Received: 24 November 2020 | Revised: 16 January 2021 | Accepted: 2 February 2021

IET Optoelectronics

DOI: 10.1049/ote2.12021

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH PAPER

PH-RLS: A parallel hybrid recursive least square algorithm for self-mixing interferometric laser sensor

Zohaib A Khan[1] | **Tassadaq Hussain**[1,2] | **Usman Zabit**[3] | **Muhammad Usman**[2,3] | **Eduard Ayguade**[4]

[1]Riphah International University, Islamabad, Pakistan

[2]UCERD Pvt Ltd, Islamabad, Pakistan

[3]School of Electrical Engineering and Computer Science NUST, Islamabad, Pakistan

[4]Barcelona Supercomputing Center, Barcelona, Spain

Correspondence

Zohaib A Khan, Riphah International University, Islamabad, Pakistan.
Email: zohaib.154@gmail.com

Funding information

The authors state that they did not receive any funds

Abstract

The authors present the parallel-hybrid recursive least square (PH-RLS) algorithm for an accurate self-mixing interferometric laser vibration sensor coupled with an accelerometer under industrial conditions. Previously, this was achieved by using a conventional RLS algorithm to cancel the parasitic vibrations where the sensor itself is not in the stationary environment. This algorithm operates in sequential mode and due to its compute and data-intensive nature, the algorithm does not work for real-time applications, hence requires parallel computing. Therefore, the existing conventional RLS C program is parallelized by using hybrid OpenACe C/MPI (Open Accelerators/Message Passing Interface) parallel programming models and tested on Barcelona Supercomputing Center CTE-Power9 Supercomputer. The computational performance of the proposed PH-RLS algorithm is compared with the existing conventional RLS code by executing on multi distributed processors and uni-core processor architecture, respectively. While comparing the performance of conventional RLS with a PH-RLS algorithm on eight nodes of CTE-Power9 supercomputer, the results show that the PH-RLS algorithm gets 5857 times of performance improvement as compared to the conventional RLS implementation on a single node system. The results show that the proposed PH-RLS also gives a scalable performance for a different range of vibration signals, making it a suitable choice for real-time self-mixing interferometer sensing systems working under industrial conditions.

1 | INTRODUCTION

Self-mixing (SM) or optical feedback interferometry technique has been regularly used during the last few decades for distance [1], velocity [2], vibration [3], displacement [4] and biomedical [5] sensing application.

It is characterized by a simple implementation, compact structure, self-aligned, and a low-cost sensor which makes it an attractive choice for industry and large-scale consumer embedded sensing applications [6]. These sensors have worked very well in the laboratory condition. However, in the presence of parasitic mechanical vibrations, that is under industrial conditions, it fails to provide the correct results.

The Recently adaptive solution is presented in which the parasitic vibrations are cancelled by using adaptive filter algorithms [7].

The integration of adaptive filters with an accelerometer coupled self-mixing laser sensor makes it possible to use the sensor in an embedded/industrial environment where extraneous parasitic movements are present. For this purpose, least mean squares (LMS) and recursive least squares (RLS) algorithms based adaptive filters are employed for the cancellation of parasitic mechanical vibrations. RLS algorithm provides very good results when compared to LMS algorithm with better convergence rate and without any adjustment of filter parameters, but it requires high performance processing and memory management systems with the increase in filter order and a sampling rate of the input signal.

The RLS filter has been widely used for many other applications like noise cancellation, prediction, and system identification both in stationary and non-stationary environments [8] but its computational complexity and memory requirements

This is an open access article under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs License, which permits use and distribution in any medium, provided the original work is properly cited, the use is non-commercial and no modifications or adaptations are made.

© 2021 The Authors. *IET Optoelectronics* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.

have made it an impossible choice, for real-time signal processing. The RLS-based filter was applied to a nonlinear channel identification problem and includes its application of satellite communication and digital recording systems to record tracks in time-varying scenarios [9]. So in order to reduce the computational cost, a sliding-window approach has been presented in which the data vector is divided into the streams and then a window of size N is created by considering only the last pairs of the stream. The processing is done over the entire window stream and the corresponding output streams are generated. This method has reduced the computational complexity but this approach may have some limitations in time-varying applications because the RLS filter would need to update its weight vector and thus requires learning time for each stream.

Different algorithms of RLS are also used for the cancellation of impulsive noise [10]. For this purpose, the RLS algorithm has been modified to the new form known as recursive least M-estimate algorithm (RLM) [11] in which M-estimator cost function is used instead of conventional least-squares based cost function. RLM is more robust as compared to RLS in case of impulsive noise or the signals contaminated with Gaussian noise but the RLM has similar complexity issues as that of RLS.

Fast transversal recursive least squares (FT-RLS) algorithm was proposed for the noise cancellation applications [12]. FT-RLS becomes an attractive choice for the cancellation of white noise even when large filter order is required because of its less complexity and equal performance as compared to conventional RLS. The low complexity of FT-RLS has made it a highly recommended choice but it is very sensitive to quantization effect and causes stability problems. RLS algorithm was implemented on FPGA (field programmable gate array) with fixed-point operations by using less number of elements of adaptive array antenna due to its fast convergence time [13]. As RLS algorithm requires many multiplications and divisions operations which make it unsuitable for its implementation on DSP (digital signal processing) or FPGA. So, the computation is simplified by using only four array elements, which makes its implementation possible on FPGA. Thus the implementation of the RLS algorithm on FPGA or DSP is not possible for the practical applications where a large number of array elements are required to achieve high directivity, gain, and strong signal strength.

In order to solve the complex and compute-intensive nature of the RLS algorithm, the sensing applications require high performance parallel computing methodology. Thus, to get better performance and to manage the algorithm complexity, the sensor applications are required to execute in parallel on the multi-core heterogeneous processing system architecture [14].

Here, the authors have presented an open accelerators (OpenACC)- and message passing interface (MPI)-based parallel version of the RLS algorithm termed as parallel-hybrid recursive least square (PH-RLS) and executed on Barcelona Supercomputing Center CTE-Power9 supercomputer. The methodology of the PH-RLS algorithm uses earlier published conventional RLS C program [7], finds control and computation by using control data flow graph (CDFG) model and based on the CDFG the algorithm is partitioned into multiple tasks, schedules the tasks on distributed processing system using MPI programming environment and executes each task on a shared memory multi-core processor system using OpenACC. The results show that the PH-RLS gives a scalable performance for different target vibrations even when higher filter order and sampling rate is used. Performance, scalability, and portability have been achieved while programming the parallel-hybrid implementation of the RLS algorithm.

2 | SELF-MIXING INTERFEROMETRY

When laser diode beams direct on a target, a small portion of the emitted laser beam is reflected from the target and then re-inserted into the laser cavity, and causes the interference with the emitted laser beam, thus causes both amplitude and frequency modulation (shown in Figure 1). The amplitude modulation results in the variation of power laser beam emitted by the laser diode, which is known as 'self-mixing interferometric' (SMI) signal and can be thus expressed by (Equation (1)) [15].

In Equation (1), P_0 is the emitted optical power under free-running conditions, m represents the modulation index, and $x_f(t)$ is the laser output phase in the presence of feedback, given by Equation (2).

$$P[t] = P_0[1 + m.cos[x_f(t)]] \tag{1}$$

$$X_f(t) = 2\Pi \frac{D(t)}{\frac{\lambda_f(t)}{2}} \tag{2}$$

The $D(t)$ in Equation (2) represents the target displacement. The emission wavelength subject to feedback $\lambda_f(t)$ is given by the phase Equation (3).

$$x_0(t) - x_f(t) - Csin[x_f(t) + arctan(\alpha)] = 0 \tag{3}$$

where α is the line-width enhancement factor [16], C is the feedback coupling factor that defines the SM operating regime [17] and $x_0(t)$ is the laser output phase in the absence of feedback, found by replacing $\lambda_f(t)$ with λ in Equation (2), where λ is the laser diode emission wavelength under free running conditions.

2.1 | Signal processing

The signal processing required for the adaptive SSA-SM sensor (shown in Figure 2) can be grouped into three major parts namely self-mixing interferometric and acceleration signal, adaptive filtering, and RLS algorithm.

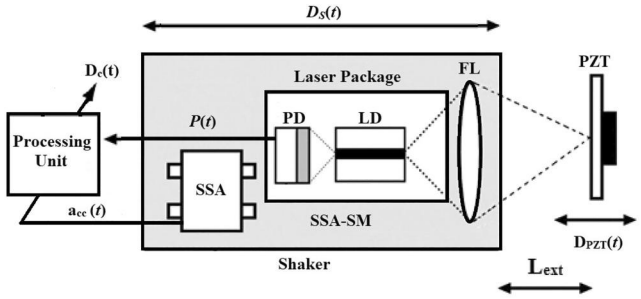

FIGURE 1 Block diagram of the adaptive solid-state accelerometer coupled self-mixing (SSA-SM) sensor: photo diode (PD), laser diode (LD), focusing lens (FL), and piezoelectric transducer (PZT)

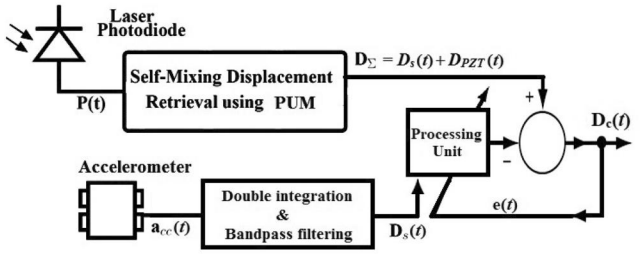

FIGURE 2 Block diagram of the signal processing required for the adaptive SSA-SM sensor: PUM (phase unwrapping method)

2.2 | Self-mixing interferometric and acceleration signal

The configuration of the SMI sensing system is shown in Figure 1. It can be seen that, in the SMI laser sensor, the photodiode (PD) and laser diode (LD) are placed in the same package. The variation in the optical power of laser diode caused by interference or SM phenomenon is acquired by PD. This acquired SMI signal P(t) by photodiode is then processed by different unwrapping algorithms to recover the information signal. Here, the phase unwrapping method (PUM) has been used for vibration retrieval which offers a precision of $\lambda/16$ [18].

The signal-processing steps required for retrieval of correct vibration $D_c(t)$ is shown in Figure 2. The signal acquired from $P(t)$, by using PUM is corrupted signal denoted as $D_\Sigma(t)$, and it is the sum of piezoelectric transducer (PZT) which serve as a target and parasitic mechanical vibrations, that is $D_\Sigma = D_{PZT}(t) + D_s(t)$. These mechanical vibrations are generated by mechanical shaker. Solid-state accelerometers (SSA) has been used to measure the parasitic vibration $D_s(t)$, by double integrating the acceleration signal $a_{cc}(t)$ and bandpass filtering. The bandpass filtering is done to avoid saturation in high-frequency acceleration signals. The corrupted signal D_Σ and parasitic vibration $D_s(t)$ signal is then fed into adaptive processing unit to get the corrected vibration signal $D_c(t)$.

2.3 | Adaptive filtering

Adaptive filters are used in many applications because of their ability to operate in unknown and changing environments. Adaptive filters continuously tune its weight vector by using an adaptive algorithm with the objective of minimizing cost/error function. There are many applications of adaptive filters, some of them includes acoustic echo cancellation, adaptive interference cancellation, adaptive line enhancer, active noise control, adaptive channel equalizer, computer vision, space-time signal processing, beam-forming [8] for mobile communication, acoustic feedback reduction for hearing aids, and adaptive array processing.

These applications provide a useful platform to evaluate the performance of different adaptive algorithms by comparing its different cost functions like least squares (LS), means square error (MSE), and weighted least squares (WLS) etc. The two commonly used adaptation algorithms are the RLS and the LMS. The choice of the adaptive algorithm varies from application to application depending on the characteristics of the algorithm which includes computational complexity, the speed of convergence, the minimum error of convergence, and the numerical stability. The main benefit of the LMS algorithm is its simplicity. However, for complex signals having a wide spectral dynamic band, the LMS algorithm has an irregular rate of convergence. Moreover, for non-stationary having a high rate of change, LMS is not a good choice. On the other side, the RLS algorithm can work in a non-stationary environment [8], with a better convergence rate as compared to the LMS algorithm. This excellent performance makes the RLS algorithm a better choice while working in time-varying scenarios but at the cost of an increased computational burden and stability issues.

2.4 | Recursive least square algorithm

The RLS algorithm consists of the multiplication of estimated error $e(n)$ and the gain vector $k(n)$. The gain vector consists of $\phi^{-1}(n)$, the inverse of the deterministic correlation matrix, and input vector $X(n)$ [8]. The error signal is the difference between desired $d(n)$ and input signal $X(n)$ (Equation (4)). In this case, the input signal $X(n)$ is represented as the parasitic signal $D_s(t)$ and the desired signal becomes the corrupted signal donated as D_Σ.

$$e(n) = d(n) - w^T(n-1)X(n) \tag{4}$$

The filter weight vector is updated by relation

$$w(n) = w(n-1) + k(n)e(n) \tag{5}$$

where $k(n)$ represent filter gain and computed by relation shown in Equation (6).

$$k(n) = \frac{\lambda^{-1}\phi_{yy}(n-1)X(n)}{1 + \lambda^{-1}X^T(n)k(n)X(n)} \tag{6}$$

where ϕ_{yy} is the correlation matrix and λ is the adaptation or forgetting factor used for adjusting the filter gain. The correlation matrix is updated by Equation (7) [8].

```
1  Begin
2  Const: N, f_len, λ, δ, I
3  Input: x_(N,1), d_(N,1)      // N*1
4  φ(0) ← δ^-1 * I
5  W_(M,1) ← 0
6  φ ← eye(f_len)/δ
7  N ← length (x)
8  Variable: e_(N,1) ← 0  // N*1
9  Variable: U_(M-N,1) ← 0  // N*1
10 T1: For i ← f_len : N
11    T1-1: For j ← I : i - f_len ; j--
12          U(j,1) ← x(j,1)
13    End T1-1
14 K ← λ^-1 * φ * U / (1+(λ^-1 * U´ * φ * U))
15 e(i) = d(i) − W´ * U
16 W = W + K * conj(e(n))
17 φ = λ^-1 * (φ − λ^-1 * K * U´ * φ)
18 End T1
19 End
```

FIGURE 3 Conventional RLS pseudo-code

$$\phi_{yy} = \lambda^{-1}\phi_{yy}(n-1) - \lambda^{-1}k(n)X^{T}(n)\phi_{yy}(n-1) \tag{7}$$

The initial value of correlation matrix is set to be

$$\phi_{yy} = \delta I \tag{8}$$

The computational complexity analysis of the RLS algorithm can be realized in a better way by understanding its pseudo-code shown in Figure 3. The pseudo-code of the RLS algorithm has been devised with the dimension of each vector shown here. It is evident from the pseudo-code that by increasing the filter order and sampling rate, the size of the filter gain and auto-correlation matrix also increases, making the RLS algorithm more complex and drastically increases arithmetic operations and memory requirements.

3 | PARALLEL-HYBRID RLS DEVELOPMENT METHODOLOGY

This section explains the development methodology of the PH-RLS algorithm and is further subdivided into two subsections, the PH-RLS programming and PH-RLS processing system.

3.1 | PH-RLS programming

Before programming PH-RLS for distributed computing, we need to understand the complexity and code behaviour. The complexity of the RLS algorithm can be seen from the calculations of filter gain and updating the correlation matrix ϕ_{yy} (discussed in Section 2). The complexity point of the RLS algorithm (shown in Figure 3) can be estimated in terms of the number of multiplications. For a single input, the RLS algorithm requires $4N^2 + 4N + 2$ multiplications in total [19] (division counted as multiplication), where N is the filter's order. For a 1024 order filter, the RLS algorithm requires approximately 4 million operations to process an input sample, therefore its operational intensity (floating point operations per Second Byte) (FLOPS/Byte) would be $(4 \times 10^6 \ FLOPS/2Byte)$ 2.5×2^6 FLOPS/Byte. To overcome the computational cost of RLS, for real-time implementation, the algorithm needs to perform efficient data distribution, task management, and parallel processing.

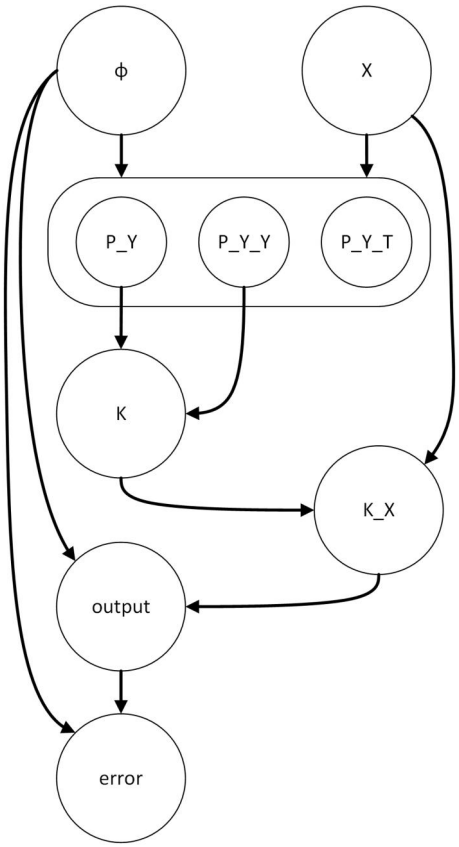

FIGURE 4 Control data flow graph of OpenMPI implementation of PH-RLS algorithm

For parallel hybrid programming, a CDFG model of the PH-RLS algorithm is generated (shown in Figure 4). Depending upon the CDFG model, the RLS algorithm is subdivided into four different parts: partitioning, communication, agglomeration, and mapping. The partitioning and communication targets concurrency and scalability, whereas agglomeration, and mapping focus on data locality and other performance-related issues. The partitioning stage of the RLS algorithm is subdivided into multiple smaller tasks that can perform computation concurrently on an accelerator. Each task includes iterations (loop), branches (conditions), and operations (computation). The communication links the tasks and manages the dependent data. The agglomeration combines the tasks if necessary. The first two stages of a design are evaluated concerning performance requirements and implementation costs. Depending upon the available hardware resources if necessary, tasks are combined into larger tasks to improve the performance or to reduce development costs. In the end, each task is assigned to a processor core that meets the required goals of maximizing hardware

resource utilization and reducing communication costs. During mapping, the task is prioritized statically or based on hardware intelligence it can be determined in real-time by load-balancing algorithms. The CDFG model shows that the RLS algorithm does not only compute and data-intensive but also complex in nature.

The parallel-hybrid version of RLS implementation is shown in Figure 5. The PH-RLS utilizes GPU accelerators to get real-time performance for all the parallelizable tasks (T1–T7) of the algorithm. These tasks either include multiplications (including divisions), reductions, or assignments. The complexity of multiplications and assignments has been reduced by an order with the help of asynchronous parallel GPU blocks and threads. Reductions are not parallel in nature as they need to accumulate the function down to a single value. Hence each order of complexity for reduction has been reduced to a logarithmic complexity. Through these techniques that involve heterogeneous computing, an algorithm complexity of $4N^2 + 4N + 2$ has been reduced to $N\log\ N + 3\log\ N + 3$.

To overcome the memory requirements of the RLS, the memory of each task is allocated on the GPU accelerator. The memories of inputs and outputs samples are assigned in the CPU which relieves the processor for the task of allocating memory. Therefore, the parallel implementation of the algorithm is limited by GPU memory, not CPU memory. To overcome the data dependencies, PH-RLS is programmed to execute the tasks in the pipeline by distributing the tasks on seven distributed compute nodes. Each compute-node receives the required data, processes it, and forwards it over to the relevant nodes. Input to the pipeline is received in batches and computation is initiated as soon as data becomes available to the respective node. Similarly, these techniques have been used to reduce the complexity of the RLS algorithm by involving heterogeneous computing.

```
Begin
Const: N, M
Input X[M][N], phi[M][N], d[M][N]
Start:
T1: For i in 0:N -- Parallel
    T1-1: For j in 0:M -- Striped Parallel
        P_Y[0][i] += X[0][j] * phi[i][j]
    end T1-1
end T1
-----------------------------------------------
T2: For i in 0:M -- Striped Parallel
    P_Y_Y += X[0][i] * P_Y[0][i] + 1
end T2
-----------------------------------------------
T3: For i in 0:M -- Striped Parallel
    K[0][i] = P_Y[0][i] / P_Y_Y
end T3
-----------------------------------------------
T4: For i in 0:N -- Parallel
    T4-1: For j in 0:M -- Parallel
        P_Y_T[i][j] = transpose( P_Y[i][j] )
    end T4-1
end T4
-----------------------------------------------
T5: For i in 0:N -- Parallel
    T5-1: For j in 0:M -- Striped Parallel
        K_X[i][j] += K[0][j] * X[j][0]
    end T5-1
end T5
-----------------------------------------------
T6: For i in 0:N -- Parallel
    T6-1: For j in 0:M
        T6-2: For k in 0:N -- Striped Parallel
            out[i][j] += K_X[i][k] * phi[k][j]
        end T6-2
    end T6-1
end T6
-----------------------------------------------
T7: For i in 0:N -- Parallel
    T7-1: For j in 0:M -- Parallel
        error[i][j] = d[i][j] - out[i][j]
    end T7-1
end T7
-----------------------------------------------
```

FIGURE 5 Parallel-Hybrid version of RLS for CPU-GPU based distributed supercomputer

3.2 | PH-RLS processing system

Here, the CTE-Power9-V100 supercomputer system architecture is described, which is used to execute the PH-RLS algorithm.

CTE-Power9 is an IBM Power9 processors and Nvidia V100 GPU-based supercomputer system interconnected by using Infiniband interconnection and utilizes the Red-Hat Linux Operating System. The parallel-hybrid processing system uses four nodes of CTE-Power9, and each node is equipped with 16 cores and 4 threads/core, and a total of 160 threads, and 4 NVidia-V100 (volta) GPUs.

4 | RESULTS AND DISCUSSION

The performance and scalability of the proposed PH-RLS algorithm is measured by testing it for different input samples and filter orders on a different number of distributed computing resources of CTE-Power supercomputer (discussed in Section 3.2). The section is further subdivided into four subsections; experimental setup, results of experimental signal, performance comparison, and scalability.

4.1 | Experimental setup

The interferometry systems are experimentally tested for the RLS algorithm by using the experimental laboratory setup shown in Figure 6. The SSA used SF1500 accelerometer, which is from Colibrys® (full-scale range of ±3 g and a typical noise resolution of 0.3μg/Hz1/2). The SMI sensor has a Sanyo® DL7140 laser diode (λ = 785 nm) having an output power of 50 mW. The SSA was fixed on the SM sensor head so that it could measure the parasitic movement $D_s(t)$ of the SM sensor head with high precision and accuracy. The SSA-SM sensor head was fixed on a mechanical shaker that was used to generate parasitic vibrations undergone by the sensor. The displacement retrieval results are compared from a reference

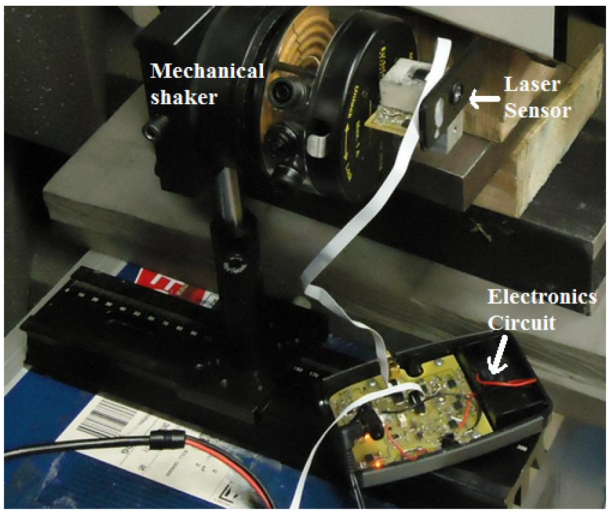

FIGURE 6 SMI laboratory setup: Photograph of SSA-SM sensor using the SF1500 accelerometer and a SM displacement sensor-based on DL7140 laser diode. The sensor head has been mounted on a mechanical shaker to undergo parasitic vibrations

commercial piezoelectric sensor from Physik Instrument (P753.2CD) with 2 nm accuracy. For displacement retrieval from the experimental setup, the PZT is set at the frequency of $f_{PZT} = 65$ Hz with a peak to peak amplitude of 5 μm while the parasitic vibrations disturbing the sensor is set at $f_s = 92$ Hz with a peak to peak amplitude of 4 μm (measured by SF1500 SSA). The corresponding experimental SM signal is shown in Figure 7(a) which has been unwrapped by using PUM to provide $D_{\sum}$ shown in Figure 7(b). In spite of the higher vibration amplitudes, the use of PH-RLS has corrected it to provide $D_c(t)$ which matches very well with the reference target capacitive feedback sensor vibration $D_{PZT}(t)$ shown in Figure 7(d). The error $e(t) = D_{PZT}(t) - D_c(t)$ is shown in Figure 7(e) with an RMS value of 11.9 nm. In this case the filter order and sampling rate is set to 1024 and 100 K respectively.

4.2 | Results of experimental signal

To do the performance comparison of conventional RLS and PH-RLS, the sampling rate is varied with respect to different filter order. The quality of the SMI signal is determined by the ratio of sampling rate and the frequency of target motion [20]. The optimum sampling rate required for correct fringe detection is dependent on the amplitude, frequency, and wavelength of the laser diode [21]. Thus by increasing the amplitude of displacement signal or increasing the resolution of laser diode wavelength, requires a higher sampling rate. For the sake of understanding, if we assume that a SM laser sensor uses a laser diode that emits the light with a wavelength (λ) of 1 μm and it is known that 20 samples are required to accurately detect one fringe [22], therefore as per Nyquist criteria 40 samples are required to accurately detect a signal having a velocity of 1 μm/s. Similarly for the target velocities of 25 μm/s, 0.25 mm/s, 2.5 mm/s, 25 mm/s, and

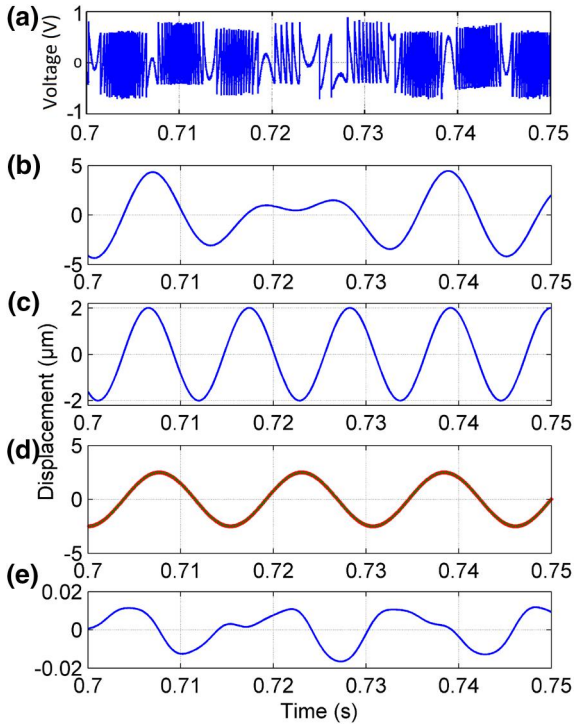

FIGURE 7 Experimental signals for $f_s = 92$ Hz and $f_{PZT} = 65$ Hz : (a) experimental SM interferometric signal, (b) corrupted vibration $D_{\sum}$ retrieved by PUM, (c) parasitic vibration $D_s(t)$ measured by SF 1500 SSA, (d) corrected vibration $D_c(t)$ (green) and reference capacitive feedback sensor vibration $D_{PZT}(t)$ (dotted red), and (e) $e(t) = D_{PZT}(t) - D_c(t)$

250mm/s the systems require 1 K, 10 K, 100 K, 1 M, and 10 M samples per second, respectively. Therefore, the sampling rate is a key parameter in the SMI sensing system and can adversely affect its performance if optimum sampling rate criteria is not satisfied.

In order to test the performance of PH-RLS and to establish its comparison with conventional RLS in-term of processing time, both algorithms are tested on 1 K, 10 K, 100 K, 1 M, and 10 M samples with the filter order of 10, 100, and 1024, respectively. The higher filter order is required to get the sharp roll-off between the passband and stopband which blocks the undesired frequency components in a better way. Table 1 shows the execution time of the conventional RLS algorithm and PH-RLS on a uni-processor and uni CPU-GPU processor cores, respectively. The results show that conventional RLS algorithm is compute-intensive and requires high performance computing resources and therefore, it is not easy to implement a conventional RLS algorithm in a real-time environment where high filter order or sampling rate is required.

4.3 | Performance comparison of PH-RLS and conventional RLS

Here, the performance comparison between conventional RLS and PH-RLS has been established in terms of

TABLE 1 Conventional RLS and PH-RLS execution time using a single processor and GPU, respectively

Velocity (m/s)	Input-samples	Filterll-order	Memory-allocation (MB)	Sequential-RLS time (s)	PH-RLS time (s)
25×10^{-6}	1024	10	0.35	0.073	0.11
0.25×10^{-3}	10,240		2.7	6.91	0.11
2.5×10^{-3}	102,400		27.5	673.5	0.24
25×10^{-3}	1,048,576		278	69113	11.81
250×10^{-3}	10,485,760		2883	Memory-out	121.11
25×10^{-6}	1024	100	2.8	0.73	0.12
0.25×10^{-3}	10,240		27.8	71.35	0.13
2.5×10^{-3}	102,400		287.5	7735.34	1.68
25×10^{-3}	1,048,576		2833.2	Memory-out	160.16
250×10^{-3}	10,485,760		28,332	Memory-out	Memory-out
25×10^{-6}	1024	1024	28	7.46	0.12
0.25×10^{-3}	10,240		278	729.55	0.27
2.5×10^{-3}	102,400		2836	78986.5	20.9
25×10^{-3}	1,048,576		28,467	Memory-out	Memory-out
250×10^{-3}	10,485,760		287,328	Memory-out	Memory-out

execution time (in seconds) as tabulated in Table 1. The third column, 'Memory-Allocation', shows memory required by the RLS algorithm for different inputs and filter orders. This shows that the RLS algorithm is memory bound and requires a huge amount of memory for a bigger number of input samples and filter order. The fourth column, 'Conventional RLS Time', shows time (in seconds) taken by the conventional RLS algorithm to process the inputs using a single processor core of the CTE-Power system. and the last column, PH-RLS time, presents the time taken by the PH-RLS using a single CPU-GPU core. The results show that conventional implementation outperforms the PH-RLS for the case when an input sample of 1024 and a filter order of 10 is used for a target velocity of 25 μm/s. This is because of a memory limitation as for a few inputs, memory allocation and transfer from host to the device is more time-intensive than the computation itself.

Secondly, for the case, when the target is vibrating with the velocity of 2.5 mm/s and the filter order 10 is used, the conventional RLS takes 673.5 s to process the signal for displacement retrieval. This shows that conventional RLS cannot process the data size of 100 K signals easily while on the other hand, PH-RLS has processed it in just 0.24 s, which shows that PH-RLS performance is almost 2806 times better than conventional RLS.

Thirdly, for the input signal size of 10M, the conventional RLS failed to process the signal, the memory-out shows that the algorithm takes huge memory for input, output, and local data-sets and unable to fit in the main memory of the uni-processor architecture, whereas PH-RLS takes 121.11 s to process the same signal because PH-RLS places the local data-sets on the GPU memory.

Lastly, when the filter order is set to 100 and the target is vibrating with the velocity of 250 mm/s, both conventional and PH-RLS failed to retrieve the displacement signal. In this case, the PH-RLS also failed because the memory runs out for a single GPU machine. Similarly, the memory is running out for both algorithms when the input signal of size 1 M and 10 M is used against the filter order of 1024. This shows the complex nature of the RLS algorithm, especially when higher filter order and the sampling rate are used.

4.4 | Parallel performance and scalablity

The performance of PH-RLS for the difficult cases (higher filter order and input data) is measured where it failed to process the input signal on a single GPU machine (discussed in the previous section) of CTE-Power9 Supercomputer (discussed in Section 3.2). In order to achieve this, the PH-RLS is executed on different heterogeneous cores having multiple GPUs to make its real-time implementation possible. The conventional RLS algorithm can be executed only in sequential mode, that is it can occupy only a single processor for execution while on other hand PH-RLS because of an integrated parallel hybrid approach, makes it execution possible on multiple nodes. The performance comparison has been established in terms of the execution time of PH-RLS by executing it on a different number of GPU-based distributed nodes as shown in Table 2. It should be noted that the processing time (shown in Table 2) includes local, shared memory, and external memories read/write, processing, and inter-process communication time of shared memory system.

TABLE 2 Parallel-hybrid RLS algorithm (PH-RLS) scalablity. Execution time against different number of distributed nodes

Velocity (m/s)	Input-samples	Filter-order	Node-1 GPU-2	Node-1 GPU-4	Node-2 GPU-8	Node-4 GPU-16	Node-8 GPU-32
250 x 10^{-3}	10485760	100	memory-out	5899.8	3005.5	1385.14	601.1
25 x 10^{-3}	1048576	1024	memory-out	5930.5	3021.2	1392.3	604.2
250 x 10^{-3}	10485760	1024	memory-out	memory-out	300757	138609.7	60151.4

Column "Node-1 GPU-2" presents time in seconds to process the PH-RLS on a single processing system having two GPUs. Similarly, the other columns show the processing time for 4,8,16 and 32 GPUs.

The results show that the execution time of PH-RLS reduces with the increase in distribution nodes. For the first case, when a target is vibrating with a velocity of 250 mm/s and the filter order of 100 is used, the GPU-32 has almost 9 times performance improvement against the GPU-4. If we consider the most complex case, when the filter order of 1024 and input size of 10M is used, GUP-4 failed to process and the memory runs out while the GPU-32 takes 60151.4 s to process the input signal which shows the intense complex nature of RLS algorithm. The proposed PH-RLS successfully manages processing resource and allocate more processing cores to the compute-intensive RLS tasks. The results confirm that PH-RLS is highly scalable for heterogeneous distributed computing architecture and can perform load balancing, even for higher sampling rate and filter order.

5 | CONCLUSION

The authors have implemented a CPU-GPU-based parallel hybrid version of conventional RLS algorithm named as PH-RLS. The existing conventional RLS C program is parallelized by using parallel-hybrid OpenACC/MPI parallel programming models and executed on CTE-Power9 supercomputing system. The implementation involves a thorough understanding and segmenting the compute-intensive part of the algorithm, pipelining, parallelizing, and mapping them on multiple cores of the distributed computing system architecture. To validate the performance and scalability of PH-RLS, the algorithm is tested on multiple distributed nodes of the CTE-Power9 Supercomputer of Barcelona Supercomputing Center. The results show that the proposed PH-RLS algorithm gives a scalable performance and is even capable to process the signal of 10M samples/second with filter order of 1K, which makes it the suitable choice for real-world applications where high bandwidth signals are required.

ACKNOWLEDGEMENT

The authors would like to thank the Unal Color of Education Research and Development (UCERD), Private Limited Islamabad, Barcelona Supercomputing Center Spain, and Higher Education Commission Pakistan for the support.

CONFLICT OF INTEREST

The authors state that there is no conflict of interest.

ORCID

Zohaib A Khan https://orcid.org/0000-0002-1196-1910

REFERENCES

1. Magnani, A., et al.: Real-time self-mixing interferometer for long distances. IEEE Trans. Instrument. Measure. 63(7), 1804–1809 (2014)
2. Scalise, L., et al.: Self-mixing laser diode velocimetry: application to vibration and velocity measurement. IEEE Trans. Instrument. Measure. 53(1), 223–232 (2004)
3. Zabit, U., et al.: Spectral processing of self-mixing interferometric signal phase for improved vibration sensing under weak- and moderate-feedback regime. IEEE Sensors J. 19(23), 11151–11158 (2019)
4. Ehtesham, A., et al.: Analysis and implementation of a direct phase unwrapping method for displacement measurement using self-mixing interferometry. IEEE Sensors J. 17(22), 7425–7432 (2017)
5. Ozdemir, S.K., et al.: A comparative study for the assessment on blood flow measurement using self-mixing laser speckle interferometer. IEEE Trans. Instrument. Measure. 57(2), 355–363 (2008)
6. Zabit, U., et al.: Design and analysis of an embedded accelerometer coupled self-mixing laser displacement sensor. IEEE Sensors J. 13(6), 2200–2207 (2013)
7. Khan, Z.A., et al.: Adaptive cancellation of parasitic vibrations affecting a self-mixing interferometric laser sensor. IEEE Trans. Instrument. Measure. 66(2), 332–339 (2017)
8. Diniz, P.S.: Adaptive Filtering. Springer Berlin, Germany (1997)
9. Van.Vaerenbergh, S., et al.: A sliding-window kernel RLS algorithm and its application to nonlinear channel identification 2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings, vol. vol. 5. IEEE (2006). V–V
10. Khan, Z.A., et al.: Adaptive estimation and reduction of noises affecting a self-mixing interferometric laser sensor. IEEE Sensors J. 20(17), 9806–9815 (2020)
11. Chan, S.C., Zou, Y.X.: A recursive least m-estimate algorithm for robust adaptive filtering in impulsive noise: fast algorithm and convergence performance analysis. IEEE Trans. Signal Process. 52(4), 975–991 (2004)
12. Dechene, D.: Fast Transversal Recursive Least-Squares (ft-rls) Algorithm (2007)
13. Matsumoto, N., et al.: Fixed-point digital processing of recursive least-square algorithm toward FPGA implementation of mmse adaptive array antenna Seventh International Symposium on Signal Processing and Its Applications, 2003. Proceedings., vol. 2, pp. 615–617. IEEE (2003)
14. Blanchard, P., et al.: Two-dimensional bend sensing with a single, multi-core optical fibre. Smart materials and structures. 9(2), 132 (2000)
15. Donati, S.: Developing self-mixing interferometry for instrumentation and measurements. Laser Photon. Rev. 6(3), 393–417 (2012)
16. Kim, C., et al.: Effect of linewidth enhancement factor on fringe in a self-mixing signal and improved estimation of feedback factor in laser diode. IEEE Access, 1–1 (2019).
17. Acket, G., et al.: The influence of feedback intensity on longitudinal mode properties and optical noise in index-guided semiconductor lasers. IEEE J. Quantum Electron. 20(10), 1163–1169 (1984)

18. Bernal, O.D., et al.: Study of laser feedback phase under self-mixing leading to improved phase unwrapping for vibration sensing. IEEE Sensors J. 13(12), 4962–4971 (2013)
19. Allen, B., Ghavami, M.: Adaptive Array Systems: Fundamentals and Applications. John Wiley & Sons, London, UK (2006)
20. Cai, X., et al.: Effect of sampling rate on target waveform reconstruction based on self-mixing interference. Optik. 124(10), 932–936 (2013)
21. Amin, S., et al.: Hardware implementation of metric algorithms for a self-mixing laser interferometric sensor. In: 2016 19th International multi-topic conference (INMIC), pp. 1–5. IEEE (2016)
22. Zabit, U., et al.: Self-mixing sensor for real-time measurement of harmonic and arbitrary displacements. In: 2012 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), pp. 754–758. IEEE (2012)

How to cite this article: Khan ZA, Hussain T, Zabit U, Usman M, Ayguade E. PH-RLS: A parallel hybrid recursive least square algorithm for self-mixing interferometric laser sensor. *IET Optoelectron*. 2021;1–9. https://doi.org/10.1049/ote2.12021