

Parallel processing of E-Atheer algorithm using pthread paradigm

Atheer Akram AbdulRazzaq¹, Mohammed A. Fadhel², Laith Alzubaidi^{3,4}, Omran Al-Shamma⁴

¹Businesses Informatics College, University of Information Technology and Communications, Baghdad, Iraq

²College of Computer Science and Information Technology, University of Sumer, ThiQar, Iraq

³School of Computer Science, Queensland University of Technology, Brisbane, Australia

⁴University of Information Technology and Communications, Baghdad, Iraq

Article Info

Article history:

Received Aug 18, 2021

Revised Jan 20, 2023

Accepted Jan 26, 2023

Keywords:

Database types

E-Atheer algorithm

POSIX threads (Pthreads)

Speedup

String matching algorithm

ABSTRACT

The development in the field of computer technology, and the increase in the growth rate of database, alongside the extraction of certain data from a huge pool of database involve intricate and complex processes. The processes comprise text mining, pattern recognition, retrieval of information and text processing. Thus, the need for enhancing the performance of string matching algorithms is required, which is considered as one of the challenges to the researchers. Consequently, one of the resolution to address this problem is the parallelization for exact string matching algorithms. In this study, we implemented the parallel exact string matching algorithm termed as E-Atheer with multi-core processing utilizing Pthread (POSIX) for the reduction of time consumption. The Pitch, XML, Protein, and DNA database types are utilized to test the impact of the proposed parallel algorithm. The parallelization algorithm obtained positive results in the parallel execution time, and a more superior expediting capabilities, in comparison to the sequential result. The Pitch database indicated optimal results in parallel execution time, and when utilizing long and short pattern lengths. The DNA database indicated optimal speedup performance when utilizing short and long pattern length, meanwhile the XML and Protein on the other hand indicated the worst results.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Atheer Akram AbdulRazzaq

Businesses Informatics College, University of Information Technology and Communications

Baghdad, Iraq

Email: athproof@uoitc.edu.iq

1. INTRODUCTION

String matching algorithms are the algorithms utilized for scertaining the optimal alignment through the comparison of a set of patterns to the random string. Whilst engaging in the comparison stage, it is imperative that the pattern length be equal to the text window length. Additionally, the comparison between the pattern and the text window strings is dependent upon the identification of the match between them [1]. Typically, String Matching is usually utilized in numerous computer applications such as signal and image processing, artificial intelligence (AI), web search engines, intrusion detection systems, operating systems, speech and pattern recognition [2], [3], information retrieval [4], and computational biology and chemistry. Furthermore, in recent years, the string matching algorithms are considered as the main component utilized in the application of DNA pattern matching, and the analysis of Protein sequences [5], [6]. The development and growth rate of the database is escalating at a swift rate; thus the need for enhancing the performance of exact string matching algorithms.

Certain exact known string matching algorithms such as Brute force, Boyer-Moore, Karp-Rabin and Knuth-Morris-Pratt (KMP), are prevalently and extensively utilized. It should be noted that Brute force

algorithm can be considered as the easiest form of algorithm. In this algorithm, the comparison operation between text and pattern window is executed from right to left. When the match or mismatch is obtained, the shifting process is executed to the right character only [2], [7]. The Boyer-Moore algorithm is prevalently and extensively utilized due to its high performance and efficiency. The comparison in this algorithm is initiated from the right to left. There is a possibility of a match or mismatch between the pattern and the text window, where it is noted that the shifting is dependent upon good suffix and the bad character functions [8]. As aforementioned, the Karp-Rabin algorithm is the algorithm that is utilized in the hash process, and the comparison of this algorithm is conducted from the left to right. It is dependent upon the calculation of the hash function of the pattern and the text window. The hash technique possesses a high performance and efficiency capability which reduces the consumption of time due to the use of integer numbers [9], [10]. The KMP algorithm is considered as the first liner algorithm in time. The comparison of this algorithm is executed from left to right, and the shifting process is dependent upon the last character [2].

The exact string matching algorithms are affected by three factors, which comprise the number of attempts, the number of character comparison, and the consumed time, where the efficient algorithms reduced the drawbacks of one or more of these factors. A majority of string matching algorithms enable the reduction on the number of attempts, and the number of character comparison in sequential performance, which however possess weakness in terms of the consumed time. Therefore, the researchers find that it is important to concentrate on the consumed time problems, through the utilization of the parallel processing to address this problem, as it affords the reduction of consumed time in exact string matching algorithms [11].

The parallel processing is defined as the practical way to reduce the computational time, which is dependent upon dealing with the cores or processors in computers to resolve the sequential computing problems [5], [11], [12]. Prevalently, in tandem with the increase in the enhanced development of computer technology, particularly in the architectures of the processors and multicores, there exist pertinent necessity for enhancing the performance of the exact string matching algorithms. So as to coincide with the enhancement of the system, and the reduction of the consumed time of the multicore and multiprocessors of computers.

The computing of parallel process possesses great potentials of improving the data execution time, in comparison to sequential computing that consumes a longer duration of time to obtain the results [11]. There are numerous exact string matching algorithms that utilized parallel computing by utilizing either the multicore or multi-processor techniques, for example AKRAM algorithm. This algorithm employed the parallel multiprocessors model which comprises the message passing interface (MPI). The AKRAM algorithm that was adopted in the parallelization process utilized the technique of data decomposition, which segregated the data into numerous subparts and were distributed to the processors inside the cluster. The MPI multiprocessor model indicated a high performance computing ability in contrast to the performance of sequential computing in the AKRAM algorithm [13]. Additionally, there are exact string matching algorithms that used multiprocessor parallel technology, and among them is the Karp-Rabin algorithm, which divides the string into subsets and each individual ones are compared with the pattern separately. This algorithm obtained good results when large datasets were utilized. However it demonstrated inefficient results when utilized with short patterns. Additionally, there are alternative algorithms other than the exact string matching algorithm which utilized the multicore technology such as the quick search algorithm. The quick search algorithm used the OpenMP paradigm that reduced the execution time of the algorithm. The OpenMP technology operates through the utilization of the data decomposition technique, which divided the data into subsets through the fork and join process. The OpenMP technique demonstrated good performance in parallel time in comparison with sequential time [11].

The KMP algorithm employed the hybrid technology OpenMP/MPI, where this algorithm indicated high parallelization results with large string size. However, due to the communication time, this algorithm obtained low results when it utilized more than two clusters. Conversely it obtained very good results when it utilized two clusters only [14]. Moreover, in order to enhance the filtering of intrusion detection system (IDS), the Quick search algorithm utilized the multicore techniques which entail the Pthread (POSIX) and OpenMP paradigms. It employed these two multicore implementations so as to expedite speed (speedup) of the parallelized algorithm time for a swifter IDS [15]. The graphics processing units (GPU) technology was also utilized by the string matching algorithms such as Karp-Rabin algorithm. It demonstrated that when utilized, it indicated a difference in terms of cores, threads numbers, pattern, and string sizes with high speed of up to 23x in the parallel time, with the implementation of GPU implementation in comparison with the CPU implementation [16].

In this paper, we have redesigned the exact string matching algorithm termed as E-Atheer by utilizing the parallel Model with the aim of reducing the execution time and expediting the speeding (speedup) of the algorithm. Here, we evaluated the performance of the algorithm over different factors such as using different types of databases, pattern length, threads number, in addition to the number of cores. In section 2 describes the Algorithm and implementation that explain the technique of E-Atheer algorithm and Pthreads (POSIX) technique, the generation of Pthreads code for the E-Atheer algorithm, and the implementation and environment. The results are obtained in section 3, the discussions and analysis are presented in section 4, and conclusion is introduced in section 5.

2. ALGORITHM AND IMPLEMENTATION

2.1. E-Atheer algorithm

It involves two stages, which include the preprocessing stage and the searching stage. The preprocessing stage is reliant on the selected functions of two algorithms (Atheer and Berry-Ravindran) comprising the hashing function; Berry-Ravindran bad character (Brbc) function and Boyer-Moore bad character (Bmbc) function. Meanwhile, the searching stage in the algorithm is dependent upon the comparison of the hash values of three characters between the pattern and the text window. When a match is obtained, the comparison of the three characters between pattern and text window will then take place. In the event of a match, a comparison of the hash value of remaining characters from the second characters to $(m/2-1)$ will be continued. Additionally, on the ensuing match obtained, it will then be followed by a comparison of the characters. In instances of further matching, it will then be followed by a comparison of the hash of the characters from $(m/2+1)$ to the last character-1. Finally, if there is still a match obtained, this is followed by a comparison of the characters between pattern and text window. If there is a match or mismatch obtained for each step, the new shifting of the algorithm will be based on the highest values between the m from bmBc table and $(m+1$ and $m+2)$ values from brBc table [17].

2.2. POSIX threads

The thread can be utilized as a separate flowing for the space of address in order to control it [18]. The threads are utilized in the parallel execution of the shared memory multiprocessor and multi-core architectures. The POSIX Pthreads is considered as a low level interface of programming for the operations with the OS threads [19]. The POSIX Pthreads indicated to the C language threads programs for UNIX by the IEEE the POSIX 1003.1c standard is utilized to generate different threads in the caller operation. In addition, one more use of the parallelization in the UNIX is the fork, which can generate a new operation that eventually can enable a new operation of the caller. The results of the experiments demonstrated that the Pthread is able to obtain additional enhanced results as compared with the fork. This is because the thread can be created with less operation system overhead in comparison to the fork, as the fork operation generates separate operation of execution [18].

2.3. Generating of pthreads code for E-ATHEER hybrid algorithm

This section describes the main technical contribution that is generated by the parallel C program, which utilized the libraries of Pthreads. The following steps entail the parallel operations of the Pthread paradigm that is employed in this study:

- a) The first step in the parallel program of this study considered the control of fine grained texture by the management of the thread, which directly operates on the threads, and are capable of creating new thread functions by defining new THREAD (name), and then started as StartThread (name); function. The iMaxThreads function defines the maximum number of threads, and the threads numbered from 0 to P-1, where the P is the number of possible threads.

Moreover, the initial step also utilized the Mutex functions which require the initialization before usage. It entails a predefined value that can be assigned for the static initializer: Pthread_Mutex_Initializer. The Pthread_mutex_lock () routine is utilized by the thread in order to obtain the lock on the Mutex variable. Additionally, this stage of initial step also utilize the Pthread_mutex_unlock () routine, which is needed after the threading process is finished. The data is utilized if there are other threads needed to obtain the Mutex for their data usage. In addition to that, there are other functions that are used in the management of threads, which are WaitForThread (int iThread). This type of function waits until the thread numbered as iThread finishes its execution. The function WaitForAll () waits for all the threads.

- b) In the second step in the parallel program, the variables are shared over the threads in all steps of the shared section. The arguments that are used in the algorithm are length of text n , text y , length of pattern m , and pattern x . This step is also utilized in the function of bmBc and function of brBc that is used for the shifting in the E-Atheer algorithm.
- c) The third step in the parallel program is dependent upon the decomposition of the data, where the array $y[]$ that is related to the text is divided into small chunks p . In addition the p chunks are treated using the singular threads in the parallel region. Each division cannot be precisely n/p , because of the searching technique of the pattern matching algorithm. Thus, the division process can be $n/p+m-1$.
- d) In the fourth step, after the division of the data text, each thread in the core takes one chunk, and the threads take the same pattern to each thread. It separately used the pattern with the specific chunk, and this is dependent upon the algorithm technique. In the E-Atheer algorithm, each thread utilizes the hash function for a number of three times, with pattern in three phases, alongside the first phase of the text window and searching phase. When each thread process is finished, the number of character comparisons, the calculation of the number of attempts, a comparison is made between the chunk and the pattern, and the consumed time.

- e) In the fifth step, the reduction mutex is completed by calculating all the results from the threads. The function pthread_mutex_lock (Reduction) is utilized to copy the summarized results for each thread separately, and to calculate the values when each thread ends. In addition, through the utilization of the reduction function, the parallel algorithm is able to compute the number of character comparison, the number of attempts and consumed time for each thread are computed, which is ensued by the calculation of the final results for all the threads. Additionally, the fifth step utilized the pthread_mutex_unlock (Reduction); where its function is executed after the threads completed all the functions, and to restart the creation of new threads, as illustrated in Figure 1.

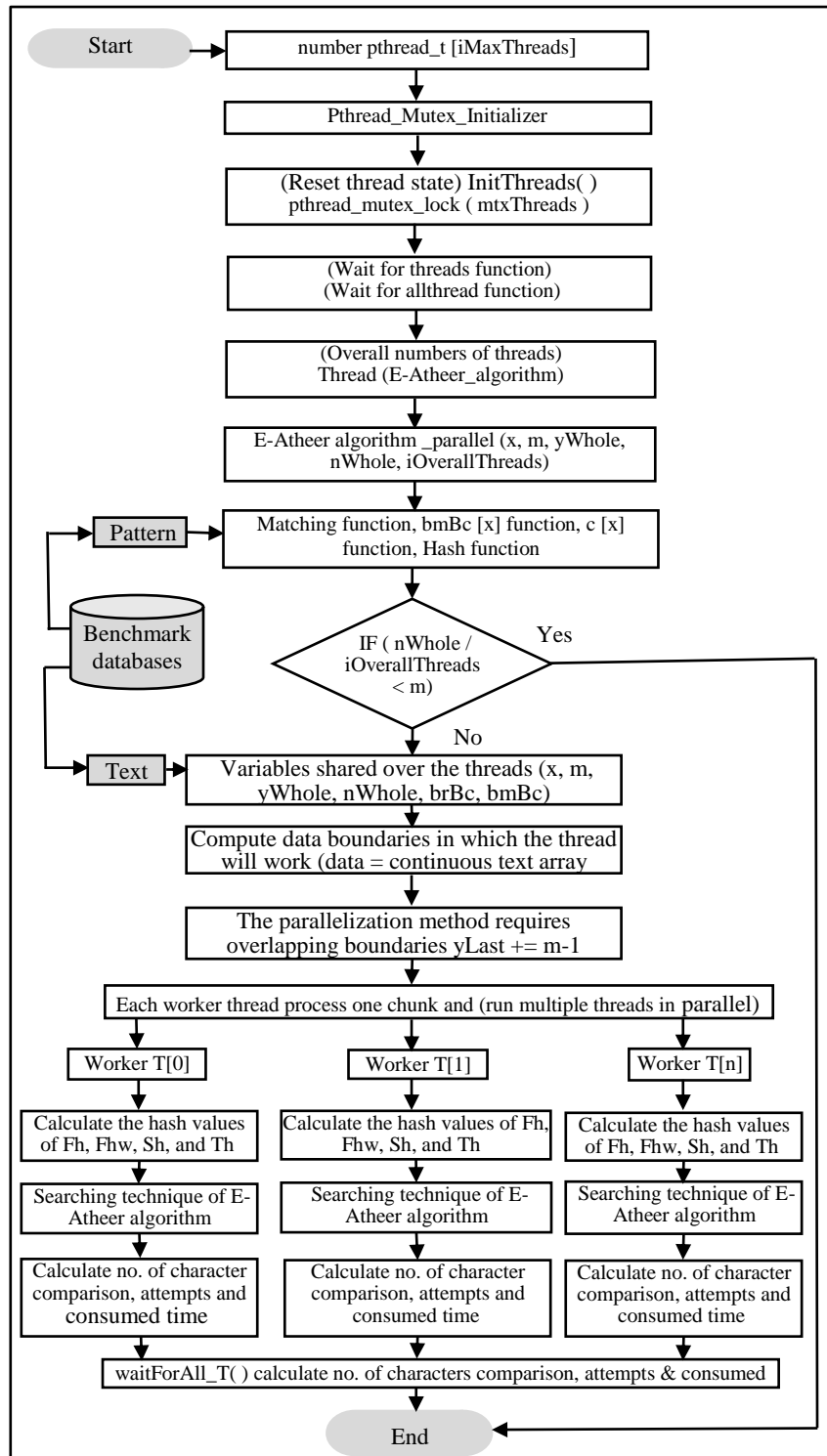


Figure 1. Flowchart for parallel of the E-Atheer algorithm using Pthread paradigm

2.4. Implementation and environment

2.4.1. Hardware and Khawarizmi cluster architecture

The experiment utilizes the Khawarizmi cluster that is available in the parallel lab of the School of Computer Science, in the Universiti Sains Malaysia (USM) (khawarizmi.cs.usm.my). This cluster possesses a single master node (2×Quad-Core Intel Xeon E5450 3.00 GHz, 2×12 MB Cache, 1333 MHz FSB) and two slave nodes (2×Quad-Core Intel Xeon 1.6 GHz, 2×4 MB Cache, 1066 MHz FSB). The cluster possesses three nodes, where inside each node there are two processors that possess four cores, with a single thread in each core. The operating system of this cluster is Linux (rocks cluster distribution 6.1, centos 6.3, 64-bit) and the compiler that is utilized in the cluster is GCC 4.4.6.

2.4.2. Performance metrics

The parallelization of the suggested algorithm is executed through the utilization of the Pthread. The evaluation of the algorithm results is conducted through the use of metrics, which is used to make a comparison between the sequential and parallel algorithms performance. These specific metrics involve the speedup, and the execution time [20]-[22].

A. Execution time

The execution time between the starting (initiation) point and the ending (termination) point of a single processor, in addition to the entire operations time, is called the sequential time. The parallel time entails the consumed time from the moment of the beginning moment of the first processor until the moment of the finished time of the last processor. The consumed time in sequential is denoted as T_s and in the parallel is denoted as T_p .

B. Speedup

It is utilized to gain the benefit of the parallel process. The speedup is reliant on the ratio of the elapsed time in the sequential stage, to the elapsed time in the parallel stage. The T_s is the consumed time of the sequential phase, T_p is the consumed time of the parallel phase, and S is the speedup. The calculated time is in milliseconds and the measurement is dependent upon following equation.

$$\text{Speedup } (S) = T_s / T_p$$

2.4.3. Experiment design

The databases that are utilized in this experiment is downloaded from the Pizza and Chili Corpus Web site (<http://pizzachili.dcc.uchile.cl/>) (Pizza Chili Corpus). The datasets that are utilized in this study are; DNA, Protein, XML, and Pitch with 200 MB data size. The average was calculated for the program after the implementation of each dataset for five times. This experiment is dependent upon a machine which is utilized in numerous prior studies. It utilized eight cores due to the fact that the cluster possesses three nodes, and inside each node there are 8 cores with a single thread for each individual one. Conversely, when more than 8 cores is utilized in this study, the results will be useless and produce a backfire. In this experiment, the figures employed the number of cores which are dependent upon the core power of two, between 2^1 to 2^3 . The time used in the results is (Seq) for sequential time, and (C2), (C4), and (C8) to represent the two, four, and eight numbers of cores respectively. The sequential results were compared with the cores' parallel results in this study. To employ the comparison operation of databases and algorithms, in addition to make the attainment of the parallel results easier in this study, the average results were utilized. Two pattern lengths were utilized in this experiment: the length of short pattern, which extended from 4 characters to 28 characters, and the length of long pattern (length power of 2), which extended from 2^5 characters to 2^{10} characters. Furthermore, the pattern lengths taken diverse colours when evaluated of the parallel times and the speedup.

3. RESULTS

In the parallel operation of E-Atheer algorithm, the dataset decomposed into several segments and were distributed to the cores by utilizing Pthreads (POSIX). The E-Atheer algorithm is evaluated by comparing the speedup, as well as, the parallel time and sequential time, when utilizing long and short pattern lengths for data of size 200 MB. The databases that are used within the experiment are diverse within the alphabet size, where this sort is utilized to analyze the behaviors of the algorithm within the different sizes of alphabet.

3.1. Parallel and sequential times

When comparing the parallel and sequential times, and when utilizing long and short pattern lengths with 200MB size of the database, the parallel time indicated the optimal performance rather than the sequential time. The Pitch databases indicated the optimal time in the most of the long and short pattern lengths. Meanwhile, the DNA database indicated the worst time achievement in the entire long and short pattern lengths, as indicated in Figures 2 and 3 respectively.

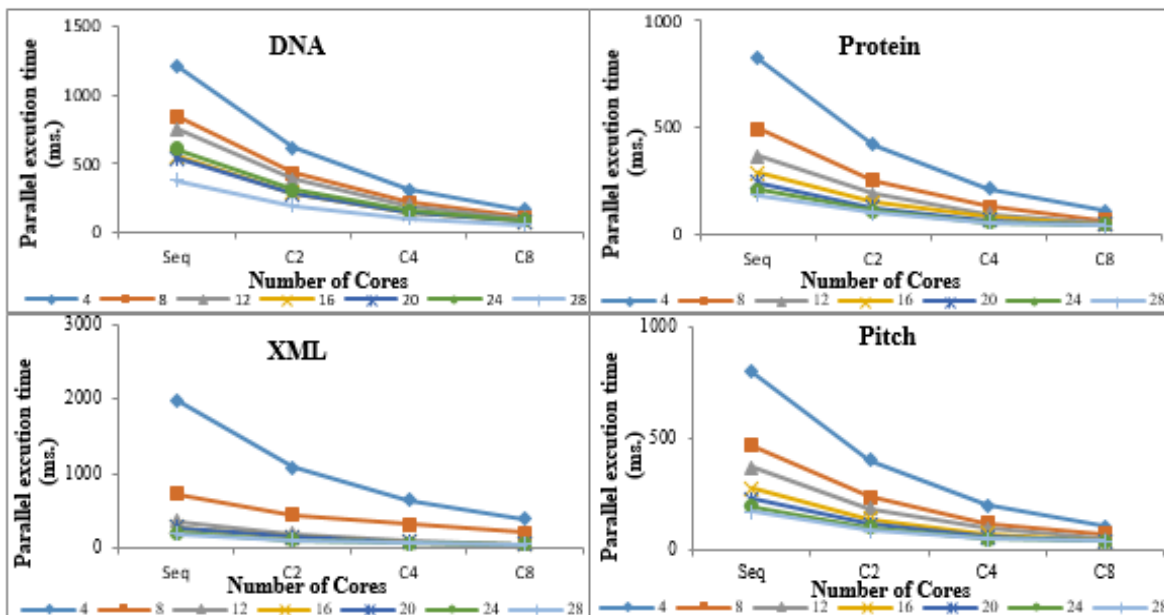


Figure 2. Evaluations of the parallel time utilizing short pattern lengths

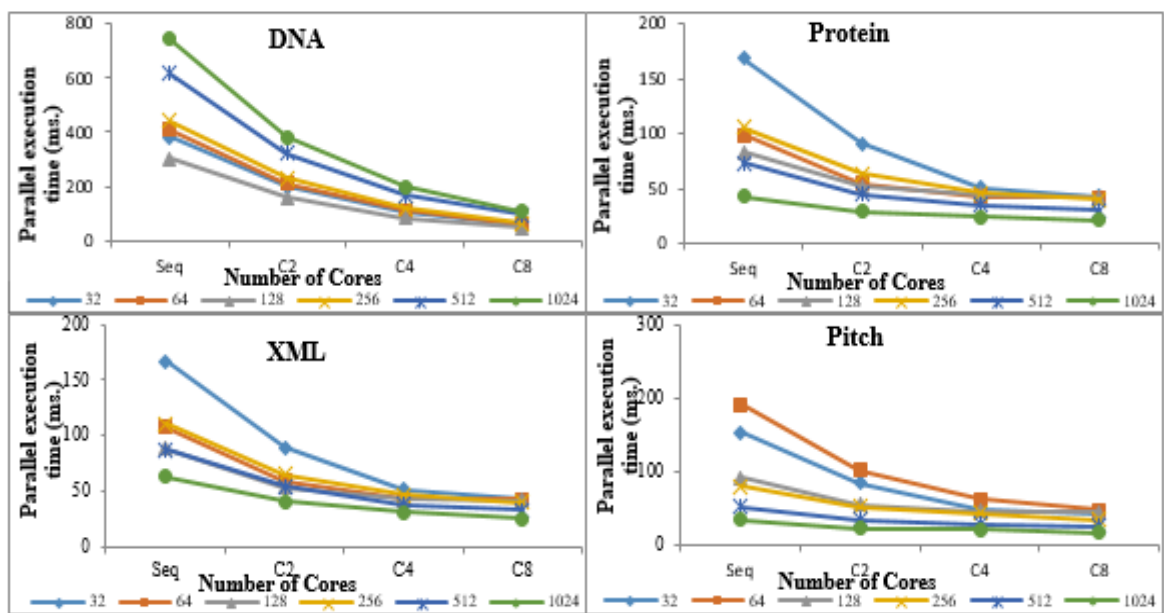


Figure 3. Evaluations of the parallel time utilizing long pattern lengths

3.2. Speedup

It indicated high results when using short pattern lengths. The results of algorithm are high with the utilization of DNA database in all the long pattern lengths. Meanwhile the other databases obtained good speedup results only when using two (2) cores. The speedup obtained good results when using 32 pattern lengths with 4 and 8 cores, whereas it is reduced when utilizing other long pattern lengths. The optimal database in the entire long and short pattern lengths is the DNA database, and contrarily the worst database is the XML in most of the short pattern lengths. Moreover, the Protien database being the worst in most of the long patterns, as indicated in Figures 4 and 5.

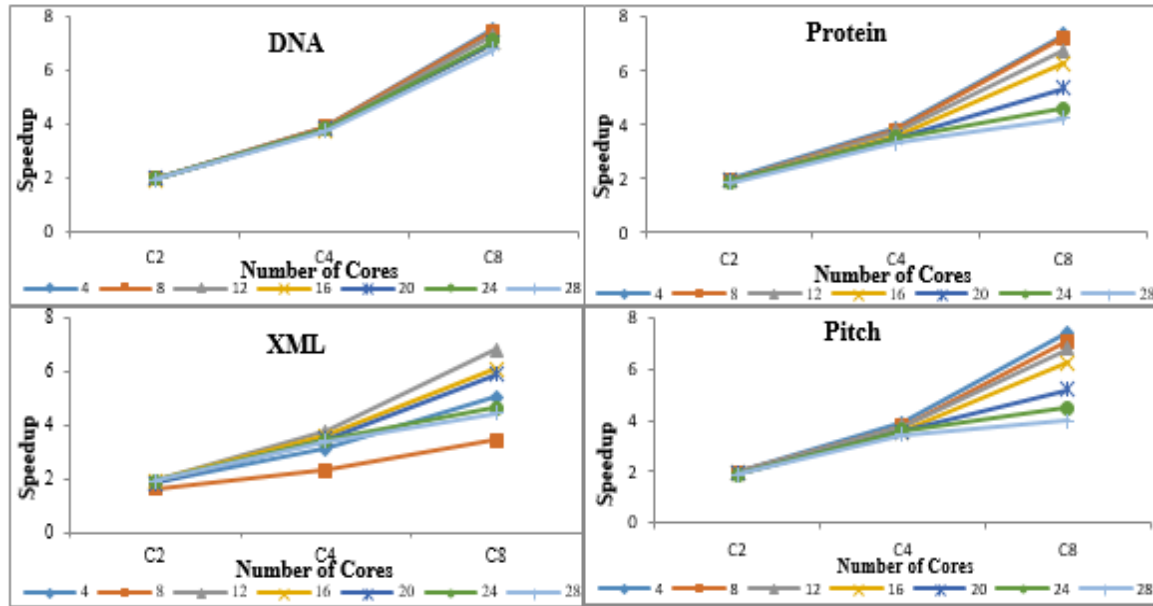


Figure 4. Evaluations of the speedup utilizing short pattern length

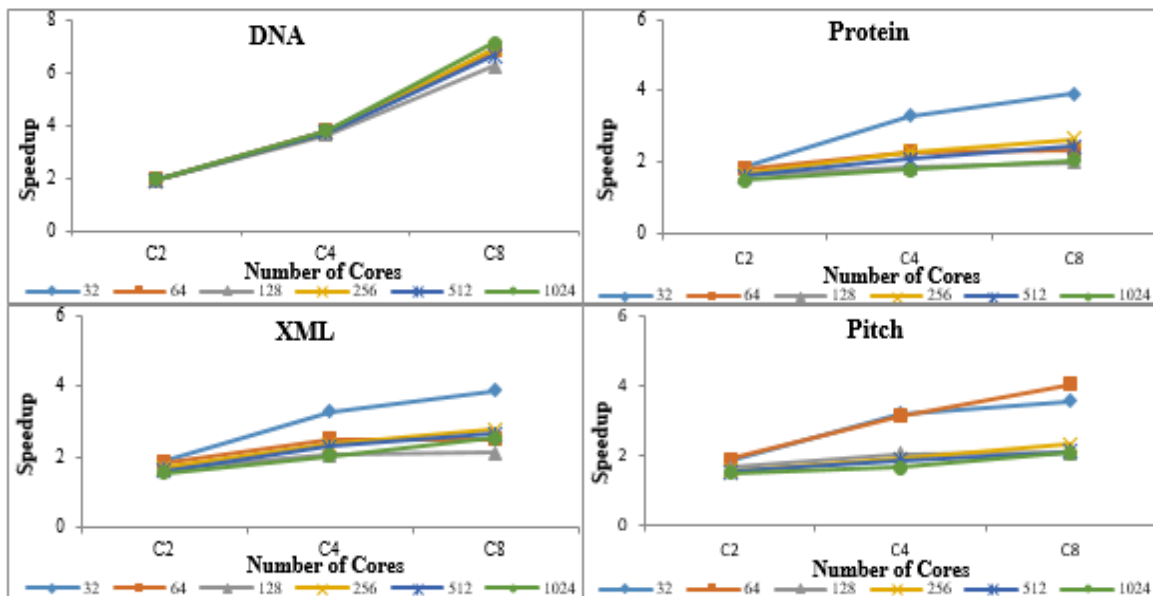


Figure 5. Evaluations of the speedup utilizing long pattern lengths

4. DISCUSSIONS AND ANALYSIS

Initially, the findings of the parallel execution time indicated the optimal performance in comparison to the sequential time. However, when the number of cores increased, the overhead is revealed to increase and to have an impact on the parallel execution time. This is due to the increase in the communication time. When the number of cores in parallel and pattern length increase, the parallel and sequential times are decrease in the long and short pattern lengths [13], [23].

The best sequential results registered are 359 and 95 ms for short and long patterns with 200 MB data size, separately. The worst sequential results registered are 699 and 483 ms for short and long patterns with 200 MB data size, separately. The particular best parallel results when utilizing short pattern length are as follows: 2 cores 185 ms, 4 cores 96 ms, and 8 cores 59 ms. The best parallel results when utilizing long pattern length are as follows: 2 cores 56 ms, 4 cores 41 ms, 8 cores 35 ms.

The particular worst parallel results when utilizing short pattern length are as follows: 2 cores 358 ms, 4 cores 189 ms, and 8 cores 120 ms. The worst parallel results when utilizing long pattern length are as follows: 2 cores 251 ms, 4 cores 130 ms, and 8 cores 71 ms, as shown in Table 1. The Pitch dataset indicated the optimal sequential and parallel times in long and short pattern lengths due to the fact that the E-Atheer algorithm utilized the hash and (bmBc) efficient functions techniques. The DNA dataset obtained the worst results in sequential and parallel time owing to the minute DNA alphabet size, which is dependent on the hash function that needs to check the repeated characters. Thus, additional time is consumed than in other datasets.

Table 1. Performance assessment for the average sequential and parallel execution times (ms) of the E-Atheer algorithm

Database types	Short pattern				Long pattern			
	Seq	C2	C4	C8	Seq	C2	C4	C8
DNA	699	358	182	96	483	251	130	71
Protein	372	192	100	59	95	56	41	36
XML	573	316	189	120	104	60	42	38
Pitch	359	185	96	59	100	58	41	35

The speedup increased when utilizing the short pattern length, while the speedup is expedited in terms of time with long pattern length when only 2 cores are utilized, during the use of all types of dataset except for the DNA database. The best results of speedup when short patterns utilized with 200 MB data sizes are separately displayed as follows: 2 cores 1.95, 4 cores 3.82, and 8 cores 7.19. The best results of speedup when long patterns utilized are separately displayed as follows: 2 cores 1.93, 4 cores 3.72, and 8 cores 6.76. The worst results of speedup when short patterns utilized with 200 MB data sizes are separately displayed as follows: 2 cores 1.85, 4 cores, 3.28, and 8 cores 5.16. The worst results of speedup obtained when using long patterns are separately displayed as follows: 2 cores 1.66, 4 cores 2.27, and 8 cores 2.56, as shown in Table 2.

Table 2. Performance assesment for average speedup of parallel E-Atheer algorithm

Database types	Short pattern			Long pattern		
	C2	C4	C8	C2	C4	C8
DNA	1.95	3.82	7.19	1.93	3.72	6.76
Protein	1.93	3.63	5.98	1.67	2.27	2.56
XML	1.85	3.28	5.16	1.71	2.42	2.74
Pitch	1.93	3.66	5.9	1.66	2.29	2.69

The DNA dataset obtained good performance with all number of cores because there are only four characters in the DNA dataset which possesses short parallel time in comparison to the sequential time for the same data type. Moreover, the parallel time decreased when the pattern length increased [23]. Furthermore, the algorithm technique is dependent upon the hash function that is used to calculate the hash value of three characters only in the first step. Therefore when more than 2 cores are utilized with large alphabet size and the long pattern, the elapsed time will be reduced and the shifting will be small unlike in the DNA dataset.

The optimal speedup results are obtained through the use of the DNA database when utilizing short and long pattern lengths, as can be observed in the results, due to the DNA dataset obtaining high performance with parallel time in comparison to the results obtained using sequential time. The speedup indicated an increase when the sequential execution time is greater than the parallel time. The XML dataset has achieved the defective results when utilizing the short pattern length, meanwhile the Protien database obtained extremely terrible results when utilizing the long pattern length. The speedup decreased when the alphabet size increased as the speedup is affected by the database type [24], [25].

5. CONCLUSION





The results in this study are represented by the parallel execution time, and speedup of the sequential and parallel of E-Atheer algorithm through the utilization of varying types of datasets with size 200MB, and with short and long lengths of patterns. The parallelization of E-Atheer algorithm obtained high performance results in comparison to sequential version when utilizing Pthread paradigm as a multi-core processing technology. Through our significant research, it is noted that the E-Atheer algorithm obtained optimal results and high performance in the parallelization by the reduction in the algorithm execution time, and indicated high results in speedup. In the parallel execution of E-Atheer algorithm, the Pitch database indicated optimal results in parallel execution time, and when utilizing long and short pattern lengths. Meanwhile the DNA

database obtained optimal results in speedup. For future work the E-Atheer algorithm may be progressed by executing to other multi core environment (e.g., GPU program) and multiprocessors models (e.g., MPI), as well as Executing hybrid parallel models, like the hybrid models of the OpenMP-MPI or GPU-MPI.





REFERENCES

- [1] S. Deusdado and P. Carvalho, "GRASPM: an efficient algorithm for exact pattern-matching in genomic sequences," *International Journal of Bioinformatics Research and Applications*, vol. 5, no. 4, pp. 385-401, 2009, doi: 10.1504/IJBRA.2009.02751.
- [2] S. I. Hakak, A. Kamsin, P. Shivakumara, G. A. Gilkar, W. Z. Khan, and M. Imran, "Exact string matching algorithms: survey, issues, and future research directions," *New Trends in Brain Signal Processing and Analysis IEEE Access*, vol. 7, pp. 69614-69637, April 2019, doi: 10.1109/ACCESS.2019.2914071.
- [3] A. A. AbdulRazzaq, N. A. Rashid, and A. M. Taha, "The enhanced hybrid algorithm for the AbdulRazzaq and Berry-Ravindran algorithms," *International Journal of Engineering and Technology*, vol. 7, no. 3, pp. 1709-1717, 2018, doi: 10.14419/ijet.v7i3.12436.
- [4] A. W. Mahmood, N. A. Rashid, and A. A. A. Rozaq, "BM-KMP hybrid algorithm for exact and subsequence string matching," *Proceeding of the 3rd International Conference on Informatics and Technology, (Informatics '09)*, 2009, pp. 81-87.
- [5] S. S. Al-Dabbagh, N. H. Barnouti, M. A. Naser, and Z. G. Ali, "Parallel quick search algorithm for the exact string matching problem using openMP," *Journal of Computer and Communications*, vol. 4, no.13, pp. 1-11, 2016, doi: 10.4236/jcc.2016.413001.
- [6] K. F. Xylogiannopoulos, "Exhaustive exact string matching: the analysis of the full human genome," *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, Aug 2019, doi: 10.1145/3341161.3343517.
- [7] L. S. Nunes, J. L. Bordim, Y. Ito, and K. Nakano, "Parallel rabin-karp algorithm implementation on GPU (preliminary version)," *Bulletin of Networking, Computing, Systems, and Software*, vol. 7, no. 1, pp. 28-32, 2018.
- [8] N. B. Nsira, T. Lecroq, and M. Elloumi, "A fast boyer-moore type pattern matching algorithm for highly similar sequence," *International Journal of Data Mining and Bioinformatics*, vol. 13, no. 3, pp. 266-88, 2015, doi: 10.1504/ijdm.2015.072101.
- [9] R. E. Putri and A. Siahaan, "Examination of document similarity using rabin-karp algorithm," *International Journal Of Recent Trends In Engineering and Research*, vol. 03, no. 08, pp. 196-201, 2017, doi: 10.23883/IJRTER.2017.3404.4SNDK.
- [10] A. B. Khoir, H. Qodim, B. Busro, and A. R. Atmadja, "Implementation of rabin-karp algorithm to determine the similarity of synoptic gospels," *1st International Conference on Advance and Scientific Innovation (ICASI)*, pp.1-7, 2019, doi: 10.1088/1742-6596/1175/1/012120.
- [11] A. A. AbdulRazzaq, "New hybrid searching techniques and multi-core implementations for exact string matching," Ph.D thesis, Computer Science School, University Science Malaysia, 2014.
- [12] X. Y. Zha and S. Sahni, "GPU-to-GPU and host-to-host multipattern string matching on a GPU," *IEEE Transactions on Computers*, vol. 62, pp. 1156-1169, 2013, doi: 10.1109/TC.2012.61.
- [13] A. A. Abdulrazzaq, N. A. Rashid, and A. H. A. Alezzi, "Parallel processing of hybrid exact string matching algorithm," *In 2013 IEEE International Conference on Control System, Computing and Engineering*, 2013, pp. 203-209, doi: 10.1109/ICCSCE.2013.6719959.
- [14] I. M. Abu-Zaid and E. K. El-Rayyes, "Parallel search using KMP algorithm in arabic string," *International Journal of Science and Technology*, vol. 2, no. 7, pp. 427-431, 2012.
- [15] A. A. Hnaif, M. Alhalaiah, O. Abouabdalla, S. Ramadass, and M. M. Kadhum, "Parallel quick search algorithm to speed packet payload filtering in NIDS," *Journal of Engineering Science and Technology*, vol. 4, no. 2, pp. 220-230, 2009.
- [16] P. Shah and O. Rachana, "Improved parallel rabin-karp algorithm using compute unified device architecture," *In International Conference on Information and Communication Technology for Intelligent Systems (Springer)*, pp. 236-244, 2017, doi: 10.1007/978-3-319-63645-0_26.
- [17] A. A. AbdulRazzaq, N. A. Rashid, A. A. Abbood, and Z. Zainol, "The improved hybrid algorithm for the atheer and berry-ravindran algorithms," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 8, no. 6, pp. 4321-4333, 2018, doi: 10.11591/ijece.v8i6.pp4321-4333.
- [18] D. Herath, C. Lakmali, and R. Ragel, "Accelerating string matching for bio-computing applications on multi-core CPUs," *In 2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*, 2012, pp. 1-6, doi: 10.1109/ICIInfS.2012.6304784.
- [19] S. S. Arslan, H. Le, J. Landman, and T. Goker, "OpenMP and POSIX threads Implementation of Jerasure 2.0," *In 2017 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, 2017, pp. 1-5, doi: 10.1109/BlackSeaCom.2017.8277690.
- [20] A. Grama, G. Karypis, V. Kumar, and A. Gupta, "Introduction to Parallel Computing," Second Edition, Addison Wesley, pp. 165-169, 2003.
- [21] Z. A. A. Alqadi, M. Aqel, and I. M. E. Emary, "Performance analysis and evaluation of parallel matrix multiplication algorithms," *World Applied Sciences Journal*, vol. 5, no. 2, pp. 211-214, 2008.
- [22] H. A. Kadhim, "New sequential and gpu-based hybrid string matching algorithms," Master thesis, Computer Science School, University Science Malaysia, 2012.
- [23] K. Hamidouche, A. Borghi, P. Esterie, J. Falcou, and S. Peyronnet, "Three high performance architectures in the parallel APMC boat," *In 2010 Ninth International Workshop on Parallel and Distributed Methods in Verification, and Second International Workshop on High Performance Computational Systems Biology*, pp. 20-27, 2010, doi: 10.1109/PDMC-HiBi.2010.12.
- [24] T. K. Pyrgiotis, C. S. Kouzinopoulos, and K. G. Margaritis, "Parallel implementation of the wu-manber algorithm using the opencl framework," *In IFIP International Conference on Artificial Intelligence Applications and Innovations (Springer)*, pp. 576-583, 2012, doi: 10.1007/978-3-642-33412-2_59.
- [25] A. A. AbdulRazzaq, Q. S. Hamad, and A. M. Taha, "Parallel implementation of maximum-shift algorithm using OpenMp," *Indonesian Journal of Electrical Engineering and Computer Science (IJECS)*, vol. 22, no. 3, pp. 1529-1539, 2021, doi: 10.11591/ijeecs.v22.i3.pp1529-1539.





BIOGRAPHIES OF AUTHORS

Dr. Atheer Akram AbdulRazzaq     was born in Baghdad, Iraq. He got his bachelor from Al Mustansiriya University, Iraq in 2006. He got his M.Sc. from Universiti Sains Malaysia in 2009. He got his Ph.D. in High performance computing (Parallel tools and applications) in 2014, School of Computer Sciences, Universiti Sains Malaysia. He is currently an assistant professor at the Businesses Informatics College, University of Information Technology and Communication, Baghdad, Iraq. His main research area is high performance computing. His research interests are in exact string matching algorithms, parallel and distributed processing, network security, and data mining. He has published numerous papers in string matching, parallel and distributed processing, network security, and genomic information processing. He can be contacted at email: athproof@uoitc.edu.iq.







Mohammed A. Fadhel     received the Master degree from University of technology/Control and System Eng./Computer engineering, Iraq in 2015. He is currently a senior lecturer in the College of Computer Science and Information Technology, University of Sumer, Iraq. Published more than 40 Research Papers in various National and International Conferences, International Journals (Scopus/SCI/SCIE/SSCI Indexed). His research interests include embedded system, image and acoustic processing, deep learning, medical diagnosis techniques, GPU, FPGA, real-time systems, sound processing, hologram technology and many more. He can be contacted at email: Mohammed.a.fadhel@uoitc.edu.iq.



Laith Alzubaidi     is currently a Ph.D. student at Queensland University of Technology/Faculty of Science and Engineering, Brisbane, Australia. He received his Master's degree in computer science from the University of Missouri/USA in 2016. In his Master's thesis, he worked on the detection and classification of breast cancer with deep learning. Laith broad research area falls in artificial intelligence and internet of things (IoT). In particular, his research interests include deep learning for medical applications and IoT. He can be contacted at email: laith.alzubaidi@hdr.qut.edu.au.



Omran Al-Shamma     currently works at the Postgraduate affairs, University of Information Technology and Communications, Baghdad, Iraq. He holds Ph.D. from University of Hertfordshire, UK 2013. Omran interests in computer design software, medical diagnosis techniques, and preliminary aircraft design software, real time surveillance system. He can be contacted at email: o.alshamma@uoitc.edu.iq.