BOTDA Fiber Sensor System Based on FPGA Accelerated Support Vector Regression

Huan Wu†, Hongda Wang†, Chester Shu, *Senior Member, IEEE*, Chiu-Sing Choy*, Senior Member, IEEE*, and Chao Lu, *Fellow, OSA*

† These authors contribute equally to this work

***Abstract*—Brillouin optical time domain analyzer (BOTDA) fiber sensors have shown strong capability in static long haul distributed temperature/strain sensing. However, in applications such as structural health monitoring and leakage detection, real-time measurement is quite necessary. The measurement time of temperature/strain in a BOTDA system includes data acquisition time and post-processing time. In this work, we propose to use hardware accelerated support vector regression (SVR) for the post-processing of the collected BOTDA data. Ideal Lorentzian curves under different temperatures with different linewidths are used to train the SVR model to determine the linear SVR decision function. The performances of SVR is evaluated under different signal-to-noise ratios (SNRs) experimentally. After the model coefficients are determined, algorithm-specific hardware accelerators based on field programmable gate arrays (FPGAs) are used to realize SVR decision function. During the implementation, hardware optimization techniques based on loop dependence analysis and batch processing are proposed to reduce the execution latency. Our FPGA implementations can achieve up to 42× speedup compared with software implementation on an i7-5960x computer. The post-processing time for 96,100 BGSs along 38.44-km FUT is only 0.46 seconds with FPGA board ZCU104, making the post-processing time no longer a limiting factor for dynamic sensing. Moreover, the energy efficiency of our FPGA implementation can reach up to 226.1× higher than software implementation based on CPU.**

***Index Terms*—Distributed optical fiber sensing, Brillouin optical time domain analyzer (BOTDA), digital signal processing, support vector machine (SVM), field programmable gate arrays (FPGA), hardware implementation.**

I. Introduction

DISTRIBUTED optical fiber sensors allow many points to be measured simultaneously, and it is compatible to ubiquitously deployed underground fiber system for telecommunication purpose, therefore, they attract interest from both industry and academic [1-4]. Distributed optical fiber sensors mainly depends on scattering effects in optical fiber, Brillouin optical time domain analyzer (BOTDA) based on stimulated Brillouin scattering (SBS) was invented in 1990 [5], it can measure strain as well as temperature. BOTDA sensors rely on stimulated Brillouin scattering (SBS) of two counter-propagating light waves, a continuous-wave (CW) signal and a pulsed pump. The frequency offset between the pump and probe is scanned around the Brillouin frequency shift (BFS) of the fiber to reconstruct the Brillouin gain spectrum (BGS). Since the change of BFS has a linear relationship with the change of temperature and strain on the fiber, an important operation in a BOTDA system is to find the BFS from the measured BGS to determine temperature or strain information along the fiber under test (FUT). In an ideal BGS, BFS is the shift in peak gain frequency. However, acquired BGSs are always contaminated by noises. Therefore, post-processing algorithms are needed to accurately determine BFS from the measured BGSs. The conventional wisdom to predict the BFS information from the BGS is Levenberg-Marquardt algorithm (LMA) curve fitting [6,7]. However, its complexity is often a limiting factor in the sensing speed of a BOTDA system especially for long sensing distance.

This work was supported by National Natural Science Foundation of China (NSFC) under Grants 61435006, CUHK Group Research Scheme, CUHK Direct Grant 4055090, Research Grants Council of Hong Kong (RGC) project: RGC GRF CUHK 14204918 and PolyU 152658/16.

Huan Wu and Chao Lu are with Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Kowloon, Hong Kong. (email: hkpolyu.wu@poly.edu.hk; chao.lu@polyu.edu.hk)

Hongda Wang, Chiu-Sing Choy and Chester Shu are with the Department of Electronic Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. (email: 1155039965@link.cuhk.edu.hk; cschoy@ee.cuhk.edu.hk; ctshu@cuhk.edu.hk)

In recent years, the performances of BOTDA are improved significantly due to the rapid developments of the technology. The sensing distance of BOTDA can achieve hundreds of kilometers [8], and the spatial resolution can be reduced to millimeter level [9]. Longer sensing distance brings larger amount of sensing data and finer resolution requires higher sampling rate and smaller frequency scanning step which result in denser sensing points. The sensing data volume keeps increasing, which adds the computational load for post-processing. In real scenario, to extract temperature/strain information from the measured BGSs with low latency is quite necessary. Several works have mentioned the challenges of post-processing in real applications. In [10], a non-curve fitting technique called cross-correlation method (XCM) was proposed based on calculation of cross-correlation between an ideal Lorentzian curve and the measured BGS to determine BFS. In [11], a modified version of XCM is implemented on FPGA to speed up the processing time at the cost of sensing accuracy. Artificial neural network (ANN) is proposed for BOTDA system to improve the sensing accuracy and processing speed [12]. However, the training of ANN is difficult due to numerous hyperparameters. Recently, we reported a machine learning method called support vector classification (SVC) to extract temperature information from measured BGSs with simple training strategy and fast processing speed [13-15].

By far, most post-processing methods are implemented on X86-based computers like windows desktop and Linux servers.

These methods normally rely on separate stages including experimental data measurement, data storage, data transmission and data post-processing. Due to the discontinuous workflow, real time post-processing is challenging because additional cost by data storage and transmission cannot be ignored, not to mention the running time of the computation-intensive algorithms. Thus, it is challenging to realize real-time surveillance and monitoring based on software solutions. For the applications which need extreme low latency, hardware solutions are alternatives and can alleviate the difficulties mentioned above [16]. Among the current hardware platforms, FPGA and ASIC are two representatives which can provide sufficient computation capability, moreover, FPGA and ASIC are more compact which means they can be easily integrated into a portable BOTDA system. Compared with ASIC, FPGA has the advantages of reconfigurability and fast deployment time especially with the help of high level synthesis [17]. The computation capability of FPGA can outperform CPU by one order of magnitude through massively parallelizing the algorithm in an efficient manner [18]. However, not all the algorithms can be easily implemented on FPGA because of the inadaptability to fixed hardware structures. LMA curve fitting is an iterative optimization method based on the error gradient estimation, which requires matrix inversion during error correction. From the hardware perspective, the iterative process and matrix inversion are expensive to realize. For cross correlation, the total multiply-accumulation (MAC) operations required is proportional to the square of frequency number in the BGS, thus pose a huge computation burden, therefore, in [11], moving average and referential triangular pulse are adopted to reduce the computational complexity at the expense of BFS accuracy. The ANN proposed in [12] is also not very suitable for efficient hardware implementation since the sigmoid nonlinear activation function in each neuron can only be approximated with a big look-up table or low-order polynomial expansion. As for SVC [13-15], *n*-class SVC is built upon *n*(*n*-1)/2 binary classifiers and each classifier has unique number of support vectors, the irregular computation pattern doesn't fit a fixed hardware structure. What's more, *n* is normally larger than 100 which generates more than 5000 independent binary classifiers, causing heavy storage requirements and computation loads. Considering the convenience and efficiency of algorithm to hardware migration, we propose a novel method called support vector regression (SVR) in our recent conference paper [19]. In this work, an in-depth comparison with other methods regarding the algorithm performances is included. Moreover, multiple optimization strategies with high level synthesis are proposed and proved experimentally. The main contributions of this work are as follows:

1) A temperature prediction method based on SVR is proposed. The experimental results prove that SVR can achieve comparable performance with existing BFS extraction methods like SVC, cross correlation, LMA Lorentzian curve fitting and ANN.

2) Hardware implementations of SVR decision function are realized on two FPGA boards. Optimizations to linear SVR decision function through loop analysis and batch processing are proposed to take advantages of high flexibility and scalability of modern FPGA devices. These optimization methods transform the decision function into matrix-matrix multiplication and matrix-vector multiplication and parallelize these operations by tiling the large matrix into smaller ones.

3) Post-processing time for 96,100 BGSs along 38.44-km FUT can be completed in 0.46 seconds with Xilinx ZCU104 using the proposed hardware optimization techniques. It achieves 42× speedup compared with the software implementation running on an i7-5960x computer. Meanwhile, the 26.5W power consumption of ZCU104 is also much lower than the conventional CPU, making the energy efficiency of our FPGA implementation 221.6× higher than software implementation based on LIBSVM [20].

The paper is organized as follows. Section II describes the principle of SVR and its training process for temperature extraction in a BOTDA system. Section III introduces the experimental setup of BOTDA and evaluates the performance of SVR under different SNRs experimentally. FPGA optimizations and implementations of linear SVR decision function are given in Section IV. Section V concludes this work.

II. Principle of SVR and Training Process for Temperature Extraction in a BOTDA System

Suppose we have training data $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l)\}$, where $\boldsymbol{x}_i \in R^n$ is training sample and $y_i \in R$ is label. In linear case, we construct a linear decision function to fit the training data:

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \tag{2.1}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product, $\boldsymbol{w}$ is the norm vector of the linear function and b is intercept. Traditional linear least-square error regression derives a decision function by minimizing the deviation between predicted value $f(\boldsymbol{x}_i)$ and given value y_i for all training data. Unlike linear least-square error fitting, SVR allows a tolerance degree to errors not greater than ε as shown in Fig. 1(a). Only the data points outside the shaded region contribute to the error and the deviations are penalized in a linear fashion as shown in Fig. 1(b). The goal of SVR is to find a function that fits current training data with a deviation no larger than ε, and at the same time as flat as possible. One way to ensure this is to minimize the norm, i.e., $\|\boldsymbol{w}\|^2 = \langle \boldsymbol{w}, \boldsymbol{w} \rangle$. We can write this problem as a convex optimization problem as follows:

$$\text{minimize: } \frac{1}{2}\|\boldsymbol{w}\|^2$$
$$\text{subject to} \begin{cases} y_i - \langle \boldsymbol{w}, x \rangle - b \le \varepsilon \\ \langle \boldsymbol{w}, x \rangle + b - y_i \le \varepsilon \end{cases} \tag{2.2}$$

The above convex optimization problem is feasible in cases where $f(\boldsymbol{x})$ actually exists and all pairs $(\boldsymbol{x}_i, y_i)$ are within ε precision. However, in most cases, not all $(\boldsymbol{x}_i, y_i)$ are within ε precision, then we can introduce slack variables ξ_i, ξ_i^* to deal with this problem. Hence, we get the following formulation:

$$\text{minimize: } \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$
$$\text{subject to} \begin{cases} y_i - \langle \boldsymbol{w}, \boldsymbol{x} \rangle - b \le \varepsilon + \xi_i \\ \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b - y_i \le \varepsilon + \xi_i^* \end{cases} \tag{2.3}$$

where $\xi_i, \xi_i^* \ge 0$, the constant $C > 0$ determines the trade-off between the flatness of $f(x)$ and the amount up to which deviations larger than ε are tolerated. Equation (2.3) is known as the primal problem of SVR algorithm and it can be transformed to dual problem and solved by quadratic programming [21]. The solution is as follows:

$$\mathbf{w} = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)(\boldsymbol{x}_i) \tag{2.4}$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\langle \boldsymbol{x}_i, \boldsymbol{x}\rangle + b \tag{2.5}$$

where α_i and α_i^* are the dual variables, $\langle \boldsymbol{x}_i, \boldsymbol{x}\rangle$ represents the inner product between training sample $\boldsymbol{x}_i$ and test sample $\boldsymbol{x}$. From Equation (2.5), we can see that once the model parameters are identified, SVR only depends on $\boldsymbol{x}_i$ with corresponding $(\alpha_i - \alpha_i^*)$ which are non-zero, these $\boldsymbol{x}_i$ are called support vectors and they are subsets of training data.

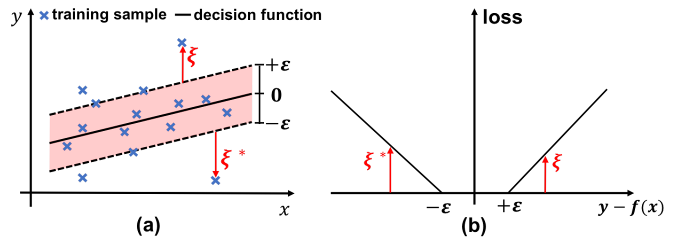

Fig. 1. (a) one-dimensional linear SVR for illustration, (b) linear loss function.

In our case, to process measured BGSs collected from a BOTDA system, a high dimensional linear SVR is used, normalized gain value at every frequency on the BGS forms feature vector $\boldsymbol{x}_i$, and corresponding temperature of the BGS is label y_i. The use of SVR includes two phases, the training phase and testing phase as shown in Fig. 2. During the training phase, the simulated ideal BGSs together with the corresponding temperature labels serving as the training samples are used to get linear decision function for temperature prediction. We design the simulated ideal BGSs by using ideal Lorentzian curve as the gain profile for the training of SVR:

$$\mathrm{g}(\nu) = \frac{g_B}{1+\left[\frac{(\nu-\nu_B)}{\Delta\nu_B/2}\right]^2} \tag{2.6}$$

where g_B, ν_B and $\Delta\nu_B$ are the peak gain, BFS and bandwidth of the BGS. Peak gain is set as 1, BFSs of the ideal BGSs from a temperature range of 0℃ to 70℃ with 0.5℃ step are determined using the temperature coefficient (0.975MHz/℃) of the fiber under test (FUT). It should be mentioned that the temperature range is determined by the application, in our case, the room temperature is around 20℃ and part of the fiber is heated to 50℃, therefore, we set the temperature range in [0,70]. The bandwidth of the BGS is determined by the pulse width / spatial resolution, to accommodate pulse width from minimum 10ns to infinity (continuous wave), we set the bandwidth of the ideal BGSs from 30MHz to 100MHz at a step of 2MHz. Finally, we have 141 × 36 ideal BGSs to train the SVR. The frequency range of ν is from 10.78GHz to 11.0GHz with 1MHz step, therefore, we have 220 frequencies. After training, we get 1,136 support vectors in the SVR model. In the testing phase, the fixed model predicts a continuous temperature value for each normalized measured BGS collected from a BOTDA system.

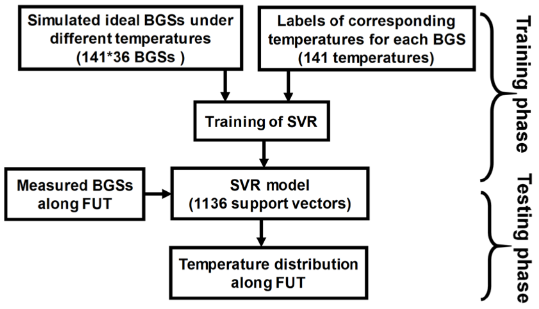
Simulated ideal BGSs under different temperatures (141*36 BGSs)
Labels of corresponding temperatures for each BGS (141 temperatures)
Training of SVR
Measured BGSs along FUT
SVR model (1136 support vectors)
Temperature distribution along FUT
Training phase
Testing phase

Fig. 2. Training and testing phase of SVR.

III. BOTDA Setup and Experimental Results

A. BOTDA Experimental Setup

The experimental setup of the BOTDA system is shown in Fig. 3. The output of a tunable laser source is set around 1550nm and is split into two branches using a coupler. The CW light in the upper branch is modulated by a Mach-Zehnder modulator (MZM) driven by a pulse pattern generator (PPG) to generate optical pump pulses. The bias controller after MZM is to stabilize the applied voltage. The pump is then amplified by an erbium-doped fiber amplifier (EDFA) and passes through a polarization scrambler (PS) to eliminate polarization dependent noise. In the lower branch, another high extinction ratio MZM is driven by a radio frequency (RF) generator. The bias controller is biased at Null point to generate a carrier suppressed double-sideband probe signal. An optical attenuator (ATT) is used to control the probe power followed by an isolator to block the signal from the pump branch. The probe signal is detected by a photodetector (PD) after the lower-frequency probe sideband is selected by using a fiber Bragg grating (FBG) filter. Local BGSs are reconstructed with RF scanned around the BFS of FUT.

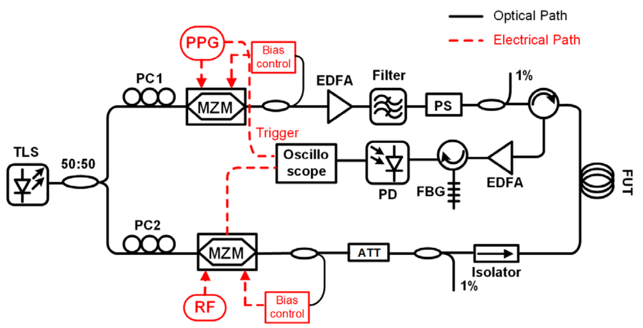

Fig. 3. BOTDA experimental setup. TLS: tunable laser source, PC: polarization controller, PPG: pulse pattern generator, RF: radio frequency, PS: polarization scrambler, MZM: Mach-Zehnder modulator, ATT: attenuator, FUT: fiber under test, FBG: fiber-Bragg grating, PD: photodetector.

Ensemble average is commonly used to increase SNR since signal is at least partially reproducible while noise is random from one measurement to the next. Conventional BOTDA systems widely adopt the ensemble average at the cost of longer data acquisition time. Even though it greatly improves the signal quality, it limits the BOTDA systems for dynamic sensing. Both experimental and algorithmic methods have been proposed to reduce the number of ensemble average [22,23].

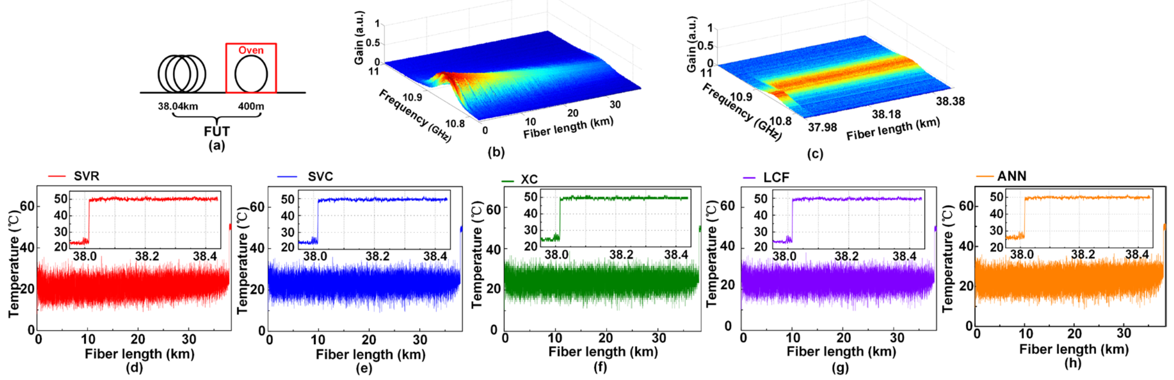

Fig. 4. (a) 38.44-km FUT with last 400m heated to 50℃. (b) Measured BGS distribution along FUT. (c) zoom-in view of the last heated section. Temperature distribution along FUT determined by (d) SVR, (e) SVC, (f) XC, (g) LCF and (h) ANN.

Table I Performances of SVR, SVC, XC, LCF and ANN Under Different SNRs

SNR (dB)	MT (s)	SVR			SVC			XC			LCF			ANN		
		STD	RMSE	MAE	STD	RMSE	MAE	STD	RMSE	MAE	STD	RMSE	MAE	STD	RMSE	MAE
4.5	2.7	2.087	2.343	1.89	1.852	1.861	1.479	2.021	2.339	1.852	1.964	1.985	1.605	2.12	2.133	1.701
6	5.4	1.578	1.584	1.264	1.556	1.562	1.241	1.655	1.783	1.432	1.563	1.61	1.276	1.783	1.789	1.439
8	17	1.262	1.295	1.037	1.127	1.161	0.919	1.225	1.29	0.996	1.122	1.159	0.928	1.321	1.357	1.078
10	33.8	0.785	0.803	0.634	0.73	0.86	0.661	0.783	0.878	0.694	0.732	0.864	0.701	0.858	0.917	0.74
12	88	0.582	0.588	0.469	0.537	0.749	0.596	0.619	0.645	0.472	0.527	0.639	0.514	0.568	0.717	0.583

Another limit for BOTDA dynamic sensing is post-processing of collected BOTDA data. Total temperature/strain sensing time of the BOTDA system can be expressed as follows:

$$T = T_{acq} + T_{pp} = \left(T_c \cdot N_{avg} + T_s\right)N_{freq} + T_{pp} \qquad (3.1)$$

where T_{acq} is the data acquisition time and T_{pp} is post-processing time, $T_c = 2nL/c$ is time of flight, L is the length of FUT, n is the refractive index of the fiber and c is light speed in vacuum. N_{avg} is the number of ensemble average, T_s is the frequency switching time of RF which is around hundreds of milliseconds and N_{freq} is the number of scanned frequencies.

B. Experimental Results

To evaluate the performance of SVR, we use the BOTDA setup in Fig. 3 to measure the BGS distribution along 38.44-km FUT. The last 400-m section of FUT is free from strain and put in a temperature oven heated to 50℃ as shown in Fig. 4(a). The sampling rate is 250MSample/s, corresponding to 96,100 sampling points for 38.44-km FUT. Fig. 4(b) shows the BGSs distribution measured with 20ns pump pulse, 1024 times averaging, and the sweeping frequency is from 10.78GHz to 11.0GHz with 1MHz frequency step. Fig. 4(c) is the zoom-in view of the last heated section.

Next, the measured BGSs are processed by SVR. For comparison, we also process the BGSs by SVC, cross correlation (XC), LMA Lorentzian curve fitting (LCF), and ANN, respectively, as shown in Fig. 4(d)-(h). The insets in Fig. 4 (d)-(h) depict the zoom-in view at the heated section. We can see that the temperature information along FUT have been successfully extracted by these algorithms. Then we investigate the performances of these algorithms under different SNRs, the pump pulse is fixed at 20ns and frequency scanning step at 1MHz. SNR is defined as the ratio between the mean amplitude of Brillouin peak and its standard deviation. We collect the BGSs from 4.5dB to 12dB by using 32 to 1024 times of averaging. According to Equation (3.1), theoretical measurement time varies from 2.7 seconds to 88 seconds when averaging number increases from 32 to 1024. The performances of these algorithms are evaluated by three merits, standard deviation (STD), root mean square error (RMSE) and mean absolute error (MAE). The performances of the five algorithms under different SNRs are shown in Table I, we can see that lower STD, RMSE and MAE can be achieved with higher SNR for all these methods at the cost of longer measurement time (MT). However, there is not a single method exhibiting overwhelming performances over others for all SNRs. For example, when SNR is 8dB, LCF achieves lowest uncertainty, however, at 4.5dB, SVC shows lowest uncertainty. Another phenomenon is that STD, RMSE and MAE may not be consistent. For instance, at 12dB, LCF has lowest uncertainty while SVR has lowest RMSE and MAE, it is because true temperature is involved in calculating RMSE and MAE. However, the true temperature is not strictly 50℃ at any fiber location in the oven at any time, the uniformity of the oven we use is about 0.5℃, which may introduce systematic error in this process. Considering the systematic error and random error in data acquisition, the performances of the five algorithms are comparable. Overall, to enhance the performances of BOTDA fiber sensors, the key is to improve the signal quality since the improvement margin by different BFS extraction methods are quite limited.

Even though various methods can be used to extract BFS information, as analyzed previously, the adaptability of these algorithms to hardware structures are different. SVR predicts the result by regular matrix-vector multiplication and inner-product as shown in Equation (2.5), which is very suitable to be parallelized and pipelined from the hardware perspective. With a dedicated FPGA accelerator, the processing speed of linear SVR can be significantly improved.

IV. FPGA Optimizations and Implementations of SVR

In this section, a hardware architecture for the linear SVR decision function is presented. In the following subsections, part A introduces the direct implementation of linear SVR decision function and discusses its drawbacks. In part B, optimizations to the direct implementation by loop analysis are proposed to reduce the latency. In part C, batch processing method is proposed to further speed up the running time. In part D, 96,100 measured BGSs from 38.44-km FUT are processed by two FPGA boards, experimental results and comparison with software implementation and other works are described. In part E, we give an in-depth theoretical analysis and discussion for FPGA acceleration with the proposed optimization techniques.

A. Direct Implementation of Linear SVR Decision Function

If we simplify $(\alpha_i - \alpha_i^*)$ in decision function Equation (2.5) as β_i and expand the inner product to a sum-of-product term, then we can have the reformulated decision function as follows:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N_s} \beta_i \sum_{j=1}^{M} SV_{ij} \cdot x_j + b \qquad (4.1)$$

where SV represents support vectors obtained from the training process, N_s is the number of support vectors and is 1136 as given in Section II, M is the dimension of input feature vector and is equal to 220. The data path of Equation (4.1) can be illustrated in Fig. 5 and the corresponding pseudocode is shown in Algorithm 1. In Fig. 5, multiply-accumulate (MAC) 1 corresponds to the inner summation of Equation (4.1) and is denoted as partial sum, while MAC 2 corresponds to the outer summation and is denoted as final sum. The total MAC operations in MAC 1 and MAC 2 are $(N_s \cdot M + N_s)N_{BGS}$, where N_{BGS} is the number of BGSs. To process 96,100 measured BGSs from 38.44-km FUT, about 2.4×10^{10} multiplications and 2.4×10^{10} summations are needed, resulting in a heavy computation burden for real-time processing.

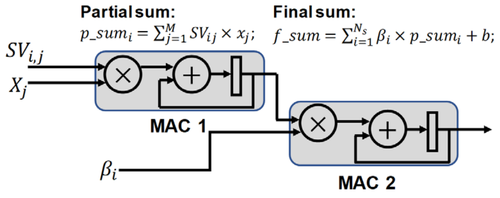
Partial sum:
$p\_sum_i = \sum_{j=1}^{M} SV_{ij} \times x_j;$
Final sum:
$f\_sum = \sum_{i=1}^{N_s} \beta_i \times p\_sum_i + b;$
$SV_{i,j}$
X_j
MAC 1
β_i
MAC 2

Fig. 5. Data path of linear SVR, note that the bias term b in equation 4.1 is omitted since it is used to initialize the register in MAC 2.

Algorithm 1: Original linear SVR without optimization

```
Input: feature vector $x[M]$
Require: support vectors $SV[N_s][M]$, support vector corresponding multipliers $\beta[N_s]$, bias
Output: regression result $f(x)$
Initialize: $p\_sum[N_s] \leftarrow 0, f\_sum \leftarrow bias$
L1: for i=0 to $N_s - 1$ do
    L2: for j=0 to $M - 1$ do
        $square \leftarrow SV[i][j] * x[j]$;
        $p\_sum[i] \leftarrow p\_sum[i] + square$;
    end for
    $temp[i] \leftarrow \beta[i] * p\_sum[i]$;
    $f\_sum \leftarrow f\_sum + temp[i]$;
end for
$f(x) \leftarrow f\_sum$;
```

In hardware design, parallel and pipeline are two common techniques to improve the performance. However, the loop-carried dependence in the inner loop L2 in Algorithm 1 causes long pipeline initiation interval and inefficient hardware utilization efficiency. Moreover, due to the existed dependence, parallelism of this direct implementation cannot be achieved without restructuring the code, thus the total latency is heavily restricted. To accelerate the decision function and enable real-time processing, optimizations must be performed to overcome the limitations.

B. Loop Dependence Analysis and Optimizations

To remove the loop-carried dependence and parallelize the partial sum computation, firstly, we need to perform loop dependence analysis [24]. In Algorithm 1, the statements inside L2 exhibit inter-dependence with respect to the iterator j, but show no inter-dependence on iterator i. Thus, we seek to change the execution order of L1 and L2 to remove the inter-dependence. However, the nested loop is imperfect (perfect nested loops mean the statements only exist inside the innermost loop), we need to take a two-step optimization.

- **Loop distribution:** We find that the statements inside L2 do not depend on the statements between L1 and L2, this means we can safely break loop L1 and distribute the statements between L1 and L2 outside. After loop distribution, a new loop L3 is formed which is only responsible for the accumulation of final sum, while L1 and L2 become a perfect nested loop and calculates the partial sum.
- **Loop interchange:** In the perfect nested loop L1- L2, loop-carried dependence prevents efficient pipeline strategy to be applied because of the long execution latency of the accumulator. The pipeline initiation interval is restricted by the propagation delay of the adder, which is normally larger than one clock cycle for floating point numbers. When working in higher frequency, the propagation delay could further consume more clock cycles, resulting in longer pipeline initiation interval. In Algorithm 1, no inter-dependence is observed between the statements inside L2 and the iterator i, therefore, we can interchange L1 and L2 to remove the dependence and make the nested loop executed consecutively in each clock cycle. After loop interchange, the partial sum is read and write simultaneously with no conflict on the access addresses, which indicates that the partial sum should be mapped to the dual port RAM on FPGA.

Algorithm 2: Optimized linear SVR with loop distribution and interchange

```
Input: feature vector $x[M]$
Require: support vectors $SV[M][N_s]$, support vector corresponding multipliers $\beta[N_s]$, bias
Output: regression result $f(x)$
Initialize: $p\_sum[N_s] \leftarrow 0, f\_sum \leftarrow bias$
L1: for i=0 to $M - 1$ do
    L2: for j=0 to $N_s - 1$ do                    ◁ loop unroll
        $square \leftarrow SV[i][j] * x[i]$;
        $p\_sum[j] \leftarrow p\_sum[j] + square$;
    end for
end for
L3: for i=0 to $N_s - 1$ do                         ◁ loop unroll
    $temp[i] \leftarrow \beta[i] * p\_sum[i]$;
    $f\_sum \leftarrow f\_sum + temp[i]$;
end for
$f(x) \leftarrow f\_sum$;
```

The pseudocode after loop distribution and loop interchange is in shown Algorithm 2. Since the execution order of L1 and L2 is changed, the support vector matrix also needs to be transposed accordingly. The total execution latency in clock cycles can be expressed as follows:

$$\text{Latency} = N_s \cdot M + N_s \cdot T_a \tag{4.2}$$

where T_a is the propagation delay of the floating point adder in clock cycles. Since the loop-carried dependence is removed by loop optimizations, the operations including memory read, compute and memory write are fully pipelined with initiation interval of 1 clock cycle. Thus, no additional memory access cost is involved in Equation (4.2). Only the total number of loop iterations contributes to the latency of the nested loop. For loop L3, the pipeline initiation interval is limited by the loop-carried dependence of the accumulator, so the latency $N_s \cdot T_a$ depends on both the number of loop iterations and the propagation delay of the accumulator in cycles. Compared with the original implementation in Algorithm 1, our proposed loop optimization method in Algorithm 2 has achieved a speedup of 7.75× at the same DSP utilization rate, demonstrating a great improvement of the pipeline efficiency.

Parallelization is another advantage after eliminating loop-carried dependence by loop distribution and interchange. In Algorithm 2 we know that $p\_sum[j]$ and $p\_sum[j+1]$ are calculated independently, thus we can unroll the loop L2 directly to increase the parallelism without changing the code structure. After unrolling, massive parallelized MAC units can be mapped to DSP slices on FPGA easily. Meanwhile, same level of parallelism can also be applied to L3 with a cascaded MAC structure to accelerate the accumulation. Assume we unroll L2 and L3 with a factor of *f* simultaneously as indicated in Algorithm 2, the total latency can be calculated as follows:

$$\text{Latency} = \underbrace{\frac{N_s \cdot M}{f}}_{\text{Partial sum}} + \underbrace{f \cdot T_a + \frac{N_s}{f} + L_{tree}(f)}_{\text{Final sum}} \tag{4.3}$$

$$L_{tree}(f) \approx \begin{cases} T_a \left\lceil \log_2 \frac{N_s}{f} \right\rceil, & f > \frac{N_s}{2f} \\ T_a \left\lceil \log_2 \frac{N_s}{f} \right\rceil + 2\left\lceil \frac{N_s}{2f^2} \right\rceil - 2, & 1 < f < \frac{N_s}{2f} \end{cases}$$

where $L_{tree}(f)$ is the latency of the adder tree inside L3 after unrolling. $L_{tree}(f)$ has different expressions with small and large unroll factors, but in both cases it has little effect on total latency, therefore it can be dropped safely in later analysis. The corresponding hardware structure for calculating the partial sum and final sum are shown in Fig. 7 and Fig. 8. According to Fig. 7, the latency of partial sum after unrolling is inversely proportional to the unroll factor *f,* since the parallel MAC array processes the multiply-accumulate operations concurrently. In Fig. 8, the latency of the final sum after unrolling consists of three terms, i.e., latency of the cascaded MAC array, latency of feeding each partial sum to the MAC array and latency of the optimized adder tree, which are equal to $f \cdot T_a$, N_s/f and $L_{tree}(f)$, respectively. The hardware structures will be further discussed in detail in next part. Note that the latency for $f = 1$ is calculated separately as Equation (4.2). To study the effect of parallelization, we apply different unroll factors to loop L2 and L3 in Algorithm 2. The target platform is Xilinx ZCU104 and the working frequency is set to 200 MHz. All the input signals and intermediate values use single-precision floating point data type. The execution latency and speedup factor are collected from Vivado HLS synthesis report, shown in Fig. 6(a). We can see that the latency for one regression decreases fast as the unroll factor increases, and the speedup almost scales linearly when the unroll factors are relatively small (≤ 36). But if we further increase the unroll factor, the linear scaling does not hold and the acceleration effect is weakened. When the unroll factor increases to 142, the real speedup is about 92×. This can be explained by the following equation:

$$\text{Latency} \approx \begin{cases} \frac{N_s \cdot M}{f}, & \text{for small } f \\ \frac{N_s \cdot M}{f} + f \cdot T_a + \frac{N_s}{f}, & \text{for large } f \end{cases} \tag{4.4}$$

For small unroll factor, the latency for final sum calculation in Equation (4.3) is negligible compared with partial sum, therefore the latency is approximately inversely proportional to *f*. For large unroll factor, the latency of the adder chain within L3 is comparable to that of the partial sum, so the linear scaling does not hold anymore. The hardware consumption is shown in Fig. 6(b), we can see the DSP consumption scales linearly with the unroll factor, while the block-RAM (BRAM) consumption doesn't change much because the support vectors dominate most of the BRAM usage. The look-up table (LUT) and flip-flop (FF) consumptions are also proportional to the unroll factor. The results prove that area-performance trade-off can be easily achieved with the proposed optimization method.

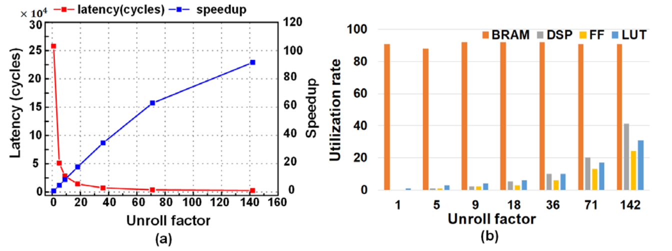

Fig. 6. (a) Speedup and latency versus unroll factor, (b) hardware utilization rate on ZCU104 versus unroll factor.

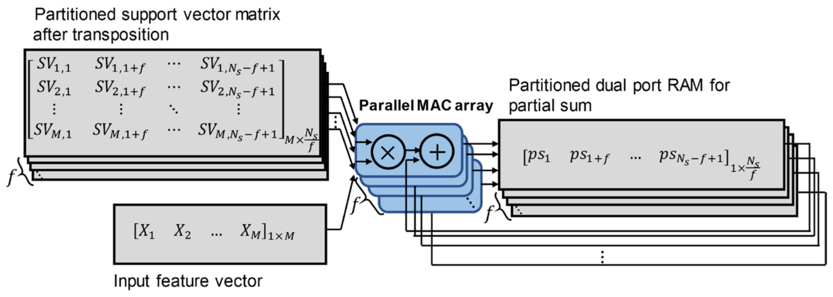
Partitioned support vector matrix after transposition
$SV_{1,1}$ $SV_{1,1+f}$ ⋯ SV_{1,N_s-f+1}
$SV_{2,1}$ $SV_{2,1+f}$ ⋯ SV_{2,N_s-f+1}
$SV_{M,1}$ $SV_{M,1+f}$ ⋯ SV_{M,N_s-f+1}
$M \times \frac{N_s}{f}$
f
Parallel MAC array
Partitioned dual port RAM for partial sum
$[ps_1 \quad ps_{1+f} \quad \dots \quad ps_{N_s-f+1}]_{1\times\frac{N_s}{f}}$
$[X_1 \quad X_2 \quad \dots \quad X_M]_{1\times M}$
Input feature vector

Fig. 7. Hardware structure for calculating the partial sum. The memory for support vector matrix and partial sum is partitioned to *f* individual blocks, respectively. Meanwhile, *f* parallel MAC units keep updating the values of the partial sum iteratively.

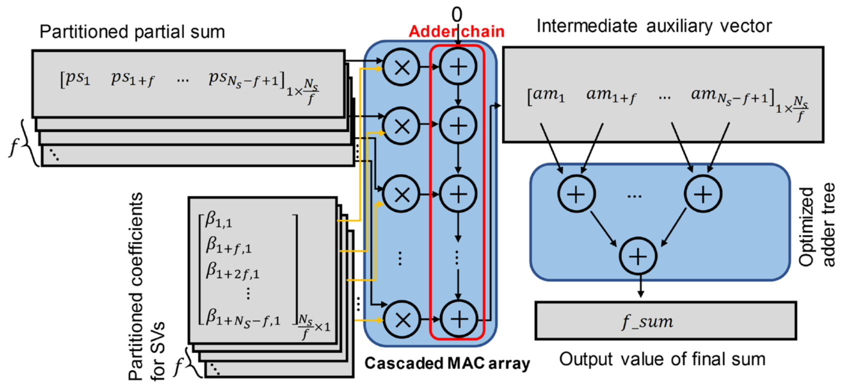
Partitioned partial sum
Adder chain
0
Intermediate auxiliary vector
Optimized adder tree
Partitioned coefficients for SVs
Cascaded MAC array
Output value of final sum
f_sum
f

Fig. 8. Hardware structure for calculating the final sum. The coefficients for SVs are also partitioned to *f* blocks as partial sum to accommodate the parallel access of the cascaded MAC array. The output of the adder chain is written to the intermediate auxiliary vector and accumulated through an optimized adder tree to generate the final sum.

C. Batch Processing Method

From the experimental results in Fig. 6(a), we know that the latency can be greatly reduced with the proposed two-step optimization method and loop unroll, thus a notable speedup can be achieved. However, the linear scaling relationship is not valid for large unroll factors. If we want to achieve high parallelism with a large unroll factor, the latency of $f \cdot T_a$ of the long adder chain becomes prominent, since it is proportional to the unroll factor *f*. Under this circumstance, very deep pipeline stages of the adder chain in L3 will cause the MAC units under-utilized.

Batch processing method is common for GPU based acceleration, especially when used in the training of deep neural networks (DNN) [25]. It can take advantage of the available on-chip memory to store multiple inputs and process them simultaneously in the pipeline. The hardware utilization efficiency can be significantly improved since the arithmetic units are no longer waiting for new inputs in idle state. In [26], the authors further extended this idea to FPGA based deep neural network (DNN) accelerator, and achieved an order of magnitude improvement compared to existing methods.

To further improve the hardware utilization efficiency of L3 with large unroll factor, for the first time, we propose a batch processing method for SVR to process a batch of input vectors at a time. With batch processing, the nested loop L1 and L2 in Algorithm 2 become a three-level nested loop L1, L2 and L3, while the original L3 loop turns into a nested loop L4 and L5. The pseudocode for batch processing is shown in Algorithm 3. Note that no change to the existing computation structure is required for batch processing, more on-chip memory for storing batch of intermediate signals is the additional overhead. The total latency of Algorithm 3 with a large unroll factor f can be calculated as follows:

$$\text{Total latency} \approx \underbrace{\frac{B \cdot N_S \cdot M}{f}}_{\text{Partial sum}} + \underbrace{f \cdot T_a + \frac{B \cdot N_S}{f}}_{\text{Final sum}} \quad (4.5)$$

where B is the batch size. If we divide the total latency by B, the average latency of the adder chain inside L3 is now shared by B inputs:

$$\text{Average Latency} \approx \frac{N_S \cdot M}{f} + \frac{f \cdot T_a}{B} + \frac{N_S}{f} \quad (4.6)$$

When B increases, the average latency of the adder chain will decrease and finally we can have the approximate average latency as follows when B is large enough:

$$\text{Average Latency} \approx \frac{N_S(M+1)}{f} \quad (4.7)$$

In Equation (4.7), we can see the latency is only dependent on the unroll factor *f*, which exhibits an inversely proportional relationship and the linear scaling of speedup holds.

Algorithm 3: Optimized linear SVR with loop distribution, loop interchange and batch processing

```
Input: multiple feature vectors $x[B][M]$
Require: support vectors $SV[M][N_s]$, support vector corresponding multipliers $\beta[N_s]$, $bias$
Output: classification results $f(x[B])$
Initialize: $p\_sum[N_s] \leftarrow 0$, $f\_sum[B] \leftarrow bias$
L1: for $k$=0 to $B-1$ do
    L2: for $i$=0 to $M-1$ do
        L3: for $j$=0 to $N_s-1$ do                ◁ loop unroll
            $square[k] \leftarrow SV[i][j] * x[k][i]$;
            $p\_sum[k][j] \leftarrow p\_sum[k][j] + square[k]$;
        end for
    end for
end for
L4: for $k$=0 to $B-1$ do
    L5: for $i$=0 to $N_s-1$ do                    ◁ loop unroll
        $temp[k][i] \leftarrow \beta[i] * p\_sum[k][i]$;
        $f\_sum[k] \leftarrow f\_sum[k] + temp[k][i]$;
    end for
end for
$f(x[B]) \leftarrow f\_sum[B]$;
```

The hardware structure for calculating the three-level nested loop L1-L2-L3 in Algorithm 3 is shown in Fig. 9. Compared with Fig. 7, a larger memory is required to store the batch of input feature vectors and partial sum matrix, while the parallel MAC array remains the same. To enable multiple access to the support vector matrix, array partition is performed to increase the memory bandwidth and the partition factor is equal to the unroll factor *f*. Moreover, the partitioned partial sum matrix is mapped to the dual port RAM to enable simultaneous read and write operations. In every clock cycle, *f* support vectors and one element from input vectors are read to the parallel MAC array, the accumulation results are written to the dual port RAM concurrently. It takes totally $B \cdot N_s \cdot M/f$ cycles to finish updating the partial sum matrix. After this, the partial sum matrix will be used to calculate the final sum.

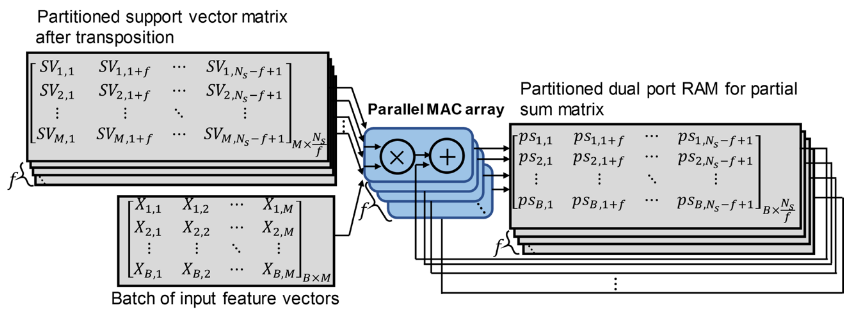
Partitioned support vector matrix after transposition
Parallel MAC array
Partitioned dual port RAM for partial sum matrix
Batch of input feature vectors

Fig. 9. Hardware structure for calculating the partial sum matrix with batch processing. The memory for support vector matrix and partial sum matrix are partitioned to *f* individual blocks, respectively. Meanwhile, *f* parallel MAC units keep updating the values of the partial sum iteratively.

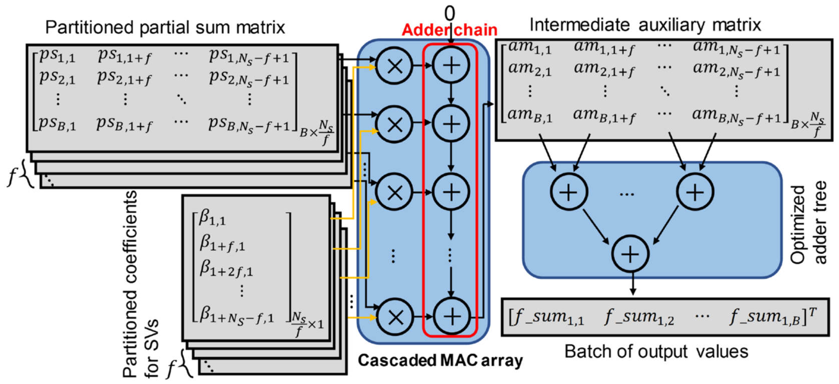
Partitioned partial sum matrix
Adder chain
Intermediate auxiliary matrix
Partitioned coefficients for SVs
Optimized adder tree
Cascaded MAC array
Batch of output values

Fig. 10. Hardware structure for calculating the final outputs with batch processing. The coefficients for SVs are also partitioned to *f* blocks as partial sum matrix to accommodate the parallel access of the cascaded MAC array. The output of the adder chain is written to the intermediate auxiliary matrix and accumulated through an optimized adder tree to generate the final output values.

The hardware structure of nested loop L4-L5 is presented in Fig. 10, which increases the memory consumption for intermediate auxiliary matrix on the basis of Fig. 8. Different from the parallel MAC array in Fig. 9, the massive MAC units are reconstructed to a cascaded MAC array. In every clock cycle, f elements from partial sum matrix and coefficients vector are fetched to the MAC array, while only one output is generated to the intermediate auxiliary matrix at a time. The long adder chain inside the MAC array is heavily pipelined to ensure the initiation interval of 1 clock cycle. It takes $B \cdot N_S/f$ cycles to feed all the inputs to the MAC array, however, the latency of the adder chain is not negligible since it is directly proportional to the unroll factor *f*. After the intermediate auxiliary matrix is completely updated, an optimized adder tree will generate the final outputs in serial, the time consumption $B \cdot L_{tree}(f)$ of this adder tree is trivial since N_S/f is normally very small for large unroll factors.

To verify the effectiveness of the proposed batch processing method, we apply different batch sizes on Algorithm 3. The unroll factor of 284 is chosen to maximize the use of the available DSP resources on ZCU104. The latency and speedup versus batch size is depicted in Fig. 11(a). We can see that the latency decreases rapidly along with the increase of batch size, and finally converges to about 900 clock cycles. Meanwhile, the speedup increases along with the batch size, and the maximum speedup achieved is 275× with batch size of 40. The hardware utilization is shown in Fig. 11(b). We can see that the DSP, BRAM and FF usage does not change much when the batch size increases. Only the LUT consumption slightly increases since the storage requirement for intermediate values like partial sum matrix and intermediate auxiliary matrix is proportional to batch size. The overall hardware utilization for large batch size does not impose heavy burden to the resources, which proves our proposed batch processing method is also area efficient for hardware implementation.

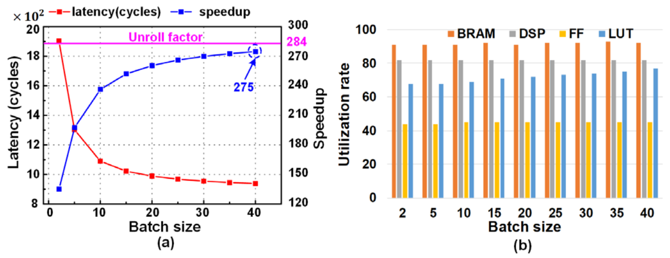

Fig. 11. (a) Speedup and latency versus batch size, (b) hardware utilization rate on ZCU104 versus batch size.

D. *Implementation Results on ZC706 and ZCU104*

Next, we implement linear SVR decision function on two different FPGA platforms based on the proposed optimization methods. Two FPGA boards are Xilinx ZC706 and ZCU104 as shown in Fig. 12, the corresponding chips are Xilinx Zynq XC7Z045 and Zynq UltraScale+ ZU7EV, respectively. The post-implementation resource utilization is shown in Table II. It can be observed that the resources are used adequately for both platforms, while DSP and BRAM resources are the main constraints since they determine the maximum parallelism degree._The performances of two FPGA boards are shown in Table III, which also includes a software implementation based on widely used LIBSVM running on a windows desktop with i7-5960x [27] CPU and 32 GB RAM. From Table III, we can see that the software implementation with LIBSVM needs 19.41 seconds for the post-processing of 96,100 BGSs from 38.44-km FUT when it works at 3GHz, taking up 18~87.8% of total measurement time. On the contrast, our implementation with ZC706 can complete the post-processing in 1.98 seconds, while the power consumption of the FPGA development board is only 14.43W when it works at 100MHz, taking up 2.2~42.3% of measurement time. Furthermore, the implementation with ZCU104 completes the post-processing in 0.46 seconds when it works at 200MHz, taking up 0.52~14.5% of measurement time. The power consumption is 26.5W. The working frequency difference between ZC706 and ZCU104 is due to the different manufacturing technology by the two FPGAs, and advanced technology can enable higher working frequency. The equivalent performance of the three platforms are 2.48GFLOPS, 24.3GFLOPS and 104GFLOPS, respectively. Since ZCU104 provides 1.92× DSP resources over ZC706, the unroll factor for ZCU104 can double compared with ZC706. The results prove that the hardware accelerators can achieve real-time post-processing for the BOTDA data, which are 9.8× and 42× faster than the software implementation. Meanwhile, we also evaluate the energy efficiency as Equation (4.8), where $\text{energy} = \text{power} \times \text{running time}$. The two FPGA implementations achieve 95.1× and 226.1× energy efficiency compared with i7-5960x, which could save plenty of energy in all-day monitoring environments.

$$\text{Energy efficiency} = \frac{\text{Energy consumed by the target accelerator}}{\text{Energy consumed by CPU}} \tag{4.8}$$

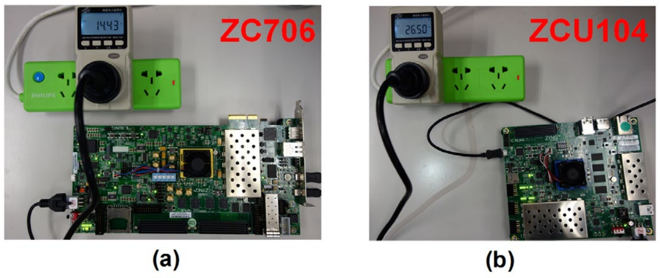

Fig. 12. FPGA boards of (a) Xilinx ZC706, (b) ZCU104.

TABLE II Post-implementation Resource Utilization of ZC706 and ZCU104

	Xilinx ZC706			Xilinx ZCU104		
	Used	Available	Utilization rate (%)	Used	Available	Utilization rate (%)
BRAM	290.5	545	53.30	286	312	91.67
DSP	710	900	78.89	1421	1728	82.23
LUT	111415	218600	50.97	149623	230400	64.94
FF	73213	437200	16.75	199529	460800	43.30

TABLE III Performance Comparison between Software Implementation and Two FPGA Platforms

Platform	**Intel i7-5960x**	**Xilinx ZC706**	**Xilinx ZCU104**
Technology	22nm	28nm	16nm
Frequency	3.0 GHz	100 MHz	200 MHz
Power	140 W	14.43 W	26.50 W
Running time(sec)	19.41	1.98	0.46
T_{pp}/T (%)	18~87.8%	2.2~42.3%	0.52~14.5%
Performance (GFLOPS)	2.48	24.3	104
Energy efficiency	1x	95.1x	221.6x

We also compare the performances of our FPGA accelerator with a recent work [11]. In [11], the authors adopt a cross correlation-based method to extract the BFS information. Since the computation complexity of cross correlation is proportional to the square of frequency number of the input BGS, the authors simplify the original algorithm through a moving average filter to narrow the search region at the cost of reduced estimation accuracy. The time consumption of processing 96100 BGSs in [11] is 0.33s, corresponding to 14.7 GOPS equivalent performance. Although our accelerator costs 0.13s longer than [11], considering the absolute performance of our accelerator is 104 GFLOPS which is about 7× higher than [11], the increased time consumption is trivial and will not pose any burden in real applications. Besides, the floating point data type in our design can provide much higher precision and dynamic range than fixed point in [11], and the quantization process is also eliminated. Moreover, our method does not involve any pre-processing like interpolation and moving average as in [11] to reduce the computation complexity, hence the overall workflow is more concise. In summary, the performance of our BOTDA fiber sensor accelerator is competitive regarding both the time consumption and the absolute performance.

Besides, we list our accelerator and several recent FPGA based support vector machine implementations in Table IV. Although our SVR model is 10.4× and 12.2× larger than [28] and [29], the average time consumption for a single classification or regression of our accelerator is only 9% and 62% of [28] and [29]. [30] has similar model size with us since it contains 5 independent classes, but its average time consumption for a single classification is 52.2× longer than ours.

TABLE IV Comparison with Other FPGA based Support Vector Machine Implementations

	EMBC'13 [29]	TCI'15 [30]	JSPS'17 [28]	Proposed
Device model	Xilinx XC4VSX35	Xilinx XC5VLX110T	Xilinx XC7Z020	Xilinx XCZU7EV
Task	Microarray classification	Image classification	Arrhythmia detection	Temperature extraction
Number of support vectors	20	100	1274	1136
Feature dimension	1024	500	18	220
Frequency (MHz)	137.7	50	25	200
Time consumption (sec)	7.64×10^{-6}	2.5×10^{-4}	5.12×10^{-5}	4.79×10^{-6}

E. *Theoretical Analysis and Discussions*

In Part B and C, we have systematically optimized the original linear SVR decision function for hardware implementation. Loop distribution and loop interchange enable efficient pipeline strategy to be used for partial sum calculation, loop unroll further greatly reduces the latency through parallelizing the MAC operations. Furthermore, the batch processing method makes the latency of the long adder chain shared by multiple inputs, which makes the linear scaling of speedup holds approximately. These optimization techniques make the SVR decision function very suitable to be mapped to FPGA, which are also reflected in the hardware structures in Fig. 9 and Fig. 10. If we further analyze Algorithm 3, we find that we have actually transformed the partial sum matrix calculation and final sum vector calculation to matrix-matrix multiplication and matrix-vector multiplication as follows:

$$\underbrace{\begin{array}{c|c|c} \text{block 1} & \text{block 2} & \cdots\cdots\ \text{block } N_s/f \\ \begin{matrix} ps_{1,1} & ps_{1,2} & ps_{1,3} \\ ps_{2,1} & ps_{2,2} & ps_{2,3} \\ \vdots & \vdots & \vdots \\ ps_{B,1} & ps_{B,2} & ps_{B,3} \end{matrix} & \begin{matrix} ps_{1,4} & ps_{1,5} & ps_{1,6} \\ ps_{2,4} & ps_{2,5} & ps_{2,6} \\ \vdots & \vdots & \vdots \\ ps_{B,4} & ps_{B,5} & ps_{B,6} \end{matrix} & \begin{matrix} \cdots & ps_{1,N_S} \\ \cdots & ps_{2,N_S} \\ \ddots & \vdots \\ \cdots & ps_{B,N_S} \end{matrix} \end{array}}_{\text{Partial sum matrix}}$$

$$= \underbrace{\begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,M} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ X_{B,1} & X_{B,2} & \cdots & X_{B,M} \end{bmatrix}}_{\text{Batch of input vectors}} \times \underbrace{\begin{array}{c|c|c} \text{block 1} & \text{block 2} & \text{block } N_s/f \\ \begin{matrix} SV_{1,1} & SV_{1,2} & SV_{1,3} \\ SV_{2,1} & SV_{2,2} & SV_{2,3} \\ \vdots & \vdots & \vdots \\ SV_{M,1} & SV_{M,2} & SV_{M,3} \end{matrix} & \begin{matrix} SV_{1,4} & SV_{1,5} & SV_{1,6} \\ SV_{2,4} & SV_{2,5} & SV_{2,6} \\ \vdots & \vdots & \vdots \\ SV_{M,4} & SV_{M,5} & SV_{M,6} \end{matrix} & \begin{matrix} \cdots & SV_{1,N_S} \\ \cdots & SV_{2,N_S} \\ \ddots & \vdots \\ \cdots & SV_{M,N_S} \end{matrix} \end{array}}_{\text{Support vector matrix}} \tag{4.9}$$

$$\underbrace{\begin{bmatrix} fs_{1,1} \\ fs_{2,1} \\ \vdots \\ fs_{B,1} \end{bmatrix}}_{\substack{\text{Btach of} \\ \text{outputs}}} = \underbrace{\begin{array}{c|c|c} \text{block 1} & \text{block 2} & \cdots\cdots\ \text{block } N_s/f \\ \begin{matrix} ps_{1,1} & ps_{1,2} & ps_{1,3} \\ ps_{2,1} & ps_{2,2} & ps_{2,3} \\ \vdots & \vdots & \vdots \\ ps_{B,1} & ps_{B,2} & ps_{B,3} \end{matrix} & \begin{matrix} ps_{1,4} & ps_{1,5} & ps_{1,6} \\ ps_{2,4} & ps_{2,5} & ps_{2,6} \\ \vdots & \vdots & \vdots \\ ps_{B,4} & ps_{B,5} & ps_{B,6} \end{matrix} & \begin{matrix} \cdots & ps_{1,N_S} \\ \cdots & ps_{2,N_S} \\ \ddots & \vdots \\ \cdots & ps_{B,N_S} \end{matrix} \end{array}}_{\text{Partial sum matrix}} \underbrace{\begin{bmatrix} \beta_{1,1} \\ \beta_{2,1} \\ \beta_{3,1} \\ \hline \beta_{4,1} \\ \beta_{5,1} \\ \beta_{6,1} \\ \hline \vdots \\ \beta_{N_s,1} \end{bmatrix}}_{\substack{\text{Coeffcients} \\ \text{for SVs}}} + b \tag{4.10}$$

For matrix-matrix multiplication in Equation (4.9), we tile the support vector matrix into small blocks and the input vectors multiply each block in serial. The partial sum matrix is also tiled accordingly. For the matrix-vector multiplication in Equation (4.10), the coefficients vector for support vectors also needs to be partitioned to maintain same level of parallelism. As a result, the two operations are both heavily parallelized, which could take the advantage of massive DSP resources and dual port RAMs on FPGA. To be more specific, the parallel MAC array for matrix-matrix multiplication and cascaded MAC array for matrix-vector multiplication are based on same amount of DSP resources, making our implementation achieve very high hardware utilization efficiency since almost no DSP resources are idle during the computation.

V. Conclusion

In this paper, a new temperature prediction method for BOTDA fiber sensor systems based on SVR is proposed. We experimentally verify that SVR can achieve comparable performances as SVC, XC, LCF and ANN under different SNRs. From the hardware perspective, SVR is more hardware friendly than other four methods without modifications and complicated pre-processing. To accelerate the processing speed of SVR, linear SVR decision function is optimized systematically. The loop-carried dependence in loop iterations is eliminated by loop distribution and loop interchange. Therefore, the pipeline efficiency of the nested loop is greatly improved. We also propose a batch processing method to further decrease the latency. Using the proposed optimization methods, linear SVR decision function is implemented on two FPGA boards Xilinx ZC706 and ZCU104 to process 96,100 BGSs from 38.44-km FUT acquired from a BOTDA system. Our hardware accelerator can achieve up to 42× speedup compared with the software implementation with i7-5960x CPU. The post-processing time for 96,100 BGSs along 38.44-km FUT is only 0.46 seconds with ZCU104, which makes our implementation capable of real-time prediction. Meanwhile, the power consumption of FPGA is also much lower than a high-end CPU, making the energy efficiency of our FPGA implementation up to 226.1× higher than the software implementation based on LIBSVM.

References

[1] A. Barrias, J. R. Casas and S. Villalba, "A review of distributed optical fiber sensors for civil engineering applications," *Sensors*, vol. 16, no. 5, pp. 748, 2016.

[2] E. Buchoud, V. Vrabie, J. Mars, G. D'Urso, A. Girard, S. Blairon and J. Henault, "Quantification of submillimeter displacements by distributed optical fiber sensor," *IEEE Trans. Instrum. Meas.*, vol. 65, no. 5, pp. 413-422, 2016.

[3] M. J. Garcia, J. A. Ortega, J. A. Chavez, J. Salazar and A. Turo, "A novel distributed fiber-optic strain sensor," *IEEE Trans. Instrum. Meas.*, vol. 51, no. 4, pp. 685-690, 2002.

[4] C. Wang and K. Shida, "A low-cost double-fiber model distributed optical fiber sensor," *IEEE Trans. Instrum., Meas*., vol. 56, no. 4, pp. 1481-1481, 2007.

[5] T. Kurashima, T. Horiguchi and M. Tateda, "Distributed-temperature sensing using stimulated Brillouin scattering in optical silica fibers," *Opt. Lett.*, vol. 15, no. 18, pp. 1038-1040, 1990.

[6] C. Li and Y. Li, "Fitting of Brillouin spectrum based on LabVIEW," *Proc. 5th Int. Conf. Wireless Commun., Netw. Mobile Comput*., pp. 1–4, 2009.

[7] C. Zhang, Y. Yang, and A. Li, "Application of Levenberg–Marquardt algorithm in the Brillouin spectrum fitting," *Proc. of SPIE*, vol. 7129, pp. 71291Y, 2008.

[8] M. Soto, S. Le Floch, L. Thevenaz, "Bipolar optical pulse coding for performance enhancement in BOTDA sensors," *Opt. Exp*., vol. 21, no. 14, pp. 16390-16397, 2013.

[9] A. Denisov, M. A. Soto, L. Thevenaz, "Going beyond 1000000 resolved points in a Brillouin distributed fiber sensors: theoretical analysis and experiment demonstration", *Light Sci. Appl.,* vol. 5, no. 6, pp. e16074, 2016.

[10] M. A. Farahani, E. Castillo-Guerra and B. G. Colpitts, "Accurate estimation of Brillouin frequency shift in Brillouin optical time domain analysis sensors using cross correlation," *Opt. Lett.*, vol. 36, no. 21, pp. 4275-4277, 2011.

[11] M. Abbasnejad and B. Alizadeh, "FPGA-based implementation of a novel method for estimating the Brillouin frequency shift in a BOTDA and BOTDR sensors," *IEEE Sensors J.*, vol. 18, no. 5, pp. 2015-2022, 2018.

[12] A. K. Azad, L. Wang, N. Guo, H. Y. Tam and C. Lu, "Signal processing using artificial neural network for BOTDA sensor system," *Opt. Exp*, vol. 24, no. 6, pp. 6769-6782, 2016.

[13] H. Wu, L. Wang, N. Guo, C. Shu and C. Lu, "Brillouin optical time-domain analyzer assisted by support vector machine for ultrafast temperature extraction," *J. Lightw. Technol*, vol. 35, no. 19, pp. 4159-4167, 2017.

[14] H. Wu, L. Wang, N. Guo, C. Shu and C. Lu, "Support vector machine assisted BOTDA utilizing combined Brillouin gain and phase information for enhanced sensing accuracy," *Opt. Exp*, vol. 25, no. 25, pp. 31210-31220, 2017.

[15] H. Wu, L. Wang, Z, Zhao, C. Shu and C. Lu, "Support vector machine based differential pulse-width pair Brillouin optical time domain analyzer," *Photon. J.*, vol. 10, no. 4, pp. 1-11, 2018**.**

[16] G. Garcia, C. Jara, J. Pomares, A. Alabdo, L. Poggi and F. Torres, "A survey on FPGA-based sensor systems: towards intelligent and reconfigurable low-power sensors for computer vision, control and signal processing," *Sensors*, vol. 14, no. 4, pp. 6247-6278, 2014.

[17] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Trans. Computer-Aided Design Integr. Circuit Syst.*, vol. 30, no. 4, pp. 473-491, 2011.

[18] B. A. Draper, J. R. Beveridge, A. P. W. Bohm, C. Ross and M. Chawathe, "Accelerated image processing on FPGAs," *IEEE Trans Image Process.*, vol. 12, no. 12, pp. 1543-1551, 2003.

[19] H. Wu, H. Wang, C. Shu, C. Choy and C. Lu, "Brillouin Optical Time Domain Analyzer Fiber Sensor Based on FPGA Accelerated Support Vector Regression," *Conf. on Opt. Fiber Commun. (OFC) 2019*, paper Th2A.18.

[20] C. Chang and C. Lin, "LIBSVM: a library for support vector machines," *ACM Trans. Intel. Sys. Tec.*, vol. 2, no. 3, pp. 27, 2011.

[21] L. Bottou and C. J. Lin, "Support vector machine solvers," *Large scale kernel machines*, vol. 3, no. 1, pp. 301-320, 2007.

[22] N. Guo, L. Wang, H. Wu, C. Jin, H. Tam and C. Lu, "Enhanced coherent BOTDA system without trace averaging," *J. Lightw. Technol,* vol. 36, no. 4, pp. 871-878, 2018.

[23] M. A. Soto, J. A. Ramírez and L. Thévenaz, "Intensifying the response of distributed optical fibre sensors using 2D and 3D image restoration," *Nat. Commun.*, vol. 7, pp. 10870, 2016.

[24] H. Wang, W. Shi and C. Choy, "Hardware design of real time epileptic seizure detection based on STFT and SVM," *IEEE Access.*, vol. 6, pp. 67277-67290, 2018.

[25] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *in Proc. ACM Int. Conf. Multi*., 2014, pp. 675-–678.

[26] T. Posewsky and D. Ziener, "Efficient deep neural network acceleration through FPGA-based batch processing," *in International Conf. on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, 2016, pp. 1-8.

[27] Intel® Core™ i7-5960X Processor Extreme Edition. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/82930/intel-core-i7-5960x-processor-extreme-edition-20m-cache-up-to-3-50-ghz.html

[28] V. Tsoutsouras, K. Koliogeorgi, S. Xydis, and D. Soudris, "An exploration framework for efcient high-level synthesis of support vector machines: Case study on ECG arrhythmia detection for Xilinx Zynq SoC," J. Signal Process. Syst., vol. 88, no. 2, pp. 127-147, 2017.

[29] H. M. Hussain, K. Benkrid and H. Seker, "Reconfiguration-based implementation of SVM classifier on FPGA for Classifying Microarray data," *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Osaka, 2013, pp. 3058-3061.

[30] M. Qasaimeh, A. Sagahyroon and T. Shanableh, "FPGA-Based Parallel Hardware Architecture for Real-Time Image Classification," *IEEE Transactions on Computational Imaging*, vol. 1, no. 1, pp. 56-70, March 2015.