ISSN 1995-0802, Lobachevskii Journal of Mathematics, 2019, Vol. 40, No. 12, pp. 2124–2132. © Pleiades Publishing, Ltd., 2019.

Discovery of Time Series Motifs on Intel Many-Core Systems

M. L. Zymbler[1*] and Ya. A. Kraeva[1]**

(Submitted by A. M. Elizarov)

[1] *South Ural State University, Chelyabinsk, 454080 Russia*

Received August 18, 2019; revised August 20, 2019; accepted August 24, 2019

Abstract—A motif is a pair of subsequences of a longer time series, which are very similar to each other. Motif discovery is applied in a wide range of subject areas involving time series: medicine, biology, entertainment, weather prediction, and others. In this paper, we propose a novel parallel algorithm for motif discovery using Intel MIC (Many Integrated Core) accelerators in the case when time series fit in the main memory. We perform parallelization through thread-level parallelism and OpenMP technology. The algorithm employs a set of matrix data structures to store and index the subsequences of a time series and to provide an efficient vectorization of computations on the Intel MIC platform. The experimental evaluation shows the high scalability of the proposed algorithm.

DOI: 10.1134/S199508021912014X

Keywords and phrases: *time series, motif discovery, OpenMP, Intel Xeon Phi, data layout, vectorization.*

1. INTRODUCTION

Time series motif is a pair of subsequences of a longer time series, which are very similar to each other [13]. The problem of finding motifs in a time series is one of the topical issues in time-series data mining and has applications in a wide range of subject areas: medicine, biology, entertainment, weather prediction, and others.

Since straightforward algorithm for computing motifs is quadratic with respect to the time series length, a number of approximate algorithms to discover motifs have been proposed [2, 8, 9, 15, 18] where time complexity is linear or logarithmic of the time series length but the associated constant factors are high. In [10], Mueen et al. introduced the MK algorithm for exact motifs discovery. Despite the fact that MK is able to reduce the run time by three orders of magnitude, it suffers in case of large time series (the order of hundreds of thousands). Thus, application of modern parallel hardware to accelerate discovery motifs in time series is a topical issue.

In this paper, we address the task of accelerating the MK algorithm on Intel MIC (Many Integrated Core) systems [3, 16]. MIC accelerators are based on the Intel x86 architecture and provide a large number of computing cores with 512-bit wide vector processing units (VPU) while supporting the same programming methods and tools as a regular Intel Xeon CPU. Intel provides two generations of MIC systems (under the codename of Intel Xeon Phi), namely Knights Corner (KNC), featuring 57 to 61 cores, and Knights Landing (KNL), featuring 64 to 72 cores. The benefits from the use of MIC accelerators usually manifest in applications where the processing of large amounts of data (at least hundreds of thousands of elements) can be arranged as loops that may be submitted to vectorization by a compiler [17]. Vectorization means the compiler's ability to transform the loops into sequences of vector operations [1] of VPUs.

In this study, we propose a parallel algorithm for exact motif discovery on Intel MIC systems, assuming that time series fit into the main memory. The paper continues our research on accelerating various time series mining tasks on Intel MIC systems (namely subsequence matching [6] and discord

*E-mail: mzym@susu.ru

**E-mail: kraevaya@susu.ru

discovery [20]). The remainder of the paper is organized as follows. In Section 2, we give the formal definitions along with a brief description of the MK algorithm. Section 3 presents the proposed parallel algorithm. In Section 4, we give the results of the experimental evaluation of our algorithm. Section 5 contains a short overview of related work. Finally, Section 6 summarizes the results obtained and suggests directions for further research.

2. PROBLEM STATEMENT AND THE SERIAL ALGORITHM

2.1. Notations and Definitions

Below, we follow [10] to give formal definitions and the statement of the problem.

A *time series* T is a sequence of real-valued elements: $T = (t_1, \ldots, t_m)$, $t_i \in \mathbb{R}$. The length of a time series is denoted by $|T|$.

A *subsequence* $T_{i,n}$ of a time series T is its contiguous subset of n elements that starts at position i: $T_{i,n} = (t_i, \ldots, t_{i+n-1})$, $1 \le n \ll m$, $1 \le i \le m - n + 1$.

A set of all subsequences of T with length n is denoted by S_T^n. Let N denote the number of subsequences in S_T^n, i.e. $N := |S_T^n| = m - n + 1$.

A *distance function* for any two subsequences is a nonnegative and symmetric function $\mathrm{Dist}\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$.

A *subsequence motif* is a pair of subsequences $\{T_{i,n}, T_{j,n}\}$ of the time series T that are most similar. That is,

$$\forall a, b, i, j Dist(T_{i,n}, T_{j,n}) \le Dist(T_{a,n}, T_{b,n}), \quad |i - j| \ge w, \quad |a - b| \ge w, \quad w > 0,$$

where w is a parameter that denotes minimum gap between subsequences in the motif, i.e. subsequences should stand apart from each other at least w positions in time series. This restriction helps to discard the trivial subsequence motifs [13].

As the distance function, we use the ubiquitous Euclidean distance measure, defined as follows. Given two subsequences $X, Y \in S_T^n$, the Euclidean distance between them is calculated as

$$\mathrm{ED}(X, Y) := \left[\sum_{i=1}^{n} (x_i - y_i)^2\right]^{1/2}.$$

2.2. The Serial Algorithm

MK [10] is one of the fastest serial algorithms for discovery exact motifs in time series. The algorithm supposes that before the search of the subsequence motif, every subsequence is to be z-normalized. The *z-normalization* of a subsequence $C \in S_n^T$ is a subsequence $\hat{C} = (\hat{c}_1, \ldots, \hat{c}_n)$ in which

$$\hat{c}_i := \frac{c_i - \mu}{\sigma}, \quad 1 \le i \le n; \quad \mu := \frac{1}{n}\sum_{i=1}^{n} c_i, \quad \sigma := \left[\frac{1}{n}\sum_{i=1}^{n} c_i^2 - \mu^2\right]^{1/2}.$$

The main idea of the algorithm is to reduce the search space by means of the reference subsequence where a randomly chosen subsequence of the time series is taken as a reference. The algorithm calculates and sorts the distances from all the subsequences of the time series to the reference subsequence in ascending order. Then, the algorithm exploits the following observation: if two objects are close in the original space, they must also be close in the linear ordering (but the opposite is not true). According to the triangular inequality, the distance in the linear ordering between subsequences of the motif is the lower bound to the distance between these subsequences in the original space. Hereinafter, to differ the above-mentioned types of distance, we call the distance in the original space as a true distance.

The algorithm maintains the bsf *(best-so-far)* variable representing a running minimum, and updates it whenever the algorithm finds a pair of time series having a smaller true distance between them. By scanning of the subsequences along the above-mentioned linear ordering, the algorithm calculates the respective lower bounds. If the lower bound has exceeded bsf, the true distance will exceed bsf as well, and the respective pair of subsequences is pruned without calculation of the true distance. If the

pair is not pruned, then the true distance is calculated. Next, bsf is updated if it is larger then the value of the true distance. Proceeding similarly, the algorithm considers all possible pairs of subsequences, which are $offsef$ ($1 \leq offset \leq N-1$) in the linear ordering.

The algorithm continues till it reaches an $offset$ for which there is no pair having lower bound larger than the bsf and staying $offset$ apart in the linear ordering. At that point the algorithm abandons the search with the exact motif pair of subsequences which are most similar to each other.

In addition, the algorithm employs multiple reference subsequences to get the tighter lower bounds. The reference subsequence with the largest standard deviation is found and is used to sort by ascending the distances between this reference subsequence and all other subsequences from the time series. If one of the lower bounds is larger than bsf, this pair is pruned. The algorithm stops if all lower bounds of all pairs of subsequences $offset$ apart are larger than bsf.

3. ACCELERATING MOTIF DISCOVERY ON INTEL MIC SYSTEMS

In this section, we present an approach to the parallelization of the MK algorithm [10] on the Intel MIC platform. We exploit the thread-level parallelism and the OpenMP technology [7]. Below, in Section 3.1 we describe internal data layout of the algorithm, and in Section 3.2, we show the proposed parallel algorithm.

3.1. Data Layout

Since OpenMP is more suitable to process arrays and matrices, we employ a set of matrix data structures that store the algorithm's data. In addition, data structures are aligned in main memory, and computations are organized with as many vectorizable loops as possible. Vectorization means a compiler's ability to transform the loops into sequences of vector operations [1] of VPUs. We should avoid unaligned memory access since it can cause inefficient vectorization due to timing overhead for loop peeling [1].

The parallel algorithm employs the data structures depicted in Fig. 1. The time series is stored as a matrix of aligned subsequences. Let us denote by $width_{VPU}$ the number of floats stored in the VPU. If n (i.e. the length of the discord to be discovered) is not a multiple of $width_{VPU}$, then the subsequence is padded with zeroes, with a number of zeroes $pad := width_{VPU} - (n \bmod width_{VPU})$. The *aligned subsequence* $\tilde{T}_{i,n}$ is defined as follows:

$$\tilde{T}_{i,n} = \begin{cases} t_i, t_{i+1}, \ldots, t_{i+n-1}, \underbrace{0, 0, \ldots, 0}_{pad}, & if \quad n \bmod width_{VPU} > 0, \\ t_i, t_{i+1}, \ldots, t_{i+n-1}, & otherwise. \end{cases}$$

Next, we store all (aligned) subsequences of a time series in the *subsequence matrix* $S_T^n \in \mathbb{R}^{N \times (n+pad)}$, which is defined as below: $S_T^n(i,j) := \tilde{t}_{i+j-1}$. Let us denote the number of randomly chosen reference subsequences exploited by the algorithm as r ($0 < r \ll N$). Then Ref, *set of reference subsequences* is defined as follows:

$$Ref := \{T_{i_1,n}, \ldots, T_{i_r,n} \mid T_{i_j,n} \in S_T^n, i_j = random(1..N)\}.$$

The *index of reference subsequences* $I_{Ref} \in \mathbb{N}^r$ contains position in T of each reference subsequence. Given a subsequence matrix S_T^n and an index of reference subsequences I_{Ref}, we define the *distance matrix* $D \in \mathbb{R}^{r \times N}$ that contains distances between all reference subsequences and all subsequences of S_T^n as follows:

$$D(i,j) := \mathrm{ED}(T_{I_{Ref}(i),n}, T_{j,n}).$$

The *reference deviation vector* is an array $SD \in \mathbb{R}^r$ that for each reference subsequence contains standard deviation in the distances from itself to all the subsequences in S_T^n.

The *reference deviation index* $I_{SD} \in \mathbb{N}^r$ contains unique numbers in range from 1 to r where each number reflects the position of the respective reference subsequence in Ref ordered by descending of their standard deviation.

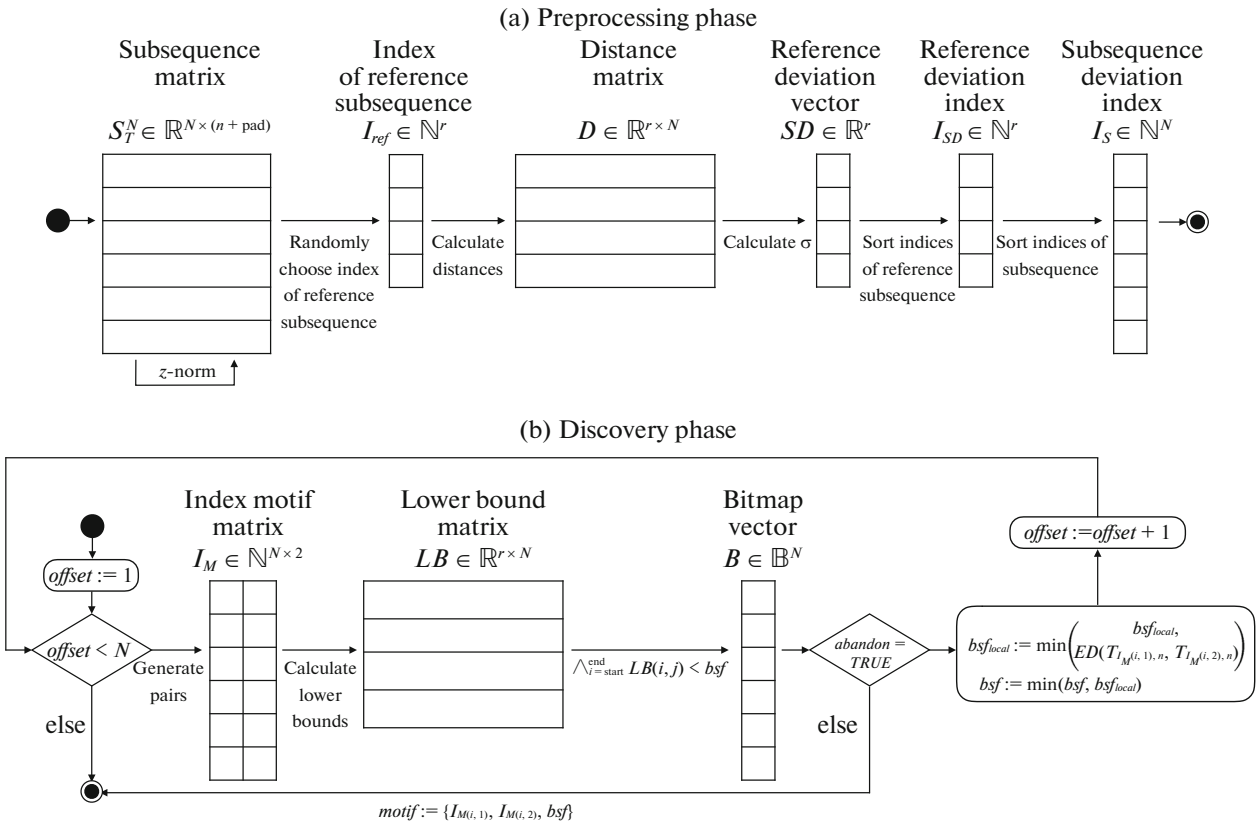

Fig. 1. Data flow of the algorithm.

The *subsequence deviation index* is an array $I_S \in \mathbb{N}^N$ that contains indices of subsequence in S_T^n ordered ascending by their distances to a reference subsequence, which has the largest standard deviation.

We also establish the *index motif matrix* $I_M \in \mathbb{N}^{N \times 2}$ to store pairs of indices of two subsequences, which are the possible motif and are located from each other by $offset$ in the deviation index I_S.

The *lower bound matrix* $LB \in \mathbb{R}^{r \times N}$ contains lower bounds between each reference subsequence and each possible motif, and calculated as follows:

$$LB(i, j) := |D(I_{SD}(i), I_M(j, 1)) - D(I_{SD}(i), I_M(j, 2))|.$$

The *bitmap vector* is an array $B \in \mathbb{B}^N$, which for each possible motif stores the logical conjunction of comparison bsf and all the motif's lower bounds. If an element of the bitmap vector equals `FALSE` then the respective motif is discarded without calculation of the distance between its subsequences. The bitmap vector is defined as follows:

$$B(i) := \wedge_{j=1}^{r}(LB(i, j) < bsf), \quad 1 \le i \le N.$$

3.2. Parallel Implementation of the Algorithm

The proposed algorithm (see pseudo-code in Algorithm 1) consists of two phases, namely preprocessing and discovery. At the *preprocessing phase*, the algorithm forms the subsequence matrix S_T^n and performs z-normalization of its rows. Then the Ref set of randomly chosen reference subsequences and its index I_{Ref} are formed. After that, algorithm calculates the distance matrix D and reference deviation vector SD. Next, reference deviation index I_{SD} and subsequence deviation index I_S are formed. The bsf threshold is initialized by the minimum in distance matrix D. The above-described computations are implemented as parallelized loops that are easily vectorized by the compiler.

Algorithm 1. PhiMotifDiscovery (in T, n, r; out $motif$)

▷ Data preprocessing

1: Znormalize(S_T^n)
2: $Ref \leftarrow r$ randomly chosen subsequences from S_T^n
3: $I_{Ref} \leftarrow r$ indices of elements in Ref
4: $D \leftarrow$ CalculateDistances(S_T^n, Ref)
5: $SD \leftarrow$ CalculateStdDev(D)
6: $I_{SD} \leftarrow$ Sort(SD); $I_S \leftarrow$ Sort($D(I_{SD}(1), \cdot)$)
7: $bsf \leftarrow \min(d_{i,j})$

▷ Parallel motif discovery

8: #pragma omp parallel
9: $start \leftarrow thread_{num} \cdot N$; $end \leftarrow thread_{num} \cdot (N+1)$
10: #pragma omp for schedule (dynamic)
11: **for all** $offset \in 1..N$ **do**
12: $bsf_{local} \leftarrow bsf$; $motif_{local} \leftarrow \{\infty; \infty; bsf_{local}\}$
13: $I_M \leftarrow$ GeneratePairs ($offset$, I_S)
14: $LB \leftarrow$ CalculateLowerBounds (D)
15: $B(i) \leftarrow \wedge_{i=start}^{end} LB(i,j) < bsf$
16: $abandon \leftarrow \vee_{i=start}^{end} B(i)$

▷ Pruning unpromising motifs

17: **if** $abandon =$ `TRUE` **then**
18: #pragma omp cancel for
19: **else**
20: **for all** $i \in start..end$ **do**
21: **if** $B(i) =$ `TRUE` **then**
22: $d \leftarrow \mathrm{ED}(T_{I_M(i,1),n}, T_{I_M(i,2),n})$
23: **if** $d < bsf_{local}$ **then**
24: $bsf_{local} \leftarrow d$; $motif_{local} \leftarrow \{I_M(i,1); I_M(i,2); bsf_{local}\}$
25: #pragma omp critical
26: **if** $bsf_{local} < bsf$ **then**
27: $bsf \leftarrow bsf_{local}$; $motif \leftarrow motif_{local}$
28: #pragma omp cancellation point for
29: **return** $motif$

At the *discovery phase*, the algorithm searches the subsequence motif through the loop along the $offset$ variable. The loop is parallelized by the standard OpenMP compiler directive `#pragma omp for`. Pruning unpromising motifs results in uneven computational loading of threads. Thus, to increase the efficiency of the parallel algorithm, we add to the above-mentioned `#pragma` the `schedule (dynamic)` parameter, which dynamically distributes loop iterations among threads.

Each of the following data structures, namely index motif matrix I_M, lower bound matrix LB, and

bitmap vector B is split into logical equal-length segments, and while scanning, a thread processes its own segment. Each thread maintains its own local threshold bsf_{local} initialized by the bsf value.

A thread performs as follows. Index motif matrix I_M is filled in by pairs of subsequence indices that are of $offset$ apart in the subsequence deviation index I_S. Then, according to the triangular inequality, lower bounds for possible motifs are calculated and the lower bound matrix LB is filled in. After that, bitmap vector B is calculated. Next, if the logical disjunction of all the elements of the bitmap vector is `TRUE` (i.e. all the lower bounds for all the pairs of $offset$ apart are larger than bsf) then the motif is found and the rest candidates can be abandoned. This implemented through the `#pragma omp cancel for` directive for a thread that reaches this line of code first and the `#pragma omp cancellation point for` directive for the rest threads.

If the motif is not found then the algorithm calculates true distance between subsequences in each possible motif for which the respective element of the bitmap vector is `TRUE`. In case the true distance is less than bsf_{local}, it is updated by the true distance value. Then the algorithm finds bsf as minimum of all the bsf_{local} values and updates the motif accordingly. Here, the critical section is used in order to update the shared variable bsf correctly.

A lot of statements of the algorithm, namely calculations of the lower bounds, the bitmap vector, Euclidean distances, and abandoning criterion are vectorized by the compiler, so this increases the overall algorithm's performance.

4. THE EXPERIMENTS

4.1. The Experimental Setup

We evaluated the proposed algorithm in experiments conducted on Intel many-core systems at the South Ural State University [5] (see Table 1 for a summary of the hardware involved).

In the experiments, we assessed the performance and scalability of the algorithm while varying the motif length. We measured the run time (after deduction of the I/O time required for reading input data and writing the results) and calculated the algorithm's speedup and parallel efficiency, which are defined as follows. The speedup and the parallel efficiency of a parallel algorithm employing k threads are calculated, respectively, as $s(k) = \frac{t_1}{t_k}$ and $e(k) = \frac{t_1}{k \cdot t_k}$, where t_1 and t_k are the run times of the algorithm when one and k threads, respectively, are employed.

In the experiments, we used two following time series, each of 4×10^5 elements. The ECG dataset [4] represents electrocardiogram signals digitized at 128 Hz. The RW dataset was generated according to the Random Walk model [14] and considered for evaluation of the MK algorithm [10].

4.2. Results and Discussion

Figure 2 depicts experimental results regarding the algorithm's scalability on the Intel MIC system. For both datasets, the algorithm showed close-to-ideal speedup and a 80 to 100 percent parallel efficiency (with respect to the length of the motif that is to be found) when the number of threads matches the number of physical cores the algorithm runs on. As expected, the algorithm demonstrates better scalability with larger values of the motif length because this provides a higher computational load. When more than one thread per physical core is employed, the algorithm shows sub-linear speedup and an accordingly diminished parallel efficiency, but without a tendency to degrade.

Experimental results concerning the algorithm's performance on various Intel platforms are depicted in Table 2. As expected, our parallel algorithm substantially outruns the serial MK algorithm. Also, as we can see, our algorithm performs better on systems with greater numbers of cores. At the same time, when our algorithm runs on the Intel MIC accelerator, it performs better than when it runs on a node equipped with two Intel Xeon CPUs.

Summing up, the proposed algorithm efficiently utilizes the vectorization capabilities of the many-core system and shows high scalability, especially in case of high computational load due to larger motif length.

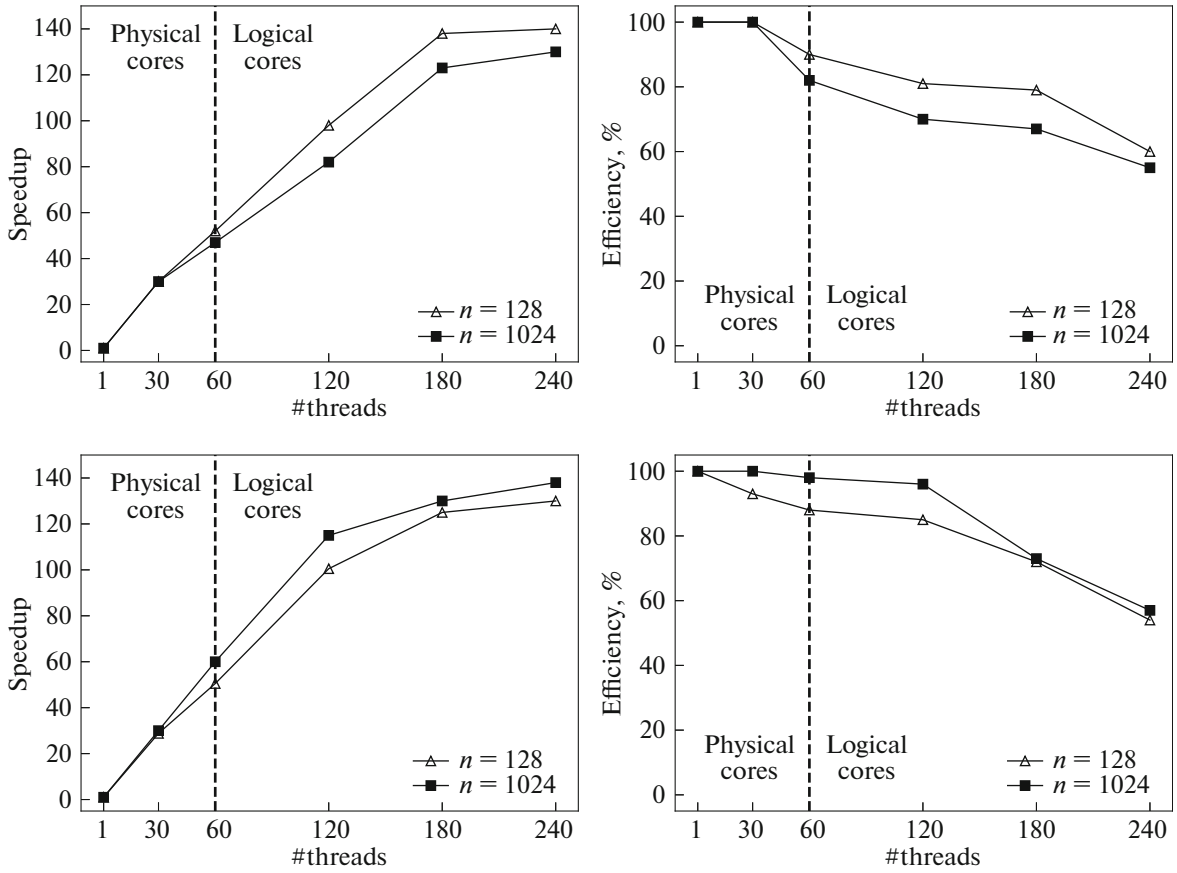

Fig. 2. Scalability of the algorithm.

5. RELATED WORK

Since straightforward algorithm for computing motifs is quadratic of the time series length, a number of approximate algorithms to discover motifs have been proposed, e.g. [2, 8, 9, 15, 18]. These algorithms are $O(m)$ or $O(m\log m)$ time complexity with very high associated constant factors.

The FLAME algorithm by Wilson et al. [19] was designed to find guaranteed motifs in DNA but the authors show that a time series can be discretized and given to FLAME. Thus the algorithm is approximate for real-valued time series.

The MK algorithm by Mueen et al. [10] finds the exact motif and is based on the following idea. Given a set of points in space that are ordered with respect to the distance to a randomly chosen reference point, any two objects, which are close in the original space, will also be close in the linear ordering (but the opposite is not true). Further, the algorithm is generalized to multiple reference points in order to reduce the search space and results in much faster motif discovery (up to three orders of magnitude faster than straightforward approach for large time series data).

The Par-MK algorithm by Narang et al. [11] is a parallel algorithm for exact discovery of time series motifs on SMP systems for the case when data fit in the main memory. Par-MK exploits thread-level parallelism and POSIX threads [12]. Par-MK employs two dimensions of parallelism, namely array parallelism and offset parallelism. The former is intended to split across threads the sorted array containing the distances with respect to the reference point. The latter is along the increasing offset value from 1 to $n - offset$. When both the dimensions of parallelism are used, the threads that work on different offsets but the same array partition are co-scheduled on the same core or multiple cores that share the same L2 cache. The authors proposed two versions of the algorithm to optimize the imbalance across the threads caused by non-uniform workload on each thread, namely Par-MK-SLB and Par-MK-DLB (static and dynamic load balancing, respectively). In the experiments on real dataset, $|T| = 1.8 \times 10^5$ and $n = 200$, the algorithm showed super-linear speedup on 32 cores due to the dynamic

Table 1. Hardware environment for the experiments.

Specifications	Host	MIC
Model, Intel Xeon	X5680	Phi SE10X
Physical cores	2×6	61
Hyperthreading factor	2×	4×
Logical cores	24	244
Frequency, GHz	3.33	1.1
VPU size, bit	128	512
Peak performance, TFLOPS	0.371	1.076

Table 2. Performance of the algorithm.

Motif length	ECG dataset				RW dataset			
	MK	Our algorithm			MK	Our algorithm		
	Host (1 core)	Host (12 cores, 24 threads)	MIC (61 cores, 244 threads)	Speedup (Host/MIC)	Host (1 core)	Host (12 cores, 24 threads)	MIC (61 cores, 244 threads)	Speedup (Host/MIC)
128	1 908.3	974.7	787.1	1.2×	2 143.7	1 036.3	820.4	1.3×
1 024	20 083.6	1 870.2	662.7	2.2×	13 274.9	1 575.8	877.4	1.8×

load balancing with superior L2 cache performance. However, on synthetic dataset $|T| = 5 \times 10^4$ with larger motif length, $n = 1024$, the algorithm's speedup drops down to 8.6× due to reduction in cache performance owing to larger size of subsequences. In further work, authors planned to investigate the algorithm's scalability on many-core architectures but to the best of our knowledge, this research has not been extended.

This paper contributes in accelerating the MK algorithm on Intel MIC system. Having been designed with the OpenMP technology and matrix data structures to efficiently exploit vectorization capabilities of the MIC system, our algorithm provides high scalability, especially in the case of greater motif length.

6. CONCLUSIONS

In this paper, we addressed the task of accelerating the discovery of time series motifs on Intel MIC (Many Integrated Core) systems. A motif is a pair of subsequences of a longer time series, which are very similar to each other. Motif discovery is applied in a wide range of subject areas involving time series: medicine, biology, entertainment, weather prediction, and others.

We proposed a novel parallel algorithm for exact motif discovery on Intel MIC systems in the case of time series fit in the main memory. Our algorithm parallelizes the serial MK algorithm by Mueen et al. [10]. The parallelization makes use of thread-level parallelism and OpenMP technology and applies matrix layout for algorithm's data to efficiently vectorize calculations. In the experimental evaluation, the proposed algorithm showed high scalability, especially in the case of high computational load due to greater motif length.

In further studies, we plan to elaborate versions of the algorithm for other hardware platforms, namely GPU accelerators and cluster systems with nodes based on Intel MIC or GPU accelerators.

FUNDING

This work was financially supported by the Russian Foundation for Basic Research (grant no. 17-07-00463) and by the Ministry of Science and Higher Education of the Russian Federation (government orders 2.7905.2017/8.9 and 14.578.21.0265).

REFERENCES

1. D. F. Bacon, S. L. Graham, and O. J. Sharp, "Compiler transformations for high-performance computing," ACM Comput. Surv. **26**, 345–420 (1994). https://doi.org/10.1145/197405.197406
2. B. Y. Chiu, E. J. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs," in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, Aug. 24–27, 2003* (2003), pp. 493–498. https://doi.org/10.1145/956750.956808
3. G. Chrysos, "Intel registered Xeon Phi coprocessor (codename Knights Corner)," in *Proceedings of the 2012 IEEE Hot Chips 24th Symposium (HCS), Cupertino, CA, USA, Aug. 27–29, 2012* (2012), pp. 1–31. https://doi.org/10.1109/HOTCHIPS.2012.7476487
4. A. Goldberger, L. Amaral, L. Glass, J. Hausdorff, P. Ivanov, R. Mark, J. Mietus, G. Moody, C. Peng, and H. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals," Circulation **101** (23), 215–220 (2000). https://doi.org/10.1161/01.CIR.101.23.e215
5. P. Kostenetskiy and P. Semenikhina, "SUSU supercomputer resources for industry and fundamental science," in *Proceedings of the 2018 Global Smart Industry Conference (GloSIC)*, Chelyabinsk, Russia, Nov. 13–15, 2018 (2018), p. 8570068. https://doi.org/10.1109/GloSIC.2018.8570068
6. Ya. Kraeva and M. Zymbler, "Scalable algorithm for subsequence similarity search in very large time series data on cluster of Phi KNL," in *Proceedings of the 20th International Conference on Data Analytics and Management in Data Intensive Domains, DAMDID/RCDL 2018, Moscow, Russia, Oct. 9–12, 2018,* Commun. Comput. Inform. Sci. **1003**, 149–164 (2019). https://doi.org/10.1007/978-3-030-23584-0_9
7. T. Mattson, "Introduction to OpenMP," in *Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing, Nov. 11–17, 2006, Tampa, FL, USA* (ACM Press, 2006). https://doi.org/10.1145/1188455.1188673
8. J. Meng, J. Yuan, M. Hans, and Y. Wu, "Mining motifs from human motion," in *Proceedings of the Eurographics 2008, Crete, Greece, April 14–18, 2008* (Eurographics Association, 2008), pp. 71–74.
9. D. Minnen, C. L. Isbell, I. A. Essa, and T. Starner, "Discovering multivariate motifs using subsequence density estimation and greedy mixture learning," in *Proceedings of the 22nd AAAI Conference on Artificial Intelligence, July 22–26, 2007, Vancouver, British Columbia, Canada*(AAAI Press, 2007), pp. 615–620.
10. A. Mueen, E. J. Keogh, Q. Zhu, S. Cash, and M. B. Westover, "Exact discovery of time series motifs," in *Proceedings of the SIAM International Conference on Data Mining, SDM 2009, April 30–May 2, 2009, Sparks, Nevada, USA* (SIAM, 2009), pp. 473–484. https://doi.org/10.1137/1.9781611972795.41
11. A. Narang and S. Bhattacherjee, "Parallel exact time series motif discovery," in *Proceedings of the 16th International Euro-Par Conference, Ischia, Italy, Aug. 31–Sept. 3, 2010,* Lect. Notes Comput. Sci. **6272**, 304–315 (2010). https://doi.org/10.1007/978-3-642-15291-7_28
12. D. A. Padua, "POSIX threads (pthreads)," in *Encyclopedia of Parallel Computing* (Springer, Berlin, 2011), pp. 1592–1593. https://doi.org/10.1007/978-0-387-09766-4_447
13. P. Patel, E. J. Keogh, J. Lin, and S. Lonardi, "Mining motifs in massive time series databases," in *Proceedings of the 2002 IEEE International Conference on Data Mining ICDM 2002, Dec. 9–12, 2002, Maebashi City, Japan* (IEEE Comput. Soc., 2002), pp. 370–377. https://doi.org/10.1109/ICDM.2002.1183925
14. K. Pearson, "The problem of the random walk," Nature (London, U.K.) **72** (1865), 294 (1905). https://doi.org/10.1038/072342a0
15. J. Shieh and E. J. Keogh, "*i*SAX: indexing and mining terabyte sized time series," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, Aug. 24–27, 2008* (ACM, 2008), pp. 623–631. https://doi.org/10.1145/1401890.1401966
16. A. Sodani, "Knights Landing (KNL): 2nd generation Intel® Xeon Phi processor," in *Proceedings of the 2015 IEEE Hot Chips 27th Symposium HCS, Cupertino, CA, USA, Aug. 22–25, 2015* (IEEE, 2015), pp. 1–24. doi 10.1109/HOTCHIPS.2015.7477467
17. I. Sokolinskaya and L. Sokolinsky, "Revised pursuit algorithm for solving non-stationary linear programming problems on modern computing clusters with manycore accelerators," in *Proceedings of the 2nd Russian Conference Supercomputing Days, RuSCDays 2016, Moscow, Russia, Sept. 26–27, 2016,* Commun. Comput. Inform. Sci. **687**, 212–223. Springer (2016). https://doi.org/10.1007/978-3-319-55669-7_17
18. Y. Tanaka, K. Iwamoto, and K. Uehara, "Discovery of time-series motif from multi-dimensional data based on MDL Principle," Machine Learning **58**, 269–300 (2005). https://doi.org/10.1007/s10994-005-5829-2
19. D. R. Wilson, and T. R. Martinez, "Reduction techniques for instance-based learning algorithms," Machine Learning **38**, 257–286 (2000). https://doi.org/10.1023/A:1007626913721
20. M. Zymbler, A. Polyakov, and M. Kipnis, "Time series discord discovery on Intel many-core systems," in *Proceedings of the 13th International Conference, PCT 2019, Kaliningrad, Russia, April 2–4, 2019,* Commun. Comput. Inform. Science **1063**, 168–182 (2019). https://doi.org/10.1007/978-3-030-28163-2_12