A NEW PIPELINED ARCHITECTURE OF THE LMS ALGORITHM WITHOUT DEGRADATION OF CONVERGENCE CHARACTERISTICS

Katsushige Matsubara, Kiyoshi Nishikawa, and Hitoshi Kiya

Dept. of Electronics and Information Engineering
Tokyo Metropolitan University
1-1 Minami Osawa, Hachioji, Tokyo, 192-03 JAPAN
Tel: +81 426 77 2745 Fax: +81 426 77 2756
E-mail: kiyoshi@eei.metro-u.ac.jp

ABSTRACT

This paper proposes an adaptive algorithm, which can be pipelined, as an extension of the delayed least mean square (DLMS) adaptive algorithm. The proposed algorithm provides a capability to achieve high throughput with less degradation of the convergence characteristic than the DLMS algorithm. An architecture for pipelined implementation of the proposed algorithm is considered, and based on this, the conditions for the implementaion are derived. An efficient implementation of the architecture with less hardware is also considered.

1. INTRODUCTION

This paper proposes an adaptive algorithm that can be pipelined as an extension of the delayed least mean square (DLMS) algorithm. Based on the proposed algorithm, we can achieves the pipelined adaptive filters which provides high throughput with less degradation of convergence characteristics.

The pipelined implementation of adaptive algorithms to achieve high throughput of the adaptive filters is considered so far. Several architectures of the RLS (recursive least squares) algorithm are considered[1] because its recursive structure is well-suited to implementation with systolic arrays. In such an implementation, however, each array requires complex operations (e.g., division or square-root calculation) so the implementation is complicated and this complication makes it hard to implement sufficiently long filters required in applications. The pipelined implementation of the LMS algorithm has therefore been considered by several researchers because of its simple structure. The original LMS algorithm, however, cannot be pipelined because it has a recursive loop in its filter update formula[2]. The algorithm has therefore been modified to enable its pipelined implementation, and two modified algorithms have been proposed so far: the relaxed look-ahead pipelined LMS (PIPLMS) algorithm[2], and the DLMS algorithm[3]. Although the DLMS can be considered a special case of the relaxed look-ahead technique, it is superior in terms of the convergence characteristics. Therefore, the DLMS is preferred in applications.

The DLMS achieves pipelined implementation of the LMS by inserting delays between taps of an adaptive filter. Although the DLMS can be pipelined, its convergence characteristics become worse as the amount of delay D increases. To reduce the amount of delay, a new architecture for the DLMS has been proposed[4][5]. Although this reduction improves the convergence characteristics, it causes some degradation of the throughput. Therefore, a method for improving the convergence characteristics without decreasing the amount of delay is needed to satisfy both the requirements for high throughput and good convergence.

We propose a new pipelined architecture based on the proposed algorithm that is an extended version of the DLMS algorithm. The convergence characteristics of this new algorithm can be improved independently of the amount of delay while maintaining high throughput. A method for implementing the proposed algorithm is also proposed, and techniques for reducing the required hardware are considered.

2. THE PROPOSED ALGORITHM

In this section we describe the proposed adaptive algorithm and show that the proposed algorithm achieves high throughput with less degradation of the convergence.

2.1. Problem of the DLMS algorithm

First, we point out the problem of the DLMS algorithm. The DLMS enables the hardware implementation of the LMS by inserting the delay D in the error feedback path, as shown in Fig. 1. The DLMS[3] is given as

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu e(n-D)\boldsymbol{x}(n-D) \quad (1)$$

$$e(n-D) = d(n-D) - \boldsymbol{x}^T(n-D)\boldsymbol{w}(n-D), \quad (2)$$

where μ is the step size parameter. $\boldsymbol{w}(n)$ is the tap-weight vector and $\boldsymbol{x}(n)$ is the tap-input vector of the adaptive filter:

$$\boldsymbol{w}(n) = [w_0(n), w_1(n), \cdots, w_{N-1}(n)]^T$$

$$\boldsymbol{x}(n) = [x(n), x(n-1), \cdots, x(n-N+1)]^T$$

where N is the length of the filter and ^{T} indicates the transpose of a vector. We can realize the pipelined implementation of the DLMS algorithm by replacing the delay D using the retiming technique[6].

When the DLMS is pipelined, higher throughput can be achieved as D increases under a given filter length N. Thus, in the DLMS, the amount of delay D is selected as $D = N$ to realize the highest throughput under N. However, on the other hand, this increasing of D narrows the selectable range for μ. The convergence characteristic of the DLMS therefore depends on the length of filter N. That is, convergence becomes poor as N increases.

2.2. Derivation of the Proposed algorithm

Next we derive the proposed algorithm. We consider reducing the effect of the delay by adding a new term to (2). We obtain the next equation by substituting (2) into (1):

$$\begin{aligned} \boldsymbol{w}(n+1) &= \boldsymbol{w}(n) + \mu d(n-D)\boldsymbol{x}(n-D) \\ &\quad -\mu \boldsymbol{x}(n-D)\boldsymbol{x}^T(n-D)\boldsymbol{w}(n-D). \end{aligned} \tag{3}$$

The effect of the delay D is expressed in the time lag between $\boldsymbol{w}(n)$ in the first term and $\boldsymbol{w}(n-D)$ in the third term in the right-hand side of (3). We can say that reduction of the effect of D is achived if D in $\boldsymbol{w}(n-D)$ can be decreased. For that purpose, we propose to expand $\boldsymbol{w}(n-D)$ using δ pairs of $e(n)\boldsymbol{x}(n)$. From (1), we can express $\boldsymbol{w}(n-D)$ as

$$\begin{aligned} \boldsymbol{w}(n-D) &= \boldsymbol{w}(n-(D-\delta)) \\ &\quad -\sum_{i=0}^{\delta-1} \mu e(n-2D+i)\boldsymbol{x}(n-2D+i). \end{aligned} \tag{4}$$

Using $\boldsymbol{w}(n-(D-\delta))$ in (4), we can rewrite (2) as

$$e(n-D) = \varepsilon_\delta(n-D) + \Lambda_\delta(n). \tag{5}$$

where $\varepsilon_\delta(n-d)$ and $\Lambda_\delta(n)$ is given by

$$\begin{aligned} \varepsilon_\delta(n-D) &= d(n-D) \\ &\quad -\boldsymbol{x}^T(n-D)\boldsymbol{w}(n-(D-\delta)) \end{aligned} \tag{6}$$

$$\begin{aligned} \Lambda_\delta(n) &= \sum_{i=0}^{\delta-1} \mu e(n-2D+i) \\ &\quad \cdot \boldsymbol{x}^T(n-D)\boldsymbol{x}(n-2D+i). \end{aligned} \tag{7}$$

By adding $\Lambda_\delta(n)$ to $e(n-D)$, we can generate the error sigal $\varepsilon_\delta(n-D)$, which can decrease the delay from D to $D-\delta$. Note that the expansion method in (4) through (7) can be regarded as a new look-ahead transformation.

Using $\varepsilon_\delta(n-D)$ as the error signal, we propose a new algorithm. The filter update formula of the proposed algorithm is given as

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu\varepsilon_\delta(n-D)\boldsymbol{x}(n-D), \tag{8}$$

where

$$\begin{aligned} \varepsilon_\delta(n-D) &= d(n-D) \\ &\quad -\boldsymbol{x}^T(n-D)\boldsymbol{w}(n-D) - \Lambda_\delta(n). \end{aligned} \tag{9}$$

In this equation δ is the parameter that enables us to control the convergence property of the algorithm and its value must be selected as in the range

$$0 \le \delta \le D. \tag{10}$$

By adjusting the parameter δ we can control the convergence characteristic of the algorithm from that of the DLMS algorithm to that of the LMS algorithm. Namely, when $\delta = D$ the algorithm reduces to the LMS algorithm and when $\delta = 0$ it reduces to the DLMS algorithm. Note that the case of $\delta = D$ is already derived in [7] and the proposed algorithm therefore includes [7] as a special case. The equivalent signal flow graph of the proposed algorithm is shown in Fig. 2.

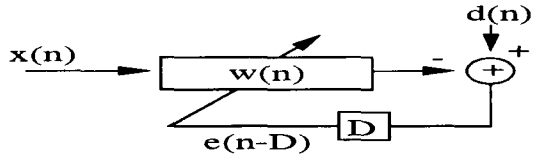

Figure 1: Equivalent structure of the DLMS algorithm: D shows the amount of delay.

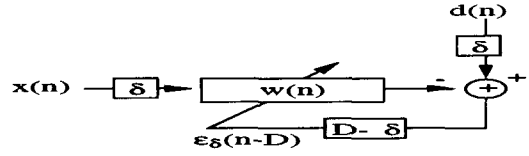

Figure 2: Equivalent structure of the proposed algorithm: D shows the amount of delay and δ is the parameter for adjusting the convergence characteristic.

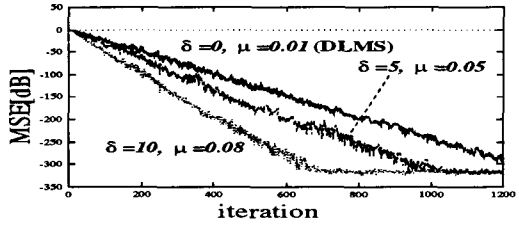

Figure 3: Convergence characteristics of the proposed algorithm with various values for δ.

2.3. Simulation

Here we show simulation results to demonstrate the effectiveness of the proposed algorithm. We simulated a system identification problem with an unknown system of tap length 10. We set the length N of the adaptive filter as $N = 10$ and set the delay $D = 10$. Three different values were used for δ: $\delta = 0$, 5, and 10. Note that $\delta = 0$ corresponds to the DLMS algorithm and $\delta = 10$ corresponds to the original LMS algorithm. In Fig 3, the simulation results are shown. From the figure, it is known the convergence improves as δ increases.

3. PROPOSED IMPLEMENTATION METHOD

This section describes the pipeline implementation of the algorithm proposed in the previous section. As can be seen from (7), the calculation of $\Lambda(n)$ does not depend on $\boldsymbol{w}(n)$, and $\boldsymbol{w}(n)$ can be pipelined using the conventional techniques, e.g., the DLMS algorithm. Therefore what we need to consider is the pipeline implementation of $\Lambda(n)$.

3.1. Pipeline implementation of $\Lambda(n)$

Fig. 4 depicts the signal flow graph for calculating $\Lambda(n)$. As shown in this figure, the architecture can be divided into two parts, A and B. Part A is for calculating the products $\boldsymbol{x}(n-D-k)\boldsymbol{x}(n-D-k-i)$, and part B is for summing up the products of multiplication of those results by $e(n)$.

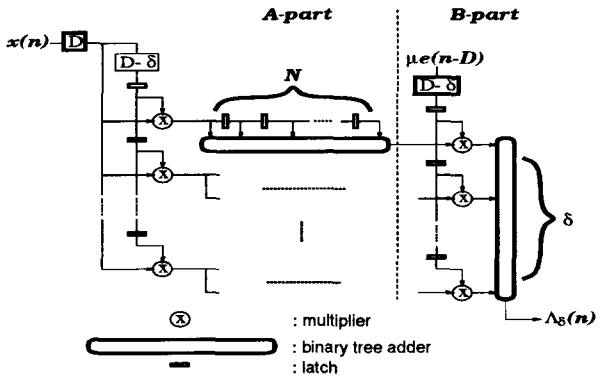

Figure 4: Signal flow graph for calculating $\Lambda(n)$.

For pipelining the circuit we use the delays in both parts, D and $D-\delta$. By applying the retiming technique[2] to these delays, as in the DLMS, we can achieve the pipelined implementation of the circuit shown in Fig. 4. In this case, however, the condition for the parameter δ will be different from (10) as is shown in the following.

3.2. The condition on δ for pipeline implementation

We define the times required for one multiplication and one addition as τ_m and τ_a, respectively, and we assume $\tau_m = \alpha\tau_a$, where α is a real number. The times required to calculate for parts A and B are expressed as

$$\tau_{AP} = (\alpha + \lceil \log_2 N \rceil)\tau_a \quad (11a)$$

$$\tau_{BP} = (\alpha + \lceil \log_2 \delta \rceil)\tau_a. \quad (11b)$$

Let t_{dt} is the given required clock rate for the filter. To satisfy t_{dt}, the critical path t_{cp} of the filter should be in the range

$$t_{cp} \leq t_{dt}. \quad (12)$$

For satisfy this condition, the delays in two parts A and B are replaced using the retiming technique.

The following amounts of delay are required for pipelining parts A and B,

$$D_A = \left\lceil \frac{(\alpha + N - 1)\tau_a}{t_{dt}} \right\rceil \quad (13a)$$

$$D_B = \left\lceil \frac{(\alpha + \delta - 1)\tau_a}{t_{dt}} \right\rceil \quad (13b)$$

where $\lceil x \rceil$ is the smallest integer larger than x. D_A and D_B are the required amount of delay for pipelining parts A and B. From (13), we can see that D in part A and $D-\delta$ in part B must satisfy the following conditions:

$$D \geq D_A + D_B \quad (14a)$$

$$(D-\delta) \geq D_B. \quad (14b)$$

In the DLMS algorithm the amount of delay D is fixed as $D = N$, so the above two conditions are reduced to the following condition:

$$\delta \leq D - D_B = N - D_B. \quad (15)$$

This is the condition for δ to achieve the pipelined implementation of the proposed algorithm. This condition shows that the selectable range of the parameter δ becomes narrower than (10) when the proposed algorithm is pipelined. This shows that the algorithm proposed in [7] cannot be pipelined because it does not satisfy (15). Note that, under condition (14), we can select D as smaller than N using the architecture proposed in [4] if the resulting throughput can satisfy the required specification. Then δ must be selected according to the first term in the right-hand side of (15).

Under conditions (14) and (15), the values for δ and D must be determined according to the required specifications and the acceptable amount of hardware. Once the values for δ and D are determined, the pipeline implementation of the proposed algorithm will be obtained by inserting the delays D and $D-\delta$ between the functional units in both A and B of Fig. 4.

4. EFFICIENT IMPLEMENTATION

The amount of hardware required in the direct implementation of the proposed architecture is larger than that of the DLMS because of the extra calculation for $\Lambda(n)$. Here we will consider a method for reducing the amount of hardware required for its implementation.

4.1. Scheduling of operators

We use the scheduling technique to reduce the amount of hardware required. Note that in the following we assume to use the operators, which have pipelined structures, for efficient calculations.

First let us consider the COs in both the whole of part A and the multiplications in part B. In these parts, the identical calculations are done independently. We schedule the COs in these parts as follows:

1. The required time t_m for processing each pipeline latch is calculated as

$$t_m = \left\lceil \frac{(2\alpha + \lceil \log_2 N \rceil + \lceil \log_2 \delta \rceil)\tau_a}{D} \right\rceil. \quad (16)$$

2. Let R be the number of reuses of each operator.
3. The critical path in this case is given as

$$t_{cp} = t_m + (R-1)\tau_a. \quad (17)$$

4. If t_{cp} satisfies the condition $t_{cp} \leq t_{dt}$ we can use the scheduling. Otherwise we should repeat the process with either decreasing R or increasing D.

The numbers of multipliers h_m and adders h_{a1} required for pipelining are given as

$$h_m = 2\left\lceil \frac{\delta}{R} \right\rceil \quad (18)$$

$$h_{a1} = (N-1)\left\lceil \frac{\delta}{R} \right\rceil \quad (19)$$

Figure 5 shows an example of $t_m = 5$ and $R = 2$ where one multiplier and six adders are used twice.

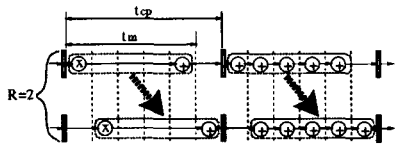

Figure 5: Reusing of the operators.

Now we consider the additions in the part B, where processing time is expressed as $\lceil \log_2 \delta \rceil \tau_a$ and where the number of stages required for pipelining is therefore given as

$$L_{pip} = \left\lceil \frac{\lceil \log_2 \delta \rceil \tau_a}{t_{dt}} \right\rceil. \quad (20)$$

In each stage the number of adders required is equal to the number of additions in the first step and we can reuse the adders in the next additions. Therefore, the number h_{a2} of adders required in this part is

$$h_{a2} = \sum_{i=0}^{L_{pip}-1} \left\lfloor \frac{\delta}{2^i} \right\rfloor, \quad (21)$$

where $\lfloor x \rfloor$ is the largest integer smaller than x.

4.2. The effect of scheduling on the performance

From the description in the previous section, the required numbers of multipliers and adders to calculate $\Lambda_\delta(n)$ are given as

$$h_m = 2\left\lceil \frac{\delta}{R} \right\rceil \quad (22)$$

$$h_a = (N-1)\left\lceil \frac{\delta}{R} \right\rceil + \sum_{i=0}^{L_{pip}-1} \left\lfloor \frac{\delta}{2^i} \right\rfloor. \quad (23)$$

Let us define the total cost of hardware as

$$C = C_m h_m + C_a h_a, \quad (24)$$

where C_m is the cost for one multiplier and C_a is the cost of one adder. As the index of the performance of the filter, we use the upper limit of the applicable clock rate

$$S = 1/t_{cp}. \quad (25)$$

Figure 6 show the relation between the clock rate S and the hardware cost C. From the results of the synthesis using PARTHENON[8], the costs C_m and C_a were determined as $C_m = 20$ and $C_a = 1$. Other conditions are $N = 30$, $\delta = 20$, and $t_m = 5$. From Fig. 6, we can see that the clock rate falls only 20% if we reduce the hardware cost by 50%.

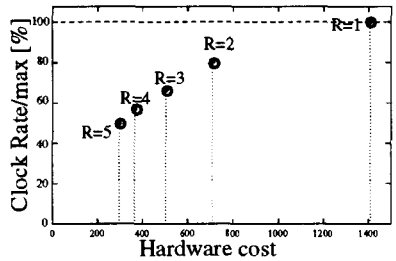

Figure 6: Clock rate vs. hardware cost

5. CONCLUSION

In this paper, we proposed an adaptive algorithm that can be pipelined. It is shown that the proposed algorithm provides a capability to achieve high throughput with less degradiation of the convergence characteristics than the DLMS algorithm. An implementation method of the proposed algorithm was considered and the condition for the proposed algorithm to implement in pipelined fashion was derived.

6. REFERENCES

[1] K. J. Raghunath and K. K. Parhi, "A 100 MHz pipelined RLS adaptive filter," in *Proc. IEEE ICASSP'95*, pp. 3187–3190, 1995.

[2] N. R. Shanbhag and K. K. Parhi, *Pipelined Adaptive Digital Filters*. Massachusetts: Kluwer Academic Publishers, 1994.

[3] G. Long, F. Ling, and J. G. Proakis, "The LMS algorithm with delayed coefficient adaptation," *IEEE Trans. Acoust., Speech & Signal Process.*, vol. 37, pp. 1397–1405, Sept. 1989.

[4] K. Matsubara, K. Nishikawa, and H. Kiya, "Pipelined adaptive filters based on delayed LMS algorithm," *IEICE Trans. (in Japanese)*, vol. J79-A, May 1996.

[5] K. Matsubara, K. Nishikawa, and H. Kiya, "Pipelined adaptive filters based on two-dimensional LMS algorithm," in *Proc. ITC-CSCC*, (Seoul, Korea), pp. 832–835, 1996.

[6] C. E. Leiserson, F. Rose, and J. Saxe, "Optimizing synchronous circuitry by retiming," in *Proc. of the Third Caltech Conf. on VLSI*, pp. 87–116, 1983.

[7] R. D. Poltmann, "Conversion of the delayed LMS algorithm into the LMS algorithm," *IEEE Signal Proc. Letters*, vol. 2, Dec. 1995.

[8] Y. Nakamura, K. Oguri, A. Nagoya, M. Yukishita, and R. Nomura, "High-level synthesis design at NTT systems labs," *IEICE Trans. Inf. & Syst.*, vol. E76-D, pp. 1047–1054, Sept. 1993.