Indonesian Journal of Electrical Engineering and Computer Science
Vol. 22, No. 3, June 2021, pp. 1529~1539
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v22.i3.pp1529-1539

Parallel implementation of maximum-shift algorithm using OpenMp

Atheer Akram AbdulRazzaq[1], Qusay Shihab Hamad[2], Ahmed Majid Taha[3]
[1-3]Businesses Informatics College, University of Information Technology and Communications, Baghdad, Iraq
[3]Soft Computing and Data Mining Center, Universiti Tun Hussein Onn, Johor Malaysia Universiti Teknologi Malaysia (UTM), Malaysia

Article Info

Article history:

Received Oct 23, 2020
Revised Apr 29, 2021
Accepted May 3, 2021

Keywords:

Database types
Efficiency
Maximum-shift algorithm
OpenMP directive
Parallel execution time
Speedup

ABSTRACT

String matching is considered as one of the center issues within the field of computer science, where there are numerous computer applications that supply the clients with string matching services. The increment within the number of databases which are created and protected in numerous computer gadgets had impacted researchers with the slant towards getting robust techniques in tending to this issue. In this study, the Maximum-Shift string matching algorithm is chosen to be executed with multi-core innovation through the utilization of OpenMP paradigm, in order to decrease the successive time, and increment the speedup and efficiency of the algorithm. The deoxyribonucleic acid (DNA), protein and the English text datasets are utilized to test the parallel execution that influences the Maximum-Shift algorithm execution when utilized with multi-core environment. The results demonstrated that the execution is affected by the performance between the parallel and consecutive execution of Maximum-Shift algorithm by data type. The English text appeared ideal comes about within the parallel execution time as compared to other datasets, whereas the DNA database set appeared the most elevated comes about when compared to other data types in terms of speedup and efficiency capabilities.

Corresponding Author:

Atheer Akram Abdul Razzaq
Businesses Informatics College
University of Information Technology and Communications
Baghdad, Iraq
Email: athproof@uoitc.edu.iq

1. INTRODUCTION

The categorization of parallel handling is subordinate upon the way of the association between the components of processing, and is categorized into two fundamental structures. These structures are the distributed memory and shared memory. In parallelization, disseminated memory computers compatibility data amid the transmission and gotten of messages through the communication arrange. In any case, within the parallelization of shared memory computers (processors of multicore and frameworks of multiprocessor), they have a common get to of shared memory. OpenMp is considered the foremost utilized application programming interface (API) for the parallel handling of shared memory. It may be a collection of run-time library schedule, mandates and develops, and OpenMP Environment Factors which are bolstered by the frameworks of multi-core, processors of shared memory, and the compilers and clusters. Moreover, the OpenMp does not require the administration of threads because it is subordinate upon certain parallelization programs highlights on the computer frameworks of shared memory [1], [2]. The parallelization algorithm has to select the parts of parallel that contains it, and it makes the appropriateness between the speed of

execution and the parallelization algorithm. Parallel is characterized as the necessity in finding instructions, chain of instructions, or certain parts of the code that are executed with diverse cores or processors at the same time [1]. The parallelization of shared memory is utilized in various computer applications such as intrusion detection systems (IDS) [3], [4] linear algebra [5], string matching algorithms [6] bioinformatics [7], data mining [8], image classification [9], and Therapeutic picture preparing [10].

This study concentrates on the issues which are related to the execution of the Maximum-Shift algorithm. Subsequently, the most address is "How to expand the speedup by diminishing the consecutive time of the Maximum-Shift algorithm by utilizing OpenMP parallel method?" Hence, the sub address of the main address is "How to demonstrate the execution enhancement of the parallel form of the Maximum-Shift string matching algorithm compared with its execution of the consecutive?" In this manner, the contribution of this paper is to examine the appropriateness of parallelization the Maximum-Shift algorithm on multi-core environment utilizing OpenMP.

2. RELATED WORKS

2.1. Exact string-matching algorithm

String matching is considered as a searching process executed to examine at two limited-lengths of strings. It can be characterized as exact string matching procedure utilized to discover all occurrences of the pattern P of length of $m\ (P\ [1] \ldots.\ P\ [m\text{-}1])$ within the text string T of length of $n\ (T\ [1] \ldots.\ T\ [n\text{-}1])$, where the m and $n > 0$ and the $m \leq n$., as well as both of the P and T have the same letter set $\sum$. [11]. It is predominantly and broadly utilized in different areas such as web search engine, artificial intelligence [12], [13], remote sensing, aerial photography, cartography, medical diagnostics [1], computational biology [14], compilers, and command interpreters [15]. The persistent challenges of string matching include the duplication of databases each two years, and this affected the speed of computers, which leads to the increment within the size of memory. Hence, utilize of the effective string-matching algorithms is fundamental to resolve these issues [11], [12].

There are various exact string matching algorithm in presence, such as the hybrid algorithm between the Berry-Ravindran and the Skip Search algorithms named as Berry Ravindran Skip Search (BRSS) algorithm. This algorithm is dependent upon the more superior properties of the original algorithms, where the skip search algorithm demonstrates favorable performance when it is utilized with small alphabets size. Meanwhile, the efficiency of the Berry-Ravindran algorithm is in the provision of a long shift during the searching technique. The preprocessing phase of this algorithm is dependent upon two tables (brBc); table created from the Berry-Ravindran algorithm, and the second table created is the bucket list. The searching phase involves the checking of the characters found in the bucket, of which the character is then arranged between the initial point of the search and the pattern to the location of that character in the bucket. The comparison is initiated from the left to the right, and when there is a match or a mismatch, the shifting depends on the bucket shift value being ≥ the bad character. Next, the bucket will use the shift value, and if the bad character shift ≥ from length of pattern, then the shifting is dependent on the bad character shift [16].

The two-sliding windows algorithm (TSW), depends on the pattern of sliding windows to check the text simultaneously from the left and right side. This algorithm utilizes two windows, one from the left and aligned to the text from the left side, and the second is from the right and is aligned to the text from the right side. The comparison process between the pattern and text occurs from both sides simultaneously, when there is a mismatch, the shifting will then occur for both of the two sides, from the left side a shift will be executed to the right, and the right will shift to the left side. The shift operation depends on the Berry- Ravindran shift technique, which takes two characters consecutively [17], [18].

In addition to this algorithm, another algorithm known as the franek-jennings-smyth (FJS) algorithm is a hybrid between Knuth-Morris-Pratt (KMP) pattern-matching algorithm and Quick search algorithm, where there are two stages to this algorithm. Within the to begin with stage there are two steps that are utilized by the algorithm for comparison. A comparison is made between the text window and the pattern, where the algorithm procedure depends on the Quick search algorithm. The comparison is started from the furthest right of the pattern, and when there's a mismatch, the shift depends on quick search implementation. Otherwise, the FJS will depend on the second stage. Within the second stage, the KMP matching technique will start a comparison from the left to right, whereas the shift is dependent on the first stage [19], [20].

2.2. Parallel of the exact string-matching algorithms

There are numerous parallel exact string matching algorithms that are implemented to extend the execution by diminishing the time consumed by the algorithms of the multi cores/ processors. One illustration of the parallel string matching algorithms that's utilized in security purposes is the Quick search algorithm utilizing OpenMp and the Pthread (Posix). The proposed strategy is seen to have moved forward

the execution of intrusion detection system (IDS), that driven to the improved discovery of the hacker technique, which transmits the new report to the network administrator to treat the assault [3].

Furthermore, there are hybrid parallel models between MPI and OpenMP utilized with string matching algorithms such as Baeza-Yates and Regnier, Karp-Rabin, Naive, Zhu-Takaoka, and the Baker-Bird exact two-dimensional online algorithms. The homogeneous cluster and multicore system were used with these algorithms to measure the performance. This study elucidated on the improvement in the performance of the algorithms when databases with big sizes of alphabet were utilized because of the increase in the mismatches in the processing operation of the algorithms [1].

There are certain exact string-matching algorithms that utilize the graphics processing unit (GPU), the Compute unified device architecture (CUDA) such as the Naive, Boyer-Moore-Horspool, Quick-Search and Knuth-Morris-Pratt algorithms which were utilized with tools of CUDA. Following the comparison of the sequential and parallel results in all of these algorithms in terms of consumed time when using different pattern sizes, the number of threads and different databases, it was discovered that the results of parallel implementation was faster of approximately 24 times than the sequential time when large text and small pattern size are utilized [21].

3. PROPOSED METHOD

This section contains the points of interest of the Maximum-Shift algorithm that included the preprocessing and searching phase technique. It explicates the imperative reasons for understanding the properties of the Maximum-Shift algorithm, as well as checking the behavior of identifying the parts of code that's time consuming and compute-intensive. Consequently, parallelization can be utilized within the programming of the OpenMP strategy.

3.1. Maximum-shift algorithm

The Maximum-Shift algorithm is a hybrid algorithm among the Quick search, Zhu-Takaoka, and Horspool algorithms. It has two phases comprising the preprocessing and searching phase. In the preprocessing phase, the algorithm depends on two selected good performance functions from Quick search and Zhu-Takaoka algorithms. The quick search bad character function, which thenceforth will be known as qsBc, is selected to improve the performance of hybrid algorithm by using maximum shift value (m+1), as shown in (1), while the Zhu-Takaoka bad character function or known as ztBc, is used in the algorithm to obtain the maximum shift value of (m-1) and (m), as shown in (2).

$$qsBc[x] = \begin{cases} (i: 0 \le i < m \text{ and } [m-i] = x) & \text{if } x \text{ occurs in } p, \\ m+1 & \text{otherwise} \end{cases} \quad (1)$$

$$ztBc[a,b] = k \begin{cases} k = m & x[0] \neq b \text{ and } ab \text{ does not occur in } x[0..m-2] \\ k = m-1 & x[0] = b \text{ and } ab \text{ does not occur in } x[0..m-2] \\ k \le m-2 & \text{and } x[m-k-2..m-k-1] = ab \text{ and } ab \text{ does not ocuur in } x[m-k-1..m-2] \end{cases} \quad (2)$$

The searching phase of the Maximum-Shift algorithm depends on the left to right comparison technique from Quick search algorithm, and also depends on the right to left comparison technique of Horspool algorithm. The comparison operation of this algorithm depends on the comparison of the first two characters from the rightmost between the text window and the pattern as the first phase. When there is a match, a comparison is made to the rest of characters from the left-most of the text window and the pattern as the second phase. If there is a match or a mismatch, then the shifting will depend on the maximum value between the *(m+1)* from qsBc function or *(m-1)* and *(m)* from ztBc function, (as shown in Figure 1) [22].

3.2. Parallel of maximum-shift algorithm using OpenMP paradigm

This part clarifies the objective of the study pertaining to the parallelization steps of Maximum-Shift algorithm. It entails the multicore implementation used on the Maximum-Shift algorithm and the performing of code by using the directives and the library functions of the OpenMP paradigm. Depending on the analysis of the Maximum-Shift algorithm in the previous part, the costly part in the string matching algorithm is in the checking of whether the character of the text window matches the character in the pattern [6]. To reduce the cost of the searching phase that includes the matching operation of the characters in the text window and the pattern, the parallelization is therefore utilized depending on the OpenMP model.

In the searching phase, the Maximum-Shift algorithm is executed by utilizing the multicore processing and the programming OpenMP environments. The platform of OpenMP executed in the programming is dependent upon the division of the whole data to the chunks (blocks) by the Fork operation and the join processes. The program is initially executed by the thread termed as the master thread, and this thread distributes the functions to the slave (workers) threads. The parallel Maximum-Shift algorithm initiates the execution in the serial program using the master thread, and continue up to the searching phase operation. In this stage, the worker threads are created. The number of threads will be 24 because the execution utilizes the cluster with one node and 24 cores. Each worker thread implements the searching phase operations on one block from the data that is divided into blocks, and when the execution is completed, the result will be transmitted into the master thread. The master thread then collects all of the results by using join operation process, and will then display the final output (final result). This process will be executed in a serial program, after the worker threads transmit the results for each data block to the master thread, which will then be automatically terminated, as shown in Figure 2.

Maximum-Shift Algorithm

```
1.  Maximum-Shift Algorithm (X [0 …..m −1], Y [0…….n−1])
2.  //Input: Pattern X, Text Y
3.  //Output: number of attempts and number of character comparisons of pattern with text
4.  ztBc[ASIZE][ASIZE]  //2D array to keep shift values
5.  qsBc[ASIZE]
6.  //Preprocessing//
7.        //  preztBc (P, m, ztBc )(preprocessing Zuh-Takaoka bad character function)
8.       // preqsBc (P, m, qsBc) (preprocessing Quick-Search bad-character function)
9.  //Searching//
10. j ← 0
11. While j <= n − m Do
12.       c1 = y[j + m − 1],  c2=y[j + m - 2];
13.       If (c1 == x[m - 1])
14.          If (c2 == x[m - 2])
15.             If(match(x,m - 2,y,j,&temp) == 1)
16.                Count   // output the first occurrence of pattern in text
17.             End If
18.          End If
19.       End If
20.     Output the first attempt and character comparisons
21.  //Shifting//
22.     j+= getmax(ztbc[y[j + m - 2]][y[j + m - 1]],qsbc[y[j + m]])
23. End While
```

Figure 1. The Maximum-Shift algorithm technique

3.3. Battuta cluster architecture

This study utilizes the Battuta Cluster Architecture, where this cluster is present in the Parallel and Distributed Processing Lab (PDCC) in the School of Computer Science, Universiti Sains Malaysia (USM). This cluster possesses one node with 24 cores, 2xIntel Xeon E5-2620 (2.00GHz/ 15MB) and it has (8GB DDR3 ECC DIMM, 2TB 7200RPM), (Nvidia Tesla Kepler K10 8GB GDDR5 Passive), (2xGigabit LAN Port/1xIPMI Port), in which case, the operating system of this cluster is Linux (Ubuntu 12.04.5 LTS) and the softwares that are used in this cluster are the OpenMP, OpenMPI and compute unified device architecture (CUDA).

3.4. The performance metrics

In this study the proposed algorithm is executed by using the OpenMP paradigm, and the results of the evaluation of this algorithm are dependent on the metrics that are utilized to compare the results of the parallel and sequential algorithms, in order to calculate the extent of improvement between them. This study used metrics consisting of the execution time, speedup, and efficiency [23]. The elapsed time that occurred between the beginning and ending of the performance of one processor is termed as the sequential time, while the time consumed between the starting time for the first processor to the finished time to the last processor is termed as the parallel time. The Ts denotes the sequential execution time, while Tp is the parallel execution time.

The speedup is utilized to obtain the advantages of a parallelization operation. The speedup is the ratio of consumed time between the sequential to the parallel stages. This ratio is calculated by using the ensuing equation.

$$\text{Speedup (S)} = Ts / Tp \quad (3)$$

Where the Ts denotes the elapsed time in the sequential stage, Tp denotes the elapsed time in the parallel stage, and S denotes the speedup. The ratio between the speedup and the number of (processors /cores) is termed as the efficiency. There are positive and reverse relationship between the number of (processors/ cores) and the speedup in the efficiency. The efficiency increases when the speedup increases. Meanwhile, in instances when the number of processors increased, consequently the efficiency will be reduced, as indicated in the ensuing (4).

$$E = S / P \quad (4)$$

Where the P denotes the number of (processors / cores), S denotes the speedup, and E denotes the efficiency.

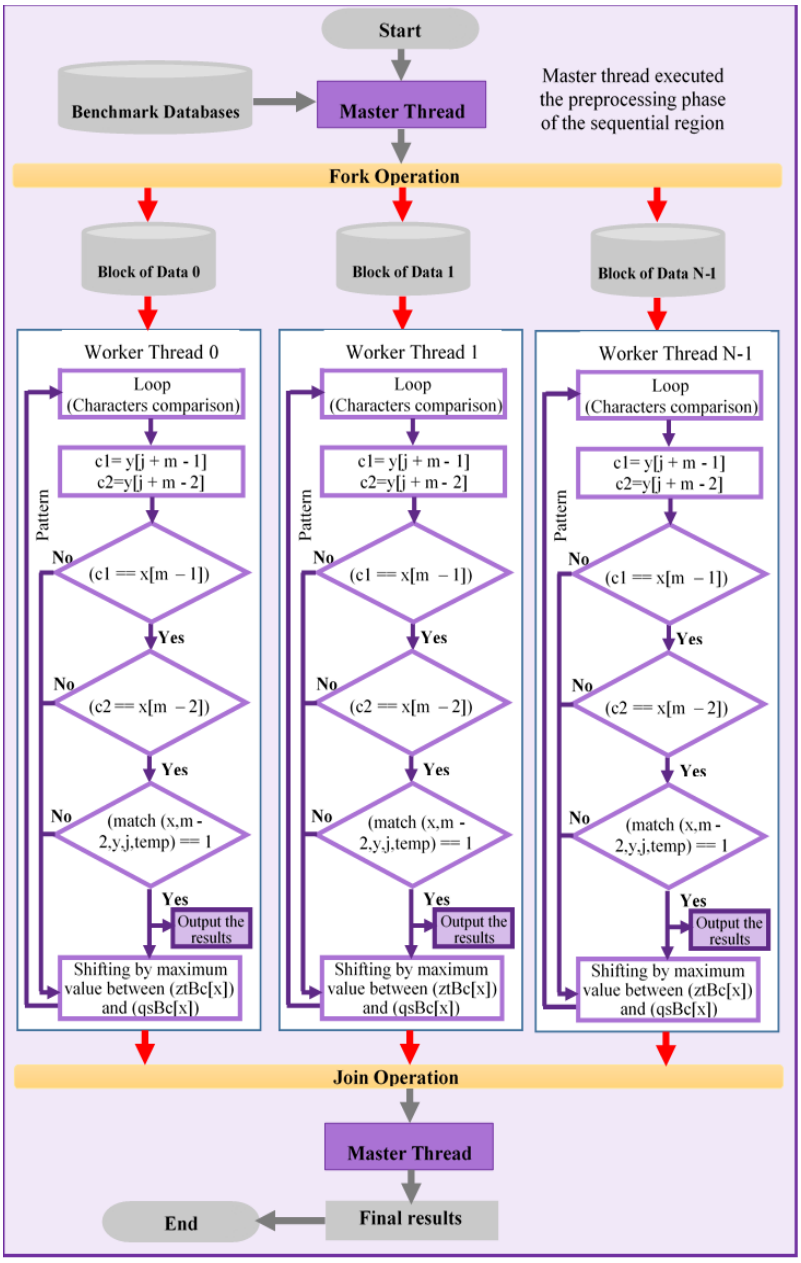

Figure 2. The parallel of maximum-shift algorithm using OpenMP model

4. THE EXPERIMENTAL RESULTS AND DISCUSSIONS

The major concept of parallelization on the Maximum-Shift Algorithm is for the improvement of the execution by comparing and measuring the improvement of the parallel to the sequential algorithm. The factors that are utilized for the evaluation of the performance of the algorithm are the execution time, speedup, and the efficiency. The standard benchmark databases that are employed in this study are deoxyribonucleic acid (DNA), Protein, and English text, where these databases are downloaded from the (http://pizzachili.dcc.uchile.cl/texts.html). The datasets that are utilized in the experiment are different in the alphabet size, where this type is used to analyze the behaviors of the algorithm in the different sizes of alphabet. The Maximum-Shift Algorithm in both the sequential and parallel algorithms implemented used the 1GB of data size, with different pattern lengths that are utilized to evaluate the behaviors of the algorithms. The pattern lengths are 4, 8, 10, 20, 40, 60, 80, 100 characters, which are selected randomly from text. In this study the numbers of cores utilized are 2, 4, 8, 16, 20, and 24 cores.

4.1. Parallel execution time evaluation

The parallel execution time includes the utilize of different pattern lengths and 1GB data size, as well as the utilize of different alphabet sizes such as DNA, Protein, and English text. The frame of DNA dataset can be found within the shape of 4 characters, which alludes to the establishment of chemical of the cell part, whereas the frame of Protein dataset comprises of 20 amino acids. In the mean time, the English text dataset can be found within the shape comprising of 100 letter set sorts. The results of the parallel execution time in the maximum shift algorithm revealed the best performance time in comparison to the sequential time. However, there is overhead that impacted on the parallelization time.

The overhead increases when the number of cores increases, due to the increase in the communication time, alongside the increase in the number of cores [24]. Through the use of 1GB data size of DNA, Protein and English database types as shown in Tables 1, 2, and 3. It is revealed that the parallel execution time decreased when the number of cores increased in all databases types. The English text dataset appeared the finest parallel execution time in Maximum-Shift algorithm when compared to DNA and Protein databases. The respective best parallel results are displayed as takes after: two cores, 164 ms; four cores, 94ms; eight cores, 73 ms; sixteen cores, 73ms; twenty cores, 74; twenty four cores, 77ms. The DNA database appeared the most noticeably worst parallel execution time in most results are displayed as follows: two cores, 4854 ms; four cores, 2143 ms; eight cores, 1172 ms; sixteen cores, 816 ms, whereas the Protein appeared the most worst results when utilizing twenty cores, 767 ms; and twenty four cores, 814 ms. The DNA got the most exceedingly bad comes about, because it takes a longer time to execute. Usually since the DNA has small alphabet size, whereas the English text has big alphabet size [25], as shown in Figure 3, Figure 4, and Figure 5.

Table 1. Sequential execution time and parallel execution time (ms.) when using 1GB of DNA database

Length of pattern	Sequential	Number of Cores: 2 cores	4 cores	8 cores	16 cores	20 cores	24 cores
4	9694	4854	2143	1172	816	719	737
8	5450	2749	1401	865	528	455	475
10	4079	2075	1266	730	498	475	442
20	3100	1559	782	499	338	352	310
40	2701	1323	700	460	353	408	350
60	3972	980	497	328	254	253	238
80	3834	920	648	324	247	208	242
100	1470	715	447	261	223	201	171

Table2. Sequential execution time and parallel execution time (ms.) when using 1GB of Protein database

Length of pattern	Sequential	Number of Cores: 2 cores	4 cores	8 cores	16 cores	20 cores	24 cores
4	5965	3274	1540	1071	791	767	814
8	4200	2134	1098	634	428	363	364
10	3477	1785	909	560	316	333	330
20	1586	785	402	292	183	203	194
40	903	444	256	162	122	130	129
60	539	279	238	133	147	122	122
80	469	242	145	189	122	120	118
100	390	208	123	129	119	121	123

Table 3. Sequential execution time and parallel execution time (ms.) when using 1GB of English database

Length of pattern	Sequential	Number of Cores: 2 cores	4 cores	8 cores	16 cores	20 cores	24 cores
4	3867	2258	993	631	407	434	405
8	3055	1530	857	423	272	215	244
10	2647	1127	811	377	212	232	213
20	1058	535	277	180	111	118	114
40	511	296	151	88	79	72	77
60	348	222	122	83	81	72	74
80	300	181	94	77	73	73	74
100	284	164	94	73	73	74	77

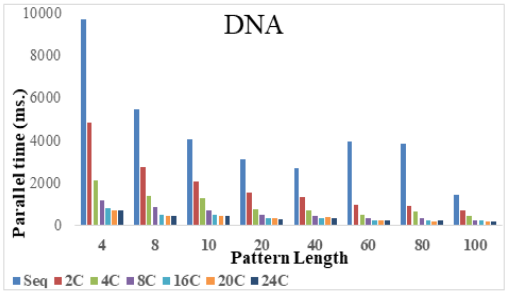

Figure 3. Evaluations of the parallel time when using 1GB data size with DNA datatype

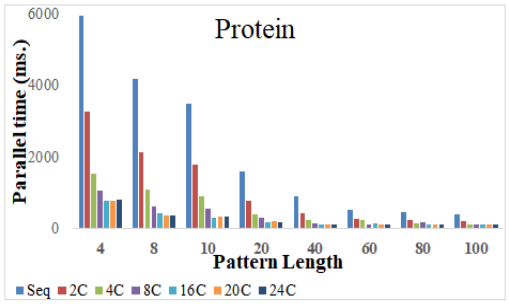

Figure 4. Evaluations of the parallel time when using 1GB data size with Protein datatype

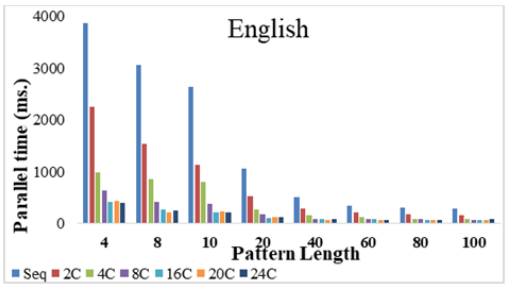

Figure 5. Evaluations of the parallel time when using 1GB data size with English text datatype

4.2. The speedup evaluation

As shown in Tables 4, 5, and 6, the speedup of the maximum shift algorithm showed the high speedup capabilities, with high parallel execution time as compared to sequential execution time. From the results of speedup, there is little overhead that appeared during the parallel process, as this is due to the communication time consumed by some of the cores were high, and due to the large data size used.

The speedup results indicated high speed outcomes when 2 and 4 cores are used, while the speedup outcome showed a gradual reduction when 8 cores and above are used in the protein database set and in English text datasets. However, in the DNA database set, the reduction of speedup began when 16 and above cores were used , as shown in Figures 6, 7 and 8, as well as in Tables 4, 5 and 6. The reason for the low speedup of protein sequence and English text when compared to the DNA dataset is because of the big alphabet size of the dataset, which had led to the reduction in the number of shifting in the searching technique of the maximum shift algorithm. Additionally, it is due to the decrease in the parallel time when long pattern and increased number of cores are used. All of the mentioned factors resulted in the low speed up when big alphabet size is used. Meanwhile, the speedup results using the DNA database set is not affected extensively because it consists of small alphabet size, in addition to the fact that during the searching phase, there is an increase in the shifting technique, which also led to an increase in elapsed duration time. The best speedup results are displayed as follows: two cores, 1.77; four cores, 3.98; eight cores, 7.28; sixteen cores, 10.46; twenty cores, 11.87; twenty four cores, 11.58. The most exceedingly bad speedup comes about are displayed as follows: two cores, 1.4; four cores, 2.02; eight cores, 2.19; sixteen cores, 3; twenty cores, 2.95; twenty four cores, 2.9, as appeared in Tables 4, 5, 6.

Table 4. The Speedup when using 1GB of DNA database

Length of pattern	Number of Cores					
	2 cores	4 cores	8 cores	16 cores	20 cores	24 cores
4	1.76	3.98	7.28	10.46	11.87	11.58
8	1.75	3.44	5.57	9.12	10.59	10.14
10	1.77	2.90	5.02	7.36	7.72	8.30
20	1.76	3.51	5.51	8.13	7.81	8.86
40	1.70	3.21	4.88	6.36	5.50	6.42
60	1.69	3.34	5.06	6.54	6.59	6.97
80	1.70	2.41	4.83	6.33	7.52	6.46
100	1.74	2.78	4.77	5.58	6.19	7.27

Table 5. The Speedup when using 1GB of Protein database

Length of pattern	Number of Cores					
	2 cores	4 cores	8 cores	16 cores	20 cores	24 cores
4	1.61	3.41	4.91	6.65	6.68	6.46
8	1.68	3.26	5.64	8.36	9.85	9.83
10	1.65	3.25	5.27	9.34	8.87	8.95
20	1.73	3.38	4.65	7.43	6.69	7.01
40	1.7	2.96	4.67	6.2	5.82	5.87
60	1.72	2.02	3.61	3.27	3.93	3.93
80	1.71	2.85	2.19	3.39	3.44	3.5
100	1.72	2.9	2.77	3	2.95	2.9

Table 6. The Speedup when using 1GB of English database

	Number of Cores					
Length of pattern	2 cores	4 cores	8 cores	16 cores	20 cores	24 cores
4	1.4	3.17	4.99	7.74	7.26	7.78
8	1.74	3.11	6.29	9.79	12.38	10.91
10	1.66	2.31	4.98	8.85	8.09	8.81
20	1.74	3.36	5.17	8.38	7.88	8.16
40	1.51	2.95	5.07	5.65	6.19	5.79
60	1.41	2.65	3.76	3.85	4.33	4.22
80	1.48	2.85	3.48	3.67	3.67	3.62
100	1.59	2.78	3.58	3.58	3.53	3.39

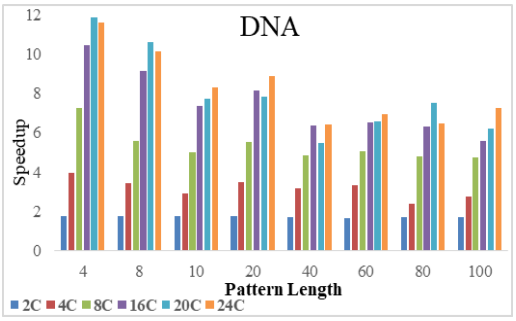

Figure 6. Evaluations of the speedup when using 1GB data size with DNA datatype

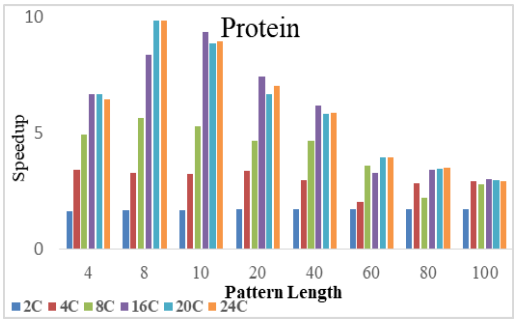

Figure 7. Evaluations of the speedup when using 1GB data size with Protein datatype

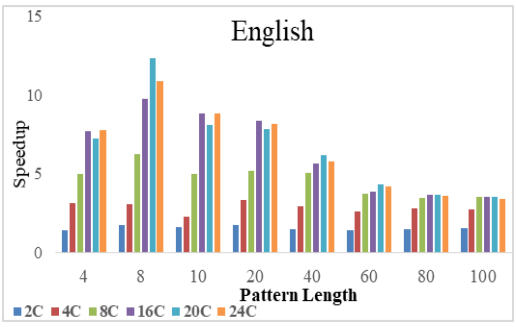

Figure 8. Evaluations of the speedup when using 1GB data size with English text datatype

4.3. The efficiency evaluation

As indicated in Tables 7, 8 and 9, the efficiency of the maximum shift algorithm is high when 2, 4, and 8 cores are used for DNA, Protein and English databases. However, the efficiency began to decrease when 16 cores and above are used alongside long pattern lengths of 60, 80, and 100. Nevertheless, when the DNA dataset is utilized, the efficiency began to reduce with the utilization of 20 cores. The reason for the weaknesses in the efficiency began with the long pattern length, and when 16 cores and above are utilized in the datasets is because of the overhead that impacted the parallel process. In this experiment, when data size of 1GB is employed, there is a reduction in the efficiency when the number of cores is increased. Furthermore, due to the positive relationship between the efficiency and the speedup, the efficiency is observed to increase when the speedup increased, and vice versa. Due to the mentioned reason, the efficiency is observed to decrease when 8 cores and core values above 8 cores and long pattern are used. The utilization of the DNA database showed the highest efficiency outcome in comparison to the Protein and English Text datatypes, as shown in Figures 9, 10 and 11. The best efficiency results are displayed as takes after: two cores, 88%; four cores, 99%; eight cores, 91%; sixteen cores, 65%; twenty cores 59%; twenty-four cores 48%. The worst efficiency results are displayed as follows: two cores, 69%; four cores, 50%; eight cores, 27%; sixteen cores, 18%; twenty cores, 14%; twenty-four cores, 12%; as shown in Tables 7, 8 and 9.

Table 7. The Efficiency when using 1GB of DNA database

Length of pattern	Number of Cores 2 cores	4 cores	8 cores	16 cores	20 cores	24 cores
4	87%	99%	91%	65%	59%	48%
8	87%	85%	69%	57%	52%	42%
10	88%	72%	62%	46%	38%	34%
20	88%	87%	68%	50%	39%	36%
40	84%	80%	61%	39%	27%	26%
60	84%	83%	63%	40%	32%	29%
80	85%	60%	60%	39%	37%	26%
100	86%	69%	59%	34%	30%	30%

Table 8. The Efficiency when using 1GB of Protein database

Length of pattern	Number of Cores 2 cores	4 cores	8 cores	16 cores	20 cores	24 cores
4	80%	85%	61%	41%	34%	26%
8	83%	81%	70%	52%	49%	40%
10	82%	81%	65%	58%	44%	37%
20	86%	84%	58%	46%	33%	29%
40	85%	73%	58%	38%	29%	24%
60	86%	50%	45%	20%	19%	16%
80	85%	71%	27%	21%	17%	14%
100	85%	72%	34%	18%	14%	12%

Table 9. The Efficiency when using 1GB of English database

Length of pattern	Number of Cores 2 cores	4 cores	8 cores	16 cores	20 cores	24 cores
4	69%	79%	62%	48%	36%	32%
8	86%	77%	78%	61%	61%	45%
10	83%	57%	62%	55%	40%	36%
20	86%	83%	64%	52%	39%	33%
40	75%	73%	63%	35%	30%	24%
60	70%	63%	46%	24%	21%	17%
80	74%	71%	43%	22%	18%	15%
100	79%	69%	44%	22%	17%	14%

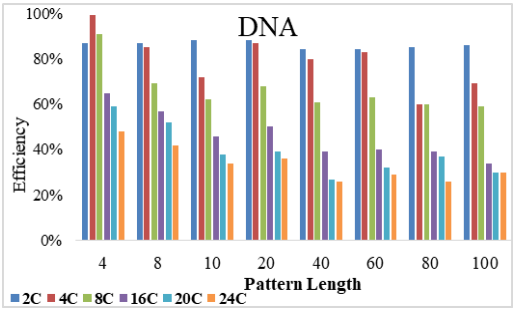

Figure 9. Evaluations of the efficiency when using 1GB data size with DNA datatype

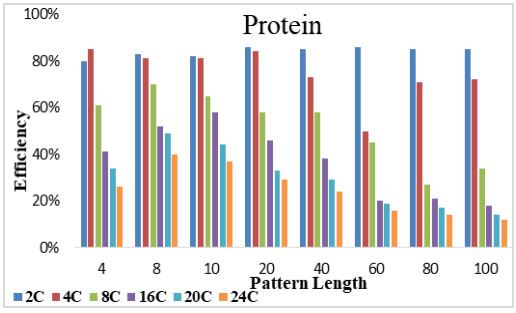

Figure 10. Evaluations of the efficiency when using 1GB data size with Protein datatype

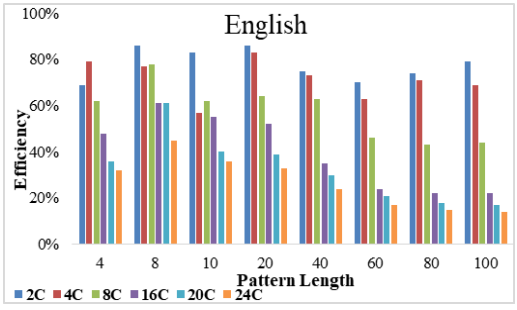

Figure 11. Evaluations of the efficiency when using 1GB data size with English text datatype

5. CONCLUSION

This study about uncovers the findings of the parallel executing time, speedup and the efficiency of the parallel time and comparing that time to sequential time of Maximum-Shift algorithm when utilized with multi-core technology and OpenMP directive, in addition to the utilization of several datasets with the utilize of 1GB size databases and pattern lengths of 4 to 100 characters. The parallel maximum-shift algorithm appeared tall execution capabilities when the OpenMP model is utilized, and when it is compared to consecutive. The Maximum-Shift algorithm gotten way better comes about with great execution after the execution of parallelization by the most excellent parallel execution time as compared to successive time, the

speedup and the effecincy factors. The English text information achieved the ideal comes about in parallel execution time, in the mean time, the utilization of the DNA database set gotten the most elevated positive comes about in speedup and in efficiency. As a conclusion, we suggest the multi-core environment as the appropriate platform for parallelization the Maximum-Shift string matching algorithm. For future work the Maximum-Shift algorithm may well be enhanced by executing the Maximum-Shift algorithm to other multi core standards such as Pthread program and multiprocessors models such as MPI, as well can improved the algorithm by parallelization the preprocessing with searching phase.

REFERENCES

[1] 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.

[2] Y. Y. Leow, C. Y. Ng, and W.F. Wong, "Generating hardware from OpenMP programs," *Proceedings of IEEE International Conference on Field Programmable Technology*, pp. 73-80, 2006, doi: 10.1109/FPT.2006.270297.

[3] A. Hnaif, A. Mohammad, O. A. Abouabdalla, and S. Ramadass, "Parallel Quick Search Algorithm to Speed Packet Payload Filtering in NIDS," *Journal of Engineering Science and Technology*, vol. 4, no. 2, pp. 1-7, 2008.

[4] 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.

[5] Jesús Cámara, Javier Cuenca, Luis-Pedro García, and Domingo 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.

[6] N. Hazim, M. A. Sahib Naser, and Zaid 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.

[7] Nhat-Phuong Tran, Myungho Lee, and Dong 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.

[8] 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.

[9] 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.

[10] 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.

[11] A. Abdulrazzaq, Nur'Aini Abdul Rashid, and A. Majid Taha, "The Enhanced Hybrid Algorithm for the AbdulRazzaq and Berry-Ravindran Algorithms," *International Journal of Engineering & Technology*, vol. 7, no. 3, pp. 1709-1717, 2018, doi: 10.14419/ijet.v7i3.12436.

[12] A. A. Abdul Razzaq, 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.

[13] A. A. Hasan, Nur'Aini Abdul Rashid, and A. Akram Abdulrazzaq, "Multi-pattern string matching algorithms comparison for intrusion detection system," *AIP Conference Proceedings*, vol. 1635, no. 1. 2014, doi: 10.1063/1.4903558.

[14] A. A. Abdulrazzaq, Nur'Aini Abdul Rashid, A. A. Hasan, and M. A. Abu-Hashem, *et al*., "New Searching Technique of Hybrid Exact String Matching algorithm," .*International Review on Computers and Software (I.RE.CO.S.)*, vol. 11, no. 10, pp. 884-897, 2017, doi: 10.15866/irecos.v11i10.10321.

[15] A. A. Abdulrazzaq, Nur'Aini Abdul Rashid, and M. A. Abu-Hashem, "A New Efficient Hybrid Exact String Matching Algorithm and Its Applications," *Life Science Journal (Life Sci J)*, vol. 11, no. 10, pp. 474-488, 2014, doi: 10.15866/irecos.v11i10.10321.

[16] A. Al-mazroi, "A Fast Hybrid Algorithm for the Exact String Matching Problem," *American Journal of Engineering and Applied Sciences*, vol. 4, no. 1, pp. 102-107, 2011, doi: 10.3844/ajeassp.2011.102.107.

[17] 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.

[18] 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.

[19] S. I. Hakak, *et al*., "Exact String Matching Algorithms: Survey, Issues, and Future Research Directions," *IEEE Access*, vol. 7, pp. 69614-69637, 2019, doi: 10.1109/ACCESS.2019.2914071.

[20] F. Franek, C. G.Jennings, and W. F. Smytha, "A simple fast hybrid pattern matching algorithm,"' *J. Discrete Algorithms*, vol. 5, no. 4, pp. 682-695, 2007, doi: 10.1007/11496656_25.
[21] S. Kouzinopoulos and K. G. Margaritis, "String Matching on a multicore GPU using CUDA," *Conference: PCI 2009, 13th Panhellenic Conference on Informatics, Corfu, Greece*, 2009, pp. 1-5, doi: 10.1109/PCI.2009.47.
[22] A. Kadhim and N. A. Abdul Rashid, "Maximum-Shift String Matching Algorithms." *International Conference on Computer and Information Sciences (ICCOINS)*," pp. 6-11, 2014, doi: 10.1109/ICCOINS.2014.6868423.
[23] 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.
[24] K. Hamidouche, A. Borghi, P. Esterie, J. Falcou, and S. Peyronnet, "Three High Performance Architectures in the Parallel APMC Boat," *Ninth International Workshop on Parallel and Distributed Methods in Verification/Second International Workshop on High Performance Computational Systems Biology, IEEE*, pp. 20-27, 2010, doi: 10.1109/PDMC-HiBi.2010.12.
[25] 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, doi:10.5220/0003769801820187.

BIOGRAPHIES OF AUTHORS

Dr. Atheer Akram Abdul Razzaq 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 a senior lecturer 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.

Qusay Shihab Hamad received his bachelor and master degrees in Computer Engineering from University of Technology, Baghdad Iraq in 2011 and 2014, respectively. Currently, he is lecturer at University of Information Technology and Communications, Baghdad Iraq. Research interest: Engineering Optimization, Computational Intelligence, Embedded System.

Dr. Ahmed Majid Taha was born in Baghdad state, Iraq. He received his Bachelors in software engineering from Al-Rafidain University College, Iraq in 2007. Later he starts his master degree in computer science in National University of Malaysia (UKM), and graduated in 2011. Then he directly began his research study to pursue a PhD degree in Artificial Intelligence in University Tenaga Nasional (UNITEN) in 2014. His research interest includes data mining, metaheuristics, algorithms and machine learning. He is currently a senior lecturer at the Businesses Informatics College, University of Information Technology and Communication, Baghdad, He is also member of Soft Computing and Data Mining Center, University Tun Hussein Onn Malaysia.