Overview and Analysis of GPU Acceleration for Regular Expressions *

Roman Gajdoš

Slovak University of Technology in Bratislava

Faculty of Informatics and Information Technologies

xgajdosr@stuba.sk

25. september 2023

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

1 Introduction

Pattern matching is widely used in a variety of different domains. Regular expressions have become a prevalent tool for text processing and sanitation due to their flexibility, conciseness, and vast support in most programming languages [5]. They appear in approximately a third of open-source projects [6]. They are employed in technical fields, ranging from database querying [11], texts editors[1], web scraping [9] to network security, such as deep packet inspection [3],

[*] Semestjrálny projekt v predmete Metódy inžinierskej práce, ak. rok 2023/24, vedenie: MSc. Mirwais Ahmadzai

[1] https://neovim.io/doc/user/change.html#%3Asubstitute

and bioinformatics [10], among others.

Regular expressions are implemented using finite automata, in either deterministic (DFA) or non-deterministic (NFA) form, each with their respective advantages and drawbacks. Each of them has their own advantages and disadvantages [16, 24, 26].

In many applications, regular expressions are applied to large amounts of input data, or require a fast response, or both. It stands to reason that efficiency in both memory and speed is the key to optimal use [21]. Here, the question is how to achieve the greatest possible efficiency for a given problem that is addressed by the regular expressions.

The processor's capacity to execute multiple expressions simultaneously is notably restricted, even in the current era of multicore processors [12]. However, its frequency and cache memory speed prove excellent for handling small datasets. For tasks that require more extensive parallelism, FPGAs (Field-Programmable Gate Arrays) or ASICs (Application-Specific Integrated Circuits) have been used. The problem is that they are slow to configure [22] and inflexible to change [8, 15].

In recent years, GPUs with their extensive parallelism, computational capabilities, and high memory bandwidth have become prevalent in numerous computing system. They have scaled at a faster rate than CPUs, providing significant computing power [15, 18]. APIs were created to allow General Purpose Graphics Processing Units (GPGPU) to accelerate processing in supported applications, replacing shading languages and simplifying their use for programmers. Two popular APIs are Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL) [7].

In this paper, we investigate a variety of GPU-based regular expression execution methods and conduct a comparative analysis of their strengths and weaknesses. Our research begins with a thorough examination of regular expressions in 3. This is followed by a comparison of their representations of finite state automata forms in 3.1. We then move into parallel computing platforms, such as CUDA and OpenCL in 3.2.1.

By combining these findings, our investigation aims to provide a comprehensive overview of GPU-accelerated regular expressions, utilizing previous studies to provide an in-depth comparative analysis in section 4 and 5. Result from this was evaluated in the 6 section. Finally, we conclude with a summary of our findings in section 7.

2 Objective and methodology

At present, we are unaware of any comprehensive analysis of the available materials on the acceleration of regular expressions on GPU. We have therefore decided to review the available data to see if there are any conflicting statements or claims that have evolved over time or are no longer valid. The outcome ought to be an in-depth, up-to-date article showing when to use GPU regular expression acceleration, what its benefits are, and when other solutions are better.

In this research, we will compare data from papers on or relevant to this topic and look at how the results have changed over time. We will also look at the various methods of accelerating regular expressions on GPUs and compare their strengths and weaknesses. We will maintain an unbiased perspective and present information in a clear and concise manner. Technical terminology will be firstly explained in separate section.

3 Background

A regular expression, or regex for short, represents a set of exactly matching strings of characters and special symbols. This set can be infinite. The string of characters is then matched against the pattern to see if it matches. Regular expressions can be constructed in several ways [20]. The most common is to use a formal language, such as the one in POSIX standard[2]. Basic syntax is described as follows: characters of alphabet are matched literally, special symbols are used to match a single/multiple character matches, optional character matches, alternation, any character, line start/end and the empty string. As described in table 1.

Symbols	**Meaning**
.	Any character
*	Zero or more matches
+	One or more matches
?	Zero or one match
|	Alternation
-	Range
\	Escape character
^	Line start
$	Line end
:	Grouping
[ ]	Start and end of character class
( )	Start and end of group
{ }	Start and end of quantifier

Table 1: Regular expression special symbols, author's own work

3.1 Finite Automata

Regular expression matching is performed by using finite automata, a mathematical model of computation that abstracts computations into a finite number of states and transitions [24]. A finite automaton comprises a directed graph in which each node symbolises a state while each edge reflects a state transition. Two widely used representations of finite automata are deterministic finite automata (DFAs) and non-deterministic finite automata (NFAs). While NFAs and DFAs achieve the same outcome, there are some practical differences in terms of resource requirements and traversal behaviour [16]. Although deterministic finite automata (DFAs) have a simpler transition system, their execution is serial and the size of transitions in DFAs may be significantly larger than their equivalent NFAs [14, 15].

[2] `https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html`

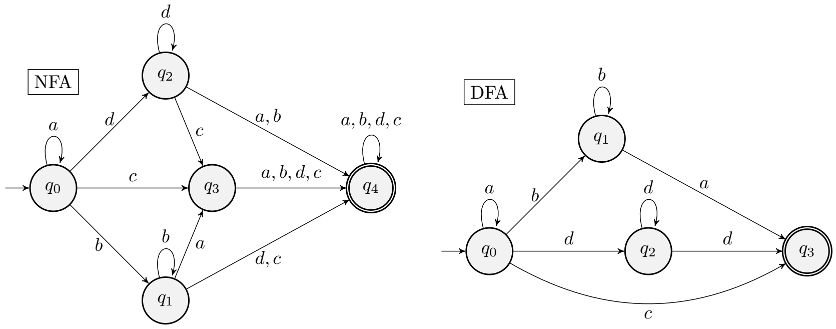

Figure 1: NFA and DFA for regular expression `a*(b+a|d*c)`, autor's own work

3.1.1 Automata processing

The regular expression's matching process is equivalent to a finite state machine traversal of the input stream. The process of matching begins with the activation of the initial states. Symbols from the input stream are sequentially utilized by the finite automaton.The process concludes once all the symbols of the input stream are processed.The incoming symbol is matched with the active states, and if it falls within the matchset of an active state, the active state transforms into a matched state [15].

We can guarantee worst-case performance by restricting the processing of each input character. Techniques to limit per-character processing involve enlarging the finite automaton. Therefore, the search space is determined by balancing the size of the automaton and the upper limit of per-character processing [16].

The benefit of using NFA is its capability to construct an NFA consisting of states that are less than or equal to the number of characters in the pattern-set. This makes the representation compact. The primary deficiency of NFAs lies in their traversal, during which the number of active states can vary from iteration to iteration, as well as the amount of work performed [24].

3.2 Graphics Processing Unit

The GPU, or Graphics Processing Unit, offers significantly higher instruction throughput and memory bandwidth than the CPU at a comparable price and power consumption. While the CPU is optimized for rapid execution of a sequence of operations referred to as a "thread" and can execute dozens of these threads concurrently, the GPU is optimized for thousands of parallel executions (offsetting the slower single-thread performance for increased throughput).This variation in capabilities is a result of differing design objectives for the GPU and CPU.[3]

All threads within a compute unit share a common instruction counter. Execution of a single compute unit occurs in lock-step, whereby each thread executes the same instruction when directed to do so. When control flow divergence occurs between threads within the same work group, divergent instructions are serialised,which can negatively impact performance. Similarly, an unbalanced

[3] From `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`

workload across threads in a work-group will result in idle threads, which reduces performance [23].

The memory hierarchy of GPUs comprises: Global memory, a larger and slower form of memory accessible by all threads in any compute units. Constant memory, a read-only section of the global memory with a specific cache for faster memory access. Local memory, connected to compute units and shared among threads in a single unit and private memory, exclusively reserved for individual threads [23].

3.2.1 Parallel computing platforms

Due to the significant performance potential of GPUs, their utilization has evolved from high-level shading languages to modern programming languages, reducing time and complexity involved in creating GPU-enabled applications [1]. Two of the most popular APIs for GPU programming are CUDA[4] (Compute Unified Device Architecture) and OpenCL[5] (Open Computing Language). CUDA is a proprietary API developed by NVIDIA, while OpenCL is s an open royalty-free standard developed by the Khronos Group (an open, member-driven consortium, publishing and maintaining open standarts) [6].

OpenCL provides an efficient and portable way to access the power of different computing platforms. However, when comparing OpenCL to CUDA, there are sometimes performance differences. These differences are often attributed to the portability of OpenCL, which can lead to performance degradation. However, in a fair comparison, there is no inherent reason for OpenCL to perform worse than CUDA. Performance differences are primarily due to programmers and compilers behaviour [7].

4 Related Work

Research into the use of GPUs for regular expression matching started with the publications "Fast Exact String Matching on the GPU" [17] and "A GPU-based Multiple-pattern Matching Algorithm for Network Intrusion Detection Systems" [10]. Their pioneering studies introduced a string matching program and multiple-pattern matching algorithm designed to take advantage of GPU processing capabilities. This work was soon followed by "Accelerating Regular Expression Matching Using Hierarchical Parallel Machines on GPU" [13] and "GPU-based NFA Implementation for Memory Efficient High Speed Regular Expression Matching" [26]. These two studies provide solutions to the performance challenges linked to regular expression matching in the context of network intrusion detection and other network functions.

Afterwards, a large, comprehensive study entitled "GPU Acceleration of Regular Expression Matching for Large Datasets: Exploring the Implementation Space" [24] was published. The study examines regular expression matching on GPUs, focusing on practical-sized and complex datasets. It explores the advantages and limitations of different automata representations and various GPU implementation techniques.

[4] https://developer.nvidia.com/cuda-toolkit

[5] https://www.khronos.org/opencl/

[6] https://www.khronos.org/

In the paper "Demystifying Automata Processing: GPUs, FPGAs or Micron's AP?" [16], the authors compare the performance of GPUs, FPGAs, and Micron's Automata Processor (AP) for regular expression matching.

Later, studies were published dealing with speeding up the processing of finite state machines. Papers "Scaling Out Speculative Execution of Finite-State Machines with Parallel Merge" [21] and "On-the-Fly Principled Speculation for FSM Parallelization" [25] introduce a speculative execution technique for finite state machines. "Why GPUs are Slow at Executing NFAs and How to Make them Faster" [14] proposed and evaluated optimisations to improve the throughput of NFA processing on GPUs by addresing tackle suboptimal data movement and underutilisation. "Asynchronous Automata Processing on GPUs" [15] have developed a lightweight approach to increase the parallelism of automata processing on GPUs by asynchronously searching for patterns in the input stream in parallel.

GPU acceleration of finite state machine input execution: Improving scale and performance. nieco o CUDA/OPENCL

5 Analysis

5.1 GPU Acceleration of Regular Expression Matching for Large Datasets: Exploring the Implementation Space

This study [24], published in 2013, was the first to make relevant measurements using real-life data. The measurements were conducted using NFA design by [4], along with their optimised NFA; and DFA design by [2], in uncompressed (U-DFA), compressed (C-DFA), and optimized (E-DFA) forms.

Their dataset used both real and synthetic pattern sets. The real patterns were taken from Snort's backdoor and spyware rules. The synthetic sets were generated using a tool described in [3] and tokens from the backdoor rules.

Measurements were made on system consisting of Xeon E5620 CPU and an NVIDIA GTX 480 GPU[7], using CentOS 5.5 and CUDA 4.

CUDA cores maybe

5.1.1 Findings

The repetition of wildcards and large character sets can lead to a state explosion when converting an NFA to a DFA. On the chart 2, we can see size of DFAs in MB. In contrast, NFAs are consistently less than 1 MB in size. For synthetic datasets, the uncompressed DFA representation has 40-50 times the memory requirement of its compressed counterpart. At the time of writing, the Dotstar.2 datasets exceed the device memory capacity when uncompressed-DFA representation is utilized.

As shown in chart 3, the U-DFA is the optimal solution for all pattern-sets, when applicable. However, this solution has high memory requirements and cannot be applied to complex pattern-sets due to a lack of graphics card memory. E-DFA offers a reasonable trade-off between U-DFA and C-DFA, resulting in a 3-5 times enhancement in speed compared to C-DFA at the expense of almost 1.5X more memory usage. Moreover, E-DFA performs better than NFA solutions on nearly all datasets.

[7] cuda cores

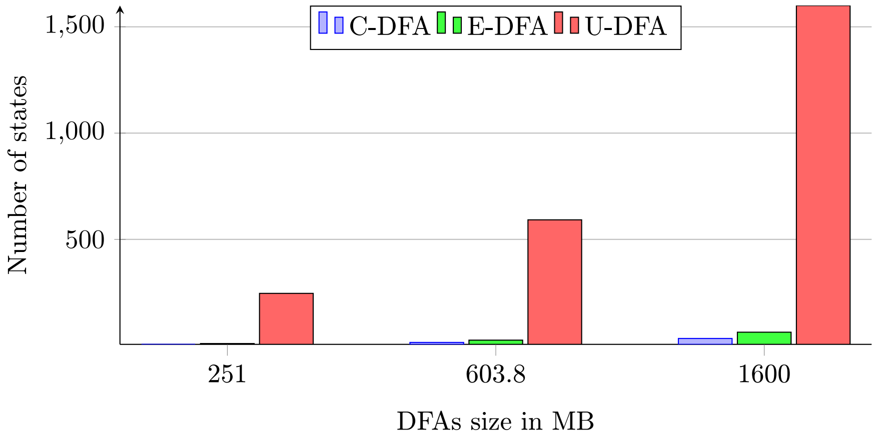

Figure 2: Scaling of DFAs size, author's own work

Figure 3: something about Speed NFA/DFA, author's own work

As shown in chart 4, both GPU implementations based on NFA and DFA algorithms outperformed their CPU equivalent.

5.2 Demystifying Automata Processing: GPUs, FPGAs or Micron's AP?

This 2017 study [16] investigated the acceleration of Finite State Automata on GPU, FPGA and Micron's AP. The automata model employed was NFA, and the algorithm used for its traversal engine was described in [19].

Their dataset consists of small NIDS, bioinformatics and synthetic pattern sets. The measurements were performed on a system that consisted of a dual 6-core Intel Xeon CPU and an Nvidia Titan X GPU[8] (Maxwell architecture), using CentOS 6.4 and CUDA 7. The FPGA Xilinx XC6VLX130T was chosen as it fell in the same price range as the GPU (approximately $1,200 at that time).

[8] cuda cores

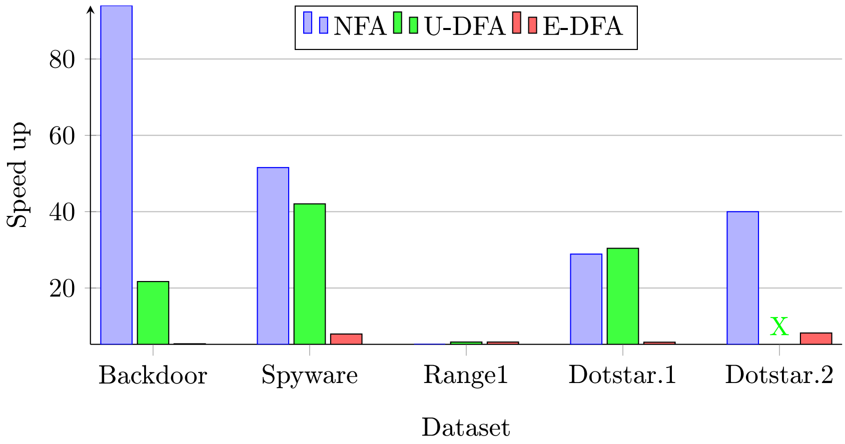

Figure 4: Speed up of GPU accelerated traversals over CPU, author's own work

5.2.1 Findings

The NFA format for GPU has a very small global memory utilisation, and even the largest dataset only occupies up to 133MB of global memory, which means we are able to fit any size dataset into a single GPU device, unlike FPGA and Micron's AP, which required more devices, as shown in the graph 5. However, the GPU records the lowest throughput data, with a 20 times lower throughput on smaller datasets up to 1000 times lower on the largest datasets.

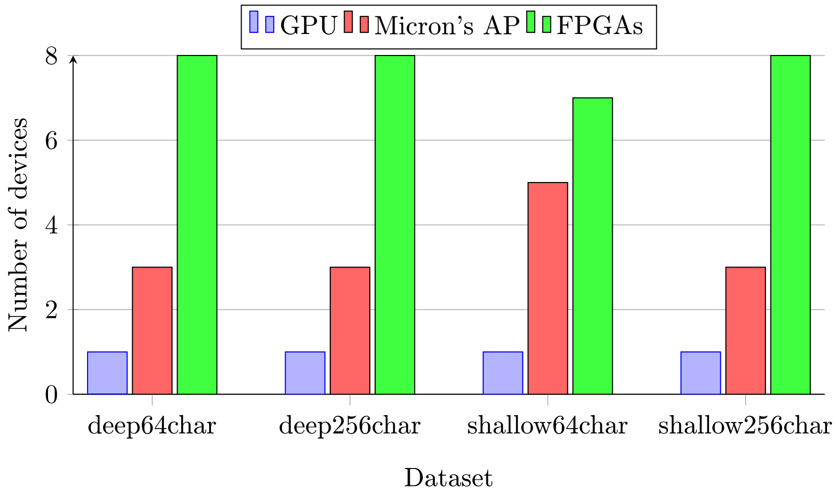

Figure 5: Number of devices, part of figure 5 from [16]

The optimization and partitioning steps for the NFA, which are common for all platforms, take between 3 and 249 seconds, varying with the size of the dataset. The GPU preprocessing task is primarily concerned with parsing the NFA partition files. As clearly demonstrated in Figure 6, GPUs provide considerably shorter preprocessing times.

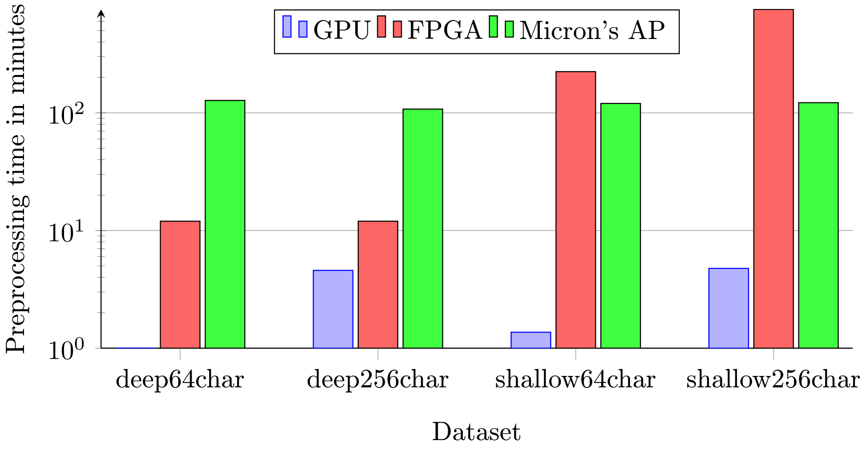

Figure 6: Preprocessing time, part of figure 5 from [16]

5.3 Papers on speeding up NFA processing on GPUs

6 Results and discussion

7 Conclusions

References

[1] A. Asaduzzaman, A. Trent, S. Osborne, C. Aldershof, and F. N. Sibai. Impact of cuda and opencl on parallel and distributed computing. In *2021 8th International Conference on Electrical and Electronics Engineering (ICEEE)*, pages 238–242, 2021.

[2] M. Becchi and P. Crowley. An improved algorithm to accelerate regular expression evaluation. In *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, pages 145–154, 2007.

[3] M. Becchi, M. Franklin, and P. Crowley. A workload for evaluating deep packet inspection architectures. In *2008 IEEE International Symposium on Workload Characterization*, pages 79–89. IEEE, 2008.

[4] N. Cascarano, P. Rolando, F. Risso, and R. Sisto. infant: Nfa pattern matching on gpgpu devices. *ACM SIGCOMM Computer Communication Review*, 40(5):20–26, 2010.

[5] C. Chapman and K. T. Stolee. Exploring regular expression usage and context in python. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, page 282–293, New York, NY, USA, 2016. Association for Computing Machinery.

[6] J. C. Davis, L. G. Michael IV, C. A. Coghlan, F. Servant, and D. Lee. Why aren't regular expressions a lingua franca? an empirical study on the

re-use and portability of regular expressions. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, page 443–454, New York, NY, USA, 2019. Association for Computing Machinery.

[7] J. Fang, A. L. Varbanescu, and H. Sips. A comprehensive performance comparison of cuda and opencl. In *2011 International Conference on Parallel Processing*, pages 216–225, 2011.

[8] A. Fuchs and D. Wentzlaff. The accelerator wall: Limits of chip specialization. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–14. IEEE, 2019.

[9] R. Gunawan, A. Rahmatulloh, I. Darmawan, and F. Firdaus. Comparison of web scraping techniques : Regular expression, html dom and xpath. In *Proceedings of the 2018 International Conference on Industrial Enterprise and System Engineering (IcoIESE 2018)*, pages 283–287. Atlantis Press, 2019/03.

[10] N.-F. Huang, H.-W. Hung, S.-H. Lai, Y.-M. Chu, and W.-Y. Tsai. A gpu-based multiple-pattern matching algorithm for network intrusion detection systems. In *22nd International Conference on Advanced Information Networking and Applications-Workshops (aina workshops 2008)*, pages 62–67. IEEE, 2008.

[11] Z. István, D. Sidler, and G. Alonso. Runtime parameterizable regular expression operators for databases. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 204–211, 2016.

[12] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey. Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu. *SIGARCH Comput. Archit. News*, 38(3):451–460, jun 2010.

[13] C.-H. Lin, C.-H. Liu, and S.-C. Chang. Accelerating regular expression matching using hierarchical parallel machines on gpu. In *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, pages 1–5, 2011.

[14] H. Liu, S. Pai, and A. Jog. Why gpus are slow at executing nfas and how to make them faster. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 251–265, New York, NY, USA, 2020. Association for Computing Machinery.

[15] H. Liu, S. Pai, and A. Jog. Asynchronous automata processing on gpus. *Proc. ACM Meas. Anal. Comput. Syst.*, 7(1), mar 2023.

[16] M. Nourian, X. Wang, X. Yu, W.-c. Feng, and M. Becchi. Demystifying automata processing: Gpus, fpgas or micron's ap? In *Proceedings of the International Conference on Supercomputing*, ICS '17, New York, NY, USA, 2017. Association for Computing Machinery.

[17] M. C. Schatz and C. Trapnell. Fast exact string matching on the gpu. *Center for Bioinformatics and Computational Biology*, 2007.

[18] Y. Sun, N. B. Agostini, S. Dong, and D. Kaeli. Summarizing cpu and gpu design trends with product data. *arXiv preprint arXiv:1911.11313*, 2019.

[19] J. Van Lunteren, C. Hagleitner, T. Heil, G. Biran, U. Shvadron, and K. Atasu. Designing a programmable wire-speed regular-expression matching accelerator. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 461–472. IEEE, 2012.

[20] X. Wang. *Techniques for efficient regular expression matching across hardware architectures.* University of Missouri-Columbia, 2014.

[21] Y. Xia, P. Jiang, and G. Agrawal. Scaling out speculative execution of finite-state machines with parallel merge. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '20, page 160–172, New York, NY, USA, 2020. Association for Computing Machinery.

[22] C. Xu, J. Su, and S. Chen. Exploring efficient grouping algorithms in regular expression matching. *PloS one*, 13(10):e0206068, 2018.

[23] V. Yaneva, A. Rajan, and C. Dubach. Gpu acceleration of finite state machine input execution: Improving scale and performance. *Software Testing, Verification and Reliability*, 32(1):e1796, 2022.

[24] X. Yu and M. Becchi. Gpu acceleration of regular expression matching for large datasets: Exploring the implementation space. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '13, New York, NY, USA, 2013. Association for Computing Machinery.

[25] Z. Zhao and X. Shen. On-the-fly principled speculation for fsm parallelization. *ACM SIGPLAN Notices*, 50(4):619–630, 2015.

[26] Y. Zu, M. Yang, Z. Xu, L. Wang, X. Tian, K. Peng, and Q. Dong. Gpu-based nfa implementation for memory efficient high speed regular expression matching. *SIGPLAN Not.*, 47(8):129–140, feb 2012.