Energy-efficient Pipelined DTW Architecture on Hybrid Embedded Platforms

Hanqing Zhou[1], Xiaowei Xu[1,2], Yu Hu[1], Guangyu Yu[1], Zeyu Yan[1], Feng Lin[2], Wenyao Xu[2]
1 School of Optical and Electronic Information, HUST, Wuhan, China
2 Department of Computer Science and Engineering, SUNY at Buffalo, New York, USA
{xuxiaowei, bryanhu}@hust.edu.cn; {flin28, wenyaoxu}@buffalo.edu

***Abstract*—It is predicted that fifty billion sensor-based devices are to be connected to the Internet by 2020 with the fast development of Internet of Things (IoT). Stream data mining on these tremendous sensor-based devices has become an urgent task. Dynamic time warping (DTW) is a popular similarity measure, which is the foundation of stream data mining. In the last decade, DTW has been well accelerated with software and reconfigurable hardware optimizations. However, energy-efficiency has not been considered, which is critical for data mining on these devices. In this paper, we propose an energy-efficient DTW acceleration architecture for stream data mining on sensor-based devices, which is based on a hybrid embedded platform of ARM and field programmable gate array (FPGA). Software optimizations for DTW are implemented on ARM, and pipelined DTW is implemented on FPGA for further accelerations. A pilot study is performed with three widely adopted stream data mining tasks: similarity search, classification, and anomaly detection. The results show that the performance improvements vary for different configurations, and the achieved average speedup and energy efficiency improvement are 7.52x and 4.23x, respectively.**

I. Introduction

With the popularity of Internet of Things (IoT), more and more embedded devices are connected to IoT. The Cisco Internet Business Solutions Group (IBSG) predicts that 50 billion devices will be connected to the Internet by 2020 [1], most of which are sensor-based embedded devices. Thus, a huge number of stream data will be produced by them. As many applications require real-time data mining such as speech recognition [2], activity recognition [3], it is an urgent task to perform stream data mining on sensor-based devices, where the constraints of computation capacity and energy capacity should be considered.

Distance measure between time series plays an important role in stream data mining, which is the foundation of higher data mining tasks, such as classifications and clusterings. Dynamic time warping (DTW) is one of the best distance measures according to a recent comparison study of several distance measures with 44 datasets [4]. It is widely applied to different fields, such as speech recognition, financial analysis and network traffic monitoring [5]. However, DTW has a quadratic time complexity, which is computation-expensive for huge data processing on sensor-based devices.

DTW has been well optimized with software and hardware methods to solve the obstacle of computation complexity. Lower bound [6] [7] is a powerful optimization method, which can prune a lot of sequences in many tasks such as similarity search and classification. Early abandon [8] is also very effective. Rakthanmanon *et al.* [8] cascaded multiple stages of software optimizations, which is considered as the most powerful implementation for similarity search to date. DTW in a streaming manner is proposed by sakurai *et al.* [9], which achieves a linear time complexity, however, allows false dismissals. The majority of these techniques achieve speedup by reducing the invoking times of DTW rather than accelerating DTW itself. However, the DTW calculation still accounts for about 80% of the total executing time [10] [11]. Therefore, reconfigurable hardware is adopted for further acceleration. Wang *et al.* [11] implemented a high-throughput DTW framework for similarity search on FPGAs. The proposed structure of processing element (PE) ring exploits the fine-grained parallelism of DTW and achieves significant speedup. Hardware acceleration of DTW has also been implemented on graphic processing units (GPU) [12] [13].

In IoT, a typical device runs on a system-on-a-chip (SoC) system for high performance and energy efficiency [14]. The reconfigurable hardware part of the SoC system handles the critical functions (or bottlenecks). The software part (e.g., central processing unit (CPU)) of SoC is responsible for software optimizations, task scheduling and data management. Usually, hardware/software partitioning is preferred on SoC for energy efficiency. Recently, Tarango *et al.* [14] proposed an implementation of DTW based on instruction set extensions for similarity search task. Critical codes in software are replaced with extended instructions customized on FPGAs. However, the processing flow of DTW is not optimized, and it does not utilize the potential parallelism of DTW. Furthermore, the energy-efficiency and performance of the used softcore are relatively low if not well optimized compared with hardcore.

In this paper, we propose a general and pipelined DTW acceleration architecture based on ARM+FPGA for stream data mining on sensor-based devices.We utilize the potential parallelism of DTW, and implement a parameterized and pipelined DTW accelerator on the reconfigurable hardware. Software optimizations of DTW for different tasks are implemented on hardcore ARM, and the DTW accelerator serves as a sub-function. With collaboration of software and hardware optimizations, the performance of the system has the potential to get significant improvements. We conduct a series of experiments with three widely adopted stream data mining tasks, similarity search, classification, and anomaly detection. Public-available datasets are also adopted. Compared with implementations on ARM, the speedup and energy reduction vary for different tasks. The average speedup and energy efficiency improvement are 7.52x and 4.23x, respectively.

978-1-5090-0172-9/15/$31.00 © 2015 IEEE

II. Background

A. Time Series and Subsequences

A time series S is a sequence of data measures, which is represented typically with uniform time intervals shown as below:

$$S = (t_1, s_1), (t_2, s_2), ..., (t_i, s_i), ..., (t_n, s_n). \quad (1)$$

For convenience, S is usually simplified as follows:

$$S = s_1, s_2, ..., s_i, ..., s_n. \quad (2)$$

As a time series is usually very long, segmentation is used to divide the long sequence into short subsequences for convenient processing. Thus, one subsequence of S can be rewritten as follows:

$$S_k = s_i, s_{i+1}, ..., s_{i+k-1}, \quad (3)$$

where k is the length of the subsequences. For convenient discussion, subsequences are also expressed as sequences.

B. Dynamic Time Warping with Constraints

DTW is a robust distance measure for time series sequences. Suppose there are two time sequences as shown in Fig.1(a), a sequence C of length n as a candidate, and a sequence T of length m as a training template, where

$$C = c_1, c_2, ..., c_i, ..., c_n,\ T = t_1, t_2, ..., t_i, ..., t_m. \quad (4)$$

Usually normalization and representation are applied to preprocess the sequences. Time series sequences must be normalized to make a meaningful comparisons [8]. Z-normalization is adapted in this work to remove offsettings and amplitudes as shown below:

$$\mu_T = \frac{1}{m}\sum_{k=1}^{m} t_k,\ \sigma_T^2 = \frac{1}{m}\sum_{k=1}^{m} t_k^2 - \mu_T^2,\ t'_k = \frac{t_k - \mu_T}{\sigma_T}. \quad (5)$$

To measure the similarity of these two time sequences, DTW creates a n-by-m matrix, MT. The value of the (i^{th}, j^{th}) element in MT represents the distance, $d(c''_i, t''_j)$, between points c_i and t_j as

$$MT(i,j) = d(c''_i, t''_j), \quad (6)$$

which is called *distance matrix calculation* as shown in Fig.1(b). There are many effective distance metric such as Manhattan distance, Euclidean distance, for *distance matrix calculation*. We choose the widely-used Manhattan distance as shown below:

$$d(c''_i, t''_j) = |c''_i - t''_j|. \quad (7)$$

With the distance matrix, the warping path can be derived. There are three well-known constrains for the warping path in DTW: *boundary conditions, continuity condition* and *monotonic condition*. *boundary conditions* means that the first/last point of C must correspond to the first/last point of T. *continuity condition* means that each element of the warping path in the matrix, MT, must have two elements of the warping path around it except the first and the last points. *monotonic condition* requires that the extending direction of the warping path is right, or top or top-right. The shortest warping path through the matrix is derived, using [15]:

$$CD(i,j) = MT(i,j) + min\begin{cases} CD(i,j-1) \\ CD(i-1,j) \\ CD(i-1,j-1) \end{cases}, \quad CD(0,0)=0, CD(0,j)=CD(i,0)=\infty,\ 1\leq i\leq n; 1\leq j\leq m, \quad (8)$$

where $CD(i,j)$ is the current minimum cumulative distance for $MT(i,j)$. This procedure is called *warping path calculation*. After that, the minimized cumulative distance can be derived. Finally, the DTW distance is calculated as shown below:

$$dtw = \sqrt{CD(n,m)}. \quad (9)$$

Usually the calculation combination of *distance matrix calculation* and *warping path calculation* are regarded as DTW matrix calculation. The time complexity of DTW is $O(n^2)$.

The Sakoe-Chiba [16] band is used as DTW constraint as shown in Fig.1(b). DTW constraint, R, can reduce the available DTW path thus achieves speedup. R is defined as the rate of the warping length over the whole sequence, and it varies from 0 to 100%. It can also avoid some unpractical matchings, e.g., the matching of the first point of one sequence to the last point in another sequence can be eliminated. DTW constraint is effective for many applications, and the choice of R depends on specific applications and configurations.

For the simplicity of the presentation, other optimization methods (e.g., lower bounds, early abandon and reordering) are not presented here. Readers can refer to [8] for details.

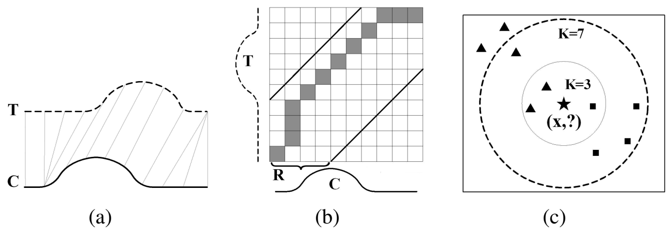


Fig. 1. (a) DTW matching: T and C are two sequences, and the lines indicate the matching relationship; (b) DTW warping path with Sakoe-Chiba band $R = 40\%$: the gray squares form the warping path between T and C on the distance matrix, and the area between two bold lines are available areas assigned by DTW constraints; (c) kNN Rule: X is classified by a majority vote of k neighbors. The star in the center is classified as triangle when $k = 3$, while if $k = 7$, the star is classified as square.

C. k-Nearest Neighbors

The k Nearest Neighbors (kNN) algorithm is one of the well-investigated methods for pattern classification, which is used to determine the class of an unclassified point by the class of the k nearest points of it [17]. k varies for different applications. An example of how kNN works is shown in Fig.1(c). kNN can also be applied to anomaly detections [18].

III. DTW Acceleration Architecture

In this section, an overview of the proposed DTW acceleration architecture is presented, emphasizing on the DTW interface and DTW-FPGA modules. The features of DTW interface are also analysed and discussed in detail.

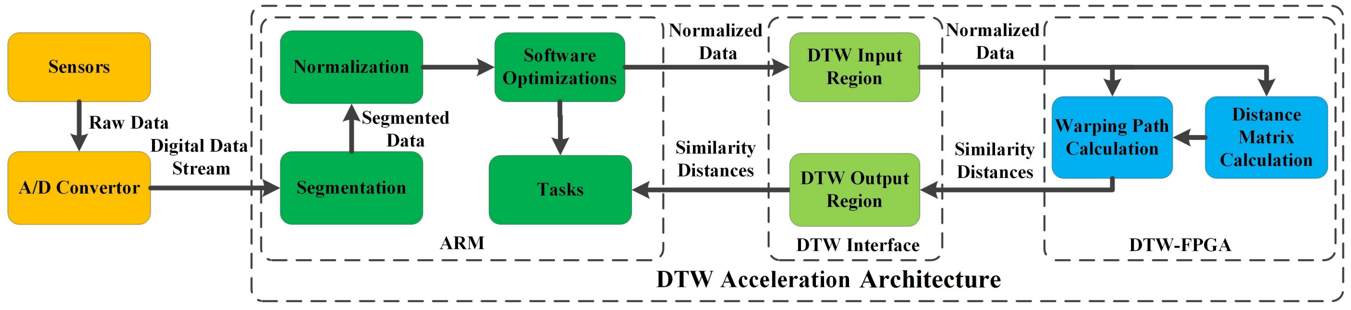


Fig. 3. Work flow of the proposed DTW acceleration architecture.

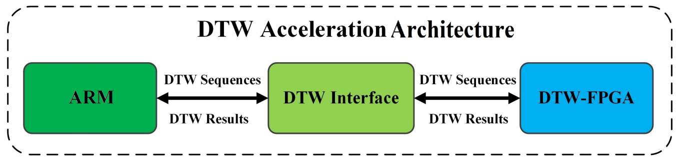


Fig. 2. Hardware structure of the proposed DTW acceleration architecture.

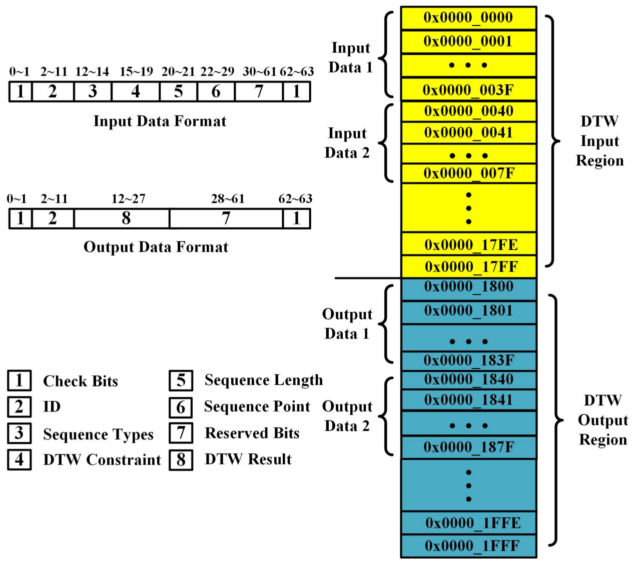


Fig. 4. Data format of DTW interface: check bits, '10' at the head of the frame, and '01' at the tail of the frame; ID, ID of the current sequence in ascending order; sequence types, the type of the sequence; DTW Constrains, the warping constraint radius (%); sequence length, the length of current sequence; sequence point, one point in current sequence; reserved bits, for further optimizations; DTW result, the DTW result of the sequence with its ID.

A. *Overview*

The structure of the proposed DTW acceleration architecture is shown in Fig. 2, which is based on a general SoC platform, Cyclone V SoC by Altera [19]. There are three components of the DTW acceleration architecture: ARM, DTW interface and DTW-FPGA.

The workflow and functionality of these three components are shown in Fig. 3. ARM is mainly responsible for implementation of software optimizations. The software optimization methods for DTW including lower bound [4] [6] [7] and early abandon [8] are implemented. Once DTW need to be calculated, the input data for DTW is sent to DTW interface. With some processing time (much faster than the software calculation), DTW interface returns the DTW result. If multiple tasks (e.g., classification, anomaly detection and similarity search) are implemented, transaction management is also handled by ARM. DTW-FPGA is a parameterized DTW implementation on FPGAs. It receives DTW input data, calculates DTW and returns the result to DTW interface. DTW interface is responsible to handle data and control transformation between ARM and DTW-FPGA. It is connected to ARM with a high-bandwidth interconnect backbone. DTW interface serves as a general-purpose memory accessible from DTW-FPGA through a high-speed bus.

B. *Parameterized DTW Interface*

DTW interface is implemented with a dual-port FPGA memory block, which is compiled with Golden System Reference Design (GSRD) and added to the device tree of ARM [20]. DTW interface has two regions, DTW input region and DTW output region, which can be accessed by ARM and DTW-FPGA. The DTW input region stores the sequences that need to be calculated, while the output region caches the DTW results. When there is DTW calculations, the task on ARM first sends the template sequences, and then sends the packaged time series sequence to the DTW input region. When the transmission is completed, the task will repeatedly read the data in the DTW output region of DTW interface, and the DTW-FPGA will process DTW calculation once it detects the correct format in the DTW input region. When the calculation is fulfilled, DTW-FPGA writes the result to the DTW output region. Then, the task detects the correct format in the DTW output region. Here, one DTW calculation is completed.

The parameterizations of DTW interface is achieved with configurable parameters, which can be set on the fly. The data format of the two regions is illustrated in Fig. 4. As there are many sequences for DTW calculation, ID is introduced to distinguish different sequences. The DTW interface supports a variety of parameters including sequence length and DTW constraint. With $Sequence\ Type$ and ID in the DTW input region, the number of template sequences can also be supported. Burst transmission is supported by DTW interface to increase throughput. As the throughput of DTW-FPGA is relatively high, burst transmission can decrease the processing time in DTW interface. As shown in Fig. 4, the size of the DTW input/output region is designed to be much larger than the size of one input/output data in order to support burst receiving. The task can send n sequences to the DTW input region for calculation. Rather than detect the data format of the

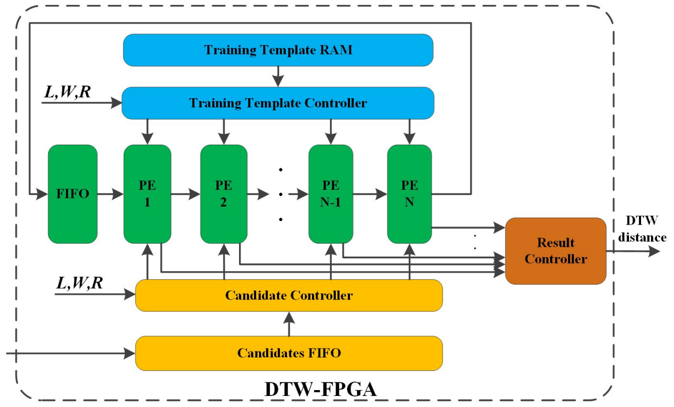


Fig. 5. Architecture of DTW-FPGA module (L: length of sequences; W: the number of training templates in DTW module; R: DTW constraint).

first output data address, the task will check the n^{th} output data address. Once the data format of the n^{th} output data address is correct, all the n DTW calculations are fulfilled, and the task can read all the n DTW results in a burst manner. The detailed discussion of the characterization of DTW interface is presented in Subsection III-D.

C. Pipelined DTW-FPGA

DTW-FPGA processes the pipelined DTW calculation and sends results to DTW interface. As shown in Fig. 5, N Processing Elements (PE) and one first in first out (FIFO) module are linked with each other like a ring, which is a modification of the work presented in [11]. Unlike the work in [11], the streaming DTW [9] is not adopted in order to guarantee accuracy. The training template RAM stores training templates. The candidate controller and training template controller receive configuration parameters from DTW interface. These parameters include length of sequences, the number of training templates in DTW module and DTW constraint. The two controllers achieve parameterizations by controlling when to send candidates and templates to which PE. The function of PE is to calculate one column of the DTW matrix. With the feature of DTW matrix calculation, the left, left-bottom and bottom elements in DTW matrix are required for the processing of PEs, which are easily achieved with connections between PEs. When the candidate length is larger than the number of PE, the FIFO besides the PE1 is used to store temporary results of PEN to support large candidate length, otherwise stores boundary conditions for PE1. The result controller is responsible for selecting the right result and send it to the output port.

Fig. 6 shows the work flow of DTW-FPGA module. The PE number is three, and the sequence length of candidates and queries are also three. The dimension of the data streams is one. The query sequence is $Q = \{3, 2, 4\}$, and there are two candidate sequences: $C_1 = \{1, 4, 3\}$ and $C_2 = \{1, 3, 1\}$. At the first cycle, the first tuple of C_1 is sent to PE1 and the [1,1] cell of DTW matrix between C_1 and Q is calculated. In the second cycle, the second tuple $C_1(2) = 4$ is sent to PE2, and cell [1, 2] and [2,1] are calculated at the same time. At the 3^{th} cycle, the third tuple $C_1(3) = 3$ is sent to PE3, and cell [1, 3], [2, 2] and [3, 1] are calculated. At the 4^{th} cycle, PE1

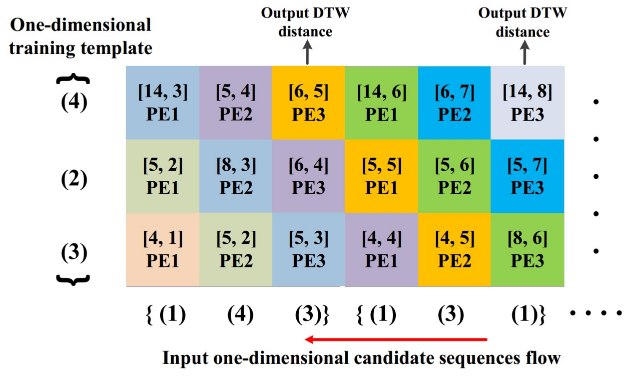


Fig. 6. An illustration of DTW-FPGA workflow: Two 3*3 matrix are the DTW matrixes between query Q and candidates C_1 and C_2. (a) is a one dimensional tuple of candidates and queries. In [b, c], b is the accumulated DTW distance and c is the cycle time. The item below [b, c] indicates which PE processes the matrix cell it locates.

has finished the calculation of the first column, and the first tuple of C_2 is sent to PE1. At the 5^{th} cycle, the calculation of DTW matrix between C_1 and Q is completed. The result controller handles the result from PE3 to the output port. The iteration processing continues. The number of cycles needed to pump one DTW distance is the length of the candidates.

D. Characterization of DTW Interface

The sequence length, N, and the burst transmission number, M, are analysed to discuss characterizations of the parameterized DTW interface in this subsection. Other parameters are also discussed. For the discussion, a task is implemented on ARM to calculate DTW for 512 times with different M and L. As the processing time of DTW interface and DTW calculation is independent of data, the test data is produced randomly. Let the time of sending a sequence be T_{IO}, and the time of calculate DTW distance be T_{DTW}. The average execution time of one DTW calculation is represented as $T_{average}$.

Fig. 7(a) and Fig. 7(b) show that $T_{average}$ varies with different L and M. In Fig. 7(a), $T_{average}$ is almost linear to L when M is larger than 16. For M smaller than 16, $T_{average}$ has a jump, whose range is nearly inverse proportional to M. In Fig. 7(b), when L is small (e.g., $L \leq 200$), $T_{average}$ decreases with the increasing M. However when M reaches some value, $T_{average}$ becomes constant. When L is relative large (e.g., $L \geq 400$), $T_{average}$ is constant.

A mathematic model is adopted to further analyze these phenomena. It should be noted that time complexities of T_{IO} and T_{DTW} are both linear. T_{IO} is linear to L, and T_{DTW} is proportional to M and L. The mathematic model is shown below:

$$T_{average} = \begin{cases} T_{IO} + \frac{T_{DTW}}{M}, & T_{IO} > T_{DTW} \\ \frac{T_{IO}}{M} + T_{DTW}, & T_{IO} \leq T_{DTW}. \end{cases} \tag{10}$$

Fig. 8 shows an illustration of the two situations of the mathematic model. When $T_{IO} > T_{DTW}$, $T_{average} = T_{IO} + T_{DTW}/M$. With the increase of M, $T_{average}$ gradually decreases, and finally approaches to the T_{IO}; when $T_{IO} < T_{DTW}$, $T_{average} = T_{IO}/M + T_{DTW}$. With the increase of M, $T_{average}$ gradually decreases, and finally approaches

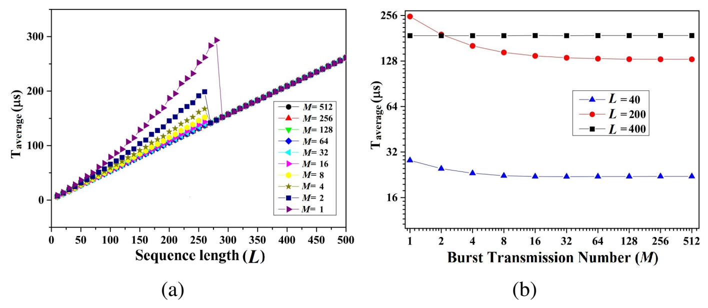


Fig. 7. Characterizations of DTW interface: (a) $T_{average}$ varies with continuous burst transmission number, L, and discrete sequence length, M and (b) $T_{average}$ varies with continuous burst transmission number, L, and discrete sequence length, M.

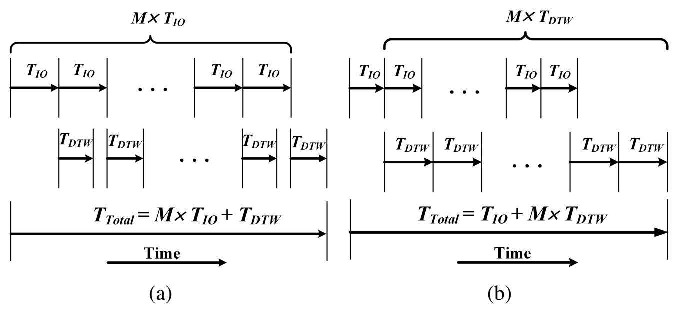


Fig. 8. An illustration of two situations for $T_{average}$: (a) $T_{IO} > T_{DTW}, T_{Total} = M \times T_{IO} + T_{DTW}$; (b) $T_{IO} > T_{DTW}$, $T_{Total} = T_{IO} + M \times T_{DTW}$.

to T_{DTW}. When $T_{IO} << T_{DTW}, T_{IO}/M$ can be ignored, therefore $T_{average} \approx T_{DTW}$.

When M and L are both small, T_{DTW} is analogous to T_{IO}. Thus, T_{IO} has a relatively large influence on $T_{average}$, which results in a jump in Fig. 7(a) and the non-constant part in Fig. 7(b). The linearity in Fig. 7(a) and constant part in Fig. 7(b) are all due to the fact that T_{IO} can be ignored when one or both of M and L are relatively large.

This principle can be also applied to analyse influences of other parameters of DTW interface. It is easy to know that the number of training templates, W, has the same influence with the length of sequence, L. As DTW constraint, R, has no impact on DTW calculation time, it does not affect the characterization of DTW interface.

IV. Experiments

With the proposed DTW acceleration architecture, we conduct a comprehensive experiment with three popular tasks in stream data mining including similarity search, classification and anomaly detection. In the experiment, the proposed architecture is expressed as ARM+FPGA for simplicity.

A. Experiment Overview

The datasets used for similarity search and kNN are obtained from the UCR time series classification/clustering page [21], which serves as a public datasets for data mining and machine learning communities. For anomaly detection, the datasets are from the MIT-BIH Long Term Database [22]

TABLE I. Configurations of the proposed DTW acceleration Architecture

Platform	Component	Parameter
FPGA	PE Number	24
	Data Precision	8 bits
	Adaptive Logic Modules (ALM)	3,469 / 41,910 (8 %)
	Register Number	2922
	Clock Frequency	60MHz
ARM	Clock Frequency	800MHz
	Programming Language	C++
	Compiler Tool	GCC 4-7-3
	Optimization Level	O3
	DTW Constraint	5%/100%

which contains electrocardiograms (ECG) with a length of about 20 hours for each patient.

The proposed DTW acceleration architecture is based on Cyclone V SX SoC [19]. A relatively high clock frequency, 60MHz is achieved on the SoC with the lowest speed grade, C8, of Altera [19]. The configurations are shown in TABLE I.

In the experiments, tasks are implemented on both ARM and ARM+FPGA for comparison. For ARM implementations, all tasks are coded in C++ with double precision. Compared with ARM implementations, implementations on ARM+FPGA move DTW calculations to DTW-FPGA. All software optimizations are adopted from the UCR suite [8], which is the state-of-art software implementation of DTW for similarity search.

For DTW constraint configuration, two conditions, 5% and 100% (or no DTW constraint), are discussed. It should be noted that the best DTW constraint to get the highest accuracy is specific for applications. As the lower bound method proposed by Keogh *et al.* [23] (LB_{Keogh} in short) is specific for DTW with constraints, it is only used with a low DTW constraint of 5% . The lower bound method proposed by Kim *et al.* [7] (LB_{Kim} in short) is adopted for both conditions. As M is involved with tasks and real-time requirement, M is set to one for clear demonstration and comparison. It should be noted that larger M can get higher performance.

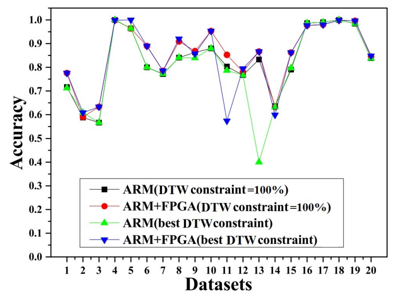
Accuracy, speedup, and energy efficiency are discussed in the experiments. Energy efficiency is associated with runtime and power consumption as shown below:

$$E_{efficiency} = \frac{ItemNumber}{Energy}, \tag{11}$$

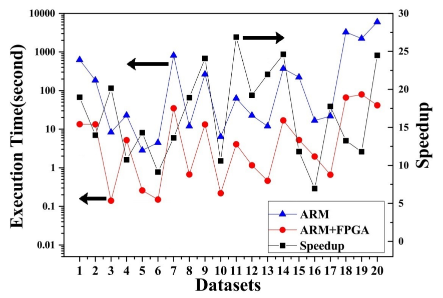
where $ItemNumber$ is the product of the total sequence number and the template number, and $Energy$ is the total consuming energy. A power estimation tool, $PowerPlay$ [24], is used to estimate the power consumption of the platform. The power consumption of the platform is shown in TABLE II. For ARM implementations, only the power of ARM contributes to the total power consumption. While for ARM+FPGA, the total power consumption is the sum of three components, ARM, DTW interface and DTW-FPGA.

TABLE II. Power consumption (mW) of the proposed DTW Accelration Architecture

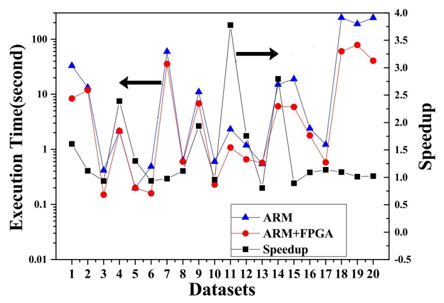
DTW Acceleration Architecture	ARM	DTW Interface	DTW-FPGA
2508.21	1422.07	487.73	598.42



(a) Accuracy

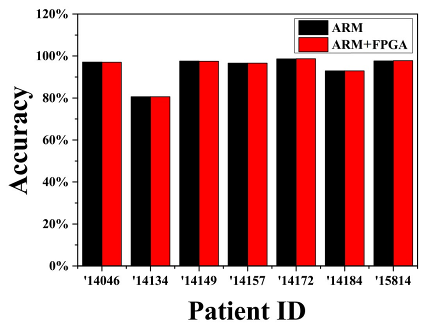


(b) DTW constraint of 100%

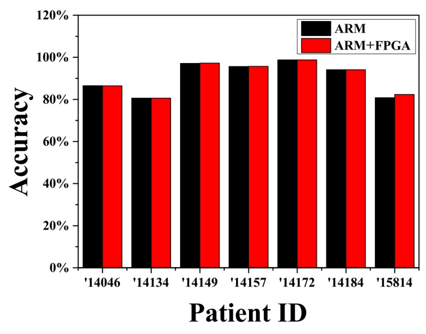


(c) Best DTW constraints

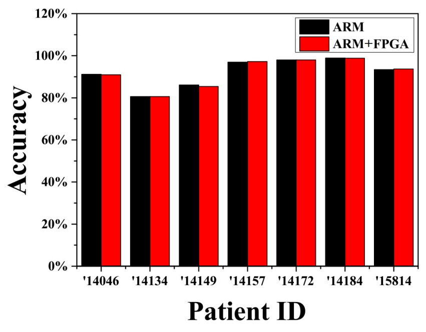
Fig. 9. Results of classification with kNN: (a) accuracy; (b) speedup with DTW constraint of 100%; (c) speedup with best DTW constraints. Datasets: 1-50Words; 2-Adiac; 3-Beef; 4-CBF; 5-Coffee; 6-ECG200; 7-FaceAll; 8-FaceFour; 9-fish; 10-Gun_Point; 11-Lighting2; 12-Lighting7; 13-OliveOil; 14-OSULeaf; 15-SwedishLeaf; 16-synthetic_control; 17-Trace; 18-Two_Patterns; 19-wafer; 20-yoga.



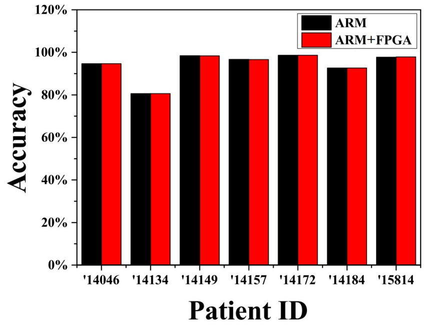
(a) Template number=4, DTW constraint=5%



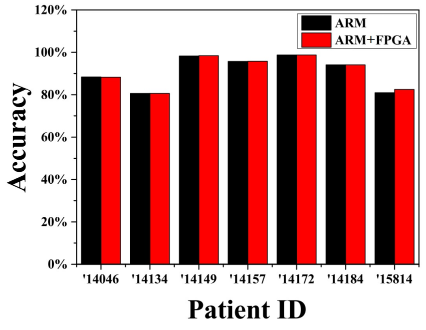
(b) Template number=40, DTW constraint=5%



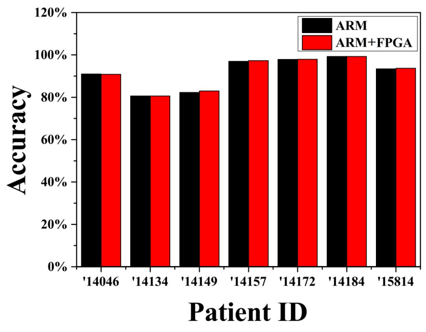
(c) Template number=400, DTW constraint=5%



(d) Template number=4, DTW constraint=100%



(e) Template number=40, DTW constraint=100%



(f) Template number=400, DTW constraint=100%

Fig. 10. Accuracy of anomaly detection with different template number and DTW constraints.

B. Similarity Search

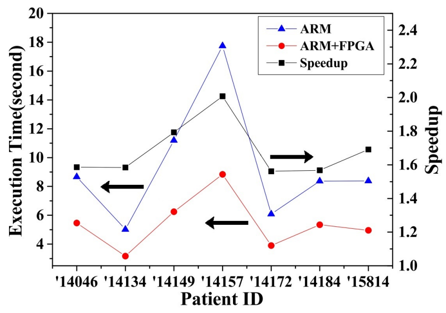
1) Experiment setup: The dataset, $Two\ patterns$ [21], with 512000 points is selected for similarity search. Similarity search is to find the candidate, which has the minimum DTW distance with the query from all the candidates. The query is sent to the DTW interface and stored by DTW-FPGA at the beginning. Then if DTW software optimizations could not prune the current sequence, it will be sent to DTW interface for DTW calculations.

2) Results: TABLE III shows the results of the similarity search task. ARM+FPGA and ARM have the same accuracy as they find the same sequence as the most similar one to the query. For DTW constraint of 100%, the first point of the result is located at the 329463^{th} point, while for DTW constrain of 5%, the first point of the result is located at the 459138^{th} point. Compared with ARM, the speedup of ARM+FPGA are 0.96x and 12.34x for DTW constraints of 5% and 100%, respectively,

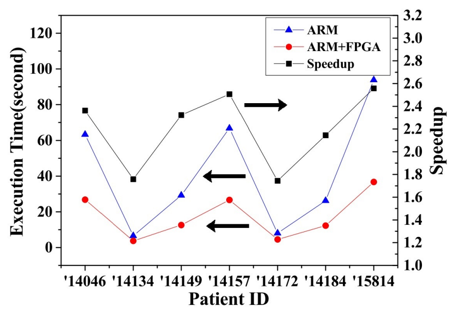
TABLE III. RESULTS OF SIMILARITY SEARCH

Platform	DTW Constraint	Execution Time (second)	Speedup	Energy Efficiency (items/mJ)	Energy Improvement
ARM	5%	6.3	N/A	57.15	N/A
ARM+FPGA	5%	6.6	0.96x	30.93	0.54x
ARM	100%	514.74	N/A	0.70	N/A
ARM+FPGA	100%	41.71	12.34x	4.89	6.98x

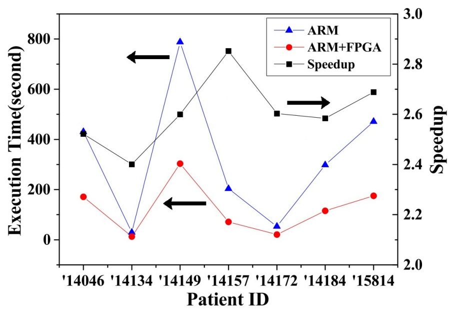
and the energy efficiency improvement are 0.54x and 6.98x for DTW constraints of 5% and 100%, respectively. There is a big difference between speedups with DTW constraints of 5% and 100%. This is due to two reasons. Firstly, the DTW execution time of ARM with DTW constraints of 5% is much less than that with DTW constraints of 100%. However, the DTW execution time of DTW-FPGA is the same for configurations with DTW constraints of 5% and 100%. The second reason



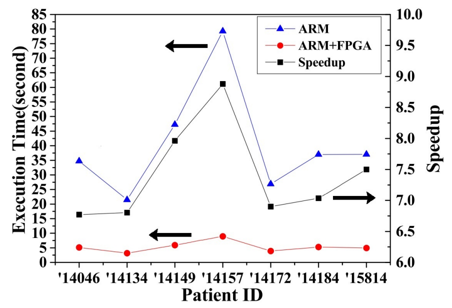
(a) Template number=4, warping constraint=5%



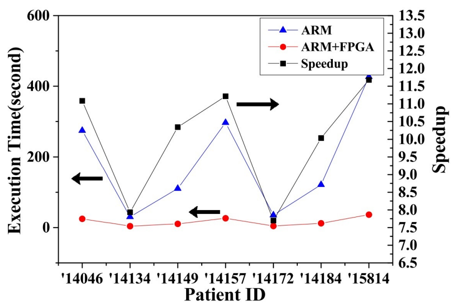
(b) Template number=40, warping constraint=5%



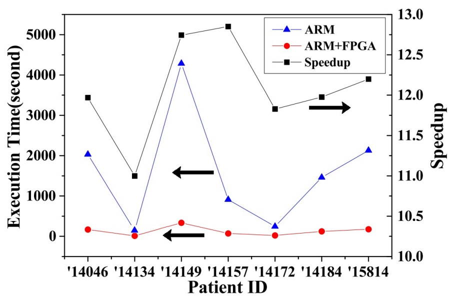
(c) Template number=400, warping constraint=5%



(d) Template number=4, DTW constraint=100%



(e) Template number=40, DTW constraint=100%



(f) Template number=400, DTW constraint=100%

Fig. 11. Execution time and speedup of anomaly detection with different template number and DTW constraints.

is that the lower bound method, LB_{Keogh}, can prune more sequences with lower DTW constraints. In some situations the energy efficiency of ARM+FPGA is lower than that of ARM, even though the speedup for ARM+FPGA is higher. This is due to the fact that the power consumption of ARM+FPGA is higher than ARM.

C. *Classification with k-Nearest Neighbors*

1) Experiment setup: 20 datasets [21] with different sequence lengths are selected for classification, and the optimal DTW constraint for each dataset is also indicated in the dataset. Thus the DTW constraint is assigned according to [21] (ranges from 0 to 12%) rather than 5%. kNN is used for classifications.

2) Results: Fig. 9(a) shows the accuracy varies with different configurations. It can be concluded that ARM+FPGA and ARM have almost the same accuracy with different DTW constraints. The execution time is presented in Fig. 9(b) and Fig. 9(c). It can be learned that ARM+FPGA is faster than ARM, and it can get an average speedup of 16.79x and 1.13x for DTW constraints of the optimal value and 100%, respectively. The average improvements of energy efficiency of ARM+FPGA over ARM are 0.64x and 9.41x for DTW constraints of the optimal value and 100%, respectively. It can be also found that energy efficiency and speedup vary with different datasets. As the phenomenon of speedup and energy efficiency are almost the same with the previous subsection, further discussions are ignored here.

D. *Anomaly Detection*

1) Experiment setup: The long ECG data [22] is divided to sequences with a length of 96 on ARM for segmentation. The peak points are from the $annotations.txt$ from the database. With the peak point a_i, we get the segmented sequence as $a_{i-31}, ..., a_i, ...a_{i+64}$. The first dimension of the two-dimension data is selected as representative. A training processing is needed to obtain $Threshold$. The first W normal sequences are stored as templates. The following 100 normal sequences are used for training $Threshold$. The rest of the ECG sequences are treated as the data to be detected.

2) Results: As shown in Fig. 10, accuracies of ARM and ARM+FPGA are almost the same with the same parameters. Fig. 11 shows the comparison of ARM and ARM+FPGA with different configurations. The speedup varies from 1.70x to 12.20x with different datasets, template number and DTW constraints. As the template number increases, the speedup also increases. This is due to the fact that templates are all normal sequences and similar to each other. Thus, the pruning power of software optimizations weaken. The energy efficiency improvements of ARM+FPGA over ARM are 0.96x-6.83x. The speedup and energy efficiency with DTW constraints of 5% and 100% have the same discipline discussed in IV-B.

E. *Further Discussion*

Accuracies of ARM and ARM+FPGA with different configurations are almost the same with the three widely-used tasks, similarity search, classification, and anomaly detection. Thus, the proposed DTW acceleration architecture can effectively process tasks almost without accuracy loss.

The speedup and energy efficiency vary for different tasks. The average improvements are 7.52x and 4.23x for the three tasks, in which the similarity search has the lowest speedup. This is due to the fact that once it gets a low DTW value, it can be used in the rest of the calculation so that a lot of sequences can be pruned. However, the processing of classification and anomaly detection is independent for each test data.

In the experiments, only two conditions of DTW constraints, DTW constraints of 5% and 100% (or no DTW constraint) are considered as examples, which are appropriate to reveal its features due to the following reasons. The pruning power of LB_{Keogh} has a firm relationship with DTW constraints: a lower DTW constraint corresponds to a tight lower bound, which leads to a great pruning power. Therefore, the speedup and energy efficient improvement both monotonously increase with DTW constraints, e.g., the speedup with a DTW constraint of 20% in the classification experiment should have a value between 1.13x and 16.79x.

There exists some situations that the energy efficiency improvement is below one, which means the proposed architecture is less energy efficient than ARM. This can be improved with two approaches. The first method is to make the DTW interface and DTW-FPGA sleep for a period of time when it is idle. The second method is realized in a more experiential way, which employs ARM only for tasks that has a low energy efficiency. This is extremely suitable for tasks with great pruning power by software optimizations.

An comparison with related work is shown in TABLE IV. The work [14] has achieved an energy efficiency improvement of 1.54x with instruction extensions for frequently-used operations. In order to realize instruction extensions, the ARM softcore is selected with a relatively low clock frequency of 100MHz. In the proposed DTW acceleration architecture, pipelined DTW implementation on FPGAs can better speedup DTW calculations. The hardcore ARM is more energy-efficient for software optimizations of DTW. Thus, we achieve a much higher energy efficiency improvement of 4.23x. It should be noted that the speedup achieved by replacing a software floating-point library in C with a a double-precision floating-point unit (FPU) in [14] is ignored here. It should also be highlighted that only similarity search is tested in [14], while our work has tested three tasks including similarity search, classification, and anomaly detection. If only similarity search is considered, the speedup and energy efficiency improvement of the proposed DTW acceleration architecture is 6.65x and 3.76x, respectively.

TABLE IV. COMPARISON WITH RELATED WORK

Approach	ARM Core (Clock Frequency)	Acceleration Method	Speedup	Energy Efficiency Improvement
[14]	Softcore (100 MHz)	Instruction extension	1.42x	1.54x
This work	Hardcore (800 MHz)	Parallel DTW	7.52x	4.23x

V. CONCLUSION

In this paper we have proposed an DTW acceleration architecture based on ARM and FPGA for stream data mining on sensor-based embedded devices. Software optimizations for DTW are implemented on ARM, and pipelined DTW is implemented on FPGAs for further accelerations. The tasks on ARM with software optimizations can easily invoke the DTW interface as a sub-function. The sequence length, the number of templates and the DTW constraint are all supported by the DTW interface. Three popular tasks in stream data mining, similarity search, classification, and anomaly detection are implemented in the experiments. Compared with ARM implementations, the average speedup and improvement of energy efficiency of the proposed DTW acceleration architecture are 7.52x and 4.23x, respectively.

REFERENCES

[1] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1, 2011.

[2] Toshiki Iwata, Hisao Ishizuka, Masao Watari, Toshiaki Hoshi, Yuichi Kawakami, and Masatoshi Mizuno. A speech recognition processor. In *Solid-State Circuits Conference*, volume 26, pages 120–121. IEEE, 1983.

[3] Reza Lotfian and Roozbeh Jafari. An ultra-low power hardware accelerator architecture for wearable computers using dynamic time warping. In *DAEE*, pages 913–916. EDA Consortium, 2013.

[4] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *VLDB Endowment*, 1(2):1542–1552, 2008.

[5] Graham Cormode. Fundamentals of analyzing and mining data streams. In *Tutorial at Workshop on Data Stream Analysis, Caserta, Italy*, 2007.

[6] Ada Wai-Chee Fu, Eamonn Keogh, Leo Yung Lau, Chotirat Ann Ratanamahatana, and Raymond Chi-Wing Wong. Scaling and time warping in time series querying. *The VLDB Journal*, 17(4):899–921, 2008.

[7] Sang-Wook Kim, Sanghyun Park, and Wesley W Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *17th ICDE*, pages 607–614. IEEE, 2001.

[8] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *ACM SIGKDD*, pages 262–270. ACM, 2012.

[9] Yasushi Sakurai, Christos Faloutsos, and Masashi Yamamuro. Stream monitoring under the time warping distance. In *ICDE*, pages 1046–1055. IEEE, 2007.

[10] Yaodong Zhang, Kiarash Adl, and James Glass. Fast spoken query detection using lower-bound dynamic time warping on graphical processing units. In $ICASSP$, pages 5173–5176. IEEE, 2012.

[11] 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 *ACM/SIGDA* $FPGA$, pages 53–62. ACM, 2013.

[12] Doruk Sart, Abdullah Mueen, Walid Najjar, Eamonn Keogh, and Vit Niennattrakul. Accelerating dynamic time warping subsequence search with gpus and fpgas. In *ICDM*, pages 1001–1006. IEEE, 2010.

[13] Christian Hundt, Bertil Schmidt, and Elmar Schomer. Cuda-accelerated alignment of subsequences in streamed time series data. In *ICPP*, pages 10–19. IEEE, 2014.

[14] Joseph Tarango, Eamonn Keogh, and Philip Brisk. Instruction set extensions for dynamic time warping. In *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, page 18. IEEE Press, 2013.

[15] Toni M Rath and Raghavan Manmatha. Word image matching using dynamic time warping. In *CVPR*, volume 2, pages II–521. IEEE, 2003.

[16] Hiroaki Sakoe, Seibi Chiba, A Waibel, and KF Lee. Dynamic programming algorithm optimization for spoken word recognition. *Readings in speech recognition*, 159, 1990.

[17] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[18] Eamonn Keogh, Jessica Lin, Ada Waichee Fu, and Helga Van Herle. Finding unusual medical time-series subsequences: Algorithms and applications. *Information Technology in Biomedicine, IEEE Transactions on*, 10(3):429–439, 2006.

[19] www.terasic.com.tw.

[20] http://rocketboards.org/foswiki/view/documentation/gsrd.

[21] http://www.cs.ucr.edu/~eamonn/time_series_data/.

[22] http://www.physionet.org/physiobank/database/ltdb/.

[23] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[24] http://www.altera.com.cn/support/software/power/sof-qts-power.html.