High-Throughput DTW accelerator with minimum area in AMD FPGA by HLS

1st Marco Hormigo-Jiménez
Industrial Engineering School
Universidad de Málaga
Málaga, Spain
marcohorjim@uma.es

2nd Javier Hormigo
Dept. Computer Architecture
Universidad de Málaga
Malaga, Spain
ORCID:0000-0002-5454-6821

***Abstract*—Dynamic Time Warping (DTW) is a dynamic programming algorithm that is known to be one of the best methods to measure the similarities between two signals, even if there are variations in the speed of those. It is extensively used in many machine learning algorithms, especially for pattern recognition and classification. U nfortunately, i t h as a q uadratic complexity, which results in very high computational costs. Furthermore, its data dependency made it also very difficult t o parallelize. Special attention has been paid to computing DTW on the edge, as a way to reduce the load of communication on Internet-of-Thing applications. In this work, we propose a minimum area implementation of the DTW algorithm in AMD FPGAs with optimal use of the resources. That is achieved by maximizing the use time of the resources and taking advantage of the inner structure of the AMD FPGAs. This architecture could be used in small devices or as a base for a multi-core implementation with very high-throughput.**

***Index Terms*—Dynamic time warping, FPGA acceleration, HLS implementation, interleaving**

I. Introduction

The Dynamic Time Warping (DTW) algorithm is a technique that allows measuring the similarity between two signals, even if they are not of the same length [1]. This method enables the two signals being compared to be aligned, stretched, or compressed. Hence, DTW allows for identifying patterns and classifying signals with greater precision than the traditional Euclidean distance. As a consequence, it is one of the preferred methods for similarity search in time series mining and signal processing in general [2]. This makes it very useful in applications such as gesture recognition [3], speech recognition [4], or signature recognition [5]. In the last years, special attention has been paid to computing DTW on the edge, as a way to reduce the load of communication on Internet-of-Thing (IoT) applications [6] [7] [2].

The main disadvantage of DTW is its high computation time. That is due to its quadratic complexity and the direct dependencies between the various steps of the algorithm that made very difficult the parallelization of an isolated computation. Furthermore, this problem is exacerbated by the fact that it has to be massively computed in those kinds of applications where it is used. The acceleration of the computation of the DTW has been addressed from software and hardware. Most of the software approaches try to reduce the number of DTWs that need to be completely computed for a specific application. Those software techniques include the reduction of the number of DTWs by pruning based on bounds estimations [8] [9] or early abandon of the computation [10] [11]. However, despite being very effective, DTW computation remains as the task with the greatest portion of the computational time of the application after applying these software techniques. Consequently, it is not surprising that the acceleration of the DTW computation itself has been also intensively studied.

The acceleration of the DTW computation itself has been addressed in conventional accelerators like GPUs [12] [13] [14]. In this case, the acceleration comes through the parallelization of the computation. In [12] and [13], the calculation of a batch of DTWs corresponding to an alignment problem is parallelized by mapping each DTW into different threads. Conversely, in [14], individual DTW computation is parallelized in GPU by using different approximate DTW algorithms that divide the DTW matrix into separate regions calculated in parallel or computing the DTW matrix along diagonals rather than rows.

Specific architectures for DTW computation have been also proposed in both FPGAs and ASIC designs. Most of these specific architectures are based on systolic arrays, mostly two dimensions ones, to maximize the throughput. Sometimes, these two-dimension (2-D) systolic arrays are complemented with sophisticated technology or arithmetic to improve efficiency. For instance, [2] uses an analog memristor-based circuit for computation, [15] uses Time-Domain computing, where signals are encoded and processed with time pulses, and [4] uses online arithmetic where values are coded in signed-digit and processed digit by digit, in a most-significant digit first fashion. These 2-D systolic arrays have to be adapted to the size of the problem which is not always possible in ASIC implementations. That is a lesser issue in FPGA implementations like in [12], where a high-level description tool is used to automatically generate the VHDL description of the 2-D Systolic array, or in [7] where an array of processing elements are arranged in a ring to manage longer signals.

This work was supported in part by MCIN/AEI/10.13039/501100011033 and European Union Next Generation EU/PRTR under Project TED2021-131527B-I00; by the Fondo Europeo de Desarrollo Regional (UMA20-FEDERJA-059); and by AMD™(Xilinx™) University Program.

These previous DTW hardware implementation uses a two-dimension systolic array to maximize the throughput but at the cost of using an enormous amount of area and a lack of flexibility to use the circuit for different problem sizes or implementation platforms. Furthermore, they are pipelined at the processing element level and they need to move a large amount of data through them to work effectively. In this work, we try a different approach to minimize the movement of data by keeping locally the DTW computation. Therefore, we opted for an iterative implementation of the DTW with the goal of minimizing the area of the DTW unit but, at the same time, efficiently using the occupied resources. The latter is achieved by pipelining the unit at the operation level(instead of a PE level) and parallelizing several independent DTW calculations using interleaving. This approach provides much more flexibility because this small unit could be used effectively in very constrained conditions or could be replicated as many times as required to achieve high throughput solutions in high-end FPGAs.

Regarding, the solution where the proposed DTW unit is replicated extensively to obtain high throughput, someone could argue that we ended up having an arrangement very similar to a systolic array with a lot of data movement. However, although it is true that input data need to be moved to the different DTW units and special care has to be taken to do it efficiently, in this case, the amount of data movement is linear with the size of the problem. In contrast, systolic arrays need to move the intermediate results whose amount grows quadratically with the size of the problem.

In this article, we will study the implementation of a minimum area design for AMD FPGAs to accelerate the computation of the DTW algorithm when using the Sakoe-Chiba band [1]. To use resources as effectively as possible, we also consider the typical application where the input signal has to be compared to an array of pattern signals. However, the proposed design would be equally valid for comparing several input signals with one pattern. To minimize the area, we have taken into account the inner structure of the AMD FPGAs, especially for the implementation of shift registers. We have used High-Level Synthesis (HLS) software tools for the design definition. This method greatly simplifies circuit development and design space exploration. In future work, we will use the proposed DTW unit as the core to implement a highly parallel architecture by replicating it to achieve the maximum throughput for a given FPGA platform.

II. The DTW Algorithm

The DTW algorithm measures the distance (or similarity) between two signals (or time series). Let us consider the signal X with n samples and Y with m samples. To compute the DTW(X,Y), the *distance matrix* calculation and the *warping path* calculation can be combined in the same process [2]. The $n \times m$ DTW matrix (W) is computed following the equation (1) for each matrix element:

$$w_{i,j} = min(w_{i,j-1}, w_{i-1,j}, w_{i-1,j-1}) + Dist(x_i, y_j) \quad (1)$$

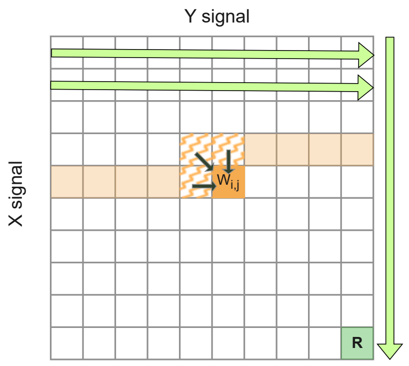

Fig. 1. DTW matrix computation scheme

where *Dist* correspond to a two-point distance measurement and $w_{0,0} = Dist(x_0, y_0)$. If any of the required elements is outside the matrix, its value equals infinite. Hence, the DTW matrix is easily computed starting with $w_{0,0}$ and computing each row in order, from left to right, and from top to bottom, as is shown in Fig. 1. The value of the last-row last-column element ($w_{n-1,m-1}$) is the DTW distance ($DTW(X,Y)$) (the green square "R"+ in Fig. 1). As shown in Fig. 1, to calculate the $w_{i,j}$ element, it is necessary to know the values of the elements highlighted with zigzag lines on the figure. This is what makes it difficult to parallelize the computation of the matrix. Although the DTW matrix is required to extract the *warping path*, most of the applications only use the DTW distance. Therefore, the whole W matrix does not need to be stored, since only the last computed row is required for the actual row computation. Consequently, only a sliding row (colored squares in Fig. 1) has to be temporarily kept in memory for the subsequent calculations.

In the literature, several distance measurement has been proposed for *Dist*, such as Euclidean or Manhattan distance. In our study, we will use the squared Euclidean distance which balances precision and computation cost. However, the proposed architecture could be easily adapted to implement any other distance measurement.

To prevent impractical matchings and accelerate the computation, generally, some constraints are applied to the warping path. In this work, we implemented the Sakoe-Chiba band [1]. This method reduces the elements to be evaluated in a row to a neighborhood of radius R around the main diagonal of the DTW matrix. In this way, only $2R + 1$ elements have to be evaluated on each row.

In a typical application, multiple input signals are compared through DTW distance with one or several patterns. To simplify the description from now on we will consider that an input signal is compared with P patterns. However, the proposed design would be equally valid for comparing several input signals with one pattern. We will also consider that both signal and pattern have the same number of samples (n), but supporting different sizes would be straightforward.

III. Accelerator Design

In this section, we describe the proposed DTW accelerator for AMD FPGAs. We reduce the utilized resources as much as possible while keeping a reasonable throughput. To do that, we compute the DTW matrix elements serially, but in a pipeline datapath. Since the computation of one matrix element depends on the previous ones, several independent DTW computations are combined to keep the pipeline always running. Moreover, shift registers are used to store temporally the previously computed elements, to take advantage of the efficient implementation of shift registers in AMD FPGAs. This low-area accelerator could be used in low-power FPGAs on the edge and IoT devices, or it could be replicated until obtaining the required throughput for cloud servers.

A. Basic Architecture

To calculate the elements of the DTW matrix, we need the previous element in the row and the elements of the previous row. Hence, only a memory with the size of a row is required to keep temporally these values. Similarly to a 2D convolution in image processing, a shift register with that size could keep the values computed in the actual row and the required values of the previous row, while providing easy data synchronization. In this case, since a Sakoe-Chiva band is used, this memory only needs to store $2R + 1$ elements.

Fig. 2 shows the proposed architecture. It consists of two well-differentiated parts, the previous-result memory and the Calculation Unit, where (1) is computed. To get an efficient implementation of the shift register in the FPGA, the previous-result memory is divided into a first register (*Last*), the shift register (*Band*), and a last register (*Output*). In this way, the three previous elements required to compute the next DTW matrix element are accessed in parallel while the shift register keeps only one output.

The Calculation Unit consists of the minimum unit and the distance unit. The Minimum Unit has two comparators and two multiplexers to select the smallest value among the data received from the previous-result memory (*Last*, last value of *Band*, and *Output*). In parallel, the Distance Unit calculates the squared Euclidean distance between x_i and y_j. Since x_i is required in all distance computations corresponding to a row, a register *Rx* temporarily stores the value of x_i, thus avoiding having to access memory multiple times. The entire calculation unit is pipelined to improve circuit performance. However, as we will see in Section III-C, some modifications are required to take full advantage of the pipelined datapath.

B. Processing Flow

On each regular clock cycle, the proposed architecture computes a new element $w_{i,j}$ of the DTW matrix. The register *Last* provides the element $w_{i,j-1}$ computed in the previous cycle. The registers *Band* and *Output* provide the elements of the previous row, $w_{i-1,j}$ and $w_{i-1,j-1}$, respectively. These values are compared in the minimum unit to output the minimum of them. This value is added to $Dist(x_i, y_j)$, computed in the Distance Unit, to get the new element. This new element is

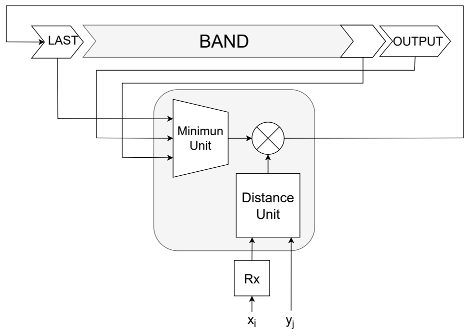

Fig. 2. Architecture diagram

stored in register *Last*, while all values in the previous-result memory are shifted as a whole shift register.

Besides the regular iteration, other special ones are used to provide proper synchronization of the architecture:

- At the beginning of a new DTW computation, all positions of the previous-result memory are filled with infinite.
- Each new row i, *Rx* is loaded with x_i for the distance calculation for all the row elements. Furthermore, the Sakoe-Chiba band has to be shifted one position to fit the matrix diagonal. Hence, in that iteration, the computation is skipped and infinite is loaded in *Last*, and the previous-result memory shifted.
- In the first and last R rows, several elements of the Sakoe-Chiba band are out of the dimension of the DTW matrix. As in the previous case, the computation of wi, j is skipped and the value infinite is used instead to fill the previous-result memory.

All these special iterations are wasting processing cycles. Additional hardware could be added to the proposed architecture to avoid these extra cycles. However, this means more area resources are utilized, and they may even increase the cycle period. Moreover, for the typical size of the signals and the band, these extra cycles are negligible compared with the total amount of cycles required for a complete DTW computation.

C. Throughput Optimization

When we have described the proposed architecture, we have mentioned that the calculation unit is pipelined and a new element of the DTW is computed on each cycle. However, the nature of the algorithm prevents us from starting the computation of a new element every clock cycle, since the value of the previous element in the row is required for calculating the next one. Even utilizing pipeline shortcuts would not be enough to reduce the initiation interval (i.e. the number of cycles between two consecutive calculations in the pipeline) of the datapath to one. To solve this problem, we will use the interleaving technique.

Since we want to calculate the DTW of one input signal with respect to P different patterns, the calculations of the different DTWs corresponding to each pattern could be interleaved. In this way, since these calculations are totally independent,

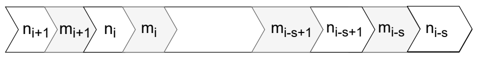

Fig. 3. *Band* register with interleaving

the number of cycles between the calculation of an element and the cycle when this element is required to compute the next one is increased. Thus, we can reduce the initiation interval of the architecture, in as many cycles as different DTW computations are managed in parallel.

If the number of patterns P equals or exceeds the number of pipeline stages of the calculation unit, we will be able to start a new calculation every cycle. In fact, since the HLS syntheses tool is usually able to reduce the initiation interval by applying pipeline shortcuts, P only needs to equal this initiation interval corresponding to the architecture before applying the interleaving technique to achieve this throughput.

On the other hand, if the number of patterns was smaller, the initiation interval would increase, losing performance. In this case, a more effective option than increasing the initiation interval would be to introduce dummy patterns (useless patterns) to complete the required minimum number of patterns to keep the initiation interval set to one. For instance, if $P = 8$ is needed to start a new computation every cycle, but we only have 6, adding two dummy patterns would only reduce the throughput by 25%. However, without the extra dummy patterns, the initiation interval must be three cycles, which means dividing the throughput by three.

Another option to keep the initiation interval set to one when P is not big enough is to reduce the target clock frequency. Doing that, the number of pipeline steps of the calculation unit could be reduced until the initiation interval without interleaving reaches P. Then, a new computation could start every clock cycle by applying interleaving, although the reduction of clock frequency will affect the throughput. Therefore, the effectiveness of each solution must be studied for each specific case. REsults in Section IV could guide this specific study.

Some minor modifications to the proposed architecture are required to implement the interleaving technique. The necessary changes on the datapath affect mostly the previous-result memory. The registers *Last* and *Output* are now register files with one position for each interleaved computation. The same happens with *Rx*, which stores the actual value for each interleaved pattern. In addition, the size of the shift register *Band* is multiplied by the number of interleaved computations. In this way, the data of the calculations of the same element but different patterns will be adjacent to each other, as shown in Fig. 3. Therefore, the implementation of the interleaving computation has a significant cost in memory resources, but it is balanced out by the increases in throughput (see Section IV).

IV. Implementation Results

The proposed DTW accelerator has been described at a high level using C++ for HLS and implemented using AMD Vitis HLS 2022.2 software and targeting a Xilinx Alveo U200 board. A behavioral description of the circuit is provided in C++ and the desired circuit is obtained by using the appropriate directives. That allows us to perform easily a design space exploration and testing of the proposed architecture. For the results shown here, all signals and computations have been performed in single-precision floating-point IEEE-754 standard. Changing the numeric format to represent the signal or to make the computation only involves changing the data type in the C++ description. Moreover, signals and patterns have the same number of samples (*"Size"*).

The proposed circuit has been validated using test vectors generated using Matlab software. The signals were generated randomly, and the golden DTW calculation between two of these signals was performed through the dtw() function located in the Matlab Signal Processing Toolbox. Multiple tests of 1000 trials each have been performed. Each trial included an input signal with multiple patterns, all generated randomly. The radius of the Sakoe-Chiva band, the signal size, and the number of patterns interleaved were varied between each test. The results were considered correct if the solutions generated by Matlab and those produced by our circuit differed by less than 0.01% since Matlab works with double-precision floating-point format.

After validation, different variations of the proposed circuit have been synthesized and implemented using the IP generation tool of Vitis HLS. Following we show the main results of this study. We should note that the results provided here are obtained after placement and routing, and the storage of the input signals are not considered.

Table I shows the results for different signal sizes with and without interleaving for 32 patterns when targeting a clock period of 3 ns. As expected, we can observe that there is not any significant change in the amount of resources utilized when the size of the signal increases. In contrast, the latency increases almost linearly with the size of the signal, whereas the throughput decreases accordingly. These results are consistent with the iterative unit we have designed. Regarding the interleaving technique, it enables us to reduce the initiation interval from 14 to 1, multiplying the throughput by more than 12. This improvement is less than the theoretical 14 times because the clock frequency is slightly reduced due to the increase of resources utilized. As we anticipated, the increment of resources is very significant, although not that much if we consider that we are computing 32 DTWs at the same time. The number of LUTs is the most affected, it increases by more than three times since they are used to implement the shift register (*Band*), and its size is multiplied by 32. The number of registers (FF) and DSPs increases by more than 50% because of the increase of the other registers in the previous-result memory, and also to fulfill the target timing. For long signals, 2 BRAM is also included for the same reasons as before. Despite the area increases, we believe the use of interleaving is advisable in general, since the throughput increases remarkably overcome these costs. We should also note that using 32 patterns is not the best choice for this example as we will see later.

To study the effect of the size of the Sakoe-Chiba band in

TABLE I
IMPLEMENTATION RESULTS FOR DIFFERENT SIGNAL SIZES WITH AND WITHOUT INTERLEAVING

Size	R	Pattern	Period (ns)	Throughput (DTW/ms)	Latency (cycles)	II	LUT	FF	DSP	BRAM
100	16	1	2.107	10.0	47655	14	555	969	5	0
100	16	32	2.346	124.2	109861	1	1870	1569	8	0
500	16	1	2.214	2.0	238059	14	549	1001	5	0
500	16	32	2.346	25.0	545061	1	1870	1569	8	0
2500	16	1	2.025	0.4	1190056	14	552	1015	5	0
2500	16	32	2.546	4.6	2721061	1	1885	1583	8	2
5000	16	1	2.227	0.2	2380056	14	556	1023	5	0
5000	16	32	2.480	2.4	5441061	1	1891	1598	8	2

the architecture, Table II shows the results for different values of R with interleaving for 32 patterns when targeting a clock period of 3 ns. As we could expect, these results show a direct correlation between the parameter R and the latency in cycles, which translates also to the throughput. Regarding the area, the increase of R only affects significantly the number of LUTs, since only the shift register is increased and as we told before, it is implemented by using LUTs.

As we can guess, increasing the number of patterns interleaved beyond the initiation interval of the basic architecture does not increase the throughput, but it increases resource utilization. This is supported by the results in Table III where the number of interleaved patterns are varied keeping the same parameters as in previous tables. Inexplicably, the HLS tool is not able to take advantage of the interleaving technique when the number of patterns is too low. Much more area and less throughput than the basic circuit is obtained when we try to interleave only 4 patterns. In contrast, a speedup of 7 with less than double area is achieved by using only 8 patterns. However, since the initiation interval of the basic circuit is 14, until 16 patterns we do not reach one new computation in every clock cycle. At this point, with 16 patterns, we reach the maximum throughput for the minimum area, except for accidental variations in the clock frequency.

An interesting finding of our study is that when only a few patterns are available to interleave, alleviating the target timing, not only reduces resource utilization but also may increase the throughput. Table IV shows the results for different numbers of patterns and target frequency. Increasing the target period allows having fewer pipeline steps and consequently reduce the Initiation Interval. As a consequence, the reduction of frequency surprisingly increases the throughput whiles reducing the area targeting 5 ns for the basic architecture, 5 ns and 10 ns for 4 patterns, and 7 ns for 8 patterns. This effect stops when the initiation interval reaches one.

Summarizing all the results, the size of the signals will only affect the throughput whereas the band size will affect the throughput and the area. Whenever possible, the interleaving technique with the minimum number of patterns to reach an initiation interval of one should be applied to obtain the maximum throughput with the minimum amount of resource utilization. When the number of patterns available prevents reaching an initiation interval of one, a detailed study of the clock frequency is required to achieve the best results.

We do not want to finish without making a comparison of the proposed design with the state of the art. However, making a fair quantitative comparison is very difficult for several reasons, such as very different target technology, the fast evolution of FPGAs, or disparities in the DTW algorithm implemented. We are going to do a comparison with the accelerator proposed in [7] which seems to be the more similar one, but it is based on a very rough estimation. The systolic array proposed in [7] can process 16-bit input signals with 1024 samples with a throughput of less than 80 complete DTWs per second. It is implemented in a low-power Lattice iCE40 UP5K FPGA using 99% of LUTs and 86% of BRAM. That means roughly 5230 4-input LUTs, 980 Kbits of BRAM, and probably 8 16x16-bit multipliers with 32-bit accumulators. On the other side, we have adapted the proposed circuits for 16-bit input signals with 1024 samples using a band of 1025 elements, which approximates the complete DTW without the band. It also performs interleaving with 8 patterns for improving the throughput. This circuit reaches about 360 DTWs per second and utilizes 4481 6-input LUTs, 504 FF, 1 DSP, and 0 BRAMs. Therefore, the proposed architecture has a throughput 4.5 times higher, using roughly 3.5 more logic resources (if we roughly approximate that 6-input LUT is 4 times more complex than 4-input LUT) and a negligible amount of memory resources compared to the architecture in [7]. We should also note that most of the logic resources used by our proposal (about 4100 LUTs) are utilized to implement the shift registers and those could have been implemented using about 128 Kbits of BRAM. Although an analysis is required for each application, these results seem to show that our architecture utilizes the resources more efficiently.

V. CONCLUSIONS AND FUTURE WORK

In this article, we have proposed an FPGA accelerator for DTW computation that allows us to efficiently use the resources while using a very small area. We achieved this by minimizing the number of partial results that are stored or transmitted, taking advantage of the inner structure of the target FPGA, and pipelining the calculation unit. To make the most of the pipelining, we used the interleaving technique to avoid having idle cycles in the calculator datapath. The use of the HLS design style allows us easy development and rapid design space exploration. A complete study of the different parameters involved in the circuit design has been provided

TABLE II
IMPLEMENTATION RESULTS FOR DIFFERENT BAND SIZE

Size	R	Pattern	Period (ns)	Throughput (DTW/ms)	Latency (cycles)	II	LUT	FF	DSP	BRAM
2500	8	32	2.386	9.3	1440549	1	1378	1581	8	2
2500	16	32	2.546	4.6	2721061	1	1885	1583	8	2
2500	32	32	2.606	2.3	5282085	1	2919	1590	8	2
2500	64	32	2.518	1.2	10404133	1	4972	1587	8	2

TABLE III
IMPLEMENTATION RESULTS FOR DIFFERENT NUMBER OF PATTERNS INTERLEAVED

Size	R	Pattern	Period (ns)	Throughput (DTW/ms)	Latency (cycles)	II	LUT	FF	DSP	BRAM
2500	16	1	2.025	0.4	1190056	14	552	1015	5	0
2500	16	4	2.488	0.3	4760151	14	1229	1681	6	0
2500	16	8	2.848	2.8	1360291	2	1121	1588	8	0
2500	16	16	2.475	4.7	1360549	1	1354	1542	8	2
2500	16	24	2.234	5.3	2040805	1	1689	1567	8	2
2500	16	32	2.546	4.6	2721061	1	1885	1583	8	2

TABLE IV
IMPLEMENTATION RESULTS CHANGING THE CLOCK FREQUENCY

Size	R	Pattern	Target timing (ns)	Period (ns)	Throughput (DTW/ms)	Latency (cycles)	II	LUT	FF	DSP	BRAM
2500	16	1	3	2.025	0.4	1190056	14	552	1015	5	0
2500	16	1	5	3.337	0.5	595049	7	588	813	5	0
2500	16	1	7	5.260	0.4	510046	6	544	772	5	0
2500	16	4	3	2.488	0.3	4760151	14	1229	1681	6	0
2500	16	4	5	3.797	2.2	476145	7	1121	1588	8	0
2500	16	4	7	5.501	0.4	2040143	6	1090	1359	6	0
2500	16	4	10	8.722	1.3	340141	5	933	1119	6	0
2500	16	8	3	2.848	2.8	1360291	2	1121	1588	8	0
2500	16	8	5	3.744	1.6	1360278	2	993	965	6	0
2500	16	8	7	5.170	2.3	680276	1	1092	1106	8	2

to facilitate its fine-tuning for a specific problem. This small architecture can be easily adapted by replication to achieve maximum performance on the target FPGA. Future work will concentrate on designing a parametric multi-core architecture based on this core and minimizing the transmission of data to achieve the maximum possible speed on a specific device.

REFERENCES

[1] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, February 1978.

[2] X. Xu, F. Lin, A. Wang, X. Yao, Q. Lu, W. Xu, Y. Shi, and Y. Hu, "Accelerating dynamic time warping with memristor-based customized fabrics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 4, pp. 729–741, April 2018.

[3] S. S. Jambhale and A. Khaparde, "Gesture recognition using dtw and piecewise dtw," in *2014 International Conference on Electronics and Communication Systems (ICECS)*, Feb 2014, pp. 1–5.

[4] M. Irwin, "A digit pipelined dynamic time warp processor (word recognition)," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 9, pp. 1412–1422, Sep. 1988.

[5] M. Okawa, "Analysis of session variability for online signature verification using local stability-weighted dtw," in *2020 IEEE 9th Global Conference on Consumer Electronics (GCCE)*, Oct 2020, pp. 220–221.

[6] H. Zhou, X. Xu, Y. Hu, G. Yu, Z. Yan, F. Lin, and W. Xu, "Energy-efficient pipelined dtw architecture on hybrid embedded platforms," in *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*, Dec 2015, pp. 1–8.

[7] S. Kang, J. Moon, and S.-W. Jun, "FPGA-accelerated time series mining on low-power IoT devices," in *IEEE 31st Int. Conf. Application-specific Systems, Architectures and Processors (ASAP)*, July 2020, pp. 33–36.

[8] S.-W. Kim, S. Park, and W. W. Chu, "An index-based approach for similarity search supporting time warping in large sequence databases," in *Proc. 17th Int. Conf. on data engineering*. IEEE, 2001, pp. 607–614.

[9] E. Keogh, L. Wei, X. Xi, M. Vlachos, S.-H. Lee, and P. Protopapas, "Supporting exact indexing of arbitrarily rotated shapes and periodic time series under euclidean and warping distance measures," *The VLDB journal*, vol. 18, pp. 611–630, 2009.

[10] J. Li and Y. Wang, "Ea dtw: Early abandon to accelerate exactly warping matching of time series," in *Int. Conf. on Intelligent Systems and Knowledge Engineering 2007*. Atlantis Press, 2007, pp. 1200–1207.

[11] 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," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 262–270.

[12] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul, "Accelerating dynamic time warping subsequence search with gpus and fpgas," in *2010 IEEE Int. Conf. on Data Mining*, 2010, pp. 1001–1006.

[13] C. Hundt, B. Schmidt, and E. Schömer, "Cuda-accelerated alignment of subsequences in streamed time series data," in *2014 43rd International Conference on Parallel Processing*. IEEE, 2014, pp. 10–19.

[14] D. Yang, T. Shaw, and T. Tsai, "A study of parallelizable alternatives to dynamic time warping for aligning long sequences," *IEEE/ACM Trans. on Audio, Speech, and Language Processing*, vol. 30, pp. 2117–2127, 2022.

[15] Z. Chen and J. Gu, "High-throughput dynamic time warping accelerator for time-series classification with pipelined mixed-signal time-domain computing," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 2, pp. 624–635, Feb 2021.