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

A Parallel BMH String Matching Algorithm Based on OpenMP

Xiao Qi
College of Information Engineering
Northwest A&F University
Yangling, China
qixiao0218@nwafu.edu.cn

Bin Liu*
College of Information Engineering
Northwest A&F University
Yangling, China
liubin0929@nwsuaf.edu.cn

Yuancheng Li
School of Computer Science and Technology
Xi'an University of Science and Technology
Xi'an, China
yuanch_li@126.com

Yanning Du
School of Computer Science & Engineering
Xi'an University of Technology
Xi'an, China
duyanning@gmail.com

Yuxiang Li
School of Information Technology
Henan University of Science and Technology
Luoyang, China
liyuxiang@haust.edu.cn

Dangdang Niu
College of Information Engineering
Northwest A&F University
Yangling, China
niudd@nwafu.edu.cn

***Abstract*—BMH（Boyer-Moore-Horspool) string matching algorithms have played an important role in the field of biological sequence alignment, text processing, spell checking, and computer virus signature matching. However, with the explosive growth of the data, the serial string matching algorithm is too slow to finish the matching task within an acceptable time. This paper proposed a parallel BMH string matching algorithm based on OpenMP. The parallelism of the BMH algorithm is first identified by theoretical analysis. Then, the sequential algorithm is designed as a parallel algorithm in a data-parallel form and the larger string matching task is divided into multiple sub-string matching tasks by partitioning strategy. Furthermore, using the thread-level parallel technique, each thread carries out string matching operations on different sub-string blocks in parallel. Compared with the serial BMH algorithm, the parallel BMH matching algorithm with 8 threads can achieve up to 3.53 times speedup. On the OpenMP platform, experimental results show that the proposed parallel algorithm can acquire a significant increase in speedup. Under the accuracy of ensuring the string matching, the optimal number of threads and the most optimal block segmentation scheme are obtained through experimental tests. It indicates that the OpenMP-based parallel BMH algorithm has excellent acceleration performance, and an advantageous application prospect in various field.**

***Index Terms*—Parallel algorithm, OpenMP, Boyer-Moore-Horspool algorithm, string matching**

I. INTRODUCTION

The string matching algorithm is applied to find the specified pattern string in a larger string. The study of string matching algorithms has always been at the core of computer science. Because the string matching algorithm plays an important role in computer theory and algorithms and the scope of application is universal, researchers always pay attention to the research of this algorithm. String matching algorithms are at the core of text processing, spell checking, DNA sequence matching, search engines and computer virus signature matching [1-6].

*Corresponding author: Bin Liu
E-mail: liubin0929@nwsuaf.edu.cn.

With the rapid development of computer science and technology, the data size to be processed is increasing significantly [7-8]. The serial string matching algorithm has been unable to meet the increasing data volume and match the demand in the current science field. At present, how to accurately match large-scale strings faces serious challenges. With the rapid development of the parallel computer system, it provides a new alternative optimize algorithm and improves the matching efficiency. Using parallel technology can meet the needs of string matching more easily and quickly. In recent years, the development of computer science and technology in the world has focused on multi-threading technology, which will become a widely used computing model [9-11]. Therefore, it is of great scientific and practical significance to research the parallelization of the string matching algorithm and design the matching algorithm based on the multi-core processor. Researchers have parallelized multiple string matching algorithms such as KMP, BMH Brute-force and Quick-Search on a GPU [12-18]. However, since fine-grained parallelism encounters a bottleneck for exploiting the parallelism of these algorithms, which have lots of coarse-grained parallelisms, researchers are turning their attention to multicore architectures and coarser-grained parallelism. Thus, this paper proposes a parallel BMH string matching algorithm based on OpenMP. A new idea is provided for the parallelization of string matching algorithms, which will be widely used in the research fields of text processing, spell check, DNA sequence matching, search engine and computer virus signature matching.

In order to solve the problem of string matching slowly, this paper proposes a parallel BMH string matching algorithm based on OpenMP, and the contribution of this paper is as follows:

- The sequential BMH string matching algorithm proposed in this paper is ported to the multi-core platform and obtain up to 3.53 times speedup under 8 threads.
- A novel parallel BMH algorithm based on OpenMP is proposed. An optimal partitioning strategy is first used to divide the larger string into multiple sub-string blocks based on existing threads. Then, using the thread-level parallel technique, different sub-string blocks are assigned to the

978-1-7281-2058-4/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCC/SmartCity/DSS.2019.00026

existing idle threads, and each thread performs a matching operation in parallel.

- A mutual exclusion mechanism is proposed for ensuring the correctness of the multithreaded program. The shared variable in this paper is used to store all the time that a pattern string in all sub-string matching when multiple threads perform string matching tasks simultaneously. Atomic operations are used to ensure that no conflicts occur when competing to write to the shared variable.

However, with the number of threads increases, the time consumed by the thread opening and reclaiming operations is greater than the time saved by the parallel matching algorithm.

To demonstrate the validity of the proposed parallel BMH string matching algorithm based on OpenMP, benchmarks are executed on the multi-core platform. The experimental results show the correctness of the parallel scheme and the maximum speedup can be achieved under the influence of various factors. Under the accuracy of ensuring the string matching operation, it indicates that the parallel BMH algorithm has excellent speedup performance. Compared to the serial BMH algorithm, the parallel algorithm can achieve a speedup of up to 3.53 times.

The remainder of this paper is organized as follows. In Section II, the BMH serial algorithm and OpenMP parallel framework are first briefly described. In Section III, parallel design and implementation of the BMH algorithm are presented. Section IV analyzes experimental results. In Section V, related work is introduced and summarized. Finally, we conclude this paper in Section VI.

II. PRELIMINARY

A. Boyer-Moore-Horspool String Matching Algorithm

In 1977, R. Boyer and S. Moore proposed a classic precision string matching algorithm--the Boyer-Moore algorithm [19]. The BMH algorithm is a simplified implementation of the BM algorithm, which was proposed by R. Nigel Horspool in 1980 [20-21]. The BMH algorithm is a suffix-based matching algorithm and a "jumping" precision string matching algorithm with sub-linear time complexity. Unlike the basic BM algorithm, the BMH algorithm uses only one heuristic rule, which facilitates the initialization and preprocessing process, eliminating the need for comparing with good suffix preprocessing steps [20].

The BMH algorithm outperforms the basic BM algorithm in most cases. The BMH algorithm logic is shown in Fig. 1, and the BMH algorithm only considers bad character heuristic rules in the process of moving the pattern string. The bad character rule specifically refers to the process of scanning from right to left. If a character is found to be mismatched, it is discussed as follows: If the character does not appear in the pattern string, then the texts starting from the character are obviously unlikely to match the pattern string successfully. In this case, skip directly. If the character appears in the pattern string, it is aligned with that character. The BMH algorithm, which uses only one rule, exhibits better performance in general than the BM algorithm. The BMH algorithm simplifies the initialization

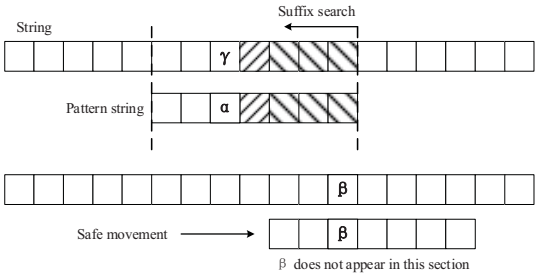

Figure 1. BMH algorithm matching procedure

process of the BM algorithm, saves the moving distance that needs to be initiated in the calculation of the suffix, and also saves the process of comparing bad characters and good suffixes.

B. OpenMP Parallel Framework

Shared memory is a large amount of memory that can be accessed by different central processors in a multi-core computer system. In the shared memory system, multiple processes can simultaneously access the contents of the same block of memory while the program is running [22-25]. The shared memory structure is shown in Fig. 2.

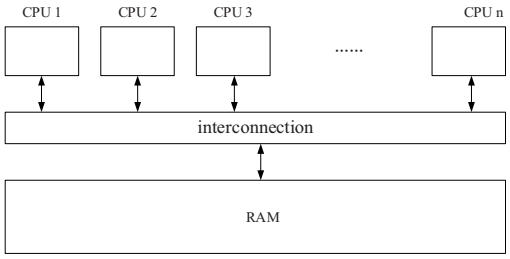

Figure 2. Shared memory structure diagram

OpenMP is a set of Application Program Interfaces that can be used to explicitly guide multithreading and parallelization of shared memory programs. As a Guided Compiler Directive, OpenMP can be used to design parallel programs based on shared memory multiprocessor systems. OpenMP is an industry standard that is very easy to use and provides "incremental parallelism", starting with serial code [26]. In OpenMP, multiple processing units in the CPU share the same memory device, and all CPU processing units need to access the same memory addressing space. Communication and synchronization operations can be performed by multiple processing units in the shared memory variable CPU. OpenMP parallel framework development is simple, highly abstract, versatile, scalable and portable, and supports parallel incremental development and data parallelism. The OpenMP parallel execution model uses the Fork-Join form [23], which is based on a multi-thread mode that is indicated by compiler instructions to indicate the parallel portion of the current program. The OpenMP parallel execution mode is shown in Fig. 3. Initially, only the main thread in the program executes the current code serially. After encountering a parallel region, the code blocks in the parallel region are executed simultaneously in multiple threads.

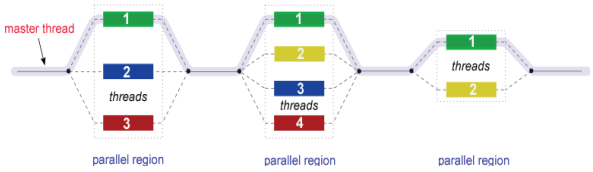

Figure 3. OpenMP parallel execution mode

III.PARALLEL DESIGN AND IMPLEMENTION OF BMH ALGORITHM

A. Parallelism Analyses

Parallelism means that a computer system can execute two or more computing tasks simultaneously. The main idea of parallel analyses is based on data segmentation, task parallelism and reduction summation. Considering the practical application process of the BMH matching algorithm, the data size of the string is extremely large. For this reason, the idea of segmenting the matched text data is adopted, and the string is distributed to each thread as evenly as possible. The area of the task parallelism and the reduction summation region are marked in the BMH algorithm flow chart.

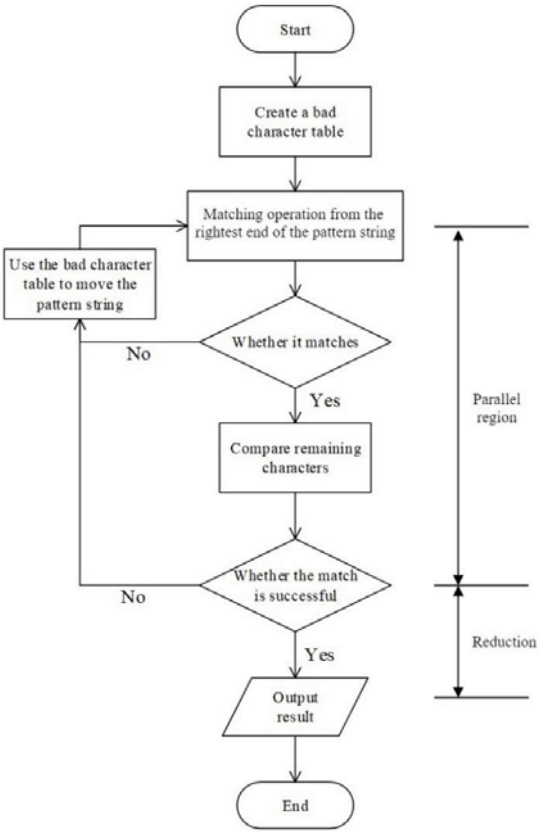

Figure 4. Parallel analyses of BMH algorithm

Parallel Analyses of the BMH Algorithm are shown in Fig. 4. As can be seen from the flow chart, there is parallelism in the process from the start of the matching algorithm to the end of the matching operation. A data partitioning operation can be performed for the long matched string, and then the string matching tasks are assigned to multiple threads to perform matching operations in parallel. After the parallel matching operation of multiple threads ends, it is necessary to count the number of matches in all threads to get all the matching times.

B. Parallel Design of BMH Algorithm

The OpenMP parallelization scheme defines two shared variables for the main thread, representing the string and the pattern string, and then reading the string and pattern strings into memory from the file and command line. When the main thread runs to the preprocessing sections, it enters the parallel region of the string matching algorithm. The initialization operation of the bad character offset table is simultaneously performed in multiple threads in the parallel region. Then, multiple threads created by the main thread encounter an implicit synchronization operation and the created thread merges with the main thread to continue execution. The main thread continues to run to the main matching algorithm, using the partitioning strategy for the next step. The string is divided into several sub-blocks according to the number of threads, and index of each thread to ensure that every two threads will not select the same block. Parallel block partition is shown in Fig. 5.

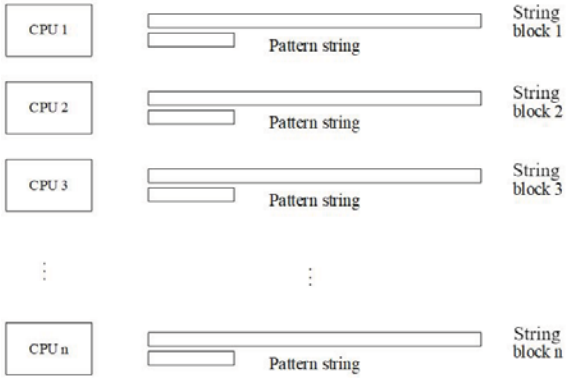

Figure 5. Parallel block partition

At the same time, in order to avoid the existence of the pattern string at the segmentation point of the string to be matched, the partially parallelizable string may be missed during the parallelization execution of the segmentation block. A common offset should be set for each partition block. The offset is determined by the size of the pattern string, and the offset size is set to the pattern string length minus one. The block offset setting is shown as Fig 6.

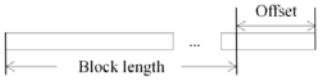

Figure 6. Block offset setting

In the process of dividing a block, after the last sliced block is allocated to the last thread that executes the block, a piece of string smaller than the block length is not processed by any

thread. When mapping to the last thread, the last remaining piece of string that is not processed by any thread is handed over to the main thread for matching. The schematic diagram is shown as Fig 7.

Figure 7. Remaining string processing

When parallelizing all the string splits, each thread needs to set a private variable to count the number of times that a string matches in a thread, and finally the matched number of times is accumulated. The solution is to use the shared variable as the sum of all matches, and each thread can accumulate the calculated partial results into the shared variable. But when two threads execute the accumulated statement at the same time, such an operation may cause an unpredictable error. Therefore, some mechanism needs to be introduced to ensure that only one thread performs the accumulating operation of variables at a time, and other threads cannot start executing this code during the current thread accumulation. Mutually exclusive access critical region among threads is shown as Fig 8. The shared variable is used to calculate all the time of the pattern string in all string matches. Use critical region to ensure that changes to shared variables run in serial mode and inter-thread must be accessed exclusively. Forcing threads to access a block of data in a mutually exclusive manner causes the originally parallelized algorithm to be executed sequentially. Although this operation will affect the parallel speedup performance, this operation is necessary to ensure the correctness of the data.

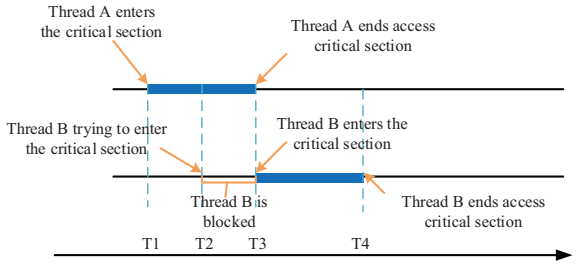

Figure 8. Mutually exclusive access critical region among threads

C. Parallel Implementation

Algorithm 1 shows the parallel implementation of the BMH algorithm. The input of the parallel algorithm includes text (representing the string to be matched) and patterns (representing the target pattern string). The output of the parallel algorithm is the sum of all matched times.

The number of threads that need to be started for the current parallel is defined by the macro. Line 1-2 in the algorithm, the current operating system starts a single-threaded process, and the process executes the code in the main function. The string from the file and the pattern string received from the command line are stored. Line 3 in the algorithm, using the parallel directive indicates that the following structure block of code should be executed by multiple threads in parallel. During the matching execution, the entire matching process is included in the parallel block of OpenMP. A thread clause after the parallel instruction is added to specify the number of threads in the code block after execution. Two private variables are set in each thread to store the number of matching pattern strings in the current thread and the currently running thread number.

Line 5-13 in the algorithm, the current thread is the main thread, after matching the allocated character block, it continues to match the last remaining string and accumulates the matched times. Line 14-20 in the algorithm, the current thread isn't main thread, the matched number of times is matched after the matched character block is matched. Variable update operations that ultimately store the sum of matches in each thread also need to be protected to avoid erroneous update operations in parallel. Line 21 in the algorithm uses the atomic instruction, the modification to the total number of matches is set to the critical region, and only one thread is allowed to modify it at a time.

Algorithm 1 - The parallel BMH string matching algorithm

Input: text(Text is the string to be matched), patterns(patterns is the pattern string in the string matching)

Output: sum count(Sum count is the total number of matches)

```
1  block_size=text/threads, last=text%threads, offset=patterns-1
2  sum_count=0,id_threads=omp_get_threads_num()
3  #pragma omp parallel num_threads(threads)
4          local_count = 0
5        if id_threads = 0 then
6          matches the string assigned by the current thread
7          if match patterns then
8              local_count++
9          end if
10         match the last remaining characters (length is last)
11         if match patterns then
12             local_count++
13         end if
14       else
15         while not reached character end do
16             matches the string assigned by the current thread
17             if match patterns then
18                 local_count++
19             end if
20           end while
21   #pragma omp atomic
22         sum_count += local_count
23   return sum_count
```

IV.EXPERIMENTAL EVALUATION

A. Experimental Setup and Testing Set

This paper has implemented the parallel BMH string matching algorithm using the OpenMP programming model and the proposed parallel algorithm is performed on a multi-core

platform. Table I shows the CPU configuration, memory size and kernel version configuration.

Three types of test data sets that are widely used in the application of string matching algorithms are used in this experiment. The test datasets respectively are the general language--a classic novel composed of English words in the Canterbury Corpus, natural language--random text files from artificial language libraries and the professional data--the gene sequence gene text file in the bioinformatics database, which consists of DNA genomic sequences consisting of tetranucleotide A, C, G, T, adenine, cytosine, guanine and pyrimidine. Date sets are shown in Table II.

TABLE I. Experimental environment

Configuration items	Item value
System version	Ubuntu 16.04.2
Compiler	GCC 5.4.0
Server model	Lenovo SystemX 3650 M5
CPU	Two Xeon E5-2650 v4
GPU	PCI-E interface Two NVIDIA Tesla P100
Memory capacity	16*32 GB RDIMM ECC 2400MHz DDR4
Memory version	4.13.0-38-generic
OpenMP version	OpenMP 3.0

TABLE II. Description of the test set

Source	Category
The Canterbury Corpus	Novel
The Artificial Corpus	100,000 characters, randomly selected from [a-z|A-Z|0-9|!|] (alphabet size 64)
The Large Corpus	Gene sequence

B. Experimental Results and Analyses

1) Speedup Performance

This experiment analyzes that the parallel BMH algorithm has the highest speedup under its thread size. The matching times of three different test sets were tested in testing the running time of the serial BMH algorithm and the parallel BMH algorithm. The 2, 4, 8, 16 and 32 are selected as the number of threads of the parallel algorithm to run these test data sets. The time at which the short mode string and the long mode string are executed is recorded, and the speedup ratio is calculated. The average of ten times was taken as a result of analysis, and the specific information of the experimental results is shown in Table III.

It can be intuitively reflected by Table III: When the test set is the same, as the number of threads increases, the acceleration ratio increases first and then decreases, and when the number of threads is 8, the maximum value is reached. The experiment achieved a maximum speedup of 2.626 when the data set was random and the number of threads was 8.

TABLE III. Optimal number of threads experimental result

Data sets	Threads	Pattern string size (character)		Average speedup
		Speedup(5)	Speedup(64)	
novel	2	1.402	1.146	1.274
	4	1.623	2.005	1.814
	8	**2. 366**	**2.292**	**2.329**
	16	2.443	2.066	2.255
	32	1.167	1.022	1.095
random	2	1.403	1.364	1.384
	4	1.445	1.585	1.515
	8	**2.926**	**2.150**	**2.538**
	16	1.759	1.506	1.633
	32	1.267	1.112	1.190
gene	2	1.225	1.077	1.151
	4	1.301	1.220	1.261
	8	**1.620**	**2.632**	**2.125**
	16	1.225	1.508	1.367
	32	1.112	1.209	1.161

2) Speedup Effect with Pattern String Length

This experiment analyzes which type of pattern string has the highest speedup for the parallel BMH algorithm. The pattern string size considers the short pattern and long pattern. The short pattern includes 2, 4, and 8 characters, and the long pattern includes 32, 64, and 128 characters. During the current test, 8 threads were used for testing. In three different test serial algorithm running time and parallel algorithm running time, the average of ten times is obtained as the result and the speedup is calculated. The results of the acceleration ratio are displayed in a line graph. The trend chart of speedup with different patter string length is shown in Fig. 9.

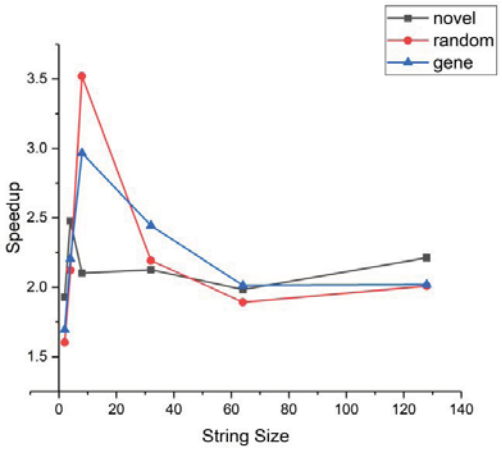

Figure 9. Speedup with pattern string length

As can be seen from the line graph, the current parallel algorithm has different speedups for different scale pattern strings in different test sets. Short pattern string (8) has a higher speedup in the matching process of random, reaching 3.52. Long pattern string (32) has the highest speedup in the gene sequence file, reaching a speedup of 2.444.

3) Speedup Effect with the Partition Block Size

This experiment analyzes that the BMH parallel algorithm has the highest speedup under the segmentation scheme of

different blocks. The basic scheme of block partitioning is based on the current number of processes, and the block length after segmentation is ***text_len/threads***. During the test, the ***text_len*** is divided by 1/4, 1/2, 1 and 2 times the current number of processors, and testing with 8 threads. The serial algorithm running time and parallel algorithm running time of the short pattern string and the long pattern string is tested in three different test sets, and the speedup is calculated. The short pattern string size is 4 characters, and the long pattern string size is 64 characters. The average of ten results is taken as a result and the speedup is calculated. The specific results of the experimental results are as Table IV.

It can be seen from Table IV that different block segmentation schemes have different speedup for different scale pattern strings in different test sets. The scheme of using the current number of threads as a partition block has the best matching efficiency and the highest speedup.

TABLE IV. Block segmentation scheme

Data sets	Segmentation scheme	Pattern string size (character)		Average speedup
		Speedup(4)	Speedup(64)	
novel	1/4	1.797	1.765	1.781
	1/2	2.351	3.020	2.686
	1	**2.456**	**3.168**	**2.812**
	2	1.556	2.403	1.980
random	1/4	2.150	2.013	2.082
	1/2	2.280	2.162	2.221
	1	**2.417**	**2.331**	**2.374**
	2	2.003	1.842	1.923
gene	1/4	2.100	2.881	2.491
	1/2	2.209	3.033	2.621
	1	**2.327**	**3.490**	**2.909**
	2	1.916	2.611	2.264

V.RELATED WORK

A. String Matching Algorithm

Since 1992, the suffix automata technology has been introduced into the string matching algorithm. In the field of computer science, the string matching algorithm with the best average time complexity is presented. The KMP (Knuth-Morris-Pratt) algorithm is one of the most classic algorithms for string matching algorithms. The BM algorithm is 3-5 times more efficient than the KMP algorithm in terms of execution efficiency [19]. The BMH algorithm is a simplified version of the Boyer-Moore algorithm, which improves the bad character rules based on the BM algorithm [20]. The algorithm eliminates the pre-processing steps of better suffixes in the process of initialization and pre-processing. In most cases, the performance is better than the basic BM algorithm. Therefore, the paper studies the BMH algorithm as the basic algorithm for the research of the accurate string matching algorithm. With the rapid development of computer science and technology, the scale of string to be processed is increasing. The matching efficiency of the existing serial string matching algorithms has been difficult to meet the increasing data volume and matching requirements in the current computer science field. In order to solve the problem of the existing string matching slowly, this paper proposes a parallel BMH string matching algorithm based on OpenMP.

B. Parallel String Matching Algorithm

In recent years, researchers have used the architectural features of GPU data parallelism to design the KMP algorithm for GPU operation [12]. Among them, the KMP algorithm using GPU parallelism improves the speedup by about 8 times compared with the serial algorithm in gene sequence matching. Charalampos et al. executed some of the standard search algorithms such as KMP, BMH and Quick-Search on a GPU and proved that the parallel implementation was faster than serial for a small search pattern size [13]. Srinivasa K.G.et al. experiments with Lorem Ipsum text documents with varied parameters show a good speedup for parallel execution of the algorithm against serial execution [14]. Choksuchat C, et al. propose a framework for processing large RDF data sets. It is based on Brute-force string matching on GPUs (BFG), Experiments show that utilizing these algorithms can achieve the speedup of 7 times for querying and for forwarding chaining compared to using the sequential version [15]. Nhat-Phuong Tran presents a multi-stream based parallelization approach for the string matching using the AC algorithm on the latest Nvidia Kepler architecture. Experimental results show that our approach delivers up to 420Gbps throughput performance on Nvidia Tesla K20 GPU [27]. Yoon J, et al. proposes an NFA-based parallel string matching scheme on the GPU. Throughputs can be enhanced with optimized resource allocation. The proposed scheme enhances the throughput of about 13%, compared with previous NFA-based sting matching schemes [28]. However, existing string matching algorithms based GPU have improved speedup performance by fine-grained parallelism, it has hit a bottleneck for exploiting the parallelism of these algorithms. Unlike their research, in this paper, a parallel BMH string matching algorithm based on OpenMP is proposed to exploit coarse-grained parallelism of the BMH string matching algorithm.

VI.CONCLUSION

This paper proposed a parallel BMH string matching algorithm and is implemented on the multi-core platform using the OpenMP programming model. The parallelism of the BMH algorithm is first identified by the expert in this field. Then, the parallel BMH algorithm is designed based on the OpenMP programming model. At last, the parallel BMH algorithm is mapped to the multi-core platform by guide compiler directives for improving its performance and speedup. Compared to the serial BMH algorithm, the proposed parallel algorithm with 8 threads can achieve a speedup of up to 3.53 times. Experimental results show that the proposed parallel algorithm can acquire a significant increase in speed up and can better meet the efficiency requirements of the string matching algorithm in the scientific computing field.

ACKNOWLEDGMENT

We thank our colleagues for their collaboration and the present work. We also thank all the reviewers for their specific comments and suggestions. This research is supported by the National Natural Science Foundation of China under Grant No. 61602388, by the Natural Science Basic Research Plan in Shaanxi Province of China under Grant No. 2017JM6059, by the Fundamental Research Funds for the Central Universities under Grants No. 2452019064 and 2452016081, by the China Postdoctoral Science Foundation under Grant No. 2017M613216, by the Postdoctoral Science Foundation of Shaanxi Province of China under Grant No. 2016BSHEDZZ121, by the Natural Science Basic Research Plan in Shaanxi Province of China under Grant No. 2015JM6355, and by the Key Program of the National Natural Science Foundation of China under Grant No. 61834005.

REFERENCES

[1] Z. Peng, Y Wang, W Yue, "A Fast Engine for Multi-String Pattern Matching," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 31, pp. 1-29, 2017.

[2] J. Tarhio, J. Holub, E. Giaquinta, "Technology Beats Algorithms (in Exact String Matching)," *Software: Practice and Experience*, vol. 47, pp. 1877-1885, 2017.

[3] S. Danish, Z. Farooqui, "Improved Approximate Multiple Pattern String Matching using Consecutive Q Grams of Pattern," *International Journal of Computer Applications*, vol. 76, pp. 1-6, 2017.

[4] S. Faro, T. Lecroq, S. Borzì, S. D. Mauro, A. Maggio, The String Matching Algorithms Research Tool," *Stringology,* pp. 99-111, 2016.

[5] C. Jianguo, L. Kenli, B. Kashif, A. Metwally, L. Keqin, Y. Philip, "Parallel Protein Community Detection in Large-Scale PPI Networks based on Multi-Source Learning," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp.1-1, 2018.

[6] X. Zhu, K. Li, A. Salah, L. Shi, K. Li, "Parallel Implementation of MAFFT on CUDA-enabled Graphics Hardware," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 12, pp. 205-218, 2014.

[7] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, K. Li, "A Parallel Random Forest Algorithm for Big Data in A Spark Cloud Computing Environment," *IEEE Transactions on Parallel and Distributed Systems,* vol. 28, pp. 919-933, 2016.

[8] B. Liu, S. He, D. He, Y. Zhang, & M. Guizani, "A Spark-Based Parallel Fuzzy C-Means Segmentation Algorithm for Agricultural Image Big Data. *IEEE Access,* vol. 7, pp. 42169-42180, 2019.

[9] J. Chen, K. Li, K. Bilal, X. Zhou, K. Li, P. Yu, "A Bi-Layered Parallel Training Architecture for Large-Scale Convolutional Neural Networks," *IEEE transactions on parallel and distributed systems,* pp. 1-1, 2018.

[10] W. Yang, K. Li, Z. Mo, K. Li, "Performance Optimization Using Partitioned SpMV on GPUs and Multicore CPUs," *IEEE Transactions on Computers,* vol. 64, pp. 2623-2636, 2014.

[11] B. Liu, Y. Zhao, Y. Li, Y. Sun, B. Feng, "A Thread Partitioning Approach for Speculative Multithreading," *the Journal of Supercomputing,* vol. 67, pp. 778-805, 2014.

[12] V. Nagaveni, G. Raju "Various String Matching Algorithms for DNA Sequences to Detect Breast Cancer using CUDA Processors," *International Journal of Engineering & Technology*, vol. 14, pp.42-47, 2014.

[13] C. S. Kouzinopoulos, K. G. Margaritis, "String Matching on a Multicore GPU Using CUDA," *In 13th Panhellenic Conference on Informatics,* 2009, pp. 14-18.

[14] K. G. Srinivasa, B. S. Devi, "GPU Based N-Gram String Matching Algorithm with Score Table Approach for String Searching in Many Documents," *Journal of The Institution of Engineers (India): Series B*, vol. 98, pp. 467-476, 2017.

[15] C. Choksuchat, C. Chantrapornchai, "Practical Parallel String Matching Framework for RDF Entailments with GPUs," *Information Systems Frontiers*, vol. 4, pp. 863-882, 2016.

[16] C. H. Lin, J. C. Li, C. H. Liu, et al, "Perfect Hashing Based Parallel Algorithms for Multiple String Matching on Graphic Processing Units," *IEEE Transactions on Parallel and Distributed Systems,* vol. 28, pp. 2639-2650, 2017.

[17] C. Y. Liao, C. H. Lin., "A Novel Parallel Dual-Character String Matching Algorithm on Graphical Processing Units," *International Conference on Algorithms and Architectures for Parallel Processing. Springer, Cham,* 2017, pp. 197-210.

[18] M. B. Baig, T. S. Li, "Parallel String Matching for Urdu Language Text," *International Conference on Intelligent Technologies and Applications,* 2018, pp. 369-378.

[19] R. S. Boyer, J. S. Moore, "A Fast String Searching Algorithm," *Communications of the ACM*, vol. 20, pp. 762-772, 1977.

[20] R. N. Horspool, "Practical Fast Searching in Strings," *Software: Practice and Experience*, vol. 10, pp. 501-506, 1980.

[21] H. M. Mahmoud, R. T. Smythe, M. Régnier, "Analysis of Boyer-Moore-Horspool string-matching heuristic," *Random Structures & Algorithms*, vol. 10, pp. 169-186, 2015.

[22] C. H. Lin, J. C. Li, S. C. Chang, "Perfect Hashing Based Parallel Algorithms for Multiple String Matching on Graphic Processing Units," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, pp. 2639-2650, 2017.

[23] X. Y. Peng, G. Q. Chen, P. C. Yu, et al, "Parallel Computing of Three-Dimensional Discontinuous Deformation Analysis Based on OpenMP.," *Computers and Geotechnics*, vol. 106, pp. 304-313, 2019.

[24] S. A. Asadollah, D. Sundmark, S. Eldh, et al, "10 Years of Research on Debugging Concurrent and Multicore Software: A Systematic Mapping Study," *Software Quality Journal*, vol. 25, pp. 1-34, 2016.

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

[26] B. Liu, J. He, Y. Geng, L. Huang, S. Li, "Toward Emotion-aware Computing: A Loop Selection Approach Based on Machine Learning for Speculative Multithreading," *IEEE Access,* vol. 5, pp. 3675-3686, 2017.

[27] N. P. Tran, M. Lee, S. Hong, et al, "Multi-stream Parallel String Matching on Kepler Architecture," *In Mobile, Ubiquitous, and Intelligent Computing*, 2014, pp. 307-313.

[28] J. Yoon, K. I. Choi, H. Kim, "A Finite Automaton-Based String Matching Engine on Graphic Processing Unit. IEICE Transactions on Fundamentals of Electronics," *Communications and Computer Sciences,* vol. 100, pp. 2031-2033, 2017.