. By allowing the switches to store flow rules it can perform a parallel lookup to quickly identify the highest-priority match for each packet. The cost and power requirements limit the number of rules the switches can support. To make matters worse, these hardware switches cannot sustain a high rate of updates to the rule table. In this paper, we show how to give applications the illusion of high-speed forwarding, large rule tables, and fast updates.**

Keywords—Software Defined Networking, Flow Caching

I. INTRODUCTION

SOFTWARE-DEFINED NETWORKING (SDN) allows control applications to install fine-grained forwarding policies in the underlying switches, using a standard API like OpenFlow. By allowing the switches to store flow rules it can perform a parallel lookup to quickly identify the highest-priority match for each packet. The cost and power requirements limit the number of rules the switches can support. To make matters worse, these hardware switches cannot sustain a high rate of updates to the rule table. Ideally, enterprise administrators could specify fine-grain policies that drive how the underlying switches forward, drop, and measure traffic.

The existing techniques for flow-based networking rely too heavily on centralized controller software that installs rules reactively, based on the first packet of each flow. On the surface, the simplest approach to flow-based management is to install all of the low-level rules in the switches in advance. However, pre-installing the rules does not scale well in networks with mobile hosts, since the same rules would need to be installed in multiple locations. In addition, the controller would need to update many switches whenever rules change. Even in the absence of mobile devices, a network with many rules might not have enough table space in the switches to store all the rules, particularly as the network grows or its policies become more complex. Instead, the system should install rules on demand or allow caching the rules in a separate cache.

In this paper, we show how to give applications the illusion of high-speed forwarding, large rule tables, and fast updates.

It is not possible to blindly apply existing cache-replacement algorithms, because of dependencies between rules with overlapping patterns. These rules can match on a wide variety of packet-header fields, and perform simple actions as forwarding, flooding, modifying the headers, and directing packets to the controller. This flexibility allows SDN enabled switches to behave as firewalls, server load balancers, network address translators, Ethernet switches, routers, or anything in between.

II. BACKGROUND

The traditional network devices have control and data-flow functions established the same device. The control to a network administrator is only from the network management plane, which can be configure each network node separately. This behavior of current network devices does not permit detailed control-plane configuration. This is the reason for introducing software-defined networking. The target of SDN is to "provide open user-controlled management of the forwarding hardware of a network element." As mentioned in [8]. SDN has the idea of centralizing control-plane, but keeping the data plane separate. So the network hardware devices keep their data plane, but hand over their switching and routing functions to the controller. This enables the administrator to configure the network hardware directly from the controller. This centralized control of the entire network makes the network highly flexible [9, 10].

A. *Proactive And Reactive Modes*

SDN Controller uses two modes of operations to fill the flow tables of connected switches. Namely they are Proactive mode and Reactive mode. Controller uses only one mode on a SDN network.

In Proactive mode SDN controller places all the rules according to the topology of the network once the switch is initiated. The rules are then relatively static.

In contrast reactive mode, rules are relatively dynamic. Initially the flow table is empty for a given Switch. Switches forwards the packets which do not have an entry in the flow table (OPF packet in). Then the controller examines the packet, decides the flow entry and send both packet (OPF packet out) and flow rule to the switch back. So each and every packet which does not have a flow entry in the switch are forwarded to the controller introducing more delays.

This paper focuses on developing a flow cache for reactive mode. So once a new switch is connected some of the flow rules can be initially taken from this cache. So that the number of packets forwarding to the controller is minimized, and the delay introduced by forwarding packets to the controller is eliminated

III. RELATED WORK

The research paper [5] shows how Software Defined Network allows control applications to install forwarding rules

in switches, using a standard API like OpenFlow. It gives information on modern switches, those rules are stored in Ternary Content Addressable Memory (TCAM). It performs a parallel lookup. That is essential to quickly identify the highest-priority match for each packet. Even though it is efficient, the cost and power consumption limit the number of rules the switches can support. And also these hardware switches cannot sustain a high rate of updates to the rule table. Furthermore it shows how to give applications high speed forwarding, large rule tables, and fast updates. It is done by combining the best of hardware and software processing. The Cache Flow system caches the most recently used rules in the small TCAM, while relying on software to handle the small amount of traffic. But existing cache-replacement algorithms cannot be applied easily because of relationships between rules with overlapping patterns. So rather than caching large chains of dependent rules, long dependency chains are joined to cache smaller groups of rules while protecting the semantics of the policy.

Research paper [6] convinces about data traffic in mobile networks due to the massive adoption of smartphones and tablets, which creates new problems in network. While it always requires more Bandwidth and Data Consumption, cost of network infrastructure is increasing rapidly in mobile networks which will point to an "end of profit" as well. So mobile operators require some methodology that allows to increase the network capacity within low network costs. So to optimize the network capacity usage, the best way is to place caches in strategic locations within network. However, the modern architecture of LTE network does not provide a suitable environment to place the caches in the most optimal locations. This paper proposes to use a SDN approach to remove the need of establishing tunnels. With SDN, packets are forwarded according to rules installed on switches through a centralized controller. The forwarding rules which are installed will be used in routing algorithms to permit scaling as signaling messages are not flooded in the network, but sent directly to the appropriate switches. Those switches use OpenFlow to permit to a centralized controller to install specific forwarding rules. It was proposed to use a caching relocation system based on simple, hence implementable, feedback loop algorithm.

OpenFlow is used to communicate with switches and routers by controllers. OpenFlow is designed to implement network policies. [8] Most of the details regarding OpenFlow packets and Controller architecture are referred from [9], [10] and [11].

IV. Methodology

A. *Caching Flow Entries*

Fig. 1 shows the separate application which works as a cache in the SDN controller. Other applications running on top of SDN controller could be routing algorithms etc. SDN controller has 2 APIs as Northbound API and Southbound API. The SDN cache is implemented using the Northbound API. The southbound API is used to communicate between SDN controller and SDN switches using OpenFlow protocol.

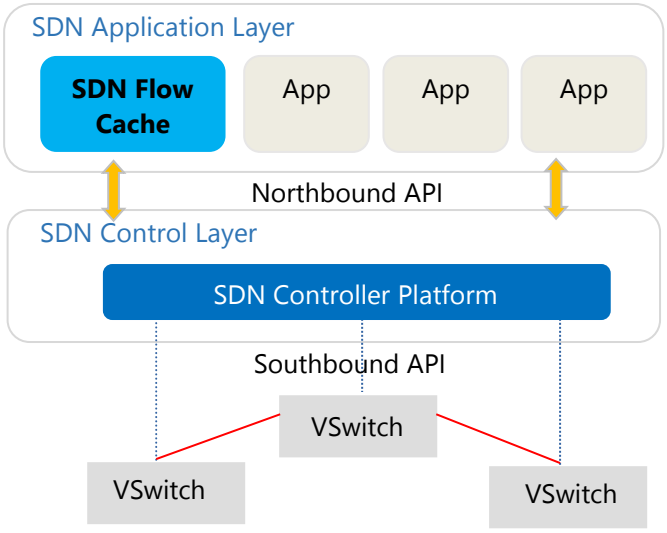

Fig. 1. Application Architecture of the Flow Cache

When a switch is connected to a Controller, after exchanging *hello* messages, the controller sends *featureReq* message to the switch to get information about the switch. Then the switch replies using a *featureRes* message which has all the features of the switch. Then the switch will send LLDP (Link Layer Discovery Protocol) packets to discover devices connected to each port of the switch. These discovered MAC addresses will be used to determine where to forward a packet in a network if the flow tables are not updated.

As an example, first consider a network with two hosts connected to two L2 Switches separately (Fig 2). When the two switches are connected the controller will know the link between S1 and S2 and the links between hosts H1 and H2 with S1 and S2 (H-S1, H2-S2, S1-S2). When H1 pings H2, the first ICMP packet will be forwarded to the Controller as S1 has no flow entry for H2 MAC address. Then the controller will send a flow to the S1 switch with the forwarding entry for H2 MAC address.

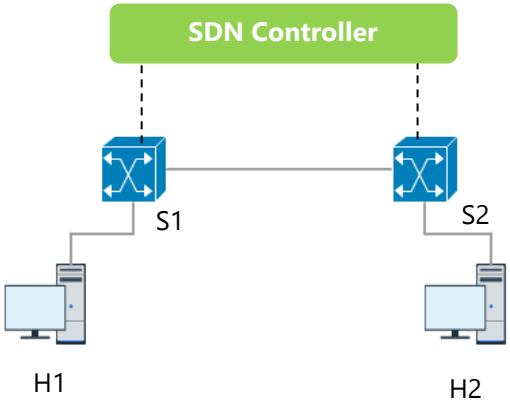

Fig. 2. Example Network I

Suppose that another Switch S3 with host H3 is connected to the network (Fig. 3). This will send LLDP packets to controller and inform the controller about H3 and S3-S2 link. When H3 pings to H2, it will also send the first packet to the controller and get the forwarding flow entry for H3. In this scenario, if the flow cache is implemented, whenever a switch connects to the network it will send flow entries for Hosts and Switches in the other section of the current network topology. So that the new switch will not have to flood the controller with packets in order to update its entries in the flow table. This will have significant impact on the networks with large number of hosts and controllers which are dynamically changes rapidly, such as Data Centers.

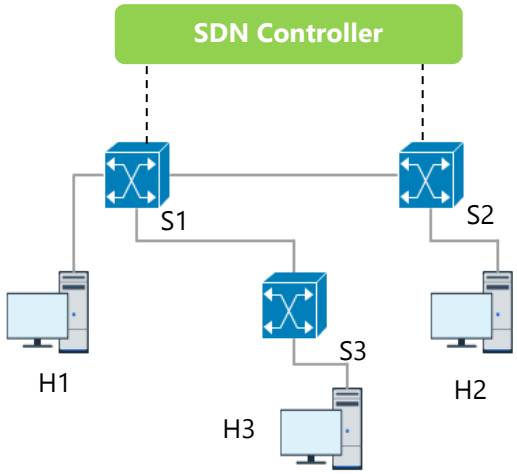

Fig. 3. Example Network II

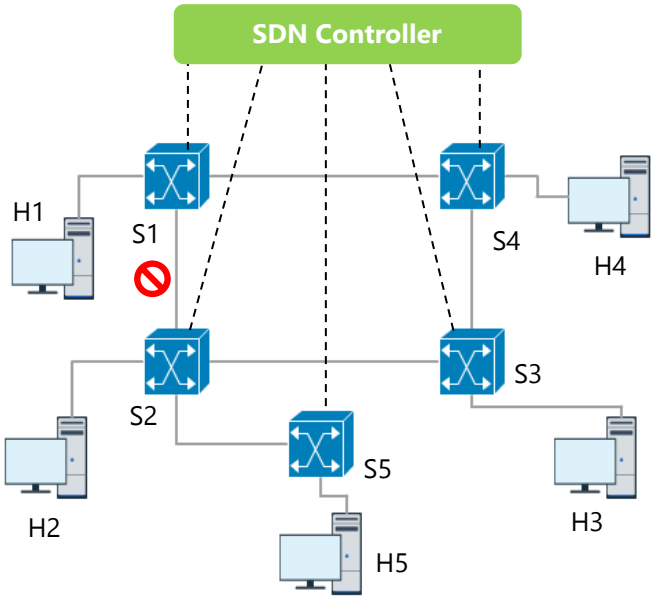

Fig. 4. Example Network III

In the next scenario, consider a network with four switches. The network with S1, S2, S3, S4 switches was the previous network before switch S5 with H5 was introduced. Suppose that the link S1-S2 is restricted for packets from other ports and it is used only for host H1 and H2. In this scenario, the new connected switch S5 should forward its packets via S3 and S4 to S1. This maneuver should be implemented in the flow installing algorithm with validation.

B. Caching Algorithms and Validation

The caching of each flow will be identified from its MAC address. Therefore it can be used as a key for a Hash Table. The object model of the Hash Table will contain an Array of Flow IDs that are created by in the controller. The Cached flows will be stored in another Hash Table and each key would be the Flow ID and object model will be a flow. When LLDP packets are identified by the controller application it will look at the Hash Table and get the flow entries and process it in an algorithm to build new flow entries. If the MAC is new it will make a new Flow Entry object and insert it to the stack install flows for links that are connected to ports.

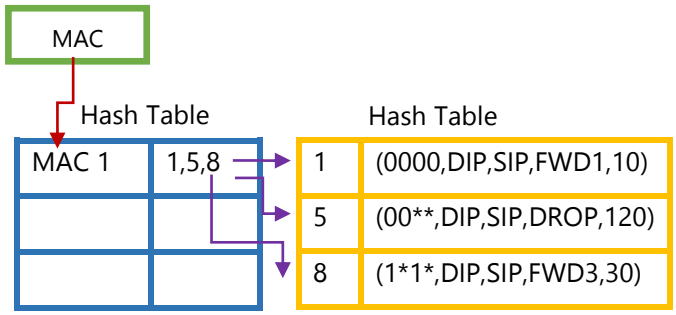

Fig. 5. Data Structure for the Flow Cache

When the user modifies the network with restrictions, the Hash table should be updated to have the restrictions and priorities in the network. This will send only the allowed connections for a new switch in the network. Algorithm 1 can be used to send only the allowed flows to a new switch.

Algorithm 1: Send Flows for Discovered Connection

```
1  func Send_Flows(conn) begin
2      rule_list = PriorityQueue(Flows ϵ conn.MAC)
3      restricted = [ ]
4      for each rule in rule_list
5          if rule.action == DROP then
6              restricted.add(rule)
7      GenerateFlows(flows - restricted)
```

The caching algorithms for SDN rules that can be used are described in the research paper [14] and any applicable algorithm can be used with caching flows.

V. IMPLEMENTATION AND EVALUATION

As described in the methodology, the application will be an instance of a modified L2Switch application in the controller. L2Switch application has all the functions necessary for a physical VSwitch device that access the controller to update flows or network configurations. A referring code base for the switch is in the dev list of OpenDaylight Controller. This development environment was used to implement the caching method for the flows.

OpenDaylight code base is in java and the application implementation is in OSGI framework in the controller. This framework also uses RESTCONF, NETCONF and YANG (RFC6020) models to communicate between network layers and controller layer. This architecture is overall known as MD SAL (Model Driven Service Abstraction Layer) which is defined by OMG[9] (Object Management Group) globally. The controller used OSGI as it can load components in run-time. RESTCONF provide the programming interfaces for the Northbound API that actually runs the SDN network manipulation in the controller. NETCONF is a network management protocol that provides configuration and operational conceptual data stores and perform CRUD operation for these data. Basically it is responsible for running configurations in network devices.

The MD SAL adaptation layers performs the communication between NETCONF and RESTCONF so that the network devices can access the network control plane in the controller.

When the application is informed about a link from the underlying applications, it will update the controller to have a Flow Entry for it. When H1-S1 link is informed it will not wait for a packet from H1 to insert flow entries. Instead controller will update the Flow Cache and send a Flow to the switch as well.

The cache will use a Hash Table to relate each MAC address that the SDN Controller receives. When this is applied in the controller it will be optimized in with relevance to other two modes in the controller as shown in the Fig. 1.

Testing was done using Mininet and it is a network emulator. It is more precisely a network emulation arrangement system. It has a collection of end-hosts, switches, routers, and links on a single Linux kernel. We can make a single system look like a complete network which is running in the same kernel, system, and user code by using a lightweight virtualization, A Mininet host behaves just like a real machine and we can send SSH requests into it and run random programs. The programs we run can send packets through the interface which seems like a real Ethernet interface, with a given link speed and delay. Packets get processed by that Ethernet switch or router, with a given amount of queuing. When two programs, communicate through Mininet, the measured performance should match that of two native machines.

To test the developed system, a new switch connected is connected to an existing network (initially with 10 nodes/ switches). Then a new host is connected to the newly added switch. Now this host pings to a host which is already there in the network (identified by the controller). The time for the 1st ping is recorded. Then the number of switches (Nodes) is increased and the same process of adding a new switch and pining is repeated. The times are recorded.

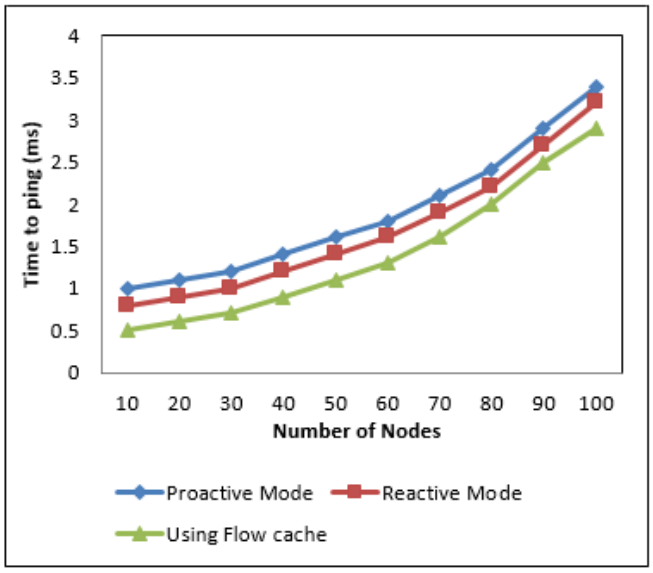

Fig. 6 Time to Ping VS Number of Nodes

VI. CONCLUSION

As in the work covered in this paper, a flow cache using a hash map was designed and implemented in OpenDaylight SDN controller and tested using mininet. Proactive mode SDN networks can be optimized by using a separate flow cache. Flow table of a newly connected switch can be updated using the cache. So that, newly connected switches won't forward the no match packets to the controller. This will reduce the network added delay. Furthermore it will reduce unnecessary packet processing at the controller.

The result statistics showed a performance upgrade with the use of cache than the usual proactive or reactive modes. The performance is up in milliseconds scale but it is really vital in applications such as transaction processing streaming.

REFERENCES

[1] Duan, Q., Yan, Y.H., Vasilakos, A.V., "A Survey on Service-Oriented Network Virtualization toward Convergence of Networking and Cloud Computing," IEEE Transactions on Network and Service Management, vol.9, no.4, pp.373–392, December 2012.

[2] Big Switch Networks, The Open SDN Architecture, http://www.bigswitch.com/sites/default/files/sdn_overview.pdf, 2012.

[3] Shin, M.K., Ki-Hyuk Nam, K.H., Kim, H.J., "Software-Defined Networking (SDN): A Reference Architecture and Open APIs," Proceedings, 2012 International Conference on ICT Convergence (ICTC), pp.360–361, 15–17 October 2012.

[4] https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[5] Infinite Cache Flow in SDN by Princeton University ftp://ftp.cs.princeton.edu/techreports/2013/966.pdf

[6] Caching Using Software-Defined Networking in LTE Networks
https://hal.inria.fr/hal-01117447/file/caching-software-defined.pdf

[7] LRU-K Page Replacement Algorithm ppt by Prof. Shahram Ghandeharizadeh
https://www.google.lk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0ahUKEwjW55ni_aHMAhVpIJoKHYwGBFYQFggaMAA&url=http%3A%2F%2Fdblab.usc.edu%2Fcsci485%2FLRU-K485.ppt&usg=AFQjCNEfoOMn2dN1Syfk7qvAZ7Au5mU_ww&sig2=Ad09dRTucn9zNu6PGs-pHQ&bvm=bv.119745492,d.bGs&cad=rja

[8] Software-Defined Networking: Why We Like It and How We Are Building On It
http://www.cisco.com/c/dam/en_us/solutions/industries/docs/gov/cis13090_sdn_sled_white_paper.pdf

[9] Openflow Protocol Standards and researches.
http://flowgrammable.org/sdn/openflow/

[10] SDN Resources at Open Networking Foundation
https://www.opennetworking.org/sdn-resources/openflow

[11] Open Cache Project by Lancaster University, United Kingdom
https://opencache-project.github.io/research.html

[12] SDNHub OpenDaylight Application Developer's tutorial
http://sdnhub.org/tutorials/opendaylight/

[13] OMG Object Management Group
http://www.omg.org/

[14] rule-caching algorithms for software-defined networks
http://www.cs.princeton.edu/~nkatta/papers/cacheflow-long14.pdf

Affordable Environmental Monitoring and Controlling System for Greenhouses

Supun Athukorala
Department of Computer Engineering
University of Peradeniya
Sri Lanka
Email: supunasp@gmail.com

Irunika Weeraratne
Department of Computer Engineering
University of Peradeniya
Sri Lanka
Email: irunikail@gmail.com

Dumindu Jayathilaka
Department of Computer Engineering
University of Peradeniya
Sri Lanka
Email: omdnadeera@gmail.com

***Abstract*—Greenhouse has been the best alternative solution to get a better crop production compared to the traditional agricultural industry.Greenhouses are used to increase harvest by controlling key factors which will aect the planet growth. Real-time monitoring of the greenhouse environment and taking necessary control decisions will result in improvement of yields and economic performance. In this Research paper, we propose an environmental monitoring and controlling system that have the ability to collect the information related to greenhouse environment using various sensors. This system provide ability to monitor and control the greenhouse remotely via a web interface and a mobile application. Using a low cost wireless sensor network, environment data on greenhouse are sent to the centralized server. It will store all this data and show the latest environment details of the greenhouse using the web interface. The web interface will provide real-time graphical display of data using charts and gauges and ability to send a control decisions to central node which is necessary to increase harvest and improve quality of crops.**

I. Introduction

Greenhouse is a farming structure where the plants which need a controlled climate conditions are cultivated. The target of a greenhouse to provide a better controllable environment to have better harvest, protect crop and crop seeding. Greenhouses are used for growing owers, vegetables, fruits and transplants in countries where the environment is not good for cultivation like high-latitude countries. Greenhouse atmosphere is closed and therefore the heat temperature can be differentiated from the external atmosphere. In most of the greenhouses key environmental factors which are needed to be controlled are temperature, irrigation, level of light and shade, fertilizer application, and humidity. Greenhouses are used to increase the harvest in an area where those previously explained key factors are not available as it needs. A greenhouse environment is a complex and dynamic environment which is needed to be controlled carefully otherwise it will aect the crop cultivation. To have a better crop production the climate conditions are needed to be adjusted by continuously monitoring those environmental factors which are explained before. For an example a plant may die because of too much water or because of not enough water. Poor light intensity and high humidity often result in poor fruit set and quality. More accurate control can save resources, eort taken and have a better crop production.

A. Wireless sensor network

A wireless sensor network is a group of nodes which are capable of recording sensory data at diverse locations and transmitting this data to a central node. Those nodes which are called sensor nodes are small, lightweight and portable. Commonly monitored parameters are temperature, humidity, pressure, oxygen percentage and Carbon dioxide percentage.The Central node is monitoring the activities of each sensors and act as a access to the sensor network. The central node is connected to a the distant server which is used to monitor data and control the system. [1]

B. Sensor Node

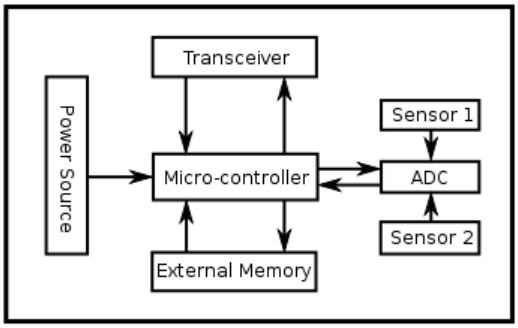

Fig. 1. The typical architecture of the sensor node

A sensor node[2] has capability of gathering, processing and transmitting a sensory data gathered using a transceiver. Therefore it is equipped with a microcomputer to process the data, transceiver to communicate, power source to act as an independent device and one or more sensors. The sensor generates electrical signals based on sensed physical eects and phenomena. The microcomputer processes sensor output. The transceiver data to the central node frequently. Wireless sensor

nodes are very small devices with limited battery source.The key factor of the sensor node is the power consumption. Sensor nodes are placed inside the greenhouses. Therefore, the external power supply is not applicable. The best solution is power from the battery. To give power from a battery, those equipment should consume low power. Therefore, the design should be a low power sensor node.

II. Wireless Sensor Network

Wireless sensor network is used to monitor the greenhouse activities and control the environment in a way that the harvest of the plants in the greenhouse is maximized. For that an clustered wireless mesh network is created and sent data to a centralized server which monitor the data given by the sensor network and give feedback to control the greenhouse. The reason to create a clustered mesh network is to reduce the cost of the implementation of the sensor network rather than using high cost nodes as sensor node and connecting them to a relay node in the sensor network and send data through the sensor network.

1 cluster = Relay Node + Sensor nodes connected to the given relay node

By clustering the low cost sensor node for the nearest relay node, it allows to create very affordable sensor network by using very few expensive relay nodes.

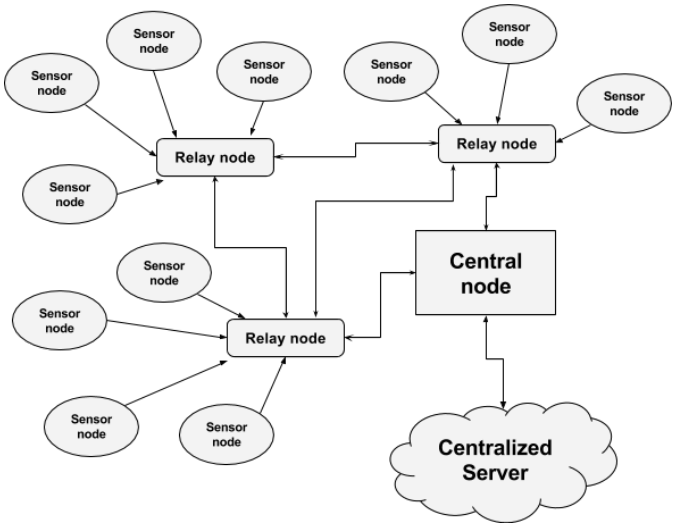

Fig. 2. Sensor Network Overview

A. Low-cost Low-power Sensor Node

Sensor nodes are the leaf of the sensor network.sensors are connected to sensor nodes. Sensor nodes gather the data and send to central nodes.Thus sensor nodes are should be capable of getting inputs from sensors, process gathered data and wirelessly communicate with central nodes.Considering those requirements and the cost,low-cost NodeMCU open-source development kit was chosen to implement the sensor nodes. To make sensor nodes are more power efficient, sleep mode used in the microprocessor(NodeMCU) and lightweight messaging protocol called MQTT was chosen over HTTP.

B. Relay Nodes in Sensor Network

Relay nodes are responsible for creating the mesh network and routing data to the central node. Relay nodes are created in a way which can automatically find the nearest relay node and send data through the sensor network. For the routing, B.A.T.M.A.N (Better Approach To Mobile Ad-hoc Network) protocol was used to create mesh network.
It also has a running wifi access point[3] where the nearest sensor nodes can connect and publish data to the network using MQTT.
Relay node is the node which create intermediate clusters of the network. It abstract data from the sensor nodes and send it to the other relay nodes which are engaged in the mesh network (B.A.T.M.A.N Network) and finally it flows though the central node to the centralized server.

C. Central Node of Sensor Network

Central node is the node which communicate with the Relay nodes and gather data to be sent to the centralized server. This includes an MQTT message broker which can brake the message and send data to the centralized server using HTTP protocol. This node is known as the sync of the system. Which means all the data gathered in the sensor network finally be routed to the central node. This is the only node which is connected to the centralized server for the communication purposes that makes the system more secured.

D. Implementation

Sensor nodes developed using NodeMCU [4] platform. It can be programmed using Lua programming language [5]. Arduino IDE and esp8266 [6] package were used instead of using Lua directly. The node was programmed as follows,

- Connects to given MQTT server
- Read sensor values from given GPIOs
- Publishes those data as a JSON object to the given topic every two seconds
- Subscribers to the topic given topic to receive control commands
- Reconnecting functionality to the server if the connection is lost

In centralized server, for the implementation Raspberry pi 2 board with wifi dongle was used and to send data central node has to connect to another network (Default using the Ethernet port).
As the MQTT message broker, Mosquitto[7] was used and it runs in the background of the central node's Linux system. Then using python mqtt-paho [8] library the data was gathered and converted in to JSON files and sent it to the centralized server for the processing and data gathering.
Beside Mosquitto server centralized server runs the B.A.T.M.A.N protocol [9] and work as the bat0 port

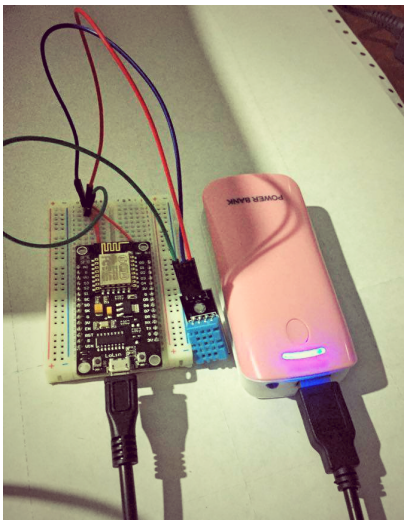

Fig. 3. Central Node Implementation

which is the default gateway for the mesh network. So it works as the sync of the mesh network.
Also using the Shell scripts and Crontab [10] facility in the Raspbian OS [11] the process was automated so even in reboot, the process runs automatically.
For the Relay node B.A.T.M.A.N protocol was used to create

Fig. 4. Central Node Implementation

the mesh network. B.A.T.M.A.N can identify the nearest member of the mesh network and configure automatically.
Besides B.A.T.M.A.N there is a access point running in every relay node which the sensor nodes can search and join the network of the relay node. It is the way that the clusters of the network are automatically formed.

.

III. Centralized Server

The primary goal of the project to create a feature where modern farmers in agricultural industry can monitor their greenhouses remotely. Therefore, an idea of the centralized server comes to reality. This centralized server is acting as the data storage for wireless sensor network. Also, it will act as a access point for the sensor network.
This centralized server has main three parts. They are

- REST API for monitoring system
- REST API to receive data from Sensor Network
- REST API to control environmental factors in greenhouse

A. REST API for monitoring system

The centralized server was developed as a fully restful system because the REST API calls will be helpful for any developers where they can use it to develop a custom User Interface or integrate those calls to any system that they already have. REST API for monitoring system consists several API calls including,

- Get the detail of the sensor network
- Get detail of the sensor nodes in the sensor network
- Get data as chart data for date range and sensor type
- Get the latest values for gauges
- Get data of a specific sensor type as chart data

To get the sensor types and number of sensors, aggregation is used. Aggregation operations process recorded data and give the calculated results. The aggregation results can be groups of similar characteristics. Using this aggregation operations, the system can identify if there are new sensor types are available. Not only that system will show the number of all the nodes available in the system. There are REST API calls to retrieve the number of sensor nodes in the system, sensor types in the system and the system details.
To have the simplicity, for the chart data, the system uses key value pair object. Therefore if chart data was requested, then system will send a list of this key-value pair objects.Those charts mostly need last 24 hour sensor data. There is also that facility and user can customize the chart data request to get chart data in a custom date range and custom interval like 'minutes','hours','days' and etc.

B. REST API to receive data from Sensor Network

Centralized server doesn't need a relational database management system. Because this system saves only the sensor data and network details in the system. Therefore a non-relational database was used in this system. There is a REST API calls to receive data from Sensor Network. These calls will receive data in a specific format which is designed to reduce number of calls to the centralized server.
The sensor network is designed to use the minimum power consumption. Therefore, the sensor nodes are in sleep mode most of the time. Frequently It will automatically send sensor data to the central node. At that time central node will send those sensor data to the centralized server. These API calls are developed to receive those data from central node. Also there are REST API calls to retrieve network details from the sensor network.

C. REST API to control environmental factors in greenhouse

System has REST API calls which will send central node about which environmental factor to control. Centralized server will action to be done (”increase”, ”decrease” and ”stop”). It will find the address of the central node and send request to the server in central node to increase/decrease or stop changing the environmental factor.

D. Implementation

Centralized server is developed based on spring framework [12]. The Spring Framework is an application framework build on the Java platform. This is a fully restful system whose REST API calls will be used by the front end of the monitoring system and the sensor networks. As a NoSQL database MongoDB [13] was used
The Centralized server is implemented using following technologies.

- Spring Core Library - Version 3.2
- MongoDB - Version 3.2.4 as storage for the system
- Gson A Java serialization/deserialization library that can convert Java Objects into JSON and back.
- MongoDB java driver - create a connection between MongoDB server.
- Jackson data binding library - converts JSON to and from Java Maps, Lists, Strings, Numbers, Boolean, and null objects.
- Spring data MongoDB - library to work with MongoDB

This system works using spring Data MongoDB library to query and aggregate data.The total system give responses in JSON format. Therefore any system can use those JSON responses and use those data to convert to any format.There is also a specific JSON object type is used to receive sensor data.

IV. Web Interface of System

The front end of the monitoring and controlling system is a Web Interface to show the sensor data of the sensor network and control environmental factors. Web interface is a better solution to real time monitoring and controlling system because it is accessible from anywhere and no setup overhead. This web interface (Front End) has main three parts. They are,

- Network details
- Latest values
- Charts of Each Sensor Types

A. Network details

The user can see the details of the sensor network. Where is the sensor network is located, Which are the sensor types in the network, how many sensor nodes are available in the network and etc.

B. Latest values

The web interface has gauges to show the latest values from the sensor network. It will return the most recent value from each sensor type.This gauges will show user that if the environmental factor is not in comfortable area. Not only the real time value, the node number and the last checked time also can be seen in the web interface. The Latest value part in the system is for user to get the idea about the latest environmental factors in the greenhouse. To get how the greenhouse environment in past can be seen in following section called chart data.

C. Charts of Each Sensor Types

User can see the data gathered and stored from sensor network as line charts. The inputs for the chart can be customized like changing date range, change time interval. The scale can be changed like hours,days,weeks and etc. Also looking at chart data user can send a signal to centralized server to increase/decrease relevant environmental factor.

D. Implementation

This is developed using AngularJS [14], JavaScript Framework. Those data are obtained by sensor network real time. Using AngularJS and Kendo UI [15] to show sensor data as chart data. The charts and gauges are implemented using Kendo UI library. This web Interface is updated frequently. It

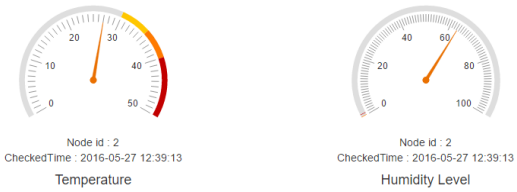

Fig. 5. gauges using kendo UI.

uses AngularJS function called $interval and update the charts and gauges freuently.
Those charts uses Kendo UI chart library which will identify the values as dates. Therefore when adding values to the charts those values need to be converted to JavaScript date. Therefore the chart will use those data and draw the chart.

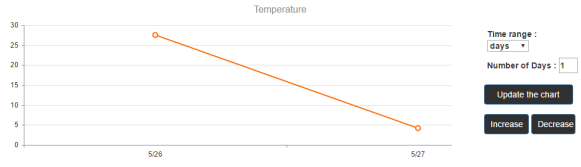

Fig. 6. line chart using kendo UI.

E. Controlling Feature in Monitoring System

Controlling feature is developed in system with basic three actions. They are ”increase”, ”decrease” and ”stop”. Web interface will send signal about either previous actions with the respective environmental factor type.

V. Conclusion

A wireless sensor network for environment monitoring and controlling of greenhouse was build successfully.The system tested in real time where data was successfully captured and displayed via a web interface.The mesh topology was used in sensor network.The captured data presented to the user through line charts and gauges implemented using kendo UI framework.Also environment parameters can be controlled via web interface controllers. Wireless Sensor Network in greenhouse monitoring and controlling is extremely useful and a low cost system presented in this paper is more affordable for both small and large scale greenhouses.

References

[1] S. feng Yang and D. S. Simbeye, "Computerized Greenhouse Environmental Monitoring and Control System Based on LabWindows/CV," https://www.researchgate.net/publication/262380106_Computerized_Greenhouse_Environmental_Monitoring_and_Control_System_Based_on_LabWindowsCVI, 2013.

[2] "Sensor Node," https://en.wikipedia.org/wiki/Sensor_node.

[3] "How to create WiFi Hotspot in Raspberry Pi," http://elinux.org/RPI-Wireless-Hotspot, 2015.

[4] "ESP8266 ESP-12E NodeMCU," https://en.wikipedia.org/wiki/NodeMCU.

[5] "Lua," https://www.lua.org/, 2016.

[6] "WiFi Module-ESP8266," https://www.sparkfun.com/products/13678.

[7] "Mosquitto," http://mosquitto.org, 2016.

[8] "pahomqtt 1.1," https://pypi.python.org/pypi/paho-mqtt/1.1, 2016.

[9] "B.A.T.M.A.N," https://en.wikipedia.org/wiki/B.A.T.M.A.N., 2016.

[10] "CronHowto," https://help.ubuntu.com/community/CronHowto, 2016.

[11] "Raspbian," https://www.raspbian.org/, 2016.

[12] "Spring Framework," https://spring.io/, 2016.

[13] "MongoDB," https://www.mongodb.com/, 2016.

[14] "AngularJS," https://angularjs.org/, 2016.

[15] "Kendo UI," http://www.telerik.com/kendo-ui, 2016.

Smart Campus Phase One: Smart Parking Sensor Network

H.M.A.P.K. Bandara, J.D.C. Jayalath, A.R.S.P. Rodrigo,
R.G. Ragel, A. Bandaranayake, Z. Maraikar
Department of Computer Engineering,
Faculty of Engineering, University of Peradeniya,
Peradeniya 20400, Sri Lanka.

***Abstract*—Smart Parking Sensor Network project aims to develop a low cost sensor based parking system to map the usage of parking areas. This system consists of sensor nodes which can detect the occupancy of parking space; relay nodes to communicate between sensor nodes and the server; server application to get data from the relay nodes and send data to mobile application; and a mobile application to display the parking areas and the occupancy of the parking areas on a map. The mobile application was developed using Android and the server application is hosted in AWS (Amazon Web Services). The vehicle detection sensor node was designed with magnetic sensor and a distance sensor. The magnetic sensor detects the presence of the vehicle and the distance sensor clarifies it.**

I. Introduction

The "Smart Campus" concept is an initiative to use ICT (Information and Communication Technology) within a University Campus to improve the quality and performance of the services, to reduce costs and resource consumption, and to engage more effectively and actively with its members. Smart Parking is a constituent of the Smart Campus, that looks to address this issue of parking within a campus. The difficulty in finding available parking spaces within a campus waste time and fuel, and create stress among many visitors and employees. In a regular park, there is no way to know where the available parking spaces are. By delivering a real time map of the availability of parking spaces, smart parking idea directly impacts on the sustainability of university, revenue and the level of the service.

This paper presents a pilot project to be at the forefront of this change at the University of Peradeniya. Phase 1 of Smart Campus introduces a Sensor Network/IoT-based Smart Parking System to map the usage of parking spaces within the University. This parking system consists of sensing devices to determine the occupancy at the parking space, a network to transfer data, a server application to process the data, and a mobile application to display occupancy of the parking spaces. Smart parking is a huge industry overseas, but not in Sri Lanka. Because of that, all the components should be bought from foreign vendors to implement a smart parking system, which includes the vehicle detection node, the relay node (to communicate between server and sensor) and the applications. The average price of a vehicle detection node is USD80 - USD160 (ROSIM-ITS) and the average price of a parking system is USD5000 per year (excluding taxes and delivery cost) [1]. Because of that, it is costly to deploy a smart parking system using imported components. Therefore, this paper suggest a low cost solution which can be developed within the country and deployed in mega scale.

II. Related Work

Commercially developed smart parking solutions are very popular in the global market. Streetline [2] is one of the leading smart parking company based in USA. Their solutions include; parking analytics platform, parking inventory management platform, parking map, real-time parking guidance application, parking data API, potential real-time violations for optimal workforce efficiency. Their vehicle detection sensor is surface mounted and primarily made with magnetic sensors. Smart Parking Limited [3] (ASX: SPZ) has been providing smart parking solutions for more than ten years. The company provides on-street, off-street and parking management solutions. Their vehicle detection sensor is made with infrared technology, and it is surface mounted. Libelium [4] designs and manufactures hardware and a complete software development kit (SDK) for wireless sensor networks so that companies can deliver reliable Internet of Things (IoT), and Smart City solutions. This platform is designed to be buried in parking spaces and to detect the arrival and departure of vehicles. The sensor node is based on magnetic sensors.

The above three are the leading companies in smart parking solutions. All three companies provide sensor nodes which are surface mounted and buried in the parking area and this paper discusses a surface mount sensor node, that can be applied even for already constructed parking areas without making lots of changes. The commercial solution providers are mainly providing their solutions in American, European and Australian continents. Applying the solution in Asian continent is expensive because of the reason discussed in the previous section. Our system is implemented in cost effective way to counter that problem. Our mobile application also contains similar functionalities to those which are in the solutions mentioned above.

There are some research based projects about vehicle detection sensors. In [5], Koszeteczky and Simson suggest a magnetic sensor-based system to identify the vehicle and estimate their powers with some algorithms. The proposed solution of this research is that a vehicle can be determined

by using magnetic sensors. In [6], Casas, Sifuentes and Pallas-Areny suggest a Node made of a magnetic sensor and an Optical sensor for detecting vehicles. They choose the optical sensor due to massive power drain of the magnetic sensor. The magnetic sensor is woken-up only when the state of the optical sensor is changed. In [7], S. Ma, C. Xu, Y. Wang, F. Li and X. bao conducted their research on reliable wireless vehicle detection using a magnetic sensor and a distance sensor and they suggest a node mode of the magnetic sensor and a distance sensor to detect the vehicles. Magnetic sensors were used in all above researches because they are the best to detect metal objects and vehicles contain a significant amount of metal parts. This paper presents a solution with a magnetic sensor as one part of the vehicle detection node and a secondary sensor, which is selected from distance and optical sensors.

III. Proposed Methodology

The architecture of the solution is given in Fig. 1. This architecture consists of three main sections, Sensor network, Server, and Mobile application. The sensor network consists of vehicle detection sensor nodes and relay nodes. Sensor nodes connect to relay nodes through a wireless communication medium (wifi/MQTT). Relay nodes connect to the server through a wireless communication medium (wifi). Sensor nodes do not connect to the server through the outer network because these nodes designed to be low cost with limited communication capability and relay nodes will be more powerful. The server application will be on the internet and relay nodes will transfer data using the MQTT protocol. The server application will send data about the occupancy of parking areas when requests come from the mobile application.

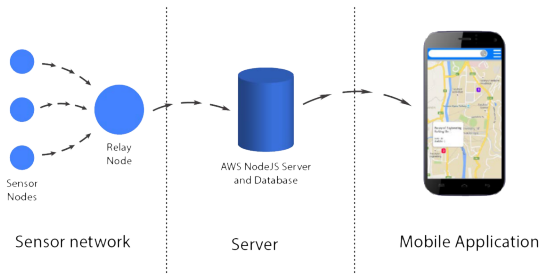

Fig. 1. System architecture

A. Vehicle Detection Sensor Node

The wireless sensor node detects the presence of a vehicle in a predetermined zone (parking space). The sensor node includes a magnetic sensor and a distance sensor. The magnetic sensor that drains a very small current is always on and detects the presence of nearby ferromagnetic material. The distance sensor is woken up by the magnetic sensor to measure the distance to the nearest object over it. Both sensors are directly connected with a microcontroller, which results in a compact, simple vehicle detector.

1) Magnetic Sensor: This sensor detects the changes in the Earth's magnetic field. The change can be happen because of a vehicle, which have significant amount of ferrous metals and their magnetic permeability is much higher than that of the surrounding air and soil. Different types of vehicles have distinct magnetic signatures A given magnetic anomaly can be produced by a small vehicle close to the sensor or a larger vehicle father away.

2) Distance Sensor: Due to the reliability in magnetic sensor readings mentioned above, we propose to add a second sensor. Distance sensor is able to determine the distance information when a vehicle is parked over it. If the magnetic sensor detects a vehicle, the distance sensor wakes up to detect whether the vehicle is over the sensor or not. For this measurement, the distance sensor should be small, low-power, less expensive and have a measurement range of 10cm-100cm. Among Laser, ultrasonic and infrared distance measurement techniques, infrared matches the above requirements well. This sensor can emit modulated light pulse and measure the distance through light transmission time accurately.

B. Relay Node

In the sensor network, sensor nodes sense the presence of vehicles and this data needs to be transmitted to the base station using wireless media (wifi). Due to various factors sensor network faces challenges such as limited power, scalability and connectivity in this task [8]. One of the solutions to overcome these challenges is to use relay nodes, whose job is only to relay data generated by other sensor nodes. It has been shown in the literature [8] that the introduction of relay nodes in sensor networks may result in prolonged lifetime as they can remove some burden from the nodes. The relay nodes may also shorten the transmission distance between a pair of distantly located nodes by acting as a hop between them [9].

C. Server and Mobile Applications

The base station which gathers data is the server application. It acts as the layer between the mobile application and the sensor network. The server application is also capable of storing the data for future analytics. The mobile application has the capability to show where the available parking places are. The application uses an event-driven method to communicate with the server [10][11]. It provides a bi-directional communication channel between a client and the server. That means the server can send data to all connected users at the same time.

IV. Design and Implementation

A. Sensor Node Architecture and Vehicle Detection Algorithm

The sensor node consists of a processing unit, a magnetic sensor, an IR sensor and at power supply. The sensor node architecture is shown in Fig.2

NodeMCU [12] is selected as the experimental processing unit. It is a low-cost development board with ESP8266 unit combining computing power with wifi. It has several sleep modes [13] in order to save power. The magnetic sensor

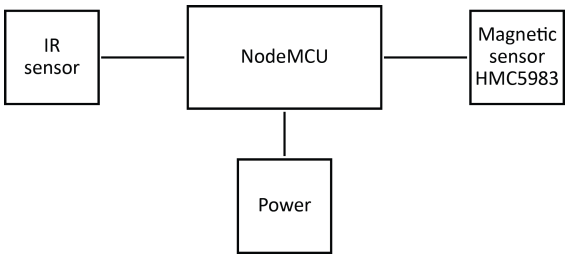

Fig. 2. Sensor node architecture

is HMC5983 [14], which is a sensitive, small and low-cost 3-axis anisotropic magnetoresistance (AMR) sensor with digital output, with a typical sensitivity range of 230 to 1370 LSb/gauss, and a field range of -8 to +8 gauss. The sensor needs extremely low current; 2 μA in idle mode and 100 μA in the measurement mode. The distance sensor is the sharp GP2Y0A21YK0F, low-cost IR sensor, composed of an integrated combination of PSD (position sensitive detector) , IRED (infrared emitting diode) and signal processing circuit. The distance measuring range is 10cm to 80cm and typical current consumption is 30mA, higher than the magnetic sensor but acceptable because it only requires to work several times when woken up by the magnetic sensor and works in 4.5V to 5.5V [15]. Fig. 3 shows the complete circuit diagram of the sensor node.

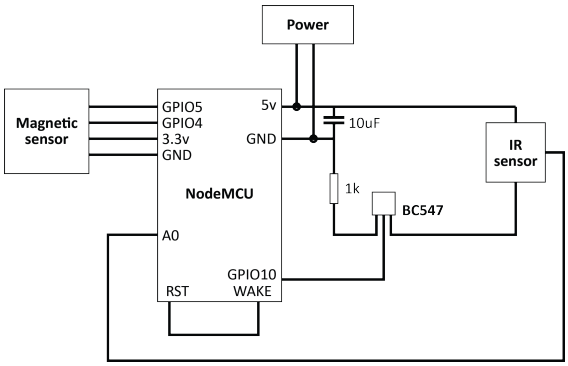

Fig. 3. Wireless vehicle detection sensor node circuit with NodeMCU, magnetic sensor and distance sensor

Vehicle detection algorithm: Fig. 4 shows the detection algorithm in detail. After the sensor node is placed on application site, the first procedure is the system initialization. The sensor node then opens the distance sensor and initialize the initial distance measurement. Then set the (to an interval of 1 minute) deep sleep mode of the NodeMCU. After each minute, the NodeMCU wakes up and measures the magnetic field strength. Then, the current measurement is compared with the previous magnetic field strength.

$$mag\_diff = |mag\_now - mag\_prev|$$

if mag_diff is larger than a predefined threshold_mag, the magnetic field is assumed to have changed. Then NodeMCU powers on the distance sensor to measure the distance for two times. Then the distance is compared with predefined distance threshold threshold_dis (the threshold_dis is between 20cm - 80cm). If the distance is within the threshold_dis, it is assumed that there is a vehicle, otherwise it means there is not. If there is, the node checks whether the variable ”occupancy” is 1. If it is, a vehicle was already there before and no need to transfer data again. If not, the node changes the variable ”occupancy” into 1 and transfer data. If the distance is not within the threshold_dis, it is assumed that there is no vehicle. Then the node checks whether the variable ”occupancy” is 0. If it is, no need of transferring data. But if not, the variable ”occupancy” changes into 0 and transfer data. Otherwise NodeMCU goes into deep sleep for a minute.

The Sensor Network was implemented using MQTT[16] (Message Queuing Telemetry Transport) protocol. This MQTT protocol follows the publisher/subscriber architecture. There is a central unit called the Message Broker in the middle of the publishers and subscribers in the network. As the Message Broker, Mosquito[[17] Message Broker was selected. It is a free and open source product. All the Nodes in the park are publishers in the network while all the mobile app users are subscribers. Subscribers can get the data(sent by the nodes) through the Message broker by listening to the relevant channel name. This Message broker deployed to the AWS Elastic Computer Cloud.

B. Server and Mobile Applications

The Server application was configured as follows. First, the application architecture was designed with a loosely coupled front-end and back-end. This architecture is preferred many developers and communities in node.js application development life cycle. Second, the express.js[18] framework was configured as the main framework in the application. In software development process it is important to have separate environments for development, testing, and production. Separate configurations were implemented for each environment. For server side rendering in the application, ejs[19](embedded JavaScript) template engine was used.

The Elastic Computer Cloud of Amazon Web service was selected to deploy the web application and database. As the database, MongoDB[20] was selected and database connection was implemented using third party tool called Mongoose[21]. Mongoose is a node.js library which is used to ORM (Object Relational Mapping) for MongoDB.

The Real-Time communication part was configures using socket.io[11] middleware. The Server behaves as a subscriber of the mosquito message broker. When a Node publishes data, the broker will send those data to relevant subscribers. Server application also receives those data. After receiving the data, Server will process those data, store them in the database and send them to connected mobile applications using socket.io middleware in real time.

The mobile application of the smart parking sensor network is called ”ParkMe”. It has the capability to show where the available parking places are. This mobile application is made

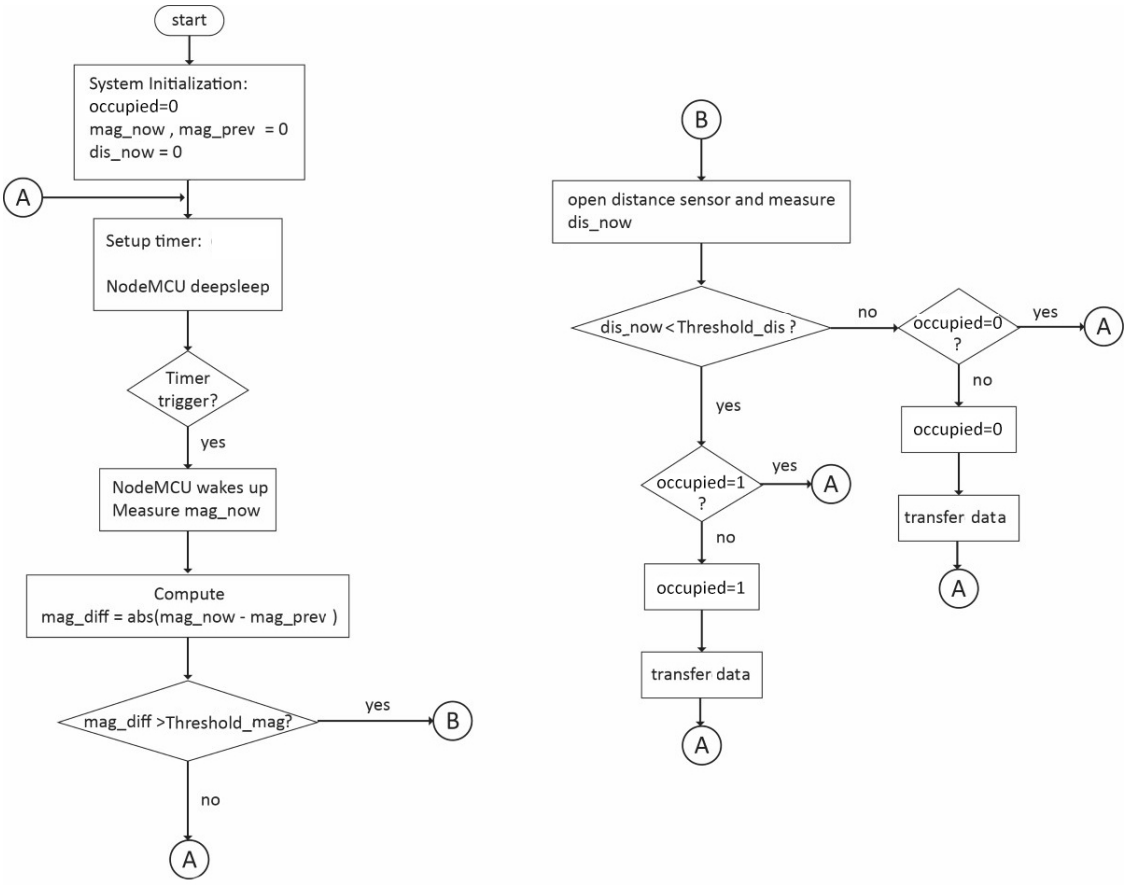

Fig. 4. Vehicle detection algorithm

available for Android users. Using this mobile application, users can check available parking places in the nearest Park, from where the user finds the parking slots and also after clicking the google map markers user can get directions to the selected parking area.

The ”ParkMe” Mobile application is developed as a native android application using Java. The Google Map API is used to show the google map in the mobile application. When using the mobile application user can change google map settings as well as the parking area types. The custom markers are used to show the available number of free parking slots and also two colors are used to show the parking area type which is indoor and outdoor. By clicking the markers, users can look into more details of the parking area, such as the available percentage of parking space of each floor if it is a multi-story park and parking chargers etc. For the communication between the server and the mobile application, the MQTT protocol is used because it is a lightweight protocol which is developed mainly for IoT application. In the mobile application for getting parking area data from the server to client mobile application, the Paho[22] Android Service was used. It is an MQTT client library written in Java for developing applications on Android.

V. Results & Discussion

To facilitate parallel development in all areas of the project the proposed system is tested as separated sections in the architecture. The mobile application was tested by setting up virtual parking areas on the server and sending data with different allocations of the parking areas. Fig.5 shows the parking area visibility on the map.

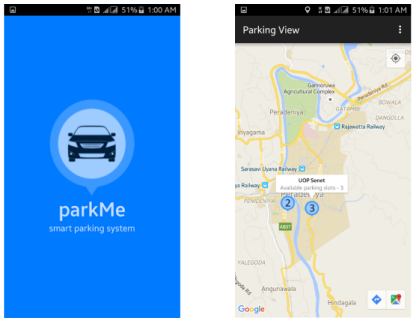

Fig. 5. Mobile application interfaces with parking areas

To test the proposed sensor nodes, we built the sensor node in Fig.3 and the test plan was to place the sensor nodes in the center of parking spaces as in Fig.6. The test was planned to

conduct with a car and an SUV in different weather conditions and times of the day.

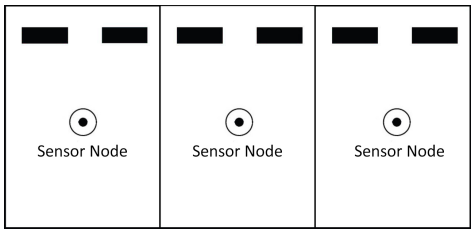

Fig. 6. Sensor node experimental area

As discussed, vehicles in nearby parking space can trigger the magnetic sensor to wake up the distance sensor, which wastes battery life. To minimize these false positives, a predefined threshold should be determined. The following scenarios were planned to carry out. Scenario 1: car enters the middle parking space, stays parked for 5 minutes (deep sleep time of the sensor node is a minute), then leaves. Observe the magnetic sensor behaviour and distance sensor behaviour of three sensor nodes. Scenario 2: SUV repeat the same as scenario 1. Observe the sensor behaviour. Scenario 3: car parks in the first parking space and the SUV enters the third parking space and repeat as scenario 1. This scenario will display the behaviour of the sensor node with different vehicles and the accuracy of the sensor node. The test plan on the sensor and the sensor network was not conducted by the time of this publication.

VI. Conclusion

In this paper, an inexpensive solution was proposed to design and develop a smart parking sensor network with a mobile application, server application, a relay node and magnetic and distance sensor based vehicle detection node. Mobile and server applications were developed. Vehicle detection node design is currently ongoing and node testing scenarios were proposed along with the vehicle detection algorithm. Continuation of this project can be done as the next phase by conducting the test scenarios and adding low-power consumption methods in vehicle detection node.

References

[1] *For Universities: streetline*. URL: http://www.streetline.com/manage-parking/for-universities/.

[2] *Streetline*. URL: http://www.streetline.com.

[3] *About Smart Parking Limited*. URL: http://www.smartparking.com.

[4] David Gascon Alicia Asin. *Smart Parking Sensor Platform enables city motorists save time and fuel*. 27 May 2011. URL: http://www.libelium.com/smart_parking/.

[5] B. Koszeteczky and G. Simon. *Magnetic-based vehicle detection with sensor network*. Tech. rep. Vesprem, Hungary: University of Pannonia, Department of Computer Science and System Technology, 2013.

[6] O. Casas E. Sifuentes and R. Pallas-Areny. *Wireless Magnetic Sensor Node for Vehicle Detection with Optical Wake-Up*. Tech. rep. 2011.

[7] Y. Wang F. Li S. Ma C. Xu and X. bao. *Reliable Wireless Vehicle Detection using Magnetic Sensor and Distance Sensor*. Tech. rep. Shanghai Institute of microsystem and Information technology, Chinese Academy of Science, 2014.

[8] Ataul Bari. *Relay Nodes in Wireless Sensor Networks: A Survey*. Tech. rep. University of Windsor, 2005.

[9] Errol L. Lloyd and Guoliang Xue. *Relay Node Placement in Wireless Sensor Networks*. Tech. rep.

[10] Drifty Co. *ionic framework*. URL: http://ionicframework.com..

[11] *socket.io*. URL: http://socket.io.

[12] *NodeMCU*. URL: nodemcu.com/index_en.html.

[13] *Reduce Power Consumption by Sleep*. URL: https://sourceforge.net/p/nodemcu/wiki/Sleep/.

[14] *HMC5983*. URL: http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures - documents / HMC5983_3_Axis_Compass_IC.pdf.

[15] *Sharp GP2Y0A21YK0F*. URL: http://www.sharpsma.com/webfm_send/1489.

[16] *MQTT*. URL: http://mqtt.org/.

[17] *MOSQUITTO*. URL: https://mosquitto.org/.

[18] *ExpressJS*. URL: https://expressjs.com/.

[19] *EmbeddedJS*. URL: http://www.embeddedjs.com/.

[20] *MongoDB*. URL: https://www.mongodb.com/.

[21] *Mongoose*. URL: http://mongoosejs.com/docs/2.7.x/docs/model-definition.html.

[22] *paho*. URL: https://www.eclipse.org/paho/.

Accelerating k-NN Classification Algorithm Using Graphics Processing Unit (GPU)

Selvaluxmiy Selvarajan, Velmakivan Ramakrishnan, Roshan Ragel, and Sampath Deegalla
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka
luxmiy21@gmail.com, makivan8@gmail.com, roshanr@ce.pdn.ac.lk, dsdeegalla@pdn.ac.lk

Abstract — The data of government, enterprises, universities and various organizations are undergoing massive increases. Those data needs to be maintained. For this purpose, there are many algorithms used in data mining. Among them, k-Nearest Neighbor (k-NN) is mostly used in many applications, which are based on classification. When the data became significant, the execution time of the algorithm will be a bottleneck for data classification.

In this paper, we describe the implementation of the k-NN algorithm on GPU, using CUDA to improve the execution time of the k-NN algorithm for large datasets.

Keywords— Compute Unified Device Architecture (CUDA), Graphic Processing Unit (GPU), k-Nearest Neighbour (k-NN)

I. INTRODUCTION

Data mining is a process of collecting, searching through and analyzing a large amount of data in a database as to discover patterns or relationship. It is helpful in data cleaning, data pre-processing and integration of databases. It is used in many areas such as biological and market research, medical imaging, and other sectors. For example, in a criminal investigation, crime analysis includes exploring and detecting crimes and their relationships with criminals. Here text based crime reports can be converted into word processing files, and these are used to perform crime matching process.

There are a lot of algorithms for data mining, in particular for classification and clustering. Classification is a process of predicting the target classes for new objects called test dataset using training dataset, which is already classified. These algorithms are easy to implement, but they fall into trouble if the test set does not have any exact match in the training set. K- Nearest Neighbor algorithm (k-NN) is a solution for this problem. This algorithm selects a k number of training records, which are most close to a new object and then find the predominance of selected records.

The execution time is a significant barrier for the classification process when the dataset becomes large. Therefore, we need to accelerate the classification process. Acceleration process can be done by parallelizing the classification algorithm. For the parallelizing purpose, we use General Purpose Graphic Processing Unit (GPGPU), since it has thousands of cores and high memory bandwidth so that thousands of threads can be run simultaneously.

II. RELATED WORK

One of the research groups [1] used GPU implementation of Brute Force k-NN search to reduce the computation time. This was implemented using nVIDIA CUDA with two kernels. In one kernel, they compute the distance matrix size of m x n which contains the distance between the training set (size of m) and testing set (size of n). This distance computation is fully parallelized since data points are independent. In the other kernel, the distance is sorted for each test data concurrently. Optimized insertion sort is used for sorting the distance. It sorts the first k elements in the array using insertion sort. Then it checks the element (y) between k+1 and m. If the value of y is less than the k^{th} element of the sorted array, they insert that element to correct place in the sorted array. In our work, we implement the finding the nearest neighbor function using sequential search.

The researchers in [2] focused on accelerating email filtration using kNN on GPU. Emails contain different elements of formatting in addition to simple text. They used MailSystem.NET open source library to extract different components of the email including the number of recipients in the "TO", "CC" and "BCC" fields, validity of the address in the "TO" and "FROM" fields, the number of occurrence of the 50 most common spam keywords in the message and so on.

They have divided their implementation into three phases: a distance calculation phase, a sorting phase and a voting phase. In the distance calculation phase, each thread of the kernel calculating the distance between query point and training point and the results are stored in a struct in GPU memory. In the second phase, they divide the distance struct according to the test data. Then those arrays send to each block to find the smallest k distance of each test data. Finally, according to the values emails are classified into Spam or not. In our work, pre-processed datasets were used for classification. The datasets were read from the file and stored in arrays. And then sequential search method is used to finding the nearest neighbor phase. Therefore, the class array did not want to duplicate with each test record's distance array.

Another research group [3] has optimized the k-NN algorithm for exact similarity search on heterogeneous CPU – GPU systems by utilizing threshold compression with partial

sort. Distance computation is done on a GPU and sorting is done on a CPU. In our work, distance calculation phase and finding the nearest neighbor phase were implemented in GPU

The above researches, serial and parallelized k-NN was implemented using same algorithms. In our research, K-NN algorithm was implemented using multiple algorithms on CPU and GPU. For example, the finding the nearest neighbor phase implemented using insertion sort, optimized insertion sort, bubble sort, and sequential search algorithms. And select the best algorithm which has minimal execution time. Therefore finding the nearest neighbor phase implemented using optimized insertion sort and sequential search on CPU and GPU respectively. And other implementations were same in both CPU and GPU.

III. BACKGROUND

A. k-NN Algorithm

k-NN is a well-known classification algorithm, which classifies a new object using the majority vote of neighbor's class. It is used in data mining applications, such as email spam filtering, content retrieval, gene expression, protein-protein interaction and 3D structure prediction and customer segmentation in online shopping websites since it gives minimal error rate and high accuracy.

The training dataset contains labeled dataset with multiple attribute values. Testing dataset is having undefined class value and multiple attribute values. And data can be text or numeric. The number of attributes should be the same for datasets.

This algorithm has three main steps.

1. Distance calculation.
   Distance can be calculated using Euclidian, Minkowski or Mahalanobis. Among them, Euclidian distance measure is used widely. The Euclidian distance is given by this equation (1),

$$D = \left(\sum (X_{(i)} - Y_{(i)})^2\right)^{1/2} \quad (1)$$

   Here X_i, Y_i are the attributes in testing dataset and training dataset respectively.

2. Finding the k- Nearest Neighbor
   In the second step, minimum k distance values will be sorted with their corresponding class values for the test data.

3. Voting
   In final phase locates the majority class value in the sorted training set and predicts that class to the test data.

B. CUDA and GPU

In early days, GPUs were mainly used for graphic processing. When nVIDIA is developing TESLA GPU architecture, they realized the programmers can use GPU like a processor and their process can be parallelizable. Therefore, nVIDIA added memory load and store instruction with additional bytes addressing to support a compiled C program.

GPUs have a high arithmetic intensity of operation and capacity to process parallel arithmetic operations. This is because it contains a larger number of ALU's than CPU and a smaller number of components for cache and flow control.

Cache memory is used to control the bandwidth requirements of applications so the data do not go back to DRAM when multiple threads that access the same memory. The bandwidth of GPU is nearly larger than 10 times or more of CPU.

In 2006, November nVIDIA introduced CUDA (Compute Undefined Device Architecture) with C/C++ Compiler, libraries and runtime software to work on GPU for parallel data computation model and develop applications.

CUDA is a parallel computing platform used for GPU. It helps to implement parallelism in the program without any knowledge on the low-level hard architecture of GPU. It is sometimes referred as Single Instruction Multiple Thread (SIMT) architectures. The Fig. 1 shows the thread organization in CUDA. There are three levels of thread abstraction:

1. Thread – Single process that executes an instantiation of a kernel.
2. Block – A 1, 2 or 3-dimensional collection of threads and they use shared memory and main memory for their process
3. Grid – 1 or 2-dimensional collection of blocks.

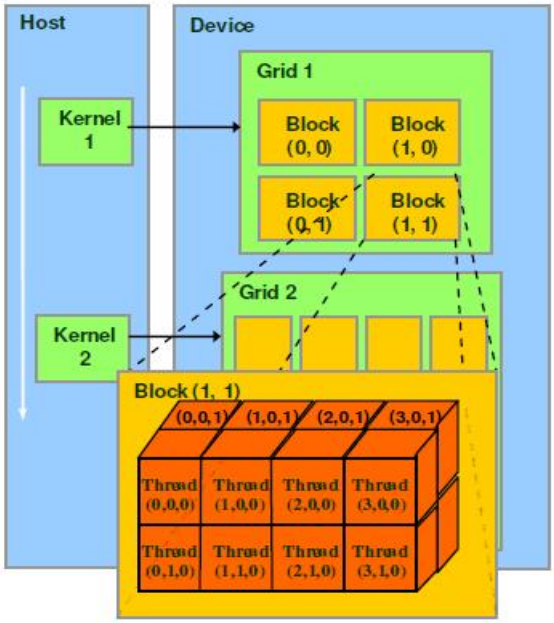

Fig.1. Thread Organization in CUDA

IV. k-NN IMPLEMENTATION IN CUDA

Initially, we implement the basic serial version of the k-NN algorithm to run on CPU using C. k-NN classification execution time in CPU becomes large when the number of record and number of attributes is very large. Distance calculation and finding the nearest neighbor functions take more than 75% of the total execution time. Therefore to reduce the execution time we need to reduce the time consumption for distance calculation and finding the nearest neighbor functions. We are using GPU to parallelize those two functions.

In CUDA, we create two kernels, one for distance calculation and another one for finding the nearest neighbor. In distance calculation, we parallelize the distance calculation between one test data and all the training dataset. But here we cannot parallelize finding the nearest neighbor for one test data. But we can find the nearest neighbor for different test data set at a time. Using this method we reduce the time consumption for distance calculation and finding the nearest neighbor to accelerate the k-NN algorithm.

A. Distance calculation kernel

Here in each block, we send one test data, and all training data and each thread calculate the distance between that test data and a training data. We handle the block in 2D thread organization with block size 16x16. Fig. 2 shows the programming structure of the distance calculation CUDA kernel. This function returns distance in 1D array. Then this array will be passed into finding the k-nearest neighbor function.

```
1 __global__ void euclideanDistance() {
2
3 int row = blockIdx.x * blockDim.x + threadIdx.x;
4 int col = blockIdx.y * blockDim.y + threadIdx.y
5 distanceId = row * testRecords+col;
6
7 if (row less than trainRecords and col less than
                                                testRecords) {
8    for (i = 0 to attributes - 1) {
9       diff <= (d_trainSet [row*attributes + i] –
                                d_testSet[col*attributes + i])
10          sum += diff * diff;
11   }
12      distanceArray [distanceId] <= sqrt(sum)
13}
14}
```

Fig .2. Programming Structure of Distance Calculation kernel

In Fig. 2, line 9 and line 10 shows the Euclidean distance calculation method. The variable `distanceId` which is in line 5 illustrates the indexing of distances between a test and train data in the output distance array.

B. Finding the k-Nearest Neighbor Kernel

In this kernel, we define the number of thread is equal to the number of testing records in the set. Here we use sequential search method. Each thread finds the index of k minimum distance values of each test data and returns a 1D array of size k for each test record to a voting function, which is implemented in CPU. Fig. 3 shows the programming structure of sequential search kernel.

```
1 __global__ void NearestNeighbour(){
2
3 int threadId = blockIdx.x * blockDim.x + threadIdx.x;
4
5 if(threadId less than testRecords){
6    for(b = 0 to < k){
7       Min <= distanceArray[threadId]
8       index = 0;
9     for(c = 1 to c){
10       if(Min greater than distanceArray[c*testRecords
                                              + threadId]){
11          Min <= distanceArray[c*testRecords + threadId]
12          index <= c
13       }
14       if((Max less than distanceArray[c*testRecords +
                                  threadId]) and (b equal to 0)){
15          Max <= distanceArray[c*testRecords + threadId]
16        }
17     }
18     distanceArray[index*testRecords + threadId] <=
                                                        Max
19     class[testRecords*b +threadId] <= trainClass[index]
20   }
21  }
22 }
```

Fig.3. Programming Structure for Finding the Nearest Neighbor

In Fig. 3, from line 6 to line 20 show the sequential search method. And line 19 show the how the index values of minimum distance stored in the final k-class output array with the size of k* number of test records.

V. EXPERIMENT SETUP AND RESULTS

The CPU program is tested on Intel® Core ™ i5-3470 CPU @ 3.20GHz processor, and the CUDA program is tested on Tesla k40:2880 cores/30720 threads, 12GB memory.

We have considered the SHUTTLE numeric dataset from Machine Learning Repository website [5]. There are 43500 training data, 14500 testing data with nine attributes and missing values does not exist. It has 7 different class values. We store this dataset in CSV file format. We divide them into various sizes of training sets and test sets to evaluate the execution time of distance calculation phase, the k-nearest neighbor phase and the k-NN algorithm.

All the below results are taken for 43500 training record and for k=5. Table I shows the execution time of distance calculation phase of the k-NN algorithm on CPU and GPU.

TABLE I TOTAL ELEXCUTION TIME BETWEEN CPU AND GPU

Number of test records	Execution time / sec		Speed Up (=T1/T2)
	CPU (T1)	*GPU (T2)*	
500	1.42	0.44	3.23
2500	7.66	0.48	15.95
4500	14.45	0.53	27.26
6500	21.99	0.60	36.65
8500	30.57	0.60	50.95
10500	41.04	0.67	61.25
12500	51.50	0.74	69.59
14500	61.65	0.80	77.06

Table I shows the total execution time of the k-NN algorithm on CPU and GPU. Here we can notice that the total execution time on GPU is decreasing when compared to the total execution time on CPU. When the numbers of records are increased by 2000, the execution time on CPU and GPU also increase. But, the increasing rate of execution time on GPU is very small compared to the CPU's time increasing rate. As a result, the speed up rate is increased. It is the ratio between the execution time of CPU and the execution times on GPU. Fig. 4 shows the execution time versus the number of records in CPU and GPU. For 43500 training records and 14500 test records GPU is 77 times faster than CPU.

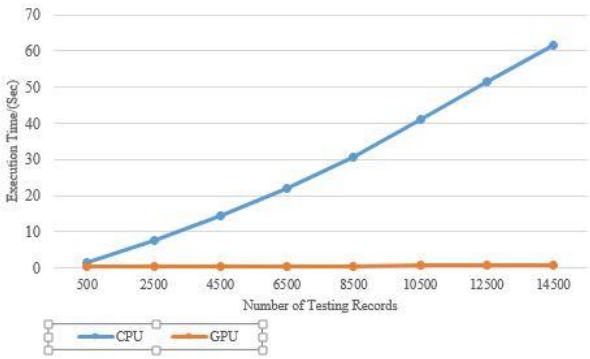

Fig.4 Execution Time VS Number of Records in CPU and GPU

VI. Conclusion

We presented a CUDA based k-NN classification algorithm implemented with two kernels, one for distances calculation and other for finding the nearest neighbor using search. The total execution time of the k-NN algorithm is reduced by 76 factors of CPU execution time for a standard dataset. The speed up factor is increasing the total size of the dataset. If the dataset is very large, then the speed up between GPU and CPU also becomes larger. The total execution time is reduced by using GPU for classification with the k-NN algorithm. Therefore, this method is more suitable for a large dataset with small dimensions.

Acknowledgment

We would like to thank the nVIDIA Research Center of University of Peradeniya who allowed us to use nVIDIA K40.

References

[1] Saravanan Thirumurukan, "A detailed introduction to k-nearest neighbor algorithm", 2010.

[2] Vincent Garcia, Eric Debreuve, Michel Barlaud, ''Fast k – nearest neighbor search using GPU'',Universit´e de Nice-Sophia Antipolis/CNRS Laboratoire I3S, 2000 route des Lucioles, 06903 Sophia Antipolis, France.

[3] Jiban K Pal, "Usefulness and application of data mining in extracting information from different perspectives", Library, Documentaion & Information Science Divition, Indian Statistical Institute, 2011.

[4] Wenbin Fang, Ka Keung Lau, Mian Lu, Xiangye Xiao, Chi Kit Lam, Philip Yang Yang, Bingsheng He, Qiong Luo, Pedro V. Sander, and Ke Yang, "Parallel data mining on graphics processors", Department of Computer Science and Engineering, Hong Kong University of Science and Technology Microsoft Research Asia, Microsoft China Co., Ltd.

[5] Statlog shuttle dataset, [Online]. Available at https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)

[6] Vincent Garcia, E´ric Debreuve, Frank Nielsen, Michel Barlaud, "k-nearset neighbor : fast gpu based implementations and application to high dimensional to high – feature matching", Laboratory dinformatique LIX, 91128 Palaiseau Cedex, France; Laboratories I3S, 2000 route des lucioles, BP 121, 06903 Sophia Antipolis Cedex, France; and Sony CSL 3-14-13 Higashi Gotanda, Shinagawa-Ku, 141-0022 Tokyo, Japan.

[7] Joshua M.Smithrud, Patric McElroy, Razvan Andony, "Massively parallel kNN using CUDA on spam – classification", Computer Science Department, Central Washington University, Ellensburg, WA, USA.

[8] Matsumoto and Man Lang Yiu, "Accelerating Exact Similarity Search on CPU – GPU SystemsTakazumi", Department of Computing, The hong Kong Polytechnic University, Hung Hom, Hong kong.

[9] D. Qiu, S. May, and A. N¨uchter, "GPU accelerated nearest neighbor search for 3D registration", in International Conference on Computer Vision Systems, Li`ege, Belgium, 2009.

Accelerating Mutual Information Analysis Based Power Analysis Attacks using the GPU

Malin Prematilake*, Buddhi Wickramasinghe†, Olitha Vithanage‡, Roshan Ragel§, Hasindu Gamarachchi¶
Department of Computer Engineering, Faculty of Engineering
University of Peradeniya
Email: *mainprematilake@gmail.com, †bcwicki@gmail.com, ‡olitha92@gmail.com, §ragelrg@gmail.com, ¶hasindu2008@gmail.com

***Abstract*—Side Channel Attacks are a popular modern cryptanalysis technique used by adversaries in embedded devices to break the security key. In these types of attacks, the attackers are keen on identifying the weaknesses of the physical implementation of the cryptosystem and utilize such vulnerabilities to extract the key. Power Analysis Attack is a form of Side Channel Attack in which, the adversary exploits power consumed by a cryptographic device during encryption to obtain the key. Mutual Information Analysis (MIA) is a concept introduced in information theory that measures the dependence between two random variables. In MIA based Power Analysis Attack, mutual information between two random variables is taken as the side channel distinguisher. Here, the two variables are physical leakages of the device and the power model based on key estimates. Since this method has more advantages to attackers compared to other methods, it is vital for cryptanalysts to find better countermeasures against this. But, due to the lack of efficient implementations it is hard for cryptanalysts to do that kind of research. In this paper, we present a methodology to accelerate MIA based Power Analysis Attacks using a GPU (Graphical Processor Unit) like NVIDIA Compute Unified Device Architecture (CUDA). Our proposed method promises to better utilize the capabilities of NVIDIA CUDA and obtain a speedup of more than 100 times compared to its sequential version.**

I. Introduction

Cryptography is a methodology used to communicate messages securely, in such a way that only the intended parties will be able to read them. In cryptography, sending party will convert the plain text message into a different form known as cipher text using a secret key. This is called encryption. At the other end, the receiver will convert the cipher text back to plain text using the same key or a different one. This reverse process of encryption is known as decryption. Usually, any cryptographic algorithm is publicly available. Therefore, secrecy of a cryptosystem is entirely dependent on the secret key. In this study, an encryption standard known as AES (Advanced Encryption Standard) is attacked [1].

Cryptanalysis is the process of identifying weaknesses in a cryptographic algorithm and retrieving the secret key using those weaknesses. There is a wide variety of methods which can be used for cryptanalysis. Those methods include traditional methods such as brute force attack and more modern methods such as differential cryptanalysis, linear cryptanalysis and integral attacks [2].

II. Power Analysis Attacks and Side channel attacks

Side Channel Attacks are also a methodology used for cryptanalysis. These attacks utilize on physical implementation weaknesses of a cryptosystem to retrieve the secret key.

A hardware based [2] or software based cryptosystem [2] is actually implemented in a physical device, access to which may be available to an adversary. Although the device may be tamper resistant, an adversary might be able to collect some valuable side channel information during the encryption or decryption process. Therefore, side channel attacks have gained more popularity.

Power Analysis Attack is a form of Side Channel Attack. Here the adversary exploits power consumption data of a cryptosystem during the encryption process to get the secret key. Power consumed by a cryptographic device depends on the data processed and operations carried out within the device [2]. For example, a significant difference in power consumption can be observed when a bit changes its value. The idea of Power Analysis Attacks is to compare physical power leakages of a cryptographic device with a set of key dependent leakage predictions in order to identify which key must have most probably given rise to the actual leakage [3]. The set of key dependent leakage predictions is usually known as a power model. Comparison between the two sets of data is done through a statistical method known as a side channel distinguisher. Types of power analysis attacks vary depending on the distinguisher.

There are mainly two types of Power Analysis Attacks,

- Simple Power Analysis (SPA) Attack
- Differential Power Analysis (DPA) Attack

and based on the distinguisher used, DPA can have several sub types such as Correlation Power Analysis (CPA) Attack and Mutual Information Analysis (MIA) based Attack [2]. Out of them, SPA is the simplest form, in which the attack is conducted using a very little number of samples [2]. On the other hand, DPA makes use of large number of power traces and statistical techniques to conduct the attack [2]. MIA based Power Analysis Attack is the most recent form of Power Analysis Attack introduced in 2008 [4].

A. Mutual Information Analysis based Power Analysis Attack

In this type of attack, mutual information between two random variables is taken as the side channel distinguisher. This method is known to have many advantages over correlation based attack [3]. A Power Analysis Attack using MIA does not require any prior knowledge or assumptions about the cryptographic device. Also, the relationship between physical leakages and power model is not necessary to be linear. Therefore, it can be used in scenarios where data dependencies are not very significant.

1) Mutual Information: Mutual Information is a concept introduced in information theory. It is a measure of dependence between two random variables. Mutual information between two random variables X and Y is, how much information about X can be obtained by observing Y [3]. When X and Y are discrete random variables, mutual information between them is given as,

$$I(X,Y) = \sum_{k=x}^{y} P(X=x, Y=y).\log(\frac{P(X=x,Y=y)}{P(X=x).P(Y=y)})$$

Here, $P(X=x, Y=y)$ is the joint probability distribution of X and Y. $P(X=x)$ and $P(Y=y)$ are marginal probability distributions of X and Y respectively. Key to calculating mutual information between two random variables is, calculating the probability density functions of joint probability between X and Y and marginal probabilities of X and Y. There are several methods of calculating probability functions, namely histogram method, kernel density estimation, data clustering and vector quantization [3].

2) MIA in the context of Power Analysis Attacks: MIA can be used as a side channel distinguisher for Power Analysis Attacks. Here, the required two random variables are physical leakages of the device and power model built based on key estimates. Key estimates are the set of all possible values that a given key can take. The power model is usually referred to as the hypothetical leakage values. There are different kinds of methods which can be used to obtain the hypothetical leakage values. Out of them, Hamming weights calculated for key estimates are used as the leakage model in this context.

III. NVIDIA CUDA

CUDA is a parallel computing platform and an application programing interface developed by NVIDIA corporation. It can be used to offload high intensive computations to the GPU [5]. The computation is distributed in a grid of thread blocks. All blocks contain the same number of threads that execute a program on the device known as the kernel [6]. A three dimensional block ID is used to identify each block. Each thread within a block can be identified by a three dimensional ID for easy indexing of the processed data and the data to be processed. Execution configuration (or the block and grid dimensions) can be set at run time and is based typically on the dimensions and size of the processed data.

Threads within a block can co-operate via a shared user managed cache memory(16 KB or 48 KB), where it lacks a similar mechanism for co-operation between blocks. This is a concern which makes programs such as histogram difficult and inefficient. Access to global memory (device's DRAM) has a high latency (order of 400-600 clock cycles) which makes I/O operations to the global memory expensive. Focused design on the kernel and execution configuration helps avoid the above mentioned latency. The performance of global memory access can be reduced severely unless access to adjacent memory locations are coalesced.

MIA is a compute intensive process. However it is made up of a set of repetitive operations. But instead of repeating them in a sequential manner, they can be executed simultaneously independent from each other. Since GPUs are built specifically for parallel programming, the MIA process can be executed much more efficiently using the GPU. In addition, MIA based power analysis attacks have many benefits for attackers compared to other attacks as mentioned above. Hence more research is now being done to discover countermeasures against this. But since MIA is a compute intensive process, testing countermeasures has become a time consuming task, which in turn reduces the efficiency of researchers. Hence speeding it up is very helpful for such work. Therefore, in this study, we have proposed an approach to make MIA based implementation much faster than the implementations that has been done so far.

IV. Related Work

Mangard [2] provides an overview of power analysis attacks.His book provides a sound knowledge of power analysis attacks, how to prepare the test bed and how to carry out an attack.

Gierlichs et al. [4] proposed for the first time that mutual information analysis can be used as a side channel distinguisher for power analysis attacks. The authors have also confirmed the proposed method using a practical experiment on an AES software implementation with a 128-bit key. However, since the experiments have been conducted only to prove the theoretical proposal, performance measurements have not been conducted. Thus, it does not discuss its efficiency compared to other methods or how it can be improved.

Charvillon and Standaert [7] in their research show how traditional statistical methods can be used with the model proposed in [4]. It is a study of how the theoretical findings given in [4] can be applied in practical situations. Experiments conducted here address the question of when to use MIA based power analysis. As mentioned in [3], correlation based power analysis attacks are much efficient when the noise factor is ignored. Therefore, here the authors have proven that MIA based power analysis attacks are more useful when the leakage model is not entirely precise. But this research only studies how efficiency can be improved by using two types of methods (histogram based and kernel density estimation based) for calculating mutual information.

Shams [5] discusses how MIA based power analysis attacks can be accelerated using a Graphical Processor Unit (GPU). An NVIDIA CUDA based device has been used in this study. Here the author has used the histogram method and how

data dependency when creating histograms limits the use of GPU in histogram calculation is clearly highlighted in this. According to the experimental results, the GPU implementation has achieved around 25 times performance improvement compared to the CPU implementation. In Hudde [8], the author has tried to improve the MIA based attack using NVIDIA CUDA. But the author has gained only 4 to 12 times of computational speed up compared to its sequential CPU counterpart. Although both these authors have been able to achieve a performance improvement compared to a CPU implementation, we expect a much better improvement through this work.

V. Problem and Motivation

Power analysis attack is one of the most studied types of side channel attacks, but the number of researches done on MIA based power analysis attack is very small and the research done on accelerating these attacks using the GPU is smaller. When we study the related work, we see that although some work has been done on MIA based power analysis attacks, there is still room left for improvement. Although Hudde [8] has achieved a speed up of 12 times and Shams [5] has achieved a speed up of 25 times, current GPU architectures provides newer capabilities which should enable a better speed up. These new capabilities cannot be utilized in old implementations. Hence a newer GPU implementation would be useful. Improving the efficiency of MIA based power analysis attacks allows cryptanalysts to identify weaknesses in current encryption standards, improve them and introduce better countermeasures. In addition, we believe that these improvements can be used in other applications where MIA is of use like in identifying cancer cells [9] and in comparing the similarities between two images [10].

VI. The MIA Algorithm

Calculation of MI value between a power model value and a power consumption data value is based on calculating marginal probability of the two variables and the joint probability between those two variables. Out of several methods proposed in [3], histogram method was chosen for the calculation of probability distributions as it is widely used and much simpler than the other methods. The algorithm for obtaining the mutual information is of five parts.

1) Generating the power model values for all bytes in each plain text sample
2) Normalizing the power consumption data
3) Calculating the marginal probabilities of the two random variables
4) Calculating the joint probability of the two random variables
5) Calculating the mutual information value

Step 1 can be done in several ways. However, in this study, the hamming weight between the plain text and the cipher text is used [7] and [3]. Steps 2 and 3 are straight forward. However, calculating the joint probability is a very computationally intensive process. Since the encryption algorithm under study is AES with a key size of 128, there are 16 bytes (key bytes) that needs to be estimated. For each key byte, there are 256 possible values. The data set used in this study consists of 200 plain text samples with 100,000 sample wave points for each sample. Since MIA values need to be calculated for each pair of data sets (i.e.[power model, power consumption data]), a total of $100,000 * 16 * 256 = 2^{12} * 10^5$ calculations must be done. This is more cleary show in Fig.1 given below.

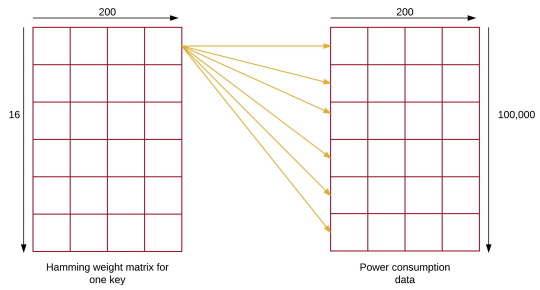

Fig. 1. Total number of MI calculations needed

Algorithm 1 Calculating joint probability

```
jProb is the joint probability of the pair
MIval is the mutual information value of the pair
marg1 is the marginal probability of power model data
marg2 is the marginal probability of wave data

pm ← matrix[256 × 16 × 200] Power model data
wd ← matrix[10^5 × 200] Power consumption data

for i ∈ {1, ..., 256} do //no. of keys
    for j ∈ {1, ..., 16} do //no. of key bytes per key
        for k ∈ {1, ..., 100000} do //wave data points
            array1 ← pm[i * 256 + j]
            array2 ← wd[k]
            jProb ← jointProb(array1, array2)
            MIval ← MI(marg1, marg2, jProb)
```

Algorithm 1 above describes how joint probability is calculated. As can be seen, calculating the joint probability is the process that requires the most time and resources due to its repetitiveness. When the basic GPU implementation was profiled using the NVIDIA Visual Profiler (NVVP) as in [11] and it showed that 70% of the computation time is spent for calculating the joint probability. Hence, attention was given to reduce the execution time of this process.

VII. Methodology and Implmentation

By implementing a parallelized version of the sequential algorithm that was initially developed, several approaches that can help to reduce the execution time were identified.

1) Better memory usage
2) Using approximated power values instead of using perfect values

A. Better memory usage

1) Design: In the sequential implementation, memory for intermediate variables (e.g. matrices) were allocated dynamically. Although this is easier in terms of implementation, it also increases the execution time. This is because allocating memory inside a CUDA kernel (dynamic memory allocation) takes a considerably larger amount of time compared to allocating it before the kernel call. Hence allocating all the memory that is needed before calling the particular CUDA kernels should be more efficient. However as mentioned before, a more complex implementation is required by this. The complexity is increased by the fact that the matrices are stored in row major order, regardless of their dimensions.

Probabilities for hamming weight matrix and the power consumption data matrix is created separately. In addition, the number of states of each dataset (i.e. the array) is stored in a separate array as well. Using the two arrays the size of each joint probability array (as shown in Fig.2) and the total size required by the joint probability matrix can be calculated (i.e. the last element of Total_Size_Array). Hence the total size needed can be allocated prior to the kernel call. As shown in Fig.2, each element of Total_Size_Array represents the total size of the probability matrix up to a given row. Hence the difference of two adjacent cells provides the size of each row of joint probability matrix.

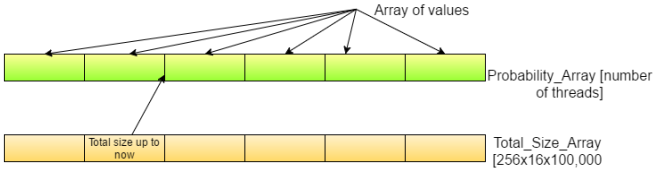

Fig. 2. Indexing of the probability matrix

2) Implementation: One of the toughest challenges in using MIA is using memory of the GPU efficiently. This is because histogram based MIA algorithm requires a lot of memory for intermediate variables. The largest of them requires an average of 58.5GB of memory, even for processing data for breaking one key out of 256 possibilities. This is for a dataset of 200 plain text samples with 100,000 sample wave points for each sample. In addition, the memory required depends on the nature of the data as well. That is, for two data sets of the same size, different amounts of memory will be needed. This is because when higher the number of different values in the set, higher the number of different bins that needs to be created for each of them.

A single thread is utilized for calculating one MIA value (i.e. one thread takes an array each from the two probability matrices and calculates the particular joint probability and the MIA value). Hence more threads utilized in one kernel call, the less time consuming the calculation will be. However, if more threads are used, then obviously the size of the matrices must be increased in order to accommodate the data. Hence, there is a limit to the number of threads that can be used. For example, the average number of threads that can be used for a dataset with 200 plain text samples with 100,000 data points is 800,000, where the data related to only 8 of the 16 key bytes of one key guess value could be processed in one kernel call.

Hence, the kernel for calculating the joint probability values and the MIA values is called repeatedly using a loop. For the above example the kernel must be called 512 times to calculate all of the MIA values. The average execution time for the data set in the above example is 14.34 minutes.

B. Using approximated values instead of using perfect values

1) Design: As it was mentioned in the previous section, a separate kernel for calculating joint probability and the resulting MI value was developed. This kernel had to be called iteratively, since joint probability distributions corresponding to all combinations of keys, key bytes and wave data points could not be stored in the GPU memory at once.

The main reason for running out of memory was the size of joint probability distributions being very large. According to the definition of joint probability, its size is given by the multiplication of the sizes of corresponding marginal probability distributions. Reducing the size of these distributions using a histogram approximation method was considered [8]. Previously, a bin was allocated to each distinct value in the data set when creating the histogram. A histogram for a power consumption dataset in this implementation had a average bin count of about 500. The same for a hamming data set was around 8. Hence the resulting joint probability distribution consisted of around 4000 bins. Instead of using one bin per value, it was decided to allocate one bin for a range of values. The two designs are given in Fig.3. In the modified design of this example, two distinct values fall into one bin. For instance, all 0s and 1s are in one bin, 2s and 3s are in one bin and so on. Thus, it is clearly seen that the number of bins has been halved. Although this might reduce the accuracy comparatively, the sizes of probability distributions will be reduced. This in turn allows utilization of more threads for joint probability and mutual information calculation.

An important point to note is that, histogram based density estimation is highly dependent on data. Therefore, when approximating the bins, accuracy will be different depending on the data. Hence, the implementation had to be done so that it is possible to change the range of bins and test.

2) Implementation: In the previous implementation where one bin is allocated to each distinct value of the dataset, first the data set was normalized so that all values were converted into integers between 0 and (maximum_value - minimum_value). Values were stored in an integer type array. The number of distinct values (Number_of_Bins) contained in each dataset (i.e. (maximum_value - minimum_value)+1) was also calculated for each data set and stored in a separate array. Histogram for a dataset is a double type array with a size equal to Number_of_Bins. The straightforward method of calculating a histogram using these data structures was given in Algorithm 2 below. Here, the Number_of_Bins is equal to the number of distinct data values in the dataset.

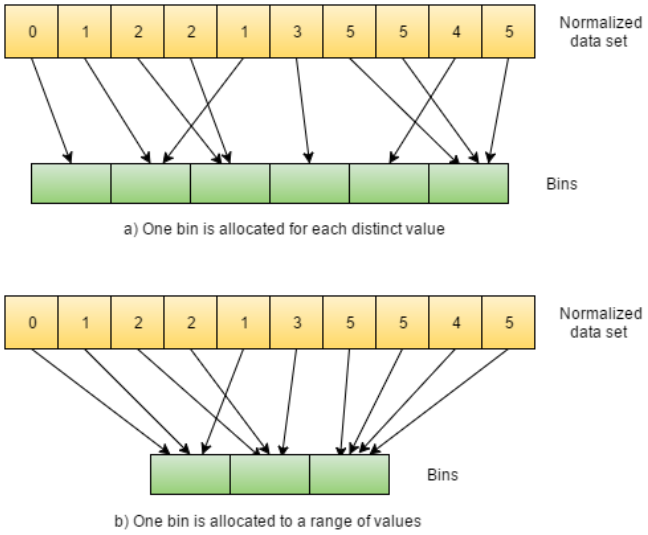

Fig. 3. Allocation of values to bins in the two designs

Algorithm 2 Basic histogram development

```
1: for Each bin number i do
2:     result[i] = 0;
3: for Each data point j do
4:     result[data[j]] ++;
```

In the new implementation, the normalizing procedure was modified in a way that the same function (i.e. Algorithm 2) can be used for histogram calculation. First, the values were converted into integers from 0 to (maximum_value - minimum_value), same as before. In this case, a constant was defined as WIDTH, which gives the number of distinct integers that falls into one bin. Number_of_Bins of the resulting histogram was calculated as given below.

$$Number\_of\_bins = ceil(((maxvalue - minvalue) + 1)/WIDTH)$$

An array with size equal to that of dataset, which stores the bin number that each data point falls into was created during the normalizing procedure. This method was directly plugged into the histogram calculation method. It should be noted that, this approximation method was only used when calculating marginal probability distribution of power consumption data. This is because, the number of bins corresponding to a power consumption dataset is several times greater than that of hamming weight data. Thus, it has a greater impact on the size of joint probability distribution. Also, since the range of values in hamming data is small, approximation will reduce the accuracy of them greatly.

After the approximation method was developed, this was combined with the memory reduction techniques discussed in the previous section. The new implementation reduced the memory requirements of joint probability calculation by a considerably large amount. Hence, the number of threads that can be allocated for the kernel execution also increased. More precisely, a grid consisting of 100,000 blocks, each having 128 threads was utilized for joint probability calculation. As a result, it was possible to process MI values related all 16 key bytes of 8 key guesses using one kernel call. The dataset used was the same as before, with 200 plain text samples with 100,000 data points.

VIII. Results and Discussion

All tests were carried out on a computer with an Intel Core i5-4570 processor, 32GB of RAM and a NVIDIA Tesla K40 GPU with 12GB of memory. It had Linux installed as the operating system. Several sets of power traces collected previously were used in these tests.

When the execution time of different GPU implementations mentioned above are compared, a gradual decrease can be seen as shown in Fig.4. Here, a data set of 200 plain text samples with 100,000 wave points for each sample is used. An average time of 4 hours and 30 minutes is required by the CPU sequential version. Compared to this, only an average of 58 minutes is required by the basic GPU version (this the first version that was created without using any of the above mentioned techniques. the CPU algorithm was parallelized as it is). However, this version does not use any of the two methods mentioned above to make it more efficient. Therefore most of the memory available in the GPU could not be utilized. Hence only 256 threads could be used in one kernel call. As a result this version is only 4.65 times faster than its sequential counterpart.

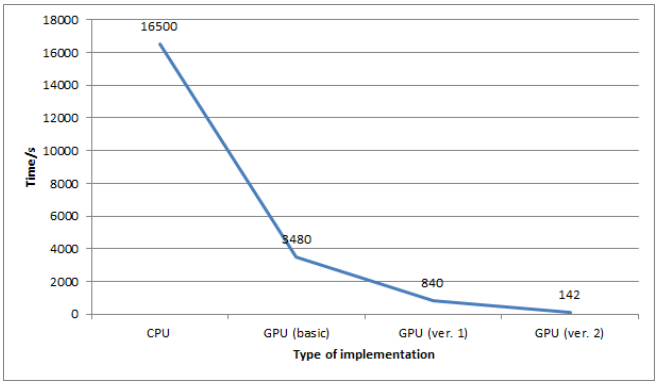

Fig. 4. Execution time of different implementations

However, less time is required by the implementation described in section VIII.a . As mentioned, it utilizes in a much more efficient manner and therefore more threads can be deployed. As a result more calculations can be done in one kernel call. For a dataset of the size mentioned above, it requires 840 seconds in average. This is a improvement of 19.2 times of the execution time, compared to the sequential version.

Execution time is further reduced by using approximated values as discussed in section VIII.b, where data related to

8 key guesses (out of 256) could be processed in one kernel call. Since this approach reduces the memory consumption further more, by combining this and the previous approach, the execution time is reduced down to an average of 142 seconds. Hence a speed up of 114 times is achieved.

By changing the width of the bin of used in the above implementation the execution time can be reduced. As shown in Fig.5 the execution time is inversely proportional to the bin width. However, this could not be improved further as the results (the key) began to be faulty for the datasets that were used.

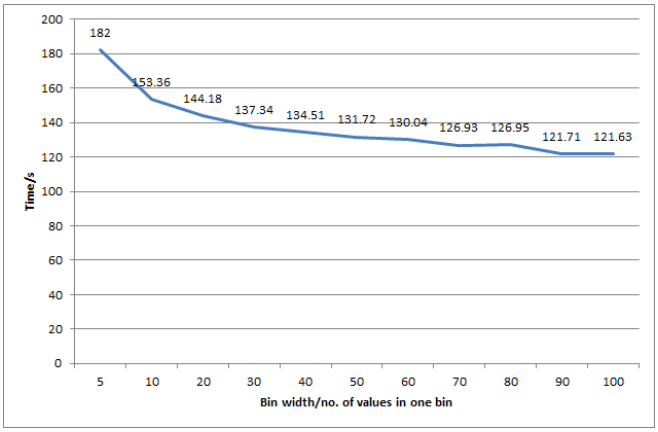

Fig. 5. Variation of execution time against different bin widths

IX. Conclusion

In this paper, an improved method for MIA based power analysis attacks using NVIDIA CUDA is presented. Though MIA is more compute intensive compared to other methods such as CPA (Correlation Power Analysis) attacks, still the fact that it does not need a linear relationship between the power consumption data and the power model makes it much more useful. It has been shown in this paper how this method could be accelerated using GPUs. Some studies have been done in the past, such as [5] and [8] but our implementation is more than four times faster than the fastest among them, mainly because our implementation uses several methods to reduce the memory usage and increase the number of parallel computations per time unit. However, we believe that more improvements could be made to this method to make it much more efficient. For this, attention must be given to reducing the execute time of joint probability density function calculation.

X. Future work

Although the histogram method has been used in this phase of the study, there are other methods for performing MIA more efficiently and out of them kernel density estimation has shown better performance than the others [3]. Kernel density estimation is non-parametric way to find the probability density function of a random variable [12]. This method considers each data point seperatly, unlike the histogram method where the data is normalised. Hence this method is expected to give much more accurate results.

Acknowledgment

The authors would like to thank the NVIDIA Corporation on behalf of NVIDIA Research Center at the Department of Computer Engineering, University of Peradeniya for letting us use the Tesla K40 GPU to successfully carry out the research.

References

[1] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[2] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008, vol. 31.

[3] L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F.-X. Standaert, and N. Veyrat-Charvillon, "Mutual information analysis: a comprehensive study," *Journal of Cryptology*, vol. 24, no. 2, pp. 269–291, 2011.

[4] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, "Mutual information analysis - a generic side-channel distinguisher," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2008, pp. 426–442.

[5] R. Shams and N. Barnes, "Speeding up mutual information computation using nvidia cuda hardware," in *Digital Image Computing Techniques and Applications, 9th Biennial Conference of the Australian Pattern Recognition Society on*. IEEE, 2007, pp. 555–560.

[6] C. Nvidia, "Compute unified device architecture programming guide," 2007.

[7] N. Veyrat-Charvillon and F.-X. Standaert, "Mutual information analysis: how, when and why?" in *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009, pp. 429–443.

[8] H. C. Hudde, "Gpu assisted mutual information analysis attacks on aes," *Bachelor Thesis, Ruhr-Universität Bochum*, 2010.

[9] G. T. Klus, A. Song, A. Schick, M. Wahde, and Z. Szallasi, "Mutual information analysis as a tool to assess the role of aneuploidy in the generation of cancer-associated differential gene expression patterns." in *Pacific Symposium on Biocomputing*, vol. 6, no. 42-51. Citeseer, 2001, p. 176.

[10] D. B. Russakoff, C. Tomasi, T. Rohlfing, and C. R. Maurer Jr, "Image similarity using mutual information of regions," in *European Conference on Computer Vision*. Springer, 2004, pp. 596–607.

[11] NVIDIA. (2015) Profiler user's guide. [Online]. Available: http://docs.nvidia.com/cuda/profiler-users-guide/index.html

[12] M. Rosenblatt *et al.*, "Remarks on some nonparametric estimates of a density function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, 1956.

A Faster GPU Implementation of Breadth-First Traversal for EDA Circuit Graphs

Dinali R. Dabarera, Himesh Karunarathna, Erandika Harshani and Roshan G. Ragel
Department of Computer Engineering
Faculty of Engineering
University of Peradeniya, Sri Lanka
Email:gdrdabarera@gmail.com, himeshsameera@gmail.com, harshanierandikanr@gmail.com, ragelrg@gmail.com

***Abstract*—With Moore's law in effect, as the complexity of digital electronic circuits increases, the amount of time spend by the electronic design automation (EDA) tools to design such circuits also increases. It brings us to the point, where we need to improve the performance of EDA algorithms to fulfill the present and the future requirements of the EDA industry. Out of many algorithms used by these tools, Breadth First Traversal (BFT) is one of the most commonly used algorithms to traverse the gates of electronic circuits.**

Therefore, we present a simple, fast and parallelizable BFT algorithm named, H-BFT. We show that the CPU implementation of H-BFT is about 75x faster than the CPU implementation of the state of the art, the Sparse Matrix Vector Product (SMVP) based BFT. Further, with the new features introduced by NVIDIA in their GPUs, we have accelerated both the state of the art SMVP based BFT implementation and our new H-BFT implementation. The best speedups we achieved via these accelerations are 150x and 25x for the SMVP-BFT and H-BFT respectively.

I. Introduction

With the doubling of transistors in the electronic circuits in every two years, the complexity of digital electronic circuits also increases day by day. This is one of the greatest challenges faced by the EDA (Electronic Design Automation) Industry. An EDA tool is a software, which is used for designing electronic circuits and systems such as printed circuit boards and integrated circuits. EDA tools are mainly used by chip designers and circuit designers to design and analyze the circuits. There are four major steps in Electronic Design Automation. They are placement, routing, optimization and post silicon validation. These steps are intrinsically difficult. For such situations, certain heuristic algorithms can be applied to find an acceptable solution first. But since the data of EDA tools are in the form of graphs, many graph algorithms are also used in EDA tools [1]. In this paper, we mainly focus on one of the most common graph algorithms, Breath-First Search, which is useful in EDA as mentioned in [1]. With the increase of the number of gates in circuits, the time taken for the circuit graph traversals also increases. This problem has become the main bottleneck in EDA [2].

There are many methods of increasing performance of EDA graph traversals. Some of them are hardware accelerations [3], use of server farms [4] and high performance computers, which costs a lot for a single EDA tool user. Therefore, use of GPU (Graphical Processing Unit) is the cheapest method of improving performance of EDA tools, which means even a circuit designer can use the graphic card of his laptop to improve the designing process. Due to the SIMD architecture of these GPUs, modern GPUs are more efficient at manipulating computer graphics, image processing and parallel computations than CPUs. At present, a GPU is found in a personal computer as a video card or embedded into its motherboard[5]. The word "GPU" was popularized by NVIDIA Corporation. With the introduction of GeForce 8 Series by NVIDIA, GPUs have become the main mode of computations against CPU [5]. It also has become a field of research called General Purpose Computing on GPU (GPGPU). The latest release of NVIDIA, "Kepler" has the worlds fastest, most efficient HPC architecture which helps in high performance computing as mentioned in [6]. There are many latest features such as unified memory architecture and dynamic parallelism that newly added to the Kepler architecture. Through this paper we will provide a better and a faster solution for the Breath-First Traversal with the use of these new features.

The rest of the paper is organized as follows: In Section II, we discuss the related work and in Section III we present the background. Then in Section IV, we describe our algorithm, the H-BFS with it's design and the implementation on both CPU and GPU. In Section V, we provide our experimental results and it's analysis. Finally we conclude in Section VI.

II. Related Work

Bell et al. had studied different data structures and algorithms for sparse matrix vector product implementation on the CUDA platform in [7], in 2008. In their study, it is mentioned that, grid-based matrix structure has occupied a performance of 36 GFLOP/s in single precision and 16 GFLOP/s in double precision on a GeForce GTX 280 GPU, while an unstructured finite-element matrix structure has occupied a performance in excess of 15 GFLOP/s and 10 GFLOP/s in single and double precision respectively.

As in [8], Bell et al. had done another research work on implementing sparse matrix vector multiplication process on throughput oriented processors. They have addressed different types of sparse matrices and verified their specific uses. According to their findings, vector approach of sparse matrices ensures contiguous memory access but lead to waste of time due to lot of computations. They also conclude that in order to effectively utilize the GPU resources, the kernels needed to have a fine-grain parallelism.

Deng et al. have done a research on increasing performance of EDA tool algorithms on GPU [9], which also describes some important irregular EDA computing patterns of Sparse Matrix Vector Product (SMVP). They had considerably ac-

celerated a SMVP based formulation of Breadth-First Search (BFS) using CUDA to get a speedup up of 10X.

One of the main GPU based EDA tool provider Rocketick, which was recently acquired by Cadence has introduced a product called RocketSim in 2011 [10]. It is a software based solution, which is installed on standard servers and accelerates leading simulators such as incisive, VCS and ModelSim. RocketSim solves the functional verification bottlenecks in chip designing by offloading most time-consuming calculations to an ultra-fast GPU based engine. It supports very large complex designs that include more than billion logic gates. RocketSim solves functional verification bottlenecks and has achieved 10X faster verilog simulations for highly complex designs.

A Breadth-First Search parallelization focused on fine grain task management is described on a research done by Merrill et al. in 2011 [11]. It has achieved an asymptotically optimal O(|V |+ |E |) work complexity. Busato et al. had implemented an efficient algorithm for Breadth First Search using Kepler GPU architectures in [12]. Their implementation of BFS has achieved an optimum work complexity.

Luo et al. have proposed an effective implementation of Breadth-First Search on GPU in [13]. They have used a hierarchical queue management technique and a three-layer kernel arrangement strategy. The experimental results have achieved up to 10 times speedup over a classical fast CPU implementation. This method is mostly suitable for accelerating sparse graphs, which are widely seen in the field of EDA.

From a survey [14] done by Deng et al. gives a detailed description about GPU architecture with its programming model and the essential design patterns for EDA computing. It shows useful information on the use of some GPU libraries such as CUBLAS [15], MAGMA[16] and CULA [16]. It also gives a brief description on some of the algorithms such as Breadth-First Search, Shortest Path, Minimum Spanning Tree, Map Reduce and Dynamic Programing.

Harish et al. have worked on accelerating large graph algorithms on GPU in [17]. This provides a faster solution for Breadth-First Search, Single Source Shortest Path and All-Pairs Shortest Path on very large graphs at very high speeds using a GPU instead of expensive supercomputers.

According to all these, we can see that many have tried to improve the performance of Breadth-First Search algorithm which is in the form of Sparse Matrix Vector Product using GPU.

III. Background - Breadth First Traversal using Sparse Matrix Vector Product

Breadth-First Search is an algorithm that can be used to traverse a graph data structure by starting from the root node and visiting the neighboring nodes first before moving to the next level neighbors. Breadth-First Search is the main graph traversal algorithm which has become the basis for many higher level analysis graph algorithms according to [11]. As in [9], BFS is mainly used to fulfill two different kinds of applications. They are levelized logic simulation and finding critical path in block based timing analysis. As many research work has been done on the Sparse Matrix Vector Product (SMVP) Implementation of Breadth-First Traversal (BFT), we built our reference model with the SMVP Implementation of BFT.

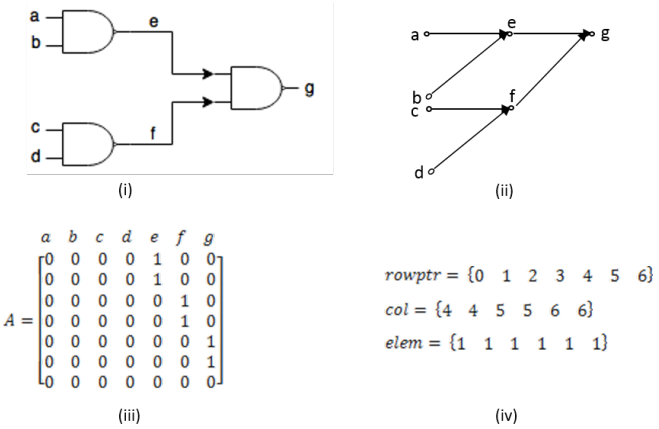

Fig. 1. Sparse Matrix and Compressed Row Format, i) Simple circuit, ii) Directed graph corresponds to i, iii) Sparse matrix of ii, iv) Compressed Row Format of iii

A. Basic Design of SMVP-BFT

EDA tools are used to design large electronic gate circuits. The circuits that are used by EDA tools can be easily interpreted as graphs, and these graphs can be stored as sparse matrices as shown in Fig. 1. There are about six ways of representing a sparse matrix [18]. They are Compressed Sparse Row (CSR) Format, Compressed Sparse Column (CSC) Format, Coordinate (COO) Format, Diagonal Format, ELLPACK format and Packet format. These formats are mainly used to save extra memory allocation for zero elements.

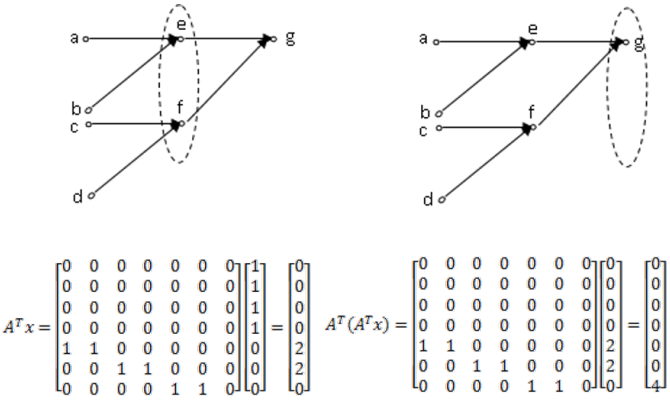

Fig. 2. BFT using Sparse Matrix Vector Multiplication

According to the related work in Section II, as shown in Fig. 2 we got to know that many have tried the Sparse Matrix Vector Product (SMVP) Implementation, in different ways. It clearly depicts how the Breadth First Search is used to traverse through the circuit. A^T is the transpose matrix of A, which is the adjacency matrix of the circuit graph and x is a vector which is the inputs to the circuit. This can have one for input pins and zero for other pins. In this example shown in Fig. 2, all three inputs were considered to be 1 and others to be

0. Each time when A^T is multiplied with the vector of the current state, the vector of the next state can be taken as the output. We can continue this iterative process until the output vector becomes a zero vector, which is our stopping condition. Therefore, through this method, Breadth-First Traversal can be easily achieved, since the output of each iteration will give the nodes/pin which are high in the same level. From this we can clearly identify the states of a electronic circuit at each timing intervals. Furthermore, this is a level by level traversal through the graph circuit using SMVP. If the number of intermediate levels of the graph is L, the SMVP-BFT can be expressed as $A^{'}..(A^{'}(A^{'} * x))$, L times until the output vector becomes a zero vector, here $A^{'} = A^T$.

```
int  *smvp_Mul(int *yHost,int *xHost, *cscValHost,
  *cscRowPtrAHostPtr, int *cscColIndexAHost,int n){
  int k=0,i=0;
  for (i = 0; i < n; i++){

  int row_start= cscRowPtrAHostPtr[i];
  int row_end= cscRowPtrAHostPtr[i + 1];
  for (k= row_start; k< row_end; k++){
       yHost[i] +=
          cscValHost[k] * [cscColIndexAHost[k]];
  }
 }
return yHost;
}
```

Listing 1: SMVP multiplication in C

B. Implementation of SMVP-BFT

We implemented the above algorithm in two ways as follows as CPU implementation and GPU implementation. The CPU version is a single threaded implementation. But for the GPU version we have used dynamic parallelism feature of Kepler GPU architecture [6]. The inputs to the SMVP-BFT are the edges of the graph and the list of input vertices to the graph circuit.

1) Serial Implementation: In the serial implementation we load the data of the edges of the graph and the input vertices into four integer vectors called $cscValHost$, $cscRowPtrAHostPtr$, $cscColIndexAHost$ and $xHost$ and do the iterative multiplication as shown in Listing 1.

In order to stop this iterative function $smvp\_Mul$, each time we are checking the output vector to verify whether it is a zero vector or not. If it is a zero vector we stop the multiplication process since we have achieved the final state.

```
__global__ void parentKernel( int * n,double *csrValA,
int *csrRowPtrA,const int *csrColIndA, int  *x,
int  *y, int * valC ){
  *valC=1;
  int i;
  int count =0;
  while(*valC){
     *valC=0;
     SMVP<<<ceil(*n/256.0),256>>>
          (n,csrValA,csrRowPtrA,csrColIndA,x,y);
     cudaDeviceSynchronize();

     count = count +1;
     checkStatus<<<ceil(*n/256.0),256>>>
                          (x,y,valC,n);
     cudaDeviceSynchronize();
     }
}
```

Listing 2: SMVP-BFT parent kernel where Dynamic Parallelism called.

2) Parallel GPU Implementation: Here we have used the same SMVP-BFT method with the use of GPU threads. From the CUDA occupancy calculator [19], we calculated that one block should contain 256 threads and the number of blocks depends on the number of vertices, n. Therefore, the number of blocks needed is $ceil(n/256.0)$. Calling GPU kernels from CPU in each iteration cause a lot of overhead and slow down the execution process. To avoid this overhead, NVIDIA has introduced dynamic parallelism in the Kepler architecture [6]. As shown in Listing 2, we use this new feature and created a parent kernel which is launched by a single thread in a single block. It launches other multiplication kernels in a loop.

As in Listing 3, a separate kernel was called inside the parent kernel to do the SMVP using GPU threads . Since this kernel is called inside the device itself, it make use of dynamic parallelism, which gives less overhead than calling from CPU.

```
__global__ void SMVP( int*  n, double *csrValA,
  int *csrRowPtrA,int *csrColIndA,int  *x, int  *y ){

  int tid= (blockDim.x * blockIdx.x ) +
  threadIdx.x;
  if(tid<*n){
       int k;
       y[tid]=0;
       for(k=csrRowPtrA[tid];k < csrRowPtrA[tid+1];k++){

         y[tid]= y[tid]+
                  ((int)csrValA[k]*x[csrColIndA[k]]);
       }
 }  }
```

Listing 3: SMVP Multiplication Kernel

IV. H-BFT: Our Breadth First Traversal

Even though we use GPU, we believe that the Sparse Matrix Vector Product is a expensive computational method, since sparse matrix is represented by three integer arrays, the multiplication process is difficult than the general matrix-vector multiplication. Therefore, in this paper we are introducing a fast and simple algorithm H-BFT for the circuit graph traversal. This is a Breadth-First Traversal with multiple input vertices which traverses the circuit level by level.

A. Design and Implementation of H-BFT on CPU

There are two main important variables involved in this algorithm. One of them is "GraphList" which is an array of structures called "Edge" of size nnz, here nnz is the number of edges in the graph. This "Edge" is a data structure that uses two integer values to store the source and the destination vertices of an edge of a graph. The other variable is called "level" which is an array of integers of size n, the number of vertices in the graph. The "level" array is used to store the levels of each vertex while traversing. These two main variables are clearly depicts in the Fig. 3.

The graph data, which is input to the H-BFS is transferred to the "GraphList". Initially the "level" array is filled with -1 values in order to indicate that these vertices are not visited. Since we know the input vertices, we make the levels of those vertices to zero in order to indicate that these vertices are inputs to the circuit.

Our CPU version is a single threaded implementation. In this case, as in Listing 4, we go through the GraphList (the array of Edge structures) one by one and check for the level

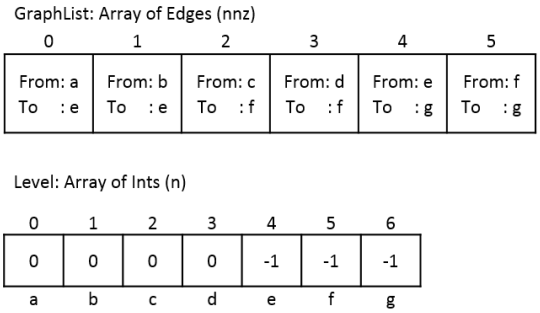

Fig. 3. GraphList and level variables for the example circuit in Fig.1

value of "From" vertex and the "to" vertex. If the level value of the "From" vertex is a non-negative value and the level value of the "to" vertex is a negative value, then the level value of the "to" vertex is updated. We continue this process, until the level array is completely filled with non-negative values.

```
struct Edge element;
for(k=0;k<*edges;k++){
        element = adjacencyList[k];
        if ((level[element.from]>=0)&&
          (level[element.to]==-1)){
                level[element.to] =
                 level[element.from]+1;
      }
}
```

Listing 4: Traversal in H-BFT

B. Design and Implementation of H-BFT on GPU

For the GPU implementation, we used the same data structures that is used in serial implementation. Furthermore, as shown in Listing 5 we use dynamic parallelism using GPU threads. From the CPU the $parentKernel$ is called using a single thread in a single block. This parent kernel is responsible for the Breadth First Traversal happening in the parallel H-BFT algorithm. This kernel will launch the children kernels $BreadthFirstSearch$ and $isLevelFilled$ repetitively till the whole level array fills with non-negative values.

```
__global__ void parentKernel(struct Edge *
adjacencyList, int * vertices, int * level,
               int * lev, int * edges){
*lev=1; int kb=0;

while(*lev){
    kb = kb+1;
    BreadthFirstSearch<<<ceil(*edges/256.0),256>>>
         (adjacencyList,vertices,level,lev,edges);
    cudaDeviceSynchronize();
    isLevelFilled<<<ceil(*vertices/256.0),256>>>
                        (level,vertices,lev);
    cudaDeviceSynchronize();
}}
```

Listing 5: parent kernel of H-BFT in CUDA which uses dynamic parallelism

As in Listing 6, $BreadthFirstSearch$ kernel is launched with nnz number of threads on GPU. The maximum block size of 256 is selected from the CUDA occupancy calculator [19] in order to achieve better performance. Therefore, the number of blocks, which is needed to launch nnz threads is $ceil(nnz/256.0)$. A single thread will go through a single Edge element in the graph dataset and check for the level values of it's "From" and "To" elements whether it is non-negative and if level value of "To" is a negative value, update the level value of "To" element similar to the CPU version. If it is non-negative, that thread will idle without doing any update. The next iteration is called only if all the threads finished their work in the current iteration. The threads will wait till one level is finished because the current level updates depends on the previous level updates. Therefore, for this process as shown in Listing 5, we used $cudaDeviceSynchronize()$ after every kernel call.

```
__global__ void BreadthFirstSearch(
   struct Edge * adjacencyList, int * vertices,
   int * level, int * lev, int * edges ){

   int tid = (blockDim.x * blockIdx.x ) +
                    threadIdx.x;
   *lev = 0;
   if(tid<*edges){

     struct Edge element =
              adjacencyList[tid];
  if (level[element.from]>=0 and
          level[element.to]==-1){
     level[element.to] =
              level[element.from]+1;
  }  }}
```

Listing 6: BreadthFirstSearch kernel of H-BFT in CUDA

V. Experimental Results and Analysis

A. Experimental Setup

The CPU implementation of both SMVP-BFT algorithms and H-BFT, which were single threaded versions were tested on a server with a 6th gen Intel i7 (6700K) processor and 32GB RAM. Their GPU implementations were tested on an NVIDIA Kepler K40 GPU card having 2880 CUDA cores and 12GB of dedicated RAM.

The graphs that we generated consist of constant number of vertices (n) and different number of edges (nnz). The EDA tools work normally work with really large circuits which has millions of gates. Therefore, we can achieve better performance by using dynamic parallelism on large circuit graphs. Due to this, we chose n as 10^6 and nnz in the range of 10^7 to 10^8 at constant intervals. We generated three sets of graphs, which fulfills three different datasets such as dataset A, dataset B and dataset C. In dataset A, the edges in the GraphList are in order of their levels from the input vertices as shown in Fig. 3 . Meanwhile in dataset B, the edges of the GraphList are in the reverse order, which means the edges of last level to the edges of the first. In the dataset C, the edges in the GraphList are placed in a random order without considering the levels of the graph.

B. GPU Speedup compared to CPU

We tested both the algorithms (SMVP-BFT and H-BFT) on the above test bench with the three sets of data. We executed one algorithm on single dataset for three times and got the average execution time since their standard deviation is very small.

1) SMVP-BFT Algorithm: According to the results we got, for all types of datasets it gave the similar execution times with the nnz. Because once the dataset is loaded to a sparse

matrix, which is in the compressed column format, it is the same multiplication process that takes place in the SMVP-BFT. Fig. 4 clearly shows that with the increase of nnz, the execution time also increases because the sparsity of the graph become decreasing and increase the number of computations that should be done.

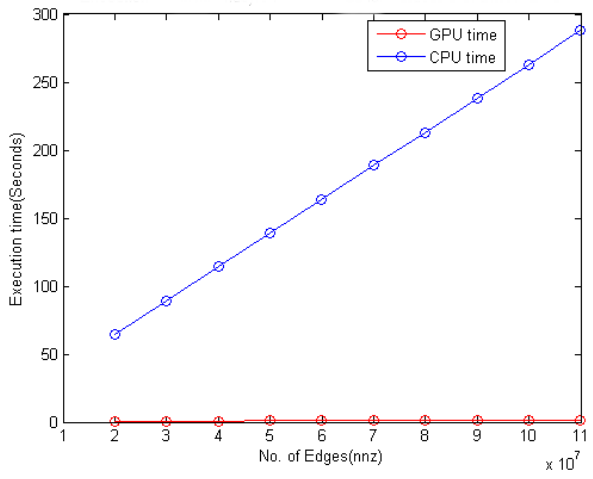

Fig. 4. Average Execution time of SMVP-BFT

From the Fig. 5, it can be seen that SMVP-BFT CUDA version has achieved a maximum speedup of 150X over its CPU implementation.

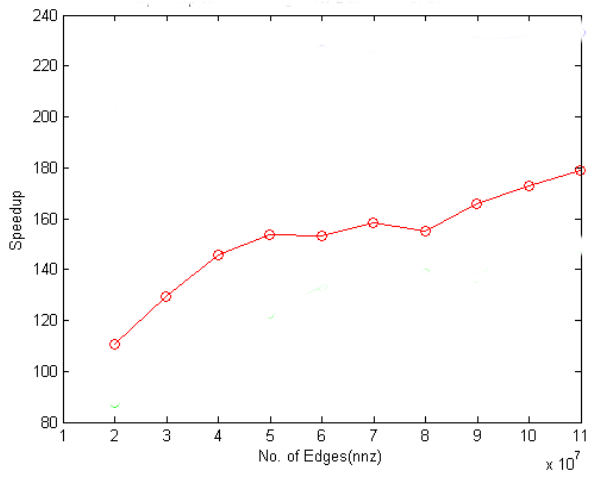

Fig. 5. Average Speedup of SMVP-BFT

2) H-BFT: Fig. 6. depicts the average execution time taken by the H-BFT to run dataset A. It clearly shows that the GPU version is faster than the CPU version. Because, when the size of the dataset getting bigger the time taken by the CPU to execute a serial H-BFT also increases. But since the edges are in order, the CPU version will complete the graph traversal in one iteration, while the GPU version takes more than one. According to Fig. 7, it is clear that in dataset B the gap between the execution time of CPU and GPU is increasing. When the GraphList is in reverse order, when we are traversing through the "Edge" element by element, since they are in reverse order the input vertices are visited at last. Therefore, to fill the "Level" array, CPU version takes more than one iterations as well as in GPU version. As seen in Fig.8, in the dataset C the gap between CPU and GPU is constant, meanwhile with the increase of nnz the CPU time increases.

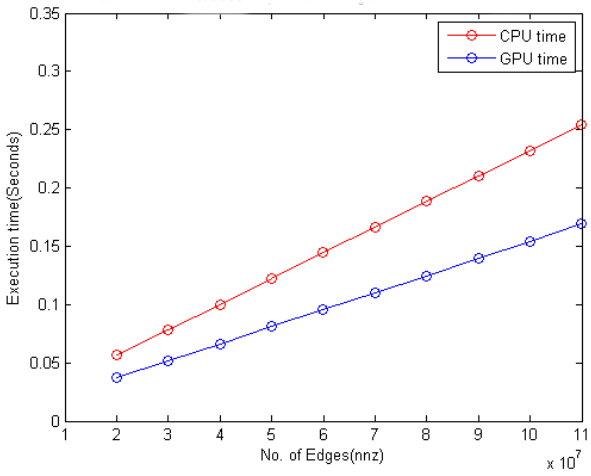

Fig. 6. Execution time of H-BFT for dataset A

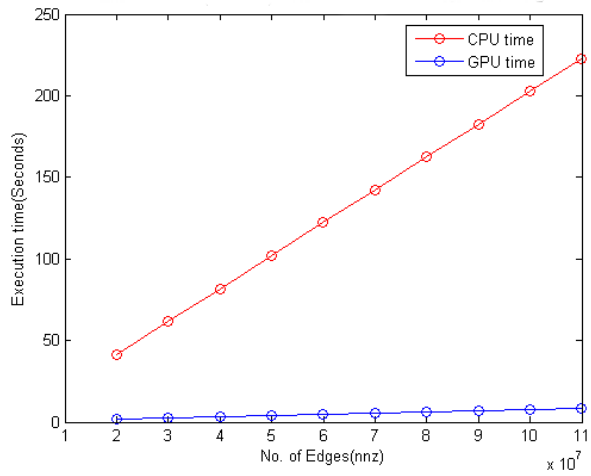

Fig. 7. Execution time of H-BFT for dataset B

When considering the speedup of GPU as depicts in Fig. 9, for dataset B, H-BFT has achieved the highest speedup of 25X over the CPU version. While the least speedup is achieved by the GPU is for dataset A, which is 2X. Furthermore, dataset C achieves around 10X speedup over the CPU version.

3) SMVP-BFT vs. H-BFT: Table I shows the execution time taken by all the algorithms when nnz=10^8. Out of the CPU versions of SMVP-BFT and H-BFT, for all the three types of datasets (A,B,C) the H-BFT is faster than the SMVP-BFT. Meanwhile for the GPU versions of SMVP-BFT and H-BFT,

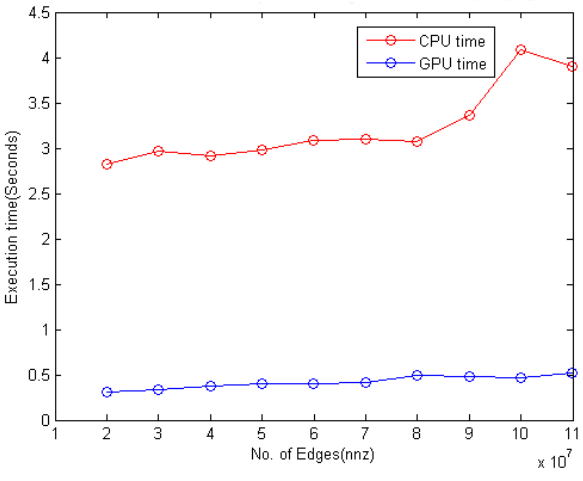

Fig. 8. Execution time of H-BFT for dataset C

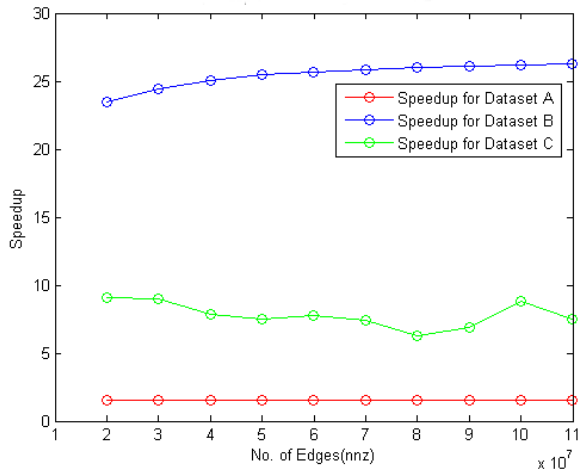

Fig. 9. Speedup of H-BFT for all datasets

for the datasets A and C , H-BFT is faster than SMVP-BFT. But for the dataset B, SMVP-BFT is faster than H-BFT.

VI. Conclusion

In this paper, we presented a comparison of two algorithms, which can be used for Breadth-First Traversal of large circuit graphs. One is SMVP-BFT, which is implemented using dynamic parallelism on GPU and the other is our newly implemented H-BFT using dynamic parallelism on GPU. Furthermore, with evidence it is proven that our solution, H-BFT is much more faster on CPU as well as on GPU (except for dataset C) than the SMVP-BFT. We believe our contribution will help to increase the performance of EDA tools where BFT is used.

Algorithm	Execution Time (s)
H-BFT CPU : dataset A	0.255
H-BFT GPU : dataset A	0.169
H-BFT CPU : dataset B	222.945
H-BFT GPU : dataset B	8.496
H-BFT CPU : dataset C	3.907
H-BFT GPU : dataset C	0.521
SMVP-BFT CPU : average All datasets	287.808
SMVP-BFT GPU : average All datasets	1.610

TABLE I
Execution time taken by all the algorithms when nnz=10^8

References

[1] Y. Deng and S. Mu, "Electronic design automation with graphic processors: A survey," *Foundations and Trends in Electronic Design Automation*, vol. 7, no. 12, pp. 1–176, 2013. [Online]. Available: http://dx.doi.org/10.1561/1000000028

[2] T. Karn, S. Rawat, D. Kirkpatrick, R. Roy, G. S. Spirakis, N. Sherwani, and C. Peterson, "Eda challenges facing future microprocessor design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1498–1506, 2000.

[3] K. Gulati and S. P. Khatri, *Hardware Acceleration of EDA Algorithms: Custom ICs, FPGAs and GPUs*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[4] TechTarget. (1999-2016) server farm (web farm, web server farm). [Online]. Available: http://whatis.techtarget.com/definition/server-farm-Web-farm-Web-server-farm

[5] N. Corporation. (2016) Introducing the new nvidia pascal architecture. [Online]. Available: http://www.nvidia.com/object/gpu-architecture.html

[6] ——. (2016) Kepler the world's fastest, most efficient hpc architecture. [Online]. Available: http://www.nvidia.com/object/nvidia-kepler.html

[7] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on cuda," Nvidia Technical Report NVR-2008-004, Nvidia Corporation, Tech. Rep., 2008.

[8] ——, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. New York, NY, USA: ACM, 2009, pp. 1–11.

[9] Y. S. Deng, B. D. Wang, and S. Mu, "Taming irregular eda applications on gpus," in *Proceedings of the 2009 International Conference on Computer-Aided Design*. ACM, 2009, pp. 539–546.

[10] Rocketic. Rocketsim. [Online]. Available: http://www.rocketick.com/rocketsim/rocketsim

[11] D. Merrill, M. Garland, and A. Grimshaw, "High-performance and scalable gpu graph traversal," *ACM Transactions on Parallel Computing*, vol. 1, no. 2, p. 14, 2015.

[12] F. Busato and N. Bombieri, "Bfs-4k: an efficient implementation of bfs for kepler gpu architectures," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 7, pp. 1826–1838, 2015.

[13] L. Luo, M. Wong, and W.-m. Hwu, "An effective gpu implementation of breadth-first search," in *Proceedings of the 47th design automation conference*. ACM, 2010, pp. 52–55.

[14] Y. Deng and S. Mu, *Electronic Design Automation with Graphic Processors: A Survey*. Now Publishers Incorporated, 2013.

[15] N. Corporation. (2016) cublas. [Online]. Available: https://developer.nvidia.com/cublas

[16] ——. (2016) Magma. [Online]. Available: https://developer.nvidia.com/magma

[17] P. Harish and P. Narayanan, "Accelerating large graph algorithms on the gpu using cuda," in *High performance computing–HiPC 2007*. Springer, 2007, pp. 197–208.

[18] N. Corporation. (2007-2015) cusparse- cuda toolkit documentation. [Online]. Available: http://docs.nvidia.com/cuda/cusparse/#matrix-formats

[19] lxkarthi. (2007) cuda-calculator. [Online]. Available: https://github.com/lxkarthi/cuda-calculator

Accelerating Lossless Compression with GPU

Dhammika Marasinghe*, Janitha Samarasinghe†, Nathasha Naranpanawa‡ and Roshan Ragel§
Department of Computer Engineering
University of Peradeniya
Peradeniya 20400 Sri Lanka
*dhammikammdb123@gmail.com, †jcsam93@gmail.com, ‡nathashanaranpanawa@gmail.com, §roshanr@pdn.ac.lk

***Abstract*—With the increasing amount of data used for various purposes, maintaining the efficiency of storing and transmitting them are proving to be a challenge. Data compression brings forward a solution to this issue.**

We perform a lossless compression of data on Graphic Processor Units (GPU) for NVIDIAs CUDA Framework, using a parallelized Lempel-Ziv-Storer-Szymanski (LZSS) compression algorithm. Algorithms were implemented using PFAC library and KMP along with LZSS. Performance results were obtained for serial and parallel CPU implementations of the LZSS algorithm as well as CUDA implementations. The improved algorithms were also benchmarked with gzip and bzip2, which are popular compression programs. We achieved a speedup of 5x for the parallelized LZSS CPU implementation compared to the serial LZSS algorithm.

Keywords: LZSS, PFAC, Compression, GPU.

I. INTRODUCTION

The computing community constantly deals with the ever increasing amount of data. The number of applications working with huge amount of data has also grown over the years. To be able to store and transmit these data in an efficient form is a challenge given that the state of resources and facilities does not increase at the same pace as the amount of data. Data compression is one of the popular solutions brought forward for this issue.

Data compression can be lossy or lossless. With lossy compression, some of the data such as redundant information will be eliminated. When the file is decompressed, only a part of the original data will be retrieved. But with lossless compression, no data is omitted and the original data is restored completely at decompression.

But again, using compression is a challenge itself, given that extra overhead and performance issues have to be considered. The execution time and the compression ratio are critical elements of compression. Many compression algorithms are subjected to these issues when performed on CPU. Lossless compression requires a considerable amount of execution time when performed on CPU, which results in a performance degradation.

The opportunity of parallelizing of compression algorithms on graphic processing units (GPUs) is useful in this context. By using NVIDIA GPUs Compute Unified Device Architecture (CUDA) Framework, parallel compression algorithms can be implemented, reducing the overhead and improving performance. Given the architectural strengths of GPUs, certain algorithms used for lossless compression have proven to be more effective on GPUs and compression can gain a very high speed-up compared to CPUs.

But compression algorithms are sequential. Hence, the speedup observed on GPU is not that significant for such sequential algorithms. For a significant performance gain, a parallelized lossless compression algorithm has to be implemented on GPU.

This paper presents an approach for lossless data compression accelaration. Section II discusses work that has been carried out regarding the same problem. Section III presents the background concepts related to the approach we have followed. Implementation details are given in Section IV and is followed by performance evaluation details in Section V. Conclusions are presented in Section VI.

II. RELATED WORK

Ozsoy & Swany (2011) [6] have examined the feasibility to use CUDA framework for LZSS lossless data compression algorithm.They presented two CPU versions and two CUDA implementations of LZSS. The CUDA implementations were optimized to utilize the global memory usage as well as the shared memory usage.Tests show that their work outperforms the serial LZSS by up to 18x. Some of the possible improvements on their work includes improved searching with better search algorithms, data structures suitable for CUDA implementation, etc.

The above work was improved upon by Ozsoy, Swany & Chauhan (2013)[7]. They have substantially improved the basic CULZSS algorithm by reducing control-flow divergence, making better use of the CUDA memory hierarchy, and overlapping CPU and GPU execution. They have also transformed the algorithm to work in a software pipeline across the CPU and the GPU, enabling it to operate on a stream of data.The throughput achieved for the improved algorithm was observed to be 34x better than the serial CPU implementation of LZSS algorithm and 2.21x better than the parallelized version. The paper discusses that in faster and more data generated cases, the step using Huffman algorithm can be a bottleneck among the other pipeline steps.

In 2014, Zu & Hua [8] presented a paper on parallelizing the LZSS algorithm using the NVIDIA CUDA framework to improve compression speed. The proposed method eliminates thread serialization by redesigning the algorithm process. Their performance evaluation focuses on compression speed, compression ratio, hash table size and decompression. For their implementation, a speedup of 2x was achieved for

compression over the improved version of CULZSS presented by Ozsoy, Swany & Chauhan (2013) [7].

Cloud et al. (2011) [9] have carried out accelerating lossless data compression with GPUs by using a modification of the Huffman algorithm. The modified implementation permits uncompressed data to be decomposed into independently compressible and decompressible blocks, allowing for concurrent compression and decompression on multiple processors. The paper presents that the GPU based encoder showed superior performance compared to the multi-core CPU encoder and single threaded CPU implementation. They also discuss how total encoding throughputs using the GPU are weighed down by the need to transfer data to and from the card.

WaveAccess, a software development company has done a research in 2011 [10] to explore the performance speedup of compression algorithms that can be achieved by utilizing the multi-core CUDA. A speedup of 4x was achieved on GPU for their implementation. Their implementation has not approached the Huffman encoding, given its sequential nature and requirements for synchronization. As future work, they have discussed implementing parallel BZIP2 for complete comparison of full-power CPU and GPU.

The paper presented by Dhore et al. (2015) [11] proposes an algorithm for video image compression using NVIDIA CUDA on GPU. The proposed framework used less storage space and also gave a better process time compared to the time required by the sequential implementation. A drawback of this implementation is that run length encoding is advantageous only when there is lots of consecutive data but is consumes more space if non-consecutive data is more.

In 2016, Tulsyan et al. [12] introduced an algorithm DKLZSS through their research on lossless compression. The algorithm introduces a dynamic KMP string matching method for parallel LZSS compression on GPGPUs. To improve the performance of the compression, they have split the input data into chunks and used a dynamic KMP algorithm on each chunk, which omits redundant processes in the original KMP algorithm when building the failure table. A main drawback of this method is that after the compression has been performed the compressed data from each thread is combined sequentially, which reduces the performance. However, tests show that this implementation outperforms the original LZSS algorithm in cases of high repetitions in the input data.

In summary, using algorithms such as Huffman, KMP and BWT could be disadvantageous given that they are sequential algorithms and hence much parallelization cannot be achieved. It is also seen that most of the performance issues of parallelized implementations are based on the efficiency of the string matching algorithms. Hence, we attempt to implement an accelerated lossless compression which focuses on the efficiency of string-matching.

III. BACKGROUND

A. Lossless Compression

We focus on lossless compression, which is a data compression technique in which the original data content of the compressed file can be retrieved exactly upon decompression. The file size is reduced, but it does not involve any loss of information. Upon reconstruction, the data will be identical to the original data of the file.

Basically, the data is rewritten in a more efficient manner when lossless compression is performed on a file. But since no content or quality is lost, the compressed file size will be larger than that of files compressed with lossy compression. In a best case scenario, lossless compression can reduce the file size by 50%, whereas lossy compression might be able to reduce it even further. Also, lossless compression is computationally expensive. Because of this, I/O performance of data transmission might not be improved even though the file size is reduced.

For applications and programs that need to process large amounts of data or improve upon them to yield more information, integrity of data needs to be preserved. Also, some programs might not tolerate lossy compression, such as text files or financial data. Lossless compression is useful in such situations.

B. LZSS Algorithm

Lempel-Ziv-Storer-Szymanski (LZSS) [2] is a dictionary encoding technique which is a derivative of LZ77. LZ77 algorithms achieve compression by replacing repeated occurrences of data with references to a single copy of that data existing earlier in the uncompressed data stream.

LZSS algorithm replaces a string of symbols with a reference to a dictionary location of the same string. For this, a minimum number matching is required, and a bit flag indicates encoding or not. The difference between LZ77 and LZSS is that while in LZ77 the dictionary reference can be longer than the substring it replaces, the reference has to be shorter than the string in LZSS.

LZSS works with two buffers: a sliding window search buffer which searches the recently encoded text and a lookahead buffer with uncoded text.

The basic LZSS algorithm is as following;

Algorithm 1 LZSS algorithm

```
1:  procedure LZSS
2:      while lookahead_buf ≠ empty do
3:          find longest match in search_buf
4:          if match and L >= min_length then
5:              output ← (Position, L)
6:              search_buf ← (move by L chars)
7:              lookahead_buf ← (move by L chars)
8:          else
9:              output ← (1, char)
10:     ▷ char is the character at the head of lookahead_buf
11:             search_buf ← (move by1 char)
12:             lookahead_buf ← (move by1 char)
13:     return output
```

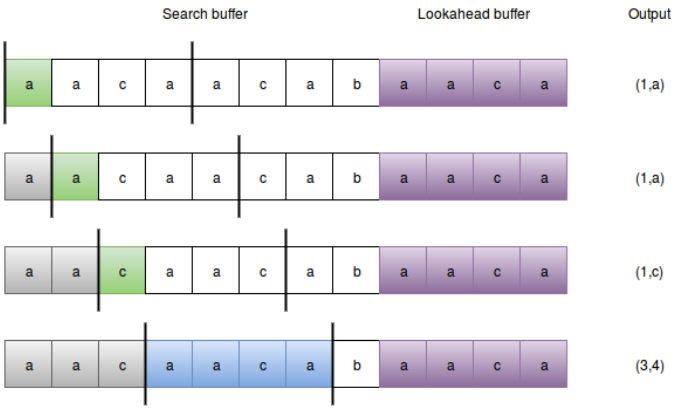

Fig. 1. An example of LZSS applied to a string

C. PFAC LIBRARY FOR AHO-CORASICK ALGORITHM

PFAC library [3] is an open source library with C level API. The abbreviation PFAC stands for the Parallel Failureless Aho-Corasick algorithm proposed in [4]. It is a variant of the well-known Aho-Corasick (AC) algorithm. PFAC accelerates pattern-matching by improving upon Aho-Corasick algorithm and achieves significant performance enhancements compared to traditional Aho-Corasick algorithm.

Aho-Corasick algorithm has two stages.

1) Converts multiple string patterns into a state machine
2) String-matching is done by traversing the state machine

PFAC improves by removing all failure transitions of Aho-Corasick algorithm and the self-loop transition of the initial state. Optimization techniques of PFAC also include reducing memory transactions of global memory and reducing latency of transition table lookup.

PFAC invokes two stages for string-matching.

1) Constructs a PFAC state machine
   The patterns to be matched are put into a state machine, with accepted states ending at each pattern construction.
2) Creates individual thread for each byte of an input stream.
   This is done in order to identify any pattern starting at the thread's starting position. The number of threads created is equal to the length of the input stream.

Performance analysis done by the developers show that PFAC achieves a significant speedup compared to traditional Aho-Corasick algorithm.

IV. IMPLEMENTATIONS

A. LZSS on CPU

1) Serial LZSS on CPU
   As a first step, a serial implementation of LZSS algorithm was executed on the CPU. The algorithm was adapted from the implementation by Okumura [5], which proposes the use of a binary tree. The algorithm works along the following steps;
   (i) A ring buffer is first initialized with only space characters, which sets the buffer at an empty state.
   (ii) Input characters are read into the ring buffer.
   (iii) Search the buffer for the longest string that matches the characters read.
   (iv) Send the length and position to the buffer if match found.
   The size of the ring buffer is set as 4096 bytes. Therefore, the position of the substring can be encoded in 12 bits. If the longest match is not longer than two characters, one character is sent as output without encoding. And the process is restarted with the next character. An extra bit is sent each time to indicate whether the output was encoded or not.
   This implementation uses multiple binary trees. As a result the speedup for the search of the longest match is increased.

- Encoding

The pseudo code for the encoding of the data is as following;
Step 1: Initialize the dictionary to a known value.
Step 2: Read an unencoded string that is the length of the maximum allowable match.
Step 3: Search for the longest matching string in the dictionary.
Step 4: If a match is found greater than or equal to the minimum allowable match length: Write the encoded flag, then the offset and length to the encoded output. Otherwise, write the unencoded flag and the first unencoded symbol to the encoded output.
Step 5: Shift a copy of the symbols written to the encoded output from the unencoded string to the dictionary.
Step 6: Read a number of symbols from the unencoded input equal to the number of symbols written in Step 4.
Step 7: Repeat from Step 3, until all the entire input has been encoded.

- Decoding

The pseudo code for the decoding process is as following;
Step 1: Initialize the dictionary to a known value.
Step 2: Read the encoded/not encoded flag.
Step 3: If the flag indicates an encoded string: Read the encoded length and offset, then copy the specified number of symbols from the dictionary to the decoded output. Otherwise, read the next character and write it to the decoded output.
Step 4: Shift a copy of the symbols written to the decoded output into the dictionary.
Step 5: Repeat from Step 2, until all the entire input has been decoded.

2) Parallel LZSS on CPU
   Given that parallelizing the LZSS algorithm on dictionary-based operations requires research which is

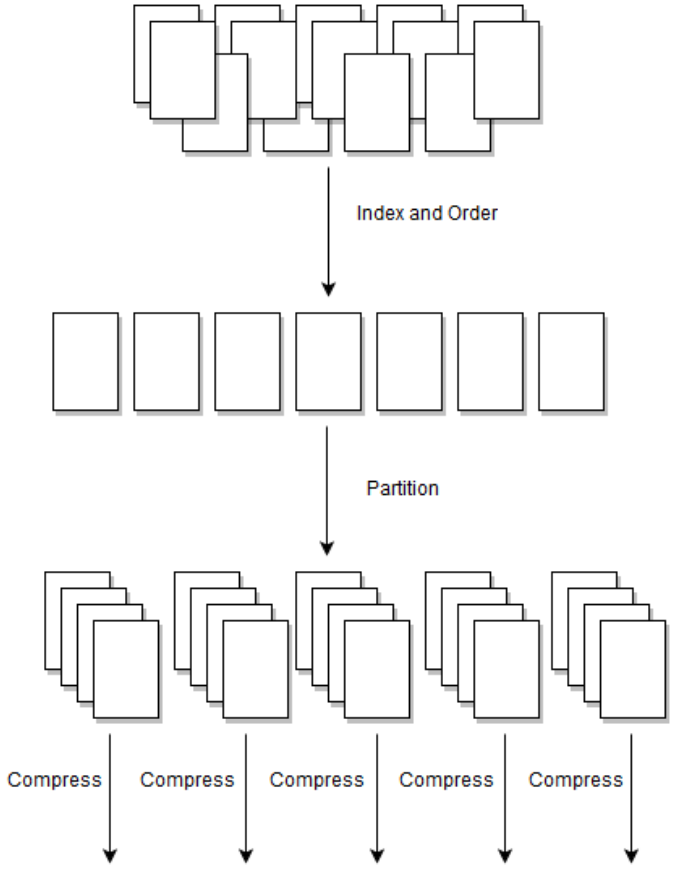

Fig. 2. Partitioning input data

out of the scope of the project, the attention was focused on using the SIMD architecture on the data to be compressed.

Considering a set of data that is very large in size, preferably a set of files for a game, each file is indexed. This will bring all the files into an order. Then, the set of files is partitioned into smaller chunks, in this case 8 sections.

Then, LZSS algorithm is executed on each of these sections of data. This process is parallelized by assigning a separate thread for each chunk of file, ensuring the performance is increased. The process is illustrated in Fig.2.

Each partitioned section of data is encoded separately using the serial LZSS algorithm. At the decoding stage, again each encoded section is decoded separately and appended in order. Since decoding does not undergo any searching process, it is fast and no optimizations are needed.

B. LZSS on GPU

1) Based on Okumura's LZSS

- Implementation Architecture

The CUDA implementation follows the same architecture of the parallel implementation of LZSS on the CPU.

As former research has been heavily based on parallelizing string comparisons, our implementation was focused on processing the file in smaller chunks thereby using the large memory and many cores of the Tesla K40 GPU in order to compress the file in parallel with minimal loss of compression ratio.

Similar to the CPU implementation, the file is read into a buffer and partitioned to a number of parts and each part is compressed in parallel in a SIMD manner. The compressed output is written to files from the output buffer once the compression is complete.

- I/O Improvements

As seen from profiling, it was found that a large amount of time was spent in reading from the file and writing to output files while compressing. The fact that we were implementing on CUDA meant that reading from a file would need CPU intervention. Due to this there was a major slowdown to be anticipated, therefore it was decided to read the entire file into a buffer then copy that buffer to the GPU. Instead of file writes as the output buffer overflows, memory was invested in a larger output buffer to store the entire encoded code.

Upon completion of all kernels, the memory was copied back to the CPU in order to be written to the output files.

- String Matching Optimizations

The implementation of LZSS that we were working on improving was implemented for optimal string matching performance with the utilization of 256 individual Binary Search Trees which hold the strings that were to be matched. Optimization of the string matching algorithm by implementing it parallely failed due to the dependency of the match in the history of matches. Therefore the BST implementation was favored over parallelization attempts as the research was focused on accelerating compression by SIMD approach.

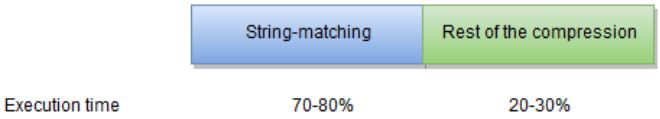

Fig. 3. Time taken for string-matching operations

Most of the performance issues of parallelized implementations are based on the efficiency of the string matching algorithm. Majority of the time is taken for string-matching operations during compression as shown in Fig.3. For a significant acceleration of compression, string-matching operations have to be optimized. Therefore, we focused on parallelizing the LZSS implementation by Michael Dipperstein [1], which provides clear separation of methods.

2) Using PFAC library
For LZSS to work, the longest match of the lookahead buffer must be found in the search buffer. But, since PFAC is a perfect pattern matching algorithm, patterns of minimum to maximum length must be passed in order to find the longest match rather than the perfect match. For this purpose the lookahead was compiled into all possible patterns as shown in Fig.4. The patterns start with 3 characters since the minimum reasonable match for compression is 3 characters. PFAC will then return an array with the match locations and corresponding pattern. Since we compile the patterns in increasing order of size, ergo match length, the largest pattern number will correspond to the longest match. We then return the longest match length and position on the search buffer back to the LZSS encoding algorithm.

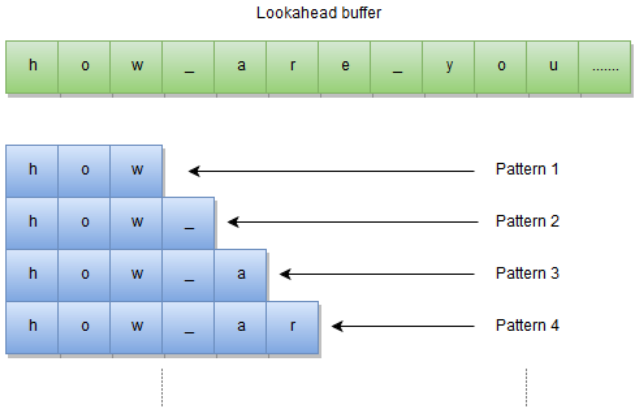

Fig. 4. Pattern compilation

3) Using CUDA threads
In this implementation, we created threads to search for a match parallely in the search buffer as shown in Fig.5.

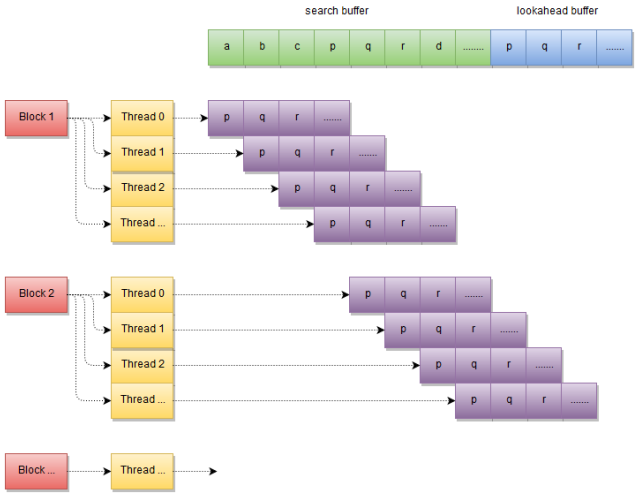

Fig. 5. Parallelized string-matching with Cuda threads

Since the search buffer is 4096 characters in length and the lookahead buffer is 18 characters in length, matching 18 characters in each thread would yeild a fast result. To attain this, we created 4096-18 = 4078 (approximated to 4080) threads in 4 blocks comprising of 1020 threads each. These threads match the entire lookahead buffer with 18 characters of the search buffer, each thread starting with an offset (blockIdx.x*blockDim.x+threadIdx.x) which is the starting point of the search on the search buffer. Once the result is returned which is an array of match length corresponding to the match position, the maximum match length is found and returned.

V. PERFORMANCE EVALUATION

A. Setup

- Testbed configurations

To evaluate how well the CUDA implementations work, we used a Tesla K40 card with CUDA version 7.5, installed on a machine with 6th gen Intel i7(6700k) CPU running at 4.0GHz and 32 GB DDR3 RAM at 2133 MHz. The CPU LZSS implementations were also tested on the same testbed.

- Data sets

The executions were benchmarked with 3 data files. The first file is a small data file of 581KB. The second is a larger file with a size of 3.2MB. The third file is much larger one with a file size of 100MB. All files are textual.

B. Results and Analysis

- Compression speed

To evaluate the performance of each technique regarding the compression speed all implementations along with gzip and bzip2 programs were executed on the benchmark files. Given below are the results obtained for time taken to compress the files;

TABLE I
EXECUTION TIME OF IMPLEMENTATIONS

Implementation	Time taken(s)		
	500kb file	3.2mb file	100mb file
Serial LZSS	0.32	0.71	20.47
Parallel LZSS (CPU)	0.04	0.15	4.06
Parallel LZSS (GPU-CUDA 512)	2.88	3.90	37.10
PFAC	128.21	770.94	inf
Serial LZSS with KMP	0.88	5.09	200.37
Parallel-matching LZSS (CUDA)	86.11	436.70	inf
gzip	0.07	0.26	5.40
bzip2	0.62	0.38	8.07

Even though we achieved a speedup of 5x for the parallelized LZSS CPU implementation compared to the serial LZSS algorithm, none of the CUDA implementations showed any favorable speedup.

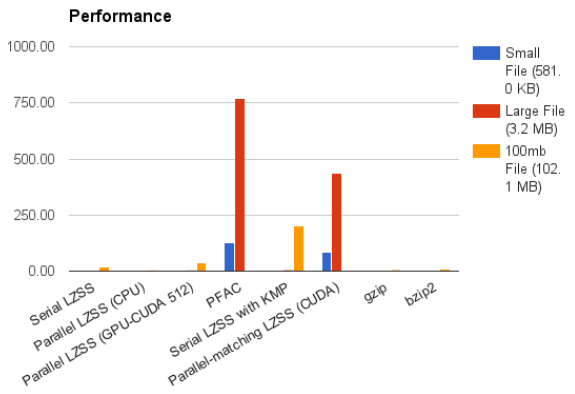

Fig. 6. Performance comparison

VI. CONCLUSIONS

Though the majority of the time is spent on string matching in the case of LZSS based compression, optimizing that using CUDA is not a viable option due to several reasons. Though string matching can be attained faster with many threads working together to find the longest match, the overhead of initializing the device, creating threads, copying data to Device and back to Host and freeing data is too high in comparison to the length of string to be matched. This is because of the sliding window principle used in LZSS. Meaning only a small part (4096 characters - known as the search buffer) of the whole file is matched with the 18 characters of the lookahead buffer. Running the entire algorithm on CUDA would also not profit in the situation seeing that a single thread on CUDA has much less resources when compared to a CPU yet the rest of the algorithm needs to run serially. In conclusion, the best method to speed up LZSS compression for massively parallel environments would be partitioning the dataset as the compression speed increases with minimal loss of compression ratio.

REFERENCES

[1] M. Dipperstein, ”LZSS (LZ77) Discussion and Implementation”, Michael.dipperstein.com, 2015. [Online]. Available: http://michael.dipperstein.com/lzss/. [Accessed: 20- Aug- 2016].

[2] ”Lempel-Ziv-Storer-Szymanski”, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Lempel-Ziv-Storer-Szymanski. [Accessed: 20- Aug- 2016].

[3] C. Liu, L. Chien, C. Lin and S. Chang, ”PFAC Library GPU-based string matching algorithm”, 2011. [Online]. Available: https://github.com/pfac-lib/pfac/blob/master/PFAC_userGuide_r1.1.pdf. [Accessed: 20- Aug- 2016].

[4] C. Lin, C. Liu, L. Chien and S. Chang, ”Accelerating Pattern Matching Using a Novel Parallel Algorithm on GPUs”, IEEE Transactions on Computers, vol. 62, no. 10, pp. 1906-1916, 2013.

[5] H. Okumura, ”Data Compression Algorithms of LARC and LHarc”, Oku.edu.mie-u.ac.jp, 1989. [Online]. Available: https://oku.edu.mie-u.ac.jp/ okumura/compression/ar002/compr2.txt. [Accessed: 20- Aug- 2016].

[6] A. Ozsoy and M. Swany, ”CULZSS: LZSS Lossless Data Compression on CUDA”, 2011 IEEE International Conference on Cluster Computing, 2011.

[7] A. Ozsoy, M. Swany and A. Chauhan, ”Optimizing LZSS compression on GPGPUs”, Future Generation Computer Systems, vol. 30, pp. 170-178, 2014.

[8] Y. Zu and B. Hua, ”GLZSS: LZSS Lossless Data Compression Can Be Faster.”, 2014. [Online]. Available: http://staff.ustc.edu.cn/ bhua/publications/GLZSS-2014.pdf. [Accessed: 20- Aug- 2016].

[9] R. Cloud, M. Curry, H. Ward, A. Skjellum and P. Bangalore, ”Accelerating Lossless Data Compression with GPUs”, Arxiv.org, 2011. [Online]. Available: http://arxiv.org/abs/1107.1525. [Accessed: 21- Aug- 2016].

[10] ”Breakthrough in CUDA data compression”, Wave-access.com, 2011. [Online]. Available: http://www.wave-access.com/public_en/blog/2011/april/22/breakthrough-in-cuda-data-compression.aspx#.Vv4qqvknJwj. [Accessed: 20- Aug- 2016].

[11] R. Dhore, D. Sapkal, P. Jadhav, S. Borkar and A. Gogawale, ”Framework for Video Image Compression Using CUDA and NVIDIAs GPU”, International Journal of Emerging Engineering R esearch and Technology, vol. 3, no. 3, 2015.

[12] V. Tulsyan, A. Sarode, A. Vasishta, T. Notani and A. Sharma, ”DKLZSS - A Dynamic KMP String Matching Method for Parallel LZSS Compression on GPGPUs”, International Journal of Computer Applications, 2016.