Accelerating Time Series Subsequence Matching on the Intel Xeon Phi Many-core Coprocessor

Ruslan Miniakhmetov, Aleksander Movchan and Mikhail Zymbler
South Ural State University, Faculty of Computational Mathematics and Informatics
Chelyabinsk, Russia
miniakhmetovrm@susu.ac.ru, movchan174@gmail.com, mzym@susu.ru

***Abstract*—The problem of time series subsequence matching occurs in a wide spectrum of subject areas. Currently Dynamic Time Warping (DTW) is the best similarity measure but despite various existing speedup techniques it is still computationally expensive. Due to this reason science community is trying to accelerate DTW calculation by means of parallel hardware. There are implementations of DTW-based subsequence matching on GPU and FPGA but there none for accelerators based on the Intel Many Integrated Core architecture. This paper presents a parallel algorithm for time series subsequence matching based on DTW distance adapted to the Intel Xeon Phi coprocessor. The experimental results on synthetic and real data sets are presented and confirm the efficiency of the algorithm.**

***Index Terms*—data mining, time series, subsequence matching, dynamic time warping, parallel computing, OpenMP, Intel Many Integrated Core architecture, Intel Xeon Phi coprocessor**

I. Introduction

A time series represents a collection of data points, measured chronologically. Currently, time series data mining touches upon a wide spectrum of scientific and applied tasks in various subject areas, e.g. medical monitoring [1], economic forecasting [2], climate modeling [3], etc. Being one of the most important problems of time series data mining, subsequence matching assumes that a query sequence and a longer time series are given, and the task is to find subsequences in the longer time series, which match the query sequence.

To measure the similarity between two time series, a number of Euclidean-based distance measures have been suggested but currently the Dynamic Time Warping (DTW) [4] is the most popular similarity measure in many applications [5]. DTW is computationally expensive and there are many approaches that have been proposed to solve this problem, e.g. pruning the subsequences using lower bound of distance [5], computation reusing [6], data indexing [7], early abandoning [8], etc. However, DTW still takes a large part of the total application runtime. That is why there are researches devoted to accelerating subsequence matching by means of parallel hardware, e.g. computer-cluster [9], multicore [10], FPGA and GPU [6].

In this regard, in our opinion not enough attention is paid to the Intel Xeon Phi coprocessor based on Intel Many Integrated Core (MIC) [11] architecture. This paper presents a parallel algorithm for time series subsequence matching based on DTW distance adapted for use on a central processor unit (CPU) accompanied with the Intel Xeon Phi coprocessor.

The rest of the paper is organized as follows. Section II gives the formal definition of the problem and briefly considers the Intel Xeon Phi architecture and programming model. In section III a description of the proposed algorithm is given. The results of the experiments evaluating the algorithm are presented in section IV. Section V discusses related work. Section VI contains concluding remarks and directions for future research.

II. Research Background

A. Problem Definition

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

A *subsequence* $T_{i,k}$ of time series T is its continuous subset of length k starting at i position.

A *query* Q is a certain subsequence to be found in T.

Subsequence matching problem aims to finding all the subsequences $T_{i,j}$ such that distance $D(T_{i,j}, Q)$ is minimal.

Dynamic Time Warping (DTW) is a similarity measure between two time series which may vary in time or speed. DTW distance between two time series X and Y, where $X = x_1, x_2, ..., x_N$ and $Y = y_1, y_2, ..., y_N$, is denoted as $D(X, Y)$ and defined as follows.

$$D(X,Y) = d(N,N),$$

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

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

B. 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×

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

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

There are three programming modes to deal with the Intel Xeon Phi coprocessor: 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).

III. Subsequence Matching on the Intel Xeon Phi Coprocessor

A. Research Roadmap

Development of the parallel algorithm of time series subsequence matching adapted for the Intel Xeon Phi coprocessor consists of the following steps, which will be discussed in detail further.

Firstly, we had developed a parallel version of the UCR-DTW serial algorithm [8]. Using OpenMP technology, we have obtained a parallel algorithm for CPU, keeping in mind a possibility of running this algorithm on the Intel Xeon Phi coprocessor in *native* mode. However, experiments have shown that, despite the one-order speedup of parallel algorithm, it works *slower* on the Intel Xeon Phi coprocessor in native mode than on CPU. This results from low operational intensity of our algorithm, i.e. insufficient FLOPs (floating point operations) per byte of data to be effectively processed on the Intel Xeon Phi coprocessor.

Next, we modified our algorithm combining CPU and coprocessor to process time series. In this version of the algorithm both CPU and the Intel Xeon Phi run the parallel version of the UCR-DTW algorithm developed at the first step. Here we used *offload* mode to transmit code and data to the Intel Xeon Phi. Experiments, where we varied the portion size of data to be transmitted to the coprocessor, show results similar to those obtained at the first step due to the same reason.

Finally, we developed an advanced version of the algorithm. The key point of this version is to support a queue of subsequences on CPU that are offloaded to the coprocessor to compute DTW. This significantly increased the operational intensity of the calculations on the coprocessor and experiments show acceptable performance of the algorithm.

B. Parallel Algorithm for CPU

The UCR-DTW algorithm proposed in [8] is one of the fastest existing subsequence matching algorithms. This algorithm (see Fig. 1; hereinafter, we use UML activity diagrams) uses a cascade estimation of the lower bound of DTW distance. If the lower bound has exceeded some threshold, the DTW distance also exceeds the threshold, so the subsequence can be pruned off.

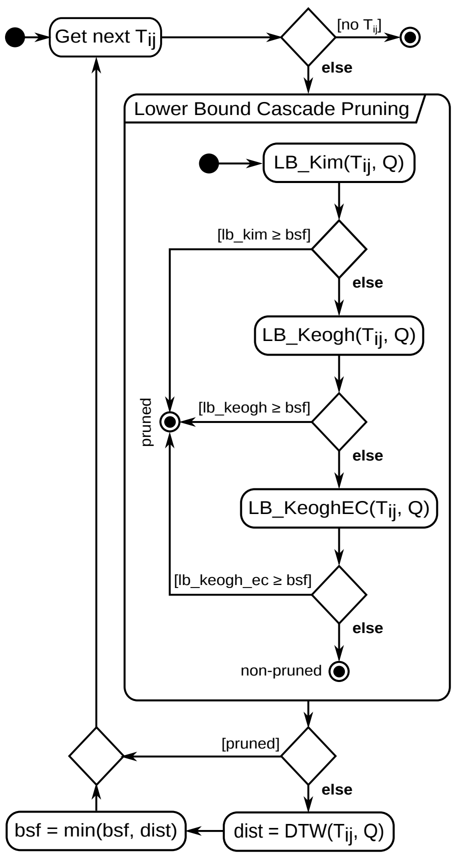

Fig. 1. Serial algorithm

A parallel version of the UCR-DTW algorithm is depicted in Fig. 2. We parallelize the original algorithm using the OpenMP technology. The time series is splitted into equal portions and each portion is processed by a separate OpenMP-thread. Here `UCR-DTW` is a subroutine that implements the original serial algorithm. It uses the `bsf` (best-so-far) shared variable that stores the distance to the nearest subsequence. This allows each thread to prune off unpromising subsequence using lower bounding. Master thread reads new data from a file simultaneously with processing of data that have been read.

We evaluated the obtained algorithm on synthetic time series both on CPU and on the Intel Xeon Phi coprocessor in native mode (see Fig. 6). Although parallel algorithm expectedly surpasses the original algorithm it works slower on the coprocessor in native mode.

This was a result of low operational intensity of our algorithm, i.e. insufficient FLOPs per byte of data to be effectively processed on the Intel Xeon Phi coprocessor.

C. Naïve Parallel Algorithm for CPU and the Intel Xeon Phi

Fig. 3 depicts the modified version of the algorithm. This version is called “naïve”, because in comparison with the previous version it only distributes work among CPU and

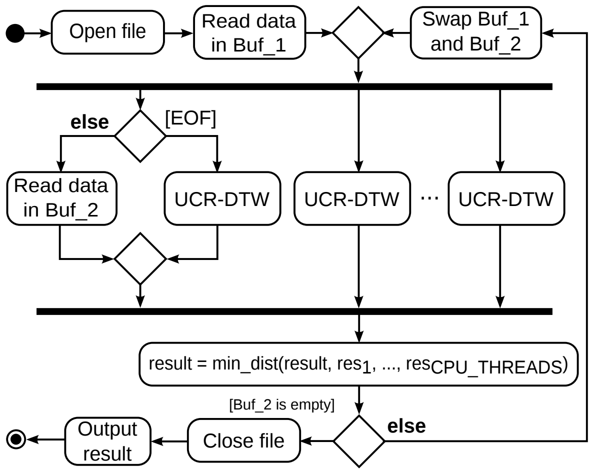

Fig. 2. Parallel algorithm for CPU

coprocessor. Here ALPHA is a parameter that determines a portion of data to be transmitted to the coprocessor. We use offload mode to organize data exchange between CPU and the coprocessor. The min_dist subroutine chooses subsequences with minimal value of DTW.

As well as in the previous step we evaluated the obtained algorithm on synthetic time series (see Fig. 6). Regardless of the ALPHA value this algorithm has worse performance in comparison with the parallel algorithm for CPU.

This is because we still have not increased operational intensity of calculations on the coprocessor. Additionally, bsf shared variable can not be synchronized between the CPU and the coprocessor while offloading is performed (the synchronization is possible at the beginning and at the end of offload section). That is why we have more non-pruned subsequences to calculate DTW.

D. *Advanced Parallel Algorithm for CPU and the Intel Xeon Phi*

The advanced version of the algorithm obtained at the previous step is depicted in Fig. 4.

The idea of the advanced algorithm is to support a queue of subsequences on CPU. Subsequences of the queue are candidates to be offloaded to the coprocessor to calculate DTW. One of the CPU threads is declared as a master and the rest as workers. When the queue is full master offloads it to the coprocessor.

Worker's activity looks like the following. A worker calculates cascade estimates 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 transmitted to the coprocessor have not been processed yet), the worker calculates DTW by itself.

The UCR-DTW* (see Fig. 5) subroutine implements worker's behavior as described above.

At the end of offload section the information about most similar subsequences found on the coprocessor is transmitted

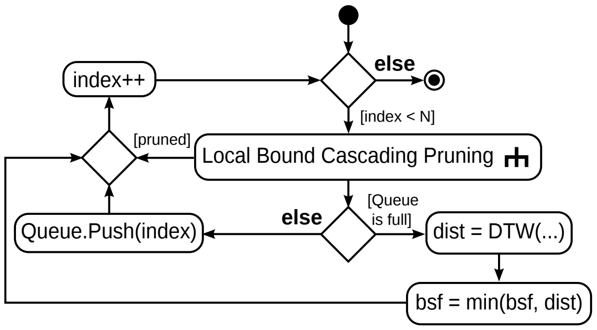

Fig. 5. UCR-DTW* subroutine

to the CPU. The final result is calculated among the most similar subsequences found on the CPU and on the coprocessor.

Additionally, we slightly modified the original source code of DTW calculation to provide vectorization of operations inside the for loop.

IV. Experimental Evaluation

To evaluate the algorithm developed we performed experiments on a Tornado SUSU [12] supercomputer's node (see Tab. I for its specifications). In these experiments subsequence matching runtime was measured while varying query length. Experiments have been performed on synthetic and real time series. We also investigated usage time of the coprocessor, impact of queue size on the speedup, and compared our algorithm with analogues for GPU and FPGA.

TABLE I
Specifications of Tornado SUSU supercomputer node

Specifications	Processor	Coprocessor
Model	Xeon X5680	Xeon Phi SE10X
Cores	6	61
Frequency, GHz	3.33	1.1
Threads per core	2	4
Peak performance, TFLOPS	0.371	1.076

A. *Performance on Synthetic and Real Data Sets*

In the first set of experiments we used synthetic time series generated by one-dimensional random walk comprising of 100 million data points.

Fig. 6 depicts the experimental results on synthetic data sets. The results 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.

The second set of experiments investigates the algorithm's performance on real electrocardiographic (ECG) data with about 20 million data points (approximately 22 hours of ECG sampled at 250 Hz).

Results of experiments on real data are shown in Fig. 7. Our algorithm shows a three times higher performance than the parallel algorithm for CPU only.

We evaluate what was the contribution of the coprocessor in computations as well. Fig. 8 depicts that increasing the query

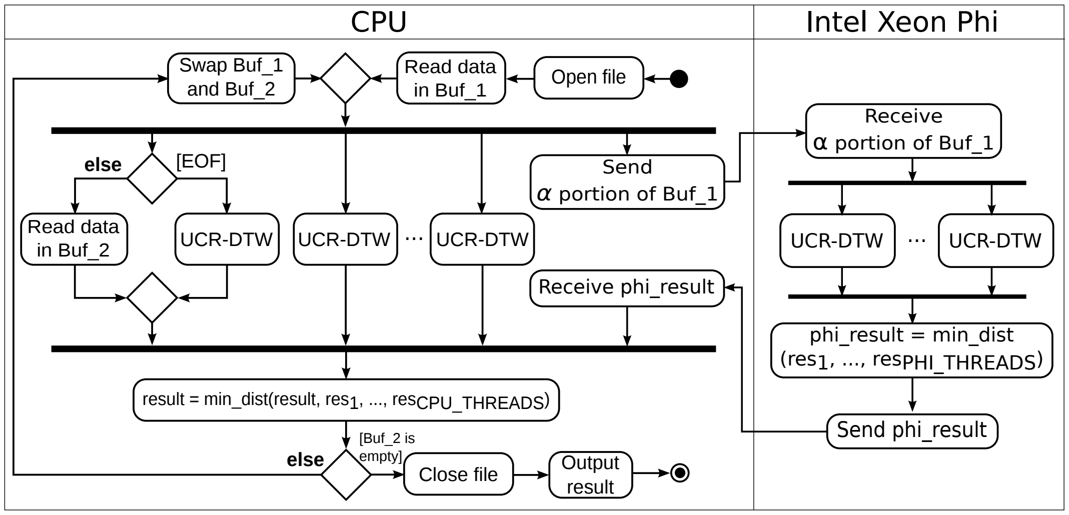

Fig. 3. Naïve parallel algorithm for CPU and the Intel Xeon Phi

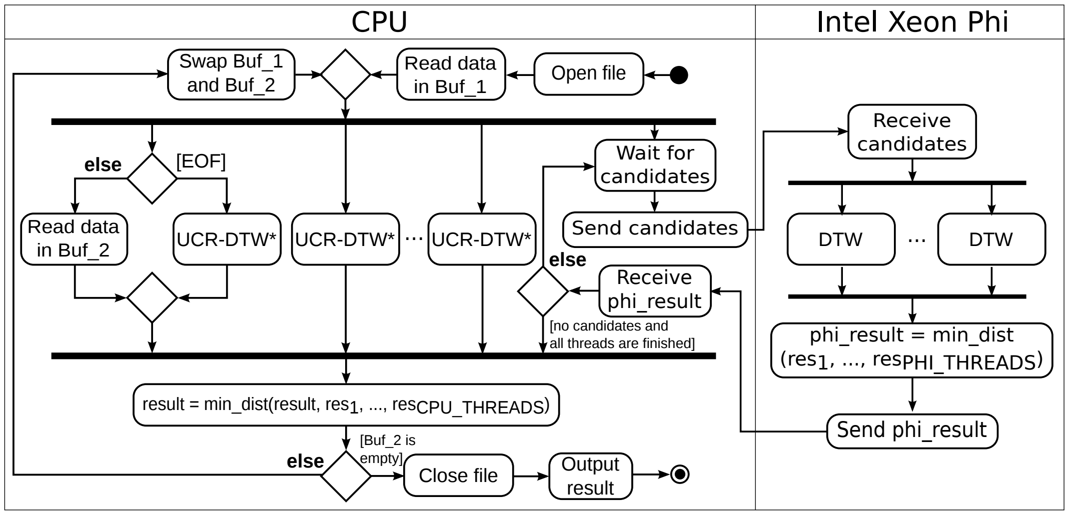

Fig. 4. Advanced parallel algorithm for CPU and the Intel Xeon Phi

length greater than 4000 leads to almost 100% usage of the Intel Xeon Phi coprocessor for both synthetic and real data sets.

B. Impact of Queue Size

The results described in subsection above have been achieved when queue size was 2400. We empirically select this value having performed some experiments beforehand (see Fig. 9).

C. Comparison with Algorithms for GPU and FPGA

We compared the performance of our algorithm with analogues for GPU and FPGA developed in [6]. We repeated the experiments presented in that paper keeping the same length of the time series and query length. The results of the experiments are depicted in Fig. 10.

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 a "fair" experiment we added to the chart *hypothetical* results for NVIDIA Tesla K40 (1.43 TFLOPS) [13] and Xilinx Virtex-7 980XT (0.99 TFLOPS) [14] multiplying real results of NVIDIA Tesla C1060 and Xilinx Virtex-5 LX-330 by a respective scaling factor. As we can see our algorithm provides significantly higher performance.

Finally, we compared relative costs of performance (see Fig. 11). The charts show that Intel-based solution is a bit more expensive than the latest one from NVIDIA but at the same time provides significantly higher performance.

V. Related Work

A substantial amount of work has been done in the area of time series data mining, including subsequence matching

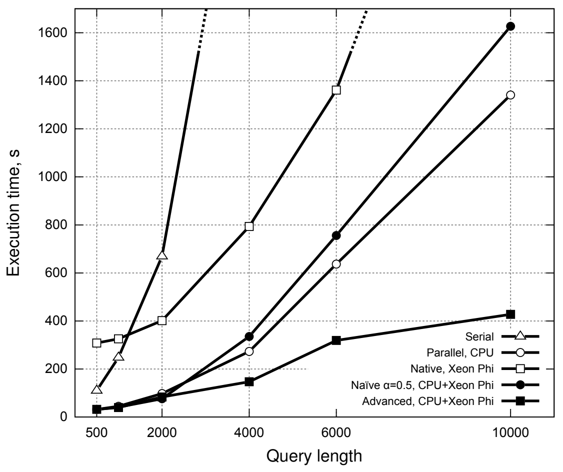

Fig. 6. Algorithm's performance on synthetic data set

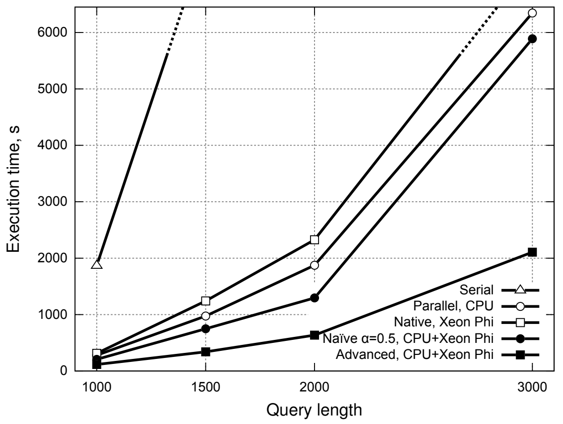

Fig. 7. Algorithm's performance on real data set

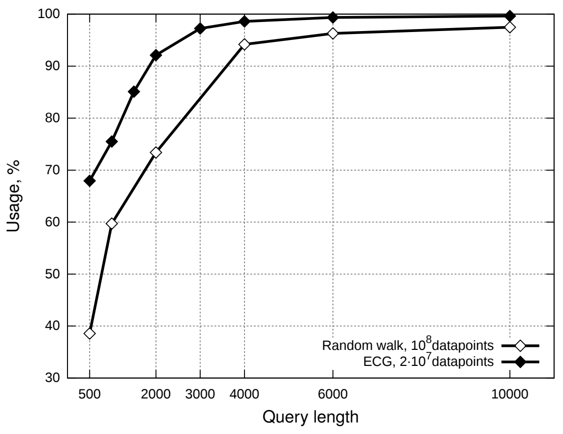

Fig. 8. Usage of coprocessor in experiments

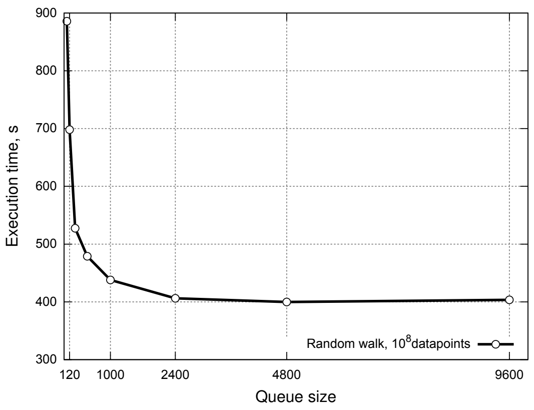

Fig. 9. Impact of queue size on the speedup

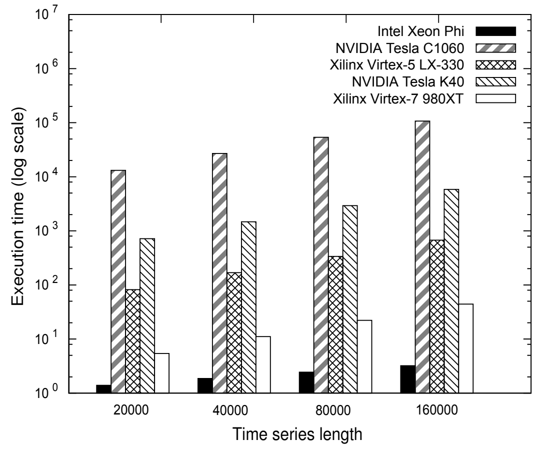

Fig. 10. Comparison of performance

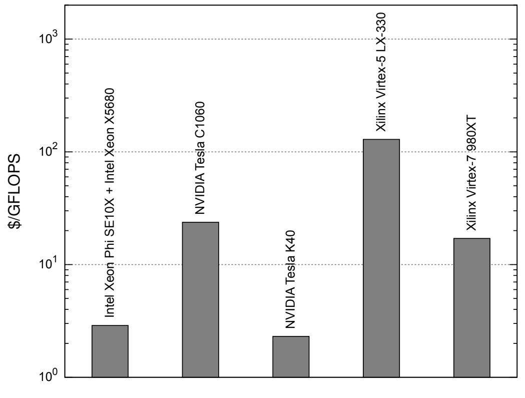

Fig. 11. Comparison of performance cost

problem [15].

One of the earliest researches [16] considers the time series as a set of multi-dimensional rectangles and constructs its spatial index to perform search of subsequences as spatial queries to the index. This work serves as a basis for many researches devoted to improving subsequence matching performance. For instance, the DualMatch algorithm [17] and its improved version, the GeneralMatch algorithm [18], split the time series into non-overlapping windows and query pattern into sliding windows. In [19] and [7] an index interpolation to overcome the storage and CPU time overhead is used. All the former techniques are based on lower bounding of the

Euclidean distance.

Nowadays the Dynamic Time Warping (DTW) [4] measure is recognized as the best similarity measure for many time series subject domains [5]. DTW-based subsequence searching techniques include a suffix tree [20], a segment-based approach based on piecewise time warping [21], an index-based approach based on prefix querying [22] and a comparison optimization approach [23]. An algorithm proposed in [8] integrates all the possible existing speedup techniques and most likely it is the fastest of the existing subsequence matching algorithms.

All the aforementioned algorithms basically focus on reducing the calling times of DTW calculation, not accelerating DTW itself. However, because of its complexity, DTW still takes a large part of the total application runtime. Due to this reason science community tries to exploit the effectiveness of parallel hardware to accelerate subsequence matching. Existing researches assume allocation of DTW calculation of different subsequences into different processing elements, e.g. computer-cluster [9], multicore [10], FPGA and GPU [6].

Despite the existing research devoted to use of the Intel Xeon Phi coprocessor in data mining [24] in our opinion not enough attention is paid to applying this coprocessor for time series subsequence matching. In this work we adapt the algorithm [8] to the Intel Xeon Phi coprocessor (what was done for the first time, to the best of our knowledge).

VI. Conclusion

In this paper we have described the parallel algorithm for time series subsequence matching adapted for CPU accompanied with the Intel Xeon Phi coprocessor. The algorithm uses data parallelism and OpenMP technology. To establish interaction between CPU and the coprocessor the *offload* execution mode is used, when the program is run by the CPU while some code and data are transmitted to the coprocessor. To provide high operational intensity of the calculations on the coprocessor the algorithm supports a queue of subsequences on CPU, where subsequences are candidates to be offloaded to the coprocessor to calculate DTW. Experiments on synthetic and real data sets have shown that our algorithm outperforms analogous algorithms for GPU and FPGA.

As future work we plan to extend our algorithm for the case of a cluster system based on nodes equipped with the Intel Xeon Phi coprocessors.

References

[1] V. Epishev, A. Isaev, R. Miniakhmetov, A. Movchan, A. Smirnov, L. Sokolinsky, M. Zymbler, and V. Ehrlich, “Physiological data mining system for elite sports,” *Bull. of South Ural State University. Series: Comput. Math. and Soft. Eng.*, vol. 2, no. 1, pp. 44–54, 2013.

[2] M. Dyshaev and I. 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.*, vol. 2, no. 4, pp. 103–108, 2013.

[3] S. Abdullaev, O. Lenskaya, A. Gayazova, D. Sobolev, A. Noskov, O. Ivanova, and G. 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.*, vol. 3, no. 1, pp. 17–32, 2014.

[4] D. J. Berndt and J. Clifford, “Using dynamic time warping to find patterns in time series,” in *KDD Workshop*, U. M. Fayyad and R. Uthurusamy, Eds. AAAI Press, 1994, pp. 359–370.

[5] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. J. Keogh, “Querying and mining of time series data: experimental comparison of representations and distance measures,” *PVLDB*, vol. 1, no. 2, pp. 1542–1552, 2008.

[6] D. Sart, A. Mueen, W. A. Najjar, E. J. Keogh, and V. Niennattrakul, “Accelerating dynamic time warping subsequence search with gpus and fpgas,” in *ICDM*, G. I. Webb, B. Liu, C. Zhang, D. Gunopulos, and X. Wu, Eds. IEEE Computer Society, 2010, pp. 1001–1006.

[7] S.-H. Lim, H. Park, and S.-W. Kim, “Using multiple indexes for efficient subsequence matching in time-series databases,” in *DASFAA*, ser. Lecture Notes in Computer Science, M.-L. Lee, K.-L. Tan, and V. Wuwongse, Eds., vol. 3882. Springer, 2006, pp. 65–79.

[8] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh, “Searching and mining trillions of time series subsequences under dynamic time warping,” in *KDD*, Q. Yang, D. Agarwal, and J. Pei, Eds. ACM, 2012, pp. 262–270.

[9] N. Takahashi, T. Yoshihisa, Y. Sakurai, and M. Kanazawa, “A parallelized data stream processing system using dynamic time warping distance,” in *CISIS*, L. Barolli, F. Xhafa, and H.-H. Hsu, Eds. IEEE Computer Society, 2009, pp. 1100–1105.

[10] S. Sharanyan, K. Arvind, and G. Rajeev, “Implementing the dynamic time warping algorithm in multithreaded environments for real time and unsupervised pattern discovery,” in *ICCCT*, D. of Computer Science and M. N. N. I. o. T. Engineering, Eds. IEEE Computer Society, 2011, pp. 394–398.

[11] A. Duran and M. Klemm, “The intel® many integrated core architecture,” in *HPCS*, W. W. Smari and V. Zeljkovic, Eds. IEEE, 2012, pp. 365–366.

[12] “tornado susu” supercomputer. South Ural State University. [Online]. Available: www.supercomputer.susu.ru/en/computers/tornado/

[13] Nvidia tesla gpu accelerators. NVIDIA Corp. [Online]. Available: www.nvidia.com/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf

[14] Virtex-7 fpgas family. Xilinx Inc. [Online]. Available: www.xilinx.com/publications/prod_mktg/Virtex7-Product-Brief.pdf

[15] T.-C. Fu, “A review on time series data mining,” *Eng. Appl. of AI*, vol. 24, no. 1, pp. 164–181, 2011.

[16] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, “Fast subsequence matching in time-series databases,” in *SIGMOD Conference*, R. T. Snodgrass and M. Winslett, Eds. ACM Press, 1994, pp. 419–429.

[17] Y.-S. Moon, K.-Y. Whang, and W.-K. Loh, “Duality-based subsequence matching in time-series databases,” in *ICDE*, D. Georgakopoulos and A. Buchmann, Eds. IEEE Computer Society, 2001, pp. 263–272.

[18] Y.-S. Moon, K.-Y. Whang, and W.-S. Han, “General match: a subsequence matching method in time-series databases based on generalized windows,” in *SIGMOD Conference*, M. J. Franklin, B. Moon, and A. Ailamaki, Eds. ACM, 2002, pp. 382–393.

[19] W.-K. Loh, Y.-S. Moon, and J. Srivastava, “Distortion-free predictive streaming time-series matching,” *Inf. Sci.*, vol. 180, no. 8, pp. 1458–1476, 2010.

[20] S.-W. Kim, J. Yoon, S. Park, and T.-H. Kim, “Shape-based retrieval of similar subsequences in time-series databases,” in *SAC*. ACM, 2002, pp. 438–445.

[21] S. Park, S.-W. Kim, and W. W. Chu, “Segment-based approach for subsequence searches in sequence databases,” in *SAC*. ACM, 2001, pp. 248–252.

[22] S. Park, S.-W. Kim, J.-S. Cho, and S. Padmanabhan, “Prefix-querying: An approach for effective subsequence matching under time warping in sequence databases,” in *CIKM*. ACM, 2001, pp. 255–262.

[23] M.-S. Kim, S.-W. Kim, and M. Shin, “Optimization of subsequence matching under time warping in time-series databases,” in *SAC*, H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, Eds. ACM, 2005, pp. 581–586.

[24] A. Heinecke, M. Klemm, D. Pflüger, A. Bode, and H.-J. Bungartz, “Extending a highly parallel data mining algorithm to the intel ® many integrated core architecture,” in *Euro-Par Workshops (2)*, ser. Lecture Notes in Computer Science, M. Alexander, P. D’Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. D. Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. L. Scott, J. L. Träff, G. Vallée, and J. Weidendorfer, Eds., vol. 7156. Springer, 2011, pp. 375–384.