

PFAC Implementation Issues and their Solutions on GPGPU's using OpenCL

Chirag Agarwal
Department of CSE
MANIT, Bhopal, India

Akhtar Rasool
Department of CSE
MANIT, Bhopal, India

Nilay Khare
Department of CSE
MANIT, Bhopal, India

ABSTRACT

Aho-Corasick is a standard string matching algorithm. It can match multiple patterns simultaneously and affirmed deterministic performance under all circumstances. Aho-Corasick feed solutions to various real world applications like intrusion detection systems, text mining, search engine, multimedia and computational biology. In order to improve performance of these applications parallelization of Aho-Corasick is crucial. PFAC (Parallel Failure Less Aho-Corasick) algorithm provides high degree of parallelization in Aho-Corasick algorithm. PFAC implementation on GPGPU's architecture may consist various implementation issues. In this paper discrete implementation issues of PFAC on GPGPU's using OpenCL, their solutions and comparative analysis are discussed.

Keywords

GPGPU, AC Techniques, PFAC Techniques, Parallel AC

1. INTRODUCTION

Aho-Corasick is a popular string matching algorithm which is efficacious in espy finite number of patterns in a given text stream with a single pass[1,2,3,4]. Due to its drastic performance it is used in various applications like intrusion detection systems [5], search Engine [6] and computational biology [7].It constructs a finite state machine in the preliminary phase and that machine is used to check and detect the presence of one or more known symbol sequences inside a data set[8].

In order edify the pattern matching efficiency and to meet with real-time requirements for various applications, parallelization of AC algorithm is must [9]. To inlay this goal a parallel version of AC is developed named as PFAC (Parallel Failure less Aho-Corasick Algorithm) [10, 11, 12]. In this parallel version of Aho-Corasick, failure function and output function have been removed [13, 14, 15]. PFAC allocates each byte of an input stream for a GPU thread to identify pattern matching at the thread starting location. It uses a state machine without back-track links. This is more efficient than straight forward Aho-Corasick , because most of the threads erode in early stages of text matching.

GPGPU (General-purpose computing on graphics processing units)[16] is the utilization of a graphics processing unit (GPU), which typically handles computation only for computer graphics[17,18]. In real world GPGPU is helpful in parallelization of various applications [19]. GPGPU has given

a point where a number of real world applications can be easily implemented and can run significantly faster [20, 21].

In this paper different versions of PFAC Algorithm on GPGPUs using OpenCL are we developed, analyzed and testes. Assorted PFAC techniques are Individual thread pattern count algorithm (ITPC), Individual thread unique pattern count algorithm (ITUPC), and PFAC Local count (PFAC-LC). While contrivance different versions PFAC on GPGPU several significant problems ensued. To evacuate these problems an optimal solution is provided that is Segmented PFAC. Various issues of PFAC on GPGPUs and comparative analysis of all techniques under various conditions are comprised in this paper.

2. RELATED WORK

Parallel Failure less Aho-Corasick Algorithm is the parallel version of Aho-Corasick algorithm [11]. This algorithm doesn't use failure function. PFAC is implemented on GPGPU to enlighten multiple patterns in a given data set. It exterminates when there is no valid state. Each thread access DFA from global memory and an input character of data set is assigned to each thread. It doesn't require backtrack whenever transaction failed. This is efficient and faster version. Most of the threads annihilate in early stage of text matching [18].

In PFAC Algorithm machine is created without any failure transactions. Total number of threads will be equal to the number of text length of given data set. Each thread has variable scanning length. If a valid state occurs in previous thread then next state will be checked by next thread. So one thread may scan more than one alphabet in a time [16].

The idea behind allocation machine to each thread is important factor to enhance the efficiency of AC-algorithm. In this algorithm GPU basically concern only to find the match location so it doesn't need any backtrack. There for all the failure transaction can be removed. There are three most important reason which tells that PFAC is more superior then straight forward implementation first there is no boundary detection problem as in straight forward implementation[22]. Second worst case and avg. case time are much shorter then straight forward implementation third some threads terminate at early stage because of none valid state [15].

Machine for searching the multiple patterns into given text stream is assigned to each thread and that is used to elicit given patterns into text stream. This also improves efficiency of algorithm [18].

The process of PFAC algorithm is explained with figure 1,2 and 3. Figure 1 represents the example of DFA (a deterministic finite state automaton) for multiple pattern {HE,HER,HIS,HIM}. In this example of DFA we have not enclosed failure function of each state in this DFA. The first stage is goto function which constructs a trie of the pattern where root represents state and edge represents transition. All the strings included with extended the nodes and making a sequence. To match a string, we start from the root node and traverse the edges according to the specified input characters. The second stage involves the insertion of the next transitions. When a string match is not found, it is possible that the suffix of the previously matched string is the prefix of another string in the trie, and hence we use the next transitions to slide to a different string (branch) in the trie [23].

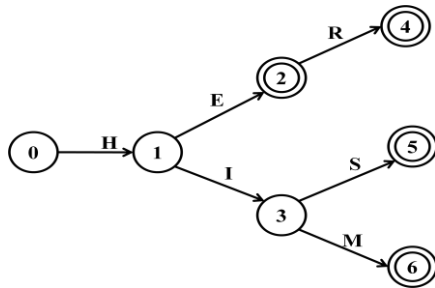


Fig 1: DFA for Aho-Corasick

DFA is placed in global memory and each thread accesses global memory to take the copy of DFA as given in figure 2. Global memory contains the DFA machine and counter. Counter is increased each time when there is a pattern match in the text stream.

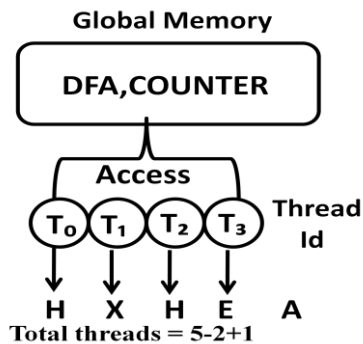


Fig 2: Assign thread to each text stream

Algorithm for searching operation is explained in the next section. Total number of threads is equal to text length - minimum pattern length of data set + 1. For example, in the below given figure 3, text length is 5, in that case total number of threads will be Text length(5) - minimum pattern length(2) + 1, that is 5. Now DFA is accessed by each thread from global memory as given in figure 2. Starting with the first thread, input character of text stream assigned to thread is H. So getting a valid state, it will jump to state first. Second thread will access two characters at a time so with previous thread it will get failed because of next input text is X. This is the most important asset of PFAC that most of threads terminate at earlier stages. Thread 2 starts and reaches at state 1, now next state is scanned by thread 3 and that is an accepted state. It will increase counter to 1. This series will fail in next stage. This procedure of Searching is given in Figure 3 and algorithm 2.1.

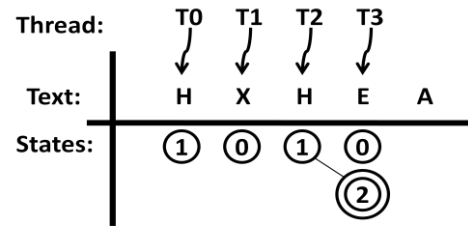


Fig 3: Searching phase

Thread termination occurs when there is no valid state for next input symbol. Here a termination table is represented which indicates the status of each thread and total pattern count.

Table 1: Status of each thread

Index	Thread ID	Terminate/count
1.	T ₀	Terminate
2.	T ₁	Terminate
3.	T ₂	Count=1 then terminate
4.	T ₃	Terminate

2.1 PFAC Algorithm

1. Start
2. Construct DFA of the given patterns.
3. $n \leftarrow$ text string length.
4. $m \leftarrow$ minimum pattern length.
5. Launch $(n-m+1)$ threads for parallel processing.
6. For each thread do
7. Thread[i] start scanning from i and go to DFA of patterns. // i is the thread index ranges from 0 to n-m.
8. If Failure occur then stop & exit threads.
9. Else If final state reach in the DFA Then
 - (a) Record the occurrences.
 - (b) Continue scanning character and make transition in the DFA.
 - (c) Go to step 8.
10. Else
 - (a) Continue scanning character and make transition in the DFA.
 - (b) Go to step 8.
11. End.

3. PROBLEMS RELATED TO PFAC IMPLEMENTATION ON GPGPU

While developing PFAC on GPGPU, abundant problems occurred. In this tier, various problems and their optimal solutions are discussed. While establishing PFAC algorithm with base technology that is individual thread pattern count (ITPC), total allocated bytes for pattern count was same as total number of threads and total number of threads was almost same as text stream length of data set. As the data set will be large enough, this technology will be feeble. While developing PFAC with synchronized variable (ITUPC), less memory is used for counter. But as the file size will expand, number of threads will be same and it will make the algorithm slower. For larger files, this will be ineffectual. In the third method (PFAC-LC), DFA and counter are placed in local memory of each thread and finally a vector counter is placed in global memory for final count. When all the values of local memory encountered in global memory to make a final count.

a lots of threads approach simultaneously. And system will efficient to handle billions of threads.

4. DIFFERENT PARALLELIZATION VERSIONS OF PFAC ON GPGPUs

4.1 Individual Thread Pattern Count Algorithm (ITPC)

This is the base technology of PFAC algorithm. In terms of processing time this method is very effective but regarding data set size it is not so efficient. This is same as malloc allocation. In this method the number of variables which is used for pattern count is equal to text length. For each count a new variable is used. If text stream will be large enough total variable size will be large and from a point the algorithm will not accept text file to remove this limitation ITPUC method is developed.

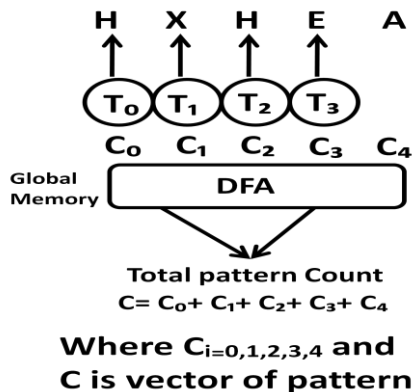


Fig 4: ITPC

In Figure 4 each thread has a separate count for storing number of occurrences of patterns for searching and then at last sum of all counts are done to calculate number of occurrences of patterns. Since allocating separate count to each thread is allocated and there are billions of thread because each letter is allocated a thread, it is impossible to implement this version of PFAC for a big data set.

4.1.1 Algorithm

1. Start
2. Construct DFA of the given patterns.
3. $n \leftarrow$ text string length.
4. $m \leftarrow$ minimum pattern length.
5. Launch $(n-m+1)$ threads for parallel processing.
6. For each thread do
7. Thread[i] start scanning from i and go to DFA of patterns. //i is the thread index ranges from 0 to $n-m$.
8. Else If final state reach in the DFA Than
 - (a) Record the occurrences.
 - (b) Continue scanning character and make transition in the DFA.
 - (c) Go to step 8.
9. Else
 - (a) Continue scanning character and make transition in the DFA.
 - (b) Go to step 8.
10. For $j = 0$ to $n-m$.

11. $\text{Count} \leftarrow \text{count} + \text{counter}[j]$. //count is total pattern counter
12. End

4.2 Individual Thread Unique Pattern Count Algorithm (ITUPC)

This method guaranteed that one variable will access the code at a time. This algorithm use atomic_add function. It provides synchronized access. It removes the limitations of ITPC but due to synchronization algorithm's processing time is increased.

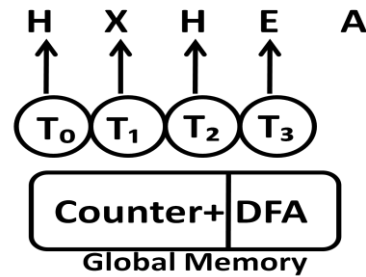


Fig 5: ITUPC

In Figure 5, count declared in global memory. Here only single count hence synchronization is needed between the threads to calculate the number of occurrences of pattern searching for. Since synchronization is used this version is less fast then ITPC version. Less memory is used for storing count here, but as the size of file of increases number of threads increases hence system is unable to handle large number of threads simultaneously.

4.2.1 Algorithm

1. Construct DFA of the given patterns.
2. $n \leftarrow$ text string length.
3. $m \leftarrow$ minimum pattern length.
4. Launch $(n-m+1)$ threads for parallel processing.
5. For each thread do
6. Thread[i] start scanning from i and go to DFA of patterns. //i is the thread index ranges from 0 to $n-m$.
7. If Failure occur than stop & exit threads.
8. Else If final state reach in the DFA Than
 - (a) Record the occurrences with unique counter. //atom_inc is used for that.
 - (b) Continue scanning character and make transition in the DFA.
 - (c) Go to step 7.
9. Else
 - (a) Continue scanning character and make transition in the DFA.
 - (b) Go to step 7.
10. End

4.3 PFAC Local Count

In PFAC with Local Count, vector is used to store the value of occurrences of patterns. Vector and DFA is placed in local memory of each thread. Vector is used to record occurrences. A final counter is placed in global memory to make total count. Problem of system to handle large number threads is retained. This version of PFAC is faster than all

others because vector is placed in the local memory instead of global memory and threads can access DFA from local memory. So access time from memory is reduced.

After the processing phase when all threads approaches to global memory for total pattern count then it was unable to handle billions of threads. So this method was unable to handle larger data sets.

4.3.1 Algorithm

1. Construct DFA of the given patterns.
2. $n \leftarrow$ text string length.
3. $m \leftarrow$ minimum pattern length.
4. Launch $(n-m+1)$ threads for parallel processing.
5. For each thread do
Thread[i] start scanning from i and go to DFA(local memory) of patterns.//i is the thread index ranges from 0 to n-m.
6. If Failure occur than stop & exit threads.
7. Else If final state reach in the DFA Than
(a)Record the occurrences in local memory with vector counter.
(b)Continue scanning character and make transition in the DFA(from local memory).
(c) Go to step 6.
8. Else
(a) Continue scanning character and make transition in the DFA.
(b) Go to step 6
9. For $j = 0$ to n-m.
10. Counter \leftarrow Counter + vector_counter[j].//count is total pattern counter which is placed in global memory
11. End

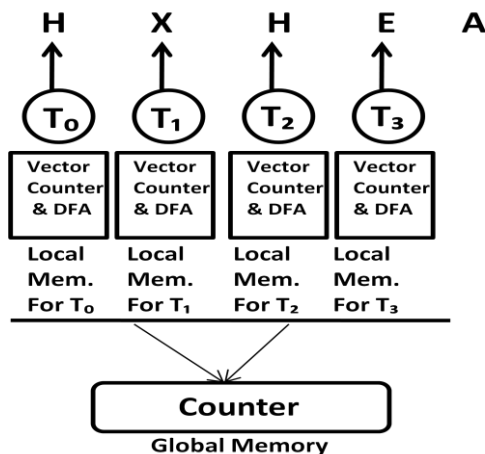


Fig. 6: PFAC-LC

5. SOLUTION TO VARIOUS PROBLEMS OF PFAC

Segmented PFAC provides solution to all issues of PFAC version. This is slower method but it provides optimal solution for large data sets. This concept is same as Paging phenomena of operating system. When the physical memory will be insufficient page out will occur. To develop this algorithm a loop is placed inside the algorithm. This method is adequate for large number of files but total execution time is increased. This version is represented by figure 7.

This version of PFAC can execute larger files also, by executing first fixed number of bytes by one kernel call and then next bytes by other kernel call, similarly we can execute whole file by multiple kernel calls. This version of PFAC takes more time in searching the patterns because of multiple kernel calls. The divided slot is optimal and based on architecture. Optimal slot is must for parallel processing.

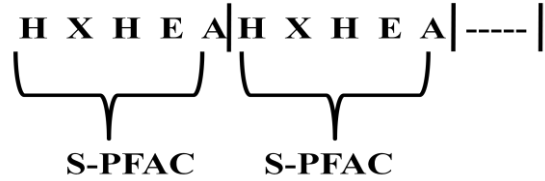


Fig 7: Segmented PFAC

However implementation of segmented PFAC cannot detect occurring patterns on boundaries. This problem is called boundary detection problem. We have discussed boundary detection problem in figure 8. As shown in figure text HE is occurring at boundary and can't be recognized by the machine. This problem can be resolved by processing overlapped computation on the boundaries. It degrades performance of the system.[24].

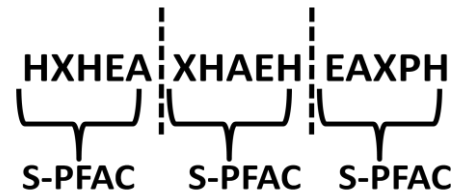


Fig. 8: Boundary detection problem

To resolve boundary detection problem overlapped computation is preferred as shown in figure 8. In this algorithm S-PFAC is overlapped to Max. pattern size-1. In our example maximum pattern length is 3 so algorithm is overlapped for 2 positions.

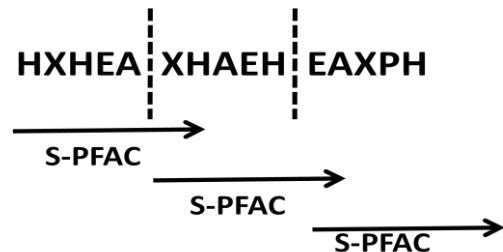


Fig. 9: Solution to boundary detection problem

Segmented PFAC method is applied on various versions of PFAC to remove the limitation of limited data sets. Modified versions are:

1. Segmented ITPC
2. Segmented ITUPC
3. Segmented PLC

These versions can simultaneously handle larger data sets. But their performances are degrading due to loop control.

Table 2 Comparison between PFAC Versions

PFAC version	Speed	Memory Allocation	Synchronization	Size of data set that can be executed
ITPC	Very fast	Large memory allocation	Not used	Small Data Sets(about 50mb)
ITUPC	Fast	Very small memory allocation	Used	Large Data Sets(about 100mb)
PFAC-LC	Very fast	Moderate memory allocation	Used	Large Sata Sets(about 100mb)

5.1.1 Algorithm

- 1) Start
- 2) Text Length←Length of Text String.
- 3) Segment size←a factor(text length)//depending upon the GPGPU architecture for optimum performance segment size is more suitable for their architecture.
- 4) For i = 0 to k do//k is a k.segmentsize←textlength
- 5) Thread[i] start scanning from i and go to DFA(local memory) of patterns.//i is the thread index ranges from 0 to n
- 6) If Failure occur than stop & exit threads.
- 7) Else If final state reach in the DFA Than
 - (a)Record the occurrences.
 - (b)Continue scanning character and make transition in the DFA.
 - (c) Go to step 6
- 8) Else
 - (a) Continue scanning character and make transition in the DFA.
 - (b) Go to step 6.
- 9) Stotal[i]= count
- 10) For i = 0 to k
- 11) Total count += stotal[i]

6. EXPERIMENTAL RESULT AND COMARITIVE ANALYSIS

Comparisons between PFAC versions is given in table 2. This table compares the versions of PFAC in terms of speed memory allocation and size of file.

PFAC versions implemented on different SIMD architecture and provide massive improvement in pattern matching efficiency.

6.1 Experimental Environment

Processor: Core i3
 RAM: 4 GB
 OS: Windows 7
 Language: Visual C++ runs on Visual Studios 2008
 GPGPU: AMD Radeon HD 6800 series
 Language (parallel implementation): OpenCL

6.2 Experimental Data for PFAC

Text File: Text Size of 50 MB, 100 MB and 200 MB having large number of occurrences of patterns.

Pattern File: One File pattern length of 20 patterns

Here we are taking total number of work items equal to text stream length of data set without setting any local workgroup size.

6.3 Experimental Results

Table 3: Experimental Results basic PFAC versions

Datasets(MB)	ITPC	ITUPC	PFAC-LC
50 MB	390 ms	510 ms	331 ms
100 MB	1000 ms	1010 ms	990 ms
200 MB	1850 ms	2010 ms	1760 ms
300 MB	2340 ms	2540 ms	2000 ms

The experimental results given in table 3. The graph for this table plotted in figure 10. Results have been taken for various lengths of data sets and corresponding execution time in seconds is calculated. Comparative analysis is represented in graph with sort of lines.

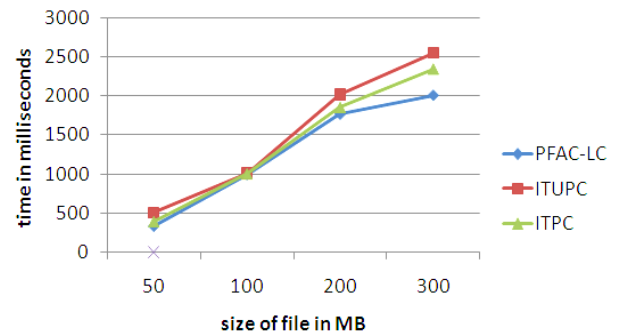


Fig. 10: Experimental results of basic PFAC versions

Table 4 : Experimental results of segmented PFAC versions

Datasets(MB)	S-ITPC	S-ITUPC	S-PLC
50 MB	1793 ms	1944 ms	1737 ms
100 MB	2145 ms	2433 ms	2104 ms
200 MB	2698 ms	2941 ms	2529 ms
300 MB	3007 ms	3437 ms	2949 ms

Table 4 represents results for segmented PFAC versions. Segmented PFAC is applied on all versions of PFAC. Graphical representation to this table is shown in figure 11.

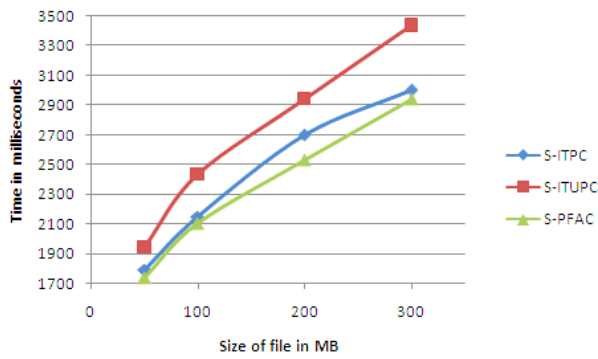


Fig. 11 : Experimental results of segmented PFAC versions

Table 5 represents data for pattern size 20,40,60,80,100. Data set size is 100 MB. Experimental results calculated for basic version of PFAC. Graphical representation for this data is displayed in figure 12.

Table 5: Execution time for various size patterns in milliseconds (data set: 100 MB)

Version	Pattern Size				
	20	40	60	80	100
ITPC	1060	1074	1071	1094	1098
ITUPC	1409	1439	1422	1471	1494
LC	997	1008	999	1021	1027

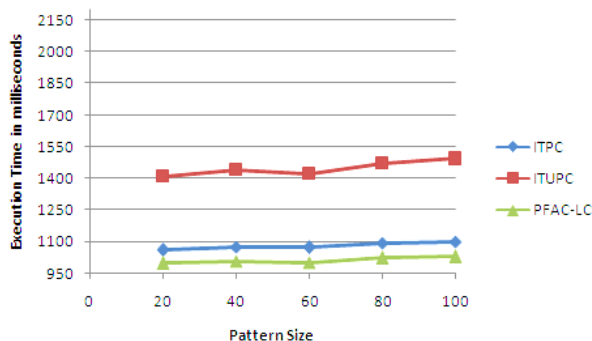


Fig. 12 : Experimental Results for various size patterns in millisecond(data set : 100 MB)

Table 6 represents data for various segmented PFAC versions. Patterns size is 20,40,60,80,100. Data set size is 100 MB. Experimental results calculated for segmented versions of PFAC. Graphical representation for this data is displayed in figure 13.

Table 6 : Experimental Results

Version	Pattern Size				
	20	40	60	80	100
S-ITPC	2145	2396	2763	2608	3001
S-ITUPC	2433	2671	3051	2909	3241
S-LC	2104	2308	2715	2581	2916

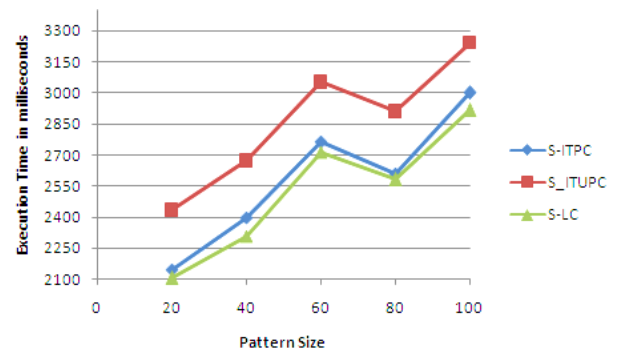


Fig. 13 : Experimental Results for segmented versions of PFAC for various size patterns in ms(data set : 100 MB)

7. CONCLUSION

Peculiar techniques of PFAC on GPGPU and their issues implemented and analyzed in this paper. Execution time of PFAC Local count was fastest but unable to handle larger data sets. To blot out these issues a new technique is endeavored that is Segmented PFAC which makes enfeeble to all the issues of PFAC techniques. Segmented PFAC was capacious to handle larger data sets. With the help of segmented PFAC actualized segmented versions of all PFAC versions in this paper. We have tested all the basic and segmented versions on different size of patterns and recorded the variations. Segmented versions are efficient for larger data sets but execution time is disparaged.

8. REFERENCES

- [1] A.V. Aho and M. J. Corasick, "Efficient String Matching: An aid Bibliographic search". In Communication of the ACM Vol. 18, issues 6, pp.-333-340, 1975.
- [2] Tumeo, O. Villa and D.G. Chavarria-Miranda, "Aho-Corasick String Matching on Shared and Distributed-Memory Parallel Architectures", Vol. 23, Issue 3, pp. 436-443, march 2012.
- [3] Cheng-Hung Lin and Shih-Chieh-Chang, "Efficient pattern matching algorithm for memory architecture", Vol. 19, issue 1, pp. 33-41, January 2011.
- [4] Chengguo Chang and Hui Wang, "Comparison of Two-Dimensional String Matching Algorithms" In the proc. International Conference on Computer Science and Electronics Engineering (ICCSEE), Vol. 3, pp. 608-611, march 2012.
- [5] S. Antonatos, et al., "Generating realistic workloads for network intrusion detection systems," Proc. 4th Int'l Workshop on Software and Performance, pp. 207-215, 2004.
- [6] Ziv Bar-Yossef and Haifa, "Efficient Search Engine Measurements", ACM Transactions on the Web (TWEB), Vol. 5, issue 4, October 2011.
- [7] Fang Xiangyan, Xiong Tinggang, Ding Yidong and Yuan Youguang, "The research and improving for multi-pattern string matching algorithm", In the proc. IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS), Vol. 1, pp. 266-270, Oct. 2010.
- [8] D. Lee, Yannakakis and Mihalis, "Testing finite-state machines: state identification and verification", IEEE Transactions on, Vol. 43, issue 3, pp. 306-320, Mar 1994.

- [9] Raphaël Clifford, Markus Jalsenius, Ely Porat and Benjamin Sach,” Pattern matching in multiple stream”,in the proc. 23rd Annual conference on Combinatorial Pattern Matching, pp. 97-109,2012.
- [10] R. Takahashi, U. Inoue, “Parallel Text Matching Using GPGPU”, in the proc. 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), pp. 242-246, Aug. 2012.
- [11] C. Lin, et al., “Accelerating String Matching Using Multi-Threaded Algorithm on GPU,” Proc. IEEE Global Telecommunications Conf.,pp. 1-5, 2010.
- [12] J. D. Owens, et al., “A Survey of General-Purpose Computation on Graphics Hardware,” Computer Graphics forum, Vol. 26, No. 1, pp. 80-113, 2007.
- [13] C. Lin, C. Liu, L. Chien, and S. Chang,” Accelerating Pattern Matching Using a Novel Parallel Algorithm on GPUs”, IEEE Transactions on computers, vol. pp, issue 1.
- [14] Zha Xinyan and S. Sahni,” Multipattern string matching on a GPU”, In the proc. IEEE conference on Computers and Communications (ISCC), pp. 277-282, July 2011.
- [15] Tran Nhat-Phuong, Lee Myungho, Hong Sugwon and Minh Shin,” Memory Efficient Parallelization for Aho-Corasick Algorithm on a GPU”, IEEE 14th International Conference on High Performance Computing and Communication, pp. 432-438, June 2012.
- [16] Jungwon Kim, Honggyu Kim, Joo Hwan Lee and Jaejin Lee,” Achieving a single compute device image in OpenCL for multiple GPUs”, Proceedings of the 16th ACM symposium on Principles and practice of parallel programming, pp. 277-288,2011.
- [17] Hyeran Jeon, Xia Yinglong and V.K. Prasanna,” Parallel Exact Inference on a CPU-GPGPU Heterogenous System”, In the proc. 39th International Conference on parallel Processing (ICPP), pp. 61-70,Sept. 2010.
- [18] Liang Hu, Che Xilong and Xie Zhenzhen,” GPGPU cloud: A paradigm for general purpose computing”, Tsinghua Science and Technology, Vol. 18, issue 1, pp. 22-23, Feb. 2013.
- [19] web resource-www.gpgpu.org
- [20] Xinyan Zha and Sartaj Sahni,” GPU-to-GPU and Host-to-Host Multipattern String Matching on a GPU”, IEEE Transactions on Computers,Volume 62, Issue 6, pp. 1156-1169,2013.
- [21] M. Potel,” A decade of applications Computer graphics]”, Computer Graphics and Applications IEEE, Vol 24, Issue 6, pp. 14-19,Dec. 2004.
- [22] <http://www.ece.ncsu.edu/asic/p183-dharmapurikar.pdf>
- [23] <http://www.ieee-icnp.org/2006/papers/s5a4.pdf>.
- [24] Cheng-Hung Lin, Sheng-Yu Tsai, Chen-Hsiung Liu, Shih-Chieh Chang and Shyu, J.-M.,” Accelerating String Matching Using Multi-Threaded Algorithm on GPU”,In the proc. Of IEEE conference on Global Telecommunications Conference (GLOBECOM 2010), pp. 1-5, Dec. 2010.