Scalable and Parameterized Dynamic Time Warping Architecture for Efficient Vehicle Re-identification

Guanbing Deng, Hanqing Zhou, Guangyu Yu, Zeyu Yan, Yu Hu, Xiaowei Xu*
School of Optical Science and Electronic Information

Huazhong University of Science and Technology, Wuhan, China

xiaoweixu@hust.edu.cn

***Abstract*—Vehicle Re-identification is an effective way for calculation of travel time and origin-destination matrices, which is critical for traffic modeling and optimization. Recently, dynamic time warping (DTW) with magnetic signature has been a popular method for vehicle re-identification. However, with the increasing number of vehicles and the real-time requirement for travel time estimation, the calculation of DTW needs further acceleration. In this paper, we propose a scalable and parameterized architecture on reconfigurable fabrics for efficient vehicle re-identification. Parameterization is adopted to support various lengths of magnetic sequences. The experimental results with magnetic signature indicate that compared 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.**

Keywords—Vehicle Re-identification; Dynamic Time Warping; Field Programmable Gate Array; Scalability; Parameterization

I. INTRODUCTION

Traffic management is becoming more and more challenging with the increasing number of vehicles. Vehicle re-identification is one of the most important methods for traffic management, which is the precondition of vehicle behavior analysis and judgment. Vehicle re-identification not only provides real-time monitoring of dynamic traffic, but also prepares the subsequent specific processing, such as sorting operation traffic related data, early warning emergency accidents and optimizing the traffic flow. Therefore, vehicle re-identification is an effective way for calculation of travel time and origin-destination matrices, which is critical for traffic modeling and optimization. With enough dynamic data collected with vehicle re-identification, the urban traffic flow can be modeled online, and it provides road participants with dynamic travel time and optimal destination path. Thus, it has the potential to improve the traffic environment, reduce traffic congestion and improve road utilization. One challenging issue in road traffic management is how to improve the precision of vehicle detection, response speed and anti-jamming ability.

In vehicle re-identification, re-identification accuracy is one of the most important parameter, which determines the quality of the vehicle re-identification. Recently, dynamic time warping (DTW) with magnetic signature has been a popular method for vehicle re-identification. Compared with the VSN240 system by UC Berkeley [2] and multiple sensor nodes strategy [3], Dynamic Time Warping based on Gaussian maximum classifier [4] is able to re-identify 80% of vehicles without any false alarms.

However, each processing unit is connected to thousands or even tens of thousands of magnetic sensor nodes to perform real-time vehicle re-identification. The huge number of magnetic signature devices produce massive quantities and multiple categories of data streams. Considering the costly ASIC for small volume product and the low-performance microcontroller, high-performance and energy-efficient FPGAs are promising to handle real-time stream mining [5] [6][7].

In this paper, we propose a scalable, parameterized and efficient FPGA-based architecture for stream mining on sensor-based devices. What we emphasize is parameterization for processing different categories of streams by detecting different categories of vehicles, and we achieve scalability efficiency at the same time with a relatively high performance. Particularly, k-Nearest Neighbor (kNN) [8], a widely used classifier, and Dynamic Time Warping (DTW), a popular distance measure [7], are adopted in the architecture. Specific data representation and precision reduction techniques are also integrated in the architecture.

This paper is organized as follows. In Section II, the re-identification method is presented. In Section III, the vehicle magnetic signature capture system is described. In Section IV, the hardware architecture for DTW acceleration is presented. The results include characterization and the comparisons with multi-core CPU-based implementation are demonstrated in Section V. Finally, we conclude this paper in Section VI.

II. METHOD

A. DTW-based Vehicle Re-identification

DTW (dynamic time warping) is used to compare two sequences, which may not be equal to the length of the time series. In most cases, two sequences as a whole has a very similar shape, but these shapes on the x axis are not aligned. So we need to wrap their time axis before compare their comparison.

Magnetic sensor signal curve is not only associated with the material of vehicles, but also associated with the speed of the vehicles. If the same car passes over the magnetic sensor at a different speed, the sensor signal will change differently, stretching or contracting on the timeline. Therefore, DTW algorithm is especially suitable for the situation. DTW through extension and shorten the time sequences to calculate the similarity between two time series. Fig. 1 shows the example of the three-axis magnetic signal curve.

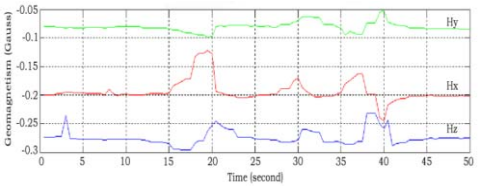


Fig. 1. Different magnetic signature when different vehicle passing.

Usually 1NN-based DTW is adopted for vehicle identification. When vehicles pass magnetic sensor nodes, their corresponding geomagnetic signals are extracted. Then by comparing geomagnetic signals on two sensor nodes, the same vehicle can be re-identified. For example, when vehicle A passes some geomagnetic sensor node, its corresponding geomagnetic signal is compared with the geomagnetic signals in the sensor nodes along the possible paths within a suitable time-window for vehicle A. Therefore, the path with the most similar geomagnetic signal will be considered as the route for vehicle A. Thus, the origin-destination matrix can be produced with a high precision. Since each sequence exists three axial magnetic curve, distance of individual elements with three axis is calculated as follows:

$$D^2(i,j) = \frac{(X_T(i)-X_S(j))^2 + (Y_T(i)-Y_S(j))^2 + (Z_T(i)-Z_S(j))^2}{3} \quad (1)$$

Where $X_T(i), Y_T(i)$, and $Z_T(i)$ are previous element values in x, y, z axis, and $X_S(j), Y_S(j)$ and $Z_S(j)$ are the current sequence in x, y, z axis. The above calculation is performed between the current sequence and all the previous sequences within a suitable time window. Thus, the number of DTW calculations is relatively large considering the high volume of traffic and the large number of magnetic sensor nodes.

III. VEHICLE MAGNETIC SIGNATURES CAPTURE SYSTEM

A. System Architecture

We implement a vehicle magnetic signature capture system adopting wireless sensor to collect traffic data in real time as shown in Fig. 2.Tens of nodes collect geomagnetic signals that will be sent to the Access Point (AP). Then the AP gathers the data collected by sensor nodes and send them to the nearby AP with high processing capacity with a timestamp. On receiving the traffic data, the AP with high processing capacity will perform vehicle re-identification process for each vehicle geomagnetic signals. In this paper, we add FPGAs to the AP and perform high-throughput DTW-based vehicle re-identification.

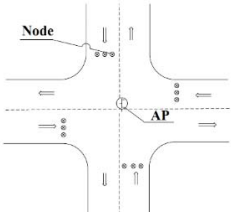


Fig. 2. Vehicle magnetic signature capture system.

B. Sensor node structure

Sensor node is composed of wireless communication module, processor module, sensor module, and power module as shown in Fig. 3.The power module provides energy and manages battery, and the wireless module communicates between sensor node and AP. The sensor module is responsible for magnetic signal collection, and then does some transformation if needed. The Microprocessor units, includes MCU, memory and embedded systems. Our sensor node and AP are only about 12cm in diameter and 3cm in thickness.

Fig.3. Hardware picture of sensor node and AP. (They share the same hardware implementation.)

IV. ARCHITECTURE

A. Architecture Overview

The top-level block diagram of the proposed stream mining architecture is shown in Fig. 4. 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 usually is a very time-consuming module. Therefore, multiple parallel distance metric modules are implemented for speedup. 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.

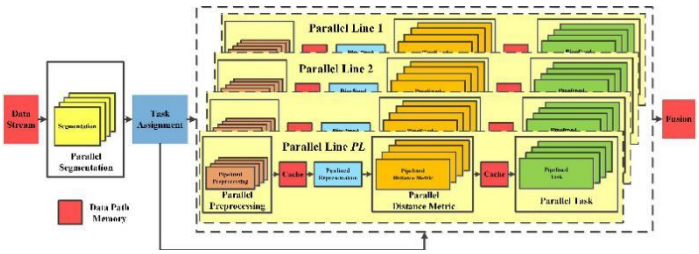


Fig. 4. Scalable and parameterized stream mining architecture.

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 [5], precision reduction is made in PAA module to reduce the data accuracy from 16 bits to 8 bits (The main concerning 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, we will discuss the enhancements in detail. The major notations and parameters used in this work are summarized in Table I.

B. DTW module

In this section, we adopt the DTW design from Wang et al. [7], which is one of the fastest recent DTW implementation. The DTW design from [7] adopted the SPRING algorithm[11], and can support arbitrary length of sequence length. As the data is not normalized in 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. 5, *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 when to send candidate subsequences 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

Table I. A Summarization of Notations Used in This Paper

Notations	Descriptions
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
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	Candidate subsequences' length
L_T	Template subsequences' length
K	K nearest neighbors in kNN
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

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.

Fig. 6 shows the work flow of the DTW module. The 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: C_1 = (1, 1), (0, 4), (1, 3), and C_2 = (2, 1), (1, 3), (2, 1). In the first cycle, the first tuple of C_1(1, 1) is sent to PE 1 and the [1, 1] cell of DTW matrix between C_1 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 C_2(2, 1) is sent to PE 1. In the 5th cycle, the calculation of DTW matrix between C_1 and Q is completed.

The *Result Controller* sends the result from PE3 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 convenient to add PEs into the ring when resources are abundant.

The subsequence length of candidates is parameterized. If the subsequence length L (suppose $L_c = L_t = 3$) is larger than the number of PEs, the FIFO in the ring is activated. As shown in Fig. 5, the first *N* columns are processed, and the results of the *N*th 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. The structure is not changed 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.

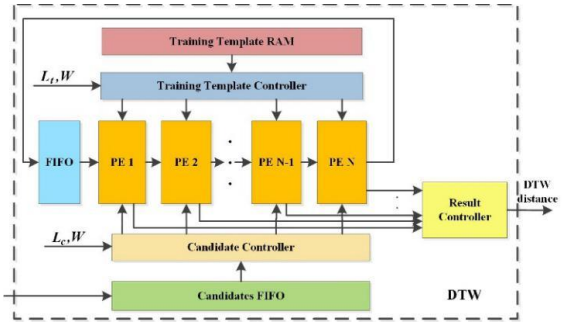


Fig. 5. DTW Module Design structure.

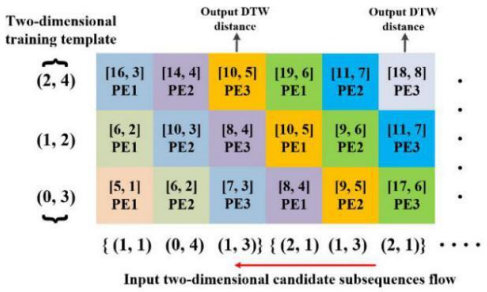


Fig. 6. Workflow illustration of DTW module: column 1-3 and 4-6 are the DTW matrixes between query Q and candidates C1 and C2, respectively. (a, b) is a two dimensional tuple. In [c, d], c is the accumulated DTW distance and d is the cycle time. The item below [c, d] indicates which PE processes the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

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. If 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 will increases from N to $N*CEIL(W/N_D)$ .The warping path constraint R is parameterized. Supposing that a PE is processing the *i*th element of one candidate and the *j*th element of one training template. If $j-R \le i \le j+R$, the accumulated DTW distance is calculated as discussed in Fig. 6, otherwise the accumulated DTW distance is the maximum 32-bit integer. Thus, R has no influence on the throughput.

C. kNN module

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. 7, *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 calculate the sum of them. 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 are sent to the output port.

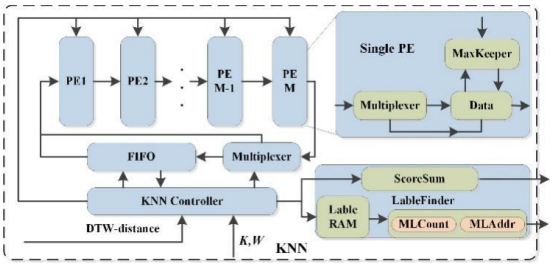


Fig. 7. kNN Design Structure.

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 than L, the performance will degrade.

V. EXPERIMENT AND RESULT

A. Setup

The magnetic sensor data is deployed at six sites inside and outside of the campus of Huazhong University of Science and Technology. In practical applications, the sensor node will be placed in the middle of the road. Therefore, we placed magnetic sensor at 3 different sites inside Huazhong University of Science and Technology campus. However，for the 3 sites outside school, we place the geomagnetic sensor at the side of road for the sake of safety, it is about 2m away from center of the lane. As depicted in Fig. 8. X-axis of the sensor is perpendicular to the moving direction of the detected vehicle. Y-axis is parallel to the moving direction of the detected vehicle. Z-axis is vertical to the road surface.

When installing the sensor node, we should pay attention to initialize it correctly, and make sure that it will not move and rotate neither be affected by external vibration. Then install the AP, making sure it is not far away from the node. The AP and sensor node establish a network through the Zigbee protocol, and use the Zigbee communication module to communicate. After receiving data, the AP will open its GPRS module to send data to internet through the TCP/IP protocol.

According to the principle of magnetic sensor for vehicle detection, the magnetic field will change due to the motion of the vehicle, making it easy to identify the existence of the vehicle. What's more, the magnetic field varies differently as different vehicle passing, depending on different material and mechanical structure of the passing vehicle.

Once sensor nodes and APs are working properly and the network server are setting up, we can view the receive data through the PC remote desktop as shown in Fig. 9. When a vehicle passes through, the magnetic signal will change immediately. By the means of screen shot, we can save the current data and count corresponding vehicles' number for some time. Totally, we collected 400 pair of vehicles.

Fig. 8. Deployment of sensor node on the road.

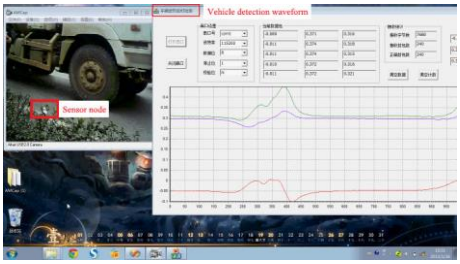


Fig. 9.Vehicle detection waveform in the PC.

B. Characterization

1) FPGA-based Architecture Characterization

We implement a 3-dimensional time series architecture in our experiment. The number of parallel preprocessing module and the number of parallel task module are both in the architecture. The resources cost of the implementation with different design parameters is shown in Fig. 10(a) and Fig. 10(b). The ALM cost varies with different parameters. In Fig. 10(a), the resource utilization rate scales well with different PL and N. As new PE scan 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 on the architecture and the number of PEs with *PL*=2 doubles that with *PL*=1. As shown in Fig. 10(b), the resource utilization is almost linear to M, which shows the scalability of the proposed architecture.

Table II shows other characterization of the proposed 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 75MHz for the system. The clock frequency of preprocessing, DTW and kNN are 75 MHz, 75MHz and 120MHz, respectively. As eight training templates are stored in RAMs for all design configurations, the cost of block RAMs are the same and the number of training templates used during operations is up to configurations for 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.

2) Character Comparisons with Prior Work

We compare DTW and kNN modules with prior work, respectively. It should be highlighted that we do not 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 III. The clock frequency of 5KHz in [12] is specific for ultra low wearable applications, which is not for parameterized computing with high performance. Compared with [5] and [7], 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 implementation 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 [9][13]. The second reason is that compared with the FPGA devices in [5] and [7], 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 [7] 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 IV. Compared with recent kNN accelerations [14] [15] [16] [17], the clock frequency of this work is relatively low. This is mainly due to the low speed grade of the selected FPGA device. Only [17] and our work support parameterized *K*. [17] adopts dynamic partial reconfiguration. However, our work can support arbitrary *K*.

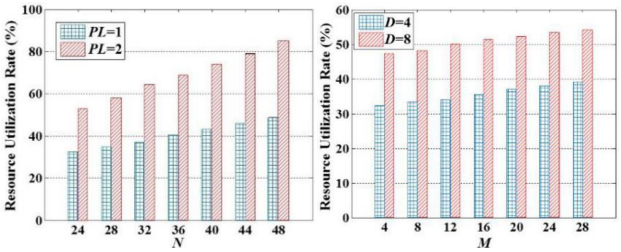


Fig. 10. Architecture implementation with (a) N_D =4, M =4, different PL and different N and (b) PL =1, N =24, different M and different N_D.

Table II. Characterization of the proposed Architecture.

Component	Parameter
System Clock	75MHz
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%

Table III. 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**
[**12**]	5KHz	Absolute distance	L	N/A	RTL
[**5**]	240MHz	Absolute distance	L	N/A	Virtex5 LX330
[**7**]	167.8MHz	Absolute distance	≥L	*Lc*	Stratix EP4SGX530
Our work	75MHz	Squared distance	≥L	*Lc, Lr , P, W, R, K*	Cyclone V SoC

Table IV. kNN implementation comparison with prior work

Work	**Clock**	**Cycles**	**Para.**	**Device**
[14]	20MHz	NofE	N/A	Stratix 1P1S40
[15]	50MHz	NofE	N/A	Schematic form
[16]	209MHz	NofE	N/A	XC2VP70-6
[17]	180.8MHz	NofE	K	XC4VFX12
Our work	120MHz	≥NofE	K	Cyclone V SoC

C. Comparison with multi-core CPU-based Implementation

A comparison between the proposed FPGA-based architecture and multi-cores CPU-based implementation is presented for vehicle re-identification. In kNN-based vehicle re-identification, the *ScoreSum* determines the most similar vehicle for the candidate.

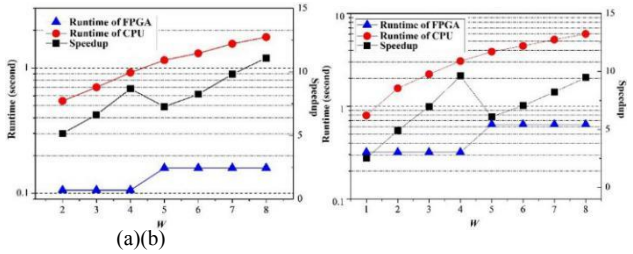
1) Performance Benchmarking

Fig. 11(a) and Fig. 11(b) show 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 $W \le 4$, 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 $5 \le W \le 8$, 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.

We can take a more in-depth comparison of the runtime of different configurations in Fig. 11(a) and Fig. 11(b). For the condition of P = 4, W = 4, R = 6, K = 2, the configurations fit well of the input subsequence length and the architecture has a large throughput. But for condition of P = 2,W = 4, R = 11, 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 = 2. Therefore, 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 = 2, both of above two circumstances trigger. Moreover, the runtime is the largest. It also can be observed that K makes not much difference; it is due to the fact that for CPU the calculation is lightly weighted and for FPGA when $K \le 24$ the kNN module will not block the processing.

Fig. 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, and 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, 660with K=36, 48, 60, respectively. For other K, the cycles are less than 600. So the runtime with K=36, 48, 60 increases and the corresponding speedup decreases.



(a)(b)

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

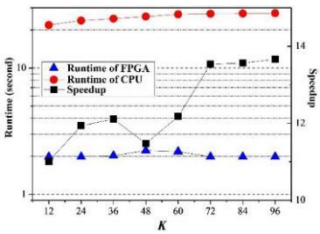
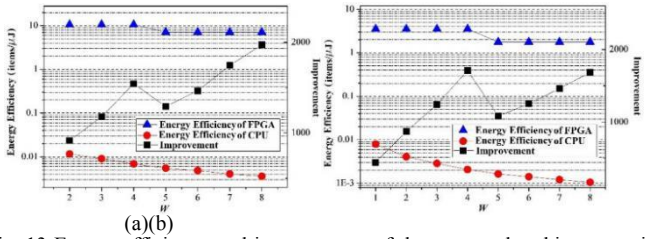


Fig. 12 Runtime of FPGA and CPU with P= 4, W = 100.

2) *Energy Efficiency Benchmarking*

Fig. 13(a) and Fig. 13(b) show a comparison of energy-efficiency of FPGA and CPU with different configurations. The improvement can be as large as 2,000x. As $E_{\text{efficiency}} = \frac{ItemNumber}{Energy} = \frac{ItemNumber}{Power*(ItemNumber/Throughput)} = \frac{Throughput}{Power}$, 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. The data in Fig. 13(a) has the same tendency with Fig. 11(a). Fig. 13(b) shows the same principle.



(a)(b)

Fig. 13 Energy efficiency and improvement of the proposed architecture with (a) P = 4, R = 6, K = 2 and (b) P = 2, R = 11, K = 1.

VI. CONCLUSION

Vehicle re-identification is of the most important part in the intelligent transportation system. Recently, dynamic time warping (DTW) with magnetic signature has been a popular method for vehicle re-identification. In this paper we presented a scalable, efficient and parameterized architecture for vehicle re-identification. Particularly, one of the most popular classifier kNN is adopted for vehicle re-identification. 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 automatically 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. In addition, 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.

REFERENCES

[1] Altera. 2014. Quartus II Power Play. (2014). http://www.altera.com.cn/support/software/power/sof-qts-power.html

[2] Cheung, S.Y., Varaiya, P.P., 2007. Traffic Surveillance by Wireless Sensor Networks: Final Report. California PATH Program, Institute of Transportation Studies, University of California at Berkeley.

[3] Yin TIAN , Hong-hui DONG, Li-min JIA , Si-yu LI,2014.A vehicle re-identification algorithm based on multi-sensor correlation.Journal of Zhejiang University-SCIENCE C (Computers & Electronics),11(2014),372-382.

[4] S Charbonnier, A Pitton, A Vassilev,Vehicle re-identification with a single magnetic sensor.Conference Record-ieee Instrumentation & Measurement Technology Conference, 2012, 8443(8):380-385.

[5] Doruk Sart, Abdullah Mueen, Walid Najjar, Eamonn Keogh, and VitNiennattrakul. 2010. Accelerating dynamic time warping subsequence search with GPUs and FPGAs. In ICDM. 1001–1006.

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

[7] Zilong Wang, Sitao Huang, Lanjun Wang, Hao Li, Yu Wang, and Huazhong Yang. 2013. Accelerating subsequence similarity search based on dynamic time warping distance with FPGA. In isFPGA, 2013.

[8] Eamonn Keogh, Xiaopeng Xi, Li Wei, and Chotirat Ann Ratanamahatana. 2006. The UCR time series classification/clustering homepage. URL= http://www. cs. ucr. edu/˜ eamonn/time series data (2006).

[9] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, JesinZakaria, and Eamonn Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 262–270.

[10] Eamonn Keogh and ShrutiKasetty. 2003. On the need for time series data mining benchmarks: a survey and empirical demonstration. Data Mining and knowledge discovery 7, 4 (2003), 349–371.

[11] Yasushi Sakurai, Christos Faloutsos, and Masashi Yamamuro. 2007. Stream monitoring under the time warping distance. In Data Engineering, 2007. ICDE 2007.1046–1055.

[12] Roozbeh Jafari and Reza Lotfian. 2011. A low power wake-up circuitry based on dynamic time warping for body sensor networks. In Body Sensor Networks, 2011 International Conference on. IEEE, 83–88.

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

[14] SM Lucas. 1998. A fast exact parallel implementation of the k-nearest neighbour pattern classifier. In Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on, Vol. 3. IEEE, 1867–1872.

[15] Yao-Jung Yeh, Hui-Ya Li, Wen-Jyi Hwang, and Chiung-Yao Fang. 2007. FPGA implementation of kNN classifier based on wavelet transform and partial distance search. In Image Analysis. Springer, 512–521.

[16] Ioannis Stamoulias and Elias S Manolakos. 2013. Parallel architectures for the kNN classifier–design of soft IP cores and FPGA implementations. ACM Transactions on Embedded Computing Systems (TECS) 13,2 (2013), 22.

[17] Hanaa M Hussain, Khaled Benkrid, and HuseyinSeker. 2012. An adaptive implementation of a dynamically reconfigurable K-nearest neighbour classifier on FPGA. In Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on. IEEE.