Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 7, July 2014, pg.261 – 267

RESEARCH ARTICLE

Faster Multipattern Matching System on GPU Based on Aho-Corasick Algorithm

Mr. Gaurav Bhamare, Prof. Satish Banait

PG Student, Department of Computer Engineering, KKwagh Institute of Engineering Education & Research Centre, Nashik, University of Pune, Maharashtra

Department of Computer Engineering, KKwagh Institute of Engineering Education & Research Centre, Nashik Uuniversity of Pune, Maharashtra

gauravbhamare07@gmail.com; banaitss@yahoo.com

Abstract— GPU Computing Have attracted lots of attention due to their large amount of data processing. The algorithm proposed in this paper is use for exact pattern matching on GPU. Among some famous algorithms, the Aho-Corasick Algorithm match multiple pattern simultaneously. a Traditional Aho-Corasick Algorithm matches the pattern by traversing state machine, known as Deterministic finite automata(DFA).Signature matching is important Technique in virus/worm detection, but Aho-corasick algorithm was developed only for string and virus/worm signature could be in regular expression . In this research work new guidelines are proposed for an efficient GPU adaptation of Aho-corasick algorithm for regular expression matching. Also several techniques are introduced to optimization on GPU, including reducing global memory access, storage format for output table. To evaluate performance proposed system will use SNORT virus database. Proposed algorithm Implemented on NVIDIA GTX-680 Graphics card using CUDA.

Keywords— Aho-Corasick, CUDA, GPU, SNORT, Graphics processing Unit, Pattern Matching

I. INTRODUCTION

In Multipattern Matching algorithm, we have to report all occurrence of pattern in given string. Multipattern string matching use in number of application such as network intrusion detection, digital forensics, natural language processing[2]. For example, Snort is open source network intrusion detection system which contained thousands of pattern that are match against packet in network for virus/worm signature detection. In the case of Snort we have to search thousands of pattern per packet in very small time. Due to increasing number of attack this traditional sequential pattern matching technique is inefficient. In the past year their many approaches have been proposed to accelerate pattern matching process. This approaches are classified into logic architecture and Memory architecture[5][6]. In logic architecture the attack pattern are stored on the logic circuit that are implemented on FPGA i.e. Field programmable gate array.in memory architecture we draw a state machine of patterns and traverse the state machine to find the pattern.

SNORT which is popular open source intrusion detection system also uses a Aho-Corasick pattern matching algorithm for detection[3][8]. SNORT uses efficient pattern matching algorithm hence its run time is independent on pattern length and liner to length of target string. File carving is the process of reassembling computer files in

the absence of file system metadata. In scalpel we use single pattern matching algorithm hence its run time linear to product of no of pattern and the target string length[7].

There are several attempt to improve performance of multi pattern matching using parallelism. For Example Xinyan Zha[12] compare performance of Aho-Corasick algorithm on CPU and GPU. Xinyan Zha also gives different parallel approaches of Aho-corasick algorithm depending on pattern storage. Huang et al.[13] implemented a Wu-Manber multiple-pattern matching algorithm on GPUs and achieved speedup twice as fast as the traditional Wu-Manber algorithm. Peng et al.[14] proposed GPU based web page matching system using advanced Aho-Corasick algorithm ,the proposed algorithm is 28 time faster than original Aho-Corasick algorithm which used in SNORT[8]. Mu et al[15] developed efficient GPU based router application and proposed GPU based routing table lookup solution which is delivered higher throughput than previous CPU based solution.

In this paper we proposed the Aho-Corasick multipattern matching algorithm for regular expression matching through use of GPU. To efficiently utilized GPU power, proposed algorithm usages several optimization technique like reducing global memory accessing ,CSR representation for storing state transition table of Aho-Corasick algorithm, more use of GPU shared memory for thread communication etc.

II. GPU AND CUDA

Graphic Processing Units (GPUs) have been developed to cope with the high performance requirements of graphical and animation tasks. They have different architecture than Central Processing Units (CPUs)[16], stressing floating point operations, fine grained concurrency, and high data rate memories. They have usually taken the form of co-processors of the CPU.

Recently it has been recognized that GPUs can carry out at high speed data parallel operations and more in general it has become possible to carry out General-Purpose computing on Graphic Processing Units (GPGPU or GP2). Usually GPUs are dedicated to graphic tasks like pixel shading, or a mixture of graphic and computing tasks[17], now have appeared GPUs that just do computing. In this discussion we will focus on the GPUs by NVIDIA and the C extension, Compute Unified Device Architecture (CUDA) used to program them. Since there are over 100M NVIDIA cards in use, your computer may have one so that you can experiment with CUDA. The following figure describes the processing of a CUDA program.

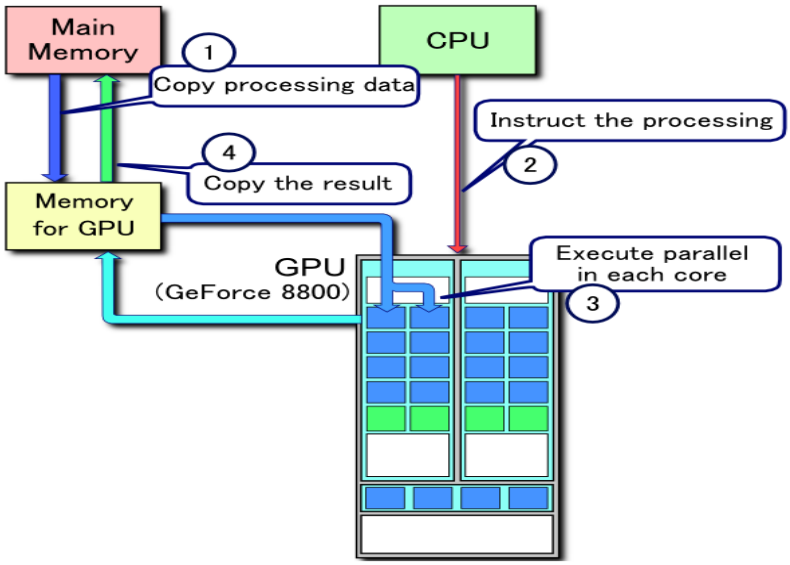

Fig 1:Processing Flow between GPU and CPU

- In the CPU, our first task is to move data to Memory of GPU from main Memory of CPU. In this, call does not return until all data has been fully transferred.
- Then we send GPU command (using cuda programming language)that GPU will execute .In this operation control is return as soon as command are execute.
- The GPU execute the commands that have been received(Process data parallels).

- After data is computed at GPU we retrieve that data in CPU by Coping from device memory to host memory(device is GPU and host is CPU). The CPU needs to know that the commands have been completed before retrieving the results. This can be done with cuda thread synchronization property.

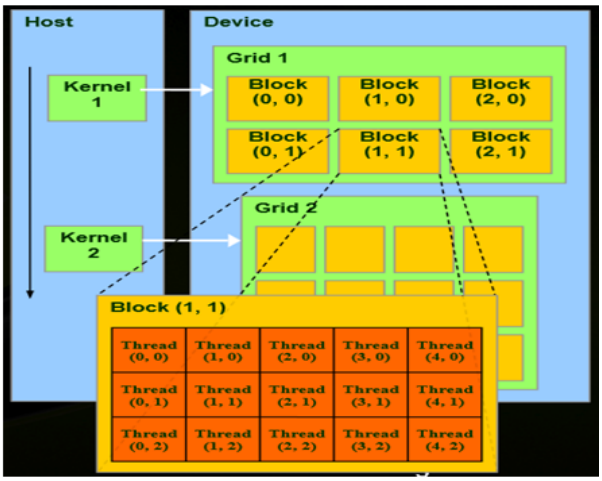

Fig 7.2 : GPU Thread ,Block & Grid

A CUDA program consists of code that will be executed on the host and code that will be executed in the device(GPU). The device code is launched by the host .The Device code contains Grid, Block ,Thread. Grid is one or two dimensional array ,block in one or two or three dimensional array with 512 thread[15]. A device can execute a single grid at a time.

III. AHO-CORASICK ALGORITHM

The Aho-Corasick algorithm was proposed in 1975 by Alfred V. Aho and Margaret J.Corasick[1] , and this is the most effective multi pattern matching algorithm. Aho-Corasick (AC) is a multi-string matching algorithm, meaning it matches the input against multiple strings at the same time. Multi-string matching algorithms generally pre-process the set of strings, and then search all of them together over the input text. The algorithm works in two parts. The first part is the building of the tree from keywords you want to search for, and the second part is searching the text for the keywords using the previously built tree (state machine). Searching for a keyword is very efficient, because it only moves through the states in the state machine. If a character is match, goto() function is executed otherwise it follows fail() function. The pattern matching machine is constructed by starting from the root node and inserting each pattern one after another[1][4]. The algorithm works as follow:

- Start at Root Node(0)
- For each pattern in P do
  - If the path end before the pattern ,then continue adding edge and state in pattern.
  - Once the pattern is identified then mark its as final state.

The time taken to search pattern is linear to pattern length .the search algorithm is as follow:

- Start at root node (0)
- For each character in i put string follow the path of constructed tree
  - If we reach to final state node then pattern is present.
  - If path terminated before reaching to end then that pattern is not present.

The Aho-Corasick Algorithm uses three function :

- The goto() function g(state, input_symbol) is the next state from current state on receiving input symbol.
- The failure function f(q). for $q \neq 0$, is the next state in case of a mismatch i.e. Failure link .
- Output function out(q) gives set of pattern that found at state q

Figure 3 show AC state machine for pattern (ABED,ABCD,EABD,AB). As shown in Fig.3, states 2, 4 and 6,10 are the final states of the patterns "AB," "ABED," and "ABCD," "EABD "while states 4 represent the final state of the pattern "ABED ." The internal state 2 becomes a final state because the pattern "AB" is a suffix of the pattern "ABED." Therefore, the state machine matches the pattern "AB" when the state machine reaches state 4. For example, consider the case where we wish to match an input stream containing "MNPSQABEABD" from the AC state machine in Fig. 2.

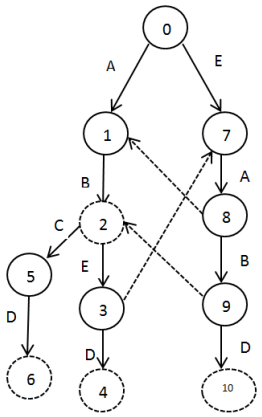

Fig 2: AC State Machine For Pattern ABED,ABCD,EABD,AB

The AC machine starts from state 0, and remain in that state for input 'M','N','P','S','Q' because AC machine don't have any valid transition from 0 state for these input. For next input 'A' it travels to state 1, and then reaches state 2 for input 'B' that is the final state of the pattern "AB". Then for next input 'E' it goes to state 3, for next input 'A' state 3 don't have any valid transition hens AC machine take failure transition and goes to state 7,next it goes to state 8 for 'A', for input 'B' it goes to next state 9,and for next input 'D' it goes to state 10 which is final state for pattern 'EABD'.Hence we get the pattern 'AB','EABD'. In summary, the AC algorithm matches all patterns in O(n) time for processing an input stream of length n[9].

IV. PROPOSED ALGORITHM

A. parallel Aho-Corasick Algorithm on GPU

Proposed work modifies the traditional Aho-corasick algorithm .

Algorithm:

Input : DFA state transition Table, Set of patterns $\{P_1,P_2,P_3..P_n\}$, Input Text T.

Output: Locations where the patterns occur In T.

Begin

- Declare n thread one for each byte.
- Curent_state=0
- Pattern_length=0
- Number_of_ pattern=0
- **For** cursor=start_of_ string To end_of_ string
- **If (**DFAtable[current_ state][T[cursor]].next state ≠0) then
    - **If(**DFAtable[current_state][T[cursor]].isFinal=0) then
    - current_state=DFAtable[current_state][T[Cursor]].next state
    - pattern_length=pattern_length +1
    - **else**
    - match_Position=cursor-pattern length
    - match_ state=current_state
    - num_Pattern=num_Pattern +1
    - **else**
    - pattern_length=0
    - cursor_state=0
- end

B. *Optimization of Device Memory*

Two important task in DFA matching is reading the input data and fetching next state from state table. This memory transfer can take lots of time.in general memory latency is hide by using several threads in parallel .multiple thread can utilized memory by overlapping data with computation. In traditional Aho-Corasick algorithm matrix is used to store state transition table. In parallel approach of Aho-Corasick algorithm, proposed algorithm use CSR representation of Sparse matrix. In Aho-Corasick algorithm the state transition table is stored in matrix, this matrix is sparse matrix i.e. a matrix where most of entries are zero.it is very inefficient use of computer memory(it is not useful to store may zero values in computer memory) and more important is computer programed need more time to run this code i.e. unnecessary computation is required ,because of these

reason proposed system use Compress sparse row format for storing state transition table in parallel AC algorithm .sparse matrix required 3 array in CSR format :Values[N],Columnindex[N],Rowindex[N]
Values[N]:This array Contains the values of non-zero element.
Columnindex[N]: This array contains column index of non zero element
Rowindex[N]:This array Contains the row-index range of non zero element
Storage space required normal matrix of N element is 'N x N' and for CSR, 'mnz+N+1' where 'mnz' is no of non-zero element in matrix and 'N' is no of Colum. In this research proposed system uses The NVIDIA CUDA Sparse Matrix library (cuSPARSE).In cuSPARSE there are collection of basic linear algebra subroutines which use for sparse matrix and using this subroutines the matrix provide up to 8x faster performance.

c. *Host Memory Optimization*
Time required to transfer the data between CPU and GPU is more, proposed system reduce number of transaction between CPU and GPU by using page lock memory. The page lock memory offer better performance as swapping is not their ,it also access directly at GPU using DMA.Hence by using page lock memory improve overall performance by reducing cost of data transfer to and from GPU.

V. RESULT

A. *Performance Measure of Serial Algorithm*
We test the performance of serial implementation of Aho-corasick algorithm against Brute force attack, Boyer Moore Algorithm, Hoorspool Algorithm. Here we use different number of pattern in length and size .Following result are tested on Intel 4th generation coreI5 Machine. Fig 3 shows time required to find the pattern in given packet .

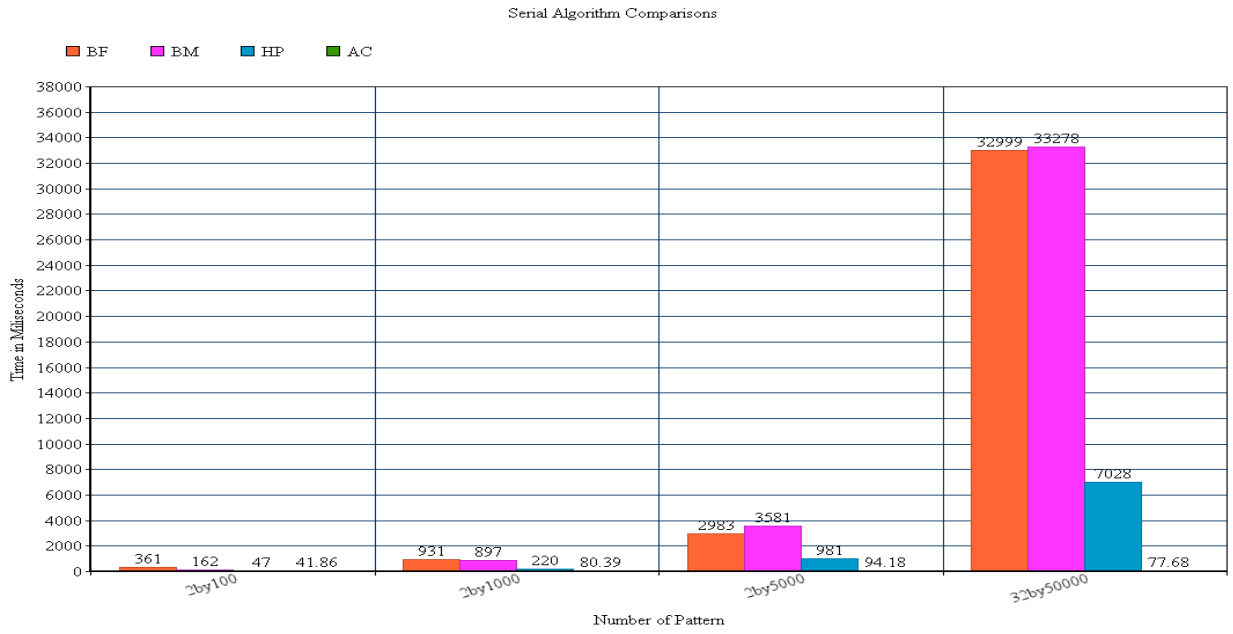

Fig3:Performance Measure of Serial Algorithm on CPU

B. *Performance Measure of Parallel Algorithm*
Parallel implantation of Aho-Corasick algorithm performance is tested on NVIDIA Gtx-680 graphics card with different number of pattern and input packet. here pattern length is varying from 2 to 32 and size is varying from 100to 50000. We implement Open Mp version of Aho-Corasick algorithm on Intel 4th generation core I5 machine and compare performance with CUDA based parallel version of Aho-Corasick algorithm for different pattern and packet. Table 1 shows the time comparison between two different parallel implementation of same Aho-Corasick algorithm.

Table I.CUDA based parallel AC vs OPenMP AC

Pattern Length	Number of Packet	AC-CUDA	AC-OpenMP
Pattern 2by100	Cuda5000	255 µs	5353 µs
Pattern 2by1000	Cuda5000	269µs	682 µs
Pattern 2by5000	Cuda5000	258 µs	588 µs
Pattern 32by50000	Cuda5000	689 µs	12685 µs
Pattern 2by100	Cuda50000	1944 µs	4070 µs
Pattern 2by1000	Cuda50000	988 µs	2300 µs
Pattern 2by5000	Cuda50000	773 µs	2186 µs
Pattern 32by50000	Cuda50000	581 µs	8273 µs

C. *Minimizing data Transfer using Pinned Memory*

Host (CPU) data allocations are pageable by default. The GPU cannot access data directly from pageable host memory, so when a data transfer from pageable host memory to device memory is invoked, the CUDA driver must first allocate a temporary page-locked, or "pinned", host array, copy the host data to the pinned array, and then transfer the data from the pinned array to device memory. data transfer rate can depend on the type of host system (motherboard, CPU, and chipset) as well as the GPU. As you can see, pinned transfers are more than twice as fast as pageable transfers.

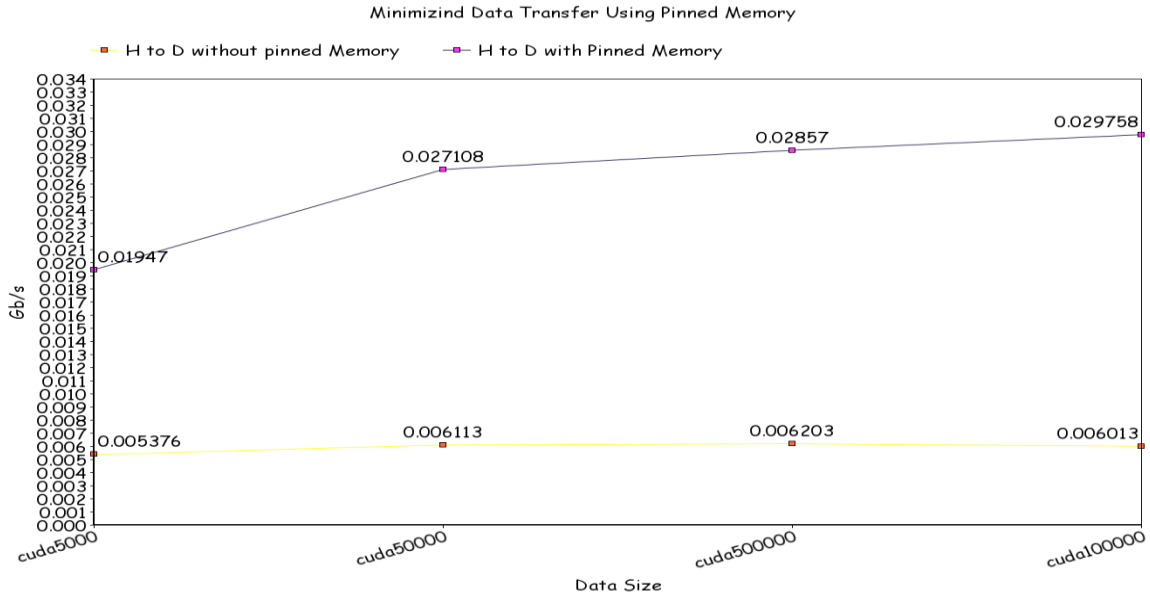

Fig 4:Minimizing Data Transfer Using Pinned Memory

D. *GPU Multistream*

In CUDA data transfer is done by default stream i.e. stream 0.The default stream is different because it is always synchronise with respective operation.no operation in default stream is began until operating in other stream is not completed .here we called one GPU default stream and one CPU stream as GPU call is asynchronous i.e. after calling kernel function it will immediately return to the next instruction ,using this as advantage we send more data for pattern matching and get better result as compare to single GPU stream. Following results are obtained by using Midstream i.e. CPU stream and GPU stream.

Table II:GPU Multistream Performance Measure

Packet on GPU	Packet on CPU	Pattern Size	Time on GPU	Time on CPU
5000	500	2by100	0.37ms	0.002016ms
5000	5000	2by100	0.37ms	0.634560ms

VI. CONCLUSIONS

GPU provide high computing power then CPU. Parallel Implementation of Aho-Corasick algorithm reduces the time required for pattern matching on large datasets. Proposed algorithm works on String and regular expression. Proposed work also uses various optimization technique on host and device. In serial algorithm Implementation Aho-Corasick shows better performance as compare to other algorithm. We also tested parallel performance of Aho-Corasick algorithm using different language and we can say that Parallel Cuda implementation of Aho-Corasick is gives better performance as compare to OPEN-MP version of Aho-Corasick.

ACKNOWLEDGMENT

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Kepler GTX 680 GPU for this research.

REFERENCES

[1] A.V. Aho and M.J. Corasick. Efficient String Matching: An Aid to Bibliographic Search. Communications of the ACM, 18(6):333–340, 1975.

[2] X. Chen, B. Fang, L. Li, and Y. Jiang. WM+: An Optimal Multi-pattern String Matching Algorithm Based on the WM Algorithm. Advanced Parallel Processing Technologies, pages 515–523, 2005.

[3] GNU Grep. Webpage containing information about the gnu grep search utility. Website, 2012. http://www.gnu.org/software/grep/.

[4] G. Navarro and M. Raffinot. Flexible Pattern Matching in Strings: Practical On-line Search Algorithms for Texts and Biological Sequences. Cambridge University Press, 2002.

[5] M. Crochemore, A. Czumaj, L. Gasieniec, T. Lecroq, W. Plandowski, and W. Rytter. Fast Practical Multi-pattern Matching. Information Processing Letters, 71(3-4):107 – 113, 1999.

[6] M. Crochemore and W. Rytter. Text Algorithms. Oxford University Press, Inc., 1994.

[7] Z. Zhou, Y. Xue, J. Liu, W. Zhang, and J. Li. MDH: A High Speed Multi-phase Dynamic Hash String Matching Algorithm for Large-Scale Pattern Set. Information and Communications Security,4861:201–215, 2007 .

[8] Snort. Webpage containing information on the snort intrusion prevention and detection system. Website, 2010. http://www.snort.org/.

[9] S. Dori and G.M. Landau. Construction of Aho Corasick Automaton in Linear Time for Integer Alphabets. Information Processing Letters, 98(2):66–72, 2006.

[10] CUDA Zone. Official webpage of the nvidia cuda api. Website, http://www.nvidia.com/object/cuda home.html.

[11] N Wilt. The CUDA Handbook: A Comprehensive Guide to GPU Programming. Addison-Wesley Professional, 2013.

[12] Xinyan Zha and Sartaj Sahni," Multipattern String Matching On A GPU".Communications of the ACM, 20(10):762–772, 1977.

[13] N. Huang, H. Hung, S.Lai et al, A GPU-based Multiple-pattern Match-ing Algorithm for Network Intrusion Detection Systems, The 22nd International Conference on Advanced Information Networking and Applications, 2008

[14] J.F. Peng, H. Chen, and S.H. Shi, "The GPU-Based String Matching System in Advanced AC Algorithm," Proc. Int'l Conf. Computer and Information Technology (CIT '10), pp. 1158-1163, 2010.

[15] S. Mu, X. Zhang, N. Zhang, J. Lu, Y.S. Deng, and S. Zhang, "IP Routing Processing with Graphic Processors," Proc. Design, Auto-mation Test in Europe Conf. Exhibition (DATE '10), pp. 93-98, 2010.

[16] Ken Thompson," Programming Techniques: Regular expression search algorithm", June 1968

[17] John E. Hopcroft and Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Publishing, Reading Massachusetts, 1979.

[18] M. O. Rabin and D. Scott. Finite automata and their decision problems. IBM Journal of Research and Development, 3(2):114–125, 1959.

