Applications of Distributed Arithmetic to Digital Signal Processing:

A Tutorial Review


VSP Lecture 11 Distributed Arithmetic
(cwliu@twins.ee.nctu.edu.tw)
Distributed Arithmetic (DA, 1974)

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

• 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

• \( x_k: \{b_{k0}, b_{k1}, b_{k2}, \ldots, b_{k(N-1)}\}, \text{wordlength}=N \)
  – 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) => 
  \[ y = \sum_{k=1}^{K} A_k \left[ -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \right] \]
Derivation of DA Technique
-continued I

• Critical step

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

\[ = \sum_{n=1}^{N-1} \left[ \sum_{k=1}^{K} A_k b_{kn} \right] 2^{-n} + \sum_{k=1}^{K} A_k (-b_{k0}) \quad (4) \]

• where \( K \) is the number of inputs (or taps) and \( N \) is the wordlength of Data
Derivation of DA Technique
-continued II

- Now 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=\(2 \times 2^K\)
Technical Overview of DA

• Advantage of DA: Efficiency of computing mechanization

• A frequently argued:
  – Slowness because of its inherent \textit{bit-serial nature} (not true)

• Some modifications to increase the speed by employing techniques:
  – Plus more arithmetic operations
  – Expense of exponentially increased memory
Derivation of DA Technique -continued III

• Example
  – Let number of inputs $K=4$
  – 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

- Hardware architecture

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

• Shorten the table

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

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

– Eq. (4)

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

(5)
Example -continued II

Only 16 words of ROM are required now.
Offset-Binary Coding (OBC)

- Change Input data from “binary” to “signed-digit”

\[
x_k = \frac{1}{2} \left[ x_k - (-x_k) \right] = \{b_{k0}, b_{k1}, \ldots, b_{k(n-1)}\} \tag{6}
\]

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

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

\[
x_k = \frac{1}{2} \left[ -\left( b_{k0} - \bar{b}_{k0} \right) + \sum_{n=1}^{N-1} \left( b_{kn} - \bar{b}_{kn} \right) 2^{-n} - 2^{-(N-1)} \right] \tag{7}
\]
Offset-Binary Coding (OBC) (cont’d)

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

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

\[ x_k = \frac{1}{2} \left[ \sum_{n=0}^{N-1} c_{kn} 2^{-n} - 2^{-(N-1)} \right] \]

代入 \( y = \sum_{k=1}^{K} A_k x_k \)

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

Where \( Q(b_n) = \sum_{k=1}^{K} \frac{A_k}{2} c_{kn} \) and \( Q(0) = -\sum_{k=1}^{K} \frac{A_k}{2} \) Constant
Offset-Binary Coding (OBC) (cont’d)

- Hardware architecture

\[
\begin{align*}
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 0000 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 0001 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 0010 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 0011 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 0100 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 0101 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 0110 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 0111 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 1000 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 1001 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 1010 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 1011 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 1100 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 1101 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 1110 \\
\sum b_{1n} b_{2n} b_{3n} b_{4n} &= 1111
\end{align*}
\]

<table>
<thead>
<tr>
<th>Input Code</th>
<th>Memory Contents, Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>(-\frac{1}{2}(A_1 + A_2 + A_3 + A_4) = -0.74)</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>(-\frac{1}{2}(A_1 + A_2 + A_3 - A_4) = -0.63)</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>(-\frac{1}{2}(A_1 + A_2 - A_3 + A_4) = 0.21)</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>(-\frac{1}{2}(A_1 + A_2 - A_3 - A_4) = 0.32)</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>(-\frac{1}{2}(A_1 - A_2 + A_3 + A_4) = -1.04)</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>(-\frac{1}{2}(A_1 - A_2 + A_3 - A_4) = -0.93)</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>(-\frac{1}{2}(A_1 - A_2 - A_3 + A_4) = -0.09)</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>(-\frac{1}{2}(A_1 - A_2 - A_3 - A_4) = 0.02)</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>(\frac{1}{2}(A_1 - A_2 - A_3 - A_4) = -0.02)</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>(\frac{1}{2}(A_1 - A_2 - A_3 + A_4) = 0.09)</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>(\frac{1}{2}(A_1 - A_2 + A_3 - A_4) = 0.93)</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>(\frac{1}{2}(A_1 - A_2 + A_3 + A_4) = 1.04)</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>(\frac{1}{2}(A_1 + A_2 - A_3 - A_4) = -0.32)</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>(\frac{1}{2}(A_1 + A_2 - A_3 + A_4) = -0.21)</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>(\frac{1}{2}(A_1 + A_2 + A_3 - A_4) = 0.63)</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>(\frac{1}{2}(A_1 + A_2 + A_3 + A_4) = 0.74)</td>
</tr>
</tbody>
</table>
Speed up of DA multiplication

- **Way I**: Plus more arithmetic operations

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

Initial condition

\[
\sum_{n=0}^{N-1} Q(b_n)2^{-n} = Q(b_0)2^{-0} + Q(b_1)2^{-1} + \ldots + Q(b_{N-2})2^{-(N-2)} + Q(b_{N-1})2^{-(N-1)}
\]

Even part

Odd part
Speedup of DA multiplication

- Way I: at the expense of linearly increased memory & arithmetic operation

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

- Way II: at the expense of exponentially increased memory
- ROM: 2*7 words → 1*128 words
Conclusions

• DA is a very efficient mechanism for computations that are dominated by inner products (convolution).

• A good way to trade combinational logic with memory for high-performance computation.

• When a many computing methods are compared, DA should be considered. It is not always (but often) best, and never poorly: save gate count around 50% to 80%.