Scalable Algorithm for Subsequence Similarity Search in Very Large Time Series Data on Cluster of Phi KNL

Yana Kraeva and Mikhail Zymbler(✉)

South Ural State University, Chelyabinsk, Russia
{kraevaya,mzym}@susu.ru

Abstract. Nowadays, subsequence similarity search under the Dynamic Time Warping (DTW) similarity measure is applied in a wide range of time series mining applications. Since the DTW measure has a quadratic computational complexity w.r.t. the length of query subsequence, a number of parallel algorithms for various many-core architectures have been developed, namely FPGA, GPU, and Intel MIC. In this paper, we propose a novel parallel algorithm for subsequence similarity search in very large time series data on computing cluster with nodes based on the Intel Xeon Phi Knights Landing (KNL) many-core processors. Computations are parallelized both at the level of all cluster nodes through MPI, and within a single cluster node through OpenMP. The algorithm involves additional data structures and redundant computations, which make it possible to effectively use Phi KNL for vector computations. Experimental evaluation of the algorithm on real-world and synthetic datasets shows that it is highly scalable.

Keywords: Time series · Similarity search · Dynamic Time Warping · Parallel algorithm · Cluster · OpenMP · MPI · Intel Xeon Phi · Knights Landing · Data layout · Vectorization

1 Introduction

Nowadays, time series are pervasive in a wide spectrum of applications with data intensive analytics, e.g. climate modelling [1], economic forecasting [21], medical monitoring [6], etc. Many time series analytical problems require subsequence similarity search as a subtask, which assumes the following. A query subsequence and a longer time series are given, and a subsequence of the time series should be found, whose similarity to the query is the maximum among all the subsequences.

Currently, Dynamic Time Warping (DTW) [3] is considered as the best similarity measure in most domains [5]. Since computation of DTW is time-consuming there are parallel algorithms for FPGA [25] and GPU [17] have been proposed.

Our research [10–13] addresses the task of accelerating similarity search with the Intel Xeon Phi many-core system, which can be considered as an attractive alternative to FPGA and GPU. Phi provides a large number of compute cores with 512-bit wide vector processing units. Phi is based on the Intel x86 architecture and supports the same

Y. Manolopoulos and S. Stupnikov (Eds.): DAMDID/RCDL 2018, CCIS 1003, pp. 149–164, 2019.
https://doi.org/10.1007/978-3-030-23584-0_9

programming methods and tools as a regular Intel Xeon. The first generation of Phi, Knights Corner (KNC) [4], is a coprocessor with up to 61 cores, which supports native applications and offloading of calculations from a host CPU. The second generation product, Knights Landing (KNL) [20], is a bootable processor with up to 72 cores, which runs applications only in native mode. In [11–13], we proposed CPU+Phi computational scheme for subsequence similarity search on Phi KNC. In [10], we changed such an approach for Phi KNL having implemented advanced data layout and computational scheme, which allow to efficiently vectorizing computations.

This paper is a revised extended version of [10]. We consider more complicated case of very large time series when computing cluster system of Phi KNL nodes is utilized for the similarity search. We propose an advanced parallel algorithm, called *PhiBestMatch*, which parallelizes computations both among cluster nodes (through MPI technology), and within a single cluster node (through OpenMP technology). We performed additional series of experiments, which showed good scalability of *PhiBestMatch*.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 gives formal statement of the problem. In Sect. 4, we present the proposed algorithm. We describe experimental evaluation of our algorithm in Sect. 5. Finally, Sect. 6 concludes the paper.

2 Related Work

In recent decade, parallel and distributed algorithms for subsequence similarity search under the DTW measure have been extensively developed for various hardware platforms.

In [26], a GPU-based implementation was proposed. The warping matrix is generated in parallel, but the warping path is searched serially. Since the matrix generation step and the path search step are split into two kernels, this leads to overheads for storage and transmission of the warping matrix for each DTW calculation.

In [17], GPU and FPGA implementations of subsequence similarity search were presented. The GPU implementation is based on the same ideas as [26]. The system consists of two modules, namely Normalizer (z-normalization of subsequences) and Warper (DTW calculation), and is generated by a C-to-VHDL tool, which exploits the fine-grained parallelism of the DTW. However, this implementation suffers from lacking flexibility, i.e. it must be recompiled if length of query is changed. In [25], authors proposed a framework for FPGA-based subsequence similarity search, which utilizes the data reusability of continuous DTW calculations to reduce the bandwidth and exploit the coarse-grain parallelism.

In [22], authors proposed subsequence similarity search on CPU cluster. Subsequences starting from different positions of the time series are sent to different nodes, and each node calculates DTW in the naïve way. In [23], authors accelerated subsequence similarity search with SMP system. They distribute different queries into different cores, and each subsequence is sent to different cores to be compared with different patterns in the naïve way. In both implementations, the data transfer becomes the bottleneck. In [18], authors proposed an approach to subsequence similarity search

on Apache Spark cluster. Time series is fragmented and fragments are shared among cluster nodes as files under HDFS (Hadoop Distributed File System). Each node processes in parallel as many fragments as its CPU cores where each core implements the UCR-DTW algorithm [15].

3 Notation and Problem Background

3.1 Definitions and Notation

A *time series* T is a sequence of real-valued elements: $T = (t_1, t_2, \ldots, t_m)$. Length of a time series T is denoted by $|T|$.

Given two time series, $X = (x_1, x_2, \ldots, x_m)$ and $Y = (y_1, y_2, \ldots, y_m)$, the *Dynamic Time Warping (DTW)* distance between X and Y is denoted by $DTW(X, Y)$ and defined as below.

$$DTW(X,Y) = d(m,m), d(i,j) = \left(x_i - y_j\right)^2 + \min\begin{cases} d(i-1,j) \\ d(i,j-1) \\ d(i-1,j-1) \end{cases}, \quad (1)$$
$$d(0,0) = 0, d(i,0) = d(0,j) = \infty, 1 \le i \le m, 1 \le j \le m.$$

In the formulas above, $\left(d_{ij}\right) \in \mathbb{R}^{m \times m}$ is considered as a *warping matrix* for the alignment of the two respective time series. A *warping path* is a contiguous set of warping matrix elements that defines a mapping between two time series. The warping path must start and finish in diagonally opposite corner cells of the warping matrix, the steps in the warping path are restricted to adjacent cells, and the points in the warping path must be monotonically spaced in time.

A *subsequence* $T_{i,k}$ of a time series T is its contiguous subset of k elements, which starts from position i: $T_{i,k} = (t_i, t_{i+1}, \ldots, t_{i+k-1})$, $1 \le i \le m - k + 1$. A set of all subsequences of T with length n is denoted by S_T^n. Let $N = |T| - n + 1 = m - n + 1$ denotes a number of subsequences in S_T^n.

Given a time series T and a time series Q as a user specified query where $m = |T| \gg |Q| = n$, the *best matching subsequence* $T_{i,n}$ meets the property

$$\exists T_{i,n} \in S_T^n \; \forall k \, DTW\left(Q, T_{i,n}\right) \le DTW\left(Q, T_{k,n}\right), 1 \le i, k \le m - n + 1. \quad (2)$$

In what follows, where there is no ambiguity, we refer to subsequence $T_{i,n}$ as C, as a candidate in match to a query Q.

3.2 The UCR-DTW Serial Algorithm

Currently, UCR-DTW [15] is the fastest serial algorithm of subsequence similarity search, which integrates a large number of algorithmic speedup techniques. Since our algorithm is based on UCR-DTW, we briefly describe its basic features.

Squared Distances. The *Euclidean distance (ED)* between two subsequences Q and C where $|Q| = |C|$, is defined as below.

$$ED(Q, C) = \sqrt{\sum_{i=1}^{n} (q_i - c_i)^2}. \tag{3}$$

Instead of use square root in DTW and ED distance calculation, it is possible to use the squares thereof since it does not change the relative rankings of subsequences.

Z-normalization. Both the query subsequence and each subsequence of the time series need to be z-normalized before the comparison [24]. The *z-normalization* of a time series T is defined as a time series $\hat{T} = (\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_m)$ where

$$\hat{t}_i = \frac{t_i - \mu}{\sigma}, \ \mu = \frac{1}{m}\sum_{i=1}^{m} t_i, \ \sigma^2 = \frac{1}{m}\sum_{i=1}^{2} t_i^2 - \mu^2. \tag{4}$$

Cascading Lower Bounds. Lower bound (LB) is an easy computable threshold of the DTW distance measure to identify and prune clearly dissimilar subsequences [5]. In what follows, we refer this threshold as the *best-so-far* distance (or *bsf* for brevity). If LB has exceeded *bsf*, the DTW distance will exceed *bsf* as well, and the respective subsequence is assumed to be clearly dissimilar and pruned without calculation of DTW. UCR-DTW initializes *bsf* as $+\infty$ and then scans the time series with sliding window and calculates *bsf* on the *ith* step as follows:

$$bsf_{(i)} = \min\left(bsf_{(i-1)}, \begin{cases} +\infty, LB(Q, T_{i,n}) > bsf_{(i-1)} \\ DTW(Q, T_{i,n}), otherwise \end{cases}\right). \tag{5}$$

UCR-DTW exploits three LBs, namely $LB_{Kim}FL$ [15], $LB_{Keogh}EC$, $LB_{Keogh}EQ$ [8] applying them in a cascade.

The $LB_{Kim}FL$ lower bound uses the distances between the First (Last) pair of points from C and Q as a lower bound, and defined as below.

$$LB_{Kim}FL(Q, C) := ED(\hat{q}_1, \hat{c}_1) + ED(\hat{q}_n, \hat{c}_n) \tag{6}$$

The $LB_{Keogh}EC$ lower bound is the distance from the closer of the two so-called envelopes of the query to a candidate subsequence, and defined as below.

$$LB_{Keogh}EC(Q, C) = \sum_{i=1}^{n} \begin{cases} (\hat{c}_i - u_i)^2, if \ \hat{c}_i > u_i \\ (\hat{c}_i - \ell_i)^2, if \ \hat{c}_i < \ell_i \\ 0, otherwise \end{cases}. \tag{7}$$

In the equation above, subsequences $U = (u_1, \ldots, u_n)$ and $L = (\ell_1, \ldots, \ell_n)$ are the *upper envelope* and *lower envelope* of the query, respectively, and defined as below.

$$u_i = \max_{i-r \le k \le i+r} \hat{q}_k, \ell_i = \min_{i-r \le k \le i+r} \hat{q}_k, \tag{8}$$

where the parameter $r\,(1 \le r \le n)$ denotes the Sakoe–Chiba band constraint [16], which states that the warping path cannot deviate more than r cells from the diagonal of the warping matrix.

The $LB_{Keogh}EQ$ lower bound is the distance from the query and the closer of the two envelopes of a candidate subsequence (i.e. the roles of the query and the candidate subsequence are reversed as opposed to $LB_{Keogh}EC$).

$$LB_{Keogh}EQ(Q, C) := LB_{Keogh}EC(C, Q). \tag{9}$$

Firstly, UCR-DTW calculates z-normalized version of the query and its envelopes, and *bsf* is assumed to be equal to infinity. Then the algorithm scans the input time series applying the cascade of LBs to the current subsequence. If the subsequence is not pruned, then DTW distance is calculated. Next, *bsf* is updated if it is greater than the value of DTW distance calculated above. By doing so, in the end, UCR-DTW finds the best matching subsequence of the given time series.

4 The *PhiBestMatch* Parallel Algorithm

In this section, we present a novel parallel algorithm for subsequence similarity search in very long time series on computing cluster of Phi KNL nodes, called *PhiBestMatch*. *PhiBestMatch* is based on the following ideas.

Computations are parallelized on two levels, namely at the level of all cluster nodes, and within a single cluster node. The time series is divided into equal-length partitions and distributed among cluster nodes. During the search in its own partition, each node communicates with rest nodes by functions of the MPI standard to improve local *bsf* and reduce the amount of computations.

Within a single cluster node, computations are performed by the thread-level parallelism and the OpenMP technology. 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 [2] of VPUs. We should avoid unaligned memory access since it can cause inefficient vectorization due to timing overhead for loop peeling [2]. Within a single cluster node, the algorithm involves additional data structures and redundant computations [10].

4.1 Partitioning of the Time Series

We partition the time series among cluster nodes as follows. Let F is a number of fragments and $T^{(k)}$ is k-th $(0 \le k \le F-1)$ partition of T, then $T^{(k)}$ is defined as a subsequence $T_{start,len}$ as below.

$$start = k \cdot \left\lfloor \frac{N}{F} \right\rfloor + 1,\ len = \begin{cases} \left\lfloor \frac{N}{F} \right\rfloor + (N \bmod F) + n - 1, k = F - 1 \\ \left\lfloor \frac{N}{F} \right\rfloor + n - 1, otherwise \end{cases}. \quad (10)$$

This means the head part of every partition except first overlaps with the tail part of the previous partition in $n - 1$ data points, where n is the query length. Such a technique prevents us from loss of the resulting subsequences in the junctions of two neighbor partitions.

4.2 Data Layout

We propose data layout aiming to provide organize computations over aligned data with as many auto-vectorizable loops as possible.

Given a subsequence C and VPU width w, we denote *pad length* as $pad = w - (n \bmod w)$ and define *aligned subsequence* $\tilde{T}_{i,n}$ as below:

$$\tilde{T}_{i,n} = \begin{cases} (t_i, t_{i+1}, \ldots, t_{i+n-1}, \underbrace{0, 0, \ldots, 0}_{pad}), \ if \ \ n \bmod w > 0 \\ (t_i, t_{i+1}, \ldots, t_{i+n-1}), \quad otherwise. \end{cases} \quad (11)$$

According to (1), $\forall Q, C : |Q| = |C| \ \ DTW(Q, C) = DTW\left(\tilde{Q}, \tilde{C}\right)$. Thus, in what follows, we will assume the aligned versions of the query and a subsequence of the input time series.

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.

$$S_T^n(i,j) := \tilde{t}_{i+j-1}. \quad (12)$$

Let us denote the number of LBs exploited by the algorithm as lb_{max} ($lb_{max} \geq 1$), and denote these LBs as $LB_1, LB_2, \ldots, LB_{lb_{max}}$, enumerating them according to the order in the lower bounding cascade. Given a time series T, we define the *LB-matrix* of all subsequences of length n from T, $L_T^n \in \mathbb{R}^{N \times lb_{max}}$ as below.

$$L_T^n(i,j) := LB_j\left(T_{i,n}, Q\right). \quad (13)$$

The *bitmap matrix* is a vector-column $B_T^n \in \mathbb{B}^N$, which for all subsequences of length n from T stores the logical conjunction of *bsf* and every LB:

$$B_T^n(i) := \bigwedge_{j=1}^{lb_{max}} \left(L_T^n(i,j) < bsf\right). \quad (14)$$

We establish the *candidate matrix* to store those subsequences from the S_T^n matrix, which have not been pruned after the lower bounding. The candidate matrix will be

processed in parallel by calculating of DTW distance measure between each row of the matrix and the query. Then the minimum of DTW distances is used as *bsf*.

To provide parallel calculations of the candidate matrix, we denote the *segment size of the matrix as* $s \in \mathbb{N}$ $(s \leq \frac{N}{p}$ where p is the number of threads employed by the parallel algorithm) and define the *candidate matrix*, $C_T^n \in \mathbb{R}^{(s \cdot p) \times (n + pad)}$ as below.

$$C_T^n(i, \cdot) := S_T^n(k, \cdot) : B_T^n(i) = TRUE. \tag{15}$$

In further experiments, we take the segment size $s = 100$.

4.3 Computational Scheme

Figure 1 depicts the *PhiBestMatch* pseudo-code, and Fig. 2 shows data structures of the algorithm. At initialization, the algorithm assigns the number of the current process to *myrank* by the MPI function. In what follows, each process deals with the subsequence matrix $S^n_{T^{(myrank)}}$ of the $T^{(myrank)}$ partition. The variable *bsf* is initialized by the DTW distance between the query and a random subsequence of the partition.

Algorithm PHIBESTMATCH
Input:
T time series to search
Q query subsequence
r warping constraint
Output:
bsf similarity of the best match subsequence
$bestmatch$ index of the best match subsequence

1: $myrank \leftarrow$ MPI_Comm_rank()
2: $N \leftarrow \left|N^{(myrank)}\right| - n + 1$
3: $subseq_{rnd} \leftarrow T^{(myrank)}_{random(1..N),n}$
4: $bsf \leftarrow$ DTW($subseq_{rnd}, Q, r, \infty$)
5: $myrank \leftarrow N$
6: PREPROCESS($T^{(myrank)}, Q, S_T^n, L_T^n$)
7: **repeat**
8: IMPROVE($T^{(myrank)}, bsf, bestmatch$)
9: $flagDone \leftarrow (processed = 0)$
10: $\{bsf, bestmatch\} \leftarrow$ MPI_Allreduce($\{bsf, bestmatch\}$, MPI_FLOAT_LONG, MPI_MIN)
11: $Stop \leftarrow$ MPI_Allreduce($myFlagDone$, MPI_BOOL, MPI_AND)
12: **until not** $Stop$
13: **return** $\{bsf, bestmatch\}$

Fig. 1. *PhiBestMatch* pseudo-code

Then we perform preprocessing by forming the subsequence matrix of the aligned subsequences, z-normalizing each subsequence, and calculating each LB of the lower bounding cascade. Strictly speaking, the latter step brings redundant calculations. In contrast, UCR-DTW calculates the next LB in the cascade only if a current subsequence

is not clearly dissimilar after the calculation of the previous LB. However, we perform precomputations once and parallelize them keeping in mind they further can be efficiently vectorized by the compiler since the absence of data dependencies in LBs.

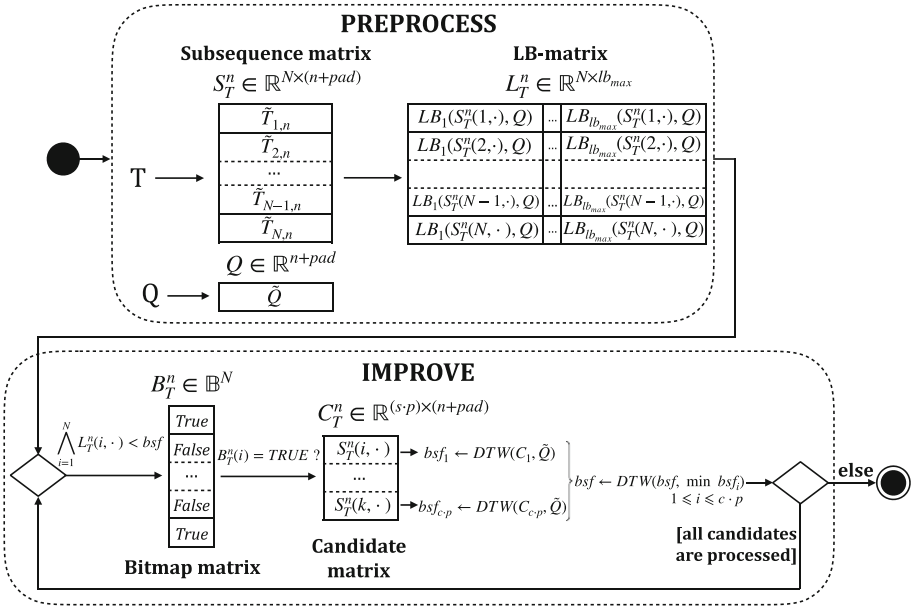


Fig. 2. Data flow of *PhiBestMatch*

After that, the algorithm improves the *bsf* threshold by the following loop until each node completes its partition. At first, the bitmap matrix is calculated in parallel based on the pre-calculated LB-matrix. Then each subsequence with TRUE in the respective element of the bitmap matrix is added to the candidate matrix. After the candidate matrix is filled, we calculate in parallel the DTW distance measure between each candidate and the query and find the minimum distance. If the minimum distance is less than *bsf* then *bsf* is updated. Then, we find the minimum value of *bsf* among all the partitions by the MPI_Allreduce global reduction operation. Finally, the latter operation is used to check if each node completes its partition.

5 Experiments

In order to evaluate the developed algorithm, we performed experiments on two platforms, namely a single cluster node and a whole cluster system.

5.1 Experimental Setup

Objectives. In the experiments on a single cluster node, we studied performance and scalability of the algorithm with respect to the r warping constraint and the n query

length. In the experiments on the cluster system, we studied the algorithm's scaled speedup with respect to the query length. Finally, we compare *PhiBestMatch* performance with analogous algorithm [18].

Measures. In the experiments, we investigated the algorithm's performance (measuring the run time after deduction of the I/O time) and scalability. We calculated the algorithm's speedup and parallel efficiency, which are defined as follows. *Speedup* and *parallel efficiency* of a parallel algorithm employing k threads are calculated, respectively, as

$$s(k) = \frac{t_1}{t_k}, e(k) = \frac{s(k)}{k}, \tag{16}$$

where t_1 and t_k are run times of the algorithm when one and k threads are employed, respectively.

In the experiments on the cluster system, we investigated *scaled speedup* of the parallel algorithm, which refers to linear increasing of the problem size proportionally with the number of computational nodes added to the system, and is calculated as follows:

$$s_{scaled} = \frac{p \cdot m}{t_{p(p \cdot m)}}, \tag{17}$$

where p is the number of nodes, m is the problem size, and $t_{p(p \cdot m)}$ is the algorithm's run time when a problem of size $p \cdot m$ is processed on p nodes.

Hardware. We performed our experiments on two supercomputers, namely Tornado SUSU [9] and NKS-1P [19] with the characteristics summarized in Table 1.

Table 1. Specifications of hardware

Specifications	Tornado SUSU		NKS-1P	
	Host	Node	Host	Node
Model, Intel Xeon	2 × X5680	Phi KNC, SE10X	2 × E5-2630v4	Phi KNL 7290
Physical cores	2 × 6	61	2 × 10	72
Hyper threading factor	2	4	2	4
Logical cores	24	244	40	288
Frequency, GHz	3.33	1.1	2.2	1.5
VPU width, bit	128	512	256	512
Peak performance, TFLOPS	0.371	1.076	0.390	3.456

For the experiments on a single node, we used the simplified version of *PhiBestMatch* [10], which treats the time series as one partition.

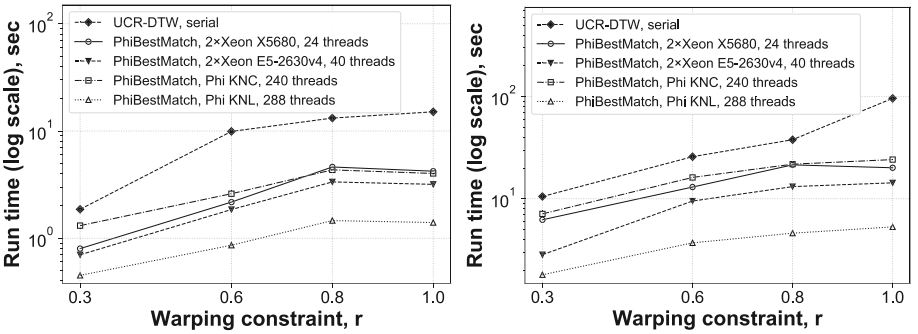
Datasets. In the experiments, we used datasets summarized in Table 2. RW-SN, RW-CS, and RW-SN are the datasets generated according to the Random Walk model [14]. The EPG (Electrical Penetration Graph) dataset is a series of signals, which was used by entomologists to study of Aster leafhopper (*macrosteles quadrilineatus*) behavior [17]. The ECG dataset [7] represents electrocardiogram signals digitized at 128 Hz.

Table 2. Datasets used in experiments

Platform	Dataset	Type	$\|T\| = m$	$\|Q\| = n$
Single cluster node	RW-SN	Synthetic	10^6	128
Single cluster node	EPG	Real	$2.5 \cdot 10^5$	360, 432, 512, 1024
Cluster system	RW-CS	Synthetic	$12.8 \cdot 10^7$	128, 512, 1024
Cluster system	ECG	Real	$12.8 \cdot 10^7$	432, 512, 1024
Cluster system	RW-SH	Synthetic	$2.2 \cdot 10^8$	128

5.2 Evaluation on a Single Cluster Node

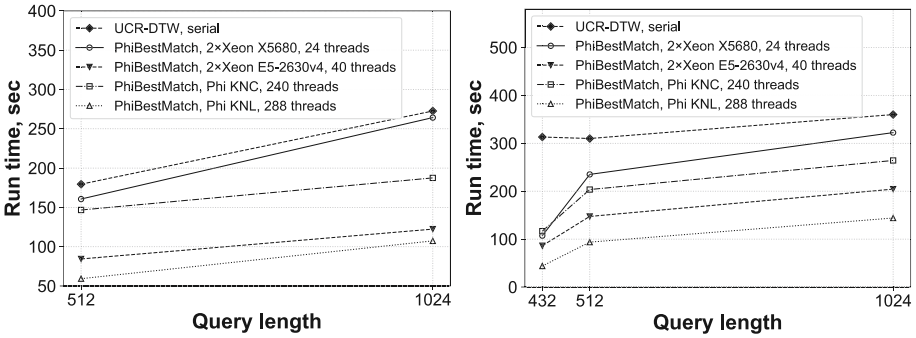
Figures 3 and 4 depict the performance of *PhiBestMatch* depending on r and n, respectively. As we can see, at lower values of the parameters (approximately, $0 < r \leq 0.5n$ and $n < 512$), the algorithm runs slightly faster or about the same way on two Intel Xeon host than on Intel Xeon Phi. At high values of the parameters ($0.5n < r \leq n$ and $n \geq 512$), the algorithm is faster on Intel Xeon Phi. It means that *PhiBestMatch* better utilizes vectorization capabilities of Intel Xeon Phi with greater computational load.



(a) RW-SN dataset, n=128 (b) EPG dataset, n=360

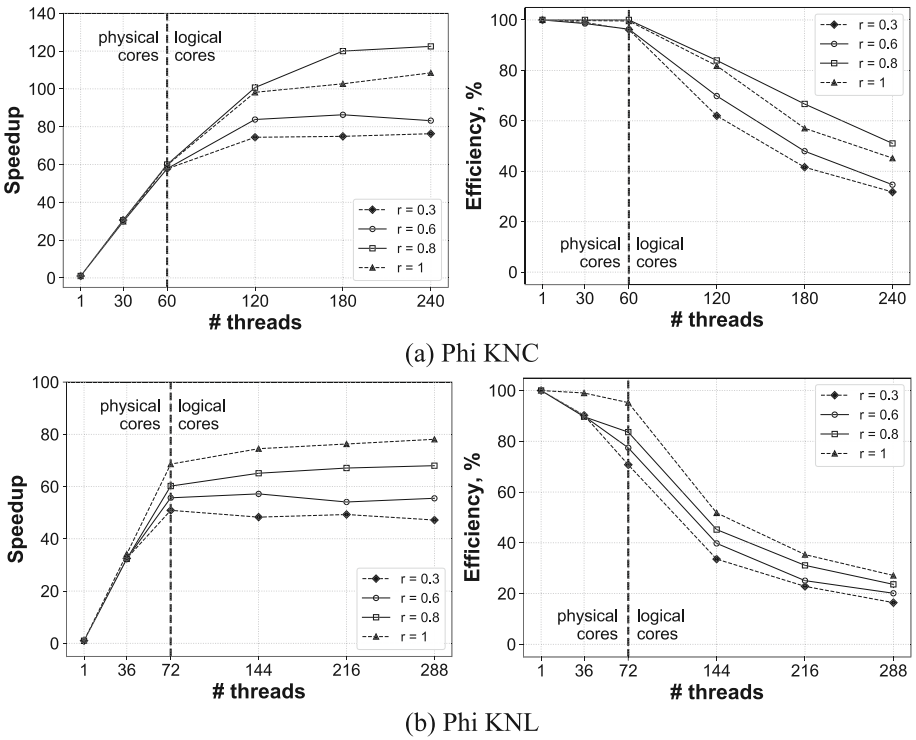
Fig. 3. *PhiBestMatch* performance w.r.t. the warping constraint

Figures 5 and 6 depict the experimental results on the synthetic (RW-SN) and the real (EPG) datasets, respectively. As we can see, *PhiBestMatch* shows speedup closer to linear and efficiency closer to 100%, if the number of threads matches the number of



(a) RW-SN dataset, r=0.8n (b) EPG dataset, r=0.8n

Fig. 4. *PhiBestMatch* performance w.r.t. the query length

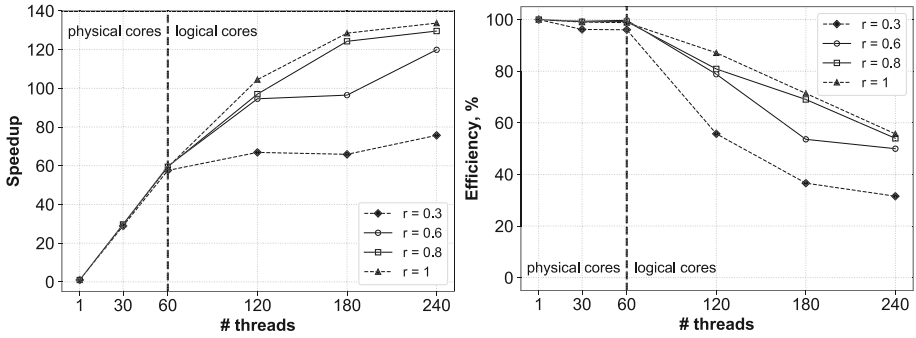


(a) Phi KNC

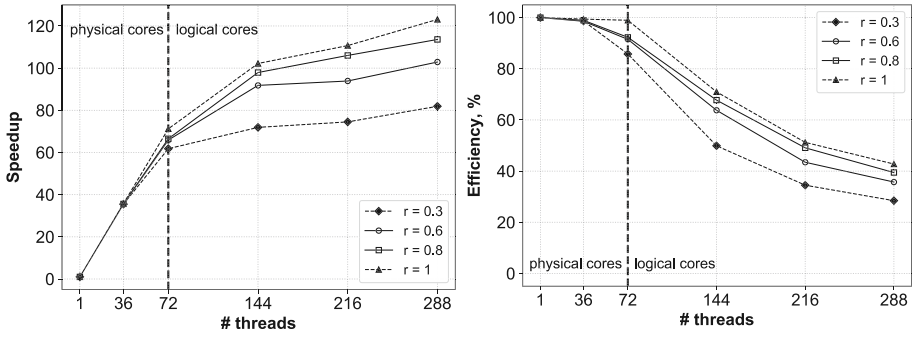
(b) Phi KNL

Fig. 5. *PhiBestMatch* speedup and parallel efficiency on synthetic data (RW-SN dataset)

physical cores the algorithm is running on. When more than one thread per physical core is used, speedup became sub-linear, and parallel efficiency decreases accordingly. The best speedup and efficiency are achieved when the r parameter ranges from 0.8 to 1 of n.



(a) Phi KNC



(b) Phi KNL

Fig. 6. *PhiBestMatch* speedup and parallel efficiency on real data (EPG dataset, $n = 360$)

5.3 Evaluation on a Cluster System

In the experiments studying *PhiBestMatch* scaled speedup, we utilized from 16 to 128 nodes of the Tornado SUSU supercomputer. We varied the query length while took the parameter $r = n$. Figures 7 and 8 depict the performance of *PhiBestMatch* on synthetic and real data, respectively.

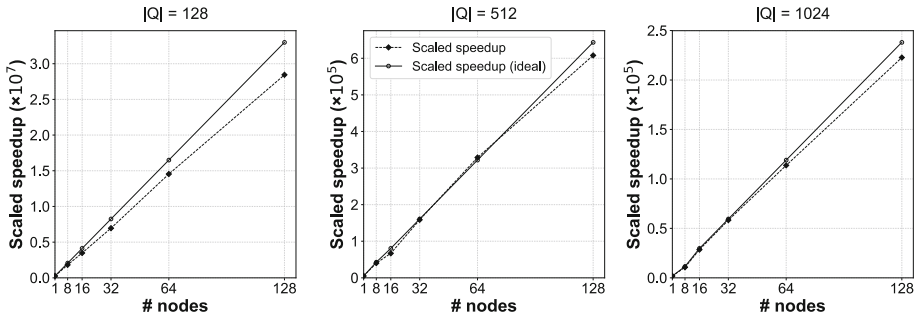


Fig. 7. *PhiBestMatch* scaled speedup on synthetic data (RW-CS dataset, $r = 0.8n$)

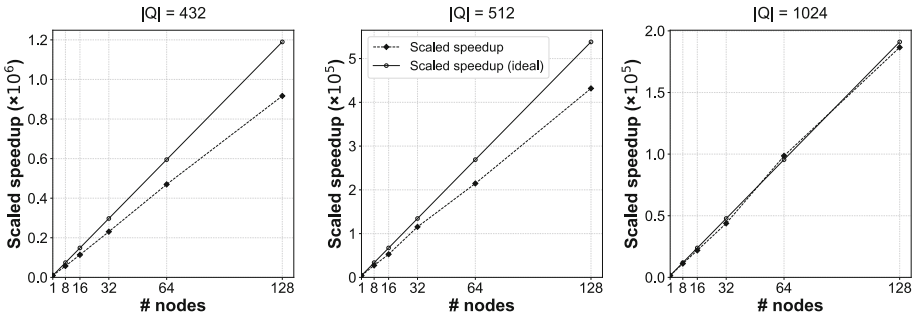


Fig. 8. *PhiBestMatch* scaled speedup on real data (ECG-CS dataset, $r = 0.8n$)

As we can see, *PhiBestMatch* shows closer to linear scaled speedup. At the same time, the similarity search for a subsequence of greater length demonstrates a higher scaled speedup, since it provides a greater amount of computations on a single node.

5.4 Comparison with Analogue

In [18], Shabib *et al.* presented the hybrid search algorithm, which exploits Apache Spark cluster of multi-core nodes. The algorithm was evaluated on six cluster nodes each with Intel Xeon E3-1200 (4-core at 3.1 GHz) CPU onboard for the RW-SH dataset with the parameter $r = 0.05n$. We compared the performance of Shabib *et al.* algorithm and *PhiBestMatch* performance on six nodes of Tornado SUSU for the same dataset and parameter r. Table 4 depicts the results.

Table 4. Performance of *PhiBestMatch* in comparison with Shabib *et al.* algorithm

PhiBestMatch, sec	Algorithm of Shabib *et al.*, sec
24.2	32

6 Conclusion

In this paper, we presented *PhiBestMatch*, a novel parallel algorithm for subsequence similarity search in very large time series data on computing cluster of the modern Intel Xeon Phi Knights Landing (Phi KNL) nodes. Phi KNL is many-core system with 512-bit wide vector processing units, which supports the same programming methods and tools as a regular Intel Xeon, and can be considered as an alternative to FPGA and GPU.

PhiBestMatch performs parallel computations on two levels, namely at the level of all cluster nodes, and within a single cluster node. The time series is divided into equal-length partitions and distributed among cluster nodes. During the search in its own partition, each node communicates with rest nodes by functions of the MPI standard to improve local best-so-far similarity threshold and reduce the amount of computations. Within a single cluster node, *PhiBestMatch* exploits the thread-level parallelism and the

OpenMP technology. The algorithm involves additional data structures, which are aligned in main memory, and redundant computations. Computations are organized with as many vectorizable loops as possible to provide the highest performance of Phi KNL.

We performed experiments on synthetic and real-word datasets, which showed good scalability of *PhiBestMatch*. Within a single cluster node, the algorithm demonstrates closer to linear speedup when the number of threads matches the number of Phi KNL physical cores the algorithm is running on. On the whole cluster system, *PhiBestMatch* showed close to linear scaled speedup. The algorithm better utilizes vectorization capabilities of Phi KNL with greater computational load (i.e. with longer query length and greater value of the Sakoe–Chiba band constraint).

Acknowledgments. This work was financially supported by the Russian Foundation for Basic Research (grant No. 17-07-00463), by Act 211 Government of the Russian Federation (contract No. 02.A03.21.0011) and by the Ministry of education and science of Russian Federation (government order 2.7905.2017/8.9). Authors thank The Siberian Branch of the Russian Academy of Sciences (SB RAS) Siberian Supercomputer Center (Novosibirsk, Russia) for the provided computational resources.

References

1. Abdullaev, S.M., Zhelnin, A.A., Lenskaya, O.Y.: The structure of mesoscale convective systems in central Russia. Russ. Meteorol. Hydrol. **37**(1), 12–20 (2012)
2. Bacon, D.F., Graham, S.L., Sharp, O.J.: Compiler transformations for high-performance computing. ACM Comput. Surv. **26**(4), 345–420 (1994). https://doi.org/10.1145/197405.197406
3. Berndt, D.J., Clifford, J.: Using dynamic time warping to find patterns in time series. In: Proceedings of the 1994 AAAI Workshop on Knowledge Discovery in Databases, Seattle, Washington, July 1994, pp. 359–370. AAAI Press (1994)
4. Chrysos, G.: Intel Xeon Phi coprocessor (codename Knights Corner). In: 2012 IEEE Hot Chips 24th Symposium (HCS), Cupertino, CA, USA, 27–29 August 2012, pp. 1–31. IEEE (2012). https://doi.org/10.1109/hotchips.2012.7476487
5. Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.: Querying and mining of time series data: experimental comparison of representations and distance measures. Proc. VLDB Endow. **1**(2), 1542–1552 (2008). https://doi.org/10.14778/1454159.1454226
6. Epishev, V., Isaev, A., Miniakhmetov, R., et al.: Physiological data mining system for elite sports. Bull. South Ural State Univ. Ser. Comput. Math. Softw. Eng. **2**(1), 44–54 (2013)
7. Goldberger, A.L., Amaral, L.A.N., Glass, L., Hausdorff, J.M., Ivanov, Pl.Ch., et al.: PhysioBank, PhysioToolkit, and PhysioNet. Circulation **101**(23), e215–e220 (2000). https://doi.org/10.1161/01.cir.101.23.e215
8. Keogh, E.J., Ratanamahatana, C.A.: Exact indexing of dynamic time warping. Knowl. Inf. Syst. **7**(3), 358–386 (2005). https://doi.org/10.1007/s10115-004-0154-9
9. Kostenetskiy, P., Semenikhina, P.: SUSU supercomputer resources for industry and fundamental science. In: GloSIC 2018, Proceedings of the Global Smart Industry Conference, Chelyabinsk, Russia, 13–15 November 2018, Article no. 8570068 (2018). https://doi.org/10.1109/glosic.2018.8570155

10. Kraeva, Ya., Zymbler, M.: An efficient subsequence similarity search on modern Intel many-core processors for data intensive applications. In: Proceedings of the 20th International Conference on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2018). CEUR Workshop Proceedings, Moscow, Russia, 9–12 October 2018, vol. 2277, pp. 143–151. CEUR-WS.org (2018)
11. Movchan, A.V., Zymbler, M.L.: Parallel algorithm for local-best-match time series subsequence similarity search on the Intel MIC architecture. Procedia Comput. Sci. **66**, 63–72 (2015). https://doi.org/10.1016/j.procs.2015.11.009%5d
12. Movchan, A.V., Zymbler, M.L.: Parallel implementation of searching the most similar subsequence in time series for computer systems with distributed memory. In: Sokolinsky, L., Starodubov, I. (eds.) PCT 2016, International Scientific Conference on Parallel Computational Technologies. CEUR Workshop Proceedings, Arkhangelsk, Russia, 29–31 March 2016, vol. 1576, pp. 615–628. CEUR-WS.org (2016)
13. Movchan, A., Zymbler, M.: Time series subsequence similarity search under dynamic time warping distance on the intel many-core accelerators. In: Amato, G., Connor, R., Falchi, F., Gennaro, C. (eds.) SISAP 2015. LNCS, vol. 9371, pp. 295–306. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25087-8_28
14. Pearson, K.: The problem of the random walk. Nature **72**(1865), 294 (1905). https://doi.org/10.1038/072342a0
15. Rakthanmanon, T., Campana, B.J.L., Mueen, A., Batista, G.E.A.P.A., Westover, M.B., et al.: Searching and mining trillions of time series subsequences under dynamic time warping. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Beijing, China, 12–16 August 2012, pp. 262–270. ACM, New York (2012). https://doi.org/10.1145/2339530.2339576
16. Sakoe, H., Chiba, S.: Dynamic Programming algorithm optimization for spoken word recognition. In: Waibel, A., Lee, K.-F. (eds.) Readings in Speech Recognition, pp. 159–165. Morgan Kaufmann Publishers Inc., San Francisco (1990)
17. Sart, D., Mueen, A., Najjar, W.A., Keogh, E.J., Niennattrakul, V.: Accelerating dynamic time warping subsequence search with GPUs and FPGAs. In: Proceedings of the 2010 IEEE International Conference on Data Mining, Sydney, Australia, 14–17 December 2010, pp. 1001–1006. IEEE Computer Society, Washington, DC (2010). https://doi.org/10.1109/icdm.2010.21
18. Shabib, A., Narang, A., Niddodi, C.P., et al.: Parallelization of searching and mining time series data using dynamic time warping. In: Proceedings of the 2015 International Conference on Advances in Computing, Communications and Informatics, Kochi, India, 10–13 August, 2015, pp. 343–348. IEEE (2015). https://doi.org/10.1109/icacci.2015.7275633
19. Siberian Supercomputing Centre of ICMMG SB RAS. http://www.sscc.icmmg.nsc.ru/hardware.html
20. Sodani, A.: Knights Landing (KNL): 2nd generation Intel Xeon Phi processor. In: 2015 IEEE Hot Chips 27th Symposium (HCS), Cupertino, CA, USA, 22–25 August 2015, pp. 1–24. IEEE (2015)
21. Sokolinskaya, I., Sokolinsky, L.: Revised pursuit algorithm for solving non-stationary linear programming problems on modern computing clusters with manycore accelerators. Commun. Comput. Inf. Sci. **687**, 212–223 (2016). https://doi.org/10.1007/978-3-319-55669-7_17
22. Srikanthan, S., Kumar, A., Gupta, R.: Implementing the dynamic time warping algorithm in multithreaded environments for real time and unsupervised pattern discovery. In: 2011 2nd International Conference on Computer and Communication Technology, Allahabad, India, 15–17 September 2011, pp. 394–398. IEEE (2015). https://doi.org/10.1109/iccct.2011.6075111

23. Takahashi, N., Yoshihisa, T., Sakurai, Y., Kanazawa, M.: A parallelized data stream processing system using dynamic time warping distance. In: 2009 International Conference on Complex, Intelligent and Software Intensive Systems, Fukuoka, Japan, 16–19 March 2009, pp. 1100–1105. IEEE (2009). https://doi.org/10.1109/cisis.2009.77
24. Tarango, J., Keogh, E.J., Brisk, P.: Instruction set extensions for dynamic time warping. In: Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis, Montreal, QC, Canada, 29 September–4 October 2013, pp. 18:1–18:10. IEEE (2013). https://doi.org/10.1109/codes-isss.2013.6659005
25. Wang, Z., Huang, S., Wang, L., Li, H., Wang, Y., et al.: Accelerating subsequence similarity search based on dynamic time warping distance with FPGA. In: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 11–13 February 2013, pp. 53–62. ACM, New York (2013). https://doi.org/10.1145/2435264.2435277
26. Zhang, Y., Adl, K., Glass, J.R.: Fast spoken query detection using lower-bound dynamic time warping on graphical processing units. In: 2012 IEEE International Conference on Acoustics, Speech and Signal Processing, Kyoto, Japan, 25–30 March 2012, pp. 5173–5176. IEEE (2012). https://doi.org/10.1109/icassp.2012.6289085