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 [2]. They appear in approximately a third of open-source projects [3]. They are employed in technical fields, ranging from database querying [7], texts editors[1], and web scraping (the process of extracting data or information from

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

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

internet sites) [6] to network security, such as deep packet inspection [1], and bioinformatics [12], 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 [11, 17, 18].

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 [15]. 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 [8]. 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 [16] and inflexible to change [5, 10].

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 [10, 13]. 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) [4].

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 2. This is followed by a comparison of their representations of finite state automata forms in 2.1. We then move into parallel computing platforms, such as CUDA and OpenCL in 2.2.1.

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 3. Finally, we conclude with a summary of our findings in section 5.

...

2 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 [14]. 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,

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

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

pozriet ci uz neni dakde

2.1 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 [17]. 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 [11]. 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 [9, 10].

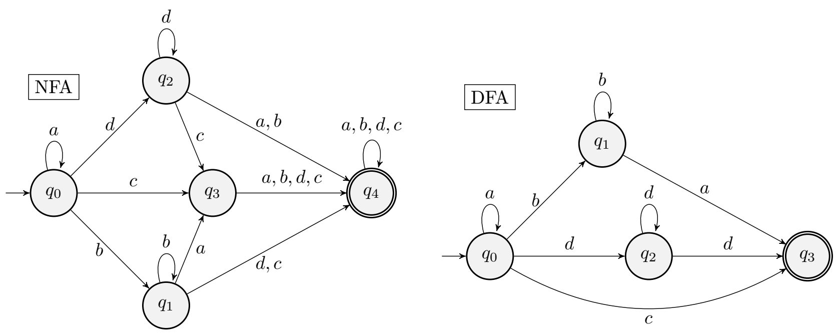

Figure 1: NFA and DFA for regular expression `a*(b+a|d*c)`, autor's own work

2.1.1 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 [10].

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 [11].

2.2 GPU ...

2.2.1 Parallel computing platforms

2.2.2 CUDA

2.2.3 OpenCL

3 Existing solutions

4 Analysis of ...

5 Conclusions

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] G. Prieto, A. Fullaondo, and J. A. Rodriguez. Prediction of nuclear export signals using weighted regular expressions (wregex). *Bioinformatics*, 30(9):1220–1227, 2014.

[13] 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.

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

[15] 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.

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

[17] 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.

[18] 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.