Best-match Time Series Subsequence Search on the Intel Many Integrated Core Architecture

Mikhail Zymbler

South Ural State University, Chelyabinsk, Russia

Abstract. Subsequence similarity search is one of the basic problems of time series data mining. Nowadays Dynamic Time Warping (DTW) is consideredered as the best similarity measure. However despite various existing software speedup techniques DTW is still computationally expensive. There are approaches to speed up DTW computation by means of parallel hardware (e.g. GPU and FPGA) but accelerators based on the Intel Many Integrated Core architecture have not been payed attention. The paper presents a parallel algorithm for best-match time series subsequence search based on DTW distance for the Intel Xeon Phi coprocessor. The experimental results on synthetic and real data sets confirm the efficiency of the algorithm.

1 Introduction

Subsequence similarity search is one of the basic problems of time series data mining and appears in various applications, e.g. climate modeling [1], medical monitoring [6], economic forecasting [5], etc. Best-match time series subsequence search 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, whose distance from the query is the minimum among all the subsequences.

Nowadays the Dynamic Time Warping (DTW) [2] is considered as the best similarity measure in many time series applications [3]. DTW is computationally expensive and there are many software approaches that have been proposed to solve this problem, e.g. lower bounding [3], computation reusing [13], data indexing [9], early abandoning [11], etc. However, DTW is still very time-consuming and there are approaches to speed up DTW computation by means of parallel hardware, e.g. computer-cluster [15], multicore [14], FPGA and GPU [13, 16] but none for the Intel Many Integrated Core [4] accelerators.

In this paper we present a parallel algorithm for best-match subsequence search based on DTW distance adapted for a central processor unit (CPU) accompanied with the Intel Xeon Phi many-core coprocessor. The remainder of the paper is organized as follows. Section 2 gives the formal definition of the problem and briefly considers the Intel Xeon Phi architecture and programming model and discusses related work. The suggested algorithm is described in section 3. Experimental results evaluating the algorithm are presented in section 4. Section 5 contains summary and directions of future work.

2 Background and Related Work

2.1 Problem Definition

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

A *query* Q is a time series to be found in T; n is a length of the query, $n \ll N$.

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. $T_{im} = t_i, t_{i+1}, \ldots, t_{i+m-1}$, where $1 \leq i \leq N$ and $i + m \leq N$.

Best-match subsequence search aims to finding a subsequence T_{in} whose Dynamic Time Warping distance from Q is the minimum among all the subsequences, i.e. $DTW(T_{in}, Q) < DTW(T_{mn}, Q)$ for any m such that $1 \leq m \leq N-n$.

In this paper we do not consider *local-best-match search* [16], which aims to finding *all* the subsequences T_{in} whose distance from Q is the minimal among their neighboring subsequences whose distance from Q is under specified threshold.

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.

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

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

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

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. 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).

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 [7, 15]. Research devoted to acceleration of DTW computation includes the following.

The SPRING algorithm [12] uses computation-reuse technique. However, this technique squeezes the algorithm's applications because data-reuse supposes non-normalized sequence. In [9] indexing technique to speed up the search was used, which need to specify the query length in advance. Authors of [8] suggested multiple index for various length queries. Lower bound technique was proposed in [7] and prunes off unpromising subsequences using the lower bound of DTW distance estimated in a cheap way. The UCR-DTW algorithm [11] 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 calling times of DTW computation, not accelerating DTW itself. However, because of its complexity, DTW still takes a large part of the total application runtime. That is why there are approaches exploiting parallel hardware by means of allocation of DTW computing of different subsequences into different processing elements.

In [14] subsequences starting from different positions of the time series are sent to different Intel Xeon processors, and each processor computes DTW. In [15] different queries are distributed onto different cores, and each subsequence is sent to different cores to be compared with different queries. GPU implementation [17] parallelize the generation of the warping matrix but still process the path search serially. GPU implementation proposed in [13] utilizes the same ideas as in [14]. FPGA implementation described in [13] 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 due to lack of insight into the FPGA can not be applied in big-scale tasks. To address these problems in [16] 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 a parallel algorithm of the time series subsequence DTW-based similarity search on the Intel Xeon Phi many-core coprocessor is presented where the UCR-DTW serial algorithm is used as a basis.

3 Best-match Subsequence Search on the Intel Xeon Phi

Development of the best-match subsequence search algorithm consists of the following steps, which will be discussed in detail further.

At first, we developed a parallel version of the UCR-DTW serial algorithm [11] using OpenMP technology. 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, i.e. CPU and the Intel Xeon Phi run parallel algorithm developed at the previous step. Here we used *offload* mode to transfer code and data to the coprocessor. Experiments, where we varied the portion size of data to be transferred 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 where the coprocessor is exploited only for DTW computations whereas CPU performs pruning and supports a queue of subsequences for the coprocessor. This significantly increased the operational intensity of the computations on the coprocessor and experiments shown acceptable performance of the algorithm.

3.1 Parallel Algorithm for CPU

The UCR-DTW algorithm proposed in [11] is one of the fastest existing subsequence matching algorithms. This algorithm (Fig. 1) 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. Here the `bsf` (best-so-far) variable stores the distance to the most similar subsequence.

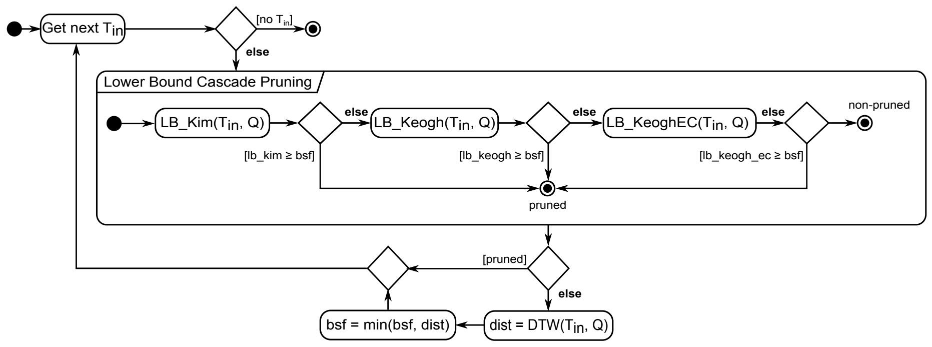


Fig. 1. Serial algorithm

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-length portions and each portion is processed by a separate OpenMP-thread. Let P denotes a number of OpenMP-threads, then a portion assigned for processing to the k-th thread, $0 \leq k \leq P-1$, is defined as a subsequence T_{sl}, where

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

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

It means that the head part of every portion except first overlaps with the tail part of previous portion in $n-1$ data points. This permits to keep possible resulting subsequences from the junctions of portions.

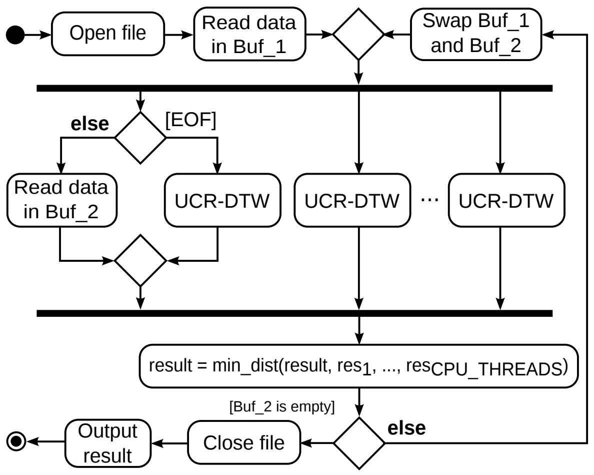


Fig. 2. Parallel algorithm for CPU

Here `UCR-DTW` is a subroutine that implements the original serial algorithm. 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. Master thread reads new data from a file simultaneously with processing of data that have been read.

The obtained algorithm is ready to run on the Intel Xeon Phi in *native* mode but experiments have shown that (Fig. 6) although parallel algorithm expectedly surpasses the serial algorithm it works slower on the coprocessor than on CPU. 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.

3.2 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 the coprocessor. Here α is a parameter that determines a proportion of data to be transferred to the coprocessor. We use *offload* mode to organize data exchange between CPU and the coprocessor. The `min_dist` subroutine chooses a subsequence with minimal value of DTW.

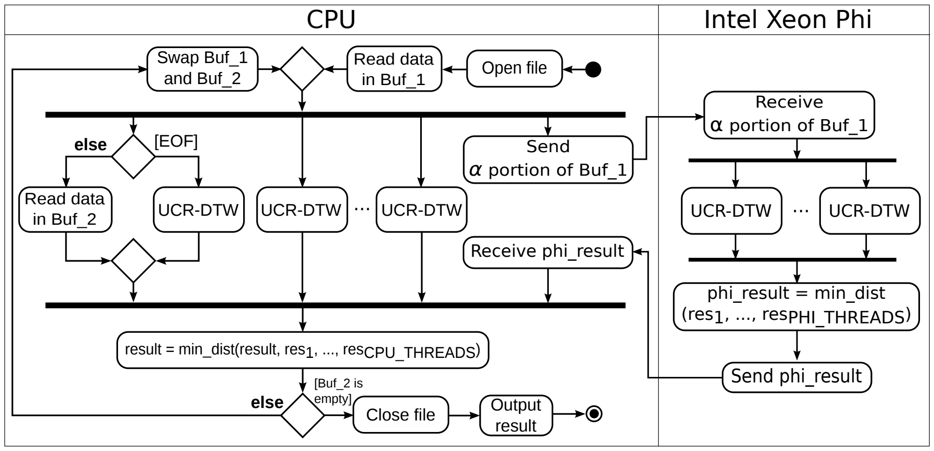


Fig. 3. Naïve parallel algorithm for CPU and the Intel Xeon Phi

As well as in the previous step we evaluated the obtained algorithm and experiments have shown that regardless of the α value the 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 compute DTW.

3.3 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 algorithm is based on the following two ideas. First, the coprocessor should be exploited only for DTW computation whereas CPU prunes unpromising subsequences and computes DTW in case if it really does not have another job. Second, CPU should support a queue of subsequences that are candidates to be offloaded to the coprocessor to compute DTW for each candidate subsequence.

To reduce amount of data transferred to the coprocessor the following technique has been used. Queue does not store each candidate subsequence T_{in} but stores its corresponding tuple (i, A), where A is an n-element array containing LB_{Keogh} lower bounds for each position of the subsequence which is used for

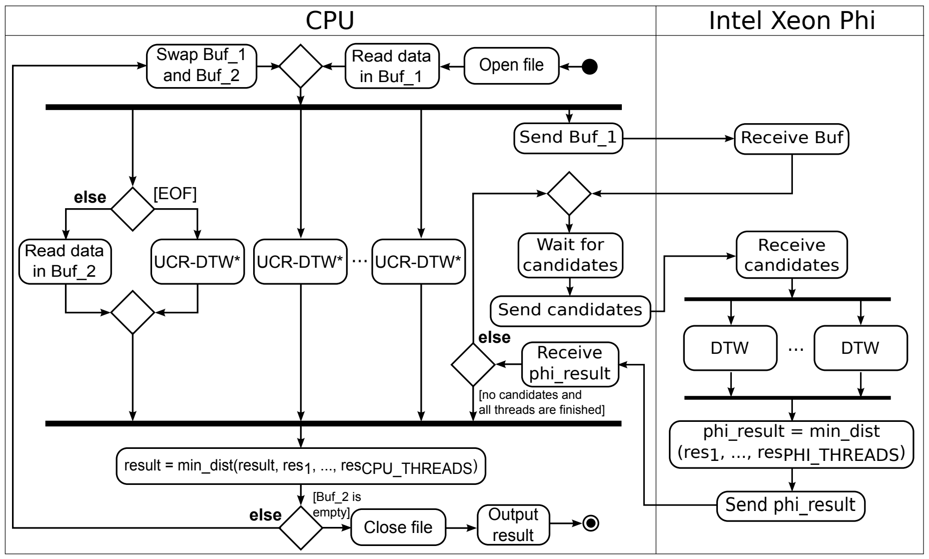


Fig. 4. Advanced parallel algorithm for CPU and the Intel Xeon Phi

early abandoning of DTW [11]. CPU offloads current part 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 (i.e. W is a parameter of the algorithm).

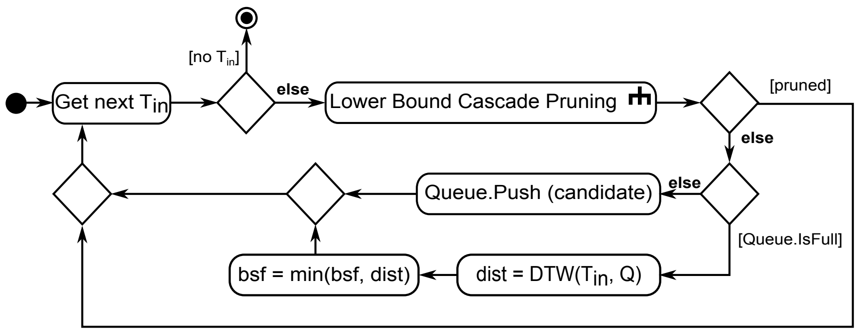


Fig. 5. UCR-DTW* subroutine

One of the CPU threads is declared as a master and the rest as workers. At start master sends a buffer with the current portion of 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.

A worker's behavior is depicted in the Fig. 5. Worker computes cascade estimates for the current 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 computes 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 computed among the most similar subsequence found on the CPU and same that found on the coprocessor.

4 Experimental Results

To evaluate the developed algorithm we performed experiments on the Tornado SUSU supercomputer's node (Tab. 1 contains its specifications).

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

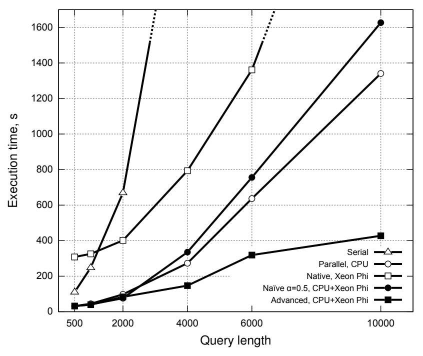
Specifications	Processor	Coprocessor
Model	Intel Xeon X5680	Intel Xeon Phi SE10X
Cores	6	61
Frequency, GHz	3.33	1.1
Threads per core	2	4
Peak performance, TFLOPS	0.371	1.076

We measured search runtime while varying query length. Experiments have been performed on synthetic and real time series. We also investigated the impact of queue size on the speedup and compared performance of the algorithm with analogues for GPU and FPGA.

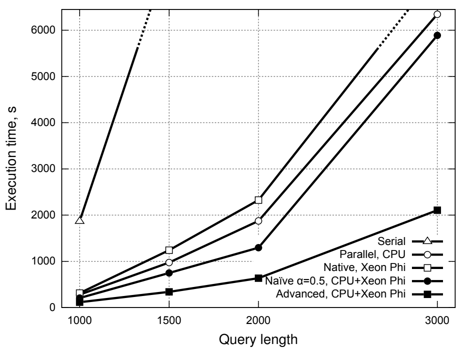
4.1 Performance

In the first experiment we used synthetic time series generated by one-dimensional random walk [10] comprising of 100 million data points. Experimental results (Fig. 6a) 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 experiment 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). Our algorithm shows (Fig. 6b) a three times higher performance than the parallel algorithm for CPU only.



(a) Random walk data set

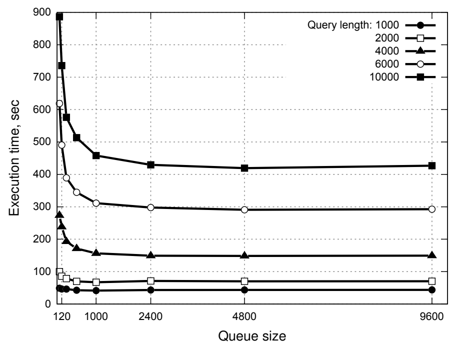


(b) ECG data set

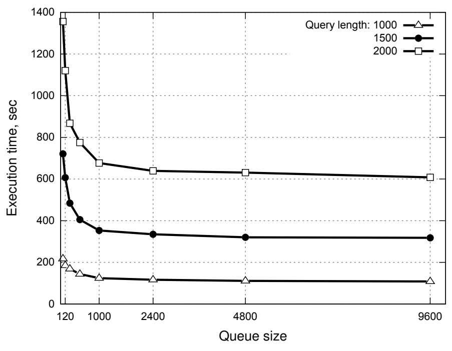
Fig. 6. Performance of the algorithm

4.2 Impact of Queue Size

Results of experiments investigating the impact of queue size on performance are depicted in Fig. 7. In the current experimental environment, i.e. hyperthreading factor of the coprocessor h is 4, number of cores of the coprocessor C is 60^1, 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 in 4.1 have been achieved with this queue size.



(a) Random walk data set



(b) ECG data set

Fig. 7. Impact of queue size on the speedup

[1] One of the coprocessor's cores is not involved in computations as it is recommended by the Intel Xeon Phi programmer's manual.

4.3 Comparison with Analogues

We compared the performance of our algorithm with analogues for GPU and FPGA developed in [13]. We repeated the experiments presented in that paper using the same data set and query length.

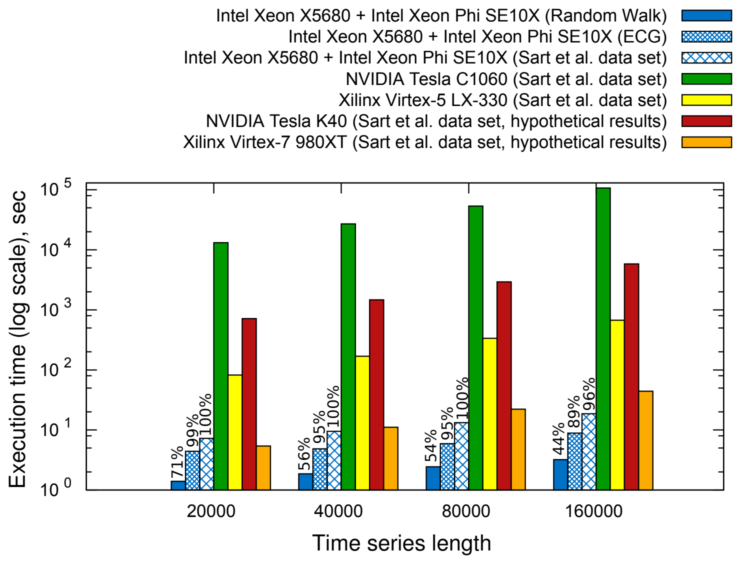


Fig. 8. Comparison of performance

The results of the experiments are depicted in Fig. 8, here percentage on the top of the bar indicates a proportion of subsequences that have not been pruned and subjected to the DTW calculation in our experiments. We also add to the chart results of experiments on random walk and ECG data sets.

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) and Xilinx Virtex-7 980XT (0.99 TFLOPS) 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 an approach to best-match time series subsequence search under DTW distance on the Intel Many Integrated Core architecture has been presented. The parallel algorithm combines capabilities of CPU and the Intel Xeon Phi many-core coprocessor. The coprocessor is exploited only for DTW computations whereas CPU performs lower bounding, prepares subsequences for the coprocessor and computes DTW as a last resort. CPU supports a queue of candidate subsequences to be offloaded to the coprocessor to compute DTW. 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 in the following directions: implement our algorithm for the cases of several coprocessors and cluster system based on nodes equipped with the Intel Xeon Phi coprocessor(s) and apply our approach to the task of local-best-match time series subsequence search.

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. 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.
9. 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, *DASFAA*, volume 3882 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 2006.
10. Karl Pearson. The problem of the random walk. *Nature*, 72(1865):294, 1905.
11. 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.
12. 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.
13. 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.
14. 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.
15. 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, *CISIS*, pages 1100–1105. IEEE Computer Society, 2009.
16. 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.
17. 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.