Time Series Subsequence Similarity Search under Dynamic Time Warping Distance on the Intel Many-core Accelerators

Aleksandr Movchan, Mikhail Zymbler

South Ural State University, Chelyabinsk, Russia

Abstract. Subsequence similarity search is one of the most important problems of time series data mining. Nowadays there is empirical evidence that Dynamic Time Warping (DTW) is the best distance metric for many applications. However in spite of sophisticated software speedup techniques DTW still computationally expensive. There are studies devoted to acceleration of the DTW computation by means of parallel hardware (e.g. computer-cluster, multi-core, FPGA and GPU). In this paper we present an approach to acceleration of the subsequence similarity search based on DTW distance using the Intel Many Integrated Core architecture. The experimental evaluation on synthetic and real data sets confirms the efficiency of the approach.

1 Introduction

Subsequence similarity search is one of the most important problems of time series data mining and appears in a wide spectrum of subject domains, e.g. climate modeling [1], economic forecasting [5], medical monitoring [6], etc. The problem assumes that a query sequence and a longer time series are given, and the task is to find a subsequence in the longer time series, which best matches with the query sequence.

Currently there is empirical evidence that the Dynamic Time Warping (DTW) [2] is the most popular similarity measure in many applications [3]. DTW is computationally expensive and there are approaches to solve this problem, e.g. lower bounding [3], computation reusing [14], data indexing [11], early abandoning [12], etc. However, DTW still costs too much and there are studies to accelerate subsequence similarity search using parallel hardware, e.g. computer-cluster [16], multi-core [15], FPGA and GPU [14, 17, 18].

In this paper we present a parallel algorithm for subsequence similarity search based on DTW distance adapted for use on a central processor unit (CPU) accompanied with the Intel Xeon Phi many-core coprocessor [4]. The remainder of the paper is organized as follows. Section 2 contains formal definition of the problem, briefly describes Intel Xeon Phi architecture and programming model and discusses related work. The proposed algorithm is presented in the section 3. The results of experimental evaluation of the algorithm are described in section 4. Section 5 contains summarizing comments and directions for future research.

2 Formal Definitions and Related Work

2.1 Formal Definitions

A *time series* T is an ordered sequence $t_1, t_2, \dots, t_N$ of real data points, measured chronologically, where N is a length of the sequence.

Dynamic Time Warping (DTW) is a similarity measure between two time series X and Y, where $X = x_1, x_2, ..., x_N$ and $Y = y_1, y_2, ..., y_N$, is defined as follows.

$$DTW(X,Y) = d(N,N), \text{ where}$$

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

$$d(0,0) = 0; d(i,0) = d(0,j) = \infty;\ i = j = 1, 2, \dots, N.$$

A *subsequence* T_{im} of time series T is its continuous subset starting from i-th position and consisting of m data points, i.e. $T_{im} = t_i, t_{i+1}, \dots, t_{i+m-1}$, where $1 \leq i \leq N$ and $i + m \leq N$.

A *query* Q is a certain subsequence to be found in T. Let n is a length of the query, $n \ll N$.

Subsequence similarity search problem aims to finding a subsequence, which is the most similar to the query with respect to a given similarity measure. Let D is a similarity measure, then the resulting subsequence is $\underset{1 \leq i \leq N-n}{\operatorname{argmin}} D(T_{in}, Q)$. We will use DTW as a similarity measure.

2.2 The Intel Xeon Phi Architecture and Programming Model

The Intel Xeon Phi coprocessor is an x86 many-core coprocessor of 61 cores, connected by a high-performance on-die bidirectional interconnect where each core supports 4× hyperthreading and contains 512-bit wide vector processor unit (VPU). Each core has two levels of cache memory: a 32 Kb L1 data cache, a 32 Kb L1 instruction cache, and a core-private 512 Kb unified L2 cache. The Intel Xeon Phi coprocessor is to be connected to a host computer via a PCI Express system interface. PCI Express is used for data transfer between CPU and the coprocessor.

Being based on Intel x86 architecture, the Intel Xeon Phi coprocessor supports the same programming tools and models as a regular Intel Xeon processor.

The Intel Xeon Phi coprocessor supports three programming modes: native, offload and symmetric. In native mode the application runs independently, on the coprocessor only. In offload mode the application is running on the host and offloads computationally intensive part of work to the coprocessor. The symmetric mode allows the coprocessor to communicate with other devices by means of Message Passing Interface (MPI).

2.3 Related Work

Currently DTW is considered as best similarity measure for many applications [3], despite the fact that it is very time-consuming [8, 16]. Research devoted to acceleration of DTW computation includes the following.

The SPRING algorithm [13] uses computation-reuse technique. However, this technique squeezes the algorithm's applications because data-reuse supposes non-normalized sequence. In [11] indexing technique to speed up the search was used, which need to specify the query length in advance. Authors of [9] suggested multiple indices for various length queries. Lower bounding [7] allows one to discard unpromising subsequences using the lower bound of DTW distance estimated in a cheap way. The UCR-DTW algorithm [12] integrates all the possible existing speedup techniques and most likely it is the fastest of the existing subsequence matching algorithms.

All the aforementioned algorithms aim to decrease the number of calls of DTW subroutine, not accelerating DTW itself. However, because of its complexity, DTW still takes a large part of the total application runtime [18]. There are approaches exploiting the allocation of DTW computation of different subsequences into different processing elements. In [15] subsequences starting from different positions of the time series are sent to different Intel Xeon processors, and each processor computes DTW. In [16] different queries are distributed onto different cores, and each subsequence is sent to different cores to be compared with different queries. GPU implementation [18] parallelize the generation of the warping matrix but still process the path search serially. GPU implementation proposed in [14] utilizes the same ideas as in [15]. FPGA implementation described in [14] focuses on the naive subsequence similarity search, and do not exploit any pre-processing techniques. It is generated by a C-to-VHDL tool and should be recompiled if length of query is changed. This algorithm supports 8-bit data precision and can not supports queries longer than 2^{10}, so it can not be applied in big-scale tasks. To address these problems in [17] a stream oriented framework was proposed. It implements coarse-grained parallelism by reusing data of different DTW computations and uses a two-phase precision reduction technique to guarantee accuracy while reducing resource cost.

In this work we present a parallel algorithm of the time series subsequence similarity search under DTW on the Intel Xeon Phi many-core coprocessor where the UCR-DTW serial algorithm is used as a basis.

3 Acceleration by the Intel Xeon Phi Coprocessor

3.1 Serial Algorithm

The UCR-DTW serial algorithm [12] is depicted in the Fig. 1. It uses a cascade of three lower bounding of DTW distance, namely LB_{Kim} [10, 12], LB_{Keogh} [8] and $LB_{KeoghEC}$ [12]. If the lower bound has exceeded some threshold, the DTW distance also exceeds the same threshold, so the subsequence can be pruned off. Here the `bsf` (best-so-far) variable stores the distance to the most similar subsequence.

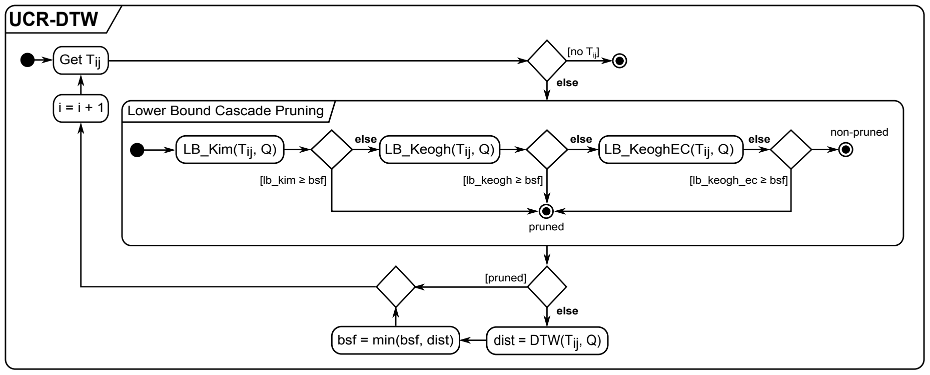


Fig. 1: Serial algorithm

3.2 Parallel Algorithm

Fig. 2 depicts a parallel version of the UCR-DTW algorithm. Parallelization of the original algorithm was performed through the OpenMP technology.

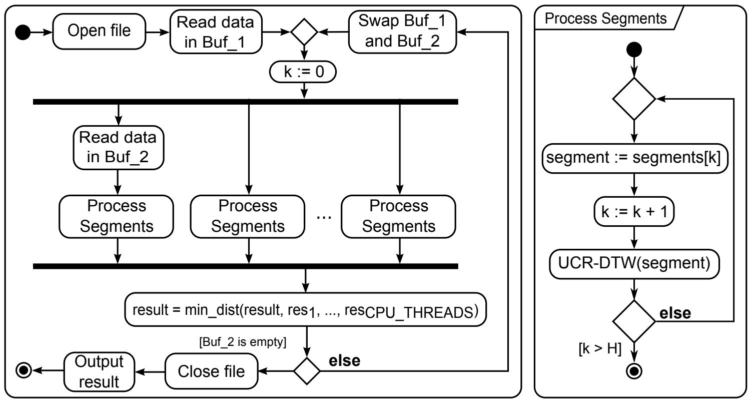


Fig. 2: Parallel algorithm for CPU

The source time series T is partitioned into H equal-length segments. Let P denotes the number of OpenMP-threads, S denotes a maximum length of segment, then H is defined as

$$H = \lceil \frac{N}{P \cdot S} \rceil \cdot P$$

A k-th segment, $0 \leq k \leq H-1$, is defined as a subsequence T_{sl}, where

$$s = \begin{cases} 1 & , k = 0 \\ k \cdot \lfloor \frac{N}{H} \rfloor - n + 2 & , else \end{cases}$$

$$l = \begin{cases} \lfloor \frac{N}{H} \rfloor & , k = 0 \\ \lfloor \frac{N}{H} \rfloor + n - 1 + (N \ mod \ H) & , k = H - 1 \\ \lfloor \frac{N}{H} \rfloor + n - 1 & , else \end{cases}$$

It means that the head part of every segment except first overlaps with the tail part of previous segment in $n-1$ data points, where n is length of the query. This prevents from losing of possible resulting subsequences, which start at tail part of previous segment.

The number of segments H is divisible by the number of threads P for better load balancing.

The algorithm is based on dynamic distribution of segments across threads. We use `k` variable, which is shared among all threads and identifies first unprocessed segment. The `k` variable initialized by 0 and while there are unprocessed segments (i.e. $k \le H$), a thread gets k-th segment, increments `k` by 1 and processes the segment by means of `UCR-DTW` subroutine, which implements an original serial algorithm. To provide correct processing of shared data we use critical section to prevent multiple threads from accessing the critical section's code at the same time, i.e. only one active thread can get k-th segment and update the `k` variable.

We reject static distribution of segments across threads (where each thread is assigned by its own segments before calculations) due to the following reason. Static distribution could result in worse load balancing because of unpredictable amount of pruned and early abandoned subsequences for each thread. So, overhead costs to provide the critical section in case of dynamic distribution is a lesser evil than highly probable load imbalance in case of static distribution.

In contrast with the serial version the `bsf` variable is shared among the threads. This allows each thread to prune off unpromising subsequence using lower bounding.

This algorithm is ready-to-use on the Intel Xeon Phi coprocessor in native mode. However, experiments have shown (Fig. 5) that the algorithm is slower than on CPU. This implementation does not provide sufficient floating point operations per byte of data to be effectively processed on the coprocessor. To overcome this we combined CPU and coprocessor to process time series as described in the next section.

3.3 Combining CPU and the Intel Xeon Phi

The parallel algorithm for CPU and the Intel Xeon Phi is depicted in Fig. 3.

The idea of the algorithm is that the coprocessor should be exploited only for DTW computations whereas CPU performs lower bounding, prepares subsequences for the coprocessor and computes DTW in case if it really does not have another job. CPU supports a queue of candidate subsequences and the coprocessor computes DTW for each candidate. Queue stores a tuple (i, A) corresponding a candidate subsequence T_{in}, where A is an n-element array containing LB_{Keogh} lower bounds for each position of the subsequence which is used for early abandoning of DTW [12].

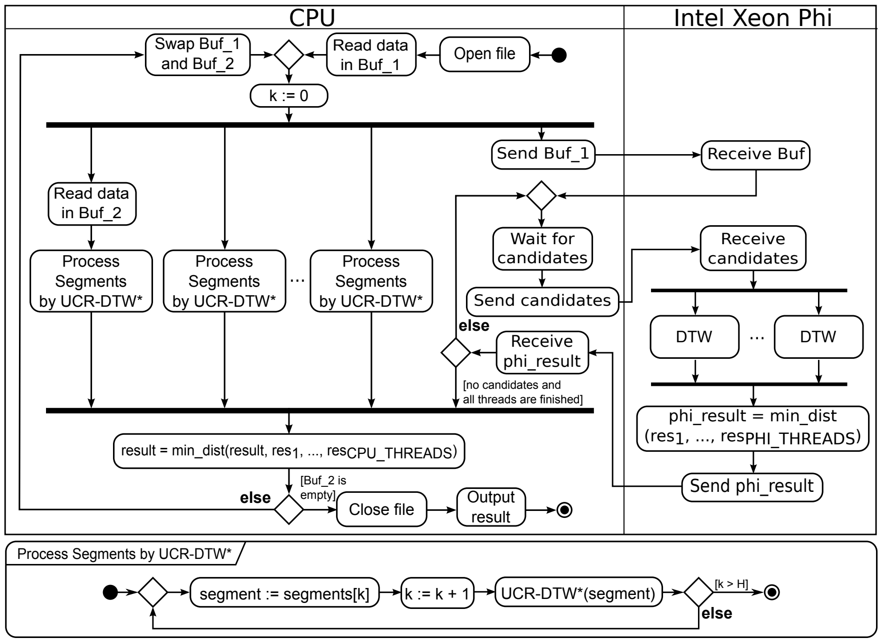


Fig. 3: Parallel algorithm for CPU and the Intel Xeon Phi

To reduce the amount of data transferred to the coprocessor, CPU offloads current buffer of the time series once whereas queue is offloaded each time it is full. The number of elements in the queue is calculated as $C \cdot h \cdot W$, where C is a number of cores of the coprocessor, h is a hyperthreading factor of the coprocessor and W is a number of candidates to be processed by a coprocessor's thread.

The algorithm could be described in the following way. One of the CPU threads is declared as a master and the rest as workers. At start master sends a buffer with the current buffer with the time series to the coprocessor. If queue is full then master offloads it to the coprocessor to perform DTW computation for the corresponding subsequences by the coprocessor's threads.

Worker's activity is similar to activity of threads in parallel algorithm for CPU only. Each worker processes segments by `UCR-DTW*` (see Fig. 4) subroutine. The `UCR-DTW*` subroutine calculates cascade of lower bounds for the subsequence. If it is dissimilar to the query then the worker prunes it off otherwise worker pushes this subsequence to the queue. If the queue is full (and data previously transferred to the coprocessor have not been processed yet), the worker calculates DTW by itself.

At the end of offload section the information about most similar subsequence found on the coprocessor is transferred to the CPU. The final result is calculated among the most similar subsequence found on the CPU and same that found on the coprocessor.

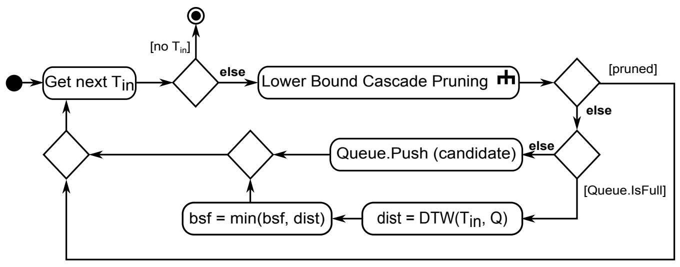


Fig. 4: UCR-DTW* subroutine

4 Experiments

Hardware. To evaluate the developed algorithm we performed experiments on the Tornado SUSU[1] supercomputer's node (see Tab. 1 for its specifications).

Table 1: Specifications of the Tornado SUSU supercomputer's node

Specifications	Processor	Coprocessor
Model	Intel Xeon X5680	Intel Xeon Phi SE10X
Cores	6	61
Frequency, GHz	3.33	1.1
Threads per core	2	4
Peak performance, TFLOPS	0.371	1.076
Memory, Gb	24	8
Cache, Mb	12	30.5

Data sets. Experiments have been performed on three time series, which are summarized in Tab. 2. The PURE RANDOM data set was generated by a random function. The RANDOM WALK data set is one-dimensional random walk time series. The ECG (electrocardiographic) data set represents approximately 22 hours of one ECG channel sampled at 250 Hz.

Table 2: Data sets used in experiments

Time series	Category	Length
PURE RANDOM	synthetic	10^6
RANDOM WALK	synthetic	10^8
ECG [12]	real	$2 \cdot 10^7$

[1] `supercomputer.susu.ru/en/computers/tornado/`

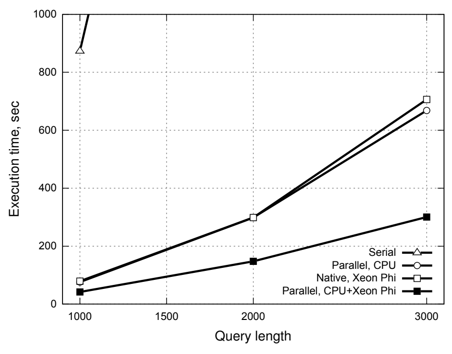
Goals. In the experiments we investigated a) performance of our algorithm, b) impact of the queue size on the speedup and c) runtime of our algorithm in comparison with analogues for GPU and FPGA.

4.1 Performance

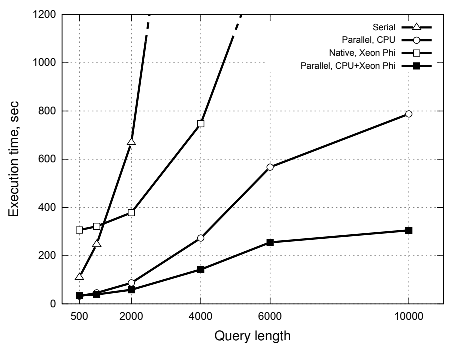
On the PURE RANDOM data set our algorithm shows (Fig. 5a) a two times higher performance than the parallel algorithm for CPU only.

Experimental results on RANDOM WALK data set (Fig. 5b) show that our algorithm is more effective for longer queries. In case of shorter queries the algorithm has the same performance as parallel algorithm for CPU only.

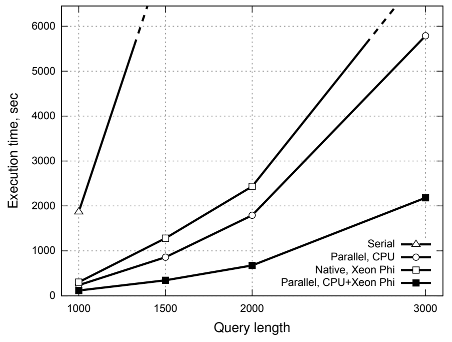
For the experiments on ECG data set we used a subsequence T_{N-nn} of the whole time series T as a query to prevent from finding the most similar subsequence at the early stage of computations and, in turn, to provide sufficient amount of work on DTW computation. Our algorithm shows (Fig. 5c) almost three times higher performance than the parallel algorithm for CPU only.



(a) PURE RANDOM data set



(b) RANDOM WALK data set

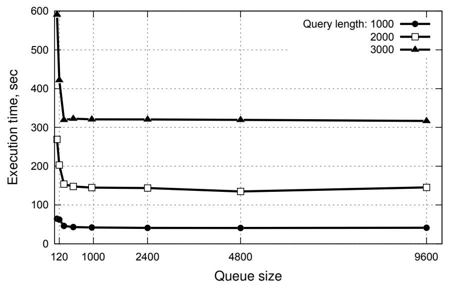


(c) ECG data set

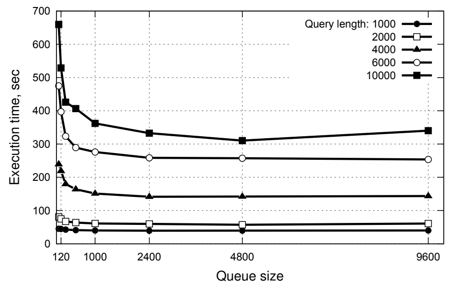
Fig. 5: Performance of the algorithm

4.2 Impact of Queue Size

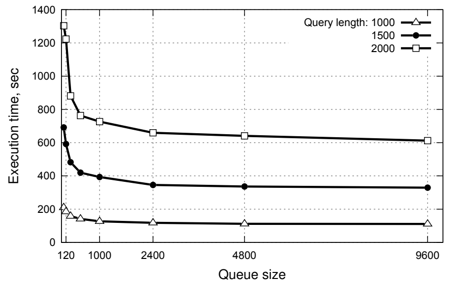
Results of the experiments are depicted in Fig. 6. In the current experimental environment, i.e. number of cores of the coprocessor C is 60^2, hyperthreading factor of the coprocessor h is 4, optimal number of candidates to be processed by a coprocessor's thread W is 10, so optimal number of the elements in the queue is 2400. Experimental results described above have been achieved with this queue size.



(a) PURE RANDOM data set



(b) RANDOM WALK data set



(c) ECG data set

Fig. 6: Impact of queue size on the speedup

4.3 Comparison with Algorithms for GPU and FPGA

We compared the performance of our algorithm with analogues for GPU and FPGA developed in [14] (there is no comparison with results in [17] because that research was devoted to a little bit different problem of search a set of local-best-match subsequences). We repeated the experiments presented in that paper using the same data set and query length.

[2] One core is not involved in computations as it is recommended by the Intel Xeon Phi programmer's manual.

The results of the experiments are depicted in Fig. 7, here percentage on the top of the bar indicates a part of subsequences that have not been pruned and subjected to the DTW computation in our experiments. We also add to the chart results of experiments on random walk and ECG data sets.

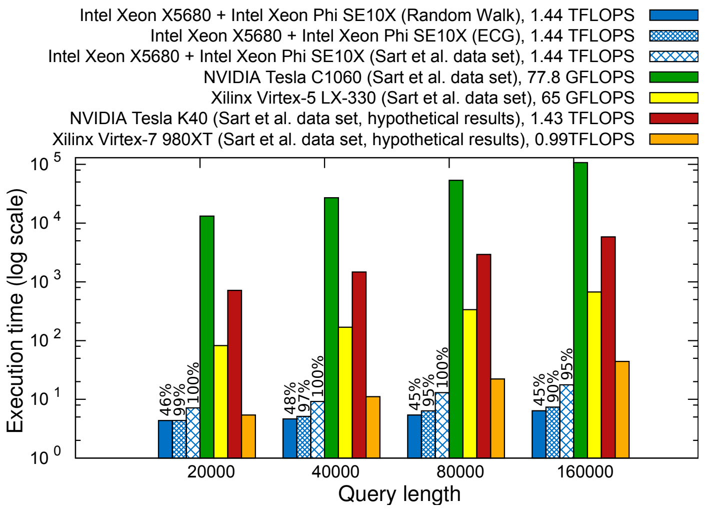


Fig. 7: Comparison of performance

We took into account that the peak performance of the hardware we used is significantly greater than its counterparts of that paper, i.e. overall peak performance of our hardware was 1.44 TFLOPS whereas GPU as NVIDIA Tesla C1060 had 77.8 GFLOPS and FPGA as Xilinx Virtex-5 LX-330 had 65 GFLOPS. To provide more "fair" comparison we added to the chart *hypothetical* results for modern NVIDIA Tesla K40 (1.43 TFLOPS)[3] and Xilinx Virtex-7 980XT (0.99 TFLOPS)[4] multiplying real results of NVIDIA Tesla C1060 and Xilinx Virtex-5 LX-330 by a respective scaling factor. As we can see our algorithm does not concede to analogous on performance.

5 Conclusion

In this paper we have presented an approach to time series subsequence similarity search under DTW distance on the Intel Many Integrated Core architecture. The parallel algorithm combines capabilities of CPU and the Intel Xeon Phi coprocessor. The coprocessor is exploited only for DTW computations whereas CPU performs lower bounding, prepares subsequences for the coprocessor and

[3] `www.nvidia.com/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf`
[4] `www.xilinx.com/publications/prod_mktg/Virtex7-Product-Brief.pdf`

computes DTW as a last resort. CPU supports a queue of candidate subsequences and the coprocessor computes DTW for every candidate. Experiments on synthetic and real data sets have shown that our algorithm does not concede to analogous algorithms for GPU and FPGA on performance.

As future work we plan to extend our research for the cases of several coprocessors and cluster system based on nodes equipped with the Intel Xeon Phi coprocessor(s).

Acknowledgment

This work was financially supported by the Ministry of education and science of the Russian Federation ("Research and development on priority directions of scientific-technological complex of Russia for 2014–2020" Federal Program, contract No. 14.574.21.0035).

References

1. Sanjar Abdullaev, Olga Lenskaya, Anna Gayazova, Dmitry Sobolev, Artem Noskov, Olga Ivanova, and Gleb Radchenko. Short-range forecasting algorithms using radar data: Translation estimate and life-cycle composite display. *Bull. of South Ural State University. Series: Comput. Math. and Soft. Eng.*, 3(1):17–32, 2014.
2. Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *KDD Workshop*, pages 359–370. AAAI Press, 1994.
3. Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn J. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *PVLDB*, 1(2):1542–1552, 2008.
4. Alejandro Duran and Michael Klemm. The Intel® Many Integrated Core Architecture. In Waleed W. Smari and Vesna Zeljkovic, editors, *HPCS*, pages 365–366. IEEE, 2012.
5. Mikhail Dyshaev and Irina Sokolinskaya. Representation of trading signals based on kaufman adaptive moving average as a system of linear inequalities. *Bull. of South Ural State University. Series: Comput. Math. and Soft. Eng.*, 2(4):103–108, 2013.
6. Vitaly Epishev, Alexander Isaev, Ruslan Miniakhmetov, Aleksander Movchan, Alexey Smirnov, Leonid Sokolinsky, Mikhail Zymbler, and Vadim Ehrlich. Physiological data mining system for elite sports. *Bull. of South Ural State University. Series: Comput. Math. and Soft. Eng.*, 2(1):44–54, 2013.
7. Ada Wai-Chee Fu, Eamonn J. Keogh, Leo Yung Hang Lau, and Chotirat (Ann) Ratanamahatana. Scaling and time warping in time series querying. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 649–660. ACM, 2005.
8. Ada Wai-Chee Fu, Eamonn J. Keogh, Leo Yung Hang Lau, Chotirat (Ann) Ratanamahatana, and Raymond Chi-Wing Wong. Scaling and time warping in time series querying. *VLDB J.*, 17(4):899–921, 2008.

9. Eamonn J. Keogh, Li Wei, Xiaopeng Xi, Michail Vlachos, Sang-Hee Lee, and Pavlos Protopapas. Supporting exact indexing of arbitrarily rotated shapes and periodic time series under euclidean and warping distance measures. *VLDB J.*, 18(3):611–630, 2009.
10. Sang-Wook Kim, Sanghyun Park, and Wesley W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In Dimitrios Georgakopoulos and Alexander Buchmann, editors, *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 607–614. IEEE Computer Society, 2001.
11. Seung-Hwan Lim, Heejin Park, and Sang-Wook Kim. Using multiple indexes for efficient subsequence matching in time-series databases. In Mong-Li Lee, Kian-Lee Tan, and Vilas Wuwongse, editors, *Database Systems for Advanced Applications, 11th International Conference, DASFAA 2006, Singapore, April 12-15, 2006, Proceedings*, volume 3882 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 2006.
12. Thanawin Rakthanmanon, Bilson J. L. Campana, Abdullah Mueen, Gustavo E. A. P. A. Batista, M. Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In Qiang Yang, Deepak Agarwal, and Jian Pei, editors, *KDD*, pages 262–270. ACM, 2012.
13. Yasushi Sakurai, Christos Faloutsos, and Masashi Yamamuro. Stream monitoring under the time warping distance. In Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis, editors, *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 1046–1055. IEEE, 2007.
14. Doruk Sart, Abdullah Mueen, Walid A. Najjar, Eamonn J. Keogh, and Vit Niennattrakul. Accelerating dynamic time warping subsequence search with gpus and fpgas. In Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos, and Xindong Wu, editors, *ICDM*, pages 1001–1006. IEEE Computer Society, 2010.
15. Srikanthan Sharanyan, Kumar Arvind, and Gupta Rajeev. Implementing the dynamic time warping algorithm in multithreaded environments for real time and unsupervised pattern discovery. In Department of Computer Science and Motial Nehru National Institute of Technology Engineering, editors, *ICCCT*, pages 394–398. IEEE Computer Society, 2011.
16. Norihiro Takahashi, Tomoki Yoshihisa, Yasushi Sakurai, and Masanori Kanazawa. A parallelized data stream processing system using dynamic time warping distance. In Leonard Barolli, Fatos Xhafa, and Hui-Huang Hsu, editors, *2009 International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2009, Fukuoka, Japan, March 16-19, 2009*, pages 1100–1105. IEEE Computer Society, 2009.
17. Zilong Wang, Sitao Huang, Lanjun Wang, Hao Li, Yu Wang, and Huazhong Yang. Accelerating subsequence similarity search based on dynamic time warping distance with FPGA. In Brad L. Hutchings and Vaughn Betz, editors, *The 2013 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '13, Monterey, CA, USA, February 11-13, 2013*, pages 53–62. ACM, 2013.
18. Yaodong Zhang, Kiarash Adl, and James R. Glass. 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, ICASSP 2012, Kyoto, Japan, March 25-30, 2012*, pages 5173–5176. IEEE, 2012.