An Efficient Subsequence Similarity Search on Modern Intel Many-core Processors for Data Intensive Applications

© Yana Kraeva © Mikhail Zymbler
South Ural State University (National Research University),
Chelyabinsk, Russia
kraevaya@susu.ru mzym@susu.ru

Abstract. Many time series analytical problems arising in a wide spectrum of data intensive applications require subsequence similarity search as a subtask. Currently, Dynamic Time Warping (DTW) is considered as the best similarity measure in most domains. Since DTW is computationally expensive there are parallel algorithms have been developed for FPGA, GPU, and Intel Xeon Phi. In our previous work, we accelerated subsequence similarity search with Intel Xeon Phi's Knights Corner generation by CPU+Phi computational scheme. Such an approach needs significant changes for the Phi's second-generation product, Knights Landing (KNL), which is an independent bootable device supporting only native applications. In this paper, we present a novel parallel algorithm for subsequence similarity search in time series for the Intel Xeon Phi KNL many-core processor. In order to efficiently exploit vectorization capabilities of Phi KNL, the algorithm provides the sophisticated data layout and computational scheme. We performed experiments on synthetic and real-word datasets, which showed good scalability of the algorithm.

Keywords: time series, subsequence similarity search, dynamic time warping, parallel algorithm, OpenMP, vectorization, Intel Xeon Phi Knights Landing.

CEUR-WS.org/Vol-2277/paper26.pdf

1 Introduction

Nowadays, time series are pervasive in a wide spectrum of applications with data intensive analytics, e.g. climate modelling [1], economic forecasting [16], medical monitoring [6], etc. Many time series analytical problems require subsequence similarity search as a subtask, which assumes that a query subsequence and a longer time series are given, and we are to find a subsequence of the time series, whose similarity to the query is the maximum among all the subsequences.

At the present time, Dynamic Time Warping (DTW) [3] is considered as the best time series subsequence similarity measure in most domains [5], since it allows the subsequence to have some stretching, shrinking, warping, or different length in comparison to the query. Since computation of DTW is time-consuming ($O(n^2)$ where n is length of the query sequence) there are speedup techniques have been proposed including algorithmic developments (indexing methods [7], early abandoning strategies, embedding and computation reuse strategies [13], etc.) and parallel hardware-based solutions for FPGA [19] and GPU [14], and Intel Xeon Phi [21].

In this paper, the Intel Xeon Phi many-core system is the subject of our efforts. Architecture of Phi provides a large number of compute cores with a high local memory bandwidth and 512-bit wide vector processing units. Being based on the Intel x86 architecture, Phi supports thread-level parallelism and the same programming tools as a regular Intel Xeon CPU and serves as an attractive alternative to FPGA and GPU. Now, Intel offers two generations of Phi products, namely Knights Corner (KNC) [4] and Knights Landing (KNL) [15]. The former is a coprocessor with up to 61 cores, which supports native applications as well as offloading of calculations from a host CPU. The latter provides up to 72 cores and as opposed to predecessor is an independent bootable device, which runs applications only in native mode.

In our previous works [9] and [21], we accelerated subsequence similarity search on Phi KNC by means of the CPU+Phi computational scheme. Such an approach needs significant changes for Phi KNL because if there is no CPU, in addition to parallelization, we have to efficiently vectorize computations in order to achieve high performance.

In this paper, we address the accelerating subsequence similarity search on Phi KNL for the case when time series involved in the computations fit in main memory. The paper makes the following basic contributions. We developed a novel parallel algorithm of subsequence similarity search in time series for the Intel Xeon Phi KNL processor. The algorithm efficiently exploits vectorization capabilities of Phi KNL by means of the sophisticated data layout and computational scheme. We performed experiments on real-word datasets, which showed good scalability of the algorithm.

The rest of the paper is organized as follows. Section 2 discusses related works. Section 3 gives formal notation and statement of the problem. In Section 4, we present the proposed parallel algorithm. We describe experimental evaluation of our algorithm in Section 5. Finally, in Section 6, we summarize the results obtained and propose directions for further research.

Proceedings of the XX International Conference "Data Analytics and Management in Data Intensive Domains" (DAMDID/RCDL'2018), Moscow, Russia, October 9-12, 2018

2 Related works

The problem of subsequence similarity search under the DTW measure has been extensively studied in recent decade. Since computation of DTW is time-consuming there are speedup techniques have been proposed.

Algorithmic developments include indexing methods [7], early abandoning strategies, embedding and computation reuse strategies [13], etc. Despite the fact these techniques focus on reduction of the number of calls DTW calculation procedure, computation of DTW distance measure still takes up to 80 percent of the total run time of the similarity search [20]. Due to this reason a number of parallel hardware-based solutions have been developed.

In [17], 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 [18], 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 [20], 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 [14], GPU and FPGA implementations of subsequence similarity search were presented. The GPU implementation is based on the same ideas as [20]. 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 [19], 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.

The aforementioned developments, however, cannot be directly applied to x86-based many-core systems, Intel Xeon Phi KNC [4] and Intel Xeon Phi KNL [15].

In our previous works [9] and [21], we accelerated subsequence similarity search on the Intel Xeon Phi KNC many-core coprocessor. A naïve approach when all the subsequences of a time series are simply distributed across the threads of both CPU and Phi for calculation of DTW distance measure, gave unsatisfactory performance and speedup. This was because of insufficient workload of the coprocessor. We proposed the CPU+Phi computational scheme where the coprocessor is exploited only for DTW distance measure computations whereas CPU performs lower bounding and prepares subsequences for the coprocessor. CPU supports a queue of candidate subsequences and offloads it to the coprocessor in order to compute DTW. Such a scheme significantly outperformed naïve approach.

This development, however, cannot be directly applied to Phi KNL since it is an independent bootable many-core processor and supports only native applications. Thus, our previous approach needs significant changes. Moreover, in order to achieve high performance of similarity search on Phi KNL, in addition to parallelization, we should provide data layout and computational scheme that allow exploiting vectorization capabilities of the many-core processor thereof in the most efficient way.

3 Notation and problem background

3.1 Definitions and notations

Definition 1. 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 $|T|$.

Definition 2. 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[5].

$$DTW(X,Y) = d(m,m) \tag{1}$$

$$d(i,j) = |x_i - y_j| + min\begin{cases} d(i-1,j) \\ d(i,j-1) \\ d(i-1,j-1) \end{cases} \tag{2}$$

$$d(0,0) = 0; d(i,0) = d(0,j) = \infty; i = j = 1, \ldots, m. \tag{3}$$

In (2), a set of $d(i,j)$ is considered as an $m \times m$ *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 the 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.

Definition 3. A *subsequence* $T_{i,k}$ of a time series T is its contiguous subset of k elements, which starts from position i: $T_{i,k} = t_i, t_{i+1}, \ldots, t_{i+k-1}, 1 \le i \le m-k+1$.

Definition 4. 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

$$\forall k\ DTW(Q, T_{i,n}) \le DTW(Q, T_{k,n}), \quad 1 \le i, k \le m-n+1 \tag{4}$$

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.

Definition 5. 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

$$\hat{t}_i = \frac{t_i - \mu}{\sigma} \tag{5}$$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} t_i \tag{6}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} t^2{}_i - \mu^2 \tag{7}$$

Definition 6. The *Euclidean distance (ED)* between two (z-normalized) subsequences Q and C where $|Q| = |C|$, is defined as below.

[5] Strictly speaking, DTW allows comparing two time series of different lengths. However, for the sake of simplicity, we assume time series of equal lengths due to it is possible without losing the generality [12].

$$ED(Q,C) = \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2} \quad (8)$$

3.2 Serial algorithm

Currently, UCR-DTW [11] 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, in this section, we briefly describe its basic features that are essentially exploited in our approach.

Squared distances. Instead of use square root in DTW and ED distance calculation, it is possible to use the squares thereof. Since both functions are monotonic and concave, it does not change the relative rankings of subsequences. In what follows, where there is no ambiguity, we will still use DTW and ED assuming the squared versions of them.

Z-normalization. Both the query subsequence and each subsequence of the time series need to be z-normalized before the comparison. Z-normalization shifts and scales the time series such that the mean is zero and the standard deviation is one.

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) due to UCR-DTW tries to improve (decrease) it while scanning the time series. If the lower bound 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 exploits the following LBs, namely $LB_{Kim}FL$ [11], $LB_{Keogh}EC$ and $LB_{Keogh}EQ$ [7], and they are applied 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.

$$LB_{Kim}FL(Q,C) := ED(\hat{q}_1, \hat{c}_1) + \mathrm{ED}(\hat{q}_n, \hat{c}_n) \quad (9)$$

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.

$$LB_{Keogh}EC(Q,C) := \sqrt{\sum_{i=1}^{n} \begin{cases} (\hat{c}_i - u_i)^2, if\ \hat{c}_i > u_i \\ (\hat{c}_i - \ell_i)^2, if\ \hat{c}_i < \ell_i \\ 0, otherwise \end{cases}} \quad (10)$$

In (10), subsequence $U = u_1, \ldots, u_n$ and subsequence $L = \ell_1, \ldots, \ell_n$ are the *upper envelope* and *lower envelope* of the query, respectively, and defined as below.

$$u_i = \max_{i-r \le k \le i+r} \hat{q}_k \quad (11)$$

$$\ell_i = \min_{i-r \le k \le i+r} \hat{q}_k \quad (12)$$

where the parameter r $(1 \le r \le n)$ denotes the Sakoe–Chiba band constraint [5], 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. That is, as opposed to $LB_{Keogh}EC$, the roles of the query and the candidate subsequence are reversed:

$$LB_{Keogh}EQ(Q,C) := LB_{Keogh}EC(C,Q) \quad (13)$$

Finally, UCR-DTW performs as follows. Firstly, z-normalized version of the query and its envelopes are calculated, 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 Method

In this section, we present a novel computational scheme and data layout, which allow efficient parallelization and vectorization of subsequence similarity search on Intel Xeon Phi KNL.

Vectorization plays the key role for getting the high performance of computations with parallel architectures [2]. This means a compiler's ability to transform the loops into sequences of vector operations and utilize VPUs. Thus, in order to provide the high performance of subsequence similarity search on Phi KNL, we should organize computations with as many auto-vectorizable loops as possible.

However, many auto-vectorizable loops are not enough for the overall good performance of an algorithm. Unaligned memory access is one of the basic factors that can cause inefficient vectorization due to timing overhead for loop peeling [2]. If the start address of the processed data is not aligned by the VPU width (i.e. by the number of floats that can be loaded in VPU), the loop is split into three parts by the compiler. The first part of iterations, which access the memory from the start address to the first aligned address is peeled off and vectorized separately. The rest part of iterations from the last aligned address to the end address is split and vectorized separately as well.

According to this argumentation, we propose a data layout, which provides an aligned access to subsequences of the time series. Applying this data layout, we also propose the respective computational scheme where as many computations as possible are implemented as auto-vectorizable for-loops.

4.1 Data layout

Firstly, we provide alignment of the processed subsequences.

Definition 7. Given a subsequence C and VPU width w, we denote *pad length* as $pad = w - (n \bmod w)$ and define *alignment* of C as a subsequence $\tilde{C}$ as below.

$$\tilde{C} := \begin{cases} c_1, c_2, \ldots, c_n, \underbrace{0, 0, \ldots, 0}_{pad}, if\ n\ mod\ w > 0 \\ C,\ otherwise \end{cases} \quad (14)$$

According to Def. 2, $\forall Q, C\colon |Q| = |C|\ DTW(Q,C) = DTW(\tilde{Q}, \tilde{C})$. Thus, in what follows, we still write Q and C for the query subsequence and a subsequence of the input time series, assuming, however, the aligned versions thereof. Alignment of subsequences allows to avoid timing overheads for loop peeling.

Next, we store all the (aligned) subsequences of a time series as a matrix in order to provide auto-vectorization of computations.

Definition 8. Let us denote the number of all the

subsequences of length n of a time series T as N, $N = |T| - n + 1 = m - n + 1$. We define the *subsequence matrix* (i.e. the matrix of all aligned subsequences of length n from a time series T), $S_T^n \in \mathbb{R}^{N \times (n+pad)}$ as below.

$$S_T^n(i,j) := \tilde{t}_{i+j-1} \quad (15)$$

We also establish two more matrices regarding lower bounding of all subsequences of the input time series. The first matrix stores for each subsequence the values of all the LBs in the cascade (cf. Sect. 3.2). The second one is bitmap matrix, which stores for each subsequence the result of comparison of bsf with every LB.

Definition 9. Let us denote the number of LBs exploited by the subsequence similarity search algorithm as lb_{max} and enumerate 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.

$$L_T^n(i,j) := LB_j(T_{i,n}) \quad (16)$$

Definition 10. Given a time series T, we define the *bitmap matrix* of all subsequences of length n from T, $B_T^n \in \mathbb{B}^{N \times lb_{max}}$ as below.

$$B_T^n(i,j) := L_T^n(i,j) < bsf \quad (17)$$

Finally, we establish a matrix to store candidate subsequences, i.e. those subsequences from the S_T^n matrix, which have not been pruned after the lower bounding. This matrix will be processed in parallel by calculating of DTW distance measure between each row representing the subsequence and the query.

Definition 11. Let us denote the number of threads employed by the parallel algorithm as p $(p \geq 1)$. Given the *segment size* $s \in \mathbb{Z}_+$ $(s \leq \lceil \frac{N}{p} \rceil)$, we define the *candidate matrix*, $C_T^n \in \mathbb{R}^{(s \cdot p) \times (n+pad)}$ as below.

$$C_T^n(i,\cdot) := S_T^n(k,\cdot): \quad \forall j, 1 \leq j \leq lb_{max}, \; B_T^n(k,j) = 1 \quad (18)$$

4.1 Computational scheme

We are finally in a position to introduce our approach. Figure 1 depicts the overall scheme of the algorithm.

The algorithm performs as follows. Firstly, the lower envelope and the upper envelope of the query are calculated, and the query is z-normalized. Then, for each subsequence in the subsequence matrix, all the LBs of the lower bounding cascade are preliminarily calculated (as well as z-normalized version of each subsequence). Additionally, in order to start the search, the algorithm initializes bsf by calculating the DTW distance measure between the query and first subsequence.

After that, the algorithm carries out the following loop as long as there are subsequences that have not been pruned during the search. At first, lower bounding is performed and the bitmap matrix is calculated in parallel. Next, promising candidates are added to the candidate matrix in serial mode. Then, the DTW distance measure between the query and each subsequence of the candidate matrix is calculated in parallel, and minimal distance is found. By the end of loop, we output the index of the subsequence with minimal distance measure. Below, we describe these steps in detail.

```
Algorithm PHIBESTMATCH
  Input:
    T      time series to search
    Q      query subsequence
    r      warping constraint
    p      number of threads employed
    s      segment size
  Output:
    bsf    similarity of the best match subsequence
  Returns:
    index of the best match subsequence
1:  CALCENVELOPE(Q, r, U, L)
2:  CALCLOWERBOUNDS(S_T^n, Q, r, L_T^n)
3:  bsf ← UCRDTW(T_{1,n}, Q, r, ∞)
4:  num_cand ← N
5:  while num_cand > 0 do
6:    LOWERBOUNDING(L_T^n, bsf, B_T^n)
7:    num_cand ← FILLCANDMATR(S_T^n, B_T^n, C_T^n, p, s)
8:    if num_cand > 0 then
9:      bestmatch ← CALCCANDMATR(C_T^n,
10:         num_cand, r, p, bsf)
11: return bestmatch
```

Figure 8 Overall computational scheme

Calculation of LBs. Figure 2 depicts the pseudo-code for calculation of lower bounds.

```
Algorithm CALCLOWERBOUNDS
  Input:
    S_T^n  subsequence matrix
    Q      query subsequence
    r      warping constraint
    p      number of threads employed
  Output:
    L_T^n  LB-matrix
1:  #pragma omp parallel for num_threads(p)
2:  for i from 1 to N do
3:    ZNORMALIZE(S_T^n(i,·))
4:    L_T^n(i,1) ← LB_KIMFL(Q, S_T^n(i,·))
5:    L_T^n(i,2) ← LB_KEOGHEC(Q, S_T^n(i,·))
6:    CALCENVELOPE(S_T^n(i,·), r, U, L)
7:    L_T^n(i,3) ← LB_KEOGHEQ(S_T^n(i,·), Q, U, L)
```

Figure 9 Calculation of lower bounds

Strictly speaking, this step brings redundant calculations to our algorithm. In contrast, UCR-DTW calculates the next LB in the cascade only if a current subsequence is clearly dissimilar after the calculation of the previous LB. As opposed to UCR-DTW, we calculate all the LBs and z-normalized versions for all the subsequences because of the following reasons. Firstly, it is possible to perform such computations once and before the scanning of all the subsequences. Secondly, these computations are parallelizable in a simple way based on the data parallelism paradigm. Finally, being combined, these computations can

be efficiently vectorized by the compiler.

Lower bounding. Figure 3 depicts the pseudo-code for lower bounding of subsequences. The algorithm performs lower bounding by scanning the LB-matrix and calculating the respective row of the bitmap matrix. In the next step of the algorithm, a subsequence corresponding to the row of the bitmap matrix where each element equals to one, will be added to the candidate matrix in order to further calculate DTW distance measure.

Algorithm LOWERBOUNDING
Input:
L_T^n LB-matrix
bsf best-so-far similarity distance
p number of threads employed
Output:
B_T^n bitmap matrix

```
1: #pragma omp parallel num_threads(p)
2: whoami ← omp_get_thread_num()
3: for i from pos_whoami to ⌈N / (whoami·p)⌉ do
4:   for j from 1 to lb_max do
5:     B_T^n(i,j) ← L_T^n(i,j) < bsf
```

Figure 10 Lower bounding of subsequences

During the search, we perform many scans of the LB-matrix in parallel as long as subsequences that are not clearly dissimilar exist. Parallel processing is based on the following technique. The LB-matrix is logically divided into p equal-sized segments, and each thread scans its own segment. In order to avoid scanning of each segment from scratch, we establish the segment index as an array of p elements where each element keeps the index of the most recent candidate subsequence in the respective segment, i.e. $SegIndex = (pos_1, \ldots, pos_p)$ where

$$pos_i := \begin{cases} 0, bsf = \infty \\ k: p \cdot (i-1) + 1 \le k \le \left\lceil \frac{N}{i \cdot p} \right\rceil \wedge \\ \quad \forall j, 1 \le j \le lb_{max}, \\ \quad LB_T^n(k,j) < bsf, otherwise \end{cases} \tag{19}$$

The candidate matrix filling. Figure 4 depicts the pseudo-code for the procedure of filling the candidate matrix.

The algorithm performs scanning the bitmap matrix along the segments. We start scanning not from the beginning of a segment but from the respective segment's index, which stores the number of the most recent candidate subsequence in the segment. If a subsequence is promising, it is added to the candidate matrix.

In order to output index of the best match subsequence, we establish the candidate subsequence index as an array of $s \cdot p$ elements where each element keeps the starting position of a candidate subsequence in the input time series, i.e. $Index = (idx_1, \ldots, idx_{s \cdot p})$ where

$$idx_i := k:\ 1 \le k \le m - n + 1\ \wedge \exists S_T^n(i,\cdot) \Leftrightarrow \exists T_{i,n} \Leftrightarrow\ k = (i-1) \cdot n + 1 \tag{20}$$

Algorithm FILLCANDMATR
Input:
S_T^n subsequence matrix
B_T^n bitmap matrix
p number of threads employed
s segment size
Output:
C_T^n candidate matrix
Returns:
number of subsequences added to the candidate matrix

```
1:  num_cand ← 0
2:  for i from 1 to p do
3:    for k from 1 to s do
4:      if ∧_{j=1}^{lb_max} B_T^n(pos_i + k, j) = 1 then
5:        if num_cand < s · p then
6:          num_cand ← num_cand + 1
7:          pos_i ← pos_i + k
8:          C_T^n(num_cand,·) ← S_T^n(pos_i,·)
9:          idx_{num_cand} ← (pos_i − 1) · n + 1
10:       else
11:         break
12:   if num_cand = s · p then
13:     break
14: return num_cand
```

Figure 11 The candidate matrix filling

Processing of the candidate matrix. Figure 5 depicts the pseudo-code for calculating DTW distance measure of candidate subsequences.

Algorithm CALCCANDMATR
Input:
C_T^n candidate matrix
num_{cand} number of candidate subsequences
Q query subsequence
r warping constraint
p number of threads employed
Output:
bsf similarity of the best-so-far subsequence
Returns:
index of the best-so-far subsequence

```
1: #pragma omp parallel for num_threads(p)
2:   shared (bsf, idx) private (distance)
3: for i from 1 to num_cand do
4:   distance ← UCRDTW(C_T^n(i,·), Q, r, bsf)
5:   #pragma omp critical
6:   if bsf > distance then
7:     bsf ← distance
8:     bestmatch ← idx_i
9: return bestmatch
```

Figure 12 Processing of the candidate matrix

The algorithm performs as follows. For each row of the candidate matrix, we calculate DTW distance measure between the respective candidate and the query by means of the UCR-DTW algorithm. If this distance is less than bsf then bsf is updated. The loop is parallelized by means of the OpenMP pragma where bsf is indicated as a variable shared across all threads while the distance variable is indicated as a private for each thread. In order to correctly update the shared variable, we use pragma with critical section.

5 Experiments

5.1 Experimental setup

Objectives. In the experiments, we compared the performance of our algorithm in comparison with UCR-DTW. We also evaluated the scalability of our algorithm on Intel Xeon Phi for different datasets. We measured the run time (after deduction of I/O time) and calculated the algorithm's speedup and parallel efficiency. Here we understand these characteristics of parallel algorithm scalability as follows. Speedup and parallel efficiency of a parallel algorithm employing p threads are calculated, respectively, as $s(p) = \frac{t_1}{t_p}$ and $e(p) = \frac{s(p)}{p}$, where t_1 and t_p are the run times of the algorithm when one and p threads are employed, respectively.

Hardware. We performed our experiments on a node of the Tornado SUSU supercomputer [8] with the characteristics summarized in Table 1.

Table 1 Specifications of hardware

Specification	Xeon Phi	2×Xeon CPU
Model, Intel	SE10X	X5680
# physical cores	61	2×6
Hyper threading	4×	2×
# logical cores	244	24
Frequency, GHz	1.1	3.33
VPU width, bit	512	128
Peak performance, TFLOPS	1.076	0.371
RAM, Gb	8	8

Datasets. In the experiments, we used the following datasets (cf. Table 2). The Random Walk is a dataset that was synthetically generated according to the model of the same name [10] and used in [11]. The EPG dataset [14] is a set of signals from the so-called Electrical Penetration Graph reflecting the behavior of the Aster leafhopper (*macrosteles quadrilineatus*). A critical task that researchers perform is to search for patterns in such time series, because in the US agriculture, for one state and one crop, this insect causes losses more than two million dollars a year [14].

Table 2 Specifications of datasets

Dataset	Type	m	n
Random Walk	Synthetic	10^6	128
EPG	Real	$2.5 \cdot 10^5$	360

5.2 Results and discussion

Figure 6 depicts the performance of PHIBESTMATCH in comparison with the UCR-DTW algorithm. As we can see, our algorithm is two times faster for both EPG and Random Walk datasets when UCR-DTW runs on one CPU core and PHIBESTMATCH runs on 240 cores of Intel Xeon Phi. Being run on Intel Xeon Phi, UCR-DTW is obviously slower than PHIBESTMATCH (from 10 to 15 times).

Figure 7 and Figure 8 depict the experimental results on the Random Walk dataset and the EPG dataset, respectively. On Random Walk data, both speedup and parallel efficiency are linear when the number of threads matches of physical cores the algorithm is running on. However, when more than one thread per physical core is used, speedup became sub-linear, and efficiency decreases accordingly. That is, speedup stops increasing from 80× when 120 threads are employed, and efficiency drops from 60 percent for 120 threads to 30 percent for 240 threads.

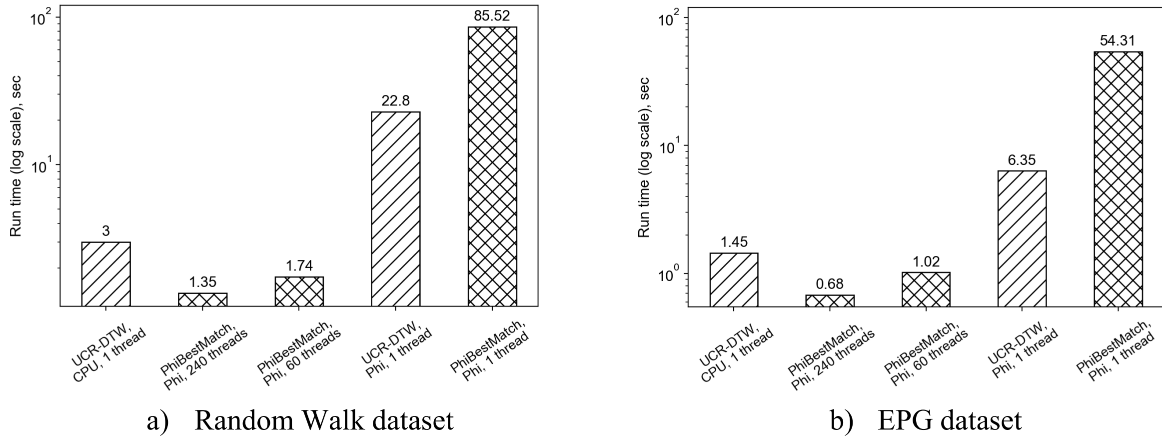
On EPG data, the algorithm shows closer to linear speedup and efficiency (up to 50× and at least 80 percent, respectively) if the number of threads employed is up to the number of physical cores. When more than one thread per physical core is used, speedup slowly increases up to 78×, and efficiency drops accordingly (similar to the picture for the Random Walk dataset).

We can conclude that the proposed algorithm demonstrates closer to linear scalability when the number of threads it runs on is up to the number of physical cores of the Intel Xeon Phi many-core processor. However, when more than one thread per physical core is used, speedup and parallel efficiency decrease significantly.

There are two possible reasons for this. Firstly, our algorithm is not completely parallel, and the candidate matrix filling is its serial part, which limits speedup. Secondly, according to its nature, DTW calculations (cf. Def. 2) can hardly ever be auto-vectorized. Thus, if during the (seamlessly auto-vectorizable) lower bounding step many subsequences have not been pruned as clearly dissimilar then they will be processed by many threads in the DTW calculation step but without auto-vectorization as it might be expected.

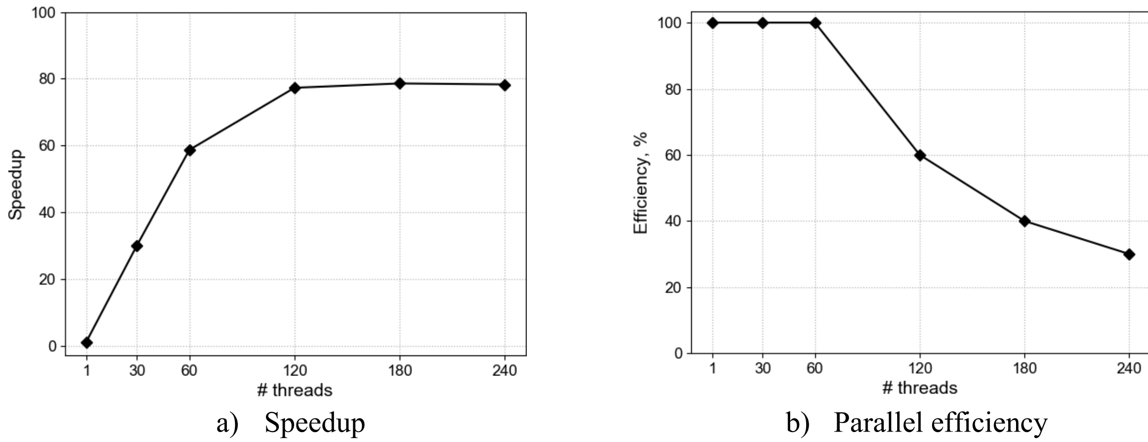
6 Conclusion

In this paper, we address the problem of accelerating subsequence similarity search on the modern Intel Xeon Phi system of Knights Landing (KNL) generation. Phi KNL is an independent bootable device, which provides up to 72 compute cores with a high local memory bandwidth and 512-bit wide vector processing units. Being based on the x86 architecture, the Intel Phi KNL many-core processor supports thread-level parallelism and the same programming tools as a regular Intel Xeon CPU, and serves as an attractive alternative to FPGA and GPU. We consider the case when time series involved in the computations fit in main memory.



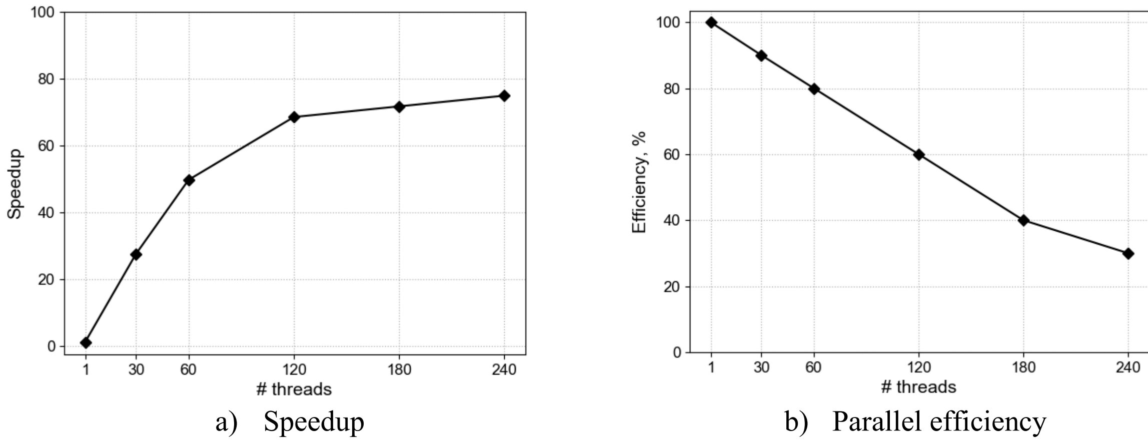
a) Random Walk dataset b) EPG dataset

Figure 13 Algorithm's performance



a) Speedup b) Parallel efficiency

Figure 14 Algorithm's scalability on the Random Walk dataset



a) Speedup b) Parallel efficiency

Figure 15 Algorithm's scalability on the EPG dataset

We developed a novel parallel algorithm of subsequence similarity search for Intel Xeon Phi KNL, called PHIBESTMATCH. Our algorithm is based on UCR-DTW [11], which is the fastest serial algorithm of subsequence similarity search due to it integrates cascade lower bounding and many other algorithmic speedup techniques. PHIBESTMATCH efficiently exploits vectorization capabilities of Phi KNL by means of the sophisticated data layout and computational scheme.

We performed experiments on synthetic and real-word datasets, which showed the following. PHIBESTMATCH being run on Intel Xeon Phi is two times faster than UCR-DTW being run on Intel Xeon. The proposed algorithm demonstrates closer to linear both speedup and parallel efficiency when the number of threads it runs on is up to the number of physical cores of Intel Xeon Phi.

In further research, we plan to move on our approach

in the following directions: advance the parallelization of the UCR-DTW algorithm for Intel Xeon Phi KNL, and extend our algorithm for the computer cluster system with nodes equipped with Intel Xeon Phi KNL.

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).

References

[1] Abdullaev, S.M., Zhelnin, A.A., Lenskaya, O.Y.: The structure of mesoscale convective systems in central Russia. Russian Meteorology and Hydrology. 37(1), pp. 12-20 (2012).

[2] Bacon, D.F., Graham, S.L., Sharp, O.J.: Compiler transformation for high-performance computing. ACM Computing Surveys. 26, pp. 345-420 (1994). doi: 10.1145/197405.197406

[3] Berndt, D.J., Clifford, J.: Using dynamic time warping to find patterns in time series. In: Fayyad, U.M., Uthurusamy, R. (eds.) KDD Workshop, pp. 359-370. AAAI Press (1994)

[4] Chrysos, G.: Intel Xeon Phi coprocessor (codename Knights Corner). In: 2012 IEEE Hot Chips 24th Symposium (HCS), pp. 1-31 (2012). doi: 10.1109/HOTCHIPS.2012.7476487

[5] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.J.: Querying and mining of time series data: experimental comparison of representations and distance measures. PVLDB 1(2), 1542-1552 (2008)

[6] Epishev, V., Isaev, A., Miniakhmetov, R. et al.: Physiological data mining system for elite sports. Bull. of South Ural State University. Series: Comput. Math. and Soft. Eng., 2(1):44-54, 2013. (in Russian) doi: 10.14529/cmse130105

[7] Keogh, E., Ratanamahatana, C.: Exact indexing of dynamic time warping. Knowl. Inf. Syst. 2005: vol. 7, no. 3, pp. 406-417.

[8] Kostenetskiy, P., Safonov, A.: SUSU supercomputer resources. In: L. Sokolinsky, I. Starodubov (eds.) PCT'2016, pp. 561-573. CEUR-WS, vol. 1576 (2016)

[9] Miniakhmetov, R., Movchan, A., Zymbler, M.: Accelerating time series subsequence matching on the Intel Xeon Phi many-core coprocessor. In: Biljanovic, P., Butkovic, Z. et al. (eds.) MIPRO 2015, pp. 1399-1404 (2015). doi: 10.1109/MIPRO.2015.7160493

[10] Pearson, K.: The problem of the random walk. Nature 72(1865), 342 (1905). doi: 10.1038/072342a0

[11] Rakthanmanon, T., Campana, B.J.L., Mueen, A., Batista, G.E.A.P.A., Westover, M.B., Zhu, Q., Zakaria, J., Keogh, E.J.: Searching and mining trillions of time series subsequences under dynamic time warping. In: Yang, Q., Agarwal, D., Pei, J. (eds.) KDD, pp. 262-270. ACM (2012). doi: 10.1145/2339530.2339576

[12] Ratanamahatana, C., Keogh, E.J.: Three myths about Dynamic Time Warping Data Mining. In: Kargupta, H., Srivastava, J., Kamath, C., Goodman, A. (eds.), SDM 2005. pp. 506-510. doi: 10.1137/1.9781611972757.50

[13] Sakurai, Y., Faloutsos, C., Yamamuro, M.: Stream monitoring under the time warping distance. In: Chirkova, R., Dogac, A., Tamer Ozsu, M., Sellis, T.K. (eds.) ICDE 2007, pp. 1046-1055. IEEE Computer Society (2007). doi: 10.1109/ICDE.2007.368963

[14] Sart, D., Mueen, A., Najjar, W.A., Keogh, E.J., Niennattrakul, V.: Accelerating dynamic time warping subsequence search with GPUs and FPGAs. In: Webb, G.I., Liu, B., Zhang, C., Gunopulos, D., Wu, X. (eds.) ICDM 2010, pp. 1001-1006. IEEE Computer Society (2010). doi: 10.1109/ICDM.2010.21

[15] Sodani, A.: Knights Landing (KNL): 2nd generation Intel Xeon Phi processor. In: 2015 IEEE Hot Chips 27th Symposium (HCS), pp. 1-24. IEEE Computer Society (2015)

[16] Sokolinskaya, I., Sokolinsky, L.B.: Scalability evaluation of NSLP algorithm for solving non-stationary linear programming problems on cluster computing systems. In: Voevodin, V., Sobolev, S. (eds.) RuSCDays 2017. CCIS, vol. 793, pp. 40-53. Springer, Heidelberg (2017). doi: 10.1007/978-3-319-71255-0_4

[17] Srikanthan, S., Kumar, A., and Gupta, R.: Implementing the dynamic time warping algorithm in multithreaded environments for real time and unsupervised pattern discovery. In: IEEE ICCCT, pp. 394-398. IEEE Computer Society (2011). doi: 10.1109/ICCCT.2011.6075111

[18] Takahashi, N., Yoshihisa, T., Sakurai, Y., Kanazawa, M.: A parallelized data stream processing system using Dynamic Time Warping distance. In: Barolli, L., Xhafa, F., Hsu, H.-H. (eds.) CISIS, pp. 1100-1105 (2009). doi: 10.1109/CISIS.2009.77

[19] Wang, Z., Huang, S., Wang, L., Li, H., Wang, Yu, Yang, H.: Accelerating subsequence similarity search based on dynamic time warping distance with FPGA. In: Hutchings, B.L., Betz, V. (eds.) ACM/SIGDA FPGA’13, pp. 53-62. ACM (2013). doi: 10.1145/2435264.2435277

[20] Zhang, Y., Adl, K., Glass, J.: Fast spoken query detection using lower-bound Dynamic Time Warping on Graphical Processing Units. In:

ICASSP, pp. 5173-5176. (2012). doi: 10.1109/ICASSP.2012.6289085

[21] Zymbler, M.: Best-Match time series subsequence search on the Intel Many Integrated Core architecture. In: Morzy, T. Valduriez, P., Bellatreche, L. (eds.) ADBIS 2015. LNCS, vol. 9282. pp. 275-286. Springer, Heidelberg (2015). doi: 10.1007/978-3-319-23135-8_19