PARALLEL PROCESSING OUTCOMES OF E-ABDULRAZZAQ ALGORITHM USING MULTI-CORE TECHNIQUE

Atheer Akram AbdulRazzaq[1]
[1]*Businesses Informatics College, University of Information Technology and Communications, Baghdad Iraq*
athproof@uoitc.edu.iq

Nur'Aini Abdul Rashid[2]
Department of Parallel and Distributed Processing, School of ComputerSciences Universiti Sains Malaysia, 11800 Pulau Pinang, Malaysia.
nurainipng@gmail.com, nuraini@cs.usm.my

***Abstract* - The string matching problem is considered one of the substantial problems in the fields of computer science like speech and pattern recognition, signal and image processing, and artificial intelligence (AI). The increase in the speedup of performance is considered an important factor in meeting the growth rate of databases, Subsequently, one of the determinations to address this issue is the parallelization for exact string matching algorithms. In this study, the E-Abdulrazzaq string matching algorithm is chosen to be executed with the multi-core environment utilizing the OpenMP paradigm which can be utilized to decrease the execution time and increase the speedup of the algorithm. The parallelization algorithm got positive results within the parallel execution time, and excellent speeding-up capabilities, in comparison to the successive result. The Protein database showed optimal results in parallel execution time, and when utilizing short and long pattern lengths. The DNA database showed optimal speedup execution when utilizing short and long pattern lengths, while no specific database obtained the worst results.**

Index Terms - Database types, E-AbdulRazzaq algorithm, OpenMP directive, Parallel execution time, Speedup.

I. INTRODUCTION

String matching is the method of identifying all alignment occurrences by comparing two finite-length strings[1], [2], [3]. String matching is among the foremost imperative issues connected in numerous computer science applications, such as speech and pattern recognition, artificial intelligence (AI), signal and image processing, intrusion detection systems, operating systems [4], web search engines [5], [6] information retrieval[7], [8] and search analysis [9]. String matching is additionally utilized to analyze protein sequences and for DNA pattern matching. The tireless challenges of string matching incorporate the duplication of databases every two years, and this influences the speed of computers, which leads to an increase in the measure of memory. Thus, the utilization of effective algorithms of string-matching is crucial to solving these issues [10].

The brute force algorithm is considered the simplest algorithm technique compared to other string-matching algorithms which scan the text pattern with text substring from left to right. when the match is complete or a mismatch occurs the shifts precisely one position to the right. Thus, the brute-force algorithm runs in O(mn) time [11]. The KMP algorithm is efficient and considered the first liner algorithm in the time. The comparison of the algorithm depends on the execution from left to right and the shifting depends on the last character[12]. The Boyer-Moore algorithm is predominantly and broadly utilized due to its high proficiency. The comparison in this algorithm is started from the right to left. There's a plausibility of a match or mismatch between the pattern and text window, where it is the shifting depends on the good suffix and the bad character functions[13]. The Karp-Rabin algorithm is the algorithm used with the hash function, and the comparison depends on execution from left to right. This algorithm depends on the computation of the hash function of the pattern and text window. It is subordinate to the calculation of the hash work of the design and the content window. The hash strategy has a high execution which diminishes the utilization of time due to the utilization of integer numbers [14], [15].In addition, The two-sliding windows algorithm (TSW), depends on the design of sliding windows to check the content at the same time from the left and right sides. This algorithm utilizes two windows, one from the left side, and the second from the right side. The comparison technique between the pattern and text occurs from both sides at the same time, when there's a mismatch, the shifting will at that point happen for both of the two sides, from the left side a move will be executed to the left, and the right will move to the right side. The shift operation depends on the Berry- Ravindran shift technique, which takes two characters continuously [16], [17].

Parallel processing is a feasible way of reducing computing time. Parallelism depends on collaboration among the processors or cores in computers to settle sequential computer problems. The parallel applications depend on two strategies: data decomposition and function decomposition. The parallel computer architecture is classified into two: shared memory and distributed memory. These techniques are classified according to how they access memory. The processors in the first technique (shared memory technique) use one access way to reach the common memory. Shared memory is used to save data as well as other purposes such as synchronization and communications among the processors [18].

Open Multi-Processing (OpenMP) is considered as a standard application programming interface (API) in the utilization of shared memory applications. OpenMP uses C/C++ implementations and has many beneficial attributes than other parallel programing interfaces, such as having less complications when using parallel implementations; more popularly used than other parallel APIs in PCs like The Message Passing Interface (MPI) and POSIX; uses data design and decompositions automatically[10], [19], [20].

The parallelization of the shared memory is used in various applications of computers such as the classification of pictures [21], Restoration of the image [22], direct polynomial math [23], information mining [24], bioinformatics [25], Intrusion detection system (IDS) [26], [27], [28].

This paper concentrates on the issues that are related to the execution of the E- AbdulRazzaq algorithm, with the aim of growing the speedup by reducing the successive time and exploring the appropriateness of the parallelization of the E-AbdulRazzaq algorithm on the multi-core technique by utilizing the OpenMP model. Here, we assessed the execution of the algorithm depending on various factors such as the length of the pattern, the number of cores, and using different database types. Section 2 clarifies the string matching algorithms using parallel processing, Section 3 describes the implementation and technique of the E-AbdulRazzaq algorithm and OpenMP technique, the results and analysis are obtained in Section 4, and the conclusion in Section 5.

II. STRING MATCHING ALGORITHMS USING PARALLEL PROCESSING

String matching algorithms are used to obtain optimal results when dealing with different data sets, and solving problems that are related to computer applications. Nowadays, many challenges on string matching are emerging, and the performance of sequential algorithms are insufficient in accomplishing the big procedures when dealing with huge datasets and the long elapsed time of these algorithms, therefore, parallel processing used in solving these problems, especially those related to large databases [29]. There are a number of exact string matching algorithms used in application parallel interfaces in solving the many problems related to these algorithms. Some algorithms, such as the Quick-Search algorithm, use multi-core technology; the Quick-Search algorithm uses two parallelization paradigms, the OpenMP and the Pthread, to enhance the filtering of the IDS by increasing the speed to make the IDS faster, and obtain the efficient algorithm in this field [24]. Other algorithms, like brute force, Boyer-Moore (BM), and KMP, also use multi-core technology. The technique of these algorithms depends on the division of the text into subtexts, and all of these subtexts are used separately by the different parallelized algorithms (KMP, BM, and brute force). Computation includes the time and the number of matching after finishing each subtext for each algorithm separately, calculating the final result of each algorithm, and then comparing the results with that of other algorithms. The BM algorithm showed the best result than the rest, and the KMP algorithm was not better than the brute force algorithm in terms of the rate of results [30].

The KMP-BM algorithm is hybrid algorithm that uses multi-core technology; this algorithm depends on dividing the large text into smaller blocks, and separately using the same pattern with all the blocks simultaneously. The last $m-1$ characters in the pattern are saved in all the blocks to avoid any problem in the character comparison between the blocks and the pattern; this procedure was used because the blocks deal with different processors [31]. There are algorithms used in the GPU technique, like the Naive, Quick- Search, KMP and Horspool algorithms. All of these algorithms use the GPU separately, and obtains significant results, and are faster than the original algorithms by dealing with the low-latency ability of the shared memory of the GPU [32].

III. THE IMPLEMENTATION OF THE ALGORITHM TECHNIQUE

3.1 E-Abdulrazzaq algorithm

The technique of the E-Abdulrazzaq algorithm depends on two phases the preprocessing phase and the searching phase. The preprocessing phase is reliant on the selected functions of two algorithms (Abdulrazzaq and Berry-Ravindran) comprising Prime and composite numbers functions; Boyer-Moore bad character (bmBc) function, Berry-Ravindran bad character (brBc) function, and the hash function. In the interim, The searching phase in the E-Abdulrazzaq algorithm has three steps. In the first step, the hashing of the prime numbers in the text window is calculated and compared with the hashing value of the prime number characters in the pattern. If a match is obtained, the comparison between the prime number characters of the text window and pattern proceeds. If a match is obtained, then the second step follows. In the second step, if a match is obtained in the first step, the comparison between hashing characters in the pattern and text window proceeds. The calculated hash value in the composite number characters in the text window. If a match is obtained in the hashing value, the comparison of the characters between the pattern and text proceeds. If a match is found between these characters, the third step follows; The comparison of character number one in the pattern with character number one in the text window proceeds. . However, if a mismatch or match is obtained for each step and in the hashing comparison or in the character comparison, the shifting will depend on the maximum value of m from the (bmBc) table and the (m + 1 and m + 2) value from the (brBc) table.

3.2 Parallel of E-Abdulrazzaq algorithm using OpenMP

This section highlights the parallel design of E-AbdulRazzaq algorithm and the parallel process includes the following steps:

a) The first step is executed *by #pragma omp* parallel shared (), which share threads that work together. The data becomes visible and can be accessed by a group of threads wherein the same address is accessible to all threads. The variables are shared in the shared region in all steps except the repetition counter loop. In this function, a set of variables, namely, pattern *x*, length of pattern m, text *y*, and length of text n, are used as parameters in the algorithms. The functions used for shifting and building tables, such as *bmBc* and *brBc*, and two functions, such as prime/composite function and set prime/ set composite function are used in the proposed algorithm.

b) In the second step, the threads are created with each core containing one thread because the cores inside the processors of the Khawarizmi cluster have single threads. The number of threads (i.e., OMP) that run in a parallel region and can be controlled by a specific function is related to the number of threads in the team, returns the number between *0* to number of *threads−1*. The master threads return the number 0. The *omp_get_thread_num (iThreads)* function is used to control the number of threads.

c) In the third step, data are decomposed to small blocks *p* by dividing the array *y [ ]* related to the text and *p* blocks processed by individual threads in parallel *(p = number of threads)*. The division cannot be strictly n/p but can be $n/p + m-1$ because of the nature of the pattern search algorithm, and this division process called the overlapping boundaries among the small blocks *p*. The dotted line is the boundary of each block and the distance confines the dotted line into the non-dotted line as pattern $length - 1(m - 1)$, which denotes the pattern length added for each block. This procedure is applied to prevent any problems when the pattern originates from data that are divided into substrings; these types of data cannot be discovered because each substring handles different threads. This procedure is also employed to avoid problems when the same pattern string is compared with substrings.

d) In the fourth step, each thread inside each core takes one block after the division of data text into blocks (e.g., thread number 1 processes block number *1* and thread number *2* processes block number *2*). All threads then take the same pattern with each thread by using the pattern separately with a specific block depending on the technique of the algorithm. In proposed algorithm each thread applies hash function twice with prime and composite numbers in the pattern stages, and then starting in searching phase. After each thread completes the comparison between the segment of text (block) and the pattern, the time consumption is calculated.

e) In the fifth step, the reduction clause is performed. The reduction clause collects the results from partial accounts (i.e., threads) as a single result. This function informs the OMP to copy summarization counters into each thread separately and to sum up their values after each thread ends. The reduction function, reduction *(+: attempts, count, comparisons)*, is used in parallel proposed algorithm to calculate the time consumption by each thread. The results are calculated and set in the master thread. Figure 1 showed the Flowchart for parallel of E-AbdulRazzaq algorithm using OpenMP model.

3.3 Implementation and environment

3.3.1 Hardware and software

This experiment uses the Khawarizmi cluster from the School of Computer Sciences at USM (khawarizmi.cs.usm.my). The cluster contains the following nodes: Dell PowerEdge 1950 (master node) (2 × Quad-Core Intel Xeon E5450 3.00 GHz, 2 × 12 MB Cache, 1333 MHz FSB) and 2 × Dell PowerEdge 1950 (Slave nodes) (2 × Quad-Core Intel Xeon 1.6 GHz, 2 × 4 MB Cache, 1066 MHz FSB). The operating system is Linux (rocks cluster distribution 6.1, centos 6.3, 64- bit). The compiler used in this cluster is GCC 4.4.6, and the purpose of this cluster is to use the OpenMP paradigm.

3.3.2 Khawarizmi cluster architecture

This cluster includes three nodes. Each node contains two processors that have four cores each and one thread in each core. The buffered memory for the master node is 16 GB (2 × 8 GB, DDR-2 667 MHz ECC 2R), and the hard drive for the master node is a 1 TB 3.5-inch 7.2K RPM SATA II. The buffered memory for each slave node is 8GB (4 × 2 GB, DDR-2 667 MHz ECC 2R), and the hard drive is a 250 GB 3.5-inch 7.2K RPM SATA II.

3.3.3 Performance comparison

The proposed algorithm is dependent on the metrics that are used to compare the results of the sequential and parallel algorithms, in arrange to calculate the degree of enhancement between them. This study utilized metrics comprising the execution time and speedup [33].

The execution time is the time passed between the beginning and wrap-up of the execution time in one processor, which contains all processing times. The parallel time is the passed time between the begin of the first processor and the conclusion time of last processor. The successive and parallel elapsed time are indicated as Ts and Tp, respectively.

Speedup is utilized to get the focal points of parallel operation. The calculation of speedup depends on the successive and parallel consumed times or the proportion of the devoured time within the serial stage to the expended time within the parallel stage. The time is calculated in milliseconds and depends on the following equation:.

$$Speedup\ (S) = Ts / Tp$$

Where Ts and Tp are the time consumed in the sequential and parallel stages, respectively [34].

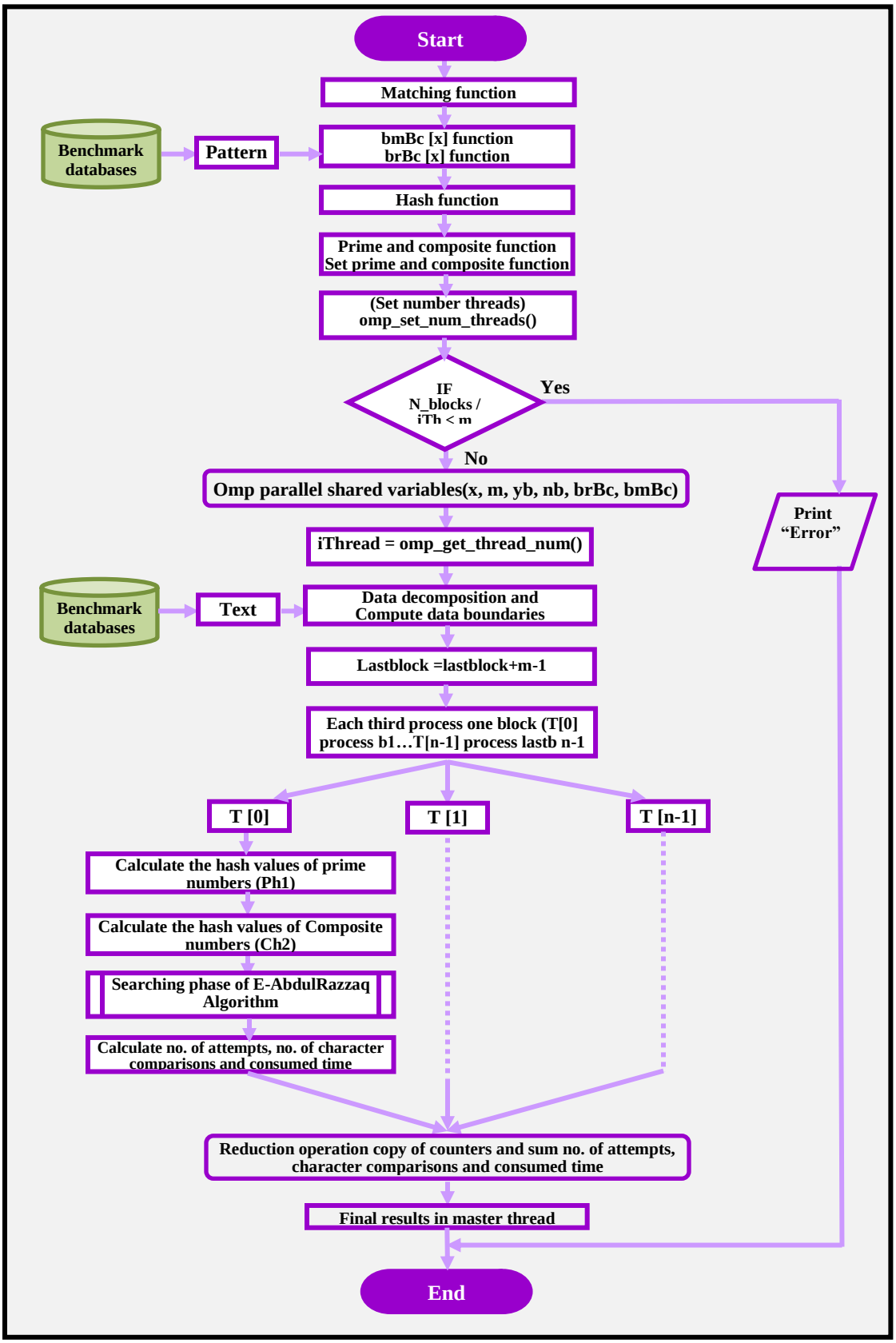

Fig.1 Flowchart for parallel E-AbdulRazzaq algorithm using OpenMP Model

3.3.4 Experiment design

The databases utilized in the experiment are downloaded from the (http://pizzachili.dcc.uchile.cl/texts.html).This experiment utilized the following databases; DNA, Protein, and English text, with 400 MB data size. The program is worked five times for each database and then the average is calculated. Eight cores are applied in this study because the number of cores in each node inside the cluster only rises to eight. Moreover, the comes about will be counterproductive and futile in the event that the number of cores increments. The number of cores utilized within the figures depends on the core power of two between $2^{\wedge 1}$ to $2^{\wedge 3}$. The sequential time utilized with the results is (Seq), and the numbers of cores utilized are two cores (C2), four cores (C4), and eight cores (C8). The sequential algorithm results are compared with the results of these cores.

IV. THE RESULTS AND ANALYSIS

The overall parallel time shows the best performance compared with the sequential time in a 400 MB data size when short and long patterns are used. The Protein database demonstrates the best time in most short and long patterns. DNA database is the worst performing database in all short and long patterns lengths, as shown in table 1., and table 2.

Table 1. Sequential and parallel execution times and short pattern length with 400MB database size

Length of pattern	Number of cores	Sequential execution time and parallel execution time (Ms.)		
		DNA	Protein	English
4	Sequential	3343	1966	**1930**
	2	1813	1111	**1095**
	4	911	558	**550**
	8	461	282	**278**
8	Sequential	2245	**1522**	1546
	2	1219	**840**	852
	4	611	**422**	429
	8	308	**214**	217
12	Sequential	2196	**1236**	1353
	2	1191	**675**	734
	4	597	**340**	371
	8	323	**173**	188
16	Sequential	1964	**1051**	1104
	2	1062	**572**	601
	4	529	**288**	303
	8	277	**147**	155
20	Sequential	2022	**1068**	1085
	2	1056	**574**	582
	4	531	**291**	294
	8	291	**148**	150
24	Sequential	1868	**959**	967
	2	986	**513**	516
	4	496	**258**	260
	8	251	**132**	133
28	Sequential	1995	**854**	857
	2	1055	**457**	458
	4	532	**232**	**232**
	8	269	**119**	**119**

Table 2. Sequential and parallel execution times and long pattern length when using 400MB database size

Length of pattern	Number of cores	Sequential execution time and parallel execution time (Ms.)		
		DNA	Protein	English
32	Sequential	2212	**843**	851
	2	1171	**448**	453
	4	591	**227**	229
	8	298	**117**	118
64	Sequential	3102	**685**	739
	2	1613	**360**	386
	4	812	**184**	197
	8	409	**96**	102
128	Sequential	3931	**638**	715
	2	1997	**332**	381
	4	1003	**170**	195
	8	505	**92**	104
256	Sequential	7130	**609**	709
	2	3624	**314**	364
	4	1820	**161**	187
	8	914	**87**	98
512	Sequential	12194	**707**	758
	2	6150	**362**	386
	4	3078	**185**	198
	8	1556	**98**	104
1024	Sequential	21877	977	**854**
	2	11014	500	**436**
	4	5508	254	**226**
	8	2776	132	**118**

The overall speedup performance is high in a 400 MB data size when short and long patterns are used. DNA obtains the best results in most short and long pattern lengths. No specific database obtains the worst data when short and long patterns are used, as shown in Table 3. and Table 4.

Table 3. Speedup and short pattern length when using 400MB database size

Length of pattern	Number of cores	Speedup (Stime / Ptime)		
		DNA	Protein	English
4	2	**1.84**	1.77	1.76
	4	**3.67**	3.52	3.51
	8	**7.25**	6.97	6.94
8	2	**1.84**	1.81	1.81
	4	**3.67**	3.61	3.60
	8	**7.29**	7.11	7.12
12	2	**1.84**	1.83	**1.84**
	4	**3.68**	3.64	3.65
	8	6.80	7.14	**7.20**
16	2	**1.85**	1.84	1.84
	4	**3.71**	3.65	3.64
	8	7.09	**7.15**	7.12
20	2	**1.91**	1.86	1.86
	4	**3.81**	3.67	3.69
	8	6.95	7.22	**7.23**
24	2	**1.89**	1.87	1.87
	4	**3.77**	3.72	3.72
	8	**7.44**	7.27	7.27
28	2	**1.89**	1.87	1.87
	4	**3.75**	3.68	3.69
	8	**7.42**	7.18	7.20

Table 4. Speedup and long pattern length when using 400MB database size

Length of pattern	Number of cores	Speedup (Stime / Ptime)		
		DNA	Protein	English
32	2	**1.89**	1.88	1.88
	4	**3.74**	3.71	3.72
	8	**7.42**	7.21	7.21
64	2	**1.92**	1.90	1.91
	4	**3.82**	3.72	3.75
	8	**7.58**	7.14	7.25
128	2	**1.97**	1.92	1.88
	4	**3.92**	3.75	3.67
	8	**7.78**	6.93	6.88
256	2	**1.97**	1.94	1.95
	4	**3.92**	3.78	3.79
	8	**7.80**	7.00	7.23
512	2	**1.98**	1.95	1.96
	4	**3.96**	3.82	3.83
	8	**7.84**	7.21	7.29
1024	2	**1.99**	1.95	1.96
	4	**3.97**	3.85	3.78
	8	**7.88**	7.40	7.24

4.1 Evaluation of parallel of the E-AbdulRazzaq algorithm

The results of the E-AbdulRazzaq algorithm in parallel time compared with sequential time are obtained by evaluating parallelization performance, which depends on two factors, namely, parallel execution time and speedup in short and long patterns with different types of databases and 400MB size.

Table 5. Performance evaluation for the average sequential and parallel execution times (ms) of the E-AbdulRazzaq algorithm

Performance comparison factors	400MB data size							
	Short				Long			
Types of databases	Seq	C2	C4	C8	Seq	C2	C4	C8
DNA	**2233**	**1197**	**601**	**311**	**8408**	**4262**	**2135**	**1076**
Protein	**1237**	**677**	**341**	**174**	**743**	**386**	**197**	**104**
English	1263	691	348	177	771	401	205	107

Parallel time results in better performance than sequential time when short and long patterns with 400MB data sizes are used. When the number of cores increased, the parallel time decreased because the communication time increased [34], [35]. Time in DNA data increases when long patterns are used because of the small alphabet size and the hash function technique, which depends on the repeated checking of characters, thus increasing time consumption in long patterns [12]. The best sequential results recorded are 1237 ms in short patterns, and 743 ms in long patterns. The worst sequential results are 2233 ms in short patterns with 400 MB data sizes, and 8408 ms in long patterns. The best parallel results when using short patterns with 400 MB data size are respectively presented as follows: two cores, 677 ms; four cores, 341 ms; eight cores, 174 ms. The best parallel results when using long patterns with 400 MB data size are respectively presented as follows: two cores, 386 ms; four cores, 197 ms; eight cores, 104 ms. The worst parallel results when short patterns with 400 MB data size are used are respectively presented as follows: with two cores, 1197 ms ; four cores, 601 ms; eight cores, 311 ms. The worst parallel results when long patterns with 400 MB data sizes are used are respectively presented as follows:: two cores, 4262 ms; four cores, 2135 ms; eight cores, 1076 ms (Table 5.).

The Protein database obtains the best sequential and parallel times when using short and long patterns with 400 MB data size because the E-AbdulRazzaq technique contains efficient functions (bmBc and hash) for dealing with the Protein database. The DNA database obtains the worst results in sequential and parallel times because of the small alphabet size, which require further shifting and time. Parallel time is affected by the type of database because the difference in the alphabet size used in a database affected its performance [36].

Table 6. Performance evaluation of the average speedup of parallel E-AbdulRazzaq

Performance comparison factors	400MB data size					
	Short			Long		
Types of databases	C2	C4	C8	C2	C4	C8
DNA	**1.87**	**3.72**	**7.17**	**1.95**	**3.89**	**7.72**
Protein	1.84	3.64	7.15	1.92	3.77	7.15
English	1.84	3.64	7.15	1.92	3.76	7.18

The results show a high speed up when using short and long patterns in 400 MB data sizes. The E-AbdulRazzaq algorithm has high sequential time because it can calculate the hashing values for all prime number characters in the first step in the searching technique, that is, this algorithm has character selectivity, does not depend on the character type of databases, and has high consumption of sequential time, particularly when long patterns are used. The best speedup results in short patterns with 400 MB data size are respectively presented as follows: two cores, 1.87; four cores, 3.72; eight cores, 7.17. The best speedup results in long patterns 400 MB data sizes are respectively presented as follows: two cores, 1.95; four cores, 3.89; eight cores, 7.72. The worst speedup results in short patterns 400 MB data size are respectively presented as follows: two cores, 1.84; four cores, 3.64; eight cores, 7.15. The worst speedup results in long patterns with 400 MB data size are respectively presented as follows: two cores, 1.92; four cores, 3.76; eight cores, 7.15 (Table 6.).

The best speedup results are found in DNA when short and long patterns with 400 MB data size are used because DNA exhibit the highest rate of improvement in parallel time compared with sequential time among the databases. Moreover, speedup increases when parallel time is shorter than sequential time and vice versa. No specific database

obtains the worst data when short and long patterns are used because stability is observed in the speedup of most databases.

V. CONCLUSION

The performance of the sequential and parallel algorithm will be measured based on the execution time and speed up when using short and long patterns with various databases and 400MB data size. The parallelization of the E-AbdulRazzaq algorithm has gotten high execution comes about in comparison to consecutive execution when using the OpenMP model as a multi-core technology. The Protein and DNA databases obtained the best and worst sequential and parallel times using enhanced hybrid algorithm, respectively. Among the databases studied, DNA obtained the best results in terms of speedup when using E-AbdulRazzaq algorithm ,while no specific database was determined to be the worst.

References

[1] C. Ryu, T. Lecroq, and K. Park, "Fast string matching for DNA sequences", Theoretical Computer Science, vol. 812, pp. 137-148, 2020.

[2] A. A. AbdulRazzaq, Nur'Aini Abdul Rashid, A. Ahmed 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.

[3] D. Jargalsaikhan, D. Hendrian, R. Yoshinaka, and A.Shinohara, Parallel Algorithm for Pattern Matching Problems Under Substring Consistent Equivalence Relations, LIPIcs, 33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022), Vol. 223, 2022, DOI: 10.4230/LIPIcs.CPM.2022.28.

[4] 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.

[5] X. Qi, B. Liu, Y. Li, Y. Du, Y. Li, D. Niu, A Parallel BMH String Matching Algorithm Based on OpenMP, 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2019. DOI: 10.1109/HPCC/ SmartCity/ DSS.2019.00026

[6] W. Dababat, and M. Itriq, Parallel Enhanced Pattern Matching Algorithm with Two Sliding Windows PETSW, International Journal of Computer Applications ,Vol; 179 (18), pp.0975 – 8887,2018.

[7] 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.

[8] P. NEAMATOLLAH , M. HADI , AND M. NAGHIBZADEH, Simple and Efficient Pattern Matching Algorithms for Biological Sequences, 2020 25th International Computer Conference, (CSICC), pp. 23838 - 23846, 2020.

[9] S .S. M. Al-Dabbag,. and Y. M. Abdal., Parallel Hybrid String Matching Algorithm Using CUDA API Function, 2021 International Conference on Computing and Communications Applications and Technologies (I3CAT), 2021. DOI: 10.1109 /I3CAT53310. 2021. 9629415.

[10] 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 (IJEECS), vol. 22, no. 3, pp. 1529-1539, 2021, doi: 10.11591/ijeecs.v22.i3.pp1529-1539.

[11] L. S. N. Nunes, J. L. Bordim, Y. I., 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.

[12] 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.

[13] 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/ijdmb.2015.072101.

[14] 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.

[15] 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.

[16] A. Hudaib, R. Al-khalid, D. Suleiman, and M. Abd Alfattah Itriq,"A Fast Pattern Matching Algorithm with Two Sliding Windows (TSW) A Fast Pattern Matching Algorithm with Two Sliding Windows (TSW)," Journal of Computer Science, vol. 4, no. 5, pp. 393-401, 2008, doi: 10.3844/jcssp.2008.393.401.

[17] A. Hudaib, D. Suleiman, and A. Awajan, "A Fast Pattern Matching Algorithm Using Changing Consecutive Characters," Journal of Software Engineering and

Applications (JSEA), vol. 9,no. 8, pp. 399-411, 2016, doi: 10.4236/jsea.2016.98026.

[18] C. S. Kouzinopoulos, Parallel and Distributed Implementations of Multiple and Two-Dimensional Pattern Matching Algorithms. Doctoral Thesis Department of Applied Informatics, University of Macedonia, 2013.

[19] D. Dhar, L. Hegde, S. M. Patil, S. Chickerur, "Parallelization of Protein Clustering Algorithm Using OpenMP," (2018, April). In InternationalConference on Advances in Computing and Data Sciences, 2018, pp. 108-118, 2018.

[20] C. S. Kouzinopoulos, P. D Michailidis, and K. G. Margaritis, "Parallel Implementation of Exact Two Dimensional Pattern Matching Algorithms using MPI and OpenMP," 9th Hellenic European Research on Computer Mathematics and its Applications Conference" pp. 1-6, 2009.

[21] M. Hemnani, "Parallel processing techniques for high performance image processing applications," 2016 IEEE Students' Conference on Electrical, Electronics and Computer Science (SCEECS), pp. 1-4, 2016, doi: 10.1109/SCEECS.2016.7509316.

[22] K. N. Rai, K. Nath Rai, and V. Kumar Singh, "A Parallel Processing Technique Based on GMO and BCS for Medical Image Encryption," International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 9, no. 3, pp. 3418-3427, 2020.

[23] J. Cámara, J. Cuenca, L. P. García, and D. Giméneza, "Empirical Modelling of Linear Algebra Shared-Memory Routines Empirical Modelling of Linear Algebra Shared-Memory Routines," Procedia Computer Science, vol 18, pp. 110-119, 2013, doi: 10.1016/j.procs.2013.05.174.

[24] R. Jin, G. Yang, and G. Agrawal, "Shared Memory Parallelization of Data Mining Algorithms: Techniques, Programming Interface, and Performance," IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 1, pp. 1-19, 2005, doi: 10.1109/TKDE.2005.18.

[25] N. P. Tran, M. Lee, and D. Hoon Choi, "Cache Locality-Centric Parallel String Matching on Many-Core Accelerator Chips," Hindawi Publishing Corporation, Scientific Programming, vol. 2015, pp. 1-21, 2015, doi: 10.1155/2015/937694.

[26] A.A. Hnaif, A. Aldahoud, M. A Alia, I. S. Al'otoum, and D. Nazzal, ,'Multiprocessing scalable string matching algorithm for network intrusion detection system', Int. J. High Performance Systems Architecture, Vol. 8, No. 3, pp.159–168, 2019.

[27]A. A. Hasan, N. Abdul Rashid, A. A. Abdulrazzaq, and M. A. Abu-Hashem, "String Matching Algorithms for Intrusion Detection System A Survey and Taxonomy," International Journal of Advancements in Computing Technology, vol. 5, no. 8, pp. 317-333, 2013, doi:10.4156/ijact.vol5.issue8.36.

[28] Hung. C.L., Hsu. T.H., Wang. H.H., Lin. C.Y., GPU-based Bit-Parallel Multiple Pattern Matching Algorithm, 2018 IEEE 20th International Conference on High Performance Computing and Communications, 2018, DOI: 10.1109/HPCC/SmartCity/DSS.2018.00205.

[29] K. B. Raju, , C. S. Rao, , and S.V. Raju, 'A Frame Work for Parallel String Matching-A Computational Approach with Omega Model." Global Journal of Computer Science and Technology Hardware & Computation (GJCST), vol.13, no.2, pp: 13-20, 2013.

[30] C. S. Rao, , K. B. Raju, , and S.V. Raju, "Parallel String Matching with ulti Core Processors-A Comparative Study for Gene Sequences". Global Journal of Computer Science and Technology Hardware & Computation (GJCST), vol.3,no.1,pp: 27-42, 2013.

[31] A. Rasool, and N. Khare," Parallelization of KMP String Matching Algorithm on Different SIMD architectures: Multi-Core and GPGPU's." International Journal of Computer Applications, vol.49, pp:26-28, 2012.

[32] C. S. Kouzinopoulos, and K. Margaritis, " String Matching on a multicoreGPU using CUDA," In Proceedings of the 13th Panhellenic Conference on nformatics, pp:14-18, 2009.

[33] A. A. Alsaheel, A. H. Alqahtani, and A. M. Alabdulatif, "Analysis of Parallel Boyer-Moore String Search Algorithm," Global journal of computer science and technology, vol. 13, no. 1, pp. 1-7 2013.

[34] 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, pp. 203-209, 2013, doi: 10.1109/ICCSCE.2013.6719959.

[35] 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.

[36] C. S. Kouzinopoulos, P. D.Michailidis, and K. G. Margaritis, "Performance Study of Parallel Hybrid Multiple Pattern Matching Algorithms for Biological Sequences", In Proceedings of the International Conference on Bioinformatics Models, Methods and Algorithms, pp:182-187, 2012.