Scientific Research Publishing

Journal of Computer and Communications, 2016, 4, 1-11
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Parallel Quick Search Algorithm for the Exact String Matching Problem Using OpenMP

Sinan Sameer Mahmood Al-Dabbagh, Nawaf Hazim Barnouti, Mustafa Abdul Sahib Naser, Zaid G. Ali

Software Engineering and Information Technology Department, Al-Mansour University College, Baghdad, Iraq
Email: Sinan.aldabbagh815@gmail.com

How to cite this paper: Al-Dabbagh, S.S.M., Barnouti, N.H., Naser, M.A.S. and Ali, Z.G. (2016) Parallel Quick Search Algorithm for the Exact String Matching Problem Using OpenMP. *Journal of Computer and Communications*, **4**, 1-11.
http://dx.doi.org/10.4236/jcc.2016.413001

Received: August 15, 2016
Accepted: October 15, 2016
Published: October 18, 2016

Abstract

String matching is seen as one of the essential problems in computer science. A variety of computer applications provide the string matching service for their end users. The remarkable boost in the number of data that is created and kept by modern computational devices influences researchers to obtain even more powerful methods for coping with this problem. In this research, the Quick Search string matching algorithm are adopted to be implemented under the multi-core environment using OpenMP directive which can be employed to reduce the overall execution time of the program. English text, Proteins and DNA data types are utilized to examine the effect of parallelization and implementation of Quick Search string matching algorithm on multi-core based environment. Experimental outcomes reveal that the overall performance of the mentioned string matching algorithm has been improved, and the improvement in the execution time which has been obtained is considerable enough to recommend the multi-core environment as the suitable platform for parallelizing the Quick Search string matching algorithm.

Keywords

String Matching, Pattern Matching, String Searching, Algorithms, Quick Search Algorithm, Exact String Matching Algorithm, Parallelization, OpenMP

1. Introduction

String matching algorithms are an important class of string algorithms that try to find a place where one or several strings (also called patterns) are found within a larger string or text. The fundamental string matching problem is defined as follows: given two strings a text and a pattern, determine whether the pattern appears in the text [1]. String matching algorithms are applied in many computer applications, such as data processing, image and voice recognition, information retrieval, computational biology

and chemistry [2]. Furthermore, string matching algorithms have become a significant component of applications which are used to search nucleotide or amino acid sequence patterns in biological sequence databases in recent years [3]. Therefore, the performance of the string matching algorithms plays a prominent role in the performance of these computer applications [4]. This research concentrates on the problems which are related to the performance of the Quick Search string matching algorithm. Therefore, the main question is "How to reduce execution time of the Quick Search string matching algorithm by using OpenMP parallel method?" Therefore, the sub question of the main question is "How to prove the performance improvement of the parallel version of the Quick Search string matching algorithm compared with its performance of the sequential version of the Quick Search string matching algorithm?" Therefore, the objective of this paper is to investigate the suitability of parallelizing the Quick Search algorithm on multi-core environment using OpenMP.

2. Related Work

2.1. Parallel Processing

Parallel Processing is defined as the efforts of multiple concurrent processing units that works together to resolve computational problems [5]. The fundamental idea of the parallel programing is to divide the task into sub-task which can be solved simultaneously on multiple Central Processing Units (CPU's), each sub-task of the program is sub divided into several of instructions and just one program of instructions to be carried out at any particular moment in time [6].

2.2. Parallel String Matching Algorithms

Parallel computation holds outstanding potential of enhancing the processing and execution times of data in comparison with sequential computation which probably takes a lot of valuable time to show results. At first, generally there are many numerous parallel string matching algorithms which have been produced every single one with the intention of accelerating the overall performance of the algorithms and preserving time via the application of multi-processors. OpenMP directives is used to parallelize the string matching algorithms in a multi-core CPU environment which has broad attraction several realms of computer science; one example of these fields is the security applications, in [7] the potential for improving the speed of Intrusion Detection System (IDS) is mentioned, which is a system use to detect the hacker that try to hack the network and report this act of sabotage to the network administrator. The OpenMP directives and Pthread API which are Parallelization methods are used to speed up the Quick Search algorithm and to test the proposed method, which was dependent on analyzing several factors—such as length of pattern and size of dataset—to select the number of threads for parallel execution.

Parallel string matching algorithms have also an astonishing position in biological applications. Therefore, in [8] the author introduces a hybrid OpenMP/MPI parallel model by utilizing the benefits of shared and distributed memory technologies to the

parallel three types of string matching algorithms. As a result, they were very capable of obtain optimum results with specific different types of biological databases in their proposed model. Additionally, in [9] the same author presents a different research indicated that the technique of data partitioning as well as the type of data are extremely essential factors that control the parallelization efficiency.

3. The Proposed Method

This section includes detailed explanations around the important features along with the behavior examination of Quick Search algorithm. The key reason of examining the behavior of the sequential Quick Search algorithm which involve the preprocessing phase as well as searching phase is to find out the compute-intensive portions of the code, that could be parallelize using OpenMP.

3.1. Sequential Quick Search Algorithm

Quick Search algorithm is a simplified version of Boyer Moore algorithm solves the string matching problem. In general, the Quick Search algorithm composes from two logical phases, pre-processing and searching phase [10]. The preprocessing and searching phases of the Quick Search algorithm, are summarized in the next subsections, as shown in **Figure 1**.

3.1.1. Pre-Processing Phase

The main idea behind the preprocessing phase of the Quick Search algorithm is to collect information about the pattern which known in advance, and use this information during the searching phase. The pattern needs to be skipped a specific amount of characters whenever a match or a mismatch is taking place during the searching process. The Quick Search algorithm use a particular structure known as a bad character table (*qsBc*) carries the shift information.

Starting with the rightmost character of the pattern, each character placement (*i*) subtracts from the value of pattern length (*m*) and stores in the (*qsBc*) table. In case there is duplicating the same character several times in the pattern the first rightmost occurrence for every character that takes place in the pattern is stored in (*qsBc*). According to the equation providing below, the (*qsBc*) table stores the minimum value of the differences between pattern length *m* and the rightmost locations of each repeated character in that pattern.

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

3.1.2. Searching Phase

In this phase, the Quick Search algorithm beginning the matching process from the leftmost character of the pattern with its corresponding character in the text window. If a match or mismatch occur the pattern shift to the right side depending on the value stored in (*qsBc*) table of the character positioned after the rightmost character of the text window, if the character that immediately follow the rightmost character in the text

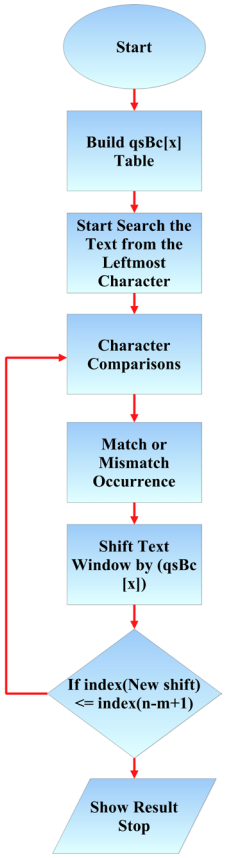

Figure 1. Quick search algorithm overview flowchart.

window is takes place in the pattern, the pattern shifts to align its own character with the character that located immediately after the rightmost character of the text window. However, in the case the character that positioned after the text window is not occurred in the pattern, the whole pattern shifts to the right side of the character that follow the rightmost character of the text window, and start a new matching process.

3.2. Parallel Quick Search Algorithm Evaluation

This section discusses the main objective of this study, which is the parallelization method of Quick Search algorithm. The Quick Search algorithm is implemented on a multi-core environment platform. The OpenMP library programming interface is used to implement the code.

According to the analysis of the sequential Quick search algorithm in previous section, the most expensive section of a string matching algorithm is to examine if the character of the pattern matches the character of the text window [11]. To avoid this cost the searching phase which contains the matching process between the characters of the pattern and the text window will parallelize using OpenMP directive.

The searching phase in the Quick Search string matching algorithm is carried out using multi-core environments platform, as well as the OpenMP which is the programming environments. The OpenMP platform executed the program by divided the

entire input data into subdivided parts through fork and join operations, the master thread distributed the works to the worker threads. The parallel Quick Search algorithm start execution the program in sequential fashion conducted by the master thread until the algorithm reach the searching phase function, at this moment slave threads generated for searching phase function, the number of threads is seven because our experiment was conducted using laptop with Core™ with 7 cores and 8 GB RAM The operating system used is Microsoft Windows 8.1. The slave threads executing the searching phase functions and return the partial result to the master thread, the master thread will assemble all the result with the help of join operation and show the output, this operation performed in sequential fashion, the slave threads will terminate itself automatically after send the results to the master thread as shown in Figure 2.

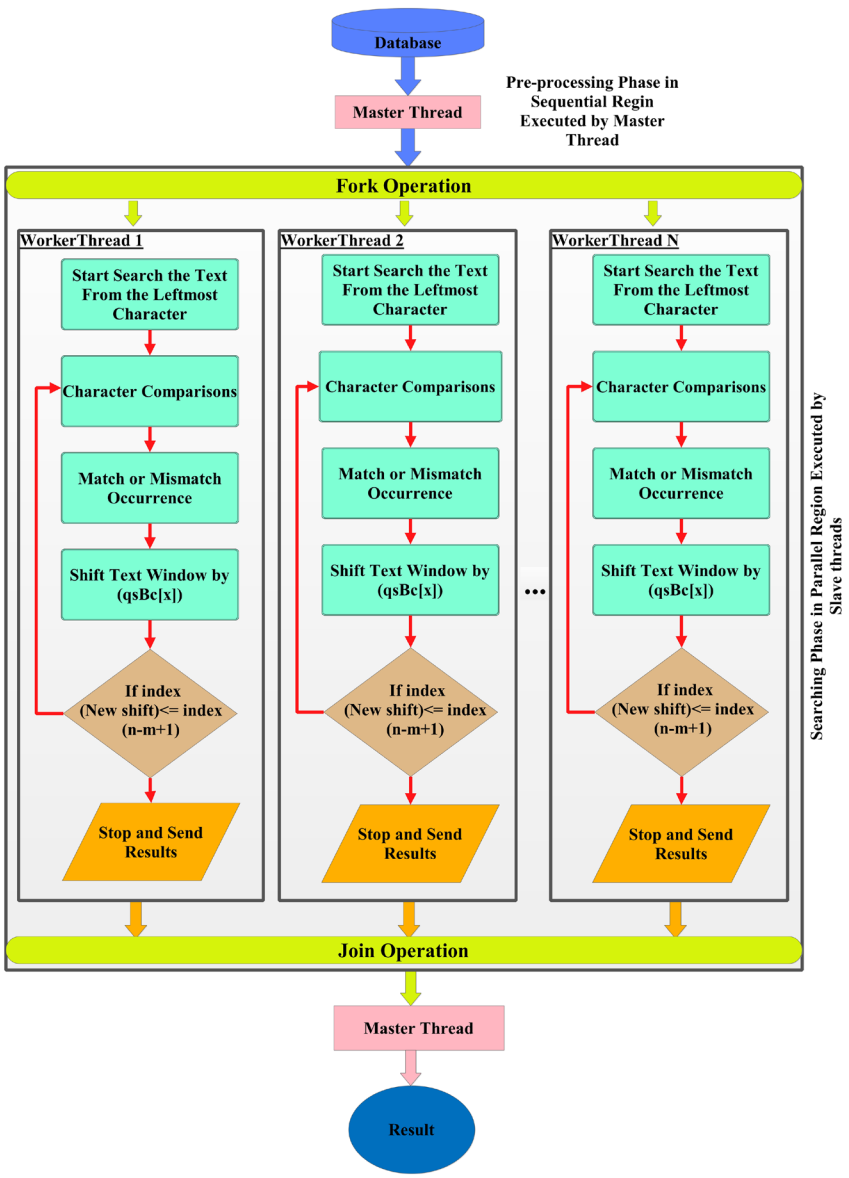

Figure 2. The proposed searching phase of the quick search algorithm.

4. Experimental Results and Discussion

The main idea behind parallelization the Quick Search algorithm is to enhance its performance, to measure the improvement in the performance of parallel Quick Search algorithm over its sequential version there is the execution time factor to evaluate the performance gain. In order to examine the performance of parallel algorithm, a standard benchmark data is used which is represented the common used of string matching algorithm, which are English text, Proteins sequence and DNA sequence. These different data types that have been downloaded from (http://pizzachili.dcc.uchile.cl/texts.html) are differences in the size of alphabets, as a way to analyze the algorithm behaviors with various alphabet sizes. The sequential and parallel program of Quick search algorithm was run with data size 200 MB. Moreover, various pattern lengths were used to assess the behaviors of the algorithm. These lengths are: 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 characters that are chosen randomly from words inside the text, the sequential and parallel program executed 5 times and the average time for all the attempts is selected [12].

4.1. Parallel Performance Evaluation

4.1.1. English Text Data Type

The execution time of the sequential and parallel Quick Search algorithm using English text data type which compose of more than 100 different alphabet types is shown in Table 1 and Table 2 respectively.

Figure 3 shows the execution time (average time) of the sequential and parallel of Quick Search algorithm using English text data type. The Quick Search algorithm show unstable behavior when compared to the proteins and DNA data types, this is due to the size of the alphabet used where the English text it consist more than 100 characters, which considered a large alphabet. The unstable behavior appear clearly in the pattern length 40 and 60, which gives the worst time and best time respectively. The execution time of parallel program show better performance compare to the execution time of sequential program.

Table 1. Sequential performance using English text data type.

Pattern length	Sequential quick search algorithm					
	Shot 1	Shot 2	Shot 3	Shot 4	Shot 5	Average
10	15.236	20.805	15.168	14.277	13.62	15.8212
20	14.559	15.752	13.245	14.366	14.346	14.4536
30	21.867	18.094	12.739	12.726	12.539	15.593
40	14.297	15.017	13.658	14.879	15.449	14.66
50	14.423	14.927	14.14	13.425	14.024	14.1878
60	14.811	15.481	15.031	16.772	14.312	15.2814
70	15.663	14.746	15.455	15.966	14.892	15.3444
80	17.375	13.512	14.688	15.632	16.277	15.4968
90	16.637	15.446	15.307	15.129	15.603	15.6244
100	14.59	13.816	20.276	13.954	16.099	15.747

Table 2. Parallel performance using English text data type.

Pattern length	Parallel quick search algorithm					
	Shot 1	Shot 2	Shot 3	Shot 4	Shot 5	Average
10	6.27	10.077	6.501	7.328	5.362	7.1076
20	11.043	8.527	5.293	5.612	5.673	7.2296
30	8.039	8.289	8.116	7.949	8.015	8.0816
40	6.237	5.606	23.524	13.493	4.37	10.646
50	8.855	7.868	7.741	7.653	7.672	7.9578
60	8.253	5.317	5.754	7.69	5.007	6.4042
70	8.973	8.681	7.625	8.555	8.636	8.494
80	9.918	8.388	6.596	14.011	8.792	9.541
90	13.921	9.107	8.843	9.552	6.77	9.6386
100	19.627	7.258	4.909	6.495	6.746	9.007

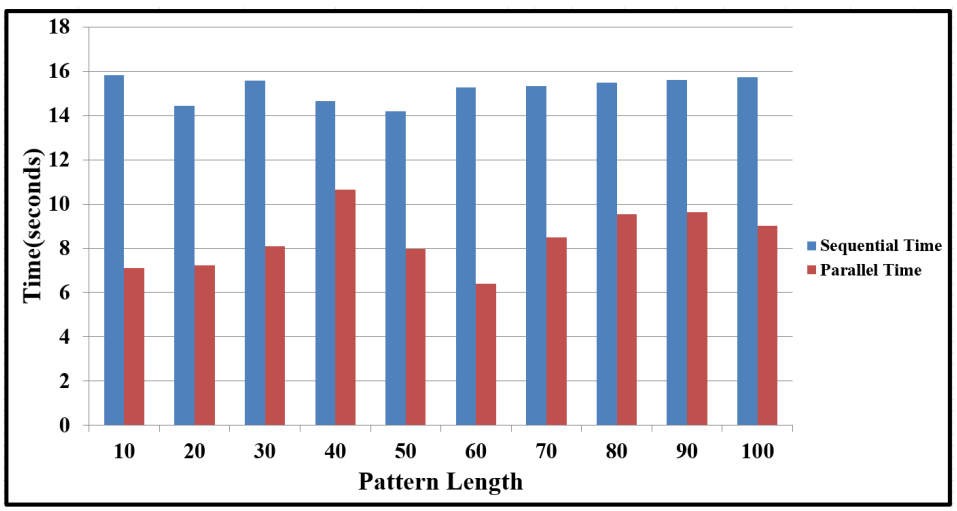

Figure 3. Execution time using English text data type.

4.1.2. Protein Sequence Data Type

The execution time of the sequential and parallel Quick Search algorithm using Protein sequence data type which compose of 20 amino acids is shown in **Table 3** and **Table 4** respectively.

Figure 4 show the execution time (average time) of the sequential and parallel of Quick Search algorithm using Proteins sequence data type. The Quick Search algorithm show stable behavior when compared to the English text and DNA data types, this is due to the size of the alphabet used where the Proteins sequence data type it consist with 20 characters, which considered a medium alphabet. The execution time of parallel program show better performance compare to the execution time of sequential program.

Table 3. Sequential performance using protein sequence data type.

Pattern length	Sequential quick search algorithm					
	Shot 1	Shot 2	Shot 3	Shot 4	Shot 5	Average
10	12.354	12.987	13.115	13.208	12.993	12.9314
20	12.453	12.459	12.541	12.47	13.036	12.5918
30	13.499	12.302	12.374	13.099	13.206	12.896
40	12.259	12.141	12.122	11.971	11.713	12.0412
50	12.378	12.37	12.026	12.099	12.39	12.2526
60	12.025	12.362	11.886	11.696	11.706	11.935
70	12.451	12.23	12.045	12.051	12.048	12.165
80	12.491	12.469	11.81	11.554	11.801	12.025
90	12.392	12.221	12.064	12.003	12.246	12.1852
100	11.546	11.803	11.512	11.509	11.578	11.5896

Table 4. Parallel performance using protein sequence data type.

Pattern length	Parallel quick search algorithm					
	Shot 1	Shot 2	Shot 3	Shot 4	Shot 5	Average
10	3.899	3.523	3.574	3.473	3.463	3.5864
20	3.842	3.474	3.413	3.063	3.069	3.3722
30	3.901	4.152	3.811	6.121	4.018	4.4006
40	3.531	3.373	3.212	3.147	3.338	3.3202
50	4.321	3.607	3.261	3.673	3.754	3.7232
60	3.7	3.697	3.425	3.367	3.448	3.5274
70	3.947	3.306	3.285	3.703	3.613	3.5708
80	3.427	3.288	3.182	3.627	3.158	3.3364
90	3.529	3.234	3.589	3.359	3.175	3.3772
100	3.283	3.31	3.129	3.159	3.619	3.3

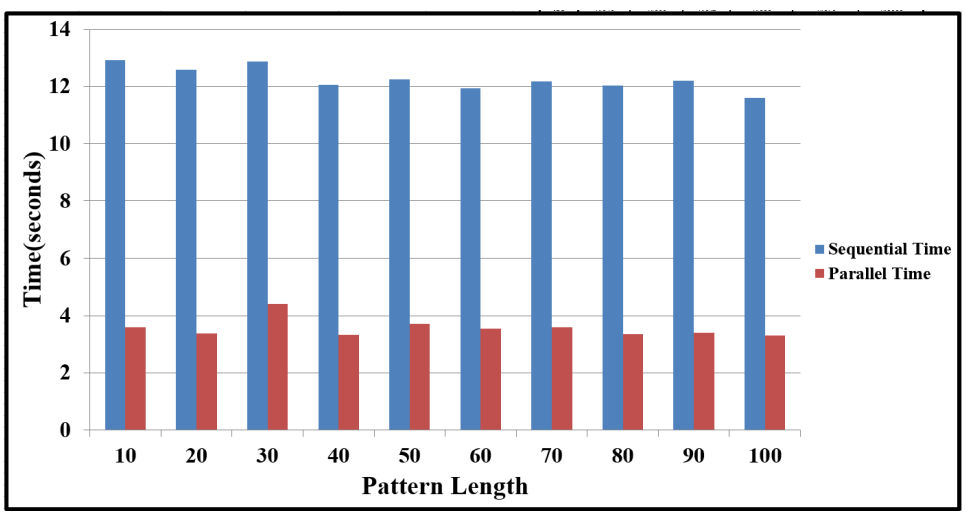

Figure 4. Execution time using protein sequence data type.

4.1.3. DNA Sequence Data Type

The execution time of the sequential and parallel Quick Search algorithm using DNA sequence data type which compose of 4 characters that indicate the chemical foundations of the cell nucleus is shown in Table 5 and Table 6 respectively.

Figure 5 shows the execution time (average time) of the sequential and parallel of Quick Search algorithm using DNA sequence data type. The Quick Search algorithm show stable behavior when compared to the English text and Proteins sequence data types, this is due to the size of the alphabet used where the DNA sequence data type it consist only 4 characters, which considered a small alphabet. The execution time of parallel program show better performance compare to the execution time of sequential program.

Table 5. Sequential performance using DNA sequence data type.

Pattern length	Sequential quick search algorithm					
	Shot 1	Shot 2	Shot 3	Shot 4	Shot 5	Average
10	15.541	15.524	15.22	15.696	15.592	15.5146
20	14.687	15.426	14.752	14.342	13.736	14.5886
30	13.055	13.36	16.34	13.336	13.022	13.8226
40	13.997	13.9	13.904	14.132	13.794	13.9454
50	13.791	13.479	13.277	13.173	13.667	13.4774
60	14.163	13.872	13.735	13.602	13.909	13.8562
70	13.671	13.814	13.311	13.14	13.492	13.4856
80	13.533	13.639	13.341	13.427	13.73	13.534
90	13.947	13.702	13.639	13.662	13.844	13.7588
100	14.814	14.901	14.695	14.501	14.573	14.6968

Table 6. Parallel performance using DNA sequence data type.

Pattern length	Parallel quick search algorithm					
	Shot 1	Shot 2	Shot 3	Shot 4	Shot 5	Average
10	5.664	6.667	5.807	5.904	5.632	5.9348
20	5.319	5.072	5.392	5.731	5.903	5.4834
30	4.402	4.48	4.005	3.98	4.042	4.1818
40	5.01	4.896	4.806	5.008	5.046	4.9532
50	4.954	4.892	4.476	4.465	4.596	4.6766
60	5.462	5.422	5.253	4.862	4.86	5.1718
70	4.923	4.692	4.892	4.733	4.67	4.782
80	4.623	4.933	4.715	4.615	4.72	4.7212
90	5.41	5.178	5.122	5.153	5.198	5.2122
100	6.088	6.319	5.984	5.783	6.079	6.0506

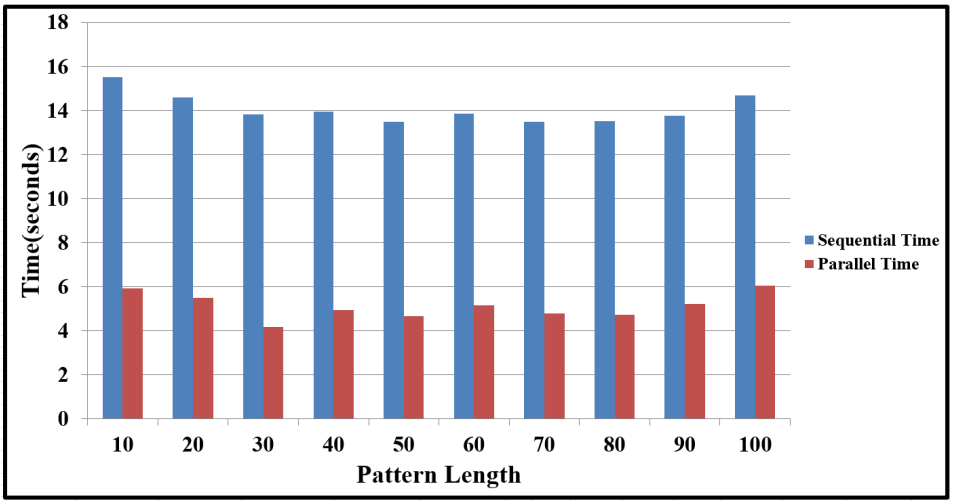

Figure 5. Execution time using DNA sequence data type.

5. Conclusion

This study aims to parallelize the Quick Search exact string matching algorithm. Based on the design presented in Section 3, the parallelization method produced a parallel Quick Search algorithm using OpenMP directive. From the results in Section 4, we can note that when parallelizing the Quick Search algorithm by using OpenMP directive under multi-core environment, the parallel program shows better performance compared to the sequential program in terms of execution time when using different data types with different patterns length. In addition, the experimental results show that when using English text data type the Quick Search algorithm gives unstable results due to the size of the alphabet which is considered a large alphabet, but it gives a stable result when using medium and small alphabet as proteins and DNA data types. As a conclusion, we recommend the multi-core environment as the suitable platform for parallelizing the Quick Search string matching algorithm. For future work the parallel Quick Search algorithm could be enhanced by parallelizing the preprocessing phase with the searching phase.

References

[1] Faro, S. and Külekci, O. (2015) Efficient Algorithms for the Order Preserving Pattern Matching Problem. arXiv:1501.04001.

[2] Faro, S. and Lecroq, T. (2013) The Exact Online String Matching Problem: A Review of the Most Recent Results. *ACM Computing Surveys* (*CSUR*), **45**, 2. http://dx.doi.org/10.1145/2431211.2431212

[3] Navarro, G. (2011) A Guided Tour to Approximate String Matching. *ACM Computing Surveys* (*CSUR*), **33**, 31-88. http://dx.doi.org/10.1145/375360.375365

[4] Raju, S.V., Babu, A.V. and Mrudula, M. (2006) Backend Engine for Parallel String Matching Using Boolean Matrix. *International Symposium on Parallel Computing in Electrical Engineering*, 13-17 September 2006. http://dx.doi.org/10.1109/PARELEC.2006.20

[5] Zha, X.Y. and Sahni, S. (2013) GPU-to-GPU and Host-to-Host Multipattern String Matching on a GPU. *IEEE Transactions on Computers*, **62**, 1156-1169.

http://dx.doi.org/10.1109/TC.2012.61

[6] Buyya, R. (2000) The Design of PARAS Microkernel. [Online]. http://www.cloudbus.org/ raj/microkernel/

[7] Hnaif, A.A., *et al*. (2008) Parallel Quick Search Algorithm to Speed Packet Payload Filtering in NIDS. *Executive Development*, **21**, 22.

[8] Kouzinopoulos, C.S., Margaritis, K.G. and Michailidis, P.D. (2011) Parallel Processing of Multiple Pattern Matching Algorithms for Biological Sequences: Methods and Performance Results. INTECH Open Access Publisher.

[9] Kouzinopoulos, C. and Margaritis, K. (2009) Parallel Implementation of Exact Two Dimensional Pattern Matching Algorithms Using MPI and OpenMP. 9*th Hellenic European Research on Computer Mathematics and Its Applications Conference*.

[10] Sunday, D.M. (1990) A Very Fast Substring Search Algorithm. *Communications of the ACM*, **33**, 132-142. http://dx.doi.org/10.1145/79173.79184

[11] Charras, C. and Lecroq, T. (2004) Handbook of Exact String Matching Algorithms. King's College.

[12] Kadhim, H.A. and Abdul Rashidx, N.A. (2014) Maximum-Shift String Matching Algorithms. *International Conference on Computer and Information Sciences* (ICCOINS).