High Throughput Implementation of RLS Algorithm Using Fewer Processing Elements

Takeo NIKI Rikita YAMADA Kiyoshi NISHIKAWA Hitoshi KIYA

Dept.of Electrical Engineering, Tokyo Metropolitan Univ.
1-1 Minami-Osawa, Hachioji-shi, Tokyo 192-0397, JAPAN
Phone: +81-426-77-2745, Fax: +81-426-77-2756
E-mail: niki@isys.eei.metro-u.ac.jp

Abstract : This paper proposes a method that enables us to implement the recursive least squares (RLS) algorithm at high throughput rate using fewer processing elements (PEs). It is known that the pipeline processing can provide a high throughput rate. But, pipelining is effective only when enough number of PEs are available. The proposed method achieves high throughput rate using a few PEs. The effectiveness of the proposed method is verified through simulations on programmable digital signal processors (in the following, DSP processors).

1. Introduction

It is well known that the RLS algorithm has a faster rate of convergence than that of the least mean square (LMS) algorithm. However, it is also known that the RLS algorithm has some disadvantages over the LMS. One of the disadvantages is that it requires high computational complexity and therefore, the throughput rate, a number of processable samples per unit time, becomes worse. Due to this degradation of the throughput rate, number of applications where the RLS algorithm can be used are reduced.

As a method to realize high throughput implementation, pipeline technique has been investigated [1, 2, 3, 4]. Systolic array architecture[5, 6, 7, 8] and PIPKAL[8] are well known methods for pipelining the RLS algorithm. When PEs are available as many as these methods require, they can provide a high throughput rate.

Conventional pipelining assume that they can use enough number of PEs. For this reason, when only a few PEs are available, they can not improve the throughput rate. For example, a DSP processor has usually only a few PEs, so it is hard to improve the throughput rate on a DSP processor using pipeline technique.

This paper proposes a method to increase the throughput rate of the RLS algorithm under a constraint that only a few PEs are available[9, 10, 11]. The proposed method has an advantage that it improves the throughput rate without faster PEs. As a result, reduction of power dissipation is possible by restricting the number of PEs. It enables us to cut down the producing cost due to the use of less expensive processor.

This paper is organized as the following. We summarize the PIPKAL as a preparation in the next section, and in section 3, the proposed method is described. In section 4, we simulate the proposed method on a DSP processor for verifying the efficiency of the proposed method.

2. Conventional method

In this section, the conventional method (PIPKAL) is explained and its issue is indicated.

2.1 Preparation

The RLS algorithm is described by the following equations

$$\mathbf{K}(n) = \frac{P(n-1)\boldsymbol{x}(n)}{1+\boldsymbol{x}^T(n)P(n-1)\boldsymbol{x}(n)} \quad (1)$$

$$y(n) = \boldsymbol{x}^T(n)\boldsymbol{W}(n-1) \quad (2)$$

$$e(n) = d(n) - y(n) \quad (3)$$

$$P(n) = P(n-M) - \mathbf{K}(n)\boldsymbol{x}^T(n)P(n-1) \quad (4)$$

$$\boldsymbol{W}(n) = \boldsymbol{W}(n-1) + \mathbf{K}(n)e(n) \quad (5)$$

where $\boldsymbol{W}(n)$ is the coefficient vector of the adaptive filter, $\mathbf{K}(n)$ is Kalman gain, and $\boldsymbol{x}(n)$ is the input vector. Note that, in this paper, vectors are vertical vectors. $d(n)$ is the desired signal and $e(n)$ is the error signal.

The RLS algorithm requires multiplication of $O(N^2)$ where N is the number of filter coefficients.

2.2 PIPKAL algorithm

PIPKAL algorithm[8] is known as one of the methods for pipelining the RLS algorithm. PIPKAL can achieve a high throughput rate when enough amount of PEs are available for updating M sets of filter coefficients in parallel where M shows the number of pipeline stages. PIPKAL enables us to achieve M times higher throughput rate as that of the RLS algorithm. PIPKAL algorithm is described as follows:

$$\mathbf{K}(n) = \frac{P(n-M)\boldsymbol{x}(n)}{1+\boldsymbol{x}^T(n)P(n-M)\boldsymbol{x}(n)} \quad (6)$$

$$y(n) = \boldsymbol{x}^T(n)\boldsymbol{W}(n-M) \quad (7)$$

$$e(n) = d(n) - y(n) \quad (8)$$

$$P(n) = P(n-M) - \mathbf{K}(n)\boldsymbol{x}^T(n)P(n-M) \quad (9)$$
$$\boldsymbol{W}(n) = \boldsymbol{W}(n-M) + \mathbf{K}(n)e(n) \quad (10)$$

Note that P, $\mathbf{K}$ and $\boldsymbol{W}$ are updated using values that were updated M times before. The RLS algorithm can be considered as a case of $2M = 1$. When enough PEs are available to update $W(n)$ in parallel, PIPKAL can achieve a higher throughput rate. Let us illustrate the case of $M = 3$ for example.

Figure 1 shows the situation when PIPKAL have enough PEs. On the other hand, when PIPKAL have only one PE, the situation appears in Figure 2. In these figures, a solid line arrow represents updating coefficients, and a broken line arrow represents an output.

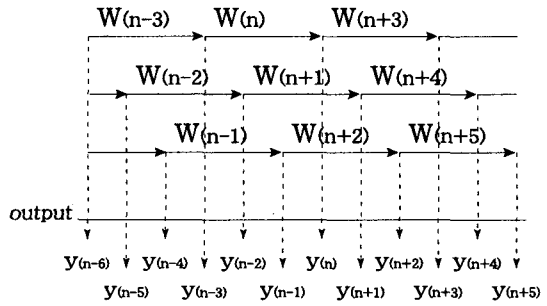

Figure 1: Filter coefficients and outputs in case that Enough PEs are available

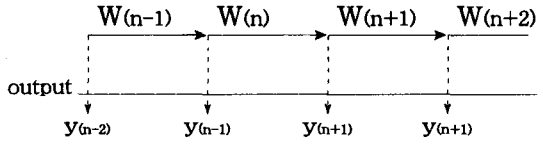

Figure 2: Filter coefficients and outputs in case that Only one PE is available

PIPKAL updates M sets of filter coefficients in parallel. As Figure 1 indicates $\boldsymbol{W}(n-3), \boldsymbol{W}(n-2)$ and $\boldsymbol{W}(n-1)$ are updated in parallel and independently. Each coefficient vecter is updated once per M times. Since filter coefficients are updated in parallel, the output samples $y(n-2), y(n-1)$, and $y(n)$ are produced while $\boldsymbol{W}(n)$ is updated once. Therefore, the throughput of PIPKAL is M times faster than that of the RLS algorithm.

When number of available PEs are limited, however, filter coefficients can not be updated in parallel. Therefore, as Figure 2 indicates, filter coefficients are updated once each time. Only one output sample is produced while $\boldsymbol{W}(n)$ is updated once, then throughput of PIPKAL is the same as that of the RLS algorithm.

In consequence, when only one or a few number of PEs are available, the throughput rate can not be improved since filter coefficients are not updated in parallel. Furthermore, the number of updating until filter coefficients convergence is M times as that of the RLS algorithm.

3. Proposed method

Here, we describe the proposed method. This method enables us to improve the throughput of the RLS algorithm when only one or a few PEs are available.

3.1 Outline of the proposed method

The standard RLS algorithm produces one output per update. Since the amount of computation required to update is constant, we should use faster PEs to increase the throughput rate.

The proposed method can achieve high throughput rate without the need for faster PEs. The proposed method comes from modified PIPKAL [12][13]. Modified PIPKAL is an improved algorithm that enables us to remove the redundancy of PIPKAL. Now, we can see from the equations(1)-(5) that only equation (2) is required to produce an output and the equation (2) is independent of equations (1), (3), (5). Therefore, while updating coefficients $W(n)$, equation (2) can be independently computed several time. As a results, it found that a few outputs can be produced. For the reasons above, the proposed method can achieve higher throughput implementation.

3.2 Implementation of the proposed method

Next, we explain the implementation of the proposed method.

Firstly, we divide the equations(1),(4) and (5) of the RLS algorithm into L parts. Then, insert the equation (2) for computation to produce an output independent of updating equation. Include this insert, we should divide the equations under the condition that number of cycles within each part are equal. An example of the division is shown in Figure 3. Where, $G_1, \cdots, G_L$ are number of cycles that is needed for computing each part i. Consequently, it can make L pieces of output per one update. Therefore, we can achieve L times faster throughput rate as that of the conventional PIPKAL in theory without changing PEs.

3.3 Example

As an example, we show the case when $L = 3$. In this case, equations are grouped as the following three parts.

- part A $\quad i = 3(n-1)$

$$\alpha(n) = P(n-1)\boldsymbol{x}(n) \quad (11)$$
$$\beta(n) = \boldsymbol{x}^T(n)\alpha(n) + 1 \quad (12)$$
$$y(i) = \boldsymbol{x}^T(i)\boldsymbol{W}(n-1) \quad (13)$$

- part B $\quad i = 3(n-1) + 1$

$$e(i) = d(i) - y(i) \quad (14)$$

$$K(n) = \frac{\alpha(n)}{\beta(n)} \quad (15)$$
$$\gamma(n) = \boldsymbol{x}^T(n)P(n-1) \quad (16)$$
$$y(i) = \boldsymbol{x}^T(i)\boldsymbol{W}(n-1) \quad (17)$$

- part C $\quad i = 3(n-1)+2$

$$\delta(n) = K(n)\gamma(n) \quad (18)$$
$$P(n) = P(n-1) - \delta(n) \quad (19)$$
$$\boldsymbol{W}(n) = \boldsymbol{W}(n-1) + K(n)e(n) \quad (20)$$
$$y(i) = \boldsymbol{x}^T(i)\boldsymbol{W}(n-1) \quad (21)$$

Where $\alpha(n), \beta(n), \gamma(n)$, and $\delta(n)$ are temporary values. Equations (17), (21) are inserted to produce an output. While filter coefficients are update once, three output samples will be produced. Therefore, the throughput rate becomes three times as that of the RLS algorithm. This example of implementation with limited processing unit is appeared in Figure 4.

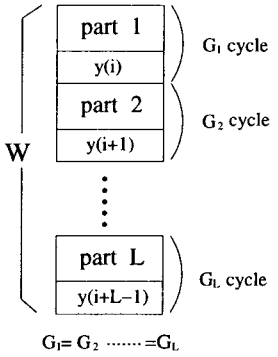

Figure 3: State of dividing in the proposed method

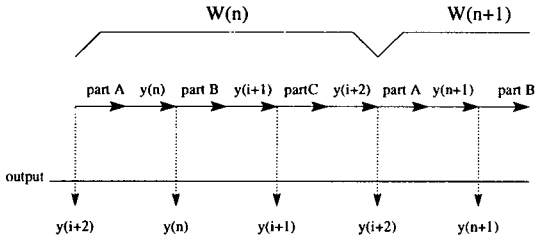

Figure 4: Implementation of the proposed method

Though we explained that each parts should have equal computational complexity. This is a difficult task in practice. For this reason, slight adjustment such as inserting NOP will be necessary.

4. Simulation

We simulated the propose method on a DSP processor for verification of the propose method. Following section shows the results of this simulation.

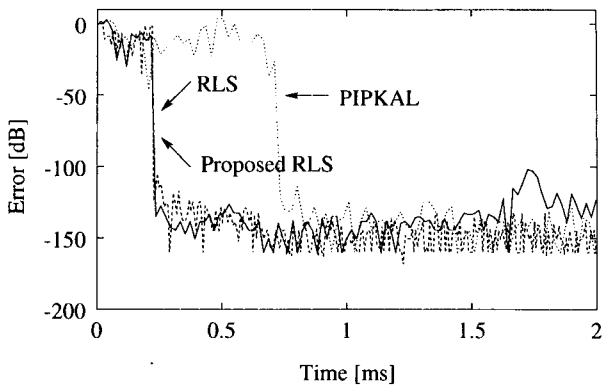

Figure 5: Rate of convergence of each algorithm (vs time)

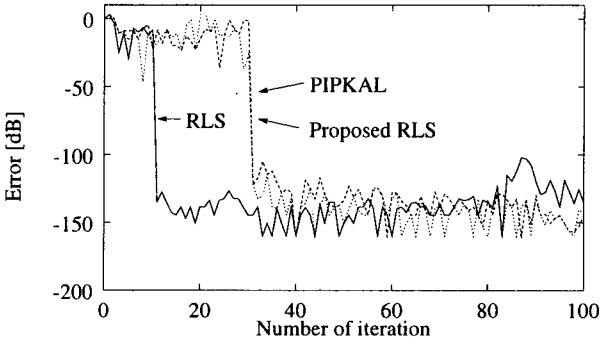

Figure 6: Rate of convergence of each algorithm (vs iteration)

4.1 simulation conditions

We used a DSP processor as an environment where number of available PEs is limited.The target DSP processor was the TMS320C6701 produced by Texas Instruments. This DSP processor has two independent PEs. Each PEs can connect independent data path. We used the Code Composer Studio as a simulation tool and instruction cycle time was set as 5ns. We used a 10 tap low-pass filter as unknown system, and also used 10 tap adaptive filter. The input signal was white noise whose mean is zero, variance is one. Dividing number of PIPKAL and the proposed method was $M = L = 3$. Dividing position of proposed method was the same as the example shown in section 3.3.

4.2 Results

The rate of convergence of the RLS algorithm, PIPKAL and the proposed method appear in Figure 5, 6. Note that there is a deference between Figure 5 and Figure 6 in terms of horizontal axis. The horizontal axis is a 'time' in Figure 5, but in Figure 6, that is a 'number of iteration'.

These figures indicate that the rate of real time convergence of the proposed method is equal to that of the RLS algorithm, and the number of output samples which the proposed method produces is equal to the that of PIPKAL. On the other hand, the rate of convergence of PIPKAL is three times slower than that of the RLS algorithm.

Figure 7 indicates the relation of the maximum number of cycles that is required to produce an output sample $y(n)$ (we call this number G_{max}) and pipeline stages M or divided number L. PIPKAL can not reduce G_{max} regardless of increase of M, the proposed method is enable us to reduce G_{max} by increase of L.

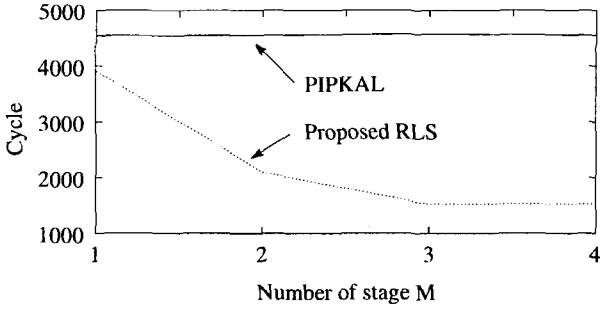

Figure 7: A number of cycles of conventional and proposal

Note that there is a difference between PIPKAL and the proposed method in the case $L = 1$. PIPKAL reads a coefficient vecter L times before from data memory, and write the updated coefficient vecter to data memory for updating L times after. As a result, PIPKAL has a large number of cycles than the proposed method.

Finally, we described the throughput rate of each algorithm in table 1. This table shows the number of samples that each algorithm produces per unit time. Where unit time is defined as the time that the RLS algorithm produces one hundred samples. This table indicates that the proposed method has three times faster throughput rate than the RLS algorithm. On the other hand, PIPKAL has the lower throughput rate than the RLS algorithm.

Table 1: Throughput of each algorithm ($L = M = 3$)

RLS	PIPKAL	proposed
100	85	276

5. Conclusion

This paper proposed the algorithm that enables us to higher throughput implementation of the RLS algorithm using fewer PEs. The proposed method can achieve L times higher throughput rate as the RLS algorithm without growing worse of the rate of convergence. As an example of fewer PEs, we simulated the proposed method using DSP processors.

Acknowledgment

This work was supported in part by the Japan Society for the Promotion of Science under Grand-in-Aid for Scientific Research 11650389 and by Texas Instruments Japan.

References

[1] J.Thomas,"Pipelined systolic architectures for DLMS adaptive filtering,"Journal of VLSI Signal Processing, vol.12, pp.223-246, 1996.

[2] S.C.Douglas, Q.Zhu, and K.F.Smith,"A pipelined architecture for LMS adaptive FIR filter architecture without adaptation delay,"IEEE Trans. Signal Processing, vol.46, pp.775-779, Mar. 1998.

[3] K.Matsubara, K.Nishikawa, and H.Kiya,"Pipelined LMS adaptive filter using a new lookahead transformation,"IEEE Trans. Circuits Syst 2, vol.46, no.1, pp.51-55, 1999.

[4] A.Harada, K.Nishikawa, and H.Kiya,"A pipelined architecture for normalized LMS adaptive digital filters,"IEICE Trans. Fundamental, vol.82-A, pp.223-229, Feb. 1999.

[5] S.Haykin,"Adaptive Filter Theory",Englewood Cliffs,NJ 07632: Prentice Hall,2nd ed.,1991.

[6] K.J.Raghumath and K.K.parhi,"Pipelined RLS adaptive filtering using scaled tangent rotations(STAR),"IEEE Trans. Signal Processing, vol.44, pp.2591-2604, Oct. 1996.

[7] K.J.Raghumath and K.K.Parhi,"Finiteprecision error analysis of QRD-RLS nad STAR-RLS adaptive filters,"IEEE Trans. Signal Processing, vol.45, pp.1193-1209, May. 1997.

[8] Naresh R.Shanbhag,Keshab K.Parhi,"Pipelined Adaptive Digital Filters", Chapter 8, Kluwer Academic Publishers, 1994.

[9] Kiyoshi Nisikawa,Hitoshi Kiya,"Fast Implementation Technique for improving throughput of RLS Adaptive filters", ICASSP-2000 (Istanbul), Jun. 2000.

[10] Kiyoshi Nisikawa,Hitoshi Kiya,"Fast Implementation Technique for Improving Throughput of RLS Adaptive Filters", IEICE trans. Fundamentals, Aug. 2000.

[11] Kiyoshi Nisikawa,Hitoshi Kiya,"Novel Implementation Technique of RLS Algorithm for Improving Throughput of Adaptive Filters", EUSIPCO-2000 (Tampere), Sep. 2000.

[12] Kiyoshi Nishikawa,Hitoshi Kiya,"Low Computational Complexity Implementation of Pipelined RLS adaptive filters", ECCTD '99, Volume2, pp.1399.

[13] Takeo Niki, Kiyoshi Nishikawa, Hitoshi Kiya,"Low Computational Complexity Implementation of Pipelined RLS Filters", IEICE General Conference, A-4-62, Mar. 1999.