IEICE TRANSACTIONS

on Fundamentals of Electronics, Communications and Computer Sciences

VOL. E101-A NO. 1
JANUARY 2018

A PUBLICATION OF THE ENGINEERING SCIENCES SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

Scalable and Parameterized Architecture for Efficient Stream Mining

Li ZHANG[†a)], ***Nonmember***, **Dawei LI**[†], ***Member***, **Xuecheng ZOU**[†], **Yu HU**[†], ***and*** **Xiaowei XU**[†], ***Nonmembers***

SUMMARY With an annual growth of billions of sensor-based devices, it is an urgent need to do stream mining for the massive data streams produced by these devices. Cloud computing is a competitive choice for this, with powerful computational capabilities. However, it sacrifices real-time feature and energy efficiency. Application-specific integrated circuit (ASIC) is with high performance and efficiency, which is not cost-effective for diverse applications. The general-purpose microcontroller is of low performance. Therefore, it is a challenge to do stream mining on these low-cost devices with scalability and efficiency. In this paper, we introduce an FPGA-based scalable and parameterized architecture for stream mining.Particularly, Dynamic Time Warping (DTW) based k-Nearest Neighbor (kNN) is adopted in the architecture. Two processing element (PE) rings for DTW and kNN are designed to achieve parameterization and scalability with high performance. We implement the proposed architecture on an FPGA and perform a comprehensive performance evaluation. The experimental results indicate thatcompared to the multi-core CPU-based implementation, our approach demonstrates over one order of magnitude on speedup and three orders of magnitude on energy-efficiency.

key words: *stream mining architecture, parameterization, dynamic time warping, k-nearest neighbor*

1. Introduction

In the era of Internet of Things (IoT), stream mining (also known as stream data mining) based on sensor-based devices has received increasing attention. The Cisco Internet Business Solutions Group (IBSG) predicts that there will be 50 billion devices connecting to the Internet by 2020 [1]. The great mass of these devices are sensor-based devices such as smart phones, which forms the so-called "Internet of Things". Usually these sensor-based devices have up to tens of sensors. Thus, the huge number of sensor-based devices produce massive quantities and multiple categories of data streams. Since the widely adopted wireless communication is energy-expensive to upload all the stream data to the cloud for further processing, cloud computing is not practical to process these huge data streams. Furthermore, cloud computing cannot support real-time processing, which is critical in many time-sensitive applications. **Thus stream mining on sensor-based devices locally is demanded**. Considering the costly ASIC for small volume product and the low-performance microcontroller, high-performance and energy-efficient FPGAs are promising in handling stream mining [2]–[5].

Manuscript received June 23, 2017.
Manuscript revised September 10, 2017.

[†]The authors are with the School of Optical and Electronic Information, Huazhong University of Science and Technology, China.
a) E-mail: andyzhang_ic@163.com

DOI: 10.1587/transfun.E101.A.219

Some embedded applications, such as portable device and smart wearable device, are required to process various forms of data. But their hardware architecture is fixed. Therefore, scalable, parameterized and efficient stream mining architecture with high-performance is preferred to process massive and different kinds of data streams on sensor-based devices.

Among stream mining, Dynamic Time Warping (DTW) based k-Nearest Neighbor (kNN) is popular. DTW is the best measure in most domains [6], and k-Nearest Neighbor (kNN) is a widely used classifier. DTW-kNN has been adopted for speech recognition [7], [8]. Kia et al. [9] applied DTW-kNN to financial market prediction. Chaovalitwongse et al. [10] analysed abnormal brain activity with DTW-kNN. Gene expression profiles analysis used DTW-kNN for missing value imputation with microarray time series data [11]. Xi et al. [12] compared DTW-kNN with implementations in ten papers, which showed that DTW-kNN gets a much higher accuracy compared with decision tree, first order logic rules, neural network, hidden Markov Models, etc..

Recently, there have been a few works of kNN acceleration on FPGAs and GPUs. Wavelet transform based kNN (Wavelet-kNN) has been implemented on FPGAs in [2] for image processing. The SIMD-style architecture has been adopted to implement a conceptual design of Euclidean distance based kNN (Euclidean-kNN) in [13]. Linear array designs [14] for Euclidean-kNN were proposed. GPUs have been used to accelerate kNN in [15]. However, all the above implementation lacks parameterization for adaptation to handle different configurations for a variety of different streams in multiple applications. A recent study [16] adopted dynamic partial reconfiguration to achieve parameterization for Euclidean-kNN. However, the cost of this implementation is high because too much FPGA area would be reserved within the reconfigurable partition for the largest possible configuration. Thus this implementation can only support limited number of K in kNN.

Meanwhile, recent DTW accelerations with hardware mainly focus on speedup. DTW subsequence search with FPGAs in [3] lacks scalability and parameterization. The subsequence search application of DTW has been further accelerated with SPRING algorithm [17] in [4]. This implementation have high scalability by adopting SPRING algorithm however without data normalization, which may allow false dismissal [3]. GPU accelerations for DTW subsequence search [3] [18] are with high power, which is not suitable for energy-sensitive sensor-based devices. There

are a few work that takes energy efficiency into consideration. In order to locally computing DTW, an ultra-low power hardware accelerator for DTW is proposed [19]. It is specific for wearable applications with a very low frequency of up to 5000 Hz. Instruction set extensions for parameterized DTW subsequence search was proposed in [5], which didn't utilize the parallelism of DTW. Lotfian et al. [20] proposed an ultra low-power DTW implementation on ASIC, which is for wearable computation with low performance. Thus almost all the existing work of DTW acceleration is specific for applications, e.g., subsequence search. Parameterization has not been taken into consideration to process multiple streams for multiple applications, which is critical in stream mining architecture.

In this paper, we propose a scalable, parameterized and efficient FPGA-based architecture for stream mining on sensor-based devices. What we emphasis is parameterization for processing different categories of streams, and we achieve scalability and efficiency at the same time with a relatively high performance. Particularly, k-Nearest Neighbor (kNN), a widely used classifier, and Dynamic Time Warping (DTW), a popular distance measure [4], are adopted in the architecture. Specific data representation and precision reduction techniques are also integrated in the architecture. In this work, our contributions are summarized as follows:

- We develop a scalable, parameterized and efficient FPGA-based architecture with high-performance for stream mining on sensor-based devices.
- Particularly, DTW and kNN are adopted in the architecture. We design two processing element (PE) structures and control modules for DTW and kNN to provide scalability and parameterization with multiple parameters. Specific data representation and precision techniques are used to reduce resources while guaranteeing accuracy. Thus, the proposed architecture can handel different kind of streams with different parameters for multiple applications.
- We implement and evaluate our architecture with an anomaly detection application. A comparison with multi-core CPU is presented with a public available dataset using multiple configurations. The results indicate that improvements of runtime and energy efficiency of the propose architecture over multi-core CPU are up to 14x and almost 2,000x, respectively.

2. Background and Preliminaries

2.1 Dynamic Time Warping

Dynamic Time Warping (DTW) is a widely-used distance measure of stream similarity [6]. There are two time sequences in Fig. 1(a), a sequence C of length n, as a candidate, and a sequence T of length m, as a training template as shown in Eq. (1).

$$C = c_1, \cdots, c_i, \cdots, c_n, T = t_1, \cdots, t_i, \cdots, t_m. \quad (1)$$

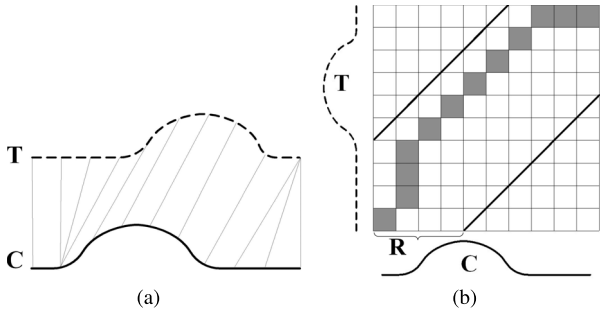


Fig. 1 (a) DTW matching: the lines indicates the matching relationship between T and C; (b) DTW warping path with Sakoe-Chiba band: the gray squares form the warping path between T and C on the distance matrix.

Time series subsequences must be normalized to make a meaningful comparison [6]. Z-Normalization is adopted to remove offset and amplitudes as shown in Equation (2). For no false dismissal, the SPRING algorithm [3] is not involved for Z-Normalization.

$$\mu_T = \frac{1}{m}\sum_{k=1}^{m} t_k, \sigma_T^2 = \frac{1}{m}\sum_{k=1}^{m} t_k^2 - \mu_T^2, t_k^{'} = \frac{t_k - \mu_T}{\sigma_T}. \quad (2)$$

The representation method, Piecewise Aggregate Approximation (PAA) [21] is used to reduce data dimensionality as shown in Eq. (3). And P varies for different requirements. PAA is an effective approximation compared with more sophisticated dimensionality reduction techniques such as fourier transforms and wavelets [21].

$$t_k^{''} = \frac{1}{P}\sum_{i=P(k-1)+1}^{Pk} t_k^{'} = \frac{\frac{1}{P}\sum_{i=P(k-1)+1}^{Pk} t_k - \mu_T}{\sigma_T}. \quad (3)$$

To measure the similarity of C and T, DTW creates an n-by-m matrix, MAR. The value of the (i^{th}, j^{th}) element in MAR represents the distance $d(c_i, t_j)$ between points c_i and t_j as shown in Eq. (4).

$$MAR(i, j) = d(c_i^{''}, t_j^{''}). \quad (4)$$

There are many effective distance metrics such as absolute value distance distance, squared distance, etc. For $d(c_i, t_j)$, the widely used distance [6], squared difference, is used for individual elements $c_i^{''}$ and $t_j^{''}$ as shown below:

$$d(c_i^{''}, t_j^{''}) = (c_i^{''} - t_j^{''})^2. \quad (5)$$

With the distance matrix MAR, the warping path can be derived. There are three well-known constrains for 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 MAR 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 as shown in Eq. (6), using [6].

$$CD(i,j) = MAR(i,j) + min\begin{cases} CD(i,j-1) \\ CD(i-1,j) \\ CD(i-1,j-1) \end{cases}, \quad (6)$$
$$CD(0,0) = 0, CD(0,j) = CD(i,0) = \infty,$$
$$1 \le i \le n; 1 \le j \le m$$

where $CD(i,j)$ is the current minimum cumulative distance for $MAR(i,j)$. This procedure is called *warping path calculation* as shown in Fig. 1(b). The Sakoe-Chiba band [6] is used to constrain the DTW warping path, and the warping path constraint R is shown in Fig. 1(b). The minimized cumulative distance is found. And the final DTW distance is calculated as below:

$$dtw = \sqrt{CD(n,m)}. \quad (7)$$

Usually the combination of *distance matrix calculation* and *warping path calculation* are regarded as *DTW matrix calculation*.

2.2 Multidimensional DTW

There are two multidimensional data streams C_N and T_N:

$$C_N = (c_{11}, \cdots, c_{r1}, \cdots, c_{N1}), \cdots (c_{1i}, \cdots, c_{ri}, \cdots, c_{Ni}), \cdots, (c_{1n}, \cdots, c_{rn}, \cdots, c_{Nn}). \quad (8)$$

$$T_N = (t_{11}, \cdots, t_{r1}, \cdots, t_{N1}), \cdots (t_{1i}, \cdots, t_{ri}, \cdots, t_{Ni}), \cdots, (t_{1n}, \cdots, t_{rn}, \cdots, t_{Nn}). \quad (9)$$

where c_{ri} and t_{ri} are the ith items of the rth dimension of C_N and T_N, respectively. N is the data dimensionality. A multidimensional DTW method [22] is applied. Compared with one-dimensional DTW, Eq. (4) for the calculation of distance matrix is updated as follows:

$$MAR(i,j) = \frac{1}{N}\sum_{k=1}^{N} d(c''_{ki}, t''_{kj}). \quad (10)$$

where c''_{ki} and t''_{kj} are the ith and jth items in the kth dimension of C_N and T_N.

2.3 k-Nearest Neighbor

The k-Nearest Neighbors(kNN) algorithm is one of the well-investigated methods for pattern classifications, which is used to determine the class of an unclassified point by the class of the K nearest points of it [2]. The unclassified point is classified by a majority vote of K neighbors. K varies for different applications.

3. The Proposed Stream Mining Architecture

3.1 Architecture Overview

The top-level block diagram of the proposed stream mining architecture is shown in Fig. 2. Data streams will first be processed with parallel segmentation, which divides the streams into short subsequences. The task assignment will send subsequences to their corresponding parallel lines, and it will configure the parameters in each parallel line. In each parallel line, preprocessing, representation, distance metric and task processing are presented. Preprocessing is to remove offsets and scaling of short sequences, which may not be involved according to specific applications. Representation is used to reduce data dimensionality for speedup. Thus, usually there is only one pipelined representation module, which corresponds to multiple parallel preprocessing and distance metric modules. Distance metric evaluates the similarity between sequences, which is usually a very time-comsuming module. Therefore, multiple parallel distance metric modules are implemented for speedup. As the complexity of tasks varies, the number of parallel task modules is determined by tasks. The combination of parameters for each parallel line is different, and then the stream mining architecture can assign streams to appropriate parallel line for efficient processing. Thus, parameterization is critical for stream mining architecture.

Particularly, DTW and kNN are employed in the architecture for anomaly detection task. The DTW-kNN algorithm for anomaly detection is presented in Algorithm 1. Usually lower bound methods [23] are employed in DTW-kNN-anomaly algorithms for speedup in software implementations. However, we do not take it into consideration in the proposed architecture. The main reason is that there is no much speedup with lower bound methods considering the used resources, which is discussed in detail in Sect. 3.5.1. As segmentation is designed for specific applications, it is not involved in this general DTW-kNN-Anomaly algorithm. In Algorithm 1, $test$ is processed with normalization and representation. Then n DTW distances are calculated between $test$ and $T[n]$, which are stored and sorted in $queueDTWdistance$. $anomalyScore$ is the sum of the first Kth smallest DTW distances. Finally, if the $anomalyScore$ is larger than the predefined $Threshold$, return $anomaly$, otherwise return $normal$.

As segmentation is designed for specific applications, it is not involved in this general architecture. We enhance the architecture design from the following four aspects: a) the representation method PAA is integrated in normalization module to reduce processing time; b) two Preprocessing modules are designed for 2-dimensional data streams; c) multiple DTW modules are implemented to achieve parallelism for training templates; d) as 8 bits integers for representing data in FPGAs makes no significant difference [3], precision reduction is made in PAA module to reduce the data accuracy from 16 bits to 8 bits (The main concern to set to 8 bits is the resource constraint of the selected FPGA device. The data length can be easily extended with resource-rich devices. As multipliers allow 18-bits data as input, the clock frequency will not decrease if the data accuracy doesn't exceed 18 bits.). In the remaining part of this section, we will discuss the enhancements in detail. The major notations and

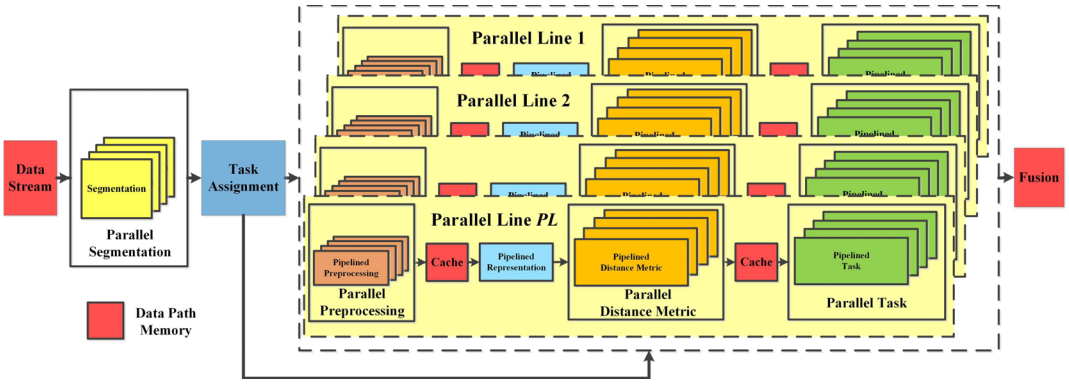


Fig. 2 Scalable and parameterized stream mining architecture.

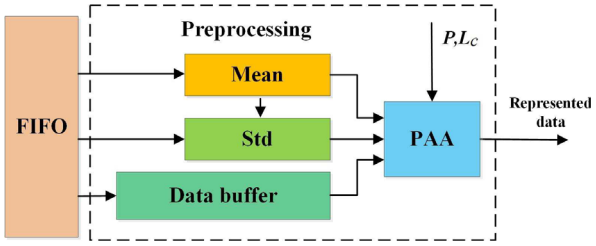


Fig. 3 Preprocessing and representation module structure.

parameters used in this work are summarised in Table 1.

Algorithm 1: DTW-kNN-Anomaly

Input: n template sequences, $T[n]$, test sequences, $test$.
Output: $anomalyState$

```
tn=Normalization(test);
tr=Representation(tn);
for i = 0 → n − 1 do
    queueDTWdistance.push(DTWdistance(tr,T[i]));
end
sortMinPriorityQueue(queueDTWdistance);
for j = 0 → k − 1 do
    anomalyScore+= queueDTWdistance.popFirst();
end
if anomalyScore > threshold then
    anomalyState = anomaly;
else
    anomalyState = normal;
end
```

3.2 Preprocessing and Representation

As displayed in Fig. 3, P tuples are pumped from the FIFO into the preprocessing module in every cycle. The $Mean$ submodule calculates the mean of subsequences with configured length L_c by PAA submodule. Std submodule calculates the standard deviation. The $Data\ buffer$ caches the coming tuples. PAA submodule reads P tuples from the $Data\ buffer$, and calculate normalization and PAA as shown in the expanding equation in Eq. (3).

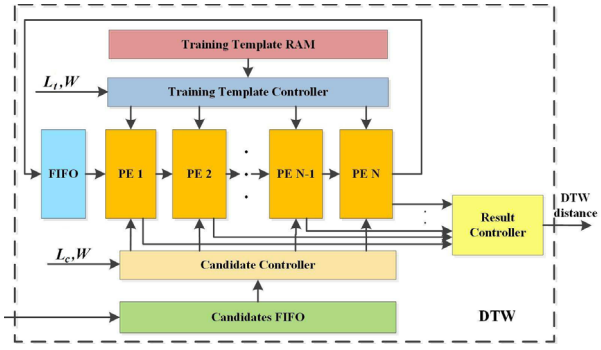


Fig. 4 DTW module design structure.

3.3 Scalable and Parameterized DTW

In this section, we adopt the DTW design from Wang et al. [4], which is one of the fastest recent DTW implementations. The DTW design from [4] adopted the SPRING algorithm [17], and can support arbitrary length of sequence length. As the data is not normalized in the SPRING algorithm, it is not adopted in our work for avoiding false dismissal. We still achieve high efficient pipelines between normalized sequences, which is discussed below in detail with examples. What's more, we further extend it with parameterization for several parameters.

As shown in Fig. 4, N PEs and one FIFO are linked with each other like a ring. The $Training\ Template\ RAM$ stores training templates, while the candidate controller determines that when to send the candidate subsequences to PE and sends to which PE. The function of PE is to calculate one column of the DTW matrix. With the principle of DTW, left, left-bottom and bottom DTW matrix cells are required for the processing of PEs, which can be easily achieved with the connection between PEs. When the candidate length is larger than the number of PE, the FIFO besides the PE 1 is used to store temporary results of PE N to support large candidate length, or to store boundary conditions for PE 1. The result controller is responsible to select the final result to the output port.

Figure 5 shows the work flow of the DTW module. The

Table 1 A summarization of notations and parameters used in this paper

Parameters and Notations		Descriptions
Architecture Scalable Features	PL	Number of Parallel lines
	N_P	The number of parallel preprocessing module
	N_T	The number of parallel task module
	N	The number of PEs in DTW module
	N_D	The number of parallel DTW modules
	M	The number of PEs in kNN module of the FPGA architecture
Architecture Parameters	P	The number of tuples used to calculate one tuple in PAA
	W	The number of training templates in DTW module
	R	Warping path constraint with Sakoe-Chiba band
	L_c	Represented sequence length of candidate subsequences
	L_t	Represented sequence length of template subsequences
	K	K nearest neighbors in kNN
Notations	C	Candidate subsequences $C = c_1, \cdots, c_n$
	T	Training template subsequences $T = t_1, \cdots, t_m$
	$CEIL(x)$	A function returns the minimum integer that is not smaller than x
	L	When L is used, an assumption of L=L_c=L_t is made

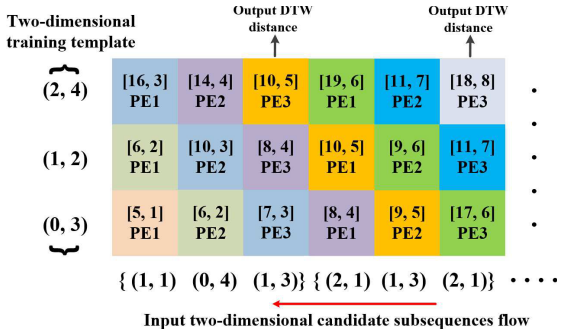


Fig. 5 Illustration of DTW workflow: column 1-3 and 4-6 are the DTW matrixes between query Q and candidates C1 and C2, respectively. (a, b) is a two dimensional tuple. In [c,d], c is the accumulated DTW distance and d is the cycle time. The item below [c, d] indicates which PE processes the DTW matrix cell. DTW matrix cells with the same color are calculated at the same cycle.

number of PEs is three, and the represented subsequence length of candidates and queries are also three. The dimension of the data streams is two. The training template is Q=(0,3), (1,2), (2,4), and there are 2 candidate subsequences: $C1$=(1,1), (0,4), (1,3) and $C2$=(2,1), (1,3), (2,1). In the first cycle, the first tuple of $C1$ (1, 1) is sent to PE 1 and the [1,1] cell of the DTW matrix between $C1$ and Q is calculated. In the second cycle, the second tuple (0, 4) is sent to PE 2, and cell [1,2] and [2,1] are calculated at the same time. In the $3th$ cycle, the third tuple (1, 3) is sent to PE 3, and cell [1, 3], [2, 2] and [3, 1] are calculated. In the $4th$ cycle, PE 1 has finished the calculation of the first column, and the first tuple of $C2$ (2, 1) is sent to PE 1. In the $5th$ cycle, the calculation of DTW matrix between $C1$ and Q is completed. The $ResultController$ sends the result from PE 3 to the output port. It indicates that pumping one DTW distance needs L (L=L_c=L_t=3) cycles.

The proposed DTW framework supports scalability and parameterization. The PE ring structure is featured with scalability. The PEs in rings have the same structure and the connections are almost the same. It is pretty easy to add PEs into the ring when resources are abundant.

The subsequence length of candidates is parameterized. If the subsequence length L (suppose L=L_c=L_t) is larger than the number of PEs, the FIFO in the ring is activated. As shown in Fig. 4, the first N columns are processed, and the results of the Nth column are stored in the FIFO. With the FIFO, the next N columns are then calculated. This process iterates until the subsequence reach its end. There is no structure change of the PE ring but the performance decreases to pump one DTW distance in every $(CEIL(L/N)) * L$ cycles. Other conditions with $L_c \neq L_t$ share the same principle.

The number of training templates can also be variable. Suppose W is larger than the number of DTW modules. When the calculation of one DTW distance is completed, the $Training\,Template\,Controller$ can pump a new training template to the PEs. The candidate subsequence has to be resent to the PEs to calculate the DTW distance with the new training template. This iteration ends when all training templates have been processed with the candidate. Suppose the represented sequence length is equal to the number of PEs and there are W training sequences, then the cycles that needed to pump one candidate increases from N to $N * CEIL(W/N_D)$.

The warping path constraint R is parameterized. Suppose a PE is processing the ith element of one candidate and the jth element of one training template. If $j - R \leq i \leq j + R$, the accumulated DTW distance is calculated as discussed in Fig. 5, otherwise the accumulated DTW distance is the maximum 32-bit integer. Thus R has no influence on the throughput.

3.4 Scalable and Parameterized kNN

Observing that the DTW module produces one DTW distance in a relatively large number of cycles, we propose a scalable kNN module. As displayed in Fig. 6, M PEs and one FIFO are linked with each other like a ring. The kNN controller receives DTW distances and sends them to PEs and the FIFO. Multiplexers are used to configure the structure of the architecture. $LableFinder$ is responsible to process the K minimum DTW distances to find the right label and

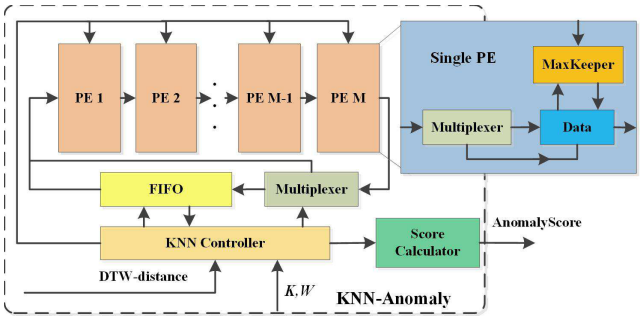


Fig. 6 kNN design structure.

calculate their sum. In the initial stage, according to the K in kNN, multiplexers are configured by the kNN controller so that the number of items that can be stored in PEs and in the FIFO is K. When DTW distances arrive, kNN controller first sends the first K distances to the ring. After the ring stores K items, the kNN controller sends DTW distances to PEs. Each DTW distance stays in the PE for K cycles. During each cycle, the DTW distance stored in the $MaxKeeper$ is compared with the data in the PE. If the DTW distance is smaller than the data, they exchange their value. After K cycles, the DTW distance stored in the $MaxKeeper$ is dropped as it is larger than all the K DTW distances in the ring. The iteration continues until DTW distances between all queries and the candidate have been processed. Then the results are sent to $LableFinder$. $LableFinder$ counts the number of class labels of the K nearest DTW distances. The maximum label counter $MLCount$ and its label address $MLAddr$ are updated at the same time. The currently incremented label counter in $LableRAM$ that is larger than $MLCount$ will be stored in the $MLCount$, and the $MLAddr$ will store the label of the incremented label counter. The $ScoreSum$ calculates the sum of the K nearest DTW distances, which can be used for anomaly detection. $MLAddr$ and $ScoreSum$ are sent to the output port.

The K in kNN is parameterized and scalable. The number of PEs in kNN module is usually equal to the number of DTW modules. Therefore the whole architecture can pipeline very efficiently. DD DTW distances are pumped into kNN module in every L cycles. Therefore, if K is not larger than L, all the items in the ring of kNN can run one circle in less than L cycles, which will not block the pipeline. If K is larger then L, the performance will degrade.

3.5 Further Discussion

3.5.1 Why Lower Bound Is Not Involved

Lower bound (LB) method is to estimate the lower bound of a computing-extensive distance metric such as DTW with a light-weight calculation. It is introduced to index time series, which is a very efficient way to speed up classification in software design. Many effective lower bound methods [23] are proved efficient under the DTW distance. A typical DTW-kNN algorithm involved with lower bound is shown in

Algorithm 2: LB-DTW-kNN-Anomaly

Input: n template sequences, $T[n]$, test sequences, $test$.
Output: $anomalyState$

```
1  tn=Normalization(test);
2  tr=Representation(tn);
3  repeat
4      queueLB.push(T[i], LBdistance(tr,T[i]));
5      index++;
6  until index == n;
7  repeat
8      first=queueLB.first();
9      if queueResult.length()< K then
10         queueResult.push(first.firstItem,
             DTWdistance(first,tr));
11     end
12     if DTWdistance(first.firstItem,tr)<
         queueResult.last().secondItem then
13         queueResult.popLast();
14         queueResult.push(first.firstItem,
             DTWdistance(first.firstItem,tr));
15     end
16 until queueLB is empty or
     first.secondItem >= queueResult.last().secondItem;
17 anomalyScore= Sum(queueResult);
18 if anomalyScore > threshold then
19     anomalyState=anomaly;
20 else
21     anomalyState=normal;
22 end
```

Algorithm 2. $queueLB$ and $queueResult$ are both min-first priority queues $queueLB$ is to store all the template sequences and their corresponding lower bound distances with $test$, which is sorted by lower bounds. While $queueResult$ stores all the template sequences and their corresponding DTW distances with $test$, which is sorted by DTW distances. The magic of lower bound is that it can eliminate a lot of DTW calculation corresponding to line 16 in Algorithm 2, which then can speedup the whole architecture.

However it is not the same case for FPGA implementations. **It should be noted that the discussion is specific for general DTW-kNN, which has a big difference with subsequence search applications**. Firstly, in LB-involved implementation LB processing is a block operation, which cannot provide much speedup without taking advantage of the FPGA features. Suppose the number of parallel lower bound modules is E, and the number of DTW modules is N_D. There is only one sort module. All the modules are pipelined. For LB-involved implementation, the sort module has to wait until all LBs between the candidate and W training templates have been handled. Suppose L=L_c=L_t, which is approximate as usually there is no much difference between the length of candidate and template sequences. The processing time of LB modules for one candidate is about $L * CEIL(W/E)$ cycles. The processing time T_{sort} of the sort module varies for different implementations. The time needed to calculate DTW distances is $L * CEIL(\beta W/N_D)$ cycles, which means $(1-\beta) * W$ DTWs are eliminated. So the maximum throughput T_{LB} is determined by the most time-consuming module as shown below:

$$T_{LB} = \frac{L}{max(L * CEIL(\frac{W}{E}), T_{sort}, L * CEIL(\frac{\beta W}{N_D})}. \quad (11)$$

For no-LB implementation, there is no block operations and the maximum throughput T_{NoLB} is shown below:

$$T_{NoLB} = \frac{L}{L * CEIL(\frac{W}{N_D})}. \quad (12)$$

For conditions with small W where $1 \leq W \leq N_D$,the throughput is the same. For large W, we adopt the LB method proposed by Keogh et al. [23] for example, which is one of the tightest LB methods. The pruning rate is about 0.75 averagely in [23] ,therefore the minimum β is 0.25 (This value is for general DTW-kNN process, and it will be much smaller for subsequence search application.). So the maximum throughput of LB-involved implementation is shown below:

$$T_{LB\_Max} = \frac{L}{L * CEIL(\frac{0.25W}{N_D})}. \quad (13)$$

The constraint of E is:

$$\frac{W}{E} \leq \frac{0.25W}{N_D} \rightarrow E \geq 4 \times N_D. \quad (14)$$

So the speedup of T_{LB} over T_{NoLB} is:

$$\frac{T_{LB}}{T_{NoLB}} = \frac{CEIL(\frac{W}{N_D})}{CEIL(\frac{0.25W}{N_D})} \leq 4. \quad (15)$$

However a tight LB method such as [23] has a high complexity, which means the cost of resources to implement a tight LB module is also expensive. Further more, implementing a sort module for large inputs is also resource-expensive. As a result, more than $4 \times N_D$ LB modules and one sort module are implemented to achieve a maximum speedup of 4x. In fact, these resources can be used to implement more DTW modules. The control of the LB-involved implementation is more complex. Therefore when taking resources into consideration, there is no significant difference between the throughput of both implementations.

The second reason is that the throughput of LB-involved implementation varies for every candidate rather than be constant, which is not suitable for anomaly detection applications. Suppose in a medical anomaly detection application, the LB-involved implementation handles electrocardiogram signals. There may exist some periods that the throughput is low and the signals are cached. In this situations some important data may be dropped. This is unacceptable because it is coupled with human life. For no-LB implementation, the throughput is consistent, which will not drop any data if the throughput is larger than the input data rate.

3.5.2 Adaptation

Adaptation is a key requirement in many time series applications such as medical monitor [24]. The tradeoff between processing time (or throughput) and accuracy can be easily made in the proposed architecture. A large P can make a huge dimensionality reduction, which thus improves throughput but reduces accuracy. While a large W can provide more training templates, and the result can be more accurate with a relatively low throughput. If $K \leq N_D$, a large K has no influence on throughput with more accurate results. Otherwise the configuration with a large K sacrifices throughput for accuracy.

4. Experiment

4.1 Experiment Setup

Considering the power requirement of typical sensor-based devices, an FPGA architecture, Cyclone V SoC [25] is used to implement the proposed architecture. The architecture has 41509 Adaptive Logical Modules (ALM) and a 1 GB DDR3 SDRAM. The speed grade is $C8$. It should be noted that our implementation can be easily transplanted to more expensive FPGA devices with more ALMs and higher speed grade such as $C6$ and $C7$. The lower speed grade means higher clock frequency can be achieved. As what we emphasis is parameterization, this medial device Cyclone V SoC is selected to show its parameterization, and discuss its scalability, energy efficiency and performance.

As the features of Input/Output (IO) for different sensors and applications are different, we just ignore this difference, and the test data is stored in the DDR3, which provide data to the architecture in a stream pattern. Therefore, the I/O are not the bottlenecks, which has no impact on evaluating the architecture performance. And the IP core of DDR3 SDRAM is implemented, and asynchronous FIFOs are also used to link the IP core with the architecture to across different clock domains.

4.2 Architecture Characterization and Comparisons

4.2.1 FPGA-Based Architecture Characterization

For the sake of demonstration, we implement a 2-dimensional time series architecture in our experiment.

The number of parallel preprocessing module and the number of parallel task module are both one in the architecture. The resources cost of the implementation with different design parameters is shown in Fig. 8(a) and Fig. 8(b). The ALM cost varies according to different parameters. In Fig. 8(a), the resource utilization rate scales well with different PL and N. As new PEs can be added into the existing DTW modules easily, the resource utilization rate is almost linear to N. We can also observe that the growth rate of conditions with $PL = 2$ almost doubles that of conditions with $PL = 1$. This indicates that new added parallel lines have low impact of the architecture and the number of PEs with $PL = 2$ doubles that with $PL = 1$. As shown in Fig. 8(b), the resource utilization is almost linear to M, which shows the scalability of the proposed architecture.

Table 2 shows other characterization of the proposed

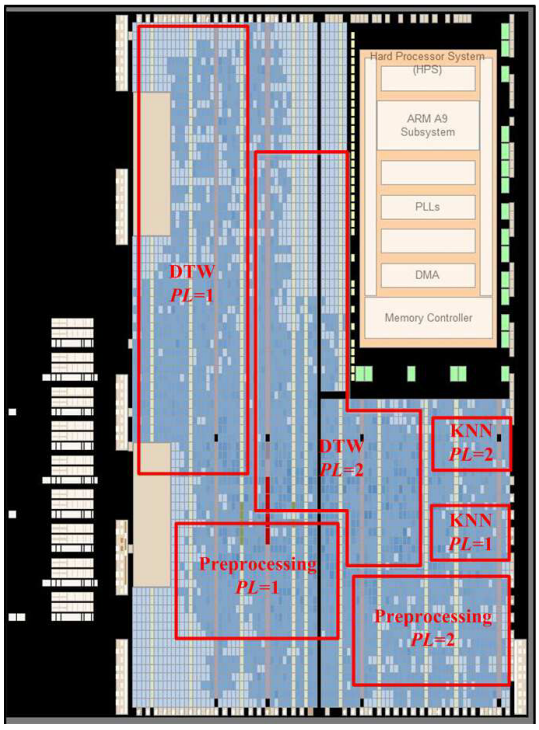


Fig. 7 The floorplan of the architecture implementation with PL=2, N_D=4 and M=4.

Table 2 Characterization of the proposed architecture.

Component	Parameter
System Clock	75 MHz
Max Throughput	900 Million Items/s
Register Utilization Rate	2%
Preprocessing Module Energy Proportionality	≤100%
DTW Module Energy Proportionality	≤75%
kNN Module Energy Proportionality	≤100%

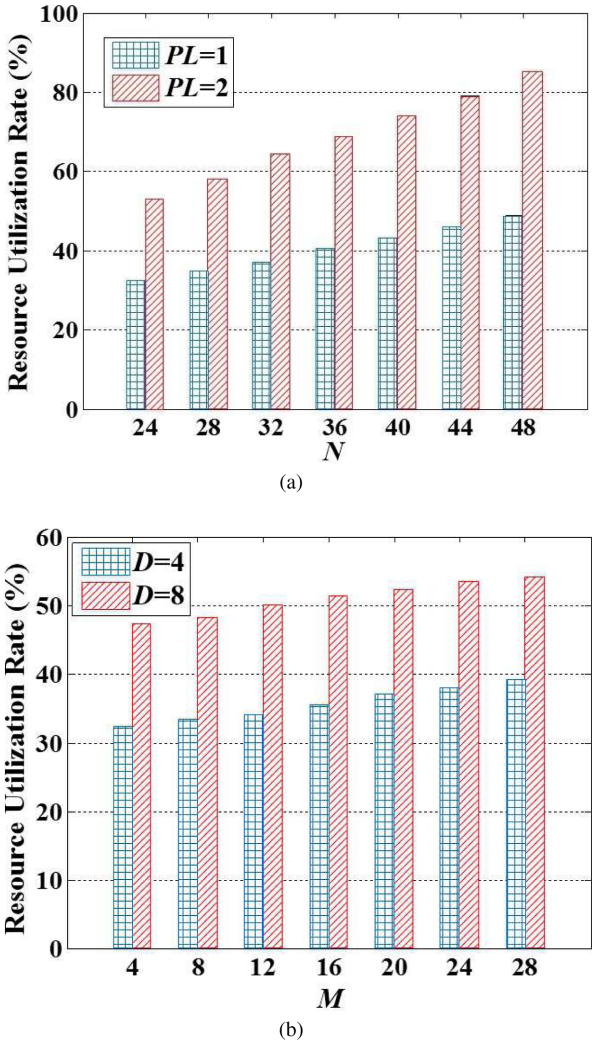


Fig. 8 Architecture implementation with (a)N_D=4, M=4, different PL and different N and (b) PL=1, N=24, different M and different N_D.

architecture. The cost of DTW modules occupies for the majority of the implementation. Though C8 is the lowest speed grade, we can still achieve a relatively high clock frequency of 75 MHz for the system. The clock frequency of preprocessing, DTW and kNN are 75 MHz, 75 Mhz and 120 MHz, respectively. For all of the design conditions, eight training templates are stored in the RAMs, the cost of block RAMs is constant. The number of training templates which are used during operation depends on the configuration of specific applications. Due to the feature of PE rings, new PEs can be integrated to the architecture, and the architecture clock can remain the same. It can be learned that the proposed architecture scales well with parallel lines, the DTW number and the PE number in kNN. The throughput varies for different configurations: L_c, L_t, P, W, R and K, and the throughput can be as large as 450 million items per second for one parallel line. For two parallel lines, the maximum throughput doubles.

Figure 7 shows the floorplan of the architecture implementation with PL=2, N_D=4 and M=4. And resource areas of two parallel lines with preprocessing, DTW and kNN are also highlighted.

4.2.2 Character Comparisons with Prior Work

We compare DTW and kNN modules with prior work, respectively. It should be highlighted that we don't claim to have higher performance, and that our achievement is parameterization to process different data streams locally with scalability, efficiency and relatively high performance. As the power of GPUs is too high (usually hundreds of watts) for general sensor-based embedded devices, it is not involved in the comparisons.

DTW comparison with prior work is shown in Table 3, in which the point distance corresponds to Eq. (4). The clock frequency of 5 KHz in [19] is specific for ultra low wearable applications, which is not for parameterized computing with high performance. Compared with [3] and [4], the clock frequency of this work is relativity low. This is due to two reasons. The first reason is that the adopted squared distance is more complex than absolute distance. Therefore, the im-

Table 3 DTW implementation comparison with prior work (Cycles needed means the cycles needed to prune one result (Clock = clock frequency, and Para.=Parameterization)).

Work	Clock	Point Distance	Cycles	Para.	Device
[19]	5 KHz	Absolute distance	L	No	RTL
[3]	240 MHz	Absolute distance	L	No	Virtex 5 LX330
[4]	167.8 MHz	Absolute distance	$\geq L$	L_c	Stratix EP4SGX530
Our work	75 MHz	Squared distance	$\geq L$	L_c, L_r, P, W, R, K	Cyclone V SoC

Table 4 kNN implementation comparison with prior work (Cycles needed means the cycles needed to prune one result. NofE. means the input number of elements for kNN module (Clock=Clock Frequency, and Para.=Parameterization)).

Work	Clock	Cycles	Para.	Device
[13]	20 MHz	NofE.	No	Stratix 1P1S40
[2]	50 MHz	NofE.	No	Schematic form
[14]	209 MHz	NofE.	No	XC2VP70-6
[16]	180.8 MHz	NofE.	K	XC4VFX12
Our work	120 MHz	≥NofE.	K	Cyclone V SoC

plementation delay of squared difference is much larger than absolute difference, resulting in a relatively low clock frequency. The squared distance is adopted because it is widely used in DTW software implementations [6], [23]. The second reason is that compared with the FPGA devices in [3] and [4], the speed grade of the selected median FPGA device is relatively low. It should be noticed that our implementation can be easily extended to high speed grade FPGAs with a higher clock frequency. Both [4] and this work provide parameterization for L_c and L_t due to PE structures, thus the cycles needed to prune one result is larger than l. Additionally, this work provides parameterization for N and R.

kNN comparison with existing work is shown in Table 4. Compared with recent kNN accelerations [2], [13], [14], [16], the clock frequency of this work is relatively low. This is mainly due to the low speed grade of the selected FPGA device. Only [16] and this work support parameterized K. [16] supports limited number of K by dynamic partial reconfiguration. However, our work can support arbitrary numbers of K.

4.3 Comparison with Multi-Core CPU for Anomaly Detection

A comparison between the proposed FPGA-based architecture and multi-cores CPU-based implementation is presented for electrocardiogram (ECG) anomaly detection. In kNN-based anomaly detection, the $ScoreSum$ in Fig. 6 determines whether the candidate is anomaly or not. If the $ScoreSum$ is larger than the predefined $Threshold$, return $anomaly$, otherwise return $normal$. The predefined $Threshold$ is produced in a training processing.

4.3.1 Experiment Overview

The two-dimensional ECG data are from the MIT-BIH Long Term Database [26] with a length of about 20 hours for each patient. We divide the long ECG data to subsequences of length 96 with software according to ECG length in [27]. The peak points are from the *annotations.txt* from the database. With the peak point a_i, we segment subsequences as $\{a_{i-31}, \cdots, a_i, \cdots, a_{i+94}\}$. Thus the length of candidate and templates sequences are equal, and L=L_c=L_t. The category (anomaly or normal) of each subsequence is also labeled in the *annotations.txt* from the database. Totally over 470,000 suitable subsequences with 7 patients are produced, in which 453,637 subsequences are normal and 24,410 subsequences are anomaly.

A training processing is presented to get $Threshold$. The first W normal subsequences are stored as training templates. The following 100 normal subsequences are used to do training to get the $Threshold$. The other normal and anomaly subsequences are used to test the architecture.

For the FPGA-based architecture, the design parameter is as follows: PL=2, $N_p = 1$, $N_t = 1$, $N_D = 4$, $N = 24$, $M = 4$. Two parallel lines is to double the processing throughput. Performance is evaluated with different combination of P, W, R and K. During the experiment, the FPGA architecture firstly stores the ECG subsequences in the DDR3, and the DDR3 provides data for the architecture in a stream pattern.

For CPU implementation, the Intel Core (TM) i5-3470 CPU with clock frequency of 3.2 GHz is selected, which has 4 cores. The DTW-kNN algorithm for anomaly detection is implemented in C with double precision. The GCC4.9.1 compiler is introduced and the optimization level is $O3$. The lower bound method proposed by Keogh [6] is adopted. The multi-core processing API, OpenMP [28] is introduced. For efficient parallel computing, each core processes a quarter of all the candidate subsequences. As in each core computation is serial, multi-threaded computation is not implemented for each core. The C implementation of DTW and the adopted lower bound methods are from [6]. As SSE (Streaming Single Instruction Multiple Data(SIMD) Extensions) has almost no optimization when the query length is shorter than 300 [3], SSE is not implemented in the software implementation. **All the subsequences are stored in RAM. Thus no file operation is involved in the experiment**. Therefore we can get a competitive software implementation.

Comparisons of accuracy, runtime and energy effi-

ciency are presented in the experiments. The energy efficiency $E_{efficiency}$ is defined as shown below:

$$E_{efficiency} = \frac{ItemNumber}{Energy}, \tag{16}$$

where $ItemNumber$ is the total number of tuples processed by the architecture and $Energy$ is the energy consumption.

An approximate method is adopted to measure the power of CPUs and FPGAs. As the features of Input/Output (IO) for different sensors and applications are different, the I/O energy is not taken into considered for generality. For CPU, the thermal design power is 77W [29], and the average utilization rate during the experiments is recorded. Therefore the estimated power of CPU is the product of the two parameters. For FPGA, the estimated power is produced from $Powerplay$, a power analyzer tool of Quartus [30]. This estimation method is widely used to estimate FPGA power [5]. The estimate power of FPGA is 424.2mW without I/O power.

4.3.2 Accuracy Benchmark

As shown in Fig. 9(a) and Fig. 9(b), accuracy comparisons with different configurations are presented. Though with precision reduction and PAA to reduce dimensionality and only four training templates, the FPGA architecture still has a 98.5% normal detection rate and a 72.5% anomaly detection rate, and a totally 96.8% detection rate. The CPU implementation also achieve high detection rate of normal detection with 98.8%, anomaly detection with 70.4% and total detection with 97.3%.

It can be noticed that accuracies are almost the same with the same configuration for CPU and FPGA in both figures. For normal detection, the accuracy of CPU overcomes FPGA in most configurations. For anomaly detection, however the accuracy of FPGA is better. Therefore precision reduction and PAA have a rather low influence on accuracy compared with CPU with double precisions. And the proposed architecture has a moderate precision.

4.3.3 Performance Benchmarking

Figure 10(a) and Fig. 10(b) shows the comparison of runtime of FPGA and CPU with different configurations. The speedup varies from 3x to 14x with different configurations.

For CPU, the runtime is almost linear to W with serial feature. Compared with CPU, the runtime of FPGA is much shorter owing to parallelism and pipelining.

The most interesting thing is that both curves of the speedup and the runtime of FPGA have jumps. For FPGA when $W \leq 4$, the number of DTW modules is enough to calculate DTW distances between all training templates and the input candidate subsequences in one run. So the throughputs are the same. When $5 \leq W \leq 8$, all the DTW distances are finished with two runs by DTW modules, leading to a lower throughput and a runtime jump. For FPGA, the speedup are divided into intervals. Each interval has four points (except the first interval) and in each interval the speedup increases with W.

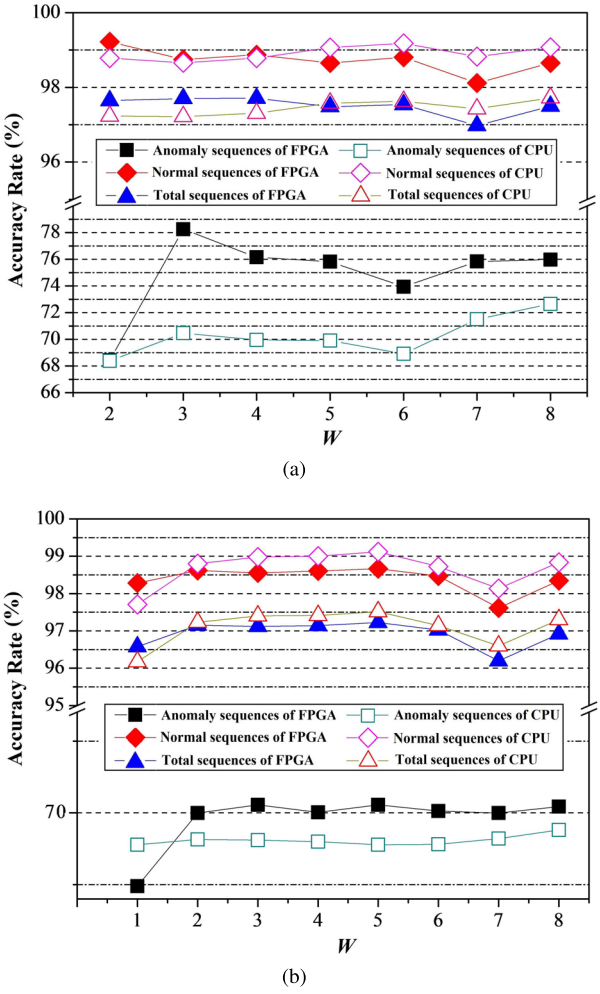


Fig. 9 Accuracy of the proposed architecture with (a) $P = 4$, $R = 6$, $K = 2$ and (b) $P = 2$, $R = 11$, $K = 1$.

We can take a more indepth comparison of the runtime of different configurations in Fig. 10(a) and Fig. 10(b). For the condition of P=4,W=4,R=6,K=2, the configurations fit well into the input subsequence length and the architecture has a large throughput. But for condition of P=2,W=4,R=13,K=1, the input subsequence length exceeds the PE number in DTW modules, and each DTW module has to process candidate subsequence with quadruple time compared with the condition of P=4,W=4, R=6,K=1. So the throughput decreases and the runtime increases. For the condition of P=4,W=8,R=6,K=2, the number of DTW modules is not enough to calculate DTW distances between all training templates and the input candidate subsequences in one run. Thus the four DTW modules have to calculate DTW distances between eight training templates and one candidate subsequences in two runs, which also results in a higher runtime. For the condition of P=2,W=8,R=6,K=1, both of above two circumstances trigger. And the runtime is the largest. It also can be observed that K makes not much

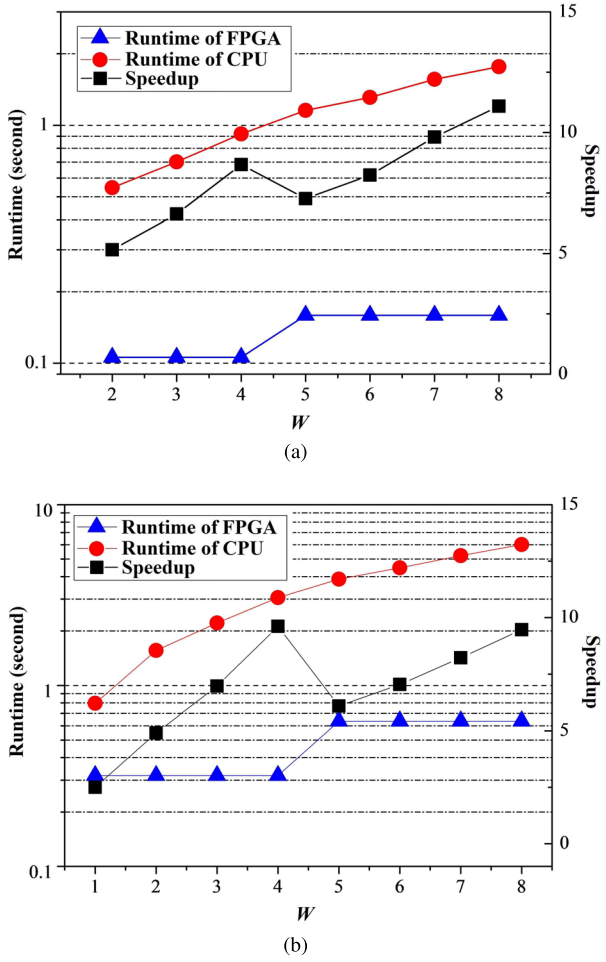


Fig. 10 Runtime and speedup of the proposed architecture with (a) $P = 4$, $R = 6$, $K = 2$ and (b) $P = 2$, $R = 11$, $K = 1$.

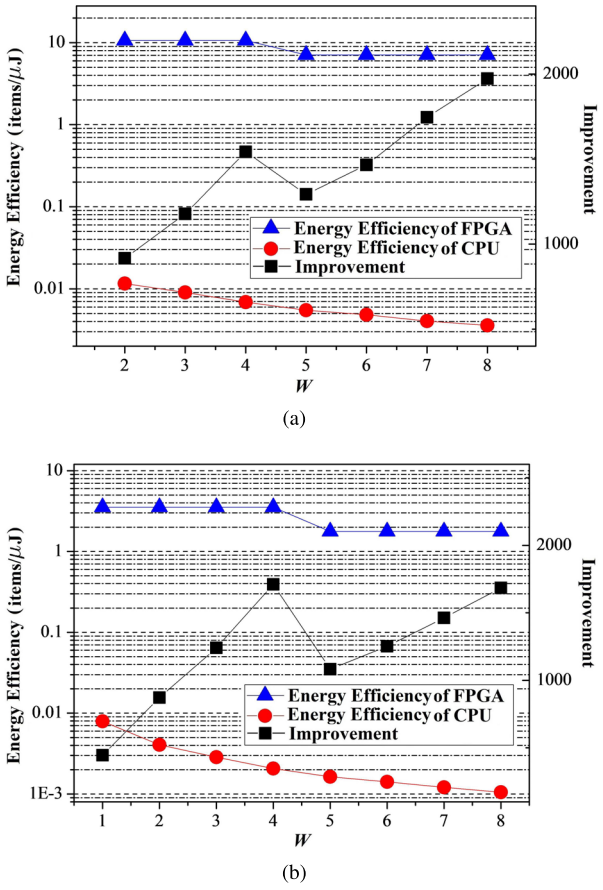


Fig. 11 Energy efficiency and improvement of the proposed architecture with (a) $P = 4$, $R = 6$, $K = 2$ and (b) $P = 2$, $R = 11$, $K = 1$.

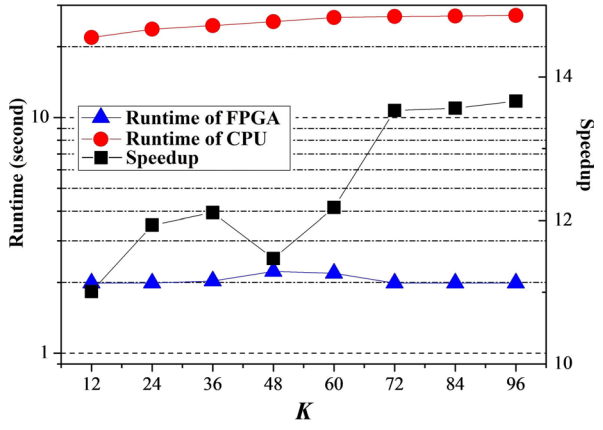


Fig. 12 Runtime of FPGA and CPU with $P = 4$, $W = 100$.

difference, it is due to the fact that for CPU the calculation is light weighted and for FPGA when $K \leq 24$ the kNN module will not block the processing.

Figure 12 shows runtime and speedup of FPGA and CPU with different K. For CPU, the runtime is almost the same with different W. As DTW calculations are the main runtime contributor for CPU,the kNN module involved with logical operations have little impact on runtime. However for FPGA, the kNN module may become the bottleneck module and degrade the performance. For DTW modules, the number of cycles to pump one candidate subsequence is fixed to $L * CEIL(W/N_D) = 600$ with different K. For kNN modules, the number of cycles to pump one candidate is $K + K * CEIL((W-K)/M) = K + K * CEIL((100-K)/4)$, which varies with K. The cycles are $612, 672, 660$ with $K = 36, 48, 60$, respectively. For other K, the cycles is less than 600. So the runtime with $K = 36, 48, 60$ increases and the corresponding speedup decreases.

4.3.4 Energy Efficiency Benchmarking

Figure 11(a) and Fig. 11(b) show a comparison of energy-efficiency of FPGA and CPU with different configurations. And the improvement can be as large as almost 2,000x. As $E_{efficiency} = \frac{ItemNumber}{Energy} = \frac{ItemNumber}{Power*(ItemNumber/Throughput)} = \frac{Throughput}{Power}$, the energy efficiency is almost inversely proportional to runtime. The energy efficiency with W is divided into intervals, in which energy efficiency is the same. And the data in Fig. 11(a) has the same tendency with Fig. 11(a). Figure 11(b) shows the

same tendency.

5. Conclusions

In this paper we presented a scalable, efficient and parameterized architecture for stream mining on sensor-based devices. Particularly, a widely-used similarity measure DTW and one of the most popular classifier kNN are adopted for stream mining. The proposed architecture can achieve scalability, energy efficiency and parameterization for multiple parameters, while achieving a high throughput of 900 million items per second. The PAA number can be configured during running to achieve different levels of representation accuracy. The length of subsequences is auto adapted. The numbers of PE in DTW module and kNN module, the number of DTW modules and the number of parallels can be easily modified before programming. Additionally, the number of tuples used to calculate one tuple in PAA, training templates number, warping path constraint and K in kNN can be dynamically configured during running. Compared with multi-core CPU based implementation, the improvement of runtime and energy efficiency are up to 14x and almost 2,000x with almost the same accuracy.

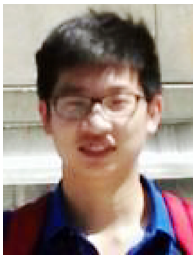
Acknowledgements

This work is sponsored by the National Science Foundation of China, under grant 61272070 and 61376031. This work is in partially supported by NSF of United States of America under grant CNS-1423061, ECCS-1462498 and CNS-1547167.

References

[1] D. Evans, "The Internet of things: How the next evolution of the Internet is changing everything," http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf, 2011.
[2] Y.J. Yeh, H.Y. Li, W.J. Hwang, and C.Y. Fang, "FPGA implementation of kNN classifier based on wavelet transform and partial distance search," Image Analysis, pp.512–521, Springer, 2007.
[3] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul, "Accelerating dynamic time warping subsequence search with GPUs and FPGAs," Data Mining (ICDM), 2010 IEEE 10th International Conference on, pp.1001–1006, IEEE, 2010.
[4] Z. Wang, S. Huang, L. Wang, H. Li, Y. Wang, and H. Yang, "Accelerating subsequence similarity search based on dynamic time warping distance with FPGA," Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp.53–62, ACM, 2013.
[5] J. Tarango, E. Keogh, and P. Brisk, "Instruction set extensions for dynamic time warping," Proc. Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, p.18, IEEE Press, 2013.
[6] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," Proc. 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.262–270, ACM, 2012.
[7] C.S. Myers and L.R. Rabiner, "Connected digit recognition using a level-building dtw algorithm," IEEE Trans. Acoust., Speech, Signal Process., vol.29, no.3, pp.351–363, 1981.
[8] Y. Zhang, K. Adl, and J. Glass, "Fast spoken query detection using lower-bound dynamic time warping on graphical processing units," Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on, pp.5173–5176, IEEE, 2012.
[9] A. Kia, SamanHaratizadeh and HadiZare, "Prediction of USD/JPY exchange rate time series directional status by KNN with dynamic time warping AS distance function," Bonfring International Journal of Data Mining, vol.3, no.2, pp.12–16, 2013.
[10] W.A. Chaovalitwongse, Y.J. Fan, and R.C. Sachdeo, "On the time series K-nearest neighbor classification of abnormal brain activity," IEEE Trans. Syst., Man, Cybern. A, Syst.,Humans, vol.37, no.6, pp.1005–1016, 2007.
[11] H.H. Hsu, A.C. Yang, and M.D. Lu, "KNN-DTW based missing value imputation for microarray time series data," J. Comput., vol.6, no.3, pp.418–425, 2011.
[12] X. Xi, E. Keogh, C. Shelton, L. Wei, and C.A. Ratanamahatana, "Fast time series classification using numerosity reduction," Proc. 23rd International Conference on Machine learning, pp.1033–1040, ACM, 2006.
[13] S. Lucas, "A fast exact parallel implementation of the k-nearest neighbour pattern classifier," Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on, pp.1867–1872, IEEE, 1998.
[14] I. Stamoulias and E.S. Manolakos, "Parallel architectures for the knn classifier–design of soft IP cores and FPGA implementations," ACM Trans. Embedded Computing Systems (TECS), vol.13, no.2, p.22, 2013.
[15] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on, pp.1–6, IEEE, 2008.
[16] H.M. Hussain, K. Benkrid, and H. Seker, "An adaptive implementation of a dynamically reconfigurable K-nearest neighbour classifier on FPGA," Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on, pp.205–212, IEEE, 2012.
[17] Y. Sakurai, C. Faloutsos, and M. Yamamuro, "Stream monitoring under the time warping distance," Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on, pp.1046–1055, IEEE, 2007.
[18] C. Hundt, B. Schmidt, and E. Schomer, "Cuda-accelerated alignment of subsequences in streamed time series data," Parallel Processing (ICPP), 2014 43rd International Conference on, pp.10–19, IEEE, 2014.
[19] R. Jafari and R. Lotfian, "A low power wake-up circuitry based on dynamic time warping for body sensor networks," Body Sensor Networks (BSN), 2011 International Conference on, pp.83–88, IEEE, 2011.
[20] R. Lotfian and R. Jafari, "An ultra-low power hardware accelerator architecture for wearable computers using dynamic time warping," Proc. Conference on Design, Automation and Test in Europe, pp.913–916, EDA Consortium, 2013.
[21] E. Keogh and S. Kasetty, "On the need for time series data mining benchmarks: A survey and empirical demonstration," Data Mining and knowledge discovery, vol.7, no.4, pp.349–371, 2003.
[22] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh, "Indexing multi-dimensional time-series with support for multiple distance measures," Proc. ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.216–225, ACM, 2003.
[23] E. Keogh and C.A. Ratanamahatana, "Exact indexing of dynamic time warping," Knowledge and Information Systems, vol.7, no.3, pp.358–386, 2005.
[24] M. Chen, S. Gonzalez, A. Vasilakos, H. Cao, and V.C. Leung, "Body area networks: A survey," Mobile Networks and Applications, vol.16, no.2, pp.171–193, 2011.
[25] Terasic, "Sockit - the development kit for new soc device," http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=816, 2011.

[26] A.L. Goldberger, L.A. Amaral, L. Glass, J.M. Hausdorff, P.C. Ivanov, R.G. Mark, J.E. Mietus, G.B. Moody, C.K. Peng, and H.E. Stanley, "Physiobank, physiotoolkit, and physionet components of a new research resource for complex physiologic signals," Circulation, vol.101, no.23, pp.e215–e220, 2000.
[27] E. Keogh, X. Xi, L. Wei, and C.A. Ratanamahatana, "The UCR time series classification/clustering homepage," URL= http://www.cs.ucr.edu/˜eamonn/time_series_data, 2006.
[28] OpenMP, "OpenMP," http://openmp.org/wp/openmp-compilers/, 2014.
[29] Intel, "Intel core i5-3470 processor," http://ark.intel.com/products/68316/Intel-Core-i5-3470-Processor-6M-Cache-up-to-3_60-GHz, 2014.
[30] Altera, "Quartus ii powerplay," http://www.altera.com.cn/support/software/power/sof-qts-power.html, 2014.



Li Zhang received the B.S. degree in electronic science and technology from Central China Normal University, Wuhan,China, in 2011, where he is currently working toward the Ph.D. degree in the School of Optical and Electronic Information, Huazhong University of Science and Technology. His current research interests include analog integrated circuits and inductively-coupling connection.

Dawei Li received the B.S. degree in electronic science and technology from Huazhong Agricultural University, Wuhan,China, in 2011, where he is currently working toward the Ph.D. degree in the School of Optical and Electronic Information, Huazhong University of Science and Technology.

Xuecheng Zou received the Ph.D. degree in electronic science and technology from Huazhong University of Science and Technology, Wuhan, China, in 1993. He is currently a Professor with the School of Optical and Electronic Information, Huazhong University of Science and Technology. His research interests include IC design, inductively-coupling connection and Internet of Things.

Yu Hu received the Ph.D. degree in Department of electronic engineering from University of California, Los Angeles. He is currently a Professor with the School of Optical and Electronic Information, Huazhong University of Science and Technology. His research interests include IC design, inductively-coupling connection and Internet of Things.

Xiaowei Xu received the Ph.D. degree in Huazhong University of Science and Technology in 2016. His current research interests include analog integrated circuits and Internet of things.