DOI 10.7603/s40601-013-0022-0

GSTF Journal on Computing (JoC), Vol. 3 No. 3, Dec 2013

Multiple Buffering for Parallel Approximate Sequence Matching using Disk-based Suffix Tree on Multi-core CPU

Yousuke Watanuki, Keiichi Tamura, Hajime Kitakami, and Yoshifumi Takahashi

***Abstract*—Suffix trees, which are trie structures that present the suffixes of sequences (e.g., strings), are widely used for sequence search in different application domains such as, text data mining, bioinformatics and computational biology. In particular, suffix trees are useful in bioinformatics applications, because they can search similar sub-sequences and extract frequent sequence patterns efficiently. In recent years, efficient construction of a suffix tree that allows faster sequence searches has become one of the most important challenges, because the number and size of the data that are stored in sequence databases have been increasing exponentially. This paper proposes a novel parallelization model for approximate sequence matching that uses disk-based suffix trees, which are built on hard disks not on memory, on a multi-core CPU. In the proposed parallelization model, we divide an entire sequence database into two or more sub-databases called partitions. For each partition, we build a disk-based suffix tree and define a task as an approximate sequence matching on one disk-based suffix tree. Moreover, the proposed parallelization model involves a multiple buffering management system to avoid conflicts among CPU-cores. We evaluated the proposed parallelization model using an actual amino acid sequence database on a PC. The experimental results show a substantial improvement in computation performance.**

***Index Terms*—parallel processing; suffix tree; multi-core CPU; buffer management; approximate sequence matching**

I. Introduction

SUFFIX trees [1], [2], [3], which are trie structures that present the suffixes of sequences (e.g., strings), are widely used for sequence search in different application domains such as, text mining, pattern matching, bioinformatics and computational biology. In particular, suffix trees are useful in bioinformatics applications, to search for sequence patterns in genome and amino acid sequences. Bioinformatics researchers have focused on developing efficient suffix tree structures and improving the performance of sequence searches on suffix trees.

In recent years, efficient construction of a suffix tree that allows high-speed sequence searches has become one of the most important challenges, because the number and size of the data that are stored in sequence databases have been increasing exponentially [4], [5], [6]. For example, because of the various sequencing efforts and the development of sequencing techniques, as well as the current trend toward lower prices of hard disk systems, genome sequence databases have been growing at exponential rates. As a result of the enormous data size and extreme growth rate, researchers on suffix trees has met new challenges including the need for effective data structures and efficient algorithms for various sequence searches on suffix trees.

This study focuses on parallel approximate sequence matching using disk-based suffix trees on a multi-core CPU. An approximate sequence matching is one of the most important similarity searches on sequence databases. Approximate sequence matching is a common real world problem in a variety of application domains such as, signal recovery, DNA sequence matching, and pattern matching for text data. Disk-based suffix trees are suffix trees that nodes of the suffix trees are stored in pages on hard disks. Suffix trees quickly outgrow the main memory on a PC or workstation for sequence collections in the order of gigabytes. Therefore, almost all modern practical works construct disk-based suffix trees for large-scale sequence databases.

The goal of this study is to develop a efficient parallelization model for parallel approximate sequence matching for large-scale sequence databases on a multi-core CPU. Nowadays, PCs and workstations have one or more multi-core CPUs. A multi-core CPU is a single microprocessor with two or more independent CPU cores on a die, which are the units that read and execute program instructions [7]. It is necessary to develop an efficient parallelization model for the parallel approximate sequence matching using disk-based suffix trees on a multi-core CPU, because a multi-core CPU has some characteristics that are different from a conventional CPU.

The main contributions of this study are as follows.

- A novel parallelization model for the parallel approximate sequence matching on disk-based suffix trees using data partition-based parallelism is proposed. The proposed parallelization model divides an entire sequence databases into two or more sub-databases called partitions. For each partition, we build a suffix tree on hard disks, and a task is defined as an approximate sequence matching on one disk-based suffix tree, which is built from a partition.
- A multiple buffering designed for multi-core CPUs is involved in the proposed parallelization model. The nodes of a disk-based suffix tree are stored in pages on hard disks. If a node in a disk-based suffix tree is accessed during the processing of a matching, the node is read from the hard disk. The disk-based suffix tree usually requires a buffering management system to handle disk input/output (I/O). The multiple buffering has a buffer on

Y.Watanuki, K.Tamura, H.Kitakami and Y.Takahashi are with the Graduate School of Information Sciences, Hiroshima City University, 3-4-1, Ozuka-Higashi, Asa-Minami-Ku, Hiroshima 731-3194, Japan; corresponding e-mail: (ktamura@hiroshima-cu.ac.jp).

DOI: 10.5176/2251-3043_3.3.271

Published online: 28 February 2014

each CPU-core and thus can avoid conflict between I/O requests from CPU-cores.
- The proposed parallelization model is evaluated by using an actual amino acid sequence databases on two types of PCs. We used a PC with a middle-spec multi-core CPU and a low-spec hard disk system, as well as a PC with a middle-spec multi-core CPU and a high-spec hard disk system, which has a redundant array of inexpensive disks (RAID) file system. The experimental results show sufficient speed-up in proportion to the number of threads in the PC with the high-spec hard disk system.

The remainder of this paper is organized as follows. In Section 2, related work is reviewed. In Section 3, the data structure of suffix trees and approximate sequence matching are described. In Section 4, we propose a novel parallelization model for parallel approximate sequence matching on a Multi-core CPU. In Section 5, we discuss experimental results. In Section 6, we conclude this paper.

II. Related Work

In the last few decades, index and its parallel processing have been the most important techniques for large-scale databases [8]. In big-data era, these parallelization techniques are critical for researchers to develop high-performance database services in different application domains. Many studies on index and its parallel processing the have been conducting. In this session, related work is reviewed and the aim of this study is specified.

In the long history of study on index and its parallel processing, there has been considerable research on B-tree [9], which is a well-known index for relational databases. The task of developing parallelism of B-tree [10] is beneficial for developing high-performance database services, because many databases use B-tree or its subsequent indexes. R-tree [11] is also a well-known index for spatial databases, and its parallel processing is beneficial for large-scale geo-spatial databases [12].

The highlight of study on index and its parallel processing has varied with changes in modern CPU architecture trends. One of the most important current CPU architectures is the multi-core CPU [7]. Therefore, developing efficient parallelization models on a multi-core CPU fascinates researchers in many different application domains. For example, Goetz et al. [13] proposed a new B-tree optimized for many-core processors and modern memory hierarchies with flash storage and nonvolatile memory.

Likewise, in the last few decades, many researchers have tried to develop efficient data structures and construction methods, parallel construction and its query processing, on disk-based suffix trees as we see increasingly larger sequence databases [14], [15], [16], [17]. Almost all studies focus on efficient parallel construction methods and buffer management to reduce I/O, because one of the most time-consuming tasks on disk-based suffix trees is its construction time.

Amol et al. [16] proposed WAVEFRONT, which is a construction algorithm that diverges from the partition-and-merge approach to build disk-based suffix trees in a tiled fashion. Essam et al. [17] developed a disk-based suffix tree construction method, called Elastic Range (ERa), which works efficiently with very long strings that are much larger than the available memory. They implemented and evaluated their construction algorithm on a shared-nothing architecture and a multi-core CPU. ERa can index the entire human genome sequence in 19 minutes on a PC with an eight-core CPU.

Existing studies are limited, because; they focus only on the construction of disk-based suffix trees for large-scale sequence databases. This study focused on the performance of parallel sequence search using suffix trees on a multi-core CPU. Buffer management is one of the most important factors to improve the performance of disk-based suffix trees. Recently, some researchers have discussed the performance of buffer management on a multi-core CPU [18], [19]. Moreover, they have proposed a new mechanism to handle buffer management in order to reduce conflict among CPU-cores.

Naturally, some studies developed buffer management systems for disk-based suffix trees. Srikanta et al. [20] developed TOP-Q, is which improves the disk access behavior of the suffix tree. However, TOP-Q focuses on buffer management during the construction of suffix-trees. In this paper, we propose a multiple buffer management system for the parallel processing of approximate sequence searches using suffix trees on a multi-core CPU. The goal of this study is to develop an efficient management system for the parallel processing of disk-based suffix trees on a multi-core CPU.

III. Approximate Sequence Matching on Disk-based Suffix Tree

A. Suffix Tree

A sequence database SD consists of n tuples, where a tuple t_i consists of two items: a tuple ID tid_i and a sequence data S_i Therefore, sequence database is denoted by $SD = \{t_1, \cdots, t_n\}$, where $t_i = (tid_i, S_i)$. Let Σ be a set of symbols, and there are $|\Sigma|$ symbols in it. In gene sequence databases and amino acid databases, Σ is a set of alphabets. Each sequence S_i is represented as a sequence of elements in Σ and is denoted by $S_i = s_{i,1} \cdots s_{i,L(S_i)}\$$, where $s_{i,j} \in \Sigma$, the length of sequence S_i is denoted by $L(S_i)$, and \$ is a terminating symbol.

For example, suppose that there are three sequences, "ACGTACGA", "TGTT", and "CGAG" in a sequence database SD. Each sequence is stored in a tuple. Therefore, tuple t_1=(1,ACGTACGA\$), t_2=(2,TGTT\$), and t_3=(3,CGAG\$), where 1, 2, and 3 are tuple IDs. Let the l-th suffix of sequence S_i be $S_i[l..L(S_i)]$. For example, $S_2[0..L(S_2)]$= TGTT\$, $S_2[1..L(S_2)]$=GTT\$, $S_2[2..L(S_2)]$= TT\$, and $S_2[3..L(S_2)]$= T\$. In this paper, we call l suffix number SN.

A suffix tree is a rooted tree and is one of the particular trie trees, and sequence corresponds to a branch with the following properties. Every node has at least two edges and every edge has a label that represents a subsequence of the sequences in SD. Each of the node edges starts with a different symbol in Σ. For every node u, $p(u)$ denotes the path from the root node to u, and $\overline{u}$ is the concatenation of edge labels on the path from the root to u. A leaf node can be represented as

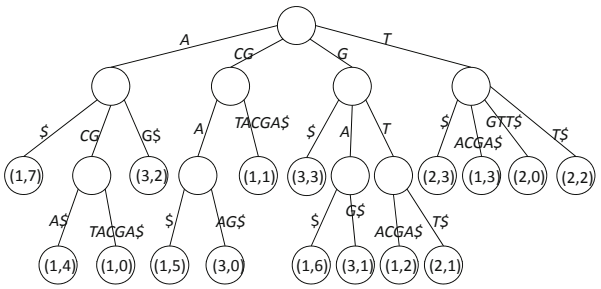


Fig. 1. Example of suffix tree.

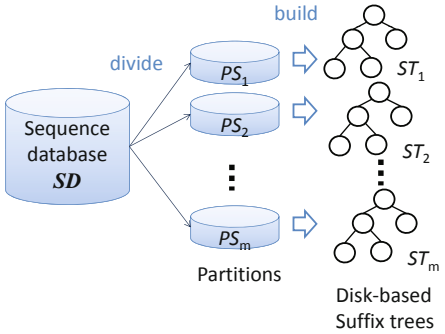


Fig. 2. Data parallelism.

(tuple ID, suffix number SN) with a pair of tuple ID and suffix number SN. Every leaf node corresponds to a suffix of S sequences in SD.

B. Disk-based Suffix Tree

The size of a suffix trees for a large-scale sequence database usually becomes larger than the size of main memory. To manage large-scale suffix tree, suffix trees are constructed on hard disk systems. Suffix trees on hard disk systems are called disk-based suffix trees. For managing disk-based suffix trees, a buffer management system is needed to read the node of the suffix tree that is stored on the hard disks. A node of the suffix tree is stored in a page, which is a unit of disk I/O in the operating system (OS). In this study, a page that contains nodes is called a node page. If a process of search on a disk-based suffix traverses a node of a disk-based suffix tree, the process requests the buffer manager to read a node page from disks that contain the requested node. The fetched page is stored in a buffer in the buffer manager. If the process traverses the node of the suffix tree again, the process reads the node from the buffer not the hard disks.

C. Approximate Sequence Matching

In this paper, we focus on an approximate sequence matching, which is a similarity sequence search under Humming distance. Let $Q = (Q_1, Q_1, \cdots, Q_n)$ be a key sequence pattern and $S_i = (s_{i,1}, \cdots, s_{i,L(S_i)})$ be a sequence. We want to find all matches with max_error or fewer mismatches, meaning positions k such that $|\{j : Q_j = s_{i,k+j-1}\}| \leq max\_error$. For example, suppose that key sequence pattern is "ACGT" and max_error = 2. In this example, sub-sequence "ACGA" and "ACGT" in SD in Fig.1 are matching, because mismatching is fewer than max_error =2. Node page is that intended to treat 32 node data of suffix tree as a unit. Node data is data of node configured suffix tree, and it is structure that has node number, child node number of including this node, the position and the length of suffix in the sequence database. The processing steps of the approximate sequence matching using a suffix tree are as follows.

(1) Node page is read from disk with the buffer manager.
(2) Searching for root node of suffix tree from this loaded node page. The root node $rnode$ is stored into $mnode$.
(3) The structure of node data contains offset of child node connecting this node. Root node has offset of child node. For each edge in $mnode$, a sub-sequence is contained in the edge matches $Q_1 \cdots Q_l$, where l is the length of the sub-sequence, and the number of mismatching is counted. At this time, the sub-sequence is connected to X.
(4) Searching for the node page contained child node using this offset. First, searching for node page stored in buffer area. If the node page is not stored in the buffer area, reading new node page from disk using buffer manager.
(5) Repeat (3),(4), move to the following child node if it matches the compared sub-sequence of node branch X_i and Q_i (X_i connects to X). If the two sequence are different, counting as an error. If the error is greater than max_error, look at the other branches without looking at its branch. If there is no branch to check, check the other branches back to a node on one.
(6) If it matches the length of X and the length n of Q, approximate sequence matching is successful.
(7) Following all the nodes in the suffix tree, when there are no more nodes that can follow, approximate matching is ended.

IV. Parallel Approximate Sequence Matching on Multi-Core CPU

In this section, the proposed parallelization model for parallel approximate sequence matching using disk-based suffix trees on a multi-core CPU is described.

A. Data Parallelism

The proposed parallelization model utilizes the data parallelism that divides the input database into two or more partitions. In a multi-core CPU environment, each CPU-core performs the same processing on different partitions. The entire sequence database SD is divided into two or more partitions $PSD = \{PS_1, PS_2 \cdots PS_m\}$, where m is the number of partitions, where $SD = PS_1 \cup PS_2 \cdots PS_m$, $PS_i \cap PS_j = \phi$. A disk-based suffix tree is built using each partition. Therefore, there are m disk-based suffix trees in our system, where m is the number of partitions (Fig. 2).

An approximate sequence matching on the sequence database SD can be performed in parallel using these disk-based suffix trees, which are built on a partition, because approximate sequence matching for each partition can be

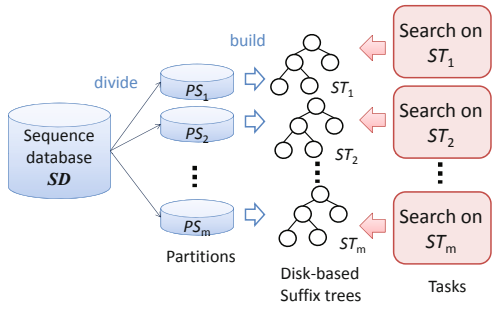


Fig. 3. Task model for parallel approximate sequence matching.

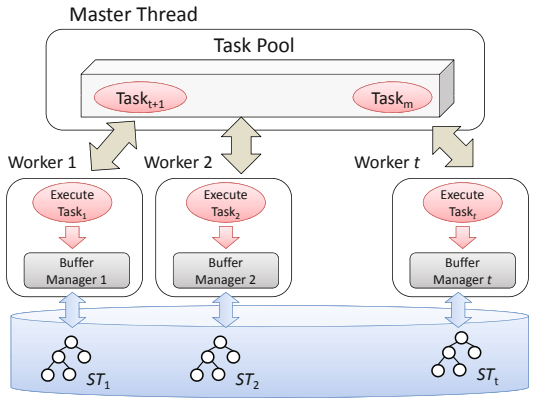


Fig. 4. Master worker model for parallel approximate sequence matching..

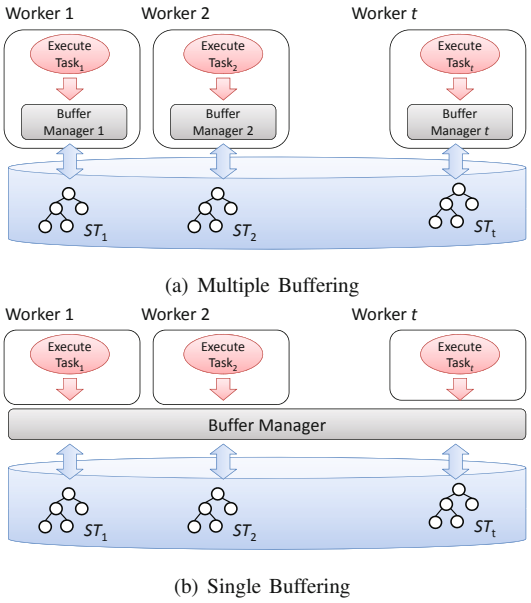
performed independently of the other partitions. Let AS_i be the result of approximate sequence matching on the i-th disk-based suffix tree. The result of the approximate sequence matching is $AS = AS_1 \cup AS_2 \cdots AS_m$.

B. Task Model

In this study, a task is defined as approximate sequence matching on a disk-based suffix tree that is built using a partition (Fig. 3). There are m tasks, where m is the number of disk-based suffix trees. Moreover, we use the master worker model to execute tasks in parallel. In this model, there are two components; a master process and a worker process. The master worker model always consists of one master process and multiple worker processes. In multi-core CPU environments, two components are represented as a master thread and a worker thread instead.

Fig. 4 shows the master worker model for parallel approximate sequence matching. The master thread manages the task pool that contains tasks, which are used for approximate sequence matching on disk-based suffix trees that are built for each partition. The master thread generates initial tasks and manages the task pool. Each worker thread gets a task from the task pool. Then, the worker thread performs an approximate sequence matching process on the disk-based suffix tree. If the worker thread finishes executing the task, the worker gets another task from the task pool.

If a worker thread traverses a node of a disk-based suffix tree, the worker thread requests the buffer manager to read a node page from disks that contain the requested node. The fetched page is stored in a buffer in the buffer manager. If the worker thread traverses the node of the disk-based suffix tree again, the worker thread reads the node from the buffer not the hard disks.



(a) Multiple Buffering

(b) Single Buffering

Fig. 5. Multiple buffering and single buffering.

C. Multiple Buffering

To improve disk I/Os during execution of parallel processing, the proposed parallelization model utilizes a novel buffer management system for a disk-based suffix tree on a multi-core CPU. The proposed buffer management system creates multiple buffer managers that provide a dedicated buffer for each worker thread. Conventional buffer management has one buffer manager on the system and the buffer manager is shared by multiple worker threads. On the other hand, in our proposed buffer management system, each worker thread has its own buffer manager (Fig. 5).

The advantage of the proposed buffer management system is that it can avoid conflicts among worker threads. For example, when the search process is performed only one buffer for all threads in Fig. 5(b), it might occur data races between threads by multiple threads is writing data to buffer area of the same node when the node data read from the disk stored in the buffer area. The multiple buffer management shown in Fig. 5(a) prevents conflicts between worker threads, because each worker thread has its own buffer manager.

D. Algorithm

This section describes the algorithm of parallel approximate sequence matching.

1) Master Thread: The processing steps of the master thread are as follows.

(1) The master thread receives the query parameters, database name SD, the number of threads t, query sequence pattern Q, the maximum number of error max_error from the user.

(2) The master thread creates a task pool.
(3) The master thread opens the database named SD, and gets information of partitions.
(4) The master thread puts tasks into the task pool.
(5) The master thread generates t worker threads.
(6) The master thread receives the results of approximate sequence matching, and then stores them into the result set.
(7) If the task pool is not empty, the process goes back to step (6) otherwise, the process goes to step (8).
(8) The master thread destroys the task pool.
(9) The master thread closes the database, and the result set is sent to the user.

2) Worker Thread: The processing steps of a worker thread are as follows.

(1) The worker thread receives the query parameters, query sequence pattern Q and the maximum number of errors max_error.
(2) The worker thread creates a buffer manager.
(3) The worker thread gets a task from the task pool.
(4) The worker thread executes approximate sequence matching on the suffix tree related to the task using its own buffer manager.
(5) The worker thread sends the result of the approximate sequence matching to the master thread.
(6) If the task pool is not empty, the process goes back to step (3); otherwise, the process goes to step (7).
(7) The worker thread destroys the buffer manager.
(8) The worker thread is terminated.

V. Experiment

We implemented the proposed parallelization model and evaluated it using an actual amino acid database. This section shows the content and results of experiments.

A. Experimental Setup

We used an amino acid database to evaluate the proposed parallelization model. The size of the database is approximately 1.0GB, and the entire database are divided into 1,000 partitions. The size of each partition is approximately 1.0MB. The average size of a disk-based suffix tree is approximately 250.0MB. Therefore, the total size of the disk-based suffix trees is 250.0GB.

We performed two experiments, and we compared the proposed parallelization model, which has multiple buffer managers with the master-worker model using a single-buffer. In the experiments, we used two types of computers. One is a PC with a middle-spec multi-core CPU and a low-spec hard disk system (PC1). The other is a PC with a middle-spec multi-core CPU and a high-spec hard disk system, which has a RAID file system. The specifications of these two PCs are as follows.

1) PC1 CPU: AMD FX-8120 3.1GHz/8M/8C; memory: 16GB; HDD:1TB; OS:Ubuntu 11.04
2) PC2 CPU: Xeon X5675 3.06GHz/12M/6C (×2); memory: PC/10600 DDR3 4,096MB ECC Reg. (×18); HDD: SATA 2TB 7200rpm (×24) RAID6; OS: CentOS 6.0

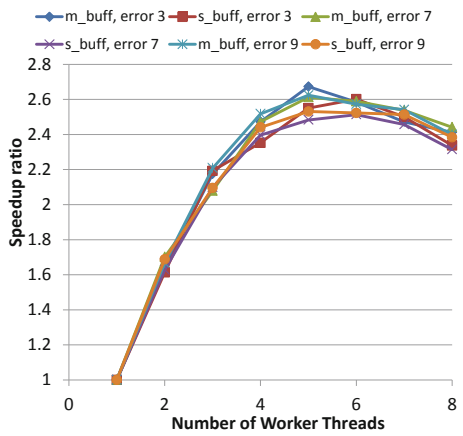


Fig. 6. Result of speed-up ratios using PC1 (We used several the maximum number of errors max_error).

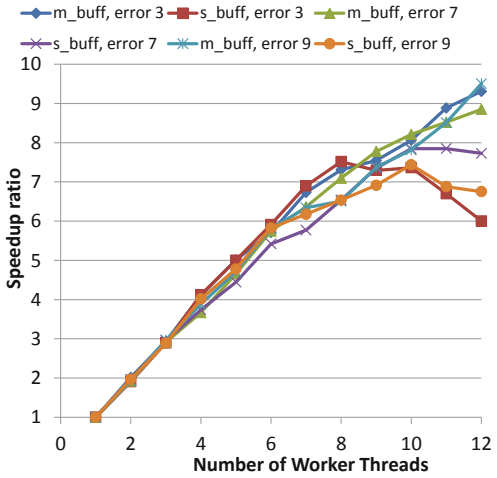


Fig. 7. Result of speed-up ratios using PC2 (We used several the maximum number of errors max_error).

B. Experiment 1

In Experiment 1, an approximate sequence matchings that query sequence pattern Q is "ASDFZZDSSSC" is used. We use several parameters of the maximum number of errors max_error is 3, 6, or 9. Each buffer manager has 150.0MB of buffer space. In the experiment, we measured the computation time of approximate sequence matchings.

Fig. 6 and 7 show the result of speed-up ratios using PC1 and PC2 respectively. For example, in these figures, the graph legends "m_buff, error 7" and "s_buff, error 7" indicate the measurement is obtained using multiple buffering with $max\_error = 7$ and using single buffering with $max\_error = 7$, respectively. The horizontal axis is the number of worker threads. The vertical axis is the speedup ratio.

Fig. 6 shows the results of experiments using PC1, which has a mid-spec multi-core CPU and a low-spec hard disk. As indicated in the figure, the speed-up ratios of multiple buffering are better than those of single buffering. However,

the speed-up ratios start decreasing when the number of worker thread is five. PC1 has a low-spec hard disk and disk I/Os start to conflict between worker threads when the number of worker thread is five.

Fig. 7 shows the results of experiments using PC2. PC2 has a mid-spec multi-core CPU and a high-spec hard disk. As indicated in the figure, the speed-up rations of multiple buffering are better than those of single buffering. Moreover, the speed-up ratios do not decrease compared with the results of PC1. The proposed parallelization model were able to obtain a sufficient speed improvement ratio relative to the number of threads of approximately 3.67 with 4 worker-threads, 7 with 8 worker-threads, and 8.7 with 12 worker-threads. In the single buffering, sufficient speedup ratio is obtained with up to 10 worker-threads; however the speedup ratios start decreasing when the number of worker threads is 9 or 10.

PC2 has a high-spec hard disk system, which is implemented on RAID6 and a high-performance RAID card. This lends significant performance improvement. PC1 causes I/O conflicts between worker threads and an I/O bottleneck because of the low-spec hard disk system, whereas the number of I/O conflicts in PC2 is fewer than that in PC1, because the high-spec hard disk system provides good I/O performance for the multiple buffer management system.

C. Experiment 2

In Experiment 2, we executed an approximate sequence matching those parameters are Q is "ASDFZZDSSS" and max_error is 3. In this experiment, we changed the size of the buffer space in each buffer manager to 85.0MB, 150.0MB, and 300.0MB. Moreover, we measured computation time while changing the number of threads from 1 to 8 in PC1, and from 1 to 12 in PC2.

Fig. 8 and 9 show the speed-up ratios using PC1 and PC2, respectively. For example, in these figures, the graph legends "m_buff, 85" and "s_buff, 85" indicate the measurement is obtained using multiple buffering and a buffer size of 85.0MB and using single buffering with a buffer size of 85.0MB respectively. The horizontal axis is the number of worker threads. The vertical axis is the speedup ratio.

When we look at "m_buff" in Fig. 8, the value of the speedup ratio increases with a buffer space of 300.0MB at 6 worker-threads, whereas the value decreases with a buffer space of 85.0MB, and 150.0MB. This is because the speedup ratio might be influenced by the processing needed to delete the node page of the buffer space.

Comparing the "m_buff" and "s_buff" curves in Fig. 8, and Fig. 9, we find that the "m_buff" curves show better values in Fig. 9 regardless of the size of the buffer space. If there are a large number of threads, the difference between "m_buff" and "s_buff" is particularly remarkable. The cause of this, when using single buffering as a whole, it occur conflicts between threads by writing against the same buffer space occur in multiple worker-threads, and the speedup ratio might be influenced by the processing needed to delete the node page of the buffer space. These results indicate that multi-buffer management is suitable for parallel processing.

In addition, with a buffer space of 85.0MB and 150.0MB, we found sufficient speedup ratios of approximately 5.5 with 6 threads, and 10.5 with 12 threads. However, with a buffer space of 300.0MB, the speedup ratio was smaller than the other on 11, 12 threads as about 8.2 with 10 threads, about 8.7 with 12 threads. The cause of this, when buffer space is 300.0MB, it is considered that processing looking for a particular node page from the buffer space is large.

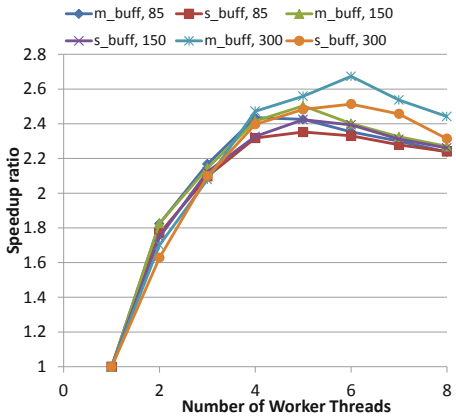


Fig. 8. Speed-up ratios of changing the buffer space using PC1.

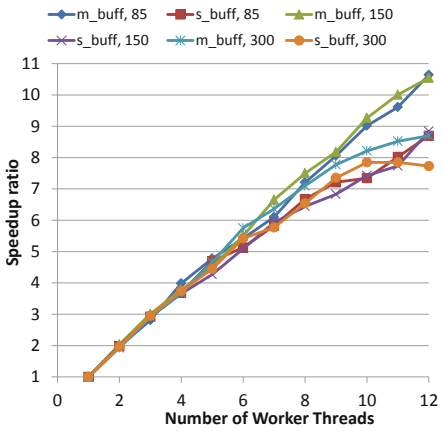


Fig. 9. Speed-up ratios of changing the buffer space using PC2.

VI. Conclusion

In this paper, we propose a novel parallelization model for approximate sequence matching that uses disk-based suffix trees, which are built on hard disks not on memory, on a multi-core CPU. The proposed parallelization model is based on data parallelism, and we divide an entire sequence database into two or more sub-databases called partitions. For each partition, a disk-based suffix tree is built and a task is defined as an approximate sequence matching on one disk-based suffix tree. Moreover, the proposed parallelization model involves a multiple buffering management system to avoid conflicts among CPU-cores. The experiments using an actual amino acid sequence database on PCs shows the proposed model is

good performance. In our future work, we intend to investigate the trade-off relationship between speed-up and the number of partitions. In addition, we will develop efficient buffer size management for the parallelization model for approximate sequence matching.

ACKNOWLEDGMENT

This work was supported in part by Hiroshima City University Grant for Special Academic Research (General Studies).

REFERENCES

[1] P. Weiner, "Linear pattern matching algorithms," in *Proceedings of the 14th Annual Symposium on Switching and Automata Theory (swat 1973)*, SWAT '73, pp. 1–11, 1973.
[2] E. M. McCreight, "A space-economical suffix tree construction algorithm," *Journal of the ACM*, vol. 23, pp. 262–272, Apr. 1976.
[3] D. Gusfield, *Algorithms on strings, trees, and sequences: computer science and computational biology*. New York, NY, USA: Cambridge University Press, 1997.
[4] Y. Tian, S. Tata, R. A. Hankins, and J. M. Patel, "Practical methods for constructing suffix trees," *The VLDB Journal*, vol. 14, no. 3, pp. 281–299, 2005.
[5] B. Phoophakdee and M. J. Zaki, "Genome-scale disk-based suffix tree indexing," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pp. 833–844, 2007.
[6] M. Barsky, U. Stege, A. Thomo, and C. Upton, "Suffix trees for very large genomic sequences," in *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pp. 1417–1420, 2009.
[7] M. R. M. Mark D. Hill, "Amdahl's law in the multicore era," vol. 41 of *IEEE Computer 2008*, pp. 33–38, 2008.
[8] D. J. DeWitt and J. Gray, "Parallel database systems: the future of database processing or a passing fad?," *ACM SIGMOD Record*, vol. 19, pp. 104–112, Dec. 1990.
[9] D. Comer, "Ubiquitous b-tree," *ACM Computing Surveys (CSUR)*, vol. 11, pp. 121–137, June 1979.
[10] B. Seeger and P.-A. Larson, "Multi-disk b-trees," in *Proceedings of the 1991 ACM SIGMOD international conference on Management of data*, SIGMOD '91, pp. 436–445, 1991.
[11] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, SIGMOD '84, pp. 47–57, 1984.
[12] I. Kamel and C. Faloutsos, "Parallel r-trees," in *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, SIGMOD '92, pp. 195–204, 1992.
[13] G. Graefe, H. Kimura, and H. Kuno, "Foster b-trees," *ACM Trans. Database Syst.*, vol. 37, pp. 17:1–17:29, Sept. 2012.
[14] R. Hariharan, "Optimal parallel suffix tree construction," in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, STOC '94, pp. 290–299, 1994.
[15] D. Tsirogiannis and N. Koudas, "Suffix tree construction algorithms on modern hardware," in *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pp. 263–274, 2010.
[16] A. Ghoting and K. Makarychev, "I/o efficient algorithms for serial and parallel suffix tree construction," *ACM Trans. Database Syst.*, vol. 35, pp. 25:1–25:37, Oct. 2010.
[17] E. Mansour, A. Allam, S. Skiadopoulos, and P. Kalnis, "Era: efficient serial and parallel suffix tree construction for very long strings," *Proc. VLDB Endow.*, vol. 5, pp. 49–60, Sept. 2011.
[18] R. Johnson, I. Pandis, N. Hardavellas, A. Ailamaki, and B. Falsafi, "Shore-mt: a scalable storage manager for the multicore era," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pp. 24–35, 2009.
[19] X. Ding, K. Wang, and X. Zhang, "Srm-buffer: an os buffer management technique to prevent last level cache from thrashing in multicores," in *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pp. 243–256, 2011.
[20] S. J. Bedathur and J. R. Haritsa, "Engineering a fast online persistent suffix tree construction.," in *ICDE*, pp. 720–731, 2004.

Yousuke Watanuki is a student at the Department of Intelligent Systems, Graduate School of Information Sciences, Hiroshima City University, Hiroshima, Japan. His research interests include suffix tree and parallel computing.

Keiichi Tamura received his B.Eng., M.Eng., and Ph.D. degrees in Information Science from Kyushu University, Fukuoka, Japan, in 1998, 2000, and 2005, respectively. He is presently Associate Professor at the Department of Intelligent Systems, Graduate School of Information Sciences, Hiroshima City University, Hiroshima, Japan. He has been in organizing committee of IEEE SMC Hiroshima Chapter since 2012. His research interests include parallel computing, data engineering, data mining, and evolutionary computation. He is a member of IEEE, Information Processing Society of Japan, Database Society of Japan, The Japanese Society for Artificial Intelligence, Japan Society for Fuzzy Theory and Intelligent Informatics.

Hajime Kitakami has been a Professor in the Department of Intelligent Systems, Graduate School of Information Sciences, Hiroshima City University in Japan since 1994. He received the M.Eng. from Tohoku University in 1976 and Ph.D. in engineering from Kyushu University in 1992. His paper was recorded as the 25th Anniversary Best Paper Award of Information Processing Society of Japan (IPSJ) in 1985. He received Paper Award from Japanese Society for Engineering Education (JSEE) in 2003. His research interests include database, data mining, distributed parallel processing, and bioinformatics. He has been an editorial board member for Transactions on Mathematical Modeling and its Applications (TOM), Journal of the Information Processing Society of Japan (IPSJ) since 2006. Also, he has been an editorial board member for Journal of the Database Society of Japan (DBSJ) since 2008.

Yoshifumi Takahashi Yoshifumi Takahashi received a Master of Information Engineering degree from Hiroshima City University, Japan in 2010. He is now a doctoral student in the Graduate School of Information Science, Hiroshima City University.