Left-to-Right Arithmetic Paradigm: Computing while Communicating - Terminate Gracefully at Any Moment

DDECS2020 Keynote Presentation

Prof. Miloš D. Ercegovac
milos@cs.ucla.edu
http://web.cs.ucla.edu/~milos/
Computer Science Department
University of California at Los Angeles
Introductory Comment

Computer arithmetic has always played an important and critical role in the design of general-purpose and special-purpose processors. It has been an active research area since the early days of computers (1950s), albeit relatively small compared to other areas in computer architecture and computer science. Computer arithmetic, in brief, investigates theoretical/hardware/software aspects in number representation, arithmetic algorithms for basic operations, function evaluation, in fixed-point and floating-point arithmetic. Computer arithmetic is a synergy of applied mathematics, algorithms, digital design, VLSI implementation, software and compilers, and applications. Every new generation of processors, such as GPUs, super-scalar and multi-core processors, digital signal processors, and, most recently, processors for AI and machine learning, rely on progress in computer arithmetic to increase performance and reduce energy, and cost.
Professor Ercegovac earned his PhD ('75) and MS ('72) in computer science from the University of Illinois, Urbana-Champaign, and BS in electrical engineering ('65) from the University of Belgrade, Serbia. He specializes in research and teaching in digital arithmetic, digital design, and computer system architecture. His recent research is in the areas...
of approximate arithmetic, composite algorithms, complex arithmetic, design for low power and arithmetic in application-specific architectures. His research contributions have been extensively published in journals and conference proceedings. He is a coauthor of two textbooks on digital design and of a monograph and a book in the area of digital arithmetic. Dr. Ercegovac has been involved in organizing the IEEE Symposia on Computer Arithmetic since 1978. He served as an associate editor of the IEEE Transactions on Computers and as a subject area editor for the Journal of Parallel and Distributed Computing. He received a Medal of Ecole Normale Superieure de Lyon, France, in 2015, a Distinguished Alumni Educator Award in 2013 from the Department of Computer Science, University of Illinois Urbana-Champaign, and Lockheed-Martin Corporation Excellence in Teaching Award in 2009. Dr. Ercegovac is a foreign member of the Serbian Academy of Sciences and Arts, Fellow and Life Member of the IEEE, and a member of the ACM.
MAIN IDEA of LR and ONLINE ARITHMETIC

- Why compute less significant before more significant digits?
- Compute while communicating: why waste time waiting for operands?
- Latency parameter: online delay $\delta$ - a delay before the most significant digit of a result appears. A small integer.
LEFT-TO-RIGHT AND ONLINE ARITHMETIC

- The inputs applied and output delivered one digit at a time - serially. All digits of the same operands/results share the same digit lines. The systems are usually clocked so that one digit is applied per clock cycle. It is possible to have some operands in parallel form.

- There are two serial modes: Least-significant digit first (LSDF) and most-significant digit first (MSDF)

- In left-to-right arithmetic the result is generated most significant digit first (MSDF mode) and one or more operands are in digit-parallel form and one or more operands are in digit-serial form.

- In on-line arithmetic the results and the operands are processed one-digit at a time, most significant digit first (MSDF mode)
DIGIT-LEVEL OVERLAP BASIS FOR PERFORMANCE

Operation:

- Squaring
- Addition
- Square root
- Division

(a)

\[ x_h, y_h \]

\[ a_i, b_i \]

\[ i = h - \delta_1 - 1 \]

\[ \Delta = p \]

\[ f_k, k = i - \delta_2 - 1 \]

\[ \Delta = p \]

\[ g_p, p = k - \delta_3 - 1 \]

\[ \Delta = p \]

\[ c_q, q = p - \delta_4 - 1 \]

\[ s_q \]
ONLINE ALGORITHM MODEL

- Operands \( x \) and \( y \), result \( z \): \( n \) radix-\( r \) digits, redundant

- In cycle \( j \) the result digit \( z_{j+1} \) is computed

- Cycles labeled \(-\delta, \ldots, 0, 1, \ldots, n\)

- In cycle \( j \) receive the operand digits \( x_{j+1+\delta} \) and \( y_{j+1+\delta} \), and output \( z_j \)

- \( x[j], y[j] \) and \( z[j] \) are the numerical values of the corresponding signals when their representation consists of the first \( j + \delta \) digits for the operands, and \( j \) digits for the result.
In cycle $j$

**online operands**

- $x[j + 1] = (x[j], x[j+1+\delta])$
- $y[j + 1] = (y[j], y[j+1+\delta])$

**result digit**

- $z_{j+1} = F(w[j], x[j], x[j+1+\delta], y[j], y[j+1+\delta], z[j])$

**online result**

- $z[j + 1] = (z[j], z_{j+1})$

**residual**

- $w[j + 1] = G(w[j], x[j], x[j+1+\delta], y[j], y[j+1+\delta], z[j], z_{j+1})$
\[ F; G \]
\[ X; Y \]
\[ x[j] \]
\[ y[j] \]
\[ w[j+1] \]
\[ w[j] \]

(a) Digit-serial
(b) Digit-parallel

Input
Output
Online delay ($\delta$)

– critical parameter: determines the throughput (no pipelining)

<table>
<thead>
<tr>
<th>Operation</th>
<th>LSDF</th>
<th>MSDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>0</td>
<td>2 ($r = 2$)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 ($r \geq 4$)</td>
</tr>
<tr>
<td>Multiplication</td>
<td>0</td>
<td>3 ($r = 2$)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 ($r = 4$)</td>
</tr>
<tr>
<td>(only MS half)</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>Division</td>
<td>$2n$</td>
<td>4</td>
</tr>
<tr>
<td>Square root</td>
<td>$2n$</td>
<td>4</td>
</tr>
<tr>
<td>Max/min</td>
<td>$n$</td>
<td>0</td>
</tr>
</tbody>
</table>
GENERIC FORM OF EXECUTION AND IMPLEMENTATION.

- Execution: \( n + \delta \) iterations of the recurrence, each one clock cycle

- Iterations (cycles) labeled from \(-\delta\) to \( n - 1\)

- One digit of each input introduced during cycles \(-\delta\) to \( n - 1 - \delta\) and digits value 0 thereafter

- Result digits 0 for cycles \(-\delta\) to \(-1\) and \( z_1 \) is produced in cycle 0

- Result digit \( z_j \) is output in cycle \( j \) (one extra cycle to output \( z_n \))
The actions in cycle $j$:

- Input $x_{j+1+\delta}$ and $y_{j+1+\delta}$.

- Update $x[j + 1] = (x[j], x_{j+1+\delta})$ and $y[j + 1] = (y[j], y_{j+1+\delta})$ by appending the input digits.

- Compute $v[j] = rw[j] + H_1$

- Determine $z_{j+1}$ using the selection function.

- Update $z[j + 1] = (z[j], z_{j+1+\delta})$ by appending the result digits.

- Compute the next residual $w[j + 1] = v[j] + H_2(z_{j+1})$

- Output result digit $z_j$
IMPLEMENTATION

• Similar structure in all online algorithms:

  → all implemented with the same basic components, including

  (i) Registers to store operands, results, and residual vectors;

  (ii) Multiplication of vector by digit - trivial in radix 2;

  (iii) Append units to append a new digit to a vector - concatenation;
(iv) Two-operand and multioperand redundant adders, such as signed digit adders, [3:2] carry-save adders and their generalization to [4:2] and [5:2] adders;

(v) Converters from redundant representations (i.e., signed digit and carry save) to conventional representations - on-the-fly converters (OTFC);

(vi) Carry-propagate adders of limited precision (3 to 6 bits) to produce estimates of the residual used in the digit-selection; and

(vii) Digit-selection schemes to obtain output digits - low precision.
Online vs. Conventional Arithmetic: Performance

- \( z = E(x) \), \( z \) is scalar, \( x \) argument vector of \( n \)-digit elements, \( E \) a scalar expression

- Network of \( L \) levels of non-pipelined units to evaluate \( E \)

- Conventional arithmetic: units at level \( i \) wait for all units at level \( i - 1 \) to finish

- Online arithmetic: unit at level \( i \) begins when \( \delta + 1 \) input digits are received
• Latency of a network of $L$ levels of online units:

$$T_{OL} \leq \left[ \sum_{i=1}^{L} (\delta_{max} + 1) + n \right] t_d$$

• Latency of a network of $L$ levels of conventional units:

$$T_{CONV} \leq \sum_{i=1}^{L} (T_{imax} + t_{LOAD})$$

$- t_{imax}$ time of slowest operation at level $i$; $t_{LOAD}$ time to transfer operands
SPEEDUP ANALYSIS

• Assume
  - $\delta_{imax} = 3$
  - $T_{imax} = cnt_d, c = 1$ for $T = O(n)$, $c = (\log n)^2/n$ for $T = O(\log^2 n)$
  - $t_{LOAD} = t_d$

  - Same number of units

  - Speedup

$$S = \frac{T_{CONV}}{T_{OL}} = \frac{L(cn + 1)}{n + 4L}$$
SPEEDUP ANALYSIS (cont)

- Minimum $L$ so that $T_{OL} < T_{CONV}$

$$L_{min} = \left\lceil \frac{n}{cn - 3} \right\rceil$$

Let $n = 32$ and $c = \frac{25}{32}$, then $L_{min} = 2$

- Speedup for large $L$

$$S \rightarrow (cn + 1)/\delta_{max}$$

Let $n = 32$ and $c = \frac{25}{32}$, then $S_{max} < 9$
SPEEDUP ANALYSIS (cont)

• Evaluation of vector expressions
  
  – $V$ vector operands, one vector result, each of $M$ elements ($n$ digits each)
  – Pipelined units: conventional with $N$ stages; online array type, also pipelined
  – $L$ network levels

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$c$</td>
<td>$n$</td>
<td>$N$</td>
<td>$M$</td>
<td>$S_p$</td>
</tr>
<tr>
<td>25/32</td>
<td>32</td>
<td>4</td>
<td>100</td>
<td>4.9</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>4</td>
<td>100</td>
<td>6.2</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>8</td>
<td>1000</td>
<td>3.9</td>
</tr>
<tr>
<td>1</td>
<td>64</td>
<td>4</td>
<td>1000</td>
<td>15.0</td>
</tr>
</tbody>
</table>
• For large number of operands, $M \to \infty$, the speedup is

$$S_p = \frac{c n}{N}$$

• For large number of levels, $L \to \infty$, the speedup is

$$S_p = \frac{c n}{4}$$

• The additional speedup due to online arithmetic: 2 to 16 for typical precision

• What about cost? From analysis,

$$C_{OL} < C_{CONV} \text{ if } C_{OL\text{-module}} \leq 2 \times C_{CONV\text{-module}}$$
WHY IS LR ARITHMETIC USEFUL?

In LR mode, we can start an operation before finishing the previous one. Implications:

- Overlap data transmission and computation
- Limit data dependency: a constant delay independent of $n$
  
  $\rightarrow$ Suitable for arithmetic-intensive applications with data dependencies (recursive computations)

- Reduce width of interconnections
Reduce signal activities - saving energy

- Required precision for selection, i.e., $\hat{v}^{(j)}$
- Residual precision requirement to satisfy selection criteria in last cycle.

Figure 2.4: Illustration of the required precision of on-line and reduced precision on-line residuals.

(a) Activity profile and required precision of implementation for an on-line unit not employing a reduced precision implementation, i.e., having $m$ radix-$r$ fractional digit precision.

(b) Because of the requirement on $\hat{v}^{(j)}$ for selection, a reduced precision implementation of the recurrence module, having $m^*$ radix-$r$ fractional digits of precision can be employed. The reduced precision implementation also benefits from reduced switching activity, by only utilizing just enough precision to determine the correct output.
2.2. SYNERGY OF ON-LINE AND CDR ARITHMETIC

Fig. 2.4: Illustration of the required precision of on-line and reduced precision on-line residuals $w_j$. (a) Activity profile and required precision of implementation for an on-line unit not employing a reduced precision implementation, i.e., having $m$ radix-$r$ fractional digit precision. (b) Because of the requirement on $\hat{v}_j$ for selection, a reduced precision implementation of the recurrence module, having $m^*$ radix-$r$ fractional digits of precision can be employed. The reduced precision implementation also benefits from reduced switching activity, by only utilizing just enough precision to determine the correct output.
• Easy support for variable precision computations: stop when sufficient accuracy achieved

• Zero-bias truncation instead of rounding: $1/2 \ ulp$ error

• Expose parallelism between all operations: overlap always possible

• LR model provides a basis for composite (fused) operations

• Simplifies both partitioning into modules and their designs
• Radix-$r$ digit slice is the basic element → design effort largely independent of precision

1*** paths for appending input digits
** left-shifted bits of the residual
*** the width of the MS slice depends on the selection function
• Reduced energy wrt serial-parallel operation: not all slices need to be active all the time
• Overall throughput is determined by the delay of a digit computation: digit-level pipelining possible
• LR arithmetic requires redundant representations

- e.g., signed-digit sets \{-1,0,1\} for \(r=2\), \{-2, -1,0,1,2\} for \(r=4\), carry-save \{0,1,2\}, etc

- higher cost/bit: 100% for binary, 50% for \(r=4\), 20% for \(r=16\), ... interestingly 0% for \(r=10\)

• On-the-fly conversion: Converts redundant LR (online) results into conventional representation without extra delay.

• Conventional digit-recurrence division, square root, function evaluators produce redundant results MSD first - compatible with LR arithmetic
In left-to-right carry-free (LRCF) multiplication (linear array):

- MS and LS parts of the product obtained in parallel with the reduction of partial products

- The final MS part of the product produced on-the-fly with carry-select adders

- This eliminates the delay of the final addition, typically around 20-30% of the total delay
Reduction of intermodule communication bandwidth:

- **Approach A:** Source module: compute result MSDF, convert on-the-fly, transmit in parallel to the destination module – *full precision communication*

- **Approach B:** Source module: compute result MSDF, transmit serially to the destination module during computation; convert to parallel using OTFC in the destination module – *serial communication*

Approaches A and B have the same total latency but vastly different communication cost.
Source Module \(\Rightarrow\) Conversion \(\Rightarrow\) Destination Module

- **Digit-serial**: \(\{-1, 0, 1\}\)
- **MSD first**

- **Digit-parallel conventional**
Composite (fused) Algorithms

- To reduce the overall online delay of a group of operations, several operations combined into a single *multi-operation online algorithm*

- Example: Sum of squares $x^2 + y^2 + z^2$; Inputs in $[1/2, 1)$, output in $[1/4, 3)$.

- Online delay $\delta_{ss} = 0$ when the output digit is over-redundant vs. (3+2+2=7) of a network of separate online operators

- Example: Givens rotation operator $y = \frac{x}{(x^2+y^2)^{1/2}}$

- Off-diagonal operations overlapped with computation of rotation factors
SUM OF SQUARES

1. [Initialize]
   \[ w[0] = x[0] = y[0] = z[0] = 0 \]

2. [Recurrence]
   \[ j = 0 \ldots n - 1 \]
   \[ v[j] = 2w[j] + (2x[j] + x_{j+1})x_j + (2y[j] + y_{j+1})y_j + (2z[j] + z_{j+1})z_j \]
   \[ w[j + 1] \leftarrow \text{csfrac}(v[j]) \]
   \[ s_{j+1} \leftarrow \text{csint}(v[j]) \]
   \[ x[j + 1] \leftarrow (x[j], x_{j+1}); \ y[j + 1] \leftarrow (y[j], y_{j+1}); \ z[j + 1] \leftarrow (z[j], z_{j+1}) \]
   \[ S_{out} \leftarrow s_{j+1} \]
   end for
Note: the fractional portion of the 5-2 CSA produces at most three carries
Higher Order LR Arithmetic: the E-Method

Instead of evaluating an expression, find equivalent system of linear equations, and use online operations to solve it. Typically, the first component of the solution is equivalent to the value of the expression. Specifically,

1. Transform an arithmetic expression into a system of linear equations \( L \):

\[
f(x) = E(x, p), \ p \text{ parameters}
\]

\[
f(x) \Rightarrow L : A \cdot y = b
\]

such that \( y_1 = f(x) \).
2. Solve the system using LR digit-by-digit vector recurrences in \( m \) steps for \( m \) digit result.

Typical recurrence: \( w_{2[j + 1]} = 2(w_{2[j]} - d_{2j} - q_1 \cdot d_{1j} + x \cdot d_{3j}) \)

3. Coefficient matrix corresponds to the matrix divisor and the right-hand side vector to the vector dividend. The quotient is the solution vector.

4. The elements of the solution vector are obtained in parallel starting with the most significant digits.

5. Redundancy makes the cycle time independent of precision and simplifies the selection of result digits.
Notation

- Matrices and vectors of elements in boldface: the coefficient matrix $\mathbf{A}$ of order $N$; the solution vector $\mathbf{y} = (y_1, \ldots, y_N)$; the right-hand side vector $\mathbf{b} = (b_1, \ldots, b_N)$.

- The residual vector at step $j$;

$$\mathbf{w}[j] = (w_1[j], \ldots, w_N[j])$$

(1)

- The result digit-vector at step $j$

$$\mathbf{d}[j] = (d_1[j], \ldots, d_N[j])$$

(2)

where digit $d_{kj} \in \{-1, 0, 1\}$ is the $j$-th digit of $y_k = \sum_{j=1}^{m} d_{kj} 2^{-j}$. 

---

38
E-METHOD ALGORITHM

1. [Initialize]
   \[ w[0] = b; \quad d[0] = 0 \]
2. [Recurrence]
   \[
   \begin{align*}
   \text{for } j = 0 \ldots m - 1 \\
   v[j] &= 2(w[j] - Ad[j]) \\
   d[j + 1] &\leftarrow SEL(v_{est}[j]) \\
   w[j + 1] &\leftarrow v[j] \\
   y_1[j + 1] &\leftarrow CONVERT(y_1[j], SEL(v_{est}[j]))
   \end{align*}
   \]
3. [Result]
   \[ y_1[m] \approx f(x) \]

– Corresponds to SRT division in vector form
where

- Residuals in redundant form, represented by the pseudo-sum $WS$ and stored-carry $WC$ bit-vectors.

- $SEL$ is the digit selection function

$$dk_{j+1} = SEL(vk_{est}[j]) = \begin{cases} 
1 & \text{if } vk_{est}[j] \geq 0.5 \\
0 & \text{if } -0.5 \leq vk_{est}[j] \leq 0 \\
-1 & \text{if } vk_{est}[j] \leq -1 
\end{cases}$$

where $vk_{est}[j]$ is the estimate of $vk[j]$ truncated to one fractional bit.

Note that the multiplications in the term $A \times d[j]$ are implemented as digit-vector by digit multipliers - trivial in radices 2 and 4.
Examples of mapping: a rational function $R_{2,3}(x)$ is mapped to the matrix/vector form:

\[
\begin{bmatrix}
1 & -x & 0 & 0 \\
q_1 & 1 & -x & 0 \\
q_2 & 0 & 1 & -x \\
q_3 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4
\end{bmatrix}
= \begin{bmatrix}
p_0 \\
p_1 \\
p_2 \\
0
\end{bmatrix}
\]

Solving the system produces $y$ such that:

\[
y_1 = R_{3,2}(x) = \frac{p_2x^2 + p_1x + p_0}{q_3x^3 + q_2x^2 + q_1x + 1}
\]

– Note: no division used in solving $L$
Similarly, a polynomial $P_3(x)$ is mapped to the following system

$$
\begin{bmatrix}
1 & -x & 0 & 0 \\
0 & 1 & -x & 0 \\
0 & 0 & 1 & -x \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4
\end{bmatrix}
=
\begin{bmatrix}
p_0 \\
p_1 \\
p_2 \\
p_3
\end{bmatrix}
$$

such that $y_1 = P_3(x) = p_3x^3 + p_2x^2 + p_1x + p_0$
EXAMPLE: EVALUATION OF A RATIONAL FUNCTION

- To evaluate the rational function $\sinh(x) \approx R_{3,4}(x)$ to $m$ bits, we iterate $m$ times.

- The coefficients are obtained from rational function approximation of $\sinh(x)$ in the interval $x \in [0, 1/6]$ with a relative error less than $10^{-13}$.

- To satisfy the bounds and to have $a_{1,1} = 1$, the original coefficients are divided by $q_0$. We restrict the argument $x$ to $[0, 1/8]$ and divide all normalized coefficients of $P$ by 2 to make them $\leq 3/4$. This scaling requires one additional iteration. In illustrating the algorithm, we show only the first 12 steps producing the first 13 bits of the solution. The approximation has a relative error of $2^{-45}$ after 46 steps.
We illustrate the algorithm for $m = 12$. The normalized coefficients, rounded to 12 bits, are shown in hexadecimal:

\[
\begin{align*}
p_3 &= 0.0d8 \\
p_2 &= 0.000 \\
p_1 &= 0.800 \\
p_0 &= 0.000 \\
q_4 &= 0.007 \\
q_3 &= 0.000 \\
q_2 &= -0.0fa \\
q_1 &= 0.000 \\
q_0 &= 1.000
\end{align*}
\]
The recurrences are

\begin{align*}
    w_1[j + 1] &= 2(w_1[j] - d_1[j] + x \cdot d_2[j]) \\
    w_2[j + 1] &= 2(w_2[j] - d_2[j] - q_1 \cdot d_1[j] + x \cdot d_3[j]) \\
    w_3[j + 1] &= 2(w_3[j] - d_3[j] - q_2 \cdot d_1[j] + x \cdot d_4[j]) \\
    w_4[j + 1] &= 2(w_4[j] - d_4[j] - q_3 \cdot d_1[j] + x \cdot d_5[j]) \\
    w_5[j + 1] &= 2(w_5[j] - d_5[j] - q_4 \cdot d_1[j])
\end{align*}

The initial residuals are

\[(w_1[0], w_2[0], w_3[0], w_4[0], w_5[0]) = (0, p_1, 0, p_3, 0)\]
THE NETWORK FOR EVALUATING RATIONAL FUNCTION

Module 1

Module 2

Module 3

Module 4

Module 5

On-the-Fly Converter

$x \quad 0$

$d_1 j$

$d_2 j$

$d_3 j$

$d_4 j$

$d_5 j$

$x \quad p_1 \quad 0$

$x \quad 0 \quad q_2$

$x \quad p_3 \quad 0$

$0 \quad q_4$

$p_1 \quad p_300 \quad q \quad 20q \quad 40$

parallel

serial

$R_{3,4}(x)$
THE COMPUTATION TRACE

Evaluation of $\sinh(0.10197)$ using rational approximation $R_{3,4}(x)$ and radix-2 E-method.

The error $|\sinh(x) - y_1[13]| < 2^{-12}$. $y_1[13]$ is computed to compensate for the initial scaling of $p$ coefficients by 2.

The evaluation of $R_{3,4}(x)$ for $x = 0.000110100001$ with 12-bit precision, showing non-redundant next residual $v_1$ (for simplicity). Other residuals are not shown.
<table>
<thead>
<tr>
<th>$j$</th>
<th>$v_1[j]$</th>
<th>$d_{1j+1}$</th>
<th>$d_{2j+1}$</th>
<th>$d_{3j+1}$</th>
<th>$d_{4j+1}$</th>
<th>$d_{5j+1}$</th>
<th>$y_1[j+1]*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000000000000000</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000000000000000</td>
</tr>
<tr>
<td>1</td>
<td>0.001101000001000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000000000000000</td>
</tr>
<tr>
<td>2</td>
<td>0.011010000010000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000000000000000</td>
</tr>
<tr>
<td>3</td>
<td>0.110100000100000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.001000000000000</td>
</tr>
<tr>
<td>4</td>
<td>-0.010111110000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.001000000000000</td>
</tr>
<tr>
<td>5</td>
<td>-0.101111100000000</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0.000110000000000</td>
</tr>
<tr>
<td>6</td>
<td>0.100001000000000</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0.000111000000000</td>
</tr>
<tr>
<td>7</td>
<td>-0.111110000000000</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0.000110100000000</td>
</tr>
<tr>
<td>8</td>
<td>0.000100000000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.000110100000000</td>
</tr>
<tr>
<td>9</td>
<td>0.001000000000000</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.000110100000000</td>
</tr>
<tr>
<td>10</td>
<td>0.011101000001000</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0.000110100010000</td>
</tr>
<tr>
<td>11</td>
<td>0.111010000010000</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.000110100001000</td>
</tr>
<tr>
<td>12</td>
<td>-0.011000111010000</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0.000110100010000</td>
</tr>
</tbody>
</table>
IMPLEMENTATION

SEL block produces estimate and performs selection
M block performs subtraction of \( d_{kj} \)

(initialized with coefficient \( q_{k-1} \))

(initialized with coefficient \( p_{k-1} \))

(register control signals not shown)
Example: FUSED EXPRESSION LR EVALUATION

\[ h = \frac{a(f + gc) + e(1 + cd)}{1 + ab + cd} \]

mapped to

\[
\begin{bmatrix}
1 & -a & 0 \\
b & 1 & -c \\
0 & d & 1
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3
\end{bmatrix}
= 
\begin{bmatrix}
e \\
f \\
g
\end{bmatrix}
\]

Solving the system produces \( y \) such that:

\[ y_1 = h \]
• Conventional:
  – Cost: 5 multiplications, 3 additions, 1 division; full interconnect
  – Time to evaluate $h$: $2t_{\text{mult}} + t_{\text{add}} + t_{\text{div}}$;

• Online:
  – Cost: 3 eqv. serial-parallel (SP) multiplications; serial interconnect
  – Time: $m$ iterations for $m$-bit result i.e., time of a single SP multiplication
LR iterative E-method vs Jacobi Method

Consider an $n$-th order system of linear equations

$$L : A \cdot y = (I - G) \cdot y = b$$

Classical Jacobi method solves $L$ by iterating

$$y[j] = b + G \cdot y[j - 1], \quad j = 1, 2, \ldots$$

This step requires $(n - 1)$ full precision multiplications and $(n - 1)$ full precision additions per row per step
The E-method solves $L$ by iterating

$$d[j] + z[j] = r(z[j - 1] + G \cdot d[j - 1]), \quad j = 1, 2, \ldots, m$$

where $d[j]$ is a vector of digits, $z[j]$ a vector of $m$-digit fractions, and $G$ is a matrix with $m$-digit fractions as coefficients.

Therefore, the step uses $(n - 1) m$-digit $\times$ single digit multiplications and $(n - 1)$ redundant additions.

– A significant reduction in cost, delay and energy
LR ARITHMETIC: SUMMARY OF FEATURES

+ Digit-in/digit-out, left-to-right model of computation (after online delay of $\delta$ steps, small integer)

+ Overlaps communication and computation

+ Exposes massive digit-level concurrency via overlap, masks serial nature of individual operations

+ Handles well deep data-dependent expressions and recursive algorithms: full result not needed to start next operation

+ Data-dependency penalty: online delay $\delta$ – it does not matter if linear or nonlinear systems
+ Inherently variable precision; can stop any time; truncated result with unbiased error

+ Modular design with digit-serial input/output between modules: economy of design effort

+ No time penalty for operand alignment in FLPT addition

+ Online algorithms implementation similar to implementation of digit-recurrence algorithms

+ Algorithms and implementations developed for most of basic arithmetic operations and for certain composite operations

+ Larger set of operations possible than with LSDF approach
+ Higher level operators possible: e.g., E-method for solving systems of linear equations

On the other hand:

- Requires redundant representations – higher cost for lower radix than conventional implementations

- Single operations are serial (serial-parallel equivalent)

- Online delay sensitive to MSD cancellation: rare event

- More complex design for mult, div, sqrt than conventional designs
LR ARITHMETIC - POTENTIAL USES

• Wide range of uses because online arithmetic is possible in all operations

• Flexible arithmetic design technique for accelerators

• In multipliers: Left-to-right carry-free multiplication (LRCF) avoids the final adder

• Applicable in the design of inner products, sum of products, sum of squares, convolutions

• In composite (fused) arithmetic algorithms for matrix multiplication, norms, and sparse matrix operations
• In low-precision and variable-precision arithmetic designs

• In function approximation using polynomials and rational functions: use the E-method. $m$ steps for $m$ digit precision, time independent of degree, cost proportional to degree

• In recursive computations, e.g., IIR filters, root finders.

• In convolutional neural networks, multilayer perceptrons, and in backpropagation

• Reconfigurable architectures - fused operations, minimal interconnect, and variable-precision

• Low energy arithmetic - minimal signal activity
Bibliography - Ercegovac and collaborators


[Selected papers authored or co-authored by M.D. Ercegovac and his students; covers Online Arithmetic and the E-Method]

PhD and MS Theses: at UCLA and U of Illinois


Journal papers

• M.D. Ercegovac and J.-M. Muller, An Efficient Method for Evaluating Complex Polynomials. Journal of Signal Processing Systems, Volume 58, Issue 1, Page 17, Springer 2010, also published online http://www.springerlink.com/content/5582844402n0t2x1/


M.D. Ercegovac and T. Lang. On-the-fly conversion of redundant into conventional


**Conference papers**


