EFFICIENT PARALLEL IMPLEMENTATION OF THE LMS ALGORITHM ON A MULTI-ALU ARCHITECTURE

Wayne T. Padgett

Agere Systems,
StarCore® Technology Center, Atlanta, GA USA

ABSTRACT

A block exact formulation [1] of the LMS algorithm is well suited for efficient implementation on multiple-ALU architectures. The method can be applied to any number of ALUs. An example analysis is given for the StarCore SC140 core which shows a 33% speed increase for large filters.

1. INTRODUCTION

It can be very difficult to implement the least mean square (LMS) adaptive filter algorithm efficiently on a digital signal processor (DSP) with multiple arithmetic logic units (ALUs). Parallelism is hard to achieve when the filter coefficients change for each sample.

Benesty and Duhamel [1] have shown how to formulate the exact LMS algorithm in a block form and rearrange the computations to reduce the total number of multiplies and adds. The block exact form holds the filter constant during a block and then corrects for updates within the block. This form allows the LMS algorithm to be tailored to the resources of a multi-ALU processor without the data rearrangement. Optimizing the utilization of processor resources can lead to large performance increases even without reducing the number of multiplies and adds.

Because the block form can be derived for any block size, this technique is applicable to any number of ALUs in a custom system on a chip (SOC), or to various generic commercial multi-ALU DSPs. An example analysis with source code is presented for the StarCore SC140 DSP core. For large numbers of filter taps the resulting method translates to a 33% increase in performance over a traditional technique. `Matlab` and assembly code for algorithm implementations are available on the web [2].

2. IMPLEMENTATION CONSTRAINTS

A multi-ALU DSP is designed to compute several multiply or multiply-accumulate (MAC) operations in parallel. This requires the ability to move data and instructions in parallel to keep up with the ALUs. Wide data buses for moving operands in parallel bring data alignment problems with them, which may reduce algorithm performance. Most DSP architectures are capable of circular addressing to avoid data movement in delay buffers, but data alignment can cause problems.

In an FIR filter, several taps can be computed at a time, but the delay buffer needs to shift one sample, not several. Since most DSPs are highly optimized for FIR filter computation, a variety of techniques have been developed to deal with the problem of needing to shift a circular buffer by a single sample when the data has a bus width of several words.

2.1. Standard Techniques

Multiplexing the incoming and outgoing data to allow "misaligned" loads can solve the problem, but this requires extra hardware, and will incur a speed or silicon area penalty. If the goal is to maximize performance, mis-aligned loads should be avoided.

Multi-sample programming [3] involves simply computing several outputs at a time so that the delay buffer also needs to shift several outputs at a time. A side benefit of this technique is that many data values can be reused, reducing the required data bus bandwidth and power requirements. Unfortunately, the LMS algorithm changes the filter for each output sample so that multiple outputs can't be computed at the same time.

Multiple-loop programming involves writing several copies of the filter code, one for each possible data alignment, and using the correct one for each filter output. This method has two disadvantages: it increases code size by roughly the same multiple as the number of ALUs, and it tends to require more registers because it needs to realign the data in the processor.

Multiple-buffer programming is a variant of multiple loop programming which stores the four possible alignments of the delay buffer in memory. This avoids the need for extra registers, but it shares the code size penalty and also multiplies the data buffer storage memory requirement by the number of ALUs.

2.2. SC140 Architecture

The StarCore SC140 has four ALUs and can compute four MACs in one cycle. To support the ALUs, it has two data buses, each of which can move four 16 bit operands to or from memory in a cycle. Peak performance is most important in the inner-loops of an algorithm. Like many DSPs, the SC140 has a zero overhead loop to maximize efficiency in

Wayne T. Padgett is on sabbatical leave from Rose-Hulman Inst. of Tech. Email: Wayne.Padgett@Rose-Hulman.edu . He would like to acknowledge the insightful comments and suggestions of the StarCore Applications Team, especially Mao Zeng, Joe Monaco, Kevin Shay, and Stephen Dew.

the inner-loops. To optimize performance, four MACs must be executed every cycle if the process is arithmetic limited, or two four operand moves must be executed every cycle if the process is data movement limited. The processor uses a VLIW architecture with a variable length execution set (VLES) capable of six instructions each cycle, four ALU instructions, and two address-unit (move) instructions. Each VLES may contain one to six instructions.

Move operations can be implemented for a single operand, pairs of operands, or groups of four operands. The addressing modes available are flexible and include direct, indirect, indirect plus index, modulo, and reverse-carry modes. Misaligned loads are not available on the SC140 to avoid the performance penalties mentioned above.

3. BLOCK EXACT LMS FORMULATION

The goal of the block formulation of the LMS is to convert the computation from one that involves a new filter at each sample time, to one that uses a fixed filter for an appropriate number of samples (four in the case of the SC140), followed by an update correction process. This has the advantage of making the LMS into an efficiently implementable fixed FIR filter, plus a small correction. The data alignment problems are removed, with only a small overhead due to the corrections. Because of the overhead, this technique is most beneficial for filters with large numbers of taps and small block sizes. A typical phone-line echo canceller needs enough taps to benefit substantially.

The BELMS algorithm [1] can be derived as follows: let the input values $x[n]$ and the filter coefficients $h_l[n]$ be defined in terms of column vectors:

$$\begin{aligned} X[n] &= [x[n], x[n-1], x[n-2], \ldots x[n-L+1]]^T \\ H[n] &= [h_0[n], h_1[n], h_2[n], \ldots h_{L-1}[n]]^T . \end{aligned} \tag{1}$$

Then the LMS algorithm can be written with the error output $e[n]$ and the new filter coefficients $H[n+1]$ given by

$$\begin{aligned} e[n] &= d[n] - X^T[n]H[n] \\ H[n+1] &= H[n] + \mu e[n]X[n] \end{aligned} \tag{2}$$

where $d[n]$ is the desired filter output. Note that the standard FIR filter computation $y[n] = X^T[n]H[n]$ is implicit in these equations.

If four delayed versions of these equations are written, for $e[n]$, $e[n-1]$, $e[n-2]$, and $e[n-3]$ so that we obtain three more equations of the form

$$\begin{aligned} e[n-3] &= d[n-3] - X^T[n-3]H[n-3] \\ H[n-2] &= H[n-3] + \mu e[n-3]X[n-3], \end{aligned} \tag{3}$$

then the definition of $H[n]$ in terms of $H[n-1]$ can be substituted into (2), followed by the definition of $H[n-1]$ in terms of $H[n-2]$ until only $H[n-3]$ appears in all the equations for $e[n]$, $e[n-1]$, $e[n-2]$, and $e[n-3]$. The result of this procedure is

$$\begin{aligned} e[n] &= d[n] - X^T[n]H[n-3] + \\ &\quad \mu e[n-3]X^T[n]X[n-3] + \\ &\quad \mu e[n-2]X^T[n]X[n-2] + \\ &\quad \mu e[n-1]X^T[n]X[n-1] \\ e[n-1] &= d[n-1] - X^T[n-1]H[n-3] - \\ &\quad \mu e[n-3]X^T[n-1]X[n-3] - \\ &\quad \mu e[n-2]X^T[n-1]X[n-2] \\ e[n-2] &= d[n-2] - X^T[n-2]H[n-3] - \\ &\quad \mu e[n-3]X^T[n-2]X[n-3] \\ e[n-3] &= d[n-3] - X^T[n-3]H[n-3] \end{aligned} \tag{4}$$

This procedure produces the desired fixed filter formulation, but the outputs can still only be computed one at a time, since each still depends on previous error values. This problem is resolved by inverting a matrix to remove the $e[n]$ values from the right hand side. First, the (4) is rewritten in matrix form using the substitution $s_p[n-q] = \mu X^T[n-q]X[n-q-p]$. The result is

$$\begin{aligned} \begin{bmatrix} e[n-3] \\ e[n-2] \\ e[n-1] \\ e[n] \end{bmatrix} &= \begin{bmatrix} d[n-3] \\ d[n-2] \\ d[n-1] \\ d[n] \end{bmatrix} - \begin{bmatrix} X^T[n-3] \\ X^T[n-2] \\ X^T[n-1] \\ X^T[n] \end{bmatrix} H[n-3] \\ &\quad - \begin{bmatrix} 0 & 0 & 0 & 0 \\ s_1[n-2] & 0 & 0 & 0 \\ s_2[n-1] & s_1[n-1] & 0 & 0 \\ s_3[n] & s_2[n] & s_1[n] & 0 \end{bmatrix} \begin{bmatrix} e[n-3] \\ e[n-2] \\ e[n-1] \\ e[n] \end{bmatrix} \end{aligned} \tag{5}$$

Fortunately, (5) can be rewritten more compactly as

$$\mathbf{e}[n] = \mathbf{d}[n] - \mathbf{X}[n]H[n-3] - \mathbf{S}[n]\mathbf{e}[n]. \tag{6}$$

Then, (6) can be rearranged as

$$(\mathbf{I} + \mathbf{S}[n])\mathbf{e}[n] = \mathbf{d}[n] - \mathbf{X}[n]H[n-3] \tag{7}$$

and then

$$\mathbf{e}[n] = (\mathbf{I} + \mathbf{S}[n])^{-1}(\mathbf{d}[n] - \mathbf{X}[n]H[n-3]). \tag{8}$$

Inverting the four by four matrix by hand is not too difficult, but even larger matrices (block sizes) are practical since the matrix is always lower triangular [1]. The final result for $\mathbf{G}[n] = (\mathbf{I} + \mathbf{S}[n])^{-1}$ with block size four is shown in (9) on the next page. Finally, the BELMS can be expressed as the computation of four samples of a fixed FIR filter, $\mathbf{X}[n]H[n-3]$, with the resulting errors corrected by the fairly simple multiplication by $\mathbf{G}[n]$. While the computation of the values of $s_p[n-q]$ may appear to remove any benefit, these values can actually be computed recursively. Therefore, neither the cost of computing $\mathbf{G}[n]$ nor the cost of the matrix multiply depend on the number of filter taps.

4. PERFORMANCE COMPARISON

Three algorithms are considered: the single-sample LMS (LMS-SS), the four loop LMS (LMS-4L), and the BELMS. The LMS-SS is similar to the standard LMS code given in

$$\mathbf{G}[n] = (\mathbf{I} + \mathbf{S}[n])^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -s_1[n-2] & 1 & 0 & 0 \\ \begin{pmatrix} -s_2[n-1]+ \\ s_1[n-2]s_1[n-1] \end{pmatrix} & -s_1[n-1] & 1 & 0 \\ \begin{pmatrix} -s_3[n]+ \\ s_1[n-2]s_2[n]+ \\ s_1[n]s_2[n-1]- \\ s_1[n-2]s_1[n-1]s_1[n] \end{pmatrix} & \begin{pmatrix} -s_2[n]+ \\ s_1[n]s_1[n-1] \end{pmatrix} & -s_1[n] & 1 \end{bmatrix}. \quad (9)$$

the StarCore Filter Library [2], although it has been optimized slightly. The LMS-4L is the LMS implemented in four loops, one for each data alignment. The BELMS algorithm is the LMS rearranged as described above. Note that the BELMS is *not* equivalent to the well-known block LMS – the block LMS only updates once per block, while the BELMS computes an entire block, then corrects the error outputs as if they had been updated at each sample time.

Fig. 1 shows the number of cycles required by each algorithm for 16, 32, 64, 128, 256, and 512 taps. This plot was generated using a simulator with a realistic memory model, so memory conflicts can occur and cause some accesses to take more than one cycle. Therefore, the theoretical numbers do not match the simulation results perfectly. Table 1 gives a detailed comparison of various algorithm features. The specific features and important numbers are discussed further below.

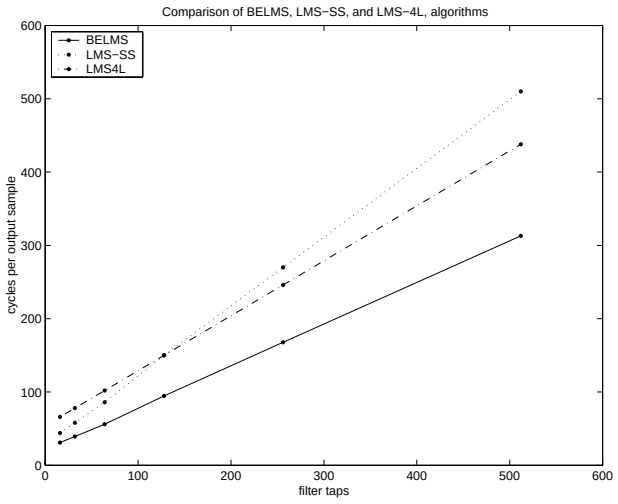

Fig. 1. A Comparison of the algorithms for various numbers of filter taps.

4.1. LMS-SS

The single sample LMS (LMS-SS) is the most straightforward method. It consists of two parts, the standard FIR filter to generate a new output and error value, and the coefficient update. Because of data alignment issues, the LMS-SS cannot use a circular buffer. The shifting of the data buffer is done inside the FIR loop. The FIR loop requires one cycle to MAC four delay values with four coefficients, and uses two moves to bring in five delay values (four are used). A second cycle does four register transfers to realign the data and two moves to bring in four new coefficients and write four shifted values to the delay buffer. So the FIR loop processes four taps in two cycles for $T/2$ cycles per output sample where T is the number of filter taps.

Table 1. Algorithm Comparison

	LMS-SS	LMS-4L	BELMS
Best Possible	$\frac{T}{2}+\frac{3T}{8}=\frac{7T}{8}$	$\frac{T}{4}+\frac{3T}{8}=\frac{5T}{8}$	$\frac{T}{4}+\frac{T}{4}=\frac{T}{2}$
Implemented	$\frac{T}{2}+\frac{3T}{8}=\frac{7T}{8}$	$\frac{T}{4}+\frac{T}{2}=\frac{3T}{4}$	$\frac{T}{4}+\frac{T}{4}=\frac{T}{2}$
Simulated	$0.9422T$	$0.75T$	$0.5705T$
Overhead	28.0498	54	21.131
Code Size (bytes)	266	1082	1370
FIR % issue	100%	100%	100%
FIR % BW	81%	100%	24%
FIR MACS/cyc	2	4	4
update % issue	78%	58%	96%
update % BW	100%	75%	19%
update MACS/cyc	2.67	2	4

The coefficient update loop is limited by the number of reads and writes, not by the arithmetic. To maximize the data bus bandwidth, it uses two four-register sets as accumulators, processing eight coefficients in three cycles. This achieves the maximum of six four-operand moves in three cycles, but requires T to be a multiple of eight.

The LMS-SS should be able to perform at $0.875T$ cycles plus overhead, but memory conflicts in the simulation increased the number of cycles per output to $0.9422T$. Note that the code size is smallest for this method and it performs well for small numbers of filter taps.

Table 1 includes several more rows describing the following algorithm characteristics:

- "% issue" the percentage of maximum instructions issued per cycle (maximum six possible),
- "% BW" the percentage of maximum operands moved per cycle (maximum eight possible), and
- MACs/cyc the number of MACs executed per cycle (maximum four possible).

These quantities are given for both the FIR loops and the coefficient update loops of each algorithm.

4.2. LMS-4L

The four loop LMS (LMS-4L) is a better comparison for the BELMS algorithm for large filter sizes, since it can take

advantage of modulo addressing. It also has a similar code size to the BELMS algorithm. Although the overhead appears much larger for LMS-4L than the other two methods, it could probably be reduced somewhat. The parameter of interest is the increase in cycles per tap. The switch to modulo addressing allows the LMS-4L to achieve four taps per cycle, $T/4$, in the FIR loop, maximizing the resources available (instructions issued, data bandwidth, and MACs).

The LMS-4L does data alignment in registers in both loops. In the coefficient update loop, the extra registers used for data alignment are not available to keep two copies of the coefficient accumulators as is done in the LMS-SS. Therefore, the LMS-4L requires eight cycles to process four taps, or $T/2$ cycles for the update loop. This problem could be resolved by going to a four buffer approach, but only with the associated data memory penalty. The four buffer approach was not implemented, but its theoretical performance is shown in Table 1 as "Best Possible" for the LMS-4L. Note that the utilization numbers are low for the update loop of the LMS-4L. The simulated cycle counts in Fig. 1 are averaged over the four loops.

4.3. BELMS

Because the BELMS algorithm operates in blocks of four output samples, it is a good fit for the four-ALU SC140. As noted above, the BELMS algorithm can be re-derived for any desired block size or number of ALUs. The FIR loop can be implemented using a multi-sample algorithm since the filter is fixed for four samples. Even though the loop processes four taps per cycle, the data bandwidth is only 24% of the maximum, reducing power consumption. In the update loop, four coefficient corrections are computed at time, relieving the data movement limitation, so that the loop is now arithmetic limited. The processor can handle four coefficient updates per cycle, doing four MACs every cycle, again with reduced data bandwidth.

The result is that for filters large enough to neglect overhead, the BELMS method achieves $T/2$ cycles per output, while the next best choice LMS-4L only achieves $3T/4$ cycles per output, a 33% improvement.

This improvement is reduced to about 24% in the simulation due to differences in memory conflicts. Because the BELMS allows reuse of coefficients, it maintains a 20% performance advantage even over the memory intensive four buffer method ($T/2$ vs. $5T/8$).

The BELMS implementation is done on a frame of 40 samples at a time, so overhead values are averaged over 40 outputs. Also, the computations of the autocorrelation values must be computed once before recursion starts, and this is done once per frame in this implementation. The one non-recursive computation depends on T, but can be divided over as large a frame size as desired given frame latency constraints.

5. CONCLUSIONS

The BELMS algorithm allows the LMS to be customized for a multi-ALU architecture. Given typical hardware constraints, the BELMS is the most computationally efficient algorithm available for large numbers of filter taps. Because of the optimal use of processor resources, the BELMS algorithm produces a 33% increase in performance over a four loop LMS method for the StarCore SC140. This performance improvement is possible for a variety of telecommunications systems without any hardware modifications.

6. REFERENCES

[1] J. Benesty and P. Duhamel, "A fast exact least mean square adaptive algorithm," *IEEE Trans. on SP*, vol. 40, no. 12, pp. 2904–2920, Dec. 1992.

[2] Agere Systems and Motorola, Inc., StarCore Technology Center, Atlanta, GA, *Source Code*, Dec. 2001, `http://www.starcore-dsp.com/docs/articles.html` .

[3] Agere Systems and Motorola, Inc., StarCore Technology Center, Atlanta, GA, *Multisample Programming Technique*, Dec. 2001, SC140MLTAN/D, `http://www.starcore-dsp.com/docs/index.html` .

[4] D. Manolakis, V. Ingle, and S. Kogon, *Statistical and Adaptive Signal Processing*, McGraw-Hill, 2000.