ISSN No:-2456-2165

Enhancement of Hakak's Split-Based Searching Algorithm through Multiprocessing

Enrico Sebastian Digman, Robbie Shane Orantoy, John Alfred Velasco,
Mark Christopher Blanco, Richard Regala, Dan Michael Cortez
Computer Science Department
Pamantasan ng Lungsod ng Maynila (University of the City of Manila)
Manila, NCR, Philippines

Abstract:- One of the recent string-matching algorithms classified as a Hybrid Boyer Moore Approach is Hakak's Split-Based Searching Algorithm which works by dividing the pattern into two parts while concentrating most of the searching process only to the second half of the pattern. However, the algorithm calls for optimization to its searching phase since it still employs the single shifting for instances where a mismatch is met. This paper presents an enhanced version of the algorithm by applying multiprocessing emphasizing its ability to make use of multiple CPU (Central Processing Unit) cores in finding occurrences of the pattern during the searching phase. Through this, concurrency and parallelism can be reached in the existing algorithm that is only limited to using a single processor prior enhancement. In conclusion, this study successfully enhanced the existing algorithm in terms of time complexity by maximizing the usage of memory resources.

Keywords:- *Hakak's Split-Based Searching Algorithm, String Matching, Multiprocessing.*

I. INTRODUCTION

String matching is defined as the process of searching whether a specific group of characters or 'pattern' exists in a group of strings or 'text'. With the continuous amount of information piling up on several databases, many problems may arise with regards to the resources to be spent when executing steps of searching, retrieving, and interpreting text information. Existing string-matching algorithms play a significant role to solve real-world problems concerning various fields of application such as text mining; this type of algorithm specializes on analyzing information which can be used by the following: Intrusion Detection Systems, Search Engines, Plagiarism Detection, Bioinformatics, and others. Out of all the widely known string matching algorithms available, many enhanced algorithms were already created with intensive and thorough research, one of these is the Hakak's Split-Based Searching Algorithm. Hakak's Split-Based Searching Algorithm aims to work on a unique approach of searching which aims to outpower similar algorithms in terms of the time consumption along the process of finding patterns on a text string.

Hakak's Split-Based Searching Algorithm's process starts with the *Preprocessing Phase* where it implements a unique approach of finding occurrences in the text by splitting the string patterns into two; only the second half of the pattern will be compared to the text. The end of the preprocessing phase will be followed by the *Searching Phase* where the second half of the pattern will be searched in the text string. It also uses the brute force approach when finding the index of the matching pattern. Once a match is verified, it will then compare the first half of the pattern to the part of the text string that is parallel before the first character of the pattern's second half. This algorithm presented positive results where it outperforms other string-matching algorithms in terms of time and space efficiency. However, its implementation of single shifting leads to a disadvantage especially on instances where patterns are to be searched on large-sized texts. To address the problem, an enhancement of the Hakak's Split-Based Searching Algorithm is proposed.

II. REVIEW OF RELATED LITERATURE

String matching algorithms usually comprise a preprocessing and searching phase highlighting that the latter makes use of a shift value keeping in mind the number of character comparisons while searching [1]. Their paper emphasizes that the searching phase may be improved by employing other bad character shift functions, such as the Berry-Ravindran algorithm which works by calculating the shift value placing emphasis on the bad character shift of the two consecutive characters. Other findings in regards with the searching phase of other algorithms include the Boyer-Moore algorithm having iterative processes of considerable length in searching for the pattern in the text [2]. Since Hakak's Split-Based Searching Algorithm is considered a Hybrid Boyer-Moore Approach, it still retains the concept of the bad character rule but utilizes the single shifting method. Considering it only makes use of a single block of character, the searching for the pattern may not be done efficiently. Since Hakak's Split-Based Searching Algorithm still employs the single shifting technique in order to search different texts, it is hypothesized that the idea of multiprocessing or parallelization of the algorithm will be helpful for the performance of the algorithm considering the time and space complexity.

A methodology is proposed in order to achieve multiple executions of the same task through the usage of multiprocessors. Multiprocessing is a kind of technology that utilizes two or more CPU cores within a system allowing the allocation and sharing of resources between each of them [3]. Nowadays, the relevance of multicore technology has shaped the industry's way of handling and processing data. Technological advancements suggest the exploration and optimization of current string matching algorithms' capability of processing texts. Since most exact string matching algorithms make use of a uniprocessor, intensive research and enhancements are needed in order to surpass the limitations encountered by these algorithms since they are not capable of a high degree of parallelizing of tasks.

The prominent challenge for the development of string matching algorithms requires high and efficient performance since applications such as network intrusion detection systems need to match with today's network speed. Shortcomings in identifying encountered attacks are highly recommended to be enhanced. Also, the relevance of big data in the industry requires efficient handling of big blocks of data [4]. Parallel processing is also highlighted as a technique that varies between multiprocessing and multithreading. Multiprocessing works by the separation of processes for execution of the same tasks in the program, whereas multithreading utilizes breaking down large tasks into a lot of tasks. An example of a study where it was implemented is when Knuth Morris Pratt, Karp-Rabin, and Boyer-Moore algorithms were integrated into a distributed multiprocessing environment introducing the concept of parallelization of exact string matching algorithms. These were applied in Snort's string matching engine utilizing Local Area Networks [5].

III. PROPOSED METHOD

In order to attain the objective of improving the existing algorithm's time complexity, this paper proposes the use of multiprocessing at the algorithm's searching phase.

A. Implementing multiprocessing in the searching phase of Hakak's Split-Based Searching Algorithm

Since the existing algorithm made use of the divide and conquer approach in the form of splitting, the researchers also took into consideration the same concept but through a different method which is multiprocessing. [4] also mentioned that multiprocessing works by separation of processes for execution of same tasks in the program. Since most CPU cores are not usually used by a computer, it is hypothesized that these can be used rather than still being idle. Time consumption is still the primary focus of a string-matching algorithm but in today's technology, algorithms must also be flexible and able to adapt to any computer's specification. The experiment shall be done in PyCharm Community Edition 2021.3.2 on AMD Ryzen 5 3500U Processor with 8 cores, 4 GB RAM (Random Access Memory) using Windows 10. To implement this in the experimental research design, the *multiprocessing* package of Python will be used to give the researchers the ability to maximize multiple processors in a single computer. The *Process* class is specifically used to initialize the processes for each of the CPU cores proven and then using *join ()* method so that all the processes are completed first before continuing to the next line of code.

B. Comparing the performance of Proposed Algorithm with Hakak's Split-Based Searching Algorithm

To verify the efficiency of the proposed algorithm, both algorithms (the existing Hakak's Split-Based Searching Algorithm and the proposed enhanced version) are to be compared side by side in terms of their time and memory consumption. The dataset that will be used for experimentation is Hakak, et al.'s S1 Dataset, specifically the bible.txt file. With regards to the total runtime per pattern search, both algorithms will be running ten (10) times using different pattern lengths such as short patterns (having a length less than 4), medium patterns (having a length of 4 to 7), and long patterns (having a length greater than 7); a python module: *datetime* will be used to measure the average runtime of the iterative processes performed in this comparison in milliseconds (ms). On the other hand, the quantification of both algorithms' memory consumption requires another Python module which is *memory-profiler*, using mebibytes (MiB) as unit of measurement. By performing these methods, it will now be possible to check and compare the efficiency of the existing and enhanced algorithms.

C. Identification of essential formulas to be used for the Proposed Algorithm and multiprocessing

$$length = n \,/\, P + m2 - 1 \tag{1}$$

$$start = (i * length) - ((m2 - 1) * i) \tag{2}$$

$$end = start + length \tag{3}$$

$$Position\ to\ Map = p2[i_0] - m2 \tag{4}$$

Multiprocessing, as proposed to enhance the existing algorithm, requires designating substrings of the text to each CPU core where searching is to be done simultaneously. To reach the substring length, it is important to take note of the number of processors to be distributed (P) and the length of the text (n). Considering the circumstances where patterns can potentially be found in between ends of two succeeding substrings, the formula includes adding the value of the length of the second half of the pattern (m2) subtracted by 1. This way, those areas will now be reached as the formula slightly extends the searching threshold.

Given that the substring length has already been computed, the starting index for searching at each processor must be found by making use of formula no. 2. The formula subtracts the product of the counter (i) and m2 subtracted by 1 to find the next substring which needs to start at an index where it overlaps with the end of the earlier substring. After executing the earlier formula, formula no. 3 figures out the ending index of the substring assigned to each processor in searching by adding the values computed from the first two formulas.

Following the execution of earlier formulas, the searching phase will now begin to search for pattern matches in a concurrent manner. If the second half of the pattern (p2) is found, formula no. 4 must be used to know the current location of p2 in the substring. Subtracting the current index by m2 will find the index of where the first half of the pattern must be mapped in the substring.

IV. RESULT AND DISCUSSIONS

A. Comparative Analysis of Initialization of Different No. of CPU Cores

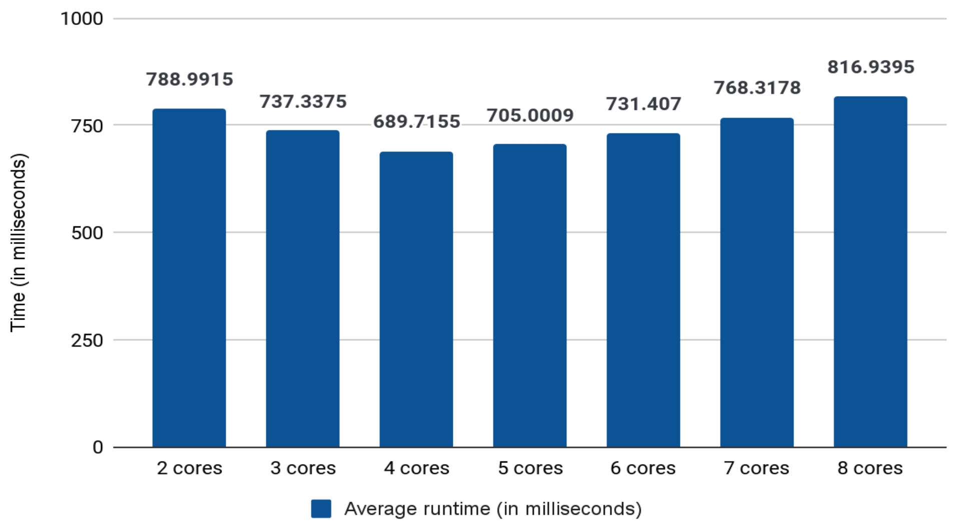

Fig.1: Average running time of the proposed algorithm using different number of CPU cores

Fig. 1 shows how the proposed algorithm's average runtime is directly affected by the number of CPU cores being distributed for running. Among all the CPU cores used in testing, 4 cores garnered the shortest average runtime at 689.7155ms (about half second). In addition, slower average runtime is attributed to using insufficient and excessive numbers of CPU cores. Not using enough cores in the process causes the algorithm to run slow since the partitioning of subtext will be longer, taking a considerable time in shifting the pattern. Moreover, the usage of too many cores will be like running the algorithm sequentially since there is a time overhead when spawning a new child process. Running the algorithm at 4 CPU cores, which is found to be the best processor for this algorithm shows that it is 14.394% and 18.446%faster than using a 2-core processor 788.9914999999999ms (about 1 second) and an 8-core processor 816.9395000000001ms (about 1 second), respectively.

B. Average running time Results

Pattern Length	Text	Hakak's Split Based Searching Algorithm Average Running Time (ms)	Proposed Algorithm Average Running Time (ms)	No.of Occurrences
Less than 4 characters	God	1395.9604999999997	914.8524000000001	4040
	for	1341.6218000000003	1073.334	12619
4 to 7 characters	wroth	1308.4778000000001	692.2360000000001	47
	finish	1386.2564	691.5103	54
Greater than 7 characters	that Adam	1245.5148	782.0433	1
	continually	1386.0278	834.1506999999999	79

Table 1: Comparison of Hakak's Split-Based Searching Algorithm and Proposed Algorithm

As shown in Table 1, the Proposed Algorithm was able to surpass Hakak's Split Based Searching Algorithm in terms of the average running time based on 10 rounds. Specifically, patterns with length of 4 to 7 characters showed a larger gap in terms of milliseconds compared to other pattern lengths which means that the Proposed Algorithm is at its' best performance when searching for the said pattern length. The researchers also seen that the no. of occurrences of a pattern greatly affects the searching performance.

C. Memory Consumption Results

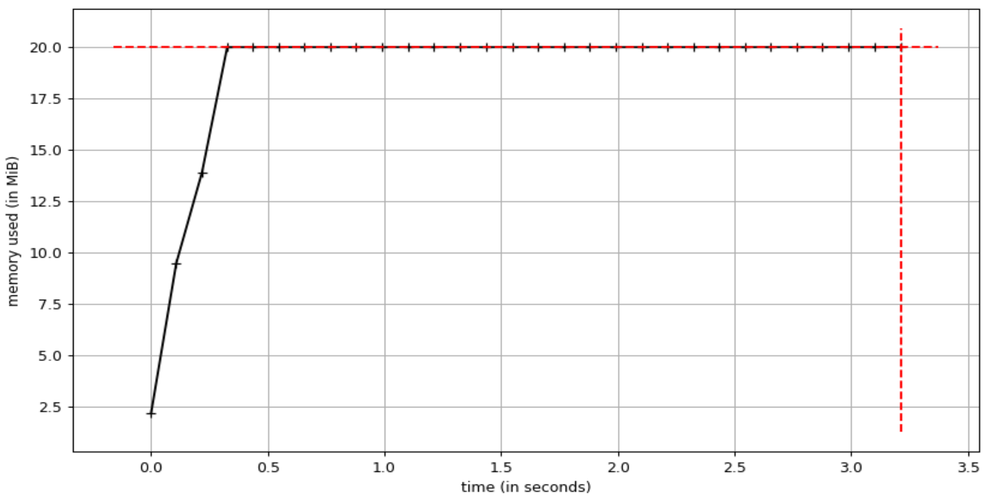

Fig. 2: Analysis of Hakak's Split-Based Searching Algorithm's Memory Consumption

Fig. 2 shows the existing algorithm's memory consumption throughout the entire process. The algorithm started from 0 MiB and at once allotted 20 MiB for the whole duration of the process. Also, there was no scenario of any other memory being used in the latter part of the string-matching algorithm. This result happened due to the splitting of the pattern and allotting most of the entire process only to the second half. Furthermore, the existing algorithm's motivation includes the elimination of creating a shifting table in the preprocessing phase thus only making use of a single shifting method when a mismatch is met in the searching phase. Compared to traditional string-matching algorithms, Hakak's Split-Based Searching algorithm considered using minimal memory while concurrently addressing the concern of time consumed. The existing algorithm may still be enhanced while taking into consideration that the time consumed is less but at the same time maximizing the resources the computer has.

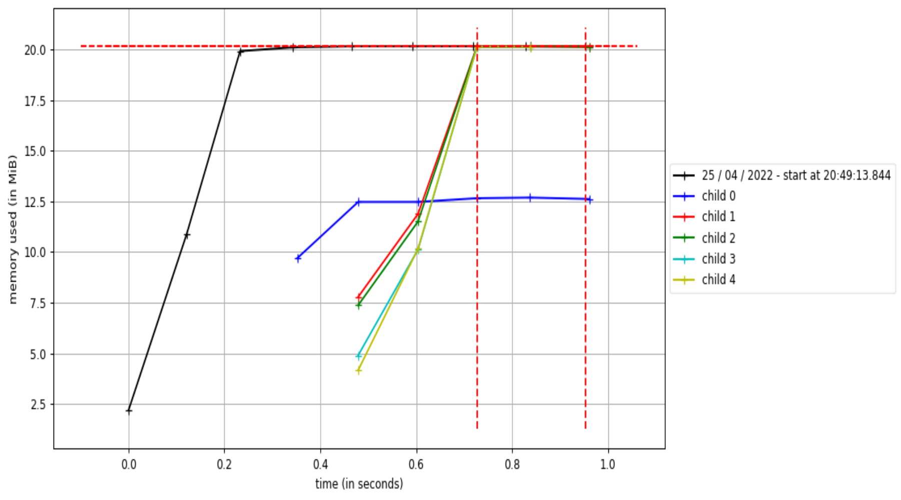

Fig.3: Analysis of Proposed Algorithm's Memory Consumption

Fig. 3 displays the memory consumption of the proposed algorithm emphasizing the use of CPU cores or child processes. The black line portrays the data consumed by the main CPU which is 20 MiB. Also, no added memory is used in the main CPU because of its child processes. Before starting the multiprocessing, a specific CPU core, portrayed as the blue line, was initialized known as the *Manager* who handles the shared memory allocation for the list of results. After a time, 4 child processes were spawned taking into consideration the use of 4 CPU cores allotted for multiprocessing. These child processes begin their task of them are properly initialized because of the *join ()* method. Each child processor is assigned an index from the subtexts proven which serves as their starting point of searching. They work concurrently but if any child process finds a match even if the other processes are not finished, it appends the results to the Manager at once. The *Manager's* memory increased by a little amount because of the results that were combined from the child processes.

V. CONCLUSION AND RECOMMENDATION

Through the researchers' extensive analysis of gathered data, they were able to draw these conclusions. First, the single shifting text window is inefficient and may compromise the time consumed in searching the pattern. Since the technology of multiprocessing was introduced in this study, avoiding the usage of too many or too little amount of CPU cores allows string matching algorithms to provide a high degree of parallelizing of processes. Considering its elements, a manager can be shared across all processes and is more dynamic compared to shared memory, however, it is slower since it is spawning a new child process. Also, big datasets are suitable for shared memory resources in order not to compromise the time for passing the arguments to the CPU cores.

After an in-depth evaluation of the study’s results and conclusion, the researchers recommend implementing other forms of parallelization as enhancements for string matching algorithms such as multithreading, distributed computing, etc. Also, the use of the GPU may enable advanced parallelization of multiple data setsand is capable of efficient computations. Architectures such as CUDA, MPI (Message Passing Interface), and OpenMP allow programmers to manipulate specific functions for parallelization. When it comes to the proposed algorithm, it may produce efficient results when searching texts such as DNA sequences since the pattern is more likely to be found in the text. Additionally, the proposed algorithm may perform differently depending on the user's computer specification, specifically the number of CPU cores available. Experimentation of the study may consider the programming language to be used when coding different string matching algorithms for comparative analysis since some have specific functions that may help in coding.

ACKNOWLEDGMENT

The researchers would like to express their utmost gratitude to God Almighty for granting them the proper skill, intellect, and enough strength to make this endeavor come to reality. A plenty of thanks to their parents and kind friends for their undying support and continuous encouragement to keep going in pursuit of completing this research. Finally, the researchers were beyond delighted to acknowledge their research adviser and other professors of the Pamantasan ng Lungsod ng Maynila for sharing their ability as well as supplying motivational words and constructive criticisms which makes a lot of help throughout the course of carrying out this piece of work.

REFERENCES

[1.] Hudaib, A., Al-Khalid, R., Suleiman, D., Itriq, M., & Al-Anani, A. (2008). A fast pattern matching algorithm with two sliding windows (TSW). Journal of Computer Science, 4(5), 393.

[2.] Xu, B., Zhou, X., & Li, J. (2006, June). Recursive shift indexing: a fast multi-pattern string matching Algorithm. In *Proc. of the 4th International Conference on Applied Cryptography and Network Security (ACNS)* (pp. 64-73). Springer-Verlag Singapore

[3.] Soewito, B., & Weng, N. (2007, October). Methodology for evaluating dna pattern searching algorithms on multiprocessor. In *2007 IEEE 7th International Symposium on BioInformatics and BioEngineering* (pp. 570-577). IEEE.

[4.] Hnaif, A. A., Aldahoud, A., Alia, M. A., Al'otoum, I. S., & Nazzal, D. (2018). Multiprocessing scalable string matching algorithm for network intrusion detection system. *International Journal of High Performance Systems Architecture*, 8(3), 159-168.

[5.] Al-Mamory, S. O. (2012). Speed enhancement of snort network intrusion detection system. *Journal of Babylon University, Pure and Applied Science*, *20*(1), 1