# A General Hardware-Oriented Method for Evaluation of Functions and Computations in a Digital Computer

MILOŠ D. ERCEGOVAC

*Abstract*—**A parallel computational method, amenable for efficient hardware-level implementation, is described. It provides a simple and fast algorithm for the evaluation of polynomials, certain rational functions and arithmetic expressions, solving a class of systems of linear equations, or performing the basic arithmetic operations in a fixed-point number representation system. The time required to perform the computation is of the order of $m$ carry-free addition operations, $m$ being the number of digits in the solution. In particular, the method is suitable for fast evaluation of mathematical functions in hardware.**

*Index Terms*—**Arithmetic expressions, digital computer arithmetic, evaluation of real-valued functions, fixed-point representation, hardware-level implementation, integral powers, linear systems, on-line algorithms, parallel computation, polynomials, rational functions, redundant number systems.**

## I. Introduction

THE SUBJECT of this paper is a fast computational method amenable for efficient hardware-level implementation. It provides a viable alternative to the commonly known parallel algorithms implemented in multiprocessor systems, and to the strictly hardware-oriented algorithms, in which the fixed-point number representation system is used [1].

Fast algorithms are of basic interest in the theory of computation, and of an increasingly practical importance in the organization and design of computing systems. With respect to implementation, it is convenient to classify fast methods as 1) those that use a multiplicity of general-purpose processors with the corresponding algorithms specified on the software (instruction) level, and 2) those that require special-purpose processors with algorithms embedded in the hardware (operation) level. The first class is characterized by an unrestricted domain of applications, while the second class, limited to the special applications, provides a higher speed and better efficiency of implementation with respect to a given algorithm.

The evident progress of hardware technology does enhance the performance of the methods in both classes, but the problem of algorithm design, which is optimal with respect to the given technology and performance criteria, becomes even more challenging. When considering a computational method for possible implementation, one is usually concerned with 1) its application domain, 2) the required set of algorithms, and 3) the required set of primitive operators. With these, one also associates a set of desired or required properties, for instance, the speed, the complexity and the cost of implementation, the fault-tolerance capability, numerical characteristics of the algorithms, etc. Ideally, an implemented computational method should have as large a domain of application as possible, a single but simple algorithm, and only those primitive operators that are efficiently realizable in the given implementation technology.

The objective here is to define a method that would have a sufficient generality in applications and such functional properties in order to justify a hardware-level implementation. In other words, the intention is to combine the favorable properties of the two previously mentioned classes of computational methods.

One of the original motivations was the problem of fast evaluation of commonly used mathematical functions. The proposed method evolved while attempting to solve this problem in a new way. For that reason, we will restrict our attention in the remainder of this section to some of the known practical methods of evaluation of functions. One class of these methods is based on the classical approximation techniques, in particular, the minimax or near-minimax polynomial or rational approximations [2]. These methods, for all practical purposes, can be considered general. The other class contains those methods that are based on certain specific properties of the functions being evaluated: they are devised with implementation efficiency and speed as objectives, but they have a limited domain of application by definition [10]. For the former class, since the approximation problem can be assumed to be solved, one is concerned only with the problem of efficient evaluation of the corresponding approximating functions. For polynomials, a direct hardware-implemented evaluation scheme, based on Horner's or Clenshaw's recurrences, suitable for the summation of the Chebyshev series [2] in time with $0(n)$ multiplications, gives only a slight improvement in performance over the software versions, so that its implementation, except in microprogrammed machines, is hardly justified. The fast polynomial evaluation schemes, on the other hand, provide a significant gain in speed at the expense of a considerable hardware complexity. Tung [6] considered the implementation of an efficient polynomial evaluator, based on the fast primitive

Manuscript received January 10, 1976; revised January 3, 1977. This work was supported in part by National Science Foundation Grant DCR 73-07998 and in part by National Science Foundation Grant DCR 72-03633 A03. This paper was presented at the 3rd IEEE Symposium on Computer Arithmetic, Dallas, TX, November 1975.

The author was with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801. He is now with the Department of Computer Science, School of Engineering and Applied Science, University of California, Los Angeles, CA 90024.

operators and a redundant number representation. The structure proposed there is versatile but complex on the level of the basic building block as well as in the overall control and intercommunication requirements. For the rational functions, fast parallel schemes offer even less efficiency; yet, the rational approximations would be preferable in many instances. The method proposed here has the same generality in such an application, but offers a better performance. Its algorithm is simple and requires two or three operand additions as the primitive operator for the evaluation of polynomial or rational functions, respectively. The time required for evaluation is of the order of one carry-save-type multiplication time if the coefficients of the approximating functions satisfy certain range conditions. Otherwise, the required scaling will cause a logarithmic extension in the working precision. In addition to a simple basic computing block, the interconnection requirements of this method are much simpler than those of the previously mentioned methods. The main limitation is the restriction to the fixed-point number representation system.

The second class of methods being considered, as mentioned previously, includes those methods that utilize certain functional properties to achieve an efficient and fast implementation. Although these methods appear limited in application, some of them can generate enough different functions in order to justify a hardware implementation. Their efficiency comes from simple computational algorithms and primitive operators, like add and shift, which can be conveniently implemented. The proposed method is compatible in this respect: its computational algorithm is simple and problem invariant. There is no shift operator, which in the previously proposed methods must have a variable shift capability. When a redundant representation is introduced in order to increase the speed of computation, a variable shift operator can considerably affect the complexity of implementation. Some of the most important methods in this class are: Volder's coordinate rotation technique (CORDIC), described in [3] and later generalized in the form of a unified algorithm in [8]; the normalization methods, based on an iterative cotransformation of a number pair $(x,y)$, such that a function $f(x,y)$ remains invariant as proposed in [5], [7], [8]; and the pseudodivision and the pseudomultiplication methods for some elementary functions as described in [4]. A combination of some of these methods provides for fast evaluation of the most often used elementary functions (for instance, square roots, logarithms, exponentials, trigonometric, hyperbolic, and their inverse functions). The method proposed here is more versatile in this respect and it is generally comparable in speed. Although some of the methods previously mentioned use fewer implementation resources, the proposed method excels in simplicity with respect to the basic computing block, the overall structure, and its control.

Moreover, the proposed method can be applied in solving problems other than function evaluation. Certain arithmetic expressions, multiple products and sums, inner products, integral powers, and solving of systems of linear equations under certain conditions, are among the possible applications. Basic arithmetics, in particular, multiplication and division can be performed by this method. Furthermore, it has a useful functional property in that the results are generated in a digit-by-digit fashion with the most significant digits appearing first so that an overlap of computations can be utilized. The algorithm shares some properties with the incremental computations in the digital differential analyzers (DDA), although it is not based on an integration principle. The potential of such an algorithm, systematically using variable powers of two increments rather than constant increments as in DDA, and its possible application in implementing multiprocessing systems have been recognized by Campeau in [11]–[13].

## II. The Computational Method

The proposed evaluation method, named the $E$-method for brevity, consists of the following.

1) A problem-dependent correspondence rule $C_f$, which associates variables and constants of a given computational problem $f$ with a system $L$ of simultaneous linear equations $\boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}$ in such a way that a) there is a one–one correspondence between dependent variables (results) of the problem $f$ and the elements of the solution vector $\boldsymbol{y}$ of the system $L$, and b) the elements of $\boldsymbol{A}$ and $\boldsymbol{b}$ satisfy certain conditions, as specified later. A computational problem $f$ is said to be $L$-reducible if there exists such a correspondence rule $C_f$.

2) A parallel algorithm for solving the system $L$ in time linearly proportional to the desired number of correct digits of the solution $\boldsymbol{y}$, which is also amenable to an efficient hardware-level implementation.

*Example 1:* Let

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

be a real-valued rational function. Then $C_P{:}R_{2,3}(x) \rightarrow L$ specifies

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

so that $y_1 = R_{2,3}(x)$. □

*Example 2:* Let $f_i(x) = x^i$, $i = 2,3,4$. The $C_{\{f_i\}}{:}\{f_i(x)\} \rightarrow L$ specifies

$$L{:}\begin{bmatrix} 1 & -x & & \\ & 1 & -x & \\ & & 1 & -x \\ & & & 1 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ x \end{bmatrix}$$

so that $y_{5-i} = x^i$, $i = 2,3,4$. □

### A. The Computational Algorithm

The algorithm for solving a system $L$ is of an iterative, digit-by-digit nature. It generates one digit of each of the elements of the solution vector in one step, starting with the most significant digits. Since several functional properties of the algorithm depend on the redundant number representation systems, we begin with the following definitions.

*Definition 1:* An $m$ digit radix $r$ representation of a number $x$, $|x| < 1$, is a polynomial expansion

$$x = \text{sign } x \cdot \sum_{i=1}^{m} x_i r^{-i}$$

where $x_i \in D$, $\forall_i$, and $D$ is a digit set.

*Definition 2:* Given the radix $r$, a set of consecutive integers $D$, including zero is 1) nonredundant if its cardinality satisfies

$$|D| = r$$

2) redundant if

$$|D| > r.$$

*Definition 3:* A symmetric redundant digit set is defined as

$$D_\rho = \{-\rho,-(\rho-1),\cdots,-1,0,1,\cdots,\rho-1,\rho\}$$

where

$$\frac{r}{2} \le \rho \le r-1.$$

*Definition 4:* A symmetric redundant digit set $D_\rho$ is said to be 1) minimally redundant if

$$|D_\rho| = r+1$$

i.e., $\rho = r/2$ (assuming an even radix $r$); 2) maximally redundant if

$$|D_\rho| = 2r-1$$

i.e., $\rho = r-1$. Let $D$ and $D_\rho$ denote nonredundant and redundant digit sets, respectively. Then the representation of a number $x$ is nonredundant or redundant depending whether $x_i \in D$ or $x_i \in D_\rho$. The effects of the redundancy on the performance and the complexity of implementation of the algorithm will be discussed later.

For an $n$th-order system $L$, let $\boldsymbol{G} = \boldsymbol{I} - \boldsymbol{A} = (g_{ij})_{n\times n}$, where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{A}$ is the nonsingular matrix defined by the correspondence rule $C_f$. Let $||\boldsymbol{y}||$ and $||\boldsymbol{A}||$ denote the maximum vector norm

$$||\boldsymbol{y}||_\infty = \max_i |y_i|$$

and the consistent matrix norm

$$||\boldsymbol{A}||_\infty = \max_i \left(\sum_{j=1}^{n} |a_{ij}|\right)$$

respectively.

Define the basic recursion as

$$\boldsymbol{w}^{(j)} = r(\boldsymbol{z}^{(j-1)} + \boldsymbol{G}\boldsymbol{d}^{(j-1)}) \tag{1}$$

where

$r$ the radix;
$j$ the recursion index,
$j = 1,2,\cdots,m+1$, and $m$ is the number of radix $r$ digits in the solution;

$\boldsymbol{w}^{(j)} = [w_1{}^{(j)},\cdots,w_n{}^{(j)}]$;
$\boldsymbol{d}^{(j)} = [d_1{}^{(j)},\cdots,d_n{}^{(j)}]$ the $j$th digit vector;
$\boldsymbol{z}^{(j)} = [z_1{}^{(j)},\cdots,z_n{}^{(j)}]$ the $j$th residual vector.

The digits $d_i{}^{(j)}$ are defined by the digit selection function as follows:

$$d_i{}^{(j)} = s(w_i{}^{(j)}) = \begin{cases} \text{sign } w_i{}^{(j)} \lfloor |w_i{}^{(j)}| + \tfrac{1}{2} \rfloor, & \text{if } |w_i{}^{(j)}| \le \rho \\ \text{sign } w_i{}^{(j)} \lfloor |w_i{}^{(j)}| \rfloor, & \text{otherwise,} \end{cases} \tag{2}$$

where $\lfloor w \rfloor$ denotes the integer part of $w$. The vector selection function is defined as

$$\boldsymbol{d}^{(j)} = S(\boldsymbol{w}^{(j)}) = [s(w_1{}^{(j)}),\cdots,s(w_n{}^{(j)})]. \tag{3}$$

Clearly,

$$d_i{}^{(j)} \in D_\rho = \{-\rho,\cdots,-1,0,1,\cdots,\rho\}, \forall_i \forall_j. \tag{4}$$

The residual vector $\boldsymbol{z}^{(j)}$ is defined as

$$\boldsymbol{z}^{(j)} = \boldsymbol{w}^{(j)} - \boldsymbol{d}^{(j)}. \tag{5}$$

*Theorem 1*

Let

$$\tfrac{1}{2} \le \zeta < 1$$

and

$$0 < \alpha \le \frac{1}{r}\left[1 - \frac{\zeta}{\rho}(r-1)\right]. \tag{6}$$

If

$$||\boldsymbol{G}|| \le \alpha \tag{7}$$

$$||\boldsymbol{z}^{(0)}|| = ||\boldsymbol{b}|| \le \zeta \tag{8}$$

$$||\boldsymbol{d}^{(0)}|| = 0$$

$$\boldsymbol{d}^{(j)} = S(\boldsymbol{w}^{(j)})$$

then

a) $||\boldsymbol{z}^{(j)}|| \le \zeta, \quad \forall_j$ (9)

b) $||\boldsymbol{y} - \tilde{\boldsymbol{y}}|| < r^{-m}$ (10)

after $(m+1)$ recursive steps (1), where $\boldsymbol{y} = \boldsymbol{A}^{-1} \cdot \boldsymbol{b}$ is the solution of the system $L$, and

$$\tilde{\boldsymbol{y}} = \sum_{j=1}^{m+1} \boldsymbol{d}^{(j)} r^{-j}$$

$$= \left[\sum_{j=1}^{m+1} d_1{}^{(j)} r^{-j},\cdots,\sum_{j=1}^{m+1} d_n{}^{(j)} r^{-j}\right]$$

$$= [\tilde{y}_1,\cdots,\tilde{y}_n]$$

is the generated solution of $(m+1)$ radix $r$ digits of precision.

*Proof:* a) The consistency of the recursion (1) with respect to the selection function (2) is shown inductively. Assume that

$$||\boldsymbol{z}^{(j-1)}|| \leq \zeta.$$

Then

$$\begin{aligned} ||\boldsymbol{w}^{(j)}|| &= r||\boldsymbol{z}^{(j-1)} + \boldsymbol{G}\cdot\boldsymbol{d}^{(j-1)}|| \\ &\leq r||\boldsymbol{z}^{(j-1)}|| + r||\boldsymbol{G}||\cdot||\boldsymbol{d}^{(j-1)}|| \\ &\leq r\zeta + r\alpha\rho \\ &= \rho + \zeta. \end{aligned} \tag{11}$$

Since $||\boldsymbol{d}^{(j)}|| \leq \rho$, by definition of the selection function (2), it follows from (5) that

$$||\boldsymbol{z}^{(j)}|| \leq \zeta.$$

Since $||\boldsymbol{z}^{(0)}|| \leq \zeta$ and $||\boldsymbol{d}^{(0)}|| = 0$, the claim a) is proved.

b) The convergence is proved by showing that the solution error vector

$$\boldsymbol{h} = [h_1, h_2, \cdots, h_n] = \boldsymbol{y} - \tilde{\boldsymbol{y}} \tag{12}$$

satisfies

$$||\boldsymbol{h}|| < r^{-m}. \tag{13}$$

After $m+1$ steps, the following holds:

$$\boldsymbol{d}^{(m+1)} + \boldsymbol{z}^{(m+1)} = r^{m+1}\cdot\boldsymbol{b} + \boldsymbol{G}\cdot\left[\sum_{j=1}^{m}\boldsymbol{d}^{(j)}r^{m+1-j}\right] - \boldsymbol{I}\cdot\left[\sum_{j=1}^{m}\boldsymbol{d}^{(j)}r^{m+1-j}\right]$$

or

$$r^{-m-1}\boldsymbol{z}^{(m+1)} = \boldsymbol{b} - \boldsymbol{A}\cdot\tilde{\boldsymbol{y}} - \boldsymbol{G}\cdot\boldsymbol{d}^{(m+1)}r^{-m-1}. \tag{14}$$

Let

$$\begin{aligned} \boldsymbol{e}^{(m+1)} &= [e_1{}^{(m+1)}, e_2{}^{(m+1)}, \cdots, e_n{}^{(m+1)}] \\ &= (\boldsymbol{z}^{(m+1)} + \boldsymbol{G}\cdot\boldsymbol{d}^{(m+1)})r^{-m-1}. \end{aligned}$$

Since $\boldsymbol{y} = \boldsymbol{A}^{-1}\cdot\boldsymbol{b}$

$$\boldsymbol{h} = \boldsymbol{A}^{-1}\cdot(-\boldsymbol{e}^{(m+1)}). \tag{15}$$

After $m+1$ steps

$$\begin{aligned} ||\boldsymbol{h}|| &= ||\boldsymbol{A}^{-1}\cdot(-\boldsymbol{e}^{(m+1)})|| \\ &\leq ||\boldsymbol{A}^{-1}||\cdot||\boldsymbol{e}^{(m+1)}||. \end{aligned} \tag{16}$$

Since $||\boldsymbol{G}|| \leq \alpha < 1$ for $r \geq 2$, the matrix $\boldsymbol{G}$ is convergent, i.e.,

$$\lim_{p\to\infty}\boldsymbol{G}^p = 0.$$

Therefore [18, p. 113],

$$\begin{aligned} ||\boldsymbol{A}^{-1}|| &= ||\boldsymbol{I} + \boldsymbol{G} + \boldsymbol{G}^2 + \cdots|| \\ &\leq ||\boldsymbol{I}|| + ||\boldsymbol{G}|| + ||\boldsymbol{G}^2|| + \cdots \\ &\leq 1 + \sum_{p=1}^{\infty}\alpha^p \\ &= \frac{1}{1-\alpha} \leq \frac{4}{3}. \end{aligned} \tag{17}$$

Since $||\boldsymbol{e}^{(m+1)}|| \leq r^{-m-1}(\zeta + \alpha\rho)$

$$||\boldsymbol{h}|| \leq \frac{1}{1-\alpha}\cdot\frac{\zeta + \alpha z\rho}{r}\cdot r^{-m} = \gamma\cdot r^{-m} \tag{18}$$

where $\gamma = 1/2(r-1)$ for minimally redundant digit set $D_\rho$ and $\gamma = 1/r$ for maximally redundant digit set $D_\rho$. Since $\gamma < 1$ for $r > 1$

$$||\boldsymbol{h}|| < r^{-m}. \qquad \square$$

Theorem 1 completes the general specification of the computational part of the $E$-method, which is summarized below for reference convenience:

*Part 1 (Correspondence):* Given an $L$-reducible problem $f$, apply the correspondence rule $C_f$ to the variables and constants of $f$ in order to obtain the coefficient matrix $\boldsymbol{A}$ and the vector $\boldsymbol{b}$ of the system of linear equations $L$. $L$-reducibility implies that the system $L$ is nonsingular and that the elements of $\boldsymbol{A}$ and $\boldsymbol{b}$ can be made conformable to the conditions

$$\sum_{\substack{j=1\\ j\neq i}}^{n}|a_{ij}| \leq \alpha, \qquad \forall i$$

$$|b_i| \leq \zeta, \qquad \forall i.$$

*Part 2 (Computation)*

*Algorithm E:*

/Initialization/

$$\boldsymbol{w}^{(0)} \leftarrow \boldsymbol{b};\ \boldsymbol{d}^{(0)} \leftarrow 0 \tag{19}$$

/Recursion/

for $j = 1,2,\cdots,m+1$ do:

$\boldsymbol{w}^{(j)} \leftarrow r\cdot[\boldsymbol{w}^{(j-1)} - \boldsymbol{A}\cdot\boldsymbol{d}^{(j-1)}]$

$\boldsymbol{d}^{(j)} \leftarrow S(\boldsymbol{w}^{(j)})$.

/Termination/

HALT

The result(s) of $f$, for the given precision $m$, are represented by

$$\begin{aligned} \tilde{\boldsymbol{y}} &= \sum_{j=1}^{m+1}\boldsymbol{d}^{(j)}r^{-j} \\ &= [\tilde{y}_1, \tilde{y}_2, \cdots, \tilde{y}_n]. \end{aligned}$$

Several properties of the $E$-method, which affect the speed, the complexity of implementation, and the domain of application, are discussed next. The additional considerations are given in [1] and in Sections III and IV.

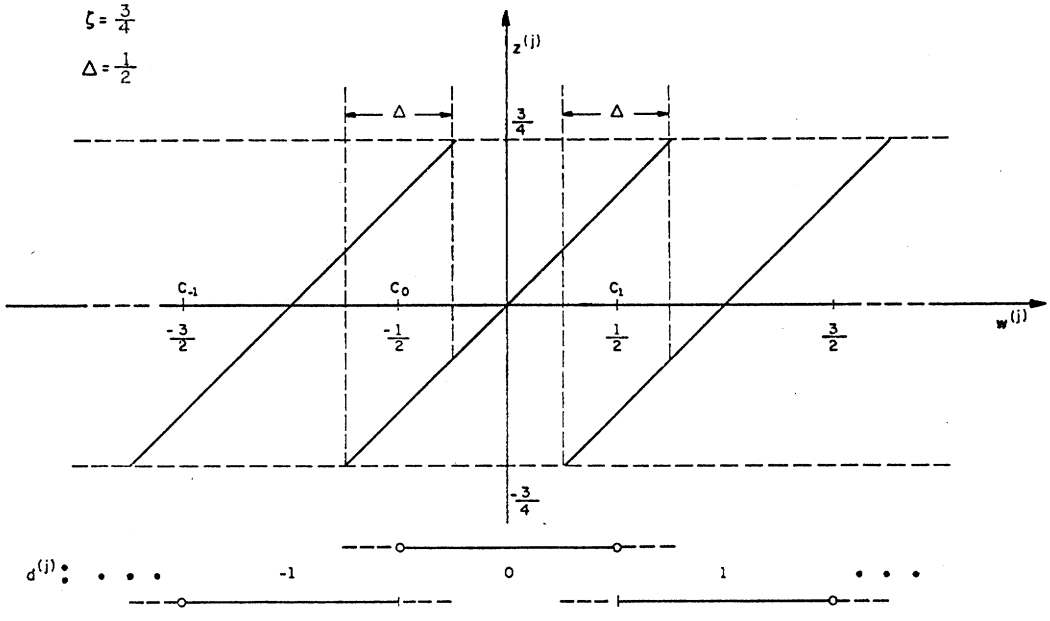


Fig. 1. Limited precision selection procedure.

### B. Primitive Operators

The primitive operators required for implementation of the basic recursion (1), are the multiplication by one signed digit, the multioperand addition, and the selection function (2). The number of operands in the multioperand addition reduces to two (three) in the evaluation of polynomials (rational functions). The selection function $s(w_i^{(j)})$ corresponds to rounding of $w_i^{(j)}$ to the integer part. A more detailed analysis of the selection problem [1] shows that the selection can be performed using a truncated value of $w_i^{(j)}$, denoted $\hat{w}_i^{(j)}$, instead of its full precision value, if the residual bound $\zeta$ satisfies $\zeta > 1/2$. This selection procedure is illustrated in Fig. 1. For $\zeta = 1/2$, each subinterval of $w_i^{(j)}$ corresponds to a unique value of $d_i^{(j)}$ that must be selected in order to satisfy the consistency requirement of the recursion. Consequently, the value of $w_i^{(j)}$ must be computed in the nonredundant form to the full working precision. However, if $\zeta > 1/2$, then an overlap $\Delta$ exists between the subintervals of $w_i^{(j)}$ such that the choice of, say, $d_i^{(j)} = k$ and $d_i^{(j)} = k + 1$ is equally valid. The freedom of choice in the selection procedure has two important implications [20], [21]. First, the choice of comparison constants that are representable as "simple" numbers (e.g., 1/2, 3/4, etc.) is possible. Second, the required precision of the argument used in the selection is of the same order of magnitude as the precision of the overlap $\Delta$. The comparison constants implied by the selection function (2), are "simple" numbers ($\pm 1/2$, $\pm 3/2$, etc.) by definition; $\zeta = 1/2$ could be used as well. However, given an overlap $\Delta$, it can be easily seen from Fig. 1 that $s(\hat{w}_i^{(j)})$ generates a correct digit $d_i^{(j)}$ if

$$|w_i^{(j)} - \hat{w}_i^{(j)}| \leq \frac{\Delta}{2}. \tag{20}$$

Therefore, $\hat{w}_i^{(j)}$ can be computed by a limited carry propagation addition followed by the carry assimilation over the $\sim|\log_r |\Delta||$ most significant digit positions. In other words, $w_i^{(j)}$ can always be computed in a redundant form and then partially converted to the nonredundant form in order to get $\hat{w}_i^{(j)}$. Thus the time required to perform the basic recursive step can be made independent of the number of digits in the operands. The relationships between bounds $\alpha$ and $\zeta$, the overlap $\Delta$, and the amount of redundancy are shown in Table I.

The definition of the algorithm and the previous discussion indicate the necessity and usefulness of the redundancy in number representation systems. The elements $\tilde{y}_i$ of the computed solution vector $\tilde{\boldsymbol{y}}$ must be in redundant form. Conversion to the nonredundant representation can be accomplished in one addition time at the end of the algorithm. The quantities $w_i^{(j)}$ should be represented in a redundant form for the reasons previously discussed, while all other quantities (the elements $a_{ij}$ and $b_i$) need not be in a redundant form.

The overall simplicity of the algorithm and its amenability to an efficient hardware-level implementation comes from the following functional properties. The algorithm generates the most significant digits of the result $\tilde{\boldsymbol{y}}$ first in such a way that once digit $y_i^{(j)} = d_i^{(j)}$ is generated at the step $j$, a) it will not be affected at any subsequent step $k$, $k > j$, and b) it is used directly only at the step $j + 1$. Consequently, the matrix–vector product $\boldsymbol{A} \cdot \boldsymbol{d}^{(j)}$ is simple to generate, since $\boldsymbol{d}^{(j)}$ is a vector consisting of one digit components. Clearly, the convergence rate of the algorithm is limited by definition to one digit per step.

### C. Error Behavior

Since the algorithm utilizes only fixed-point addition operations without overflow, no roundoff errors are generated. The errors due to the finite precision representa-

**TABLE I**
**Relationships Between Bounds**

| Mode of Operation: | Redundancy in $\mathfrak{D}_\rho$: | |
|---|---|---|
| | Minimal $\rho = \frac{r}{2}$ | Maximal $\rho = r - 1$ |
| Operand Precision Dependent $\zeta = \frac{1}{2}$ $\Delta = 0$ | $\alpha \leq \frac{1}{r^2}$ | $\alpha \leq \frac{1}{2r}$ |
| Operand Precision Independent $\zeta = \frac{1}{2}(1+\Delta)$ $0 < \Delta < \frac{2\rho}{r-1} - 1$ | $\alpha \leq \frac{1}{r^2}[1-\Delta(r-1)]$ | $\alpha \leq \frac{1}{2r}[1-\Delta]$ |

tion of the elements of $\boldsymbol{A}$ and $\boldsymbol{b}$ are analyzed in [1]. If $m$ is the desired number of radix $r$ digits in the answer, then the effects of these representation errors are compensated by extending the working precision to

$$m' = m + 1 + \left\lceil \log_r \frac{2n'}{\Delta} \right\rceil \tag{21}$$

radix $r$ digits, the number of nonzero elements in $\boldsymbol{A}$ is $n'$, and $\Delta$ is the overlap between $w_i^{(j)}$ selection subintervals, as previously discussed. Typically, $1/16 \leq \Delta \leq 1/2$. For example, in the case of rational function evaluation $n' = 3$, so that the extended working precision is $m' = m + 1 + \lceil \log_2 12 \rceil = m + 5$, assuming $r = 2$ and $\Delta = 1/2$.

### D. Scaling

In general, an adjustment of the size of the elements $a_{ij}$ and $b_i$, i.e., the scaling, is necessary as implied by the conditions (7) and (8) of Theorem 1. The simplest problem in scaling which may arise is that

$$||\boldsymbol{A}|| \leq 1 + \alpha$$

but

$$||\boldsymbol{b}|| \not\leq \zeta.$$

However, instead of solving

$$\boldsymbol{Ay} = \boldsymbol{b}$$

one can solve an equivalent system, scaled as follows:

$$\boldsymbol{ASy} = \boldsymbol{Sb}$$

or

$$\boldsymbol{Ay'} = \boldsymbol{b'} \tag{22}$$

where

$$||\boldsymbol{b'}|| = ||\boldsymbol{Sb}|| \leq \zeta.$$

The scaling matrix $\boldsymbol{S} = (s_{ij})_{n\times n}$ is defined as

$$s_{ij} = \begin{cases} r^{-\sigma}, & \text{for } i = j \\ 0, & \text{for } i \neq j \end{cases} \tag{23}$$

where $\sigma$ is a positive integer such that

$$r^{-\sigma}||\boldsymbol{b}|| \leq \zeta$$

or

$$\sigma = \left\lceil \log_r \frac{||\boldsymbol{b}||}{\zeta} \right\rceil. \tag{24}$$

Clearly, in order to retain the same number of significant digits in $\boldsymbol{y'}$ as in $\boldsymbol{y}$, one must carry $\sigma$ extra steps. Then

$$\boldsymbol{y} = \boldsymbol{S}^{-1}\boldsymbol{y'}.$$

Multiplications with $\boldsymbol{S}$ and $\boldsymbol{S}^{-1}$ involve only a shift of $\sigma$ positions right and left, respectively.

The problem of scaling the matrix $\boldsymbol{A}$ may be considered equivalent to a problem of transforming a given matrix $\boldsymbol{A}$, with arbitrarily large elements $a_{ij}$, into a diagonally dominant matrix $\hat{\boldsymbol{A}}$, such that

$$|\hat{a}_{ii}| \geq k|\hat{a}_{ij}|, \qquad \forall_j \forall_i \tag{25}$$

and $k$ is chosen in such a way so that the condition (7) holds after normalization of the diagonal elements to 1. This, in general, requires an amount of work incommensurable with the expected performance of the $E$-method since $\boldsymbol{A}$ depends on the independent variables of the original problem. If the matrix $\boldsymbol{A}$ is triangular, then an efficient scaling can be devised using only shift operations. Thus an

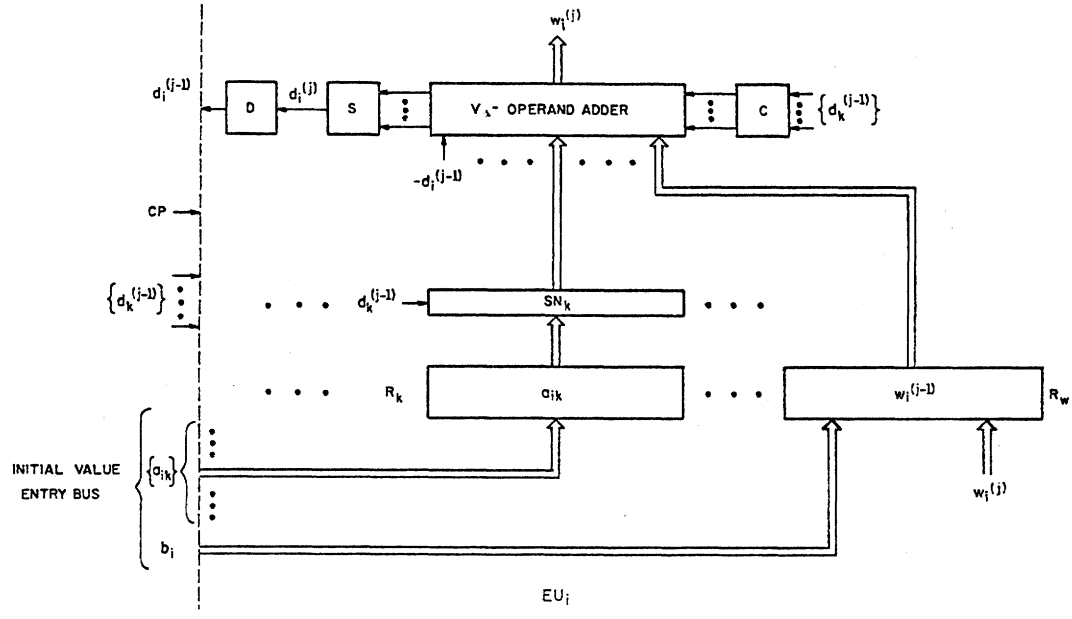


Fig. 2. Elementary unit: Global structure.

effective scaling procedure exists for polynomials but not for rational functions. This is implied by the corresponding rules $C_P$ and $C_R$ discussed later. With respect to scaling, two classes of applications are distinguished. In one class there are problems, such as the evaluation of mathematical functions where all parameters are known *a priori*. Then, once the best polynomial or rational $L$-reducible approximation is determined, the scaling is performed once for a given argument range. The other class contains the problems with parameters known only at the execution time so that the scaling has to be done each time the $E$-method is applied. In this case, the scaling introduces an overhead, making the extension of the $E$-method to a floating-point number system desirable.

## III. On Implementation and Performance of the $E$-Method

The basic computational block, the elementary unit $EU_i$, is a hardware structure implementing the basic recursion formula (1) or, more precisely, the corresponding scalar recursion

$$w_i^{(j)} = r(w_i^{(j-1)} - \sum_{k=1}^{n} a_{ik} d_k^{(j-1)}). \tag{26}$$

As indicated by Fig. 2, where the global structure of the elementary unit $EU_i$ is shown, the evaluation of $w_i^{(j)}$ basically requires a $v$-operand adder. In many practical applications $v$ is rather small. In order for the time of addition to be independent of operand precision, the $w_i^{(j)}$ are represented in a redundant form.

The selection procedure, defined by the selection function $s(w_i^{(j)})$ is performed by the block $S$ which forms $w_i^{(j)}$ by converting a few (3–5) of the most significant digits of $w_i^{(j)}$ into nonredundant form. After rounding $\hat{w}_i^{(j)}$, the integer part represents the selected digit $d_i^{(j)}$. The required precision of $\hat{w}_i^{(j)}$ is basically determined by the overlap $\Delta$ and the number $v$ of summands. The previous value $d_i^{(j-1)}$ is saved in register $D$. The single, signed digit radix-$r$ multipliers $d_k^{(j)}$ are incorporated through the selection networks $\{SN_k\}$, each capable of forming required multiples of $a_{ik}$. The carry generator $C$ would be needed if, for example, a radix-complement representation of negative numbers is adopted. In that case, the selection networks merely form direct or complement of a possibly shifted value of $a_{ik}$. The complexity of the selection networks increases for higher radices, and since the additional multiples appear as summands, complexity of the adder will also be increased. Therefore, a higher radix, while reducing the necessary number of steps for a given precision, does increase both the time to perform the basic recursion and the complexity of the corresponding elementary unit.

The central part of an $EU$, the multioperand adder, can be implemented in various ways in order to achieve the desired speed/cost factor. The registers $R_k$ store the corresponding coefficients $a_{ik}$ throughout a particular evaluation; their number for the elementary unit $EU_i$ is determined by the number of nonzero elements in the $i$th row of the matrix $\boldsymbol{G}(x)$; i.e., the number of inputs to $EU_i$. Register $R_w$ and the corresponding data path must accommodate the redundantly represented $w_i^{(j)}$. The initialization, i.e., the execution of the particular correspondence rule $C_f$, is performed by loading the coefficients $a_{ik}$ and $b_i$ via Initial Value Entry bus.

The control requirements of an $EU$ are very simple: assuming a synchronous mode of operation of the entire configuration for the elementary units, only the synchronizing pulses on which the transfer of $w_i^{(j)}$ into $R_w$ occurs should be provided. By controlling their number one can achieve a variable precision mode of operation.

The time required to perform one recursive step on an

elementary unit is defined as

$$t_0 = t_A + t_S + t_T \tag{27}$$

where $t_A$ is the time to generate $w_i^{(j)}$ in a redundant form, $t_S$ is the selection time, and $t_T$ is the register transfer time. Both $t_S$ and $t_T$ correspond to a few gate delays, (3–4) $t_g$, so that $t_A$ appears as the dominant factor which depends on the number $v$ of summands and the adder structure. For practical reasons, $v$ may be defined to denote the number of radix-2 summands; i.e., the higher radix or redundantly represented operands are replaced by their binary equivalents. Then a simple adder structure consisting of $v - 2$ levels of full-adder rows, will have $t_A = 2(v - 2)t_g$, assuming $2t_g$ per full-adder level. More sophisticated adder structures [14], i.e., Dadda-type, can considerably reduce this time. When $v$ is small, as in polynomial evaluation, then $t_0 = 0(10t_g)$, assuming radix 2.

### A. Graph Representation of Computing Configurations

An $L$-reducible problem $f$ of order $n$ is solved fastest by the $E$-method on a structure consisting of $n$ interconnected elementary units operating in parallel. The computational algorithm $E$ indicates that the intercommunication requirements are simple due to the fact that the elementary units are functionally related to each other only via the digit vector $\boldsymbol{d}$. This implies that the physical connection between the elementary units $EU_i$ and $EU_j$ only needs accommodate a transfer of one signed radix-$r$ digit. In multiprocessor structures used for fast parallel computations, the processor intercommunications usually require a full precision width.

A computing structure for solving a given problem $f$ by the $E$-method is conveniently specified by a computation graph $G_f(V, \boldsymbol{K})$ where $V = \{V_i | i = 1, \cdots, n\}$ is a set of vertices and $K$ is a connecting matrix defining a set of directed arcs. Each vertex $V_i$ corresponds to an elementary unit $EU_i$ symbolically represented as in Fig. 3 where the outgoing arc $d_i$ carries the digit generated by $EU_i$ and one or more incoming arcs $d_j$ (inputs to $EU_i$) carry the digits generated by $\{EU_j\}$. The connection matrix $\boldsymbol{K} = (k_{ij})_{n\times n}$ is in one–one correspondence with the matrix $\boldsymbol{G} = \boldsymbol{I} - \boldsymbol{A}$ of the system $L$ as follows:

$$k_{ij} = \begin{cases} 1, & \text{if } g_{ij} \neq 0 \\ 0, & \text{if } g_{ij} = 0 \end{cases} \tag{28}$$

and $k_{ij} = 1$ specifies that the arc $d_j$ is an incoming arc for the vertex $V_i$. The use of $d_i$ for internal functions in $EU_i$, as indicated in Fig. 2, need not be specified on the graph unless $g_{ii} \neq 0$.

The computation graph $G_f$ may be acyclic or cyclic. The graph $G_f$ is acyclic if and only if its connection matrix $\boldsymbol{K}$ is strictly upper (lower) triangular. In a practical sense, a cyclic graph $G_f$ corresponds to a problem $f$ which involves division.

A graph $G_f$ provides a direct estimate of the required execution time and implementation complexity. If $N(\boldsymbol{K})$ denotes the number of ones in the connection matrix $\boldsymbol{K}_{n\times n}$, then the computational structure requires at most $N(\boldsymbol{K}) + 2n$ $m'$-digit registers, $n$ adders with total of $N(\boldsymbol{K}) + 2n$ operands, and $N(\boldsymbol{K})$ single-digit interconnections. It is assumed that two registers are sufficient to store $w_i$ value in a redundant form. If $N_i = |\{d_k\}|$ denotes the number of inputs to the $i$th elementary unit, then $EU_i$ requires $v = (N_i + 2)$ operand adder and that many registers.

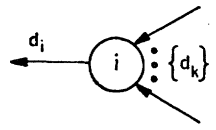


Fig. 3. Elementary unit: Graph representation.

The time $t_0$ required to perform one recursive step is clearly determined by the parameters of the elementary unit $EU_k$ such that $N_k = \max_i(N_i)$. The total time, then, for the $E$-method evaluation, assuming an $m$-digit precision, is

$$T_E(f) = (m + 1)t_0 \tag{29}$$

if no scaling is required, and

$$T_E(f) = (m + 1 + \sigma)t_0$$

otherwise. The integer $\sigma > 0$ is defined by the particular scaling requirements.

We conclude this section with the following remarks. The functional properties of the $E$-method, namely the step-invariant nature of its computatonal algorithm and linearity of its primitive operator, makes an adaptation both to a variable precision mode of operation and a variable number of elementary units rather straightforward.

## IV. On Applications of the $E$-Method

Several applications of the $E$-method, beginning with the evaluation of polynomials, are described. Additional examples can be found in [1]. In all applications, a fixed-point number system is assumed.

### A. Evaluation of Polynomials

Let

$$P_\mu(x) = \sum_{i=0}^{\mu} p_i x^i \tag{30}$$

be a $\mu$-degree polynomial with real-valued coefficients $\{p_i\}$ and the argument $x \in [a,b]$.

Assume that $\boldsymbol{p}$ and $\boldsymbol{x}$ are vectors with $\{p_i\}$ and all representable values of $x \in [a,b]$ as the components, respectively.

If

$$||\boldsymbol{p}|| \leq \zeta$$

and

$$||\boldsymbol{x}|| \leq \alpha \tag{31}$$

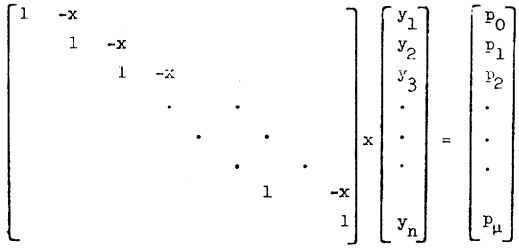

Fig. 4. Correspondence rule $C_P$.

then the problem of evaluating the polynomial $P_\mu(x)$ is immediately reducible to the system $L:\boldsymbol{A}(x)\boldsymbol{y} = \boldsymbol{b}$ of order $n = \mu + 1$ according to the following correspondence rule $C_P$:

$$a_{ij} = \begin{cases} 1, & \text{for } i = j \\ -x, & \text{for } j = i+1,\ i \le \mu \\ 0, & \text{otherwise} \end{cases}$$

$$b_i = \begin{cases} p_{i-1}, & \text{for } i = 1,2,\cdots,\mu+1 \\ 0, & \text{otherwise} \end{cases} \tag{32}$$

as illustrated in Fig. 4. The system $L$ is then solved using the Algorithm $E$, so that after $m + 1$ steps

$$\tilde{y} = \sum_{j=1}^{m+1} \boldsymbol{d}^{(j)} r^{-j}$$

satisfies

$$|P_\mu(x) - \tilde{y}_1| < r^{-m} \tag{33}$$

and

$$\left| \sum_{k=i-1}^{\mu} p_k x^{k-i+1} - \tilde{y}_i \right| < r^{-m}$$

for $i = 2,3,\cdots,\mu+1$.

The computational graph $G_P$ is shown in Fig. 5. All the elementary units $EU_i$, $i = 1,2,\cdots,n$ are identical: each one requires an adder with one redundant $(w_i^{(j)})$ and one nonredundant $(x \cdot d_{i+1}^{(j)})$ operand.

When the conditions on the norms (8) are not satisfied, the following scaling procedure can be used. Let $\boldsymbol{S} = (s_{ij})_{n\times n}$ be a diagonal scaling matrix with $s_{ii} = r^{-(i-1)\sigma_A}$, and $\sigma_A$ a positive integer or zero. Then its inverse $\boldsymbol{S}^{-1}$ is also a diagonal matrix with $s'_{ii} = r^{(i-1)\sigma_A}$. Consider

$$\boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}\boldsymbol{S}^{-1}\boldsymbol{y} = \boldsymbol{S}^{-1}\boldsymbol{b}. \tag{34}$$

Let $\boldsymbol{A}^* = \boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}$, $\boldsymbol{y}^* = \boldsymbol{S}^{-1}\boldsymbol{y}$, and $\boldsymbol{b}^* = \boldsymbol{S}^{-1}\boldsymbol{b}$. If

$$\sigma_A = \begin{cases} \left\lceil \log_r \dfrac{||\boldsymbol{x}||}{\alpha} \right\rceil, & \text{if } ||\boldsymbol{x}|| > \alpha \\ 0, & \text{otherwise} \end{cases}$$

then $||\boldsymbol{A}^*|| \le \alpha$. Since $||\boldsymbol{b}^*|| \le r^{\mu\sigma_A}||\boldsymbol{b}||$, additional scaling, as described in Section II-D, must be performed with

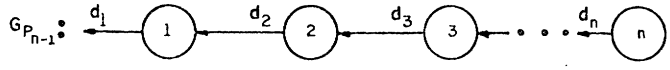

Fig. 5. Computational graph $G_P$.

$$\sigma_b = \begin{cases} \left\lceil \log_r \dfrac{||\boldsymbol{b}^*||}{\zeta} \right\rceil, & \text{if } ||\boldsymbol{b}^*|| > \zeta \\ 0, & \text{otherwise.} \end{cases} \tag{35}$$

After $\sigma_A$ and $\sigma_b$ are determined, the scaling is performed by shifting

$$\begin{aligned} a_{i,i+1} &= -x \cdot r^{-\sigma_A}, && \text{for } i = 1,2,\cdots,\mu \\ b_i &= p_{i-1} \cdot r^{-(\sigma_b-(i-1)\sigma_A)} && \text{for } i = 1,2,\cdots,\mu+1. \end{aligned} \tag{36}$$

It can be shown that

$$T_E(P_\mu) = (m + 1 + \sigma_b)t_0 \tag{37}$$

when scaling is required. As an example, we determine the scaling requirements for the polynomial approximation $P_g(x) \approx \log_2(x)$, $x \in [\frac{1}{2},1]$ with a precision of 8 decimal digits [2, pp. 110 and 225]. Assuming an implementation with $r = 2$, $m = 27$, $\zeta = \frac{3}{4}$, and $\alpha = \frac{1}{8}$, we obtain $\sigma_A = 3$ and $\sigma_b = 29$. Therefore, the required number of steps to evaluate a given polynomial is $m + 1 + 29 = 57$.

We now consider the performance of the $E$-method in polynomial evaluation relative to a conventional or $S$-method and a parallel or $P$-method. Both these methods are assumed to be using an iterative multiplication algorithm so that the basic processing unit in all three methods can be considered equivalent in complexity and speed. Furthermore, we assume a fixed-point representation domain with no scaling requirements. Denoting addition time as $t_0$, we have the evaluation times and the number of processors for $E$-, $S$-, and $P$-method as follows:

$$\begin{aligned} T_E(P_\mu) &= (m+1)t_0, & n_E &= \mu + 1 \\ T_S(P_\mu) &= \mu m t_0, & n_S &= 1 \\ T_P(P_\mu) &= (\log_2 \mu + (0(\log_2 \mu)^{1/2})m)t_0, & n_P &= 2\mu. \end{aligned} \tag{38}$$

We have assumed that the $P$-method uses the Maruyama–Munro–Paterson algorithm for parallel polynomial evaluation [16].

Following Kuck [15], the $E$-method algorithm for polynomial evaluation has the speedup factor $S_E$

$$S_E = \frac{T_S}{T_E} = \mu\left(\frac{m}{m+1}\right) \approx \mu \tag{39}$$

the efficiency factor $E_E$

$$E_E = \frac{S_E}{n_E} = \left(\frac{\mu}{\mu+1}\right)\left(\frac{m}{m+1}\right) > 0.5, \quad \text{for } \mu > 1,\ m > 3 \tag{40}$$

and the cost factor $C_E$

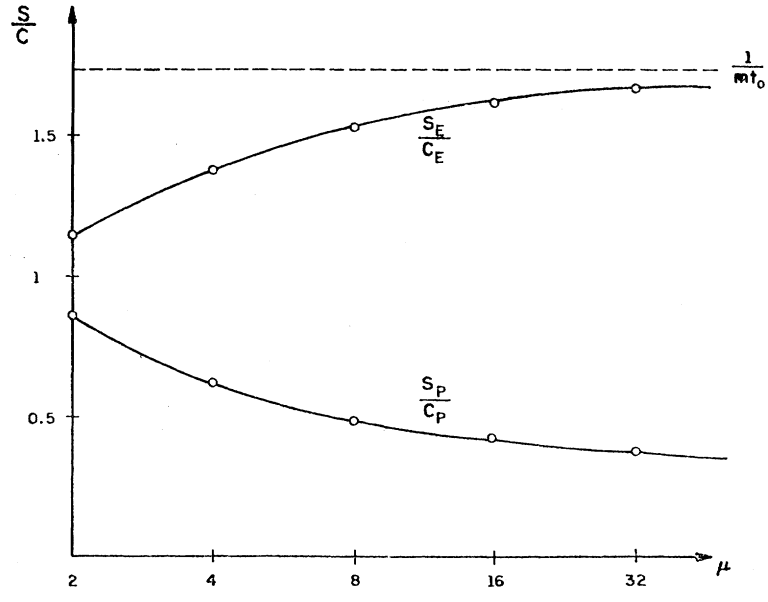


Fig. 6. Evaluation of polynomials: Performance measures.

$$C_E = n_E \cdot T_E = (\mu + 1)(m + 1)t_0. \tag{41}$$

Similarly, for the $P$-method algorithm

$$S_P = \frac{T_S}{T_P} \approx \frac{M}{\sqrt{M} + m} \cdot \frac{\mu}{\sqrt{M}} \approx \frac{\mu}{\sqrt{M}}$$

$$E_P = \frac{S_P}{n_P} \approx \frac{1}{2\sqrt{M}} \cdot \frac{m}{\sqrt{M} + m}$$

$$C_P \approx 2\mu(M + \sqrt{M}m)t_0 \tag{42}$$

where $M = \log_2 \mu$. As a reasonable measure of performance, Kuck suggests the ratio of effectiveness, given by speedup, and cost of the method so that the $A$-method is better in performance that the $B$-method if

$$\max\left(\frac{S_A}{C_A}, \frac{S_B}{C_B}\right) = \frac{S_A}{C_A}. \tag{43}$$

It can be seen that

$$\lim_{\mu \to \infty} \frac{S_E}{C_E} = \frac{m}{(m + 1)^2 t_0} \approx \frac{1}{mt_0}$$

while

$$\lim_{\mu \to \infty} \frac{S_P}{C_P} = 0. \tag{44}$$

With respect to the precision, both $\lim_{m \to \infty}(S_E/C_E) = 0$ and $\lim_{m \to \infty}(S_P/C_P) = 0$, but

$$\lim_{m \to \infty} \frac{S_E}{C_E} \cdot \frac{C_P}{S_P} = \frac{2\mu \log_2 \mu}{\mu + 1} > 1, \qquad \text{for } \mu > 1. \tag{45}$$

This indicates that the $E$-method algorithm is better in performance than any $P$-method algorithm for the evaluation of polynomials under the stated assumptions. Furthermore, the $E$-method algorithm has a behavior of performance measure qualitatively different from the considered $P$-method algorithm, as illustrated in Fig. 6. In this example, it is assumed that $m = 56$, $r = 2$, and, for presentation convenience, that $t_0 = 10^{-2}$ units. Of course, the speed of a $P$-method could be increased beyond the speed of the $E$-method by using faster multiplication algorithms, but at the expense of increased cost.

### B. Evaluation of Rational Functions

Let $R_{\mu,\nu}(x)$ be a real-valued rational function

$$R_{\mu,\nu}(x) = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{\sum_{i=0}^{\mu} p_i x^i}{\sum_{i=0}^{\nu} q_i x^i}. \tag{46}$$

Without loss of generality, it is assumed that $q_0 = 1$. Let

$$L: \boldsymbol{Ay} = \boldsymbol{b}$$

be a nonhomogeneous system of $n$ simultaneous linear equations.

*Theorem 2*

If $\max(\mu,\nu) \leq n - 1$ and the coefficients $a_{ij}$, $b_i$ of the system $L$ are put into correspondence with the coefficients $p_i$, $q_i$ and the argument $x$ according to the following rule $C_R$:

$$a_{ij} = \begin{cases} 1, & \text{for } i = j \\ q_{i-1}, & \text{for } j = 1 \text{ and } i = 2,3,\cdots,\nu + 1 \\ -x, & \text{for } j = i + 1 \text{ and } i = 1,2,\cdots,n - 1 \\ 0, & \text{otherwise} \end{cases}$$

$$b_i = \begin{cases} p_{i-1}, & \text{for } i = 1,2,\cdots,\mu + 1 \\ 0, & \text{otherwise,} \end{cases} \tag{47}$$

then

$$y_1 = \frac{P_\mu(x)}{Q_\nu(x)} = R_{\mu,\nu}(x). \qquad \square$$

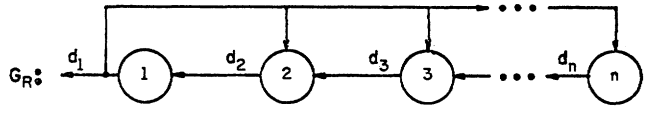


Fig. 7. Computational graph $G_R$.

*Proof:* By the Laplace expansion of the determinants. The correspondence rule $C_R$ defines a linear system $L$: $\boldsymbol{Ay} = \boldsymbol{b}$ for a given $L$-reducible rational function $R_{\mu,\nu}(x)$. Algorithm $E$ can be carried out on a configuration represented by the graph $G_R$ with $n = \max(\mu,\nu) + 1$, as illustrated in Fig. 7. The adder structure of the elementary unit needs to accommodate at most two nonredundant and one redundant operand and that, for radix 2, can be achieved with a two-level conventional adder. Note that no explicit division is required in evaluating rational functions.

No general scaling technique has been found for rational functions. The conditions under which a rational function is $L$-reducible are considered now in some detail. Let

$$||\boldsymbol{p}|| = \max_i |p_i|$$

$$||\boldsymbol{q}|| = \max_{i \neq 0} |q_i|$$

$$||\boldsymbol{x}|| = \max_{x \in [a,b]} |x|. \tag{48}$$

Then $R_{\mu,\nu}(x)$ is $L$-reducible if

$$\text{(i)} \quad ||\boldsymbol{q} + \boldsymbol{x}|| \leq \alpha$$

$$\text{(ii)} \quad ||\boldsymbol{p}|| \leq \zeta. \tag{49}$$

The condition is reduced to $||\boldsymbol{x}|| \leq \alpha$ for the first row of $\boldsymbol{A}$ and to $|q_\nu| \leq \alpha$ for the $(\nu + 1)$st row if $\mu \leq \nu$. The scaling implications of (ii) have been considered in Section II-D. For convenience, replace (i) with

$$\text{(i')} \quad ||\boldsymbol{q}|| \leq \alpha(1 - c), \qquad 0 < c < 1$$

$$\text{(i'')} \quad ||\boldsymbol{x}|| \leq c\alpha. \tag{50}$$

Assuming that (i'') is satisfied, a rational function $R_{\mu,\nu}(x)$ with $q_0 = 1$ is $L$-reducible if

$$|q_i| \leq \frac{1 - c}{c}\alpha, \qquad i = 1,2,\cdots,\nu. \tag{51}$$

Derivation of rational approximations under the conditions (i') and (i'') is currently being investigated. An example of a rational-function evaluation is given in the Appendix. The relative performance analysis, analogous to one previously described for polynomials, shows that $L$-reducible rational functions are evaluated faster and more efficiently using the $E$-method compared to a $P$-method, under the stated assumptions.

### C. Evaluation of Elementary Functions

With a capability to evaluate polynomials and certain rational functions, the $E$-method can be used for the evaluation of arbitrary functions. The evaluation of a given function is characterized by a set of coefficients, which can be kept in a local storage of the computing configuration, and a number of interconnected elementary units. In general, given a sufficient number of the two-input elementary units, an evaluation of a polynomial approximation would take $T_E(f) = 0(10m)t_g$, where $t_g$ is a gate delay and the radix is binary. Since the relationship between the speed and the cost of the $E$-method is linear, a wide range of evaluation requirements can be accommodated [1]. Furthermore, the evaluation could be performed in a variable precision. For example, the approximation $R_{3,2}(x)$ to $\ln \Gamma(1/x)$, for $x \in [10^{-3}, 12^{-1}]$ and 42 bits of accuracy, is evaluated in 43 carry-free additions, using 4 elementary units. The approximation $P_8(x)$ to $2^x$, for $x \in [0,1]$ and 40 bits of accuracy, can be evaluated in 47 steps using 9 elementary units, or in 94 steps using 5 elementary units.

No attempt has been made to derive polynomial and rational approximations for the elementary functions under the conditions (7) and (8).

### D. On Performing the Basic Arithmetics

The basic recursion of the Algorithm $E$ can clearly be used for additions, subtractions, or multiplications. The result obtained will be in a redundant form. By considering division as an $L$-reducible problem [1], Algorithm $E$ can be used to generate the quotient and the remainder in a deterministic fashion [17], with basic recursive step executable in time independent of the length of the operands. The computational graphs for division and multiplication are shown in Fig. 8.

### E. Evaluation of Certain Arithmetic Expressions

A set of elementary units can be applied in evaluating certain expressions, such as the multiple products or sums, or the inner products. For example, the multiple product $P_c = \Pi_{i=1}^{p} c_i$ can be evaluated as a degenerate case of polynomial. Since

$$\left| \prod_{k=i}^{p} c_k - \tilde{y}_i \right| < r^{-m}, \qquad i = 1,2,\cdots,p \tag{52}$$

if $c_i = x$, $\forall i$, the $E$-method generates the positive integral powers of $x$: $x^2, x^3, x^4, \cdots x^p$ in $(m + 1)$ steps on $p$ elementary units. In general, any $L$-reducible arithmetic expression can be evaluated in $0(m)$ steps. For example, to evaluate

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

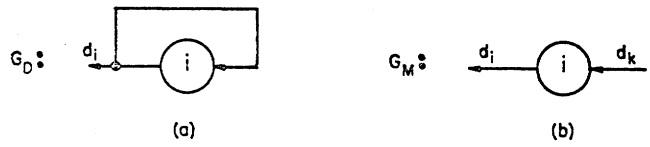


Fig. 8. Computational graphs for division and multiplication.

one may solve

$$\begin{bmatrix} 1 & -a & 0 \\ b & 1 & -c \\ 0 & d & 1 \end{bmatrix} y = \begin{bmatrix} e \\ f \\ g \end{bmatrix}.$$

Certain implications of the $E$-method on the solution of systems of linear equations, and a possible reduction of the required complexity of iterative algorithms in general, are discussed in some detail in [1]. It may be expected that the proposed computational technique can be used in many special-purpose computing systems or devices.

## V. Conclusions

In this paper, a general evaluation method, amenable to an efficient hardware-level implementation, is presented. The proposed evaluation method is characterized by several important performance features and appears applicable in many common computational problems, such as the evaluation of polynomials and rational functions. The method illustrates the importance of the following issues in the design of algorithms: 1) the choice of problem representation; 2) the redundancy of operations; and 3) the redundancy in a number representation system.

The first issue deals with minimizing the number of algorithms to be implemented in order to solve a set of different problems. As is demonstrated here, the replacement of a given set of problems by a unique, isomorphic problem provides a single algorithm to be implemented. The algorithm can, hopefully, satisfy the speed and cost objectives, among the other properties, such as the small number of different primitive operators simple to implement in a given technology, modular structure, and simple control. In the $E$-method addition appears essentially as the only required primitive operator. The problem representation, considered as a way in which the computations are to be performed, has direct implications on the available parallelism and, hence, the achievable speed. In a traditional approach, with four basic arithmetics as the primitive, indivisible operators, the parallelism, is exposed or introduced by transformations of the original computation sequence so that the time dependencies between the required operations are minimized. The $E$-method demonstrates another approach in parallelism exposition: a systematic left-to-right, digit-wise processing which minimizes the necessary delay between dependent computations and achieves a parallelism of one digit delay. This approach is also related to the second issue. Namely, it can effectively reduce the required complexity of an iteratively defined algorithm by reducing the number of necessary operations. It is of interest to remark that in many parallel algorithms it is, on the contrary, necessary to introduce redundant operations in order to achieve parallelism [19].

The problem of redundancy in number representation systems has long been recognized as a central issue in achieving efficiently fast algorithms [20], [21] and it will suffice here to note that the $E$-method is another example where redundancy in number representation has an essential role.

The method is defined for a fixed-point number representation system and this limits its effective application domain. The extension to a floating-point system is currently under investigation. The $E$-method can be incorporated in a computing system in two ways: as an autonomous arithmetic processor with several elementary units, or by providing the processors in a multiprocessor system with the capabilities of an elementary unit. It would be also of interest to consider the changes in an instruction set that could make the $E$-method efficient for use on a software level.

## Appendix

As an example of the $E$-method, we present the evaluation of $R_{3,4}(x) \approx \sinh(x)$, $x \in [0,1/8]$, with the precision of 13 decimal digits. The coefficients [2, pp. 104 and 216], before the normalization to $q_0 = 1$, are

$$p_0 = 0.0$$

$$p_1 = 0.5353890456087786*10^3$$

$$p_2 = 0.0$$

$$p_3 = 0.564627450687849*10^2$$

$$q_0 = 0.535389045608794*10^3$$

$$q_1 = 0.0$$

$$q_2 = 0.327694331123347*10^2$$

$$q_3 = 0.0$$

$$q_4 = 1.0.$$

The other parameters are: $r = 2$, $n = 5$, $\zeta = 3/4$, and $\alpha = 1/8$. No scaling was required since all $p_i/q_0$, $q_i/q_0$ satisfy the range conditions. For $x = 0.1019734533301$, the evaluation is illustrated in Fig. 9. The generated value $\tilde{y}_1$ satisfies $|(\sinh(x) - \tilde{y}_1)/\sinh(x)| < 2^{-45}$.

| j | $w_1^{(j)}$ | $d_1$ | $d_2$ | $d_3$ | $d_4$ | $d_5$ | $\tilde{y}_1$ |
|---|---|---|---|---|---|---|---|
| 1 | 0.000000000000000 | 0 | 1 | 0 | 0 | 0 | 0.000000000000000 |
| 2 | 0.20394690666018 | 0 | 0 | 0 | 0 | 0 | 0.000000000000000 |
| 3 | 0.40789381332036 | 0 | 0 | 0 | 0 | 0 | 0.000000000000000 |
| 4 | 0.81578762664072 | 1 | 0 | 0 | 1 | 0 | 0.125000000000000 |
| 5 | -0.36842474671856 | 0 | 0 | 0 | 0 | 0 | 0.125000000000000 |
| 6 | -0.73684949343712 | -1 | 0 | 1 | -1 | 0 | 0.093750000000000 |
| 7 | 0.52630101312576 | 1 | 0 | -1 | 1 | 0 | 0.109375000000000 |
| 8 | -0.94739797374848 | -1 | 0 | 0 | -1 | 0 | 0.101562500000000 |
| 9 | 0.10520405250304 | 0 | 0 | 0 | 1 | 0 | 0.101562500000000 |
| 10 | 0.21040810500608 | 0 | 1 | 1 | 0 | 0 | 0.101562500000000 |
| 11 | 0.62476311667234 | 1 | 0 | -1 | 0 | 0 | 0.102539062500000 |
| 12 | -0.75047376665532 | -1 | -1 | 1 | 0 | 0 | 0.102050781250000 |
| 13 | 0.29510556002918 | 0 | 1 | 0 | 0 | -1 | 0.102050781250000 |
| 14 | 0.79415802671854 | 1 | 0 | -1 | 0 | 0 | 0.102172851562500 |
| 15 | -0.41168394656292 | 0 | 0 | 1 | -1 | 1 | 0.102172851562500 |
| 16 | -0.82336789312584 | -1 | 1 | -1 | 1 | 0 | 0.102142333984375 |
| 17 | 0.55721112040850 | 1 | 0 | 0 | 0 | -1 | 0.102157592773437 |
| 18 | -0.88557775918300 | -1 | -1 | 1 | 0 | 1 | 0.102149963378906 |
| 19 | 0.02489757497382 | 0 | 1 | 0 | 1 | 0 | 0.102149963378906 |
| 20 | 0.25374205660782 | 0 | -1 | -1 | -1 | 0 | 0.102149963378906 |
| 21 | 0.30353720655546 | 0 | 0 | 0 | 0 | 0 | 0.102149963378906 |
| 22 | 0.60707441311092 | 1 | 0 | 1 | 1 | 0 | 0.102150440216064 |
| 23 | -0.78585117377816 | -1 | 1 | 0 | -1 | 0 | 0.102150201797485 |
| 24 | 0.63224455910386 | 1 | 0 | -1 | 1 | -1 | 0.102150321006775 |
| 25 | -0.73551088179228 | -1 | 0 | 1 | -1 | 1 | 0.102150261402130 |
| 26 | 0.52897823641544 | 1 | -1 | -1 | 0 | 0 | 0.102150291204452 |
| 27 | -1.14599043382930 | -1 | 1 | 1 | 0 | 0 | 0.102150276303291 |
| 28 | -0.08803396099842 | 0 | -1 | 0 | 1 | 0 | 0.102150276303291 |
| 29 | -0.38001482865702 | 0 | 1 | 0 | -1 | -1 | 0.102150276303291 |
| 30 | -0.55608275065386 | -1 | 0 | -1 | 0 | 1 | 0.102150274440643 |
| 31 | 0.88783449869228 | 1 | -1 | 1 | 1 | 0 | 0.102150275371933 |
| 32 | -0.42827790927562 | 0 | 0 | 0 | 0 | 0 | 0.102150275371933 |
| 33 | -0.85655581855124 | -1 | 0 | -1 | 0 | 0 | 0.102150275139129 |
| 34 | 0.28688836289752 | 0 | 1 | 0 | 1 | 0 | 0.102150275139129 |
| 35 | 0.77772363245522 | 1 | -1 | 0 | -1 | 0 | 0.102150275197339 |
| 36 | -0.64849964174974 | -1 | 0 | 0 | 1 | -1 | 0.102150275168235 |
| 37 | 0.70300071650052 | 1 | 1 | 1 | -1 | 1 | 0.102150275182787 |
| 38 | -0.39005166033878 | 0 | 0 | 0 | 0 | -1 | 0.102150275182787 |
| 39 | -0.78011332067756 | -1 | 0 | -1 | 0 | 1 | 0.102150275179151 |
| 40 | 0.43979335864488 | 0 | 1 | 0 | 0 | 0 | 0.102150275179151 |
| 41 | 1.08353362394994 | 1 | -1 | 1 | -1 | 0 | 0.102150275180060 |
| 42 | -0.03687965876030 | 0 | 0 | -1 | 0 | 0 | 0.102150275180060 |
| 43 | -0.07375931752060 | 0 | 1 | 1 | 0 | -1 | 0.102150275180060 |
| 44 | 0.05642827161898 | 0 | 0 | -1 | 0 | 0 | 0.102150275180060 |
| 45 | 0.11285654323796 | 0 | -1 | 1 | 1 | 0 | 0.102150275180060 |
| 46 | 0.02176617981574 | 0 | 1 | 0 | 0 | 1 | 0.102150275180060 |

Fig. 9. Evaluation of $\sinh(x) \approx R_{3,4}(x)$.

## ACKNOWLEDGMENT

The author wishes to thank Prof. J. E. Robertson of the University of Illinois for the invaluable advice, encouragement, and support, and Prof. D. J. Kuck, Prof. A. H. Sameh, and Prof. C. L. Liu, also of the University of Illinois, for their interest and discussions. The comments of the referees were very helpful.

## REFERENCES

[1] M. D. Ercegovac, "A general method for evaluation of functions and computations in a digital computer," Ph.D. dissertation, Dep. Comput. Sci., Univ. of Illinois at Urbana-Champaign, Urbana, Tech. Rep. 750, July 1975.

[2] J. F. Hart, *Computer Approximations.* New York: Wiley, 1968.

[3] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 330–334, Sept. 1959.

[4] J. E. Meggitt, "Pseudo division and pseudo multiplication processes," *IBM J. Res. Develop.*, vol. 6, pp. 210–226, Apr. 1962.

[5] W. H. Specker, "A class of algorithm for $\ln x$, $\exp x$, $\sin x$, $\cos x$, $\tan^{-1} x$, and $\cot^{-1} x$," *IEEE Trans. Electron. Comput.* (Short Notes), vol. EC-14, pp. 85–86, Feb. 1965.

[6] C. Tung, "A combinational arithmetic function generation system," Ph.D. dissertation, Dep. Eng., Univ. of California, Los Angeles, June 1968.

[7] B. G. DeLugish, "A class of algorithms for automatic evaluation of certain elementary functions in a binary computer," Ph.D. dissertation, Dep. Comput. Sci., Univ. of Illinois, Urbana, June 1970.

[8] J. S. Walther, "A unified algorithm for elementary functions," in *AFIPS Conf. Proc. Spring Joint Computer Conf.*, 1971, pp. 379–385.

[9] T. C. Chen, "Automatic computation of exponentials, logarithms, ratios and square roots," *IBM J. Res. Develop.*, vol. 16, pp. 380–388, July 1972.

[10] R. Franke, "An analysis of algorithms for hardware evaluation of elementary functions," Naval Post-Graduate School, Monterey, CA, AD-761519, May 1973.

[11] J. O. Campeau, "The block-oriented computer," *IEEE Trans. Comput.*, vol. C-18, pp. 706–718, Aug. 1969.

[12] ——, "Cellular redundancy brings new life to an old algorithm," *Electronics*, pp. 98–104, July 1969.

[13] ——, "Communication and sequential problems in the parallel processor," in *Parallel Processor Systems, Technologies and Applications*, L. C. Hobbs, Ed. New York: Spartan, 1970.

[14] I. T. Ho and T. C. Chen, "Multiple addition by residue threshold functions and their representation by array logic," *IEEE Trans. Comput.*, vol. C-22, pp. 762–767, Aug. 1973.

[15] D. J. Kuck, "On the speedup and cost of parallel computation," in *Proc. Complexity of Computational Problem Solving* (The Australian National Univ., Dec. 1974).

[16] H. T. Kung, "New algorithms and lower bounds for the parallel evaluation of certain rational expressions," Dep. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, Tech. Rep., Feb. 1974.

[17] A. Svoboda, "An algorithm for division," *Inform. Process. Mach.*, vol. 9, pp. 25–32, 1963.

[18] D. K. Faddeev and V. N. Faddeeva, *Computational Methods of Linear Algebra.* San Francisco: Freeman, 1963.

[19] D. J. Kuck, "Multioperation machine computational complexity," in *Proc. Symp. Complexity of Sequential and Parallel Numerical Algorithms.* New York: Academic, 1973.

[20] J. E. Robertson, "A new class of digital division methods," *IRE Trans. Electron. Comput.*, vol. EC-7, pp. 218–222, Sept. 1958.

[21] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 389–400, Sept. 1961.

**Miloš D. Ercegovac** was born in Belgrade, Yugoslavia, in 1942. He received the diploma in electrical engineering from the University of Belgrade in 1965 and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign, Urbana, IL, in 1972 and 1975, respectively.

From 1966 to 1970 he was associated with the Institute for Automation and Telecommunications "Mihailo Pupin" in Belgrade, Yugoslavia, and from 1970 to 1975 with the Department of Computer Science of the University of Illinois as a Research Assistant. He is currently an Assistant Professor in the Department of Computer Science of the School of Engineering and Applied Science, University of California, Los Angeles. His research interests include computer architecture, theory and practice of computer arithmetic, parallel computations, and analysis of algorithms.

Dr. Ercegovac is a member of the Association for Computing Machinery and an associate member of Sigma Xi.