Applications of Distributed Arithmetic to Digital Signal Processing:
A Tutorial Review


For Advanced VLSI Course 10-9-2002
Outline:

- Distributed Arithmetic (DA) Introduction
- Technical overview of DA
- Increasing the speed of DA multiplication
- Application of DA to a biquadratic digital filter
- Conclusions
Distributed Arithmetic (DA, 1974)

Introduction

- The most-often encountered form of computation in DSP:
  - Sum of product
  - Dot-product
  - Inner-product
  - Executed most efficiently by DA

- By careful design one may reduce gate count in a signal processing arithmetic unit by a number seldom smaller than 50% and often as large as 80%
Technical overview of DA

- Advantage of DA: efficiency of mechanization
- A frequently argued:
  - slowness because of its inherent bit-serial nature (not true)
- Some modifications to increase the speed by employing techniques:
  - Plus more arithmetic operations
  - expense of exponentially increased memory
The sum of product: \[ y = \sum_{k=1}^{K} A_k x_k \]

- where \( x_k \) is a 2’s-complement binary number scaled such that \(|x_k|<1\), and \( A_k \) is fixed coefficients

- \( x_k : \{ b_{k0}, b_{k1}, b_{k2} \ldots \ldots, b_{k(N-1)} \} \), wordlength=N

  - where \( b_{k0} \) is the sign bit

  - Express each \( x_k \) as: \[ x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \]

Sub (2) into (1) => \[ y = \sum_{k=1}^{K} A_k \left[ -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \right] \]
Critical step

\[ y = \sum_{k=1}^{K} A_k \left[ -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \right] \]

\[ \iff y = \sum_{n=1}^{N-1} \left[ \sum_{k=1}^{K} A_k b_{kn} \right] 2^{-n} + \sum_{k=1}^{K} A_k (-b_{k0}) \]

- where \( K = \text{No. of taps (inputs)} \), \( N = \text{Wordlength of Data} \)
Consider the equation (4)

\[ y = \sum_{n=1}^{N-1} \left[ \sum_{k=1}^{K} A_k b_{kn} \right] 2^{-n} + \sum_{k=1}^{K} A_k (-b_{k0}) \]

- \[ \sum_{k=1}^{K} A_k b_{kn} \] has only \(2^K\) possible values
- \[ \sum_{k=1}^{K} A_k (-b_{k0}) \] has only \(2^K\) possible values
- We can store it in a lookup-table (ROM): size = \(2 \times 2^K\)
Technique overview -continued IV

- **Example**
  - Let no. of taps $K=4$
  - The fixed coefficients are $A_1 = 0.72$, $A_2 = -0.3$, $A_3 = 0.95$, $A_4 = 0.11$

\[
y = \sum_{n=1}^{N-1} \left( \sum_{k=1}^{K} A_k b_{kn} \right) 2^{-n} + \sum_{k=1}^{K} A_k (-b_{k0})
\]

- We need $2 \times 2^K = 32$-word ROM ($k=4$)
Example

- Unfolding

\[
\sum_{k=1}^{4} A_k b_{kn} = A_1 b_{1n} + A_2 b_{2n} + A_3 b_{3n} + A_4 b_{4n}
\]

\[
\sum_{k=1}^{4} A_k (-b_{k0}) =
A_1 (-b_{1,0}) + A_2 (-b_{2,0}) + A_3 (-b_{3,0}) + A_4 (-b_{4,0})
\]
Example - continued I

- Hardware architecture

![Diagram of a 32 Word ROM with inputs and outputs showing an example configuration.]

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Input Code</th>
<th>32-Word Memory Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_0$</td>
<td>$b_{1n}$</td>
<td>$b_{2n}$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$A_0 + A_2 + A_3 = -A_4$
Example -continued II

- Shorten the table

\[
\sum_{k=1}^{4} A_k b_{kn} = A_1 b_{1n} + A_2 b_{2n} + A_3 b_{3n} + A_4 b_{4n}
\]

\[
\sum_{k=1}^{4} A_k (-b_{k0}) = - \left[ \sum_{k=1}^{4} A_k (b_{k0}) \right]
\]

- Eq. (4)

\[
\Rightarrow \quad y = \sum_{n=1}^{N-1} \left[ \sum_{k=1}^{K} A_k b_{kn} \right] 2^{-n} - \sum_{k=1}^{K} A_k (b_{k0})
\]
Example -continued II

- Hardware architecture

Only 16 words of ROM are required, now.
Technique overview - advanced

**Input**

\[
x_k = \frac{1}{2} [x_k - (-x_k)] = \{b_{k0}, b_{k1}, \ldots, b_{k(n-1)}\}
\]

\[
x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n}
\]

\[
-x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} + 2^{-(N-1)}
\]

\[
x_k = \frac{1}{2} \left[ -\left(b_{k0} - \bar{b}_{k0}\right) + \sum_{n=1}^{N-1} \left(b_{kn} - \bar{b}_{kn}\right) 2^{-n} - 2^{-(N-1)} \right]
\]
Technique overview - advanced I

\[ x_k = \frac{1}{2} \left[ -(b_{k0} - \bar{b}_{k0}) + \sum_{n=1}^{N-1} (b_{kn} - \bar{b}_{kn}) 2^{-n} - 2^{-(N-1)} \right] \]

\[ c_{kn} = \begin{cases} b_{kn} - \bar{b}_{kn}, & n \neq 0 \\ -(b_{kn} - \bar{b}_{kn}), & n = 0 \end{cases} \quad \text{where} \quad c_{kn} \in \{-1, 1\} \]

\[ x_k = \frac{1}{2} \left[ \sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)} \right] \quad \text{代入} \quad y = \sum_{k=1}^{K} A_k x_k \]

\[ y = \frac{1}{2} \sum_{k=1}^{K} \left[ \sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)} \right] = \sum_{n=0}^{N-1} Q(b_n) 2^{-n} + 2^{-(N-1)} Q(0) \]

\[ Q(b_n) = \sum_{k=1}^{K} \frac{A_k}{2} c_{kn} \]

\[ Q(0) = -\sum_{k=1}^{K} \frac{A_k}{2} \]
Technique overview - advanced II

- Hardware architecture

<table>
<thead>
<tr>
<th>Input Code</th>
<th>b₁n</th>
<th>b₂n</th>
<th>b₃n</th>
<th>b₄n</th>
<th>8-Word Memory Contents, Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1/2 (A₁ + A₂ - A₃ - A₄) = -0.74</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1/2 (A₁ + A₂ - A₃ + A₄) = -0.63</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1/2 (A₁ + A₂ - A₃ - A₄) = 0.21</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1/2 (A₁ + A₂ - A₃ - A₄) = 0.32</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1/2 (A₁ - A₂ + A₃ - A₄) = -1.04</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1/2 (A₁ - A₂ + A₃ - A₄) = -0.93</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1/2 (A₁ - A₂ + A₃ - A₄) = -0.09</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1/2 (A₁ - A₂ + A₃ - A₄) = 0.02</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/2 (A₁ - A₂ - A₃ - A₄) = -0.02</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1/2 (A₁ - A₂ - A₃ + A₄) = 0.09</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1/2 (A₁ - A₂ + A₃ - A₄) = 0.93</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1/2 (A₁ - A₂ + A₃ + A₄) = 1.04</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/2 (A₁ + A₂ - A₃ - A₄) = -0.32</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1/2 (A₁ + A₂ - A₃ + A₄) = -0.21</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1/2 (A₁ + A₂ + A₃ - A₄) = 0.63</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1/2 (A₁ + A₂ + A₃ + A₄) = 0.74</td>
</tr>
</tbody>
</table>
Increase the speed of DA multiplication

- The way I:
  - Plus more arithmetic operations

\[
 y = \sum_{n=0}^{N-1} Q(b_n)2^{-n} + 2^{-(N-1)} Q(0)
\]

\[
\sum_{n=0}^{N-1} Q(b_n)2^{-n} = Q(b_0)2^{-0} + Q(b_1)2^{-1} + \ldots + Q(b_{N-2})2^{-(N-2)} + Q(b_{N-1})2^{-(N-1)}
\]
Increase the speed of DA multiplication

- Hardware architecture of way I
  - at the expense of linearly increased memory & arithmetic operation

\[ y = \sum_{n=0}^{N-1} Q(b_n)2^{-n} + 2^{-(N-1)} Q(0) \]
Increase the speed of DA multiplication

- The way II:
  - at the expense of exponentially increased memory
    - ROM: 2*7 words => 1*128 words

- Hardware architecture
Application of DA to biquadratic digital filter

- Transfer function of a typical biquadratic digital filter:
  \[
  \frac{Y(z)}{X(z)} = \frac{A_0 + A_1 z^{-1} + A_2 z^{-2}}{1 + B_1 z^{-1} + B_2 z^{-2}}
  \]

- The time-domain description:
  \[
  Y(z) = [A_0 \ A_1 \ A_2 - B_1 - B_2 \ [x_n \ x_{n-1} \ x_{n-2} \ y_{n-1} \ y_{n-2}]]^T
  \]
Application of DA to a biquadratic digital filter

- Hardware architecture

<table>
<thead>
<tr>
<th>Table 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Code</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 1 0</td>
</tr>
<tr>
<td>0 0 0 1 1</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
</tr>
<tr>
<td>0 0 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1 0</td>
</tr>
<tr>
<td>0 0 1 1 1</td>
</tr>
<tr>
<td>0 1 0 0 0</td>
</tr>
<tr>
<td>0 1 0 0 1</td>
</tr>
<tr>
<td>0 1 0 1 0</td>
</tr>
<tr>
<td>0 1 0 1 1</td>
</tr>
<tr>
<td>0 1 1 0 0</td>
</tr>
<tr>
<td>0 1 1 0 1</td>
</tr>
<tr>
<td>0 1 1 1 0</td>
</tr>
<tr>
<td>0 1 1 1 1</td>
</tr>
</tbody>
</table>
Application of DA to biquadratic digital filter

- The vector matrix equation (state-space form)

\[
\begin{bmatrix}
    u_n \\
v_n \\
y_n
\end{bmatrix} = \begin{bmatrix}
a_1 & b_1 & c_1 \\
a_2 & c_2 & b_2 \\
a_0 & d_1 & d_2
\end{bmatrix} \begin{bmatrix}
x_{n-1} \\
u_{n-1} \\
v_{n-1}
\end{bmatrix}
\]

- The relationship between Eq. (B.1) and (B.2) is

\[
\begin{align*}
A_0 &= a_0 \\
A_1 &= a_1 d_1 + a_2 d_2 - a_0 (b_1 + b_2) \\
A_2 &= a_0 (b_1 b_2 - c_1 c_2) + a_1 (c_2 d_2 - b_2 d_1) + a_2 (c_1 d_1 - b_1 d_2) \\
B_1 &= b_1 + b_2 \\
B &= -b_1 b_2 + c_1 c_2
\end{align*}
\]
Application of DA to a biquadratic digital filter

- Normal form biquadratic filter
Application of DA to a biquadratic digital filter (DA realization with smaller ROM)
Application of DA to a biquadratic digital filter

- Increase speed

1Bit*3 BAAT DA structure

4Bit*3 BAAT DA structure
Application of DA to a biquadratic digital filter

- Increase speed II
  - 2048 x 48 bit word ROM
  - Require 4 identical ROM to process 16-bit in parallel
  - 4-input addition operations
Application of DA to a biquadratic digital filter

- Increase speed III
  - 32 words/ROM
  - Require 8 smaller ROM
  - With larger Adder Unit
    (8-input add operations)
  - Trade-off between
    Arithmetic Operators and
    Memory Unit.
Conclusions

- DA is a very efficient mechanism for computations that are dominated by inner products (convolution).
- If performance/cost ratio is critical, DA should be seriously considered as a contender.
- When many computing methods are compared, DA should be considered. It is not always (but often) best, and never poorly.