A GENERAL METHOD FOR EVALUATION OF FUNCTIONS AND COMPUTATIONS IN A DIGITAL COMPUTER

BY

MILOŠ DRAGUTIN ERCEGOVAC

Elect. Engr., University of Belgrade, 1965
M.S., University of Illinois, 1972

THESIS

Submitted in Partial Fulfillment of the Requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1975

Urbana, Illinois

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

July, 1975

WE HEREBY RECOMMEND THAT THE THESIS BY

MILOS D. ERCEGOVAC

ENTITLED A GENERAL METHOD FOR EVALUATION OF FUNCTIONS AND COMPUTATIONS IN A DIGITAL COMPUTER

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

James E. Robertson
Director of Thesis Research

J. M. Snyder
Head of Department

Committee on Final Examination†

James E. Robertson
Chairman

D. G. Gillies

† Required for doctor's degree but not for master's.

D517

Errata (UIUCDCS-R-75-750)

Page 17: last line of Example 2.1 should be:

$= \overline{1}\overline{1}3 = \ldots$

Page 49: equation (3.1) should read:

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

and the phrase "where $a_{ii} = -1$" should be eliminated.

Page 80: last sentence should read:

. . .and utilizing (4.29), any $R\mu\nu(x)$. . .

Page 100: last line, second paragraph, should read:

$$|a_{ii}| \geq 4\,(|a_{i,i-1}| + |a_{i,i+1}|)$$

Page 107:

108: the initials of the following authors should be:

Campeau, J.O.

Volder, J.E.

Walther, J.S.

The missing reference [MEG62], mentioned on page 5 is:

[MEG62] Meggitt, J.E., "Pseudo Division and Pseudo Multiplication Processes," IBM J. Res. Develop., Vol. 6, pp. 210-226, April, 1962.

Page 30: last line of Example 2.2 should read:

$y = y^* + z^{(8)} \cdot 2^{-8} + ax^{(8)} \cdot 2^{-8}$

Page 69: equation (4.15) should read

$$\delta_b = \begin{cases} \left\lceil \log_r \dfrac{||\tilde{\underline{b}}||}{\zeta} \right\rceil & \text{if } ||\tilde{\underline{b}}|| \not< \zeta \\ 0 & \text{otherwise} \end{cases}$$

To my family

A GENERAL METHOD FOR EVALUATION OF FUNCTIONS AND COMPUTATIONS IN A DIGITAL COMPUTER

Milos Dragutin Ercegovac, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1975

This thesis presents a general computational method, amenable for an efficient implementation in digital computing systems. The method provides a unique, simple and fast algorithm for solving many computational problems, such as the evaluation of polynomials, rational functions and arithmetic expressions, or solving a class of systems of linear equations, or performing the basic arithmetics. In particular, the method is well-suited for fast evaluation of commonly used mathematical functions. The method consists of i) a correspondence rule which reduces a given computational problem f into a system of linear equations L and ii) an algorithm which generates the solution to the system L and, hence, to the problem f, in O(m) addition steps with an m-digit precision. However, the time to execute each addition step is independent of the operand precision. The algorithm is deterministic and always generates the result in a digit-by-digit fashion, the most significant digit appearing first. Therefore, the algorithm provides for an overlap in a sequence of computations as well as for a variable precision operation. The method, in general, has favorable error properties and simple implementation requirements. It is believed that the proposed method represents not only a practical and widely applicable computational approach but also an effective technique for reducing the computational complexity and increasing the speed of numerical algorithms in general.

ACKNOWLEDGMENT

I am grateful to my thesis advisor, Professor James E. Robertson, for the invaluable guidance, advice and encouragement he gave during my study at the University of Illinois and to the other members of the thesis committee, Professors D. J. Kuck, A. H. Sameh, D. B. Gillies and C. L. Liu, for their interest and advice.

The support of the Department of Computer Science of the University of Illinois and the National Science Foundation is sincerely appreciated. Thanks are also due to Mrs. June Wingler for the excellent work in typing this thesis and to Mr. Dennis L. Reed for fast printing services.

Finally, I wish to thank my wife, Zorana, for her help and patience, and my family for the encouragement and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Motivations, Objectives and Related Work

The subject of this dissertation is a novel computational method, motivated by a desire to demonstrate an approach in designing fast algorithms for numerical computations as a viable alternative to the commonly known parallel algorithms, requiring multiprocessor systems on one side, and to the strictly hardware-oriented algorithms on the other side.

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 here fast methods as i) those which use a multiplicity of general-purpose processors with the corresponding algorithms specified on the software (instruction) level, and ii) those which require special-purpose processors with algorithms embedded in the hardware (operation) level. By definition, the first class carries with it a notion of generality while the second class implies limitations of the application domain. Due to implementation properties, the former class cannot achieve the speed and the efficiency of the latter 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 available technology, becomes, in some sense, even more challenging. That is, as the cost and attainable complexity of hardware change, what are the optimal algorithms and primitive operators? It is, perhaps, a convenience, from a human point of view, to insist on implementing all and only four basic arithmetics as the primitive operators, but that hardly proves the convenience with respect to the implementation environment.

When considering a computational method for possible implementation, one is usually concerned with i) its application domain, ii) the required set of algorithms, and iii) 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, 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 which are efficiently realizable in the given implementation technology.

The objective here is to define a method which would have a sufficient generality in applications and such functional properties in order to justify, without difficulties, 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 the present chapter 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 [HAR68]. These methods, for all practical purposes, can be considered general. The other class contains those methods which 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 [FRA73]. 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 [CLE62] in time O(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 [TUN68] considered the implementation of an efficient polynomial evaluator, based on the fast primitive operators and a redundant number representation. The proposed structure is versatile but complicated 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 (three) operand additions as the primitive

operator for the evaluation of polynomial (rational) functions. A corresponding implementation can be simple, yet 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 above mentioned methods.

The second class of methods being considered, as mentioned earlier, includes those methods which 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 appears even better in this respect: its computational algorithm is simpler and problem invariant. There is no shift operator, which in many cases must have a variable shifting capability. When a redundant representation is introduced in order to make the basic computation step independent in time of the length of the operands, 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 [VOL59] and later generalized, in the form of a unified algorithm in [WAL71]; the normalization methods, based on an iterative co-transformation of a number pair (x,y) such that a function f(x,y) remains invariant as proposed in [SPE65, DEL70, CHE72];

and the pseudo-division and the pseudo-multiplication methods for some elementary functions as described in [MEG62]. 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 has even more versatility in this respect and it is generally comparable in speed. Although some of the methods mentioned above may use fewer implementation resources, the proposed method excels in simplicity with respect to the basic computing block and overall structure. 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 are easily performed by this method. Furthermore, it has useful functional properties: its computational algorithm is problem- and step-invariant; 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. These properties make the method suitable for a variable precision mode of operation. The algorithm itself 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, using systematically variable powers of two increments rather than constant increments as in DDA, and its possible application in implementing multiprocessing systems have been recognized in [CAM69a, CAM69b, CAM70]. Campeau also recognized the possibility of solving a linear system

iteratively in a left-to-right mode using a DDA-like configuration but not the constant increments.

1.2 Dissertation Overview

The main result of this work is presented in Chapter 2, as a general computational method. In its exposition, we have emphasized the basic principles of the method and presented those details which we found to have direct implications on the performance of the method. Several implementation aspects are considered in Chapter 3. The physical counterpart of the basic computational expression of the method, the elementary unit, is defined in relation to a graph model representation of the entire method. Certain properties of implementation, such as its flexibility and modularity, are discussed in sufficient detail. In Chapter 4, an attempt is made to illustrate the applicability of the proposed method in various problems. For example, the evaluation of polynomials and rational polynomial functions is considered in general and as a basis for the evaluation of various functions. Basic arithmetics and certain types of arithmetic expressions are shown to be compatible with the proposed method. Finally, the application of the method in solving the systems of linear equations is considered in some detail. A summary of the results and a discussion of the possible implications appears in the final chapter.

2. THE EVALUATION METHOD

2.1 Introduction

In this chapter a general evaluation technique, named E-method, is introduced. The E-method, in general terms, can be described as

(i) A correspondence rule, C_f, which associates independent variables $\underline{x}_f$, dependent variables $\underline{y}_f$, and parameters $\underline{p}_f$ of a given computational problem $f(\underline{x}_f, \underline{p}_f)$ with a system L of simultaneous linear equations $\underline{A}_f\underline{y} = \underline{b}_f$ in such a way that there is a one-one correspondence between dependent variables $\underline{y}_f$, i.e., the results of f, and the solution $\underline{y}$ of the system L. The elements of the matrix A_f and vector b_f must satisfy certain conditions, as specified later. Symbolically,

$$(C_f \Rightarrow \underline{A}_f, \underline{b}_f) \Rightarrow (\underline{y}_f \Leftrightarrow \underline{y} = \underline{A}_f^{-1}\underline{b}_f)$$

(ii) A computational algorithm for solving the system L in time linearly proportional to the desired number of correct digits of the solution $\underline{y}$, and which is amenable to an efficient implementation.

A computational problem f is said to be L-reducible if there is a corresponding rule C_f, not necessarily unique. The E-method is applicable in all L-reducible problems: the computational algorithm remains invariant while the particular correspondence rule, no more complex than the assignment of values, characterizes the problem.

The choice of a linear system as the target of correspondence stems from an observation that the expansion of an n-th order determinant has the form of a sum of n! terms, each term being a product of n factors. Since the solution of a linear system L appears as the ratio of the corresponding determinants, there is an obvious potential to represent and accordingly evaluate certain general arithmetic expressions, rational functions in particular, as the ratios of determinants in expanded form.

The exposition of the E-method in this chapter closely follows the order in which the fundamental ideas were developed. Thus the problem of evaluating rational functions, which alone is of sufficient importance, will be used to introduce and demonstrate the correspondence part of the E-method. Its correspondence rule, C_R, will be defined in the next section while the computational algorithm will be given in Section 2.7 after discussing in some detail what appears to be the generic problem for the E-method. The generic problem and some of the associated concepts are investigated in Sections 2.3-2.6.

2.2 The Correspondence Problem of the E-method

A simple way of establishing the correspondence C_R between the coefficients and the argument of a given rational function $R_{\mu,\nu}(x)$ and a system L of simultaneous linear equations, such that the value of $R_{\mu,\nu}$ is computed as the first component of the solution vector $\underline{y}$, is described as a general example of the correspondence problem of the E-method.

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

$$R_{\mu,\nu}(x) = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{\sum_{i=0}^{\mu} p_i x^i}{\sum_{i=0}^{\nu} q_i x^i} \qquad (2.1)$$

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

$$\underline{A}(x)\ \underline{y} = \underline{b} \tag{2.2}$$

be a nonhomogeneous system of n simultaneous linear equations with

$$\underline{A}(x) = (a_{ij})_{n\times n} \tag{2.3}$$

- the nonsingular system coefficient matrix;

$$\underline{y} = [y_1, y_2, \ldots, y_n] \tag{2.4}$$

- the solution vector and

$$\underline{b} = [b_1, b_2, \ldots, b_n] \tag{2.5}$$

- the right-hand side vector.

Let $D(x)$ denote the determinant of $\underline{A}(x)$:

$$D(x) = \det \underline{A}(x) \tag{2.6}$$

Similarly,

$$D_j(x) = \det(\underline{a}_1, \underline{a}_2, \ldots, \underline{a}_{j-1}, \underline{b}, \underline{a}_{j+1}, \ldots, \underline{a}_n) \tag{2.7}$$

where

$$\underline{a}_j = (a_{1j}, a_{2j}, \ldots, a_{nj})^t \tag{2.8}$$

is the j-th column vector.

Theorem 2.1

If $\max(\mu,\nu) \leq n-1$ and the coefficients a_{ij}'s, b_i's of the system (2.2) are put into correspondence with the coefficients p_i's, q_i's and the argument x according to the following rule C_R:

$$a_{ij} = \begin{cases} 1 & \text{for } i = j; \\ q_{i-1} & \text{for } j = 1 \text{ and } i = 2,3,\ldots,\nu+1; \\ -x & \text{for } j = i+1 \text{ and } i = 1,2,\ldots,n-1; \\ 0 & \text{otherwise;} \end{cases} \tag{2.9}$$

$$b_i = \begin{cases} p_{i-1} & \text{for } i = 1,2,\ldots,\mu+1; \\ 0 & \text{otherwise,} \end{cases} \tag{2.10}$$

then

$$y_1(x) = \frac{D_1(x)}{D(x)} = \frac{P_\mu(x)}{Q_\nu(x)} = R_{\mu,\nu}(x) \, . \tag{2.11}$$

Proof:

By the Laplace expansion of the determinants

$$D(x) = \sum_{i=1}^{n} a_{i1} \, c_{i1}(x) \tag{2.12}$$

and

$$D_1(x) = \sum_{i=1}^{n} b_i \, c_{i1}(x) \tag{2.13}$$

where

$$c_{i1}(x) = (-1)^{i+1} \det A_{i1}(x) \tag{2.14}$$

is the cofactor of the element a_{i1}, and $\det A_{i1}(x)$ is its corresponding minor, defined in the usual way. In general,

$$c_{i1}(x) = \begin{cases} \prod_{k=2}^{n} a_{kk} & \text{for } i=1; \\ (-1)^{i+1} \left[\prod_{k=1}^{i-1} a_{k,k+1} \right] \left[\prod_{k=i+1}^{n} a_{kk} \right] & \text{for } i=2,3,\ldots,n \, . \end{cases} \tag{2.15}$$

In particular,

$$c_{il}(x) = \begin{cases} 1 & \text{for } i=1; \\ (-x)^{i-1} & \text{for } i=2,3,\ldots,n \end{cases} \tag{2.16}$$

and, since

$$(-1)^{i+1}[-x]^{i-1} = x^{i-1}$$

it immediately follows that

$$D(x) = \sum_{i=1}^{n} a_{i1}\, c_{i1}(x) \tag{2.17}$$

$$= 1 + \sum_{i=2}^{\nu+1} q_{i-1}\, x^{i-1}$$

$$= 1 + \sum_{i=1}^{\nu} q_i\, x^i$$

$$= Q_\nu(x)$$

and

$$D_1(x) = \sum_{i=1}^{n} b_i\, c_{i1}(x) \tag{2.18}$$

$$= \sum_{i=1}^{\mu+1} p_{i-1}\, x^{i-1}$$

$$= \sum_{i=0}^{\mu} p_i\, x^i$$

$$= P_\mu(x) \quad .$$

Therefore,

$$y_1(x) = \frac{D_1(x)}{D(x)} = \frac{P_\mu(x)}{Q_\nu(x)} = R_{\mu,\nu}(x) \qquad \square$$

Theorem 2.1 establishes the correspondence rule C_R so that the E-method can be applied to evaluate a given rational function $R_{\mu,\nu}(x)$. The correspondence rules for several other representative problems will be given in Chapter 4. Figure 2.1 illustrates how the system L: $\underline{A}(x)\ \underline{y} = \underline{b}$ appears after an initialization has been performed according to the correspondence rule C_R.

It can be noted that the correspondence rule C_R is degenerate in a sense that only one of the n generated components y_i, $i = 1,\ldots,n$, namely, y_1 is of interest.

2.3 The Generic Problem

The proposed computational algorithm of the E-method conveniently appears to be a generalization of a solution to a rather simple problem. Namely, the problem of evaluating a linear function, subject to certain conditions, may be considered here as the generic problem in the sense that the functional properties of its algorithm are preserved in the algorithm of the E-method. Moreover, the exposition of the computational technique for the generic problem, being a scalar type, proves to be straightforward and it is immediately extendable to vector type parallel algorithms, as the one used in the E-method.

Consider the linear function

$$y = ax + b$$

where a and b are coefficients and x is the argument. While y could be trivially evaluated with the help of any multiplication algorithm, the following set of imposed properties makes the problem of evaluating y relevant for our purposes.

$$\begin{bmatrix} 1 & -x & & & & & \\ q_1 & 1 & -x & & & & \\ q_2 & & 1 & -x & & & \\ \cdot & & & \cdot & \cdot & & \\ \cdot & & & & & & \\ \cdot & & & & & & \\ q_\nu & & & & & & \\ 0 & & & & & & \\ \cdot & & & & & & \\ \cdot & & & & & & \\ \cdot & & & & & 1 & -x \\ 0 & & & & & & 1 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \cdot \\ \cdot \\ \cdot \\ \\ \\ \\ \\ y_n \end{bmatrix} = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ p_\mu \\ 0 \\ \cdot \\ \cdot \\ 0 \end{bmatrix}$$

Figure 2.1 Correspondence Rule C_R

Property 1. The algorithm must generate the most significant digits of y first in such a way that once generated, digit y_j at the step j will not be affected by any subsequent step k, $k > j$;

Property 2. The basic computational step should be invariant with the only primitive arithmetic operation being addition; the selection procedure which generates one digit of the result per step should be deterministic and feasible on a limited precision so that the step execution time is independent of the length of operands.

Property 3. The algorithm should have an "on-line" capability with respect to the independent variable. Namely, if $\{x_i | i = 1,2,\ldots,m\}$ are the digits of the independent variable x then only the digit x_j need be used at the (j+1)st step.

The first property, essential for the computational approach of the E-method, provides, in a simple yet effective way for the reduction of computation per step by preventing further involvement of previously computed entities. It implies a redundant representation of, at least, the result and it also indicates that a corresponding algorithm will belong to the class of digit-by-digit algorithms, generally known to provide a very efficient and elegant basis for hardware-oriented implementations. Step-invariance and the deterministic nature of the algorithm make the control part of implementation certainly very simple, while the requirement that addition be the only primitive operator simplifies the operational part of implementation. Furthermore, the property allowing a limited precision selection provides for a cost-effective speed-up of the algorithm through the use of a limited carry propagation mode of addition.

The last property, although easily achieved in the case of the generic problem, will be seen as essential in assuring desirable effects of Property 1 in the general algorithm of the E-method.

All numerical values considered here are assumed to be represented in a finite precision, fixed-point fractional format, with a representation error $|\epsilon| < r^{-m}$. The effect of representation errors on the E-method, as discussed in Section 2.8, is minor and causes only a slight extension of the precision required to represent initial data with respect to the prescribed precision of the result. Thus the representation errors need not be of immediate concern in the following discussions. It will be assumed that all relevant initial data have the precision of their representations properly adjusted so that, for given m, they can be regarded as exact.

Definition 2.1: 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

$$x_i \in \mathcal{D}, \ \forall \, i$$

and $\mathcal{D}$ is a given digit set.

Definition 2.2: For a given radix r, a set of consecutive integers $\mathcal{D}$ is

i) a nonredundant digit set if its cardinality satisfies

$$|\mathcal{D}| = r$$

ii) a redundant digit set if

$$|\mathcal{D}| > r$$

Definition 2.3: A symmetric redundant digit set is defined as

$$\mathcal{D}_\rho = \{-\rho, -(\rho-1), \ldots, -1, 0, 1, \ldots, \rho-1, \rho\}$$

where

$$\frac{r}{2} \leq \rho \leq r - 1$$

assuming, for simplicity, an even radix r. In particular, $\mathcal{D}_\rho$ is

i) minimally redundant if

$$|\mathcal{D}_\rho| = r + 1$$

so that

$$\rho = \frac{r}{2}$$

ii) maximally redundant if

$$|\mathcal{D}_\rho| = 2r - 1$$

so that

$$\rho = r - 1 .$$

Consequently, the representation of a number x is redundant or nonredundant depending whether $x_i \epsilon \mathcal{D}_\rho$ or $x_i \epsilon \mathcal{D}$. In the case of a redundant representation

$$\text{sign } x \equiv \text{sign } x_1$$

so $x = \sum_{i=1}^{m} x_i \, r^{-i}$.

Example 2.1. For radix r = 4

$$\mathcal{D} = \{0,1,2,3\}$$

$$\mathcal{D}_{\rho min} = \{\overline{2},\overline{1},0,1,2\}$$

$$\mathcal{D}_{\rho max} = \{\overline{3},\overline{2},\overline{1},0,1,2,3\}$$

where the overbar denotes the negative sign, i.e., $\overline{2} = -2$. Then

$$\begin{aligned} x = -23_{10} &= -113_4 && \text{for } x_i \in \mathcal{D} \\ &= \overline{12}1 = \overline{2}21 && \text{for } x_i \in \mathcal{D}_\rho \\ &= \overline{11}3 = \overline{1}3\overline{13} = \overline{1}333\overline{2}1 \ \ldots && \text{for } x_i \in \mathcal{D}_{\rho max} . \end{aligned}$$ □

Theorem 2.2

Let

$$w^{(j)} = d^{(j)} + z^{(j)} = r(z^{(j-1)} + a\, x^{(j-1)}) \tag{2.19}$$

be the basic recursion where

j is the recursion index;

r is the radix;

$d^{(j)} \in \mathcal{D}_\rho$ is the j-th generated digit of the result;

$z^{(j)}$ is the j-th residual such that

$$|z^{(j)}| \leq \zeta, \ \forall\, j,$$

the bound ζ satisfying

$$\frac{1}{2} \leq \zeta < 1;$$

a is the given coefficient such that

$$|a| \leq \alpha,$$

the bound α satisfying

$$0 < \alpha \leq \frac{1}{r}\left[1 - \frac{\zeta(r-1)}{\rho}\right]$$

and

$x^{(j)} = x_j$ is the j-th digit of the independent variable x.

Then the following selection procedure, defined as a step-invariant function $s(w^{(j)})$:

$$d^{(j)} = s(w^{(j)}) = \begin{cases} \text{sign } w^{(j)} \lfloor |w^{(j)}| + 1/2 \rfloor & \text{for } |w^{(j)}| \leq \rho \\ \text{sign } w^{(j)} \lfloor |w^{(j)}| \rfloor & \text{otherwise} \end{cases}$$

where $\lfloor w \rfloor$ denotes the integer part of w, can generate in m steps, given $z^{(0)} = b$, a sequence of digits $(d^{(1)}, d^{(2)}, \ldots, d^{(m)})$ such that

$$|y - y^*| < r^{-m}$$

where

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

and

$$y = a \sum_{j=1}^{m} x^{(j)} r^{-j} + b \ .$$

Proof:

The selection function $s(w^{(j)})$, for practical purposes, represents a rounding procedure, modified at the endpoints of the domain to avoid digit values $|d^{(j)}| > \rho$. It simply maps a w-subinterval [k-1/2, k+1/2) to an integer k, the subinterval boundaries being assigned as close or open as convenient.

The consistency of the recursion formula (2.19), with respect to the selection function $s(w^{(j)})$ is established by proving inductively the boundedness of the residuals $\{z^{(j)}\}$. By the condition of the theorem:

$$|z^{(0)}| \leq \zeta \ .$$

Then, assuming

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

it follows that

$$\begin{aligned} |w^{(j)}| &\leq r|z^{(j-1)}| + r|a\, x^{(j-1)}| \\ &\leq r\,\zeta + r\left[\frac{1}{r}\left(1 - \frac{\zeta(r-1)}{\rho}\right)\right]\rho \\ &= \rho + \zeta \end{aligned}$$

By definition of the selection function $s(w^{(j)})$, the choice of digit $d^{(j)}$ can always be made such that

$$|z^{(j)}| = |w^{(j)} - d^{(j)}| \leq \zeta ,$$

as desired.

To show the convergence of the corresponding algorithm, it may be first established, by substitution, that the k-th residual satisfies the following equality:

$$z^{(k)} = r^{k}\left[b + a \sum_{j=1}^{k-1} x^{(j)}\, r^{-j} - \sum_{j=1}^{k} d^{(j)}\, r^{-j}\right], \quad k=1,2,\ldots$$

By definition

$$y = a\,x + b = a \sum_{j=1}^{m} x^{(j)}\, r^{-j} + b$$

and

$$y^{*} = \sum_{j=1}^{m} d^{(j)}\, r^{-j}$$

Therefore

$$y - y^{*} = r^{-m}(z^{(m)} + a\, x^{(m)})$$

so that

$$|y - y^*| \leq r^{-m}(|z^{(m)}| + |a||x^{(m)}|)$$

$$\leq r^{-m}(\zeta + \alpha\,\rho)$$

$$< r^{-m}$$

since

$$\zeta + \alpha\,\rho = \frac{\zeta+\rho}{r} < 1 \quad \text{for} \quad \zeta < 1,\ \rho \leq r - 1 \qquad \square$$

It may be noted that it takes (m+1) steps to form all 2m digits of y since

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

and adding the last residual can be done by concatenation. One extra step results from the way in which the recursion formula (2.19) was defined.

Theorem 2.2 indicates precisely an algorithm having all of the required properties. It does not, however, indicate explicitly how addition in the recursive formula can be replaced by a limited carry propagation addition so that the same simple selection function $s(w^{(j)})$ will remain applicable. For that purpose the selection problem and the relationships between ζ, α and ρ bounds need to be analyzed in more detail.

2.4 An Analysis of the Selection Problem

The selection function s(w) can be interpreted as a conventional rounding rule, conforming to the given conditions on ζ, α and ρ bounds. Therefore the digit selection process itself can be carried out in a deterministic fashion: no tests need to be performed since the result of the selection is exactly the integer part of the rounded $w^{(j)}$. This proved to be

a particularly practical way of performing selection when a higher radix is utilized [ERC73]. While rounding itself needs no further elaboration, a more detailed analysis of this selection problem appears useful in demonstrating that the basic recursion can be performed in time independent of the length of the operands.

The selection procedure is considered here to be defined by a function

$$s : W \rightarrow \mathcal{D}_\rho \tag{2.20}$$

where the domain

$$W = \{w^{(j)} | w^{(j)} \in I = [-\rho - \zeta, \rho + \zeta]\} \tag{2.21}$$

is the finite set of values $w^{(j)}$, defined by the recursion formula (2.19), and the range $\mathcal{D}_\rho$ is a redundant digit set.

Let

$$I_k = [\ell_k, u_k] , \quad k \in \mathcal{D}_\rho \tag{2.22}$$

be a subinterval of I, with lower and upper endpoints ℓ_k and u_k, such that if

$$w^{(j)} \in I_k$$

then

$$d^{(j)} = s(w^{(j)}) = k$$

is a valid digit choice.

For the continuity of the domain W, the overlap between the adjacent subintervals, defined as

$$\Delta_i = u_i - \ell_{i+1} , \quad i \in \mathcal{D}_\rho - \{\rho\} \tag{2.23}$$

must satisfy

$$\Delta_i \geq - r^{-m} \tag{2.24}$$

assuming m digit precision of the $w^{(j)}$ representation. The linear form of the recursion formula and the selection function $s(w^{(j)})$ imply that

$$\Delta_i = \Delta , \quad \forall i \tag{2.25}$$

Since the selection, in principle, is performed by comparing $w^{(j)}$ to a fixed set of interval breakpoints - comparison constants $\{c_i\}$ as

$$d^{(j)} = \begin{cases} k & c_k \leq w^{(j)} < c_{k+1} \\ k+1 & c_{k+1} \leq w^{(j)} < c_{k+2} , \end{cases} \tag{2.26}$$

it is important to maximize the overlap Δ between subintervals so that a convenient choice of comparison constants as "simple," low precision numbers (e.g., 1/2, 1/4, etc.) can be made. Moreover, the precision to which $w^{(j)}$ must be computed for selection in order to be correct, is of the same order as the precision of the overlap Δ. Thus a sufficiently large overlap can make the time necessary to perform the recursion independent of the precision of the operands. Let the comparison constant c_k satisfy

$$c_k = \ell_{k+1} + \frac{\Delta}{2} = u_k - \frac{\Delta}{2} \tag{2.27}$$

Then, instead of performing selection as in (2.26), it can be carried out as

$$d^{(j)} = \begin{cases} k & c_k \leq \hat{w}^{(j)} < c_{k+1} \\ k+1 & c_{k+1} \leq \hat{w}^{(j)} < c_{k+2} \end{cases} \tag{2.28}$$

provided that

$$|w^{(j)} - \hat{w}^{(j)}| \leq \frac{\Delta}{2} \tag{2.29}$$

where $\hat{w}^{(j)}$ is computed by any limited carry propagation technique. To satisfy (2.29), the carry needs to be assimilated only in $\sim|\log_r|\Delta||$ most significant positions.

The effects of the overlap Δ on the considered selection problem can be summarized as follows:

i) If $\Delta=0$ or $\Delta=-r^{-m}$, the full precision value of $w^{(j)}$ must be used in the selection even though the comparison constants are simple, i.e.,

$$c_k = k - \frac{1}{2}, \quad k\epsilon\mathcal{D}_\rho \tag{2.30}$$

Furthermore, to satisfy the consistency requirement of the recursion formula (2.19), the following must hold

$$|c_k - k| \leq \zeta, \quad \forall\, k\epsilon\mathcal{D}_\rho \tag{2.31}$$

The domain continuity implies

$$c_{k+1} - k - r^{-m} \leq \zeta \tag{2.32}$$

so that the residual bound must satisfy

$$\begin{aligned} \zeta &\geq \frac{1-r^{-m}}{2} \quad \text{if } \Delta = -r^{-m} \\ \zeta &\geq \frac{1}{2} \quad \text{if } \Delta = 0 \end{aligned} \tag{2.33}$$

This establishes the lower upper bound on the residuals with respect to the defined selection function. Such a result is intuitively clear since the value of the lower upper bound on residual $z^{(j)}$ cannot be less than one half of the smallest positive digit $d^{(j)}$ value if the rounding is to be used. The selection function $s(w^{(j)})$, in the case where the full precision is used, is

illustrated in Figure 2.2. The z-w graph is analogous to the SRT division chart [ROB58].

ii) If the overlap $\Delta > 0$, the benefits of a precision independent speed can be introduced in a rather simple way. Namely, by redefining the residual bound ζ to satisfy

$$\frac{1}{2}(1+\Delta) \leq \zeta < 1 \qquad (2.34)$$

for the given overlap Δ, the same comparison constants $\{c_i\}$, and hence the selection function s as in the full precision case, may be retained while using a low precision selection argument $\hat{w}^{(j)}$ rather than $w^{(j)}$. An example of the corresponding z-w graph with an overlap $\Delta = 1/2$ and the residual bound $\zeta = 3/4$ is given in Figure 2.3. For the selection function $s(\hat{w}^{(j)})$ to satisfy

$$s(\hat{w}^{(j)}) = s(w^{(j)})$$

i.e., to work correctly, $\hat{w}^{(j)}$ must be computed so that

$$|w^{(j)} - \hat{w}^{(j)}| \leq 1/4 ,$$

a trivial constraint indeed.

The potential for a computation time independent of the precision of operands is implicit in the recursion formula (2.19): it requires a rather superficial change in the bound ζ and an appropriate representation of $w^{(j)}$ allowing for the simple calculation of $\hat{w}^{(j)}$.

2.5 The Relationships Between α, ζ and ρ Bound

It can be observed that there is a strong interdependence between the bounds, α on the coefficient a, ζ on the residuals $\{z^{(j)}\}$ and the maximal value ρ of the digit set, by definition. The values chosen for

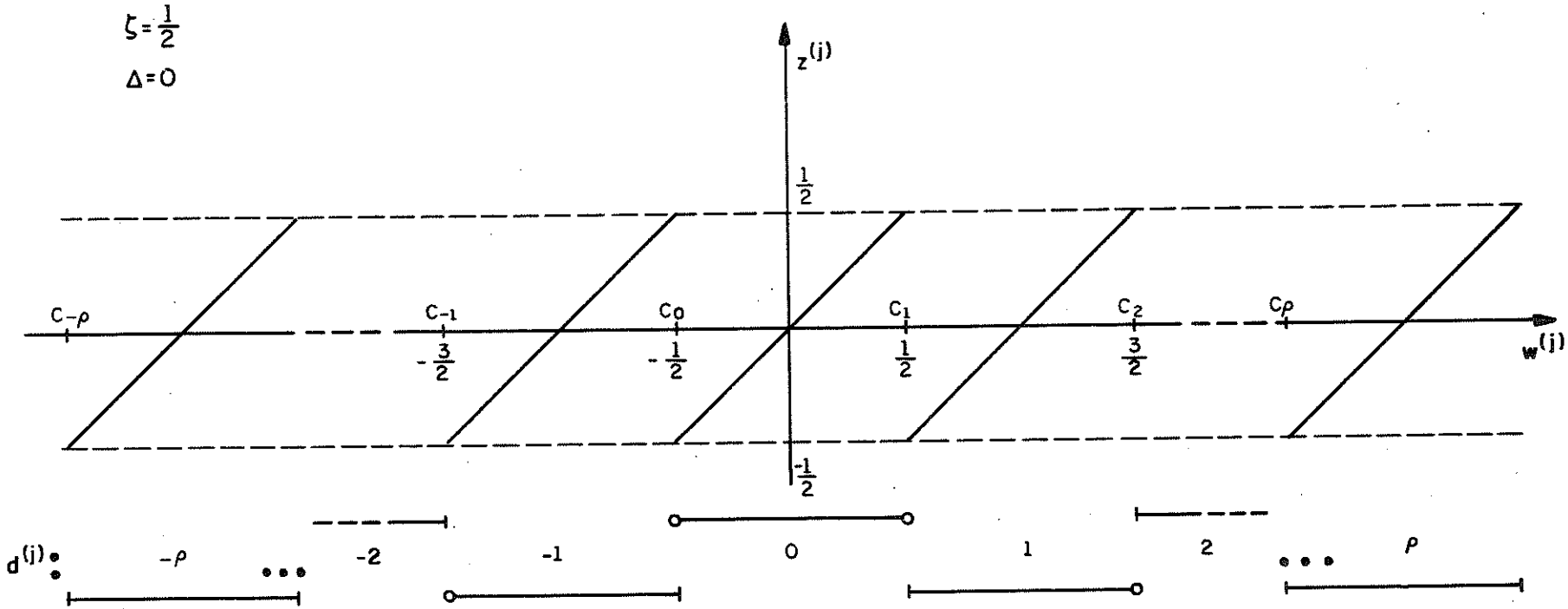


Figure 2.2 Full Precision Selection Procedure

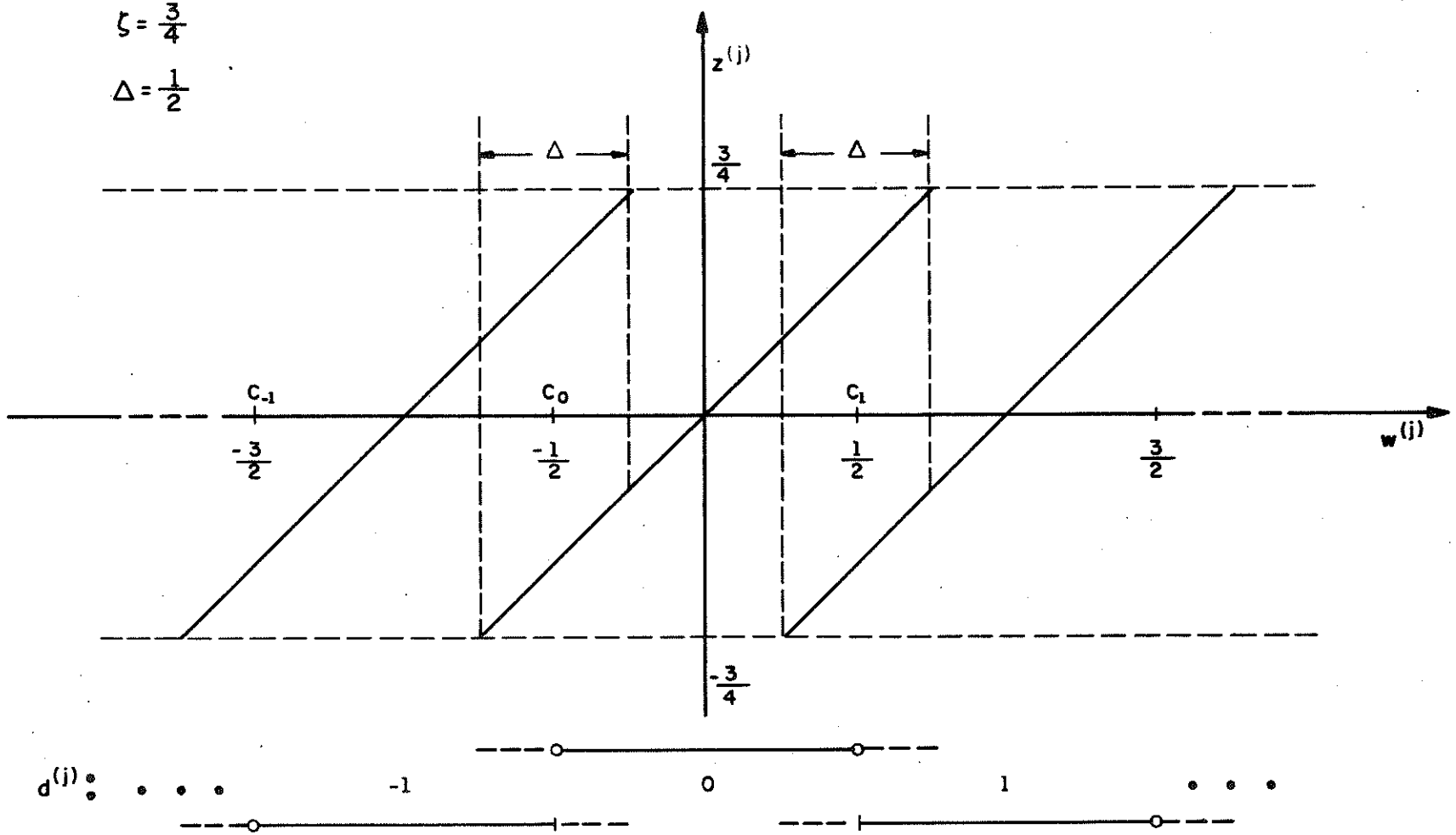


Figure 2.3 Limited Precision Selection Procedure

these bounds largely determine the basic computational properties of the generic algorithm.

The bound α on the coefficient a,

$$0 < \alpha \leq \frac{1}{r}\left[1 - \frac{\zeta(r-1)}{\rho}\right] < \frac{1}{r}, \tag{2.35}$$

is defined in this particular way so that the consistency requirement of Theorem 2.2 can be easily satisfied. It immediately follows that

$$\zeta < \frac{\rho}{r-1} \tag{2.36}$$

since

$$\alpha > 0 .$$

Also, because $\rho \leq r-1$,

$$\frac{1}{2} \leq \zeta < 1, \tag{2.37}$$

the lower bound on ζ being established earlier on the basis of the selection function consideration (2.33). Furthermore, when the selection subinterval overlap Δ is used, ζ bound is increased by $\Delta/2$, thus,

$$\Delta < \frac{2\rho}{r-1} - 1 \tag{2.38}$$

which, for a minimally redundant digit set $\mathcal{D}_\rho$, with $\rho = \frac{r}{2}$, becomes

$$\Delta < \frac{1}{r-1} \tag{2.39}$$

and for a maximally redundant digit set, where $\rho = r-1$,

$$\Delta < 1, \tag{2.40}$$

confirming a generally known property that more redundancy in the representation system provides for a faster and easier computation.

The bounds α and ζ determine required scaling for arbitrary coefficients a and b. It may be noted that, again, increased redundancy

reduces scaling although not significantly. The scaling presents no problems in the generic algorithm but, in general it would be desirable to maximize the bound α. However, since $\alpha < \frac{1}{r}$, there is little justification, not at least in a practical sense, to constrain ζ or ρ in order to increase α.

A summary of the relationships among the bounds, with respect to the operating modes of interest, is given in Table 2.1.

2.6 The Algorithm G

Although Theorem 2.2 itself specifies the algorithm for solving the generic problem, this will be formalized again here as the Algorithm G for reference convenience.

Algorithm G

/ Initialization /

1. $z^{(0)} \leftarrow b$; $x^{(0)} \leftarrow 0$;

/ Recursion /

2. for $j = 1,2,\ldots,m$:

2.1 $w^{(j)} \leftarrow r(z^{(j-1)} + ax^{(j-1)})$;

2.2 $d^{(j)} \leftarrow s(w^{(j)})$;

2.3 $z^{(j)} \leftarrow w^{(j)} - d^{(j)}$;

/ Termination /

HALT

/ The result is $y^* = \sum_{j=1}^{m} d^{(j)} r^{-j}$ / □

Table 2.1 Relationships Between Bounds

Mode of Operation:	Redundancy in $\mathcal{D}_\rho$:	
	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 $\zeta = \frac{1}{2}(1+\Delta)$ $0 < \Delta < \frac{2\rho}{r-1} - 1$	$\alpha \leq \frac{1}{r^2}[1-\Delta(r-1)]$	$\alpha \leq \frac{1}{2r}[1-\Delta]$

Example 2.2

Compute $y = ax + b$ using the Algorithm G for

$a = 0.00101011$ $\quad r = 2$

$b = 0.01011001$ $\quad \mathcal{D}_\rho = \{\bar{1}, 0, 1\}$

$x = 0.10111001$ $\quad m = 8$

j	$x^{(j)}$	$w^{(j)}$	$d^{(j)}$	$z^{(j)}$
1	1	0.1011001	1	-0.0100111
2	0	-0.0100011	0	-0.0100011
3	1	-0.1000110	$\bar{1}$	0.0111010
4	1	1.0011111	1	0.0011111
5	1	0.1101001	1	-0.0010111
6	0	-0.0000011	0	-0.0000011
7	0	-0.0000110	0	-0.0000110
8	1	-0.0001100	0	-0.0001100

The computed solution is

$$y^* = \sum_{j=1}^{8} d^{(j)} 2^{-j} = 0.10\bar{1}11000 = 0.01111000$$

while $y = 0.011110000010011$. It can be easily verified that

$$y = y^* + z^{(8)} 2^{-8} + ax^{(8)} . \qquad \square$$

2.7 The Computational Algorithm of the E-method

The computational part of the E-method consists of a unique algorithm for solving the system L, defined for a particular problem f by its correspondence rule C_f. By definition, the generated solution of the system L is, in its totality or in part, also the solution of the original problem f. This algorithm, referred to as Algorithm E, is a direct generalization of the generic, scalar Algorithm G for an n

dimensional vector space, n being the order of the system L. It retains all of the important properties of the generic algorithm: it also belongs to the class of digit-by-digit, left-to-right methods and generates the solution in at most m+1 recursive steps, if supported by a configuration of n identical elementary units. However, the Algorithm E, allows in a straightforward manner for compromises between speed and implementation complexity. Furthermore, the selection function s and the bounds α, ζ and ρ, as defined in the generic problem, remain applicable.

Let $j = 1,2,\ldots,m+1$ denote the recursion index. Define the j-th digit vector $\underline{d}^{(j)}$ as

$$\underline{d}^{(j)} = [d_1^{(j)}, d_2^{(j)}, \ldots, d_n^{(j)}] \; ; \tag{2.41}$$

the j-th residual vector $\underline{z}^{(j)}$ as

$$\underline{z}^{(j)} = [z_1^{(j)}, z_2^{(j)}, \ldots, z_n^{(j)}] \tag{2.42}$$

and their vector sum $\underline{w}^{(j)}$ as

$$\underline{w}^{(j)} = \underline{d}^{(j)} + \underline{z}^{(j)} = [w_1^{(j)}, w_2^{(j)}, \ldots, w_n^{(j)}] \tag{2.43}$$

where

$$w_i^{(j)} = d_i^{(j)} + z_i^{(j)} \; , \quad i = 1,2,\ldots,n \; , \quad \forall \, j \; .$$

Let $s(\underline{w}^{(j)})$ be the vector selection function

$$s(\underline{w}^{(j)}) = [s(w_1^{(j)}), s(w_2^{(j)}), \ldots, s(w_n^{(j)})] \tag{2.44}$$

such that

$$d_i^{(j)} = s(w_i^{(j)}) \; , \quad i = 1,2,\ldots,n \; , \quad \forall \, j$$

and each $s(w_i^{(j)})$ is defined as in Theorem 2.2 of the generic problem. Furthermore, define the n-th order matrix $\underline{G}(x)$ as

$$\underline{G}(x) = \underline{I} - \underline{A}(x) = (g_{ij})_{n\times n} \tag{2.45}$$

where $\underline{I}$ is the identity matrix and $\underline{A}(x)$ is the coefficient matrix of the system L defined by the correspondence rule C_f. The maximum vector norm

$$\|\underline{x}\|_\infty = \max_i |x_i| \tag{2.46}$$

and the consistent matrix norm

$$\|\underline{A}\|_\infty = \max_i \left(\sum_{j=1}^{n} |a_{ij}| \right), \tag{2.47}$$

as the only norms considered, will be denoted as $\|\underline{x}\|$ and $\|\underline{A}\|$, respectively.

Theorem 2.3

If

$$\|\underline{G}(x)\| \leq \alpha; \tag{2.48}$$

$$\|\underline{b}\| \leq \zeta; \tag{2.49}$$

and

$$d_i^{(j)} \in \mathcal{D}_\rho, \ \forall_i \ \forall_j$$

then the following n-th order system of linear recursions

$$\underline{w}^{(j)} = \underline{d}^{(j)} + \underline{z}^{(j)} = r(\underline{z}^{(j-1)} + \underline{G}(x)\ \underline{d}^{(j-1)}) \ , \tag{2.50}$$

$$j = 1,2,\ldots,m+1$$

with the initial conditions

$$\underline{d}^{(0)} = 0 \ ,$$

$$\underline{z}^{(0)} = \underline{b} \ ,$$

generates m leftmost correct digits of the solution

$$\underline{y} = \underline{A}^{-1}(x)\ \underline{b}$$

in at most m+1 steps as the sequence $\{\underline{d}^{(j)}\}$, $\underline{d}^{(j)}$, $\forall_j$, selected according to (2.44). Namely, the generated solution

$$\underline{y}^* = \sum_{j=1}^{m+1} \underline{d}^{(j)} r^{-j} = [\sum_{j=1}^{m+1} d_1^{(j)} r^{-j}, \ \ldots, \ \sum_{j=1}^{m+1} d_n^{(j)} r^{-j}] \tag{2.51}$$

satisfies

$$\|\underline{y} - \underline{y}^*\| < r^{-m} \ . \tag{2.52}$$

Proof:

The consistency of the system of recursions with respect to the selection function (2.44) is proved inductively. By the statement of the theorem,

$$\|\underline{z}^{(0)}\| \leq \zeta$$

Assume that

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

Then

$$\begin{aligned} \|\underline{w}^{(j)}\| &= \|\underline{d}^{(j)} + \underline{z}^{(j)}\| \qquad (2.53) \\ &= r\|\underline{z}^{(j-1)} + \underline{G}(x)\ \underline{d}^{(j-1)}\| \\ &\leq r\|\underline{z}^{(j-1)}\| + r\|\underline{G}(x)\|\cdot\|\underline{d}^{(j-1)}\| \\ &\leq r\ \zeta + r\ \alpha\ \rho \\ &= r\ \zeta + r[\frac{1}{r}(1 - \frac{\zeta(r-1)}{\rho}] \ \rho \\ &= \rho + \zeta \end{aligned}$$

Since

$$\underline{z}^{(j)} = \underline{w}^{(j)} - \underline{d}^{(j)} \text{ and sign } d_i^{(j)} = \text{sign } w_i^{(j)} ,$$

by definition of the selection function $s(\underline{w}^{(j)})$, it immediately follows that

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

The convergence is proved by showing that the solution error vector

$$\underline{h} = [h_1, h_2, \ldots, h_n] = \underline{y} - \underline{y}^* \tag{2.54}$$

satisfies

$$\|\underline{h}\| < r^{-m} . \tag{2.55}$$

After m+1 steps the following holds:

$$\underline{d}^{(m+1)} + \underline{z}^{(m+1)} = r^{m+1}\,\underline{b} + \underline{G}(x)[\sum_{j=1}^{m} \underline{d}^{(j)} r^{m+1-j}] \tag{2.56}$$

$$- \underline{I}[\sum_{j=1}^{m} \underline{d}^{(j)} r^{m+1-j}]$$

or

$$r^{-m-1}(\underline{d}^{(m+1)} + \underline{z}^{(m+1)}) = \underline{b} - \underline{A}(x)[\sum_{j=1}^{m} \underline{d}^{(j)} r^{-j}]$$

or

$$r^{-m-1}\,\underline{z}^{(m+1)} = \underline{b} - \underline{A}(x)\cdot\underline{y}^* - \underline{G}(x)\,\underline{d}^{(m+1)} r^{-m-1} . \tag{2.57}$$

Let

$$\underline{e}^{(m+1)} = [e_1^{(m+1)}, e_2^{(m+1)}, \ldots, e_n^{(m+1)}] = (\underline{z}^{(m+1)} + \underline{G}(x)\underline{d}^{(m+1)})r^{-m-1} \tag{2.58}$$

Since $\underline{y} = \underline{A}^{-1}(x)\ \underline{b}$,

$$\underline{h} = \underline{y} - \underline{y}^{*} = \underline{A}^{-1}(x)\cdot(-\ \underline{e}^{(m+1)}) \ . \tag{2.59}$$

The error after m+1 steps is, therefore, bounded by

$$\|\underline{h}\| = \|\underline{A}^{-1}(x)(-\ \underline{e}^{(m+1)})\| \tag{2.60}$$

$$\leq \|\underline{A}^{-1}(x)\|\cdot\|\underline{e}^{(m+1)}\| \ .$$

Since $\|\underline{G}(x)\| \leq \alpha < 1$ by definition for $r \geq 2$, the matrix $\underline{G}(x)$ is convergent, i.e.,

$$\lim_{p\to\infty} [\underline{G}(x)]^{p} = 0$$

By the well-known result [FAD63]:

$$\underline{A}^{-1}(x) = [\underline{I} - \underline{G}(x)]^{-1} = \lim_{p\to\infty} (\underline{I} + \underline{G}(x) + \underline{G}^{2}(x) + \ldots + \underline{G}^{p}(x)) \tag{2.61}$$

so that

$$\|\underline{A}^{-1}(x)\| = \|\underline{I} + \underline{G}(x) + \underline{G}^{2}(x) + \ldots\| \tag{2.62}$$

$$\leq \|\underline{I}\| + \|\underline{G}(x)\| + \|\underline{G}(x)\|^{2} + \ldots$$

$$\leq 1 + \sum_{p=1}^{\infty} \alpha^{p}$$

$$= \frac{1}{1-\alpha} \leq \frac{4}{3}$$

Also

$$\|\underline{e}^{(m+1)}\| = r^{-m-1}\|\underline{z}^{(m+1)} + \underline{G}(x)\ \underline{d}^{(m+1)}\| \tag{2.63}$$

$$\leq r^{-m-1}(\zeta+\alpha\rho) \ .$$

Therefore,

$$\|\underline{h}\| \leq \frac{1}{1-\alpha} \cdot \frac{\zeta+\alpha\rho}{r} \cdot r^{-m} = \gamma \cdot r^{-m} \tag{2.64}$$

where

$$\gamma = \frac{1}{2(r-1)} \tag{2.65}$$

for minimally redundant digit set and

$$\gamma = \frac{1}{r} \tag{2.66}$$

for maximally redundant digit set. Since $\gamma < 1$ for $r > 1$

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

Theorem 2.3 completes the general specification of the computational part of the E-method. Clearly, the recursion formula (2.50) may be computed in time independent of the operands precision, provided that the approximate, low-precision value $\hat{\underline{w}}^{(j)}$ of the sum $\underline{w}^{(j)}$ satisfies the selection requirement

$$\|\underline{w}^{(j)} - \hat{\underline{w}}^{(j)}\| \leq \frac{\Delta}{2} \tag{2.67}$$

where Δ is the given overlap between the selection subintervals, as discussed in the generic problem.

It may be seen from the recursions (2.50) how Property 1, one-step dependent generation of digits $\{\underline{d}^{(j)}\}$, and Property 3, on-line restriction on the usage of digits $\{\underline{d}^{(j)}\}$, as introduced in the generic problem, are critical for simplicity and speed of the Algorithm E: not only is the computation of the recursion formula reduced to additions but the effective delay between two successive applications of the recursion formula is equivalent to the time necessary to generate just a single digit. Some implications of these properties on the complexity and possible speed-up

of computations will be discussed later. Note that the argument of the original problem f appears, after initialization, as a parameter in the system of recursions (2.50), and, at a particular step j, only $\underline{d}^{(j)}$ is considered as the independent variable. The rate of convergence of the Algorithm E is, by definition, restricted in that it must be linear with respect to the radix, i.e., one radix r digit per step. However, the resulting computational simplicity of the recursive formula and, equally important, the effective way of introducing parallelism more than compensate for such a restriction.

The error properties and the scaling problem of the E-method will be discussed next, while a summary of the E-method with an example concludes this chapter. Certain aspects of the E-method related to the implementation and performance evaluation will be considered in the following chapter.

2.8 An Error Analysis of the E-method

The error behavior of the E-method appears to be quite favorable. First, it can be seen that if the system of linear equations L satisfies the bounds, as required by Theorem 2.3, then it is insensitive to small perturbations in elements of $\underline{A}(x)$ and $\underline{b}$ since the corresponding condition number of the matrix $\underline{A}(x)$ satisfies

$$\kappa(\underline{A}(x)) = \|\underline{A}(x)\| \cdot \|A^{-1}(x)\| \tag{2.68}$$

$$= \frac{1+\alpha}{1-\alpha}$$

$$\leq \frac{5}{3} \quad .$$

Second, by definition of the computational algorithm, no roundoff errors are generated when E-method is applied. However, the finiteness in the

representations of the elements of $\underline{A}(x)$ and $\underline{b}$ inevitably introduces representation errors which will systematically propagate to the left and eventually invalidate selection of digits $\{d_i^{(j)}\}$ and hence destroy the accuracy of the result. In view of (2.68), it is clear that, by considering representation errors as perturbations of the correct values of the elements of $\underline{A}(x)$ and $\underline{b}$, no serious difficulty should be expected. Indeed, it will be shown that the ill-effects of these representation errors can be compensated for by an extended precision m' in the coefficients representations with respect to the prescribed precision m of the result.

To determine $m' > m$ such that the first m+1 digit vectors $\underline{d}^{(j)}$ are correctly selected, assume that

$$\underline{\tilde{G}}(x) = \underline{G}(x) + \underline{E}_G(x) = (g_{ij})_{n\times n} + (e_{ij})_{n\times n} = (\tilde{g}_{ij})_{n\times n} \tag{2.69}$$

represents the matrix of exact elements $\{\tilde{g}_{ij}\}$ while $\underline{G}(x)$ is their finite precision representation matrix with the corresponding error matrix $\underline{E}_G(x)$ such that

$$|e_{ij}| < r^{-m'}, \ \forall_i \ \forall_j \tag{2.70}$$

Similarly,

$$\underset{\sim}{w}^{(j)} = \underline{w}^{(j)} + \underline{e}_w^{(j)} \tag{2.71}$$

and

$$\underset{\sim}{z}^{(j)} = \underline{z}^{(j)} + \underline{e}_z^{(j)} \tag{2.72}$$

Since, by definition of Algorithm E

$$w_i^{(j)} - d_i^{(j)} = z_i^{(j)}, \ \forall_i$$

then

$$\tilde{w}_i^{(j)} - e_{w_i}^{(j)} - d_i^{(j)} = z_i^{(j)}$$

or

$$\tilde{z}_i^{(j)} = z_i^{(j)} + e_{w_i}^{(j)} = z_i^{(j)} + e_{z_i}^{(j)}$$

so that

$$e_{z_i}^{(j)} = e_{w_i}^{(j)} \quad \text{and} \quad \underline{e}_z^{(j)} = \underline{e}_w^{(j)} \tag{2.73}$$

By substitution, applying (2.73), it follows that

$$\underline{e}_w^{(m+1)} = r^{m+1}[\underline{e}_z^{(0)} + \underline{E}_G \cdot \sum_{j=1}^{m} \underline{d}^{(j)}\, r^{-j}] \tag{2.74}$$

For the selection to be correct, the discrepancy between the true and computed selection argument must satisfy

$$\|\underline{e}_w^{(m+1)}\| = \|\tilde{\underline{w}}^{(m+1)} - \underline{w}^{(m+1)}\| \leq \frac{\Delta}{2} \tag{2.75}$$

Since

$$\|\underline{e}_w^{(m+1)}\| \leq r^{m+1}(\|\underline{e}_z^{(0)}\| + \|\underline{E}_G(x)\| \cdot \|\sum_{j=1}^{m} \underline{d}^{(j)}\, r^{-j}\|)$$

and

$$\|\underline{e}_z^{(0)}\| = \underline{e}_b < r^{-m'} \ ;$$

$$\|\underline{E}_G(x)\| \leq \max_i \sum_{j=1}^{n} |e_{ij}| \ ; \tag{2.76}$$

$$\|\sum_{j=1}^{m} \underline{d}^{(j)}\, r^{-j}\| < 1 \ ,$$

it can be found that the precision of coefficient representation must satisfy

$$m' \geq m + 1 + \log_r(\frac{2}{\Delta} n') \tag{2.77}$$

where n' is the maximum number of possible nonzero elements in any row of the matrix $\underline{A}(x)$. In the most important applications considered here, n' is very small. For example, in the case of the rational function evaluation problem, n' = 3 so that, for r = 2 and $\Delta = 1/2$, $m' \geq m + 1 + \log_2 12 \approx m + 5$.

2.9 The Scaling Problem

The conditions (2.48) and (2.49) of Theorem 2.3 on norms of matrix $\underline{A}(x)$ and vector $\underline{b}$, imply that, in general, an adjustment of the size of the elements a_{ij} and b_i will be required. Although scaling commonly appears whenever fixed-point representation arithmetic is used, it can be handled without serious difficulties. The E-method, however, requires more consideration of the scaling problem.

The scaling problem of the E-method will be considered here in general terms, with specifics to be given later for particular applications. The simplest problem in scaling which may arise is that

$$\|\underline{A}(x)\| \leq 1 + \alpha \tag{2.78}$$

but

$$\|\underline{b}\| \not\leq \zeta$$

However, instead of solving

$$\underline{A}(x)\ \underline{y} = \underline{b}$$

one can solve an equivalent system, scaled as follows

$$\underline{A}(x)\ \underline{S}\ \underline{y} = \underline{S}\ \underline{b} \tag{2.79}$$

or

$$\underline{A}(x)\ \underline{y}' = \underline{b}'$$

where

$$\|\underline{b}'\| = \|\underline{S}\ \underline{b}\| \leq \zeta$$

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

$$s_{ij} = \begin{cases} r^{-\sigma} & \text{for } i=j \\ 0 & \text{for } i \neq j \end{cases} \tag{2.80}$$

where σ is a positive integer such that

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

or

$$\sigma = \left\lceil \log_r \frac{\|\underline{b}\|}{\zeta} \right\rceil$$

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

$$\underline{y} = \underline{S}^{-1}\ \underline{y}'$$

Multiplications with $\underline{S}$ and $\underline{S}^{-1}$ involve only a shift of σ positions right and left, respectively. Therefore, the case (2.78) is always trivially solvable and one need be concerned only with the case when matrix $\underline{A}(x)$ requires scaling.

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

$$|\hat{a}_{ii}| \geq k|\hat{a}_{ij}|\ , \quad \forall j \neq i\ ,$$

k is a given constant. This, in general, requires prohibitively complex computation in view that $\underline{A}(x)$ depends on independent variables of the original problem. For example, consider a general technique which can be used to reduce a given system of linear equation to a form, convenient for iteration [DEM73]. By definition, in our case, $\det \underline{A}(x) \neq 0$ so

$$(\underline{A}^{-1}(x) - \underline{S})\ \underline{A}(x)\ \underline{y} = (\underline{A}^{-1}(x) - \underline{S})\ \underline{b} \tag{2.81}$$

or

$$\underline{y} = \underline{S}\ \underline{A}(x)\ \underline{y} + (\underline{A}^{-1}(x) - \underline{S})\ \underline{b}$$

where $\underline{S}$ is the scaling matrix (2.80), being defined so that

$$\|\underline{S}\ \underline{A}(x)\| \leq 1 + \alpha$$

The form, although slightly different from that of Algorithm E, is acceptable and matrix $\underline{A}(x)$ has been effectively scaled. However, computation of $(\underline{A}^{-1}(x) - \underline{S})\ \underline{b}$ would require an amount of work incomensurable with the expected performance of the E-method. There are, fortunately, many important applications where scaling of matrix $\underline{A}(x)$ does not appear to be a problem. It is of practical importance that certain problems allow for redefinition in a convenient way so that scaling again is simple matter.

With respect to scaling, two classes of applications of E-method can be distinguished. In one class there are problems, for instance, the evaluation of functions, where all parameters are known a priori so that scaling is a one-time job for a particular argument range. In this class, the E-method applies without any overhead. In the other class are the problems with arbitrary parameters so that scaling must be performed each time the E-method is applied. This involves an overhead, commonly

encountered in any other method of computation in a fixed-point representation domain. The applications of the E-method in this class would make the extension of the method to a floating-point representation domain highly desirable. It may be noted that the E-method is easily applicable when a block-floating point representation is used.

2.10 Summary of the E-method

The E-method is summarized below for reference convenience:

Part 1 (Correspondence)

Given an L-reducible computational problem f, apply the correspondence rule C_f on the arguments and parameters of f in order to obtain the coefficient matrix $\underline{A}$ and the vector $\underline{b}$ of the system of linear equations L. The correspondence rule C_f must guarantee that the elements of $\underline{A}$ and $\underline{b}$ can be made conformable to the conditions:

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

$$|b_i| \leq \zeta , \quad \forall\, i$$

Part 2 (Computation)

Algorithm E

/ Initialization /

1. $\underline{z}^{(0)} \leftarrow \underline{b}$; $\underline{G}(x) \leftarrow \underline{I}-\underline{A}(x)$; $\underline{d}^{(0)} \leftarrow 0$;

/ Recursion /

2. for j = 1,2,...,m+1:

2.1 $\underline{w}^{(j)} \leftarrow r(\underline{z}^{(j-1)} + \underline{G}(x)\, \underline{d}^{(j-1)})$;

2.2 $\underline{d}^{(j)} \leftarrow s(\underline{w}^{(j)})$;

2.3 $\underline{z}^{(j)} \leftarrow \underline{w}^{(j)} - \underline{d}^{(j)}$;

/ Termination /

3. HALT

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

$$\underline{y}^{*} = \sum_{j=1}^{m+1} \underline{d}^{(j)}\, r^{-j}$$

$$= [y_1^{*}, y_2^{*}, \ldots, y_n^{*}] \qquad /$$

Example 2.3:

As a general example of the E-method we present the evaluation of $R_{3,4}(x)$ as an approximation to sinh(x), $x \in [0,1/8]$, with a precision of 13 decimal digits. The coefficients are taken from [HAR68, SINH2002, p. 104, p. 216]. Before normalizing q_0 to 1, they appear as follows:

$p_0 = 0.0$

$p_1 = 0.5353890456087786 * 10^3$

$p_2 = 0.0$

$p_3 = 0.5646274506878 49 * 10^2$

$q_0 = 0.5353890456087 94 * 10^3$

$q_1 = 0.0$

$q_2 = -0.3276943311233 47 * 10^2$

$q_3 = 0.0$

$q_4 = 1.0$

The other parameters are: $r = 2$, $n = 5$, $\zeta = 3/4$, $\alpha = 1/8$, $q_b = 1$, $m' = 44 + 1 + 1 = 46$. For

$$x = 0.1019734533301,$$

the evaluation is illustrated in Figure 2.4. The generated value y_1^* satisfies

$$|(\sinh(x) - y_1^*)/\sinh(x)| < 2^{-45} . \qquad \square$$

j	$w_1^{(j)}$	d_1	d_2	d_3	d_4	d_5	y_1^*
1	0.00000000000000	0	1	0	0	0	0.00000000000000
2	0.20394690666018	0	0	0	0	0	0.00000000000000
3	0.40789381332036	0	0	0	0	0	0.00000000000000
4	0.81578762664072	1	0	0	1	0	0.12500000000000
5	-0.36842474671856	0	0	0	0	0	0.12500000000000
6	-0.73684949343712	-1	0	1	-1	0	0.09375000000000
7	0.52630101312576	1	0	-1	1	0	0.10937500000000
8	-0.94739797374848	-1	0	0	-1	0	0.10156250000000
9	0.10520405250304	0	0	0	1	0	0.10156250000000
10	0.21040810500608	0	1	1	0	0	0.10156250000000
11	0.62476311667234	1	0	-1	0	0	0.10253906250000
12	-0.75047376665532	-1	-1	1	0	0	0.10205078125000
13	0.29510556002918	0	1	0	0	-1	0.10205078125000
14	0.79415802671854	1	0	-1	0	0	0.10217285156250
15	-0.41168394656292	0	0	1	-1	1	0.10217285156250
16	-0.82336789312584	-1	1	-1	1	0	0.10214233398437
17	0.55721112040850	1	0	0	0	-1	0.10215759277343
18	-0.88557775918300	-1	-1	1	0	1	0.10214996337890
19	0.02489757497382	0	1	0	1	0	0.10214996337890
20	0.25374205660782	0	-1	-1	-1	0	0.10214996337890
21	0.30353720655546	0	0	0	0	0	0.10214996337890
22	0.60707441311092	1	0	1	1	0	0.10215044021606
23	-0.78585117377816	-1	1	0	-1	0	0.10215020179748

Figure 2.4 Evaluation of $\sinh(x) \approx R_{3,4}(x)$

24	C.6322445591C386	1	0	-1	1	-1	0.10215032100677
25	-0.73551088179228	-1	0	1	-1	1	0.10215026140213
26	C.52897823641544	1	-1	-1	0	0	0.10215029120445
27	-1.14599043382930	-1	1	1	0	0	0.10215027630329
28	-0.08803396099842	0	-1	0	1	0	0.10215027630329
29	-C.38001482865702	0	1	0	-1	-1	0.10215027630329
30	-0.55608275065386	-1	0	-1	0	1	0.10215027444064
31	C.88783449869228	1	-1	1	1	0	0.10215027537196
32	-0.42827790927562	0	0	0	0	0	0.10215027537196
33	-0.85655581855124	-1	0	-1	0	0	0.10215027513913
34	C.28688836289752	0	1	0	1	0	0.10215027513913
35	0.77772363245522	1	-1	0	-1	0	0.10215027519734
36	-0.64849964174974	-1	0	0	1	-1	0.10215027516824
37	0.70300071650052	1	1	1	-1	1	0.10215027518279
38	-0.39005166033878	0	0	0	0	-1	0.10215027518279
39	-C.7801C332067756	-1	0	-1	0	1	0.10215027517915
40	0.43979335864488	0	1	0	0	0	0.10215027517915
41	1.08353362394994	1	-1	1	-1	0	0.10215027518006
42	-0.03687965876030	0	0	-1	0	0	0.10215027518006
43	-0.07375931752060	0	1	1	0	-1	0.10215027518006
44	C.05642827161898	0	0	-1	0	0	0.10215027518006
45	0.11285654323796	0	-1	1	1	0	0.10215027518006
46	0.02176617981574	0	1	0	0	1	0.10215027518006

Figure 2.4 Evaluation of $\sinh(x) \approx R_{3,4}(x)$ (continued)

3. ON IMPLEMENTATION AND PERFORMANCE OF THE E-METHOD

3.1 Introduction

We introduce in this chapter a definition of the basic computing block, the elementary unit EU, and consider, in some detail, its implementation. Then, a graph model of a general computing configuration, constructed by connecting several elementary units, is defined as a convenient form of representation, exhibiting directly the basic parameters which determine the performance of the E-method. Finally, we discuss several modes of operation, in which the E-method can be applied, in relation to the general implementation properties of the method.

3.2 The Basic Computational Block

The basic computational block, the elementary unit EU_i, is a hardware structure implementing the basic recursion formula (2.50) or, more precisely, the recursion step of the Algorithm E. Due to its simplicity, it seems sufficient to indicate only the major parts and a control strategy of an EU in the present discussion, without considering low-level details of an actual implementation. Hopefully, our global description will suffice to estimate precisely enough the performance and complexity features of an EU, and, consequently of the E-method.

It seems preferable, from the implementation point of view, to restate the basic recursion formula (2.50) as follows:

$$w_i^{(j)} = r(w_i^{(j-1)} + \sum_{k=1}^{n} a_{ik}\, d_k^{(j-1)}) \qquad (3.1)$$

where $a_{ii} = -1$, so that the need for explicitly calculated residuals $z_i^{(j)}$ is avoided.

As indicated by Figure 3.1, where a global structure of the elementary unit EU_i is shown, the evaluation of $w_i^{(j)}$ basically requires an s-operand adder. In many practical applications s is rather small. In order for the time of addition to be independent of operand precision, $w_i^{(j)}$ need to be represented in a redundant form.

The selection procedure, defined by the selection function $s(\hat{w}_i^{(j)})$ is performed by the block S, which forms $\hat{w}_i^{(j)}$ by converting a few 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 precision of $\hat{w}_i^{(j)}$ is basically determined by the overlap Δ and the number s of summands. All functions of the selection block S can be easily implemented. The previous value $d_i^{(j-1)}$ is saved in register D. The coefficients $\{a_{ik}\}$ may be, for better storage efficiency and simpler adder structure, represented in a nonredundant form. 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

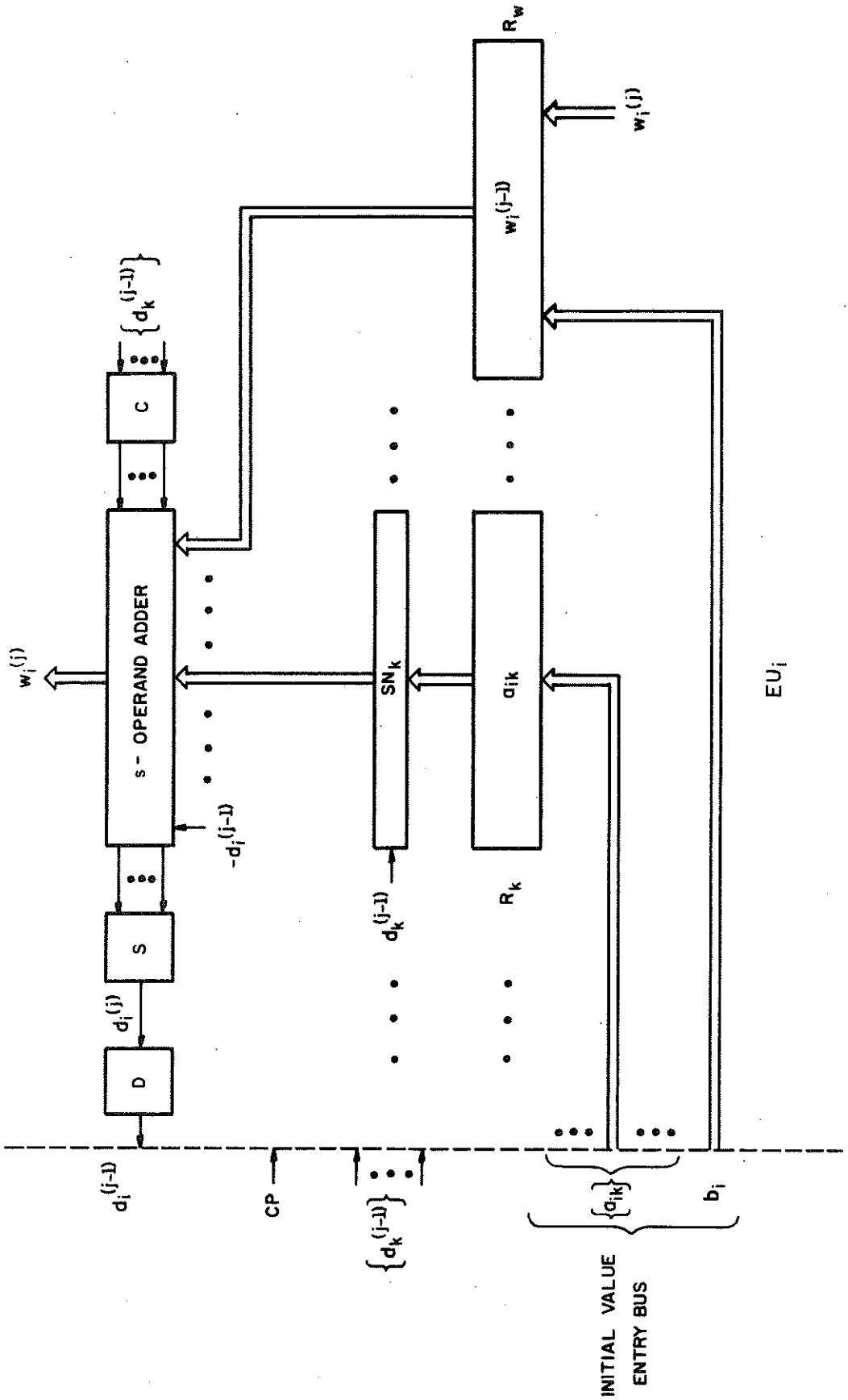


Figure 3.1 Elementary Unit: Global Structure

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 problem of finding an optimal radix under given application and implementation conditions will not be considered here.

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 $\underline{G}(x)$, 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 the initial value entry bus.

The control requirements of an EU are very simple: assuming a synchronous mode of operation of the entire configuration of the elementary units, synchronizing, clock pulses on which the transfer of $w_i^{(j)}$ into R_w occurs, are all that is needed. The same clock pulses, defining the basic step, are distributed to all units. By controlling their number, one can easily achieve a variable precision mode of operation, as discussed later.

The time required to perform one recursive step on an elementary unit is defined as:

$$t_O = t_A + t_S + t_T$$

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 s of summands and the adder structure. For practical reasons, s 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 s-2 levels of full-adder rows, will have a $t_A = 2(s-2)\ t_g$, assuming $2t_g$ per full-adder. More sophisticated adder structures, i.e., Dadda-type [H073] can considerably reduce this time. However, when s is small, like in polynomial evaluation, it can be seen that, for radix 2, $t_O = O(10t_g)$.

3.3 A Graph Representation of a General Computing Configuration

An L-reducible problem f of order n can be solved by the E-method on a structure consisting of n interconnected elementary units. 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 $\underline{d}$. This implies that the physical connection between the units EU_i and EU_j only needs to accommodate a transfer of one, signed radix r digit. In common multiprocessor structures, used for fast parallel computations, the processor intercommunications usually require full precision width.

A computing structure for solving a given problem f by the E-method may be conveniently specified by a computational graph

$$G_f(V,\underline{K}) \qquad (3.2)$$

where $V = \{V_i | i=1,\ldots,n\}$ is a set of vertices and $\underline{K}$ is a connection matrix, defining a set of directed arcs. Each vertex V_i corresponds to an elementary unit EU_i, symbolically represented as in Figure 3.2 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 $\underline{K} = (k_{ij})_{n\times n}$ is in one-one correspondence with the matrix $\underline{G}(x) = \underline{I} - \underline{A}(x)$ of the system L, as follows:

$$k_{ij} = \begin{cases} 1 & \text{if } g_{ij} \neq 0 \\ 0 & \text{if } g_{ij} = 0 \end{cases} \tag{3.3}$$

and $k_{ij} = 1$ specifies that the arc d_j is an incoming arc for the vertex V_i, i.e., the connection matrix $\underline{K}$ defines the relation "receives from" between the elementary units $\{EU_i\}$ of the computing structure. Note that the utilization of d_i for internal functions in EU_i, as indicated in Figure 3.1, need not be specified on the graph unless $g_{ii} \neq 0$.

No attempt is presently being made to treat the properties and applications of the above defined computational graphs in a rigorous and extensive way. Rather, some basic observations are made and several examples are given to illustrate usefulness of computational graphs in the E-method.

Strictly speaking, there is only one functional primitive, the elementary unit, which appears in any application of the E-method. Yet for convenience, four types of nodes, obtainable by obvious modifications of the elementary unit, are defined as primitives in a sense that no G_f, where f is an L-reducible problem, exists which cannot be constructed from these four primitives.

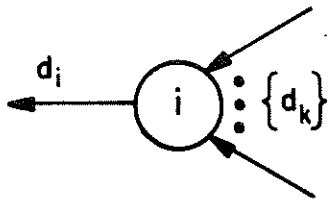


Figure 3.2 Elementary Unit: Graph Representation

The conversion primitive G_C, shown in Figure 3.3 (a), satisfies the following output function:

$$\sum_{j=1}^{\infty} d_i^{(j)} r^{-j} = c_{i0} \tag{3.4}$$

where c_{i0} is a constant, possibly in a nonredundant form. If $d_i^{(j)} \epsilon \mathcal{D}$, as is the case in the E-method, the G_C primitive performs a serial conversion of c_{i0} into the corrresponding redundant form. For completeness, one could define a primitive without an outgoing arc to perform serial conversion into a nonredundant form. However, such a primitive would require a facility for full-precision carry propagation, unlike the other primitives. For that reason, we prefer to assume that the conversion to a conventional, nonredundant representation, when necessary, would be done in a separate addition step, on a separate unit.

The multiply primitive G_M, Figure 3.3 (b), satisfies the input-output function:

$$\sum_{j=1}^{\infty} d_i^{(j)} r^{-j} = c_{i0} + c_{i1} \sum_{j=1}^{\infty} d_k^{(j)} r^{-j} \tag{3.5}$$

and it is a degenerate case of the inner product primitive G_I, Figure 3.3 (c), which satisfies

$$\sum_{j=1}^{\infty} d_i^{(j)} r^{-j} = c_{i0} + \sum_{j=1}^{\infty} \sum_{\substack{k=1 \\ k \neq i}}^{\infty} c_{ik} d_k^{(j)} r^{-j} \tag{3.6}$$

The divide primitive G_D, Figure 3.3 (d), satisfies

$$\sum_{j=1}^{\infty} d_i^{(j)} r^{-j} = \frac{c_{i0}}{1-c_{i1}} \tag{3.7}$$

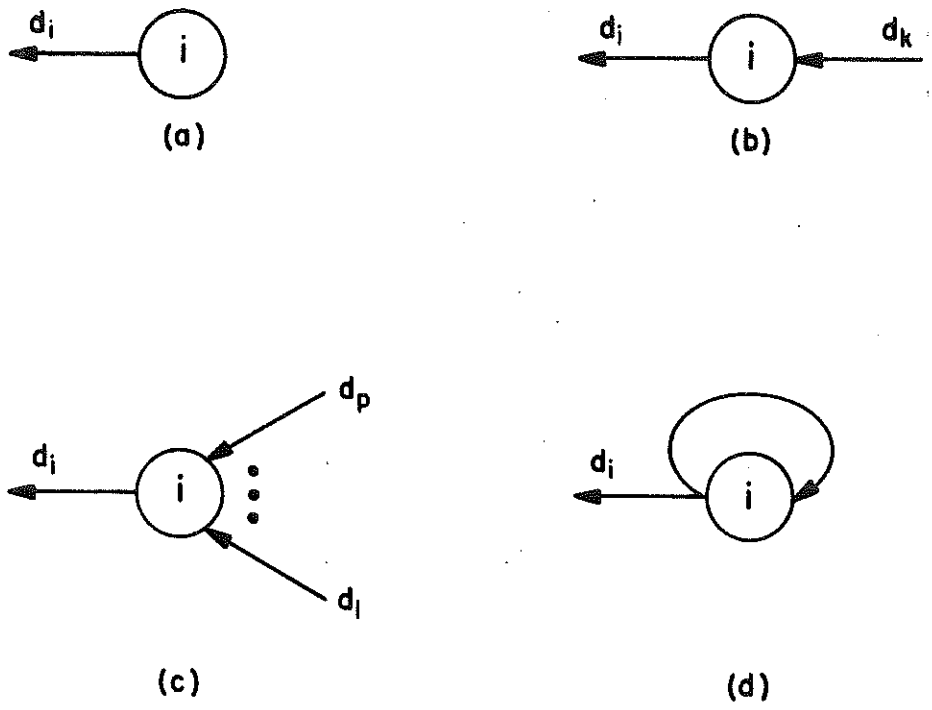


Figure 3.3 Computational Primitives

In all primitives, c_{ik} are the coefficients assumed to satisfy the conditions of Theorem 2.3. For example, the computing structure for the evaluation of a polynomial has a graph G_P, as in Figure 3.4 (a), the evaluation of a rational function has a graph G_R, Figure 3.4 (b), while the evaluation of an expression like

$$f = c_0 + \sum_{k=1}^{p} c_k \left(\frac{a_k}{1-b_k} \right) \tag{3.8}$$

requires a structure with a graph G_f, Figure 3.4 (c).

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

Importantly, a graph G_f provides a direct estimate of the required time and implementation complexity. If $N(\underline{K})$ denotes the number of ones in the connection matrix $\underline{K}_{n \times n}$ then the computational structure requires at most $N(\underline{K}) + 2n$ m-digit registers, n adders with total of $N(\underline{K}) + 2n$ operands and $N(\underline{K})$ 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 $s = (N_i+2)$ operand adder and that many registers.

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

$$T_E(f) = (m+1)\, t_0 \tag{3.9}$$

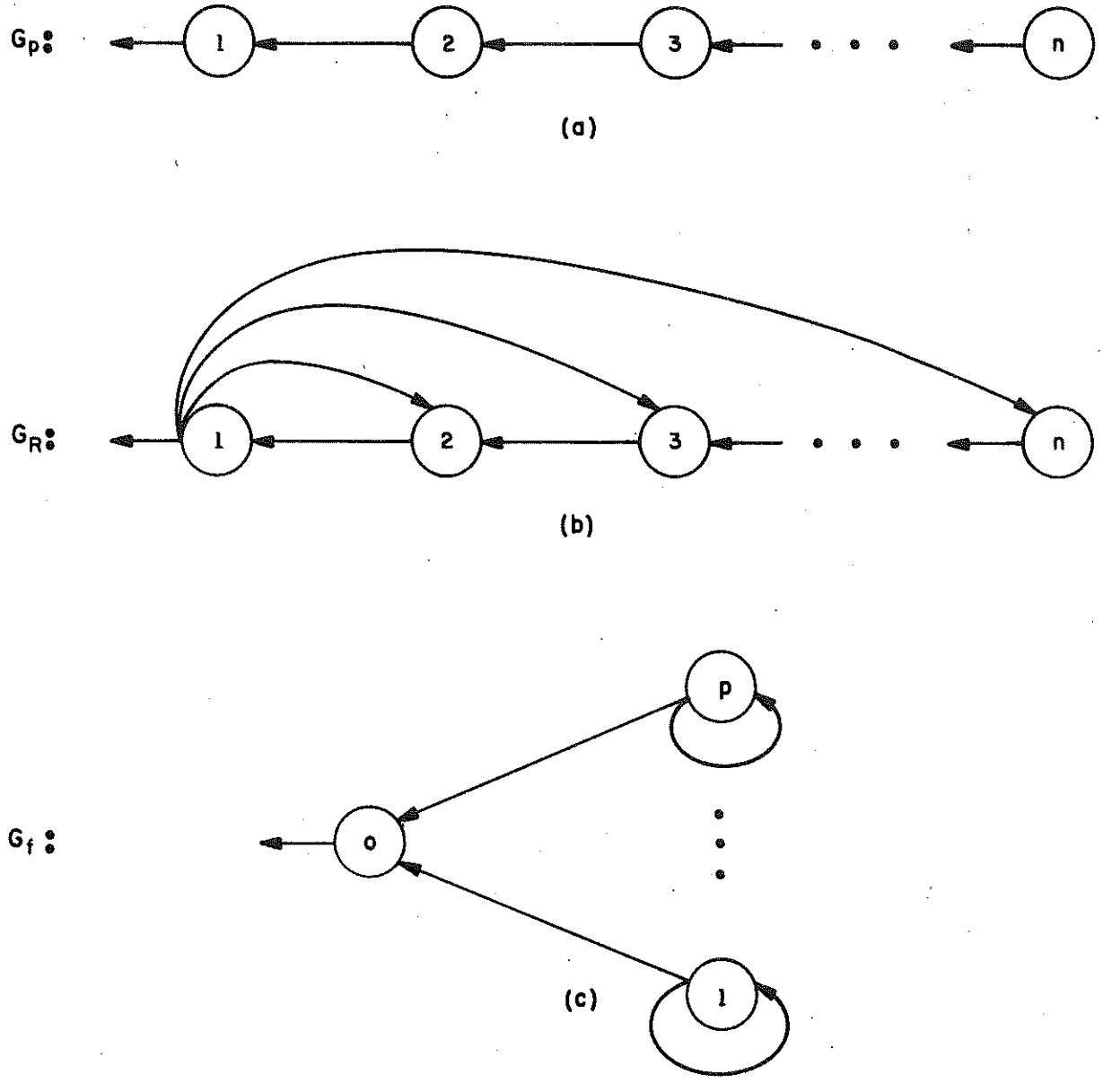


Figure 3.4 Examples of Computational Structures

if no scaling is required, and

$$T_E(f) = (m+1+\sigma)\ t_o \qquad (3.10)$$

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

3.4 On Modes of Operation and Implementation

The functional properties of the E-method, namely the step-invariant nature of its computational algorithm and linearity of its basic operator, make an adaptation both to a variable precision mode of operation and a variable number of elementary units rather straightforward.

A variable precision mode of operation is very desirable from the user's point of view. Since the computational algorithm of the E-method is deterministic, in the sense that no tests need to be performed to check the attained accuracy at each step, the desired accuracy of the result is specified directly by the given number of recursive steps. This leads in a natural way to a variable precision operation. The second aspect, i.e., the ability of the method to perform without severe degradation while using the limited resources, or its implementation flexibility in other words, is also of practical importance.

Consider the application of the E-method in a given problem of order $\tilde{n}$ and precision $\tilde{m}$. Let n denote the number of available elementary units of precision m. When $\tilde{n} \leq n$ and $\tilde{m} \leq m$, clearly, no problem exists in achieving a variable precision mode of operation: the given value of $\tilde{m}$, declared by the user, can control the number of steps to be executed by the elementary units and, hence, the precision.

If, however, $\tilde{m} > m$, each elementary unit should be augmented with additional storage to accommodate the extra precision of the operands. The control of the elementary units should be modified so as to allow multiple precision addition, with the selection process carried only when the most significant word of the sum has been generated. As shown in Figure 3.5, the additional storage could be conveniently implemented, for example, when $\tilde{m} = \ell m$, as a circular array of ℓ parallel-in, parallel-out registers, connected to a corresponding operand register in the elementary unit via already existing bus (Figure 3.1).

The same solution is applicable in the case when $\tilde{n} > n$, i.e., when the number of elementary units is insufficient. Now each array row of the additional storage would contain the data corresponding to one recursion. Also, storage for the digit vector $\underline{d}$ must be provided in the same way. Again, the control modifications are minor.

Finally, one can consider the case where the elementary unit itself has a restriction on the number s of operands it can handle simultaneously. It is easily seen that the same approach as above will suffice to accommodate the necessary number $\tilde{s}$ of operands. Let

$$\begin{aligned} k_n &= \left\lceil \frac{\tilde{n}}{n} \right\rceil \\ k_m &= \left\lceil \frac{\tilde{m}}{m} \right\rceil \\ k_s &= \left\lceil \frac{\tilde{s}}{s} \right\rceil \end{aligned} \tag{3.11}$$

then the computation time is affected as follows:

$$T_E(f) \approx k_n\, k_m\, k_s (m+1)\, t_0 \tag{3.12}$$

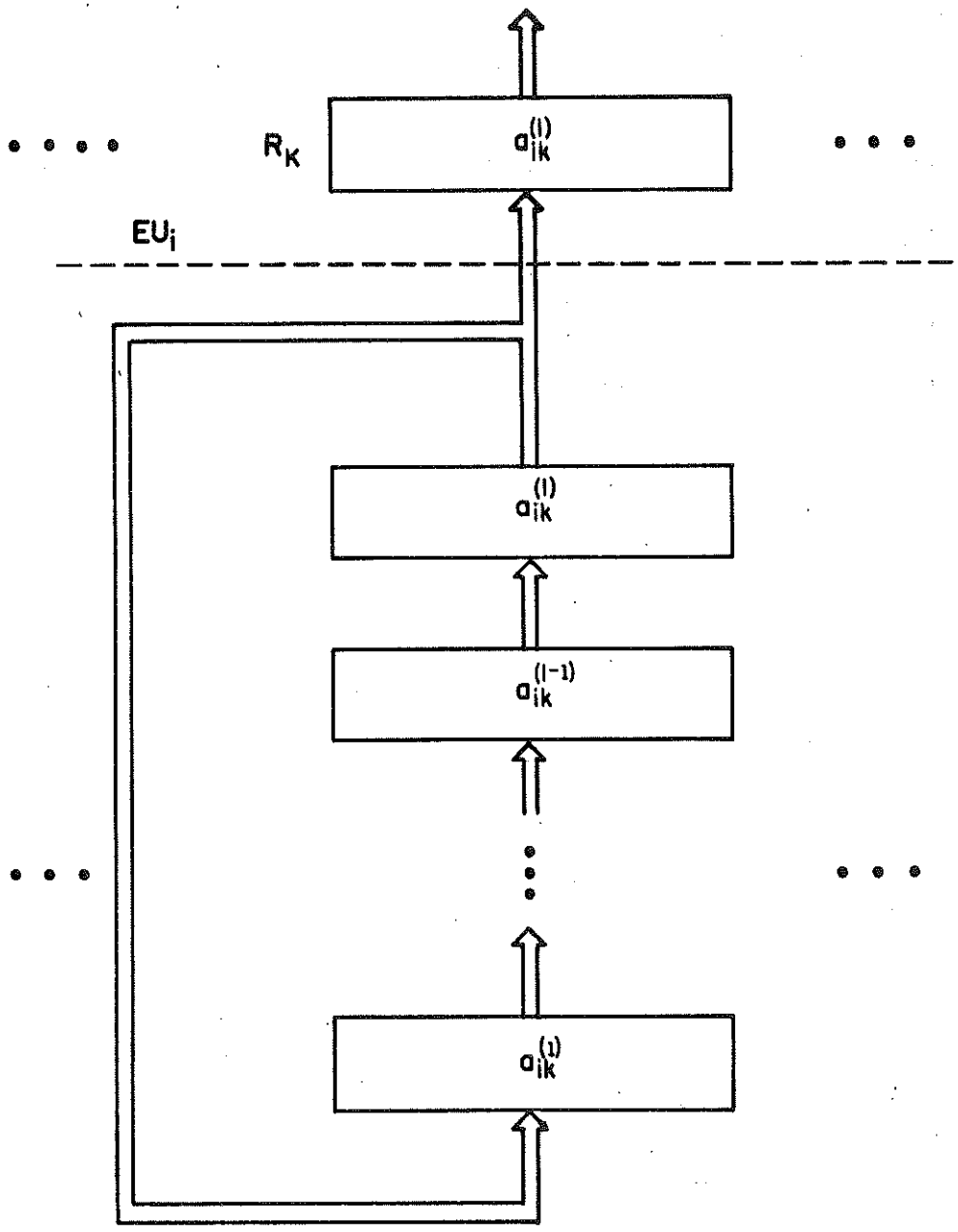


Figure 3.5 Multiple Precision Scheme

where n, m, s, and t_0 are the parameters of the available elementary units and $\tilde{n}$, $\tilde{m}$, and $\tilde{s}$ are parameters of the given problem.

The previous discussion reveals the flexibility of the E-method: it can be implemented under a wide range of speed/cost constraints in a simple way. The cost change in precision, in the number of elementary units or in their complexity, affects the speed of computation linearly. In addition, the E-method is clearly amenable to a modular implementation. For example, the basic module, implementated in a LSI technique, could be a 16 bit unit with a 4-operand adder, 4 registers and a selection and carry block which can be by-passed so that a larger precision elementary unit could be simply constructed by concatenating the required number of basic modules. An additional module could be a 8×16 bits circular register array. By combining these two types of modules one could easily achieve a desired computing structure to support the E-method.

It is of certain interest to discuss the modes of operation of a computing structure from another point of view. Since Algorithm E always generates the results in a digit fashion, the most significant digit generated first, an "on-line" mode of operation seems to be a natural way to provide for a faster execution of sequences of interrelated computations. In other words, if Algorithm E can be made to satisfy the "on-line" property (cf. p. 14) with respect to all operands appearing in the basic recursion, a significant overlap and, hence the speed-up in computing can be easily achieved. In general, the "on-line" mode of operation implies that the j-th digit of the result can be generated before the $(j+\delta)$-th digits of the operands are available. Algorithm E already satisfies the "on-line" property with respect to the digit vector $\underline{d}$ and it is a rather

straightforward modification of the basic recursion which is required in order to extend the same property to all operands. We mention also that the "on-line" capability of the E-method can be particularly important in a possible real-time application.

4. ON APPLICATIONS OF THE E-METHOD

4.1 Introduction

In this chapter several L-reducible problems are described as the representative examples of some applications of the E-method. The selected applications are, we believe, important and illustrative of the many aspects of the proposed method: its computational power, versatility, simplicity of implementation as well as certain limitations, resulting mainly from the number range restrictions.

We begin by describing the evaluation of polynomial and rational polynomial functions of a single variable. As will be seen, any polynomial is L-reducible, which is not true for rational functions. However, L-reducibility of a rational function can be assured by taking into account the appropriate conditions when the coefficients of the rational function are being defined. The results of these considerations are then used to provide a table of some elementary functions in order to illustrate the time and complexity requirements of the E-method in such applications. While a particular function or a limited set of functions can be evaluated more efficiently in some other techniques [VOL59, DEL70, WAL71] the E-method provides a fast and a reasonably efficient (in a sense that a required computing structure in a particular application is obtained by repetition of the same simple basic part) technique of evaluation for practically an unlimited number of functions. The compatibility of the basic arithmetics

with the proposed method is also considered with an emphasis on division. By looking at division as an L-reducible problem, an algorithm was derived which has a desirable way of generating the quotient digits deterministically and which can also use redundantly represented partial remainders. This algorithm appears functionally equivalent to the division algorithm described by Svoboda [SVO63]. After giving some examples of the application of the E-method in evaluating certain arithmetic expressions, such as multiple products or sums, inner products etc., this chapter is concluded with a brief consideration of the E-method as a linear solver.

We have made no attempts presently to consider the implications the E-method may have in fields like digital signal processing or linear control systems. It may be expected that the proposed computational technique can be efficiently utilized in many special purpose computing systems or devices.

In Chapter 3 it was indicated that the basic performance factors, i.e., the speed and the cost can be easily deduced from the computational graph for a particular application of the E-method. While considering problems of evaluating polynomials and rational functions, we will attempt to provide relative measures of performance with respect to a conventional sequential or S-method and a parallel or P-method of computation. We will use the speedup S, efficiency E and cost C factors as defined by Kuck [KUC74], but under different assumptions. Namely, as a matter of hardware implementation complexity standardization, we will assume that a processor used by S-, or P-method is equivalent in its capability and complexity to an elementary unit of the E-method. We will also consider all three methods in a domain of operations on a hardware level so that

the usual assumption that each arithmetic operation takes the same unit time does not hold here. The relative performance measures for the other application examples can be derived in a similar way and are omitted here.

4.2 Evaluation of Polynomials

Let

$$P_\mu(x) = \sum_{i=0}^{\mu} p_i \, x^i \tag{4.1}$$

be a μ-degree polynomial with real-valued coefficients $\{p_i\}$ and the argument $x \in [a,b]$. Define the coefficient and the argument norms as follows:

$$\|\underline{p}\| = \max_i |p_i| \tag{4.2}$$

$$\|\underline{x}\| = \max_{x \in [a,b]} |x| \tag{4.3}$$

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

If

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

and (4.4)

$$\|\underline{x}\| \leq \alpha$$

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

$$a_{ij} = \begin{cases} 1 & \text{for } i=j; \\ -x & \text{for } j=i+1,\ i \leq \mu \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

$$b_i = \begin{cases} p_{i-1} & \text{for } i=1,2,\ldots,\mu+1 \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

as illustrated in Figure 4.1. The system L is then solved using the Algorithm E so that after m+1 steps

$$\underline{y}^* = \sum_{j=1}^{m+1} \underline{d}^{(j)} r^{-j} \tag{4.7}$$

satisfies

$$|P_\mu(x) - y_1^*| < r^{-m} \tag{4.8}$$

and

$$\left| \sum_{k=i-1}^{\mu} p_k x^{k-i+1} - y_i^* \right| < r^{-m} \tag{4.9}$$

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

The computational graph G_P is shown in Figure 4.2. All the elementary units EU_i, $i = 1,2,\ldots,n$ are identical: each one requires an adder with one redundant $(w_i^{(j)})$ and one nonredundant $(x \cdot d_{i+1}^{(j)})$ operand, or, effectively, a conventional adder, where r=2.

In general the conditions on norms (4.4) may not be satisfied and an appropriate scaling of p_i's and x must be done prior to the evaluation. The required scaling in this case presents no problems.

Let $\underline{S} = (s_{ij})_{n \times n}$ be a scaling matrix, defined as follows:

$$s_{ij} = \begin{cases} r^{-(i-1)\sigma_A} & \text{for } i=j \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

$$\begin{bmatrix} 1 & -x & & & & & \\ & 1 & -x & & & & \\ & & 1 & -x & & & \\ & & & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \\ & & & & & \cdot & \cdot \\ & & & & & 1 & -x \\ & & & & & & 1 \end{bmatrix} x \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \cdot \\ \cdot \\ \cdot \\ \\ y_n \end{bmatrix} = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \cdot \\ \cdot \\ \cdot \\ \\ p_\mu \end{bmatrix}$$

Figure 4.1 Correspondence Rule C_P

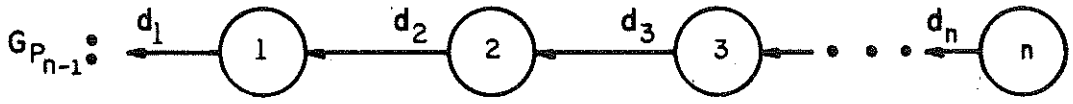


Figure 4.2 Computational Graph G_P

where σ_A is a positive integer or zero when no scaling is required. Then its inverse $\underline{S}^{-1}$ is also a diagonal matrix with $s_{ij} = r^{(i-1)\sigma_A}$ for i=j.

Consider

$$\underline{S}^{-1}\ \underline{A}(x)\ \underline{S}\ \underline{S}^{-1}\ \underline{y} = \underline{S}^{-1}\ \underline{b} \tag{4.11}$$

Let

$$\tilde{\underline{A}}(x) = \underline{S}^{-1}\ \underline{A}(x)\ \underline{S}$$

$$\tilde{\underline{y}} = \underline{S}^{-1}\ \underline{y} \tag{4.12}$$

and

$$\tilde{\underline{b}} = \underline{S}^{-1}\ \underline{b}$$

If

$$\|\underline{x}\| \cdot r^{-\sigma_A} \leq \alpha$$

i.e.,

$$\sigma_A = \begin{cases} \left\lceil \log_r \dfrac{\|\underline{x}\|}{\alpha} \right\rceil & \text{if } \|\underline{x}\| \not\leq \alpha \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

then

$$\|\tilde{\underline{A}}(x)\| \leq \alpha$$

as desired. However, since

$$\|\tilde{\underline{b}}\| \leq r^{\mu\sigma_A}\|\underline{b}\| , \tag{4.14}$$

additional scaling must be performed, as described in Section 2.9, if $\|\tilde{\underline{b}}\| \not\leq \zeta$. It can be seen that this right-hand side scaling requires

$$\sigma_b = \begin{cases} \left\lceil \log_r \dfrac{\|\tilde{\underline{b}}\|}{\rho} \right\rceil & \text{if } \|\tilde{\underline{b}}\| \not\leq \zeta \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

Note that the scaling of the matrix $\underline{A}$ does not introduce change in the prescribed number of steps since $\tilde{y}_1^* = y_1^*$, by definition of the scaling matrix $\underline{S}$. However, to satisfy ζ bound, σ_b extra digits may be necessary and therefore the time to evaluate a polynomial $P_\mu(x)$ is, in general,

$$T_E(P_\mu) = (m+1+\sigma_b)\ t_0 \tag{4.16}$$

It is interesting to observe that if no scaling is required, the time of evaluation is independent of the degree of a given polynomial.

After σ_A and σ_b are determined, the scaling is performed by shifting, i.e.,

$$\begin{aligned} a_{i,i+1} &= -\ x\cdot r^{-\sigma_A} && \text{for } i = 1,2,\ldots,\mu \\ b_i &= p_{i-1}\cdot r^{-(\sigma_b-(i-1)\sigma_A)} && \text{for } i = 1,2,\ldots,\mu+1 \end{aligned} \tag{4.17}$$

Example 4.1:

Consider the scaling requirements for the polynomial approximation $\log_2(x) \approx P_9(x)$, $x\in[1/2,1]$ with a precision of eight decimal digits. Using the coefficients $\{p_i\}$ from [HAR68, LOG2 2508, p. 110, p. 225], and assuming that $r = 2$, $m = 27$, $\zeta = 3/4$, $\alpha = 1/8$ we obtain:

$$\sigma_A = 3 \qquad \text{since} \qquad \|\underline{x}\| \le 1$$

$$\sigma_b = 29 \qquad \text{since} \qquad \|\tilde{\underline{b}}\| \le 3\cdot 2^{27}$$

Therefore, the required number of steps to evaluate given polynomial becomes

$$m' = m + 1 + \sigma_b = 57$$

□

Example 4.2:

Evaluation of a polynomial as an approximation to 2^x, $x\in[0,1]$, with a precision of 7 decimal digits for x = 0.5 is illustrated in Figure 4.3. The coefficients of $P_5(x)$ are from [HAR68, EXPB 1042, p. 102, p. 204]:

$$p_0 = 0.999999925$$

$$p_1 = 0.693153073$$

$$p_2 = 0.240153617$$

$$p_3 = 0.558263130*10^{-1}$$

$$p_4 = 0.898934003*10^{-2}$$

$$p_5 = 0.187757667*10^{-2}$$

The parameters are r = 2, n = 6, $\zeta = 3/4$, $\alpha = 1/8$, $\sigma_A = 3$, $\sigma_b = 7$, m = 24 + 1 + 7 = 32. For x = 0.5, $|\sqrt{2} - y_1^*| < 2^{-24}$.

□

We now consider the performance of the E-method in polynomial evaluation relative to the S-, and P-methods, as defined in Section 4.1. 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:

j	$w_1^{(j)}$	d_1	d_2	d_3	d_4	d_5	d_6	y_1^*
1	0.015624998	0	0	0	0	1	1	0.000000000
2	0.031249997	0	0	0	1	-1	0	0.000000000
3	0.062499995	0	0	1	0	1	0	0.000000000
4	0.124999990	0	1	0	0	-1	0	0.000000000
5	0.374999981	0	0	0	0	0	-1	0.000000000
6	0.749999962	1	-1	1	0	0	1	2.000000000
7	-0.625000075	-1	1	-1	-1	1	0	1.000000000
8	0.874999849	1	-1	1	0	-1	-1	1.500000000
9	-0.375000300	0	1	-1	0	1	0	1.500000000
10	-0.625000600	-1	-1	0	1	-1	0	1.375000000
11	0.624998799	1	0	1	0	1	0	1.437500000
12	-0.750002400	-1	0	0	0	-1	1	1.406250000
13	0.499995199	0	0	0	0	1	0	1.406250000
14	0.999990399	1	0	0	0	-1	-1	1.414062500
15	-0.000019200	0	1	1	0	0	0	1.414062500
16	0.124961599	0	-1	-1	0	1	1	1.414062500

Figure 4.3 Evaluation of $2^x \approx P_5(x)$

17	0.124923199	0	0	1	0	0	-1	1.414062500
18	0.249846399	0	1	-1	-1	-1	0	1.414062500
19	0.624692798	1	0	1	1	0	0	1.414306640
20	-0.750614402	-1	0	0	-1	1	0	1.414184570
21	0.498771194	0	0	-1	1	-1	0	1.414184570
22	0.997542389	1	-1	0	-1	1	1	1.414215087
23	-0.129915220	0	1	0	1	0	-1	1.414215087
24	-0.134830441	0	-1	0	0	-1	0	1.414215087
25	-0.394660883	0	0	0	-1	1	1	1.414215087
26	-0.789321767	-1	1	0	0	-1	0	1.414213180
27	0.546356464	1	-1	-1	0	1	-1	1.414214134
28	-1.032287071	-1	1	0	0	0	0	1.414213657
29	0.060425857	0	-1	1	0	-1	1	1.414213657
30	-0.004148284	0	1	-1	-1	1	-1	1.414213657
31	0.116703431	0	0	1	1	0	0	1.414213657
32	0.233406862	0	0	-1	0	-1	1	1.414213657

Figure 4.3 Evaluation of $2^x \approx P_5(x)$ (continued)

$$T_E(P_\mu) = (m+1)\ t_0,\ n_E = \mu + 1$$

$$T_S(P_\mu) = \mu\ m\ t_0,\ n_S = 1 \qquad (4.18)$$

$$T_P(P_\mu) \approx (\log_2 \mu + (O(\log_2 \mu)^{1/2})m)\ t_0,\ n_P = 2\mu$$

We have assumed that the P-method uses Maruyama-Munro-Paterson algorithm for parallel polynomial evaluation as given in [KUN74].

Following Kuck [KUC74], the E-method algorithm for polynomial evaluation has the speedup factor S_E:

$$S_E = \frac{T_S}{T_E} = \mu\left(\frac{m}{m+1}\right) \approx \mu\ , \qquad (4.19)$$

the efficiency factor E_E:

$$E_E = \frac{S_E}{n_E} = \left(\frac{\mu}{\mu+1}\right)\left(\frac{m}{m+1}\right) > 0.5 \text{ for } \mu > 1,\ m > 3 \qquad (4.20)$$

and the cost factor C_E:

$$C_E = n_E \cdot T_E = (\mu+1)(m+1)\ t_0 \qquad (4.21)$$

Similarly for the P-method algorithm:

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

$$E_P = \frac{S_P}{n_P} \approx \frac{1}{2\sqrt{M}} \cdot \frac{m}{\sqrt{M+m}} \qquad (4.22)$$

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

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 than the B-method if

$$\max\left(\frac{S_A}{C_A}, \frac{S_B}{C_B}\right) = \frac{S_A}{C_A} \tag{4.23}$$

It can be easily seen that

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

while (4.24)

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

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

$$\lim_{m\to\infty} \frac{S_E}{C_E} \cdot \frac{C_P}{S_P} = \frac{2\mu \log_2 \mu}{\mu+1} > 1 \quad \text{for} \quad \mu > 1 \tag{4.25}$$

This indicates that the E-method algorithm is better in performance noticeably than any P-method algorithm for the evaluation of polynomials under the previously stated conditions. Furthermore, the E-method algorithm has a behavior of performance measure qualitatively different from the considered P-method algorithm, as illustrated in Table 4.1 and Figure 4.4. In this example it is assumed that m = 56, r = 2 and, for presentation convenience, that $t_0 = 10^{-2}$ units.

The conclusion that the E-method offers better performance in terms of speed and cost than a P-method in polynomial evaluation could be made even stronger if the rather complex control and communication requirements of a P-method were compared with those of the E-method. To emphasize again, these conclusions are limited by the imposed assumptions, mentioned earlier. The speed of a P-method algorithm could certainly be increased, by using sophisticated multipliers beyond the speed of the

Table 4.1 Performance Factors

μ	E_E	E_P	T_E/t_0	T_P/t_0	S_E	S_P	$\frac{S_E}{C_E}$	$\frac{S_P}{C_P}$
2	0.65	0.49	57	57	1.96	1.96	1.14	0.86
4	0.78	0.35		81	3.92	2.77	1.38	0.62
8	0.87	0.28		102	7.84	4.38	1.53	0.49
16	0.92	0.24		118	15.68	7.58	1.61	0.42
32	0.95	0.21		133	31.36	13.46	1.67	0.38

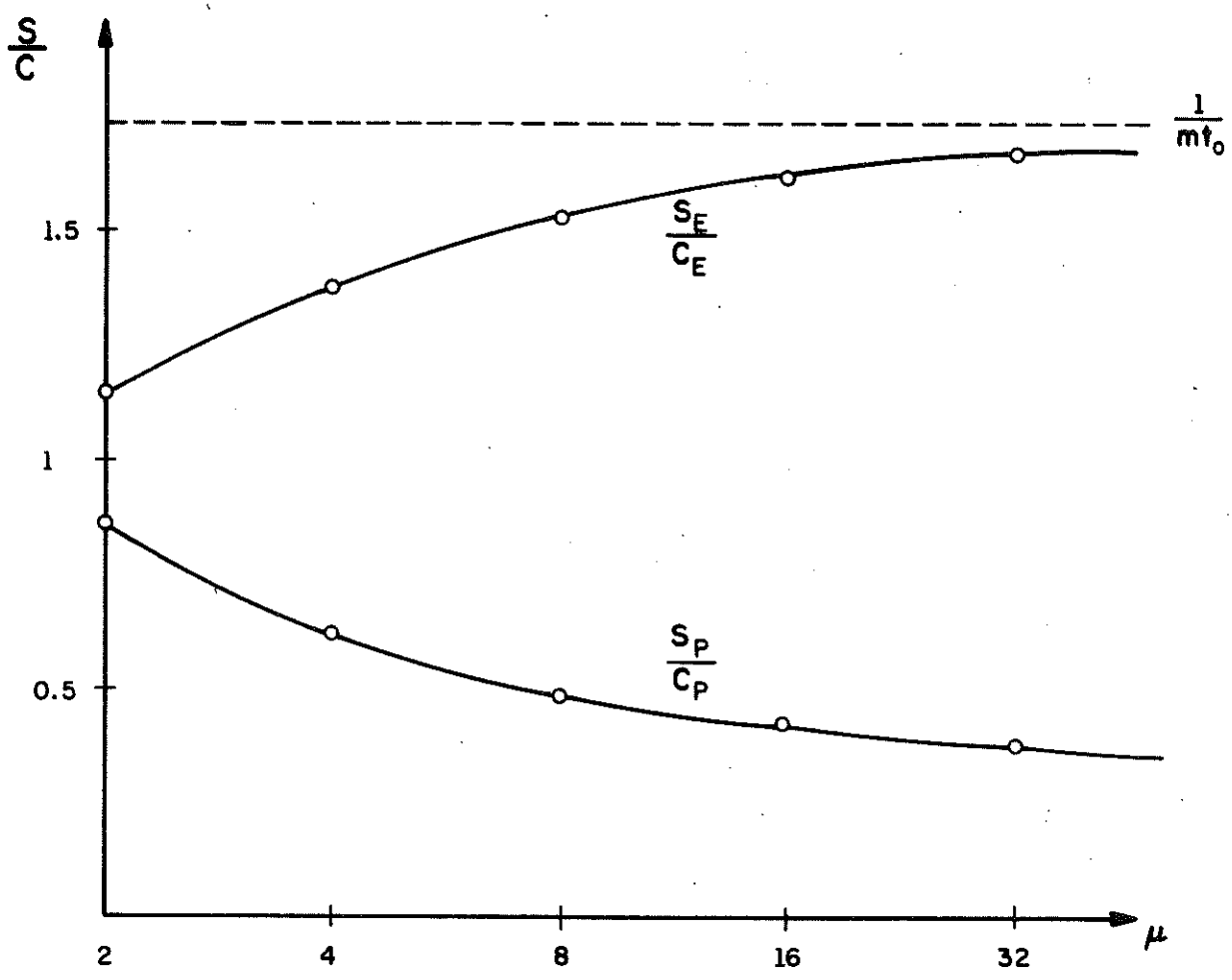


Figure 4.4 Evaluation of Polynomials: Performance Measures

E-method, but at the expense of highly increased cost. The similar conclusions about the relative performance of the E-method with respect to the P-method of polynomial evaluation with combinational arithmetic nets, as proposed by Tung and Avizienis [TUN70], were found to apply.

4.3 Evaluation of Rational Functions

The correspondence rule C_R which defines a linear system $L: \underline{A}(x)\,\underline{y} = \underline{b}$ for a given L-reducible rational function $R_{\mu,\nu}(x)$ has already been given in Section 2.2. Algorithm E can be carried out on a configuration represented by the graph G_R with $n = \max(\mu,\nu) + 1$, as illustrated in Figure 4.5. The basic recursion (2.50) becomes

$$w_i^{(j)} = d_i^{(j)} + z_i^{(j)} = r(z_i^{(j-1)} + g_{i1} d_1^{(j-1)} + g_{i,i+1} d_{i+1}^{(j-1)}) , \tag{4.26}$$

with $g_{i1} = -q_{i-1}$ and $g_{i,i+1} = x$,

$i = 2,\ldots,n-1,$

trivially modified for i=1 ($g_{i1}=0$) and for i=n ($g_{n,n+1}=0$, $d_{n+1}=0$). 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 easily achieved with a two-level conventional adder.

The conditions under which a rational function is L-reducible will be considered in some detail. Let

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

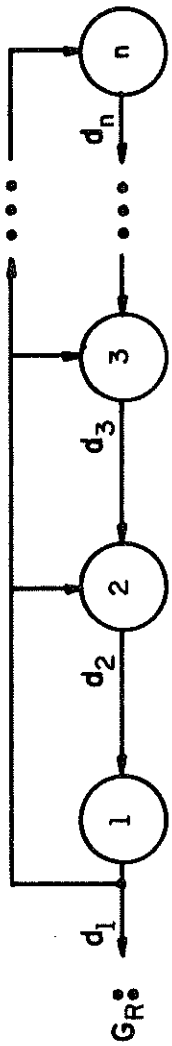


Figure 4.5 Computational Graph G_R

be a rational function with real-valued coefficients $\{p_i\}$ and $\{q_i\}$ and the argument $x\epsilon[a,b]$, assuming without loss of generality that $q_0 = 1$. Define the following norms:

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

$$\|\underline{q}\| = \max_{i\neq 0} |q_i| \tag{4.27}$$

$$\|\underline{x}\| = \max_{x\epsilon[a,b]} |x|$$

Then by the conditions of Theorem 2.3, the following must hold:

$$\begin{aligned} &\text{(i)} \quad \|\underline{q}+\underline{x}\| \leq \alpha \\ &\text{(ii)} \quad \|\underline{p}\| \leq \zeta \end{aligned} \tag{4.28}$$

The condition (i) appears as

$$\|\underline{x}\| \leq \alpha \tag{4.29}$$

for the first row of $\underline{A}(x)$ and as

$$|q_\nu| \leq \alpha \tag{4.30}$$

for the $(\nu+1)$-st row if $\mu \leq \nu$.

The scaling implications of the condition (ii) have already been discussed in Section 2.9 and will not be reconsidered here. The condition (i), for practical purposes, can be expressed as

$$\begin{aligned} &\text{(i')} \quad \|\underline{q}\| \leq \alpha(1-c), \ 0 < c < 1 \\ &\text{(i'')} \quad \|\underline{x}\| \leq c\,\alpha \end{aligned} \tag{4.31}$$

Supposing that (i'') is satisfied and utilizing any $R_{\mu,\nu}(x)$ with $q_0 = 1$, such that

$$|q_i| \leq \frac{1-c}{c}\alpha , \quad i = 1,2,\ldots,\nu \tag{4.32}$$

is L-reducible and can be evaluated in O(m) steps according to Algorithm E.

While a given rational function may or may not satisfy (i'') and (4.32), these conditions can certainly be taken as additional constraints when the coefficients $\{q_i\}$ of a rational function are being determined. This is, in particular, applicable when the coefficients of the rational function are obtained using linear programming techniques. It may be noted that certain special cases may arise such that one can always take advantage of their presence. For example, whenever $q_i = 0$, all q_j, $j > 1$, can be reduced by factor c. The multiple reductions are also possible for each q_j if $q_i = 0$ for more than one $i < j$. Similarly, if $|q_i| < \alpha$, the range of argument x can be increased. If $|x| > \alpha$, one can consider evaluating $R'_{\mu,\nu}(1/x)$ where, hopefully, $p'_i = p_{\nu-i}$, $q'_i = q_{\nu-i}$ for $\nu > \mu$, satisfy the required conditions, etc.

Example 2.3, given at the end of Chapter 2, with Figure 2.4, illustrates the evaluation of a rational function.

Let us consider the performance of the E-method in evaluating rational functions under the same assumptions as in the previous section. The sequential S-method has

$$T_S(R_{\mu,\nu}) \approx (\mu+\nu+1)\,\frac{m}{2}\,t_0 ,$$

assuming that two digits of a multiplier, or quotient at the end of the evaluation, can be retired at each step. Since

$$T_E(R_{\mu,\nu}) = (m+1)\,t_0$$

the speed-up of the E-method is

$$S_E = \frac{T_S}{T_E} \approx \frac{\mu+\nu+1}{2}$$

with an efficiency

$$E_E = \frac{S_E}{n} \approx \frac{\mu+\nu+1}{2(\max(\mu,\nu)+1)}$$

For $\mu=\nu$, $E_E \geq 0.75$. We know of no P-method algorithm for which

$$T_P(R_{\mu,\nu}) \leq T_P(P_{\max(\mu,\nu)})$$

and

$$E_P(R_{\mu,\nu}) \leq E_P(P_{\max(\mu,\nu)})$$

so, on the basis of the conclusions made in the previous section, we may again conclude that the E-method is faster and more efficient than a P-method, under the stated assumptions.

4.4 Evaluation of Elementary Functions

With a capability to efficiently evaluate polynomial and rational functions, the E-method can certainly be used for the evaluation of arbitrary functions for which suitable polynomial or rational approximations exist. Hence, the evaluation of a given function would be characterized mainly by the corresponding set of coefficients, which can be kept in a local storage area of the computing configuration, or on any other convenient level in the available storage hierarchy. Furthermore, the evaluation could be performed easily in a variable precision. In general, given a sufficient number of the two-input elementary units, an evaluation

would take $T_E(f) = O(10m)\ t_g$, where t_g is a gate delay and the radix is binary. Since the relationship between the speed and cost of the E-method is linear, a wide range of evaluation requirements can be easily accommodated.

The E-method, as described before, imposes certain conditions on the size of the operands which, in general, prevent the use of an arbitrary rational approximation. However, one can define desired rational approximations under those conditions to be optimal with respect to the E-method. We have not attempted here to derive such optimal approximations. Rather, we have decided to provide an overview of the E-method parameters for several previously specified approximations of certain functions [HAR68] so that the performance could be estimated. Table 4.2 displays these examples. The precision of the approximations, given as m radix 2 digits, is based on the relative error criterion except for the last four examples. The effects of scaling appear in the actual number of steps m'. The variations in overlap Δ, defined in Section 2.4, are sometimes necessary to accommodate given coefficients or the argument range but have no significant effect on the step time t_0. The required number of elementary units is given in column n. A computational configuration consisting of elementary units does not preclude the existence of a separate fast multiplication unit so that in place of $R_{\mu,\nu}(x)$, a more efficient form $R_{\frac{\mu}{2},\frac{\nu}{2}}(x^2)$ can be used. Note that for the form $R(x) = x\,\frac{P(x^2)}{Q(x^2)}$, which sometimes appears, the correspondence rule C_R can be easily modified as follows:

Table 4.2 Examples of Function Evaluation Requirements

Function	Approximation	Index	Argument Range	n	m	m'	Δ
2^x	$R_{5,4}(x)$	EXPB 1103	$[0, \frac{15}{128}]$	6	82	83	$\frac{1}{16}$
2^x	$P_8(x)$	EXPB 1045	$[0,1]$	9	40	47	$\frac{1}{2}$
10^x	$P_7(x)$	EXPD 1404	$[0, \frac{1}{10}]$	8	47	50	$\frac{1}{2}$
e^x	$R_{3,3}(x)$	EXPEC 1800	$[0, \frac{15}{256}]$	4	41	43	$\frac{1}{16}$
e^x	$R_{5,5}(x)$	EXPEC 1802	$[0, \frac{15}{256}]$	6	75	77	$\frac{1}{16}$
$\sinh(x)$	$R_{3,4}(x)$	SINH 2002	$[0, \frac{1}{8}]$	5	45	46	$\frac{1}{2}$
$\log_{10}(\frac{1+x}{1-x})$	$xP_4(x^2)$	LOG10 2303	$[2\sqrt{2}-3, 3-2\sqrt{2}]$	6	41	44	$\frac{1}{2}$
$\log_e(\frac{1+x}{1-x})$	$P_4(x^2)$	LOGE 2663	$[2\sqrt{2}-3, 3-2\sqrt{2}]$	5	40	43	$\frac{1}{2}$
$\sin(\frac{\pi}{6}x)$	$R_{5,4}(x)$	SIN 2941	$[0, \frac{60}{64}]$	6	43	52	$\frac{1}{16}$
$\sin(\frac{\pi}{4}x)$	$xP_4(x^2)$	SIN 3041	$[0,1]$	6	37	43	$\frac{1}{2}$

Table 4.2 Examples of Function Evaluation Requirements (continued)

Function	Approximation	Index	Argument Range	n	m	m'	Δ
$\cos(\frac{\pi}{4}x)$	$R_{3,2}(x^2)$	COS 3842	$[0, \frac{15}{32}]$	7	44	47	$\frac{1}{16}$
$\tan(\frac{\pi}{16}x)$	$xR_{1,2}(x^2)$	TAN 4122	$[0,1]$	4	42	45	$\frac{1}{8}$
$\arcsin(x)$	$xR_{1,2}(x^2)$	ARCSN 4602	$[0, \sin\frac{\pi}{24}]$	4	39	41	$\frac{1}{4}$
$\arctan(x)$	$R_{3,4}(x)$	ARCTN 5011	$[0, \tan\frac{\pi}{32}]$	5	41	45	$\frac{1}{2}$
$\Gamma(x)$	$P_9(x)$	GAMMA 5208	$[0,1]$	10	32	48	$\frac{1}{2}$
$\Gamma(x)$	$R_{6,3}(x)$	GAMMA 5231	$[0,\frac{1}{8}]$	7	38	40	$\frac{1}{16}$
$\ell n\,\Gamma(\frac{1}{x})$	$P_5(x)$	LGAM 5401	$[10^{-3},2^{-3}]$	6	37	38	$\frac{1}{2}$
$\ell n\,\Gamma(\frac{1}{x})$	$R_{3,2}(x)$	LGAM 5460	$[10^{-3},\frac{1}{12}]$	4	42	43	$\frac{1}{4}$

$$\left[\begin{array}{c|ccc} 1 & -x & 0 \ \cdots & 0 \\ \hline 0 & & & \\ \vdots & & \underline{A}(x^2) & \\ 0 & & & \end{array}\right] \underline{y} = \left[\begin{array}{c} 0 \\ \hline \\ \underline{b} \\ \\ \end{array}\right]$$

while $y_1 = R(x)$, as before.

It may be of interest to mention here that Algorithm E can be easily adapted for the evaluation of a given function on a set of argument values, which is of practical importance in many numerical problems. Let us consider the case where a function $f(x)$ is to be evaluated on a set of equidistant argument values $X_k = \{x_j | x_j = x_{j-1} + \Delta x,\ \Delta x < r^{-k}\}$, using n elementary units. Assume further that the set X_k is such that the first $\ell < k$ digits of the argument remain invariant. Since Algorithm E generates the result in a left-to-right, digit-by-digit fashion, it is sufficient to provide 2n additional registers to preserve $\underline{w}^{(\ell)}$ for continuation as well as a facility for updating the argument by Δx. These modifications are compatible with those required for a variable precision mode of operation, as discussed in Section 3.4. For an m digit precision, the time to evaluate $f(x)$, $x \in X_k$, after the first evaluation, would be

$$T_E(f) \approx (m-\ell)\ t_0$$

so that the total time for the evaluation over X_k is reduced by $m/(m-\ell)$ times.

4.5 On Performing the Basic Arithmetics

The generic algorithm, Section 2.6, by definition of the recursive step can clearly be used for additions/subtractions and multiplications.

The results obtained will be in a redundant form so that the conversion to a conventional representation, when necessary, would require a carry propagation facility. By considering division as an L-reducible problem, the quotient and the remainder can be generated by using the general algorithm of the E-method in a deterministic fashion, with basic recursive step executable in time independent of the length of the operands.

Consider the division problem

$$B = AQ_m + R_m$$

where

$$\begin{aligned} &A \text{ is the divisor, } 1 - \alpha \leq |A| \leq 1 + \alpha; \\ &B \text{ is the dividend, } |B| < |A| \leq \zeta; \\ &Q_m \text{ is m digit quotient and} \\ &R_m \text{ is the corresponding remainder.} \end{aligned} \tag{4.33}$$

Consider a degenerate system L: $\underline{A}\,\underline{y} = \underline{b}$ of order 1, defined by the division correspondence rule C_D:

$$a_{11} = a = |A|$$

$$b_1 = b = (\text{sign}A)B$$

Let $g = 1 - a$ so that

$$y = b + gy \tag{4.34}$$

where

$$y = \frac{B}{A}$$

can be generated using Algorithm E. It is easily shown that $y^* = \sum_{j=1}^{m} d^{(j)} r^{-j}$ and $w^{(m+1)}$ satisfy

$$|Q_m - y^*| < r^{-m}$$

and (4.35)

$$R_m = w^{(m+1)} r^{-m-1} .$$

The conditions (4.33) require, in general, an initial transformation of the given divisor A' and the dividend B'. For example, assuming $r = 2$, $\Delta = 0$ and $1/2 \leq |A'| < 1$ as usual, the simplest transformation of the divisor could possibly be

$$|A| = \begin{cases} 2\,|A'| \\ \frac{3}{2}\,|A'| \\ |A'| \end{cases} \quad \text{for } |A'| \in \begin{cases} [\frac{1}{2}, \frac{5}{8}) \\ [\frac{5}{8}, \frac{3}{4}) \\ [\frac{3}{4}, 1) \end{cases} \tag{4.36}$$

This transformation would become slightly more complicated when the selection overlap $\Delta \neq 0$. For $\Delta = \frac{1}{16}$, $\alpha = \frac{7}{32}$ the transformation could be carried according to:

$$|A| = \begin{cases} 2|A'| \\ \frac{3}{2}\,|A'| \\ |A'| \end{cases} \quad \text{for } |A'| \in \begin{cases} [\frac{1}{2}, \frac{19}{32}) \\ [\frac{19}{32}, \frac{25}{32}) \\ [\frac{25}{32}, 1) \end{cases} \tag{4.37}$$

which is not a serious complication in exchange for a carry-propagation free division algorithm, indeed.

As shown in Figure 4.6, both division and multiplication require one single input elementary unit. In the case of division there is a cyclic graph representation.

Example 4.3:

Compute the quotient Q_5 and the remainder R_5 using Algorithm E, with $r = 2$, $m = 5$, the dividend $B = 3/4$ and the divisor $A = 5/4$ $(g=-1/4)$

j	$w^{(j)}$	$d^{(j)}$	$z^{(j)}$
1	1.1	1	0.1
2	0.1	1	-0.1
3	-1.1	$\bar{1}$	-0.1
4	-0.1	$\bar{1}$	0.1
5	1.1	1	0.1
6	0.1		

$$Q_5 = \sum_{j=1}^{5} d^{(j)} 2^{-j} = 0.11\bar{1}\bar{1}1 = 0.10011 \quad (B/A = 3/5 = 0.\overline{\dot{1}00\dot{1}})$$

$R_5 = w^{(6)} 2^{-6} = 0.0000001$ so that

$B = Q_5 A + R_5$

□

4.6 Evaluation of Certain Arithmetic Expressions

A set of elementary units can be conveniently applied in evaluating certain expressions in O(m) recursive steps according to the algorithm of the E-method. We consider first two basic nontrivial arithmetic expressions

$$P_c = \prod_{i=1}^{p} c_i \tag{4.38}$$

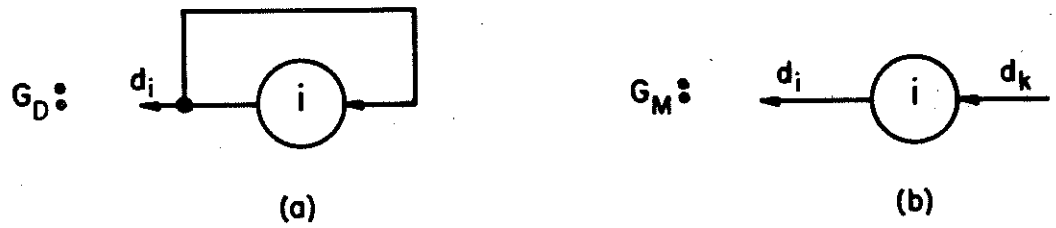


Figure 4.6 Computational Graphs for Division and Multiplication

and

$$S_c = \sum_{i=1}^{s} c_i \tag{4.39}$$

Being a degenerate case of a polynomial, the multiple product P_c can be easily evaluated in (m+1) steps with p elementary units according to the following correspondence rule:

$$a_{ij} = \begin{cases} 1 & \text{for } i=j; \\ -c_i & \text{for } j=i+1,\ i=1,\ldots,p-1 \\ 0 & \text{otherwise} \end{cases}$$

$$b_i = \begin{cases} c_p & \text{for } i=p \\ 0 & \text{otherwise} \end{cases} \tag{4.40}$$

Clearly,

$$\left| \prod_{k=i}^{p} c_k - y_i^* \right| < r^{-m}, \qquad i=1,2,\ldots,p$$

so that for $c_i = x$, $\forall$ i, the E-method generates the positive integral powers of x : $x^2, x^3, x^4, \ldots, x^p$ in (m+1) steps on p elementary units. As before, it is assumed that factors c_i satisfy the range conditions.

The multiple sum S_c can be evaluated by the E-method as illustrated by the following example.

Example 4.4:

To evaluate $S_c = \sum_{i=1}^{5} c_i$, consider the following L system:

$$\begin{bmatrix} 1 & -\alpha/2 & -\alpha/2 & & & \\ & 1 & & & & \\ & & 1 & -\alpha/2 & -\alpha/2 & \\ & & & 1 & & \\ & & & & 1 & \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}$$

For $\alpha = 1/8$, $y_1 = b_1 + 2^{-4}(b_2+b_3) + 2^{-8}(b_4+b_5)$, which indicates the necessary relations between b_i's and given c_i's. If necessary, scaling on $\underline{b}$ vector can be performed as before. □

Although fully consistent with the E-method, this approach is not a very efficient way of forming sums, since a k-input elementary unit alone has, by definition, a capability to add (k+1) operands, and it would be desirable to exploit this more efficiently. Let us briefly consider the alternate ways of evaluating multiple operand sums. For the sake of simplicity, it will be assumed that the available elementary units have k or more inputs, so that (k+1) operands can be reduced to a redundantly represented sum in one step, i.e., in time t_0. Let there be at least $n = (k^{\ell}-1)/(k-1)$ available elementary units. Then the elementary units can be arranged as an ℓ-level, radix-k tree, illustrated as the graph G_S in Figure 4.7. If the links between the nodes of G_S were capable of carrying in parallel the full precision results, then $s = (k+1)n$ operands could be added together in ℓ steps, assuming that no additional operands are allowed to enter computing scheme once the evaluation has started.

However, if only single digit interconnections between elementary units are available, as we assume to be the case for the E-method, then the same number of s operands would require $(\ell+m+1)$ steps. With the displacements of operands, relative to a level of the tree (Figure 4.7), it can be assured that the most significant digit of the sum appears on the top level after ℓ steps, followed by one more digit for each additional step.

The evaluation of the inner products can be made compatible to the E-method as follows. Let

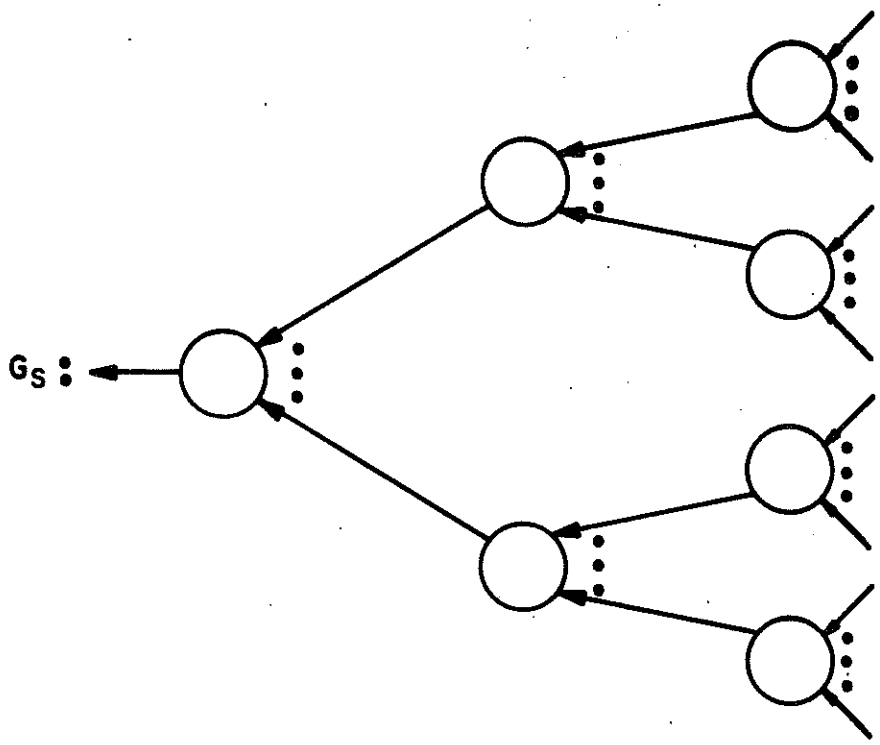


Figure 4.7 Computational Graph for Multi-Operand Summation

$$\underline{u} = (u_1, u_2, \ldots, u_p) \tag{4.41}$$

and

$$\underline{v} = (v_1, v_2, \ldots, v_p) \tag{4.42}$$

be two real-valued vectors with

$$W = \langle \underline{u}, \underline{v} \rangle = \sum_{i=1}^{p} u_i \, v_i \tag{4.43}$$

as their inner product. Define a (p+1) order system L as follows:

$$a_{ij} = \begin{cases} 1 & \text{for } i=j \\ -u_{j-1} & \text{for } i=1 \text{ and } j=2,\ldots,p+1 \\ 0 & \text{otherwise} \end{cases}$$

$$b_i = \begin{cases} 0 & \text{for } i=1 \\ v_i & \text{for } i=2,\ldots,p+1 \end{cases} \tag{4.44}$$

Then

$$y_1 = W$$

since

$$\underline{A}^{-1} = \left[\begin{array}{c|c} 1 & -\underline{u} \\ \hline \underline{0} & \underline{I} \end{array}\right] \tag{4.45}$$

If one p-input elementary unit is available, the inner product W can be evaluated in (m+1) steps where the remaining p elementary units can be shift registers. However, when only $k < p$ input elementary units are available, modifications to the above given correspondence rule are required. One obvious way would be to form independently $\lceil p/k \rceil$ partially

accumulated inner products W_j and then perform the final summation using one of the previously described approaches, as indicated in Figure 4.8. The other possible approach, more in the spirit of the E-method, is analogous to Example 4.4, and it is illustrated by the next example:

Example 4.5:

Evaluate $W = \sum_{i=1}^{5} u_i v_i$ assuming that available elementary units have no more than three inputs. A corresponding system L may appear as follows:

$$\begin{bmatrix} 1 & -u_1 & -u_2 & -\alpha/3 & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & -u_3 & -u_4 & -u_5 & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 0 \\ v_1 \\ v_2 \\ 0 \\ \frac{3}{\alpha} v_3 \\ \frac{3}{\alpha} v_4 \\ \frac{3}{\alpha} v_5 \end{bmatrix}$$

Then $y_1 = W$. Only two 3-input elementary units are required; the remaining five need not be more complex than shift registers. □

The previous example can be readily generalized for a p-dimensional inner product evaluation on k-input elementary units. The scaling may, however, introduce a tolerable number of extra steps. For example, let

$$|u_i| \leq \frac{\alpha}{k}, \quad |v_i| \leq \zeta$$

For $p > k$ it is easily seen that

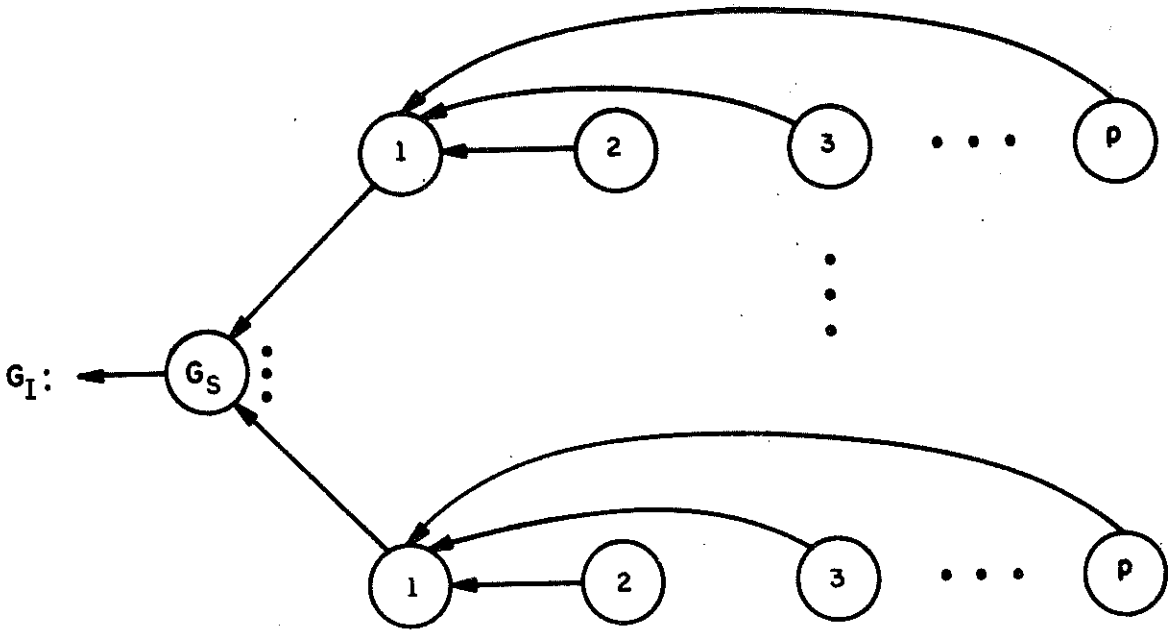


Figure 4.8 Computational Graph for Inner Product Evaluation

$$b_{p+1} = \left(\frac{k}{\alpha}\right)^{\lfloor p/k \rfloor} \cdot v_p \qquad (4.46)$$

so that an extension of

$$\sigma = \lceil \lfloor p/k \rfloor \log_r \frac{k}{\alpha} \rceil$$

may be required. For $\alpha = 1/8$, $k = 8$, $r = 2$, $p = 64$, $\sigma = 48$ while for $k = 4$, $\sigma = 80$, still of the order of commonly used precision for u_i's and v_i's.

We conclude this section with a remark that, in general, any arithmetic expression which can be put into correspondence with a system L, can be evaluated in $O(m)$ steps. For example, to evaluate

$$f = a(cd + be)$$

the following system L is solved

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

where $y_1 = f$. Or, to compute

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

one may solve

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

where $y_1 = h$.

4.7 On Solving the Systems of Linear Equations

Some general implications of the E-method on the problem of solving the systems of linear equations are briefly considered here. By definition, the proposed Algorithm E is a linear solver which can be applied for solving an n-th order nonhomogeneous system of linear equations

$$\underline{A}\,\underline{y} = \underline{b}$$

with the nonsingular coefficient matrix $\underline{A}$ in $O(m)$ additive steps if

$$\|\underline{A} - \underline{I}\| \leq \alpha$$
$$\|\underline{b}\| \leq \zeta \qquad (4.47)$$

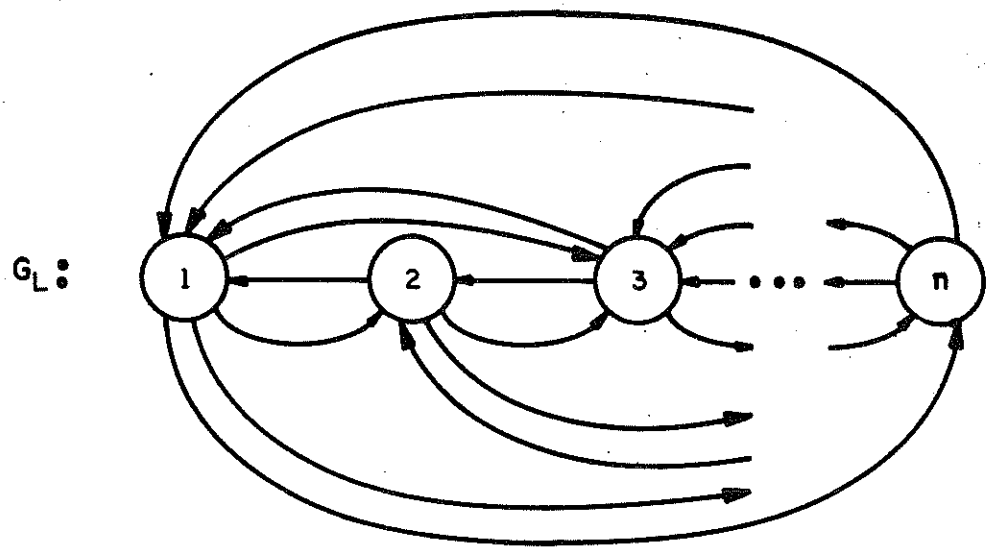
and each of n elementary units, implementing Algorithm E, has at least (n-1) inputs, in general (Figure 4.9 (a)).

When the conditions (4.47) are not satisfied, a scaling, as discussed in Section 2.9, must be considered. Whether scaling can be achieved in a practical way, depends also on the type of the coefficient matrix A. It can be easily seen, for example, that for an upper (lower) unit triangular matrix $\underline{A}$, scaling procedure, analogous to that defined for polynomials, can be applied in a practical way. Consider a unit upper triangular matrix $\underline{A} = (a_{ij})$ with $a_{ii} = 1$ and $a_{ij} = 0$ for $i < j$. Define the scaling matrix $\underline{S}$ as a diagonal matrix

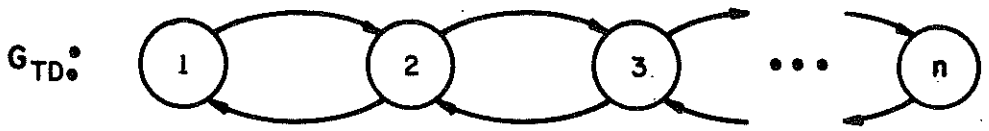
$$\underline{S} = \mathrm{diag}(1, r^{\sigma}, r^{2\sigma}, \ldots, r^{(n-1)\sigma})$$

where r is the radix and σ a positive integer. Then

$$\tilde{\underline{A}} = \underline{S}\,\underline{A}\,\underline{S}^{-1}$$



(a)



(b)

Figure 4.9 Computational Graphs for Dense and Tridiagonal Systems of Linear Equations

can be made to satisfy the required conditions by choosing an appropriate value for σ. Consequently, the number of steps, required to generate the first m correct digits of the solution $\underline{y}$ is increased by no more than $(n-1)\sigma$ steps.

For practical reasons, the number of inputs to the elementary unit may be expected to be small. The E-method remains applicable with a linear degradation of the performance as considered in Section 3.4. Under input restrictions, of particular interest are certainly sparse systems of linear equations. For example, a configuration of two-input elementary units, represented by the graph G_{TD} in Figure 4.9 (b) can be used to efficiently solve tridiagonal systems for which $|a_{ii}| \leq 4(|a_{i,i-1}| + |a_{i,i+1}|)$.

The E-method has an interesting property with implications on the number of operations, required to solve a linear system by an iterative scheme. Consider an n-th order linear system

$$(\underline{I} - \underline{G})\ \underline{y} = \underline{b} \tag{4.48}$$

solvable by the E-method. The system (4.48) could certainly be solved by a classical Jacobi algorithm, with the basic computational step defined as

$$\underline{y}^{(j)} = \underline{b} + \underline{G}\ \underline{y}^{(j-1)}, \qquad j=1,2,\ldots \tag{4.49}$$

Recall that the basic computational step of the E-method is

$$\underline{d}^{(j)} + \underline{z}^{(j)} = r(\underline{z}^{(j-1)} + \underline{G}\ \underline{d}^{(j-1)}), \qquad j=1,2,\ldots \tag{4.50}$$

where $\underline{d}$ is a single digit vector. Then, the recursion (4.49) requires (n-1) full precision multiplications and (n-1) additions per row per step, while for the same the recursion (4.50) performs (n-1) single digit multiplications and (n-1) additions, which for radix 2 becomes (n-1) additions.

The number of operations in (4.49) can be considered redundant in the sense that the parts of the generated solution are repeatedly utilized although once correctly determined and used the digits of the solution $\underline{y}$ do not contribute any critical information in the subsequent steps. In other words, one could reduce the overall complexity of an algorithm by conveniently removing the correctly computed result digits from further involvment in the algorithm. This can be easily achieved by a left-to-right processing and, interestingly enough, by introducing the redundancy in a number system representation, as is being done in the E-method. Since the correctly generated digits are used only once, i.e., digit vector $\underline{d}^{(j)}$ appears in the recursion only at the step (j+1), we may think of Algorithm E as a minimally redundant algorithm for solving a system of linear equations with respect to the required number of operations. Thus, the redundancy removing principles of the E-method may have an application even in programming iterative linear solvers on machines with a slow multiplication algorithm.

However, as a consequence of the minimized redundancy on the algorithmic level, the convergence of the algorithm becomes fixed, e.g., linear for Algorithm E. This must be considered when evaluating the merits of reducing the redundancy in number of operations of an iterative scheme.

5. CONCLUSIONS

5.1 Summary of the Results

In this dissertation a recently discovered general evaluation method, amenable to an efficient hardware-level implementation, is presented. The proposed evaluation method, referred to as the E-method, is characterized by several important performance features and appears applicable in many common computational problems.

Briefly, the E-method consists of i) a correspondence rule C_f which reduces a given computational problem $f \in F_L$ into an n-th order linear system L, F_L being the class of problems which are reducible to L, and ii) an algorithm with convenient implementation characteristics, which generates the solution to the system L and, consequently, to the original problem f, in O(m) recursive steps, where m is the desired number of digits of the result precision. The recursive steps are invariant and each of them is executable in time independent of the operand length.

The class F_L is illustrated in a sufficient number of problems to justify the claim of generality for the E-method. It includes, as the most practical examples, the evaluation of polynomials and rational functions so that all functions, for which corresponding approximations satisfying the conditions of the E-method exist, can be efficiently evaluated. The other problems, for example solving certain tridiagonal (sparse, in general) systems of linear equations, or the evaluation of certain types of arithmetic expressions or basic arithmetics, can be seen to belong to the class F_L.

The E-method requires, for the fastest evaluation, a configuration of n identical elementary units, but allows, in a straightforward manner, exploitation of its flexibility in tradeoffs between the speed and cost. The elementary unit, basically, is an s-operand adder with a corresponding number of registers. In many applications, s remains very small. The interconnections among the elementary units require only single digit links which can be conveniently specified and controlled by a connection matrix corresponding to the particualar coefficient matrix of the system L. The control part of the proposed algorithm is simple and deterministic in the sense that there are no convergence tests to be performed even though the method is iterative. The result is always generated in a digit-by-digit fashion, the most significant digits first, by applying a simple invariant selection procedure. Thus there exists a possibility for an overlapped mode of operation in a sequence of dependent computations as well as for a variable precision. A variable precision and a multi-point evaluation facility can be incorporated in a straightforward manner. The E-method has favorable error properties: it is never ill-conditioned and no round-off errors are generated, by definition. In its present form, the E-method is restricted to fixed-point or block floating-point number representation formats.

In applications, like polynomial evaluation, the E-method generates the result in one carry propagation free multiplication time, unless excessive scaling is involved, requiring an application dependent number of elementary units. For elementary function evaluations, 4-7 identical 2-input elementary units would suffice. The elementary unit has a simple and easily implementable structure.

5.2 Comments

While the proposed method provides efficient, technology-compatible solutions to many computational problems, like the evaluation of elementary functions, it also brings together the following issues which, we believe, are of fundamental importance in the design of algorithms:

i) the choice of algorithmic representation compatible with the implementation environment;

ii) the problem of redundancy on the algorithmic level;

iii) the problem of redundancy in a number representation system.

The first issue is concerned with the problem of 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 gives rise to a single algorithm to be implemented. The algorithm can, hopefully, satisfy the speed and cost objectives, among the other properties. And this, in general, would imply a small number of different primitive operators, simple enough to be efficiently implementable. In the E-method, addition appears as the only required primitive operator. The corresponding algorithmic 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 minimizes the necessary delay between

dependent computations and can achieve a parallelism up to one digit delay, having important properties like a simple, deterministic control, etc. Furthermore, this approach is 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 also affects the choice of the primitive operators. It is of interest to remark that in many parallel algorithms [KUC74], it is, on the contrary, necessary to introduce redundant operations in order to achieve parallelism.

The problem of redundancy in number representation systems has long been recognized as a central issue in achieving efficiently fast algorithms [ROB58, AVI61] and it will suffice here just to note that the E-method is another example where the redundancy in number representation has an essential role.

The theoretical basis of the E-method is simple yet, we believe, extendable to other interesting applications. Certainly, Algorithm E itself can be considered as a primitive operator which can be utilized in fast parallel schemes, not necessarily of the type defined by the E-method.

Among the problems of an immediate interest would be the development of polynomial and rational approximations to various functions, optimal with respect to the conditions of the E-method. It may be of some importance to note that the E-method in these applications utilizes the generated solution in a degenerate sense since only one, usually the first, component of the solution is of practical interest. Thus, it may be convenient to start with an approximation to the given function in the form

$$f(x) = \varphi(y_1(x),\ y_2(x),\ \ldots,\ y_k(x)),\ k \leq n$$

such that φ is a computationally simple function. This approach could possibly provide for more compatible properties of the coefficients which are affecting the scaling at an acceptable cost, all solution components y_j, $j = 1,\ldots n$ being generated at the same time. The problem of automatic scaling, i.e., an extension of the E-method to a floating-point number representation domain, a more systematic account of possible applications, a detailed design and implementation study of an elementary unit are some other subjects of practical interest. The E-method can be incorporated in a computing system in two obvious 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. Finally, it would be of interest to consider the changes in an instruction set which could make the E-method efficient for use on a software level.

LIST OF REFERENCES

[AVI61] Avizienis, A., "Signed-Digit Number Representations for Fast Parallel Arithmetic," IRE Trans. Electron. Comput. Vol. EC-10, pp. 389-400, September, 1961.

[CAM69a] Campeau, T. O., "The Block-Oriented Computer," IEEE Trans. Comput., Vol. C-18, pp. 706-718, August, 1969.

[CAM69b] Campeau, T. O., "Cellular Redundancy Brings New Life to an Old Algorithm," Electronics, pp. 98-104, July, 1969.

[CAM70] Campeau, T. O., "Communication and Sequential Problems in the Parallel Processor," in Parallel Processor Systems, Technologies and Applications, Edited by L. C. Hobbs, New York, Spartan Books, 1970.

[CHE72] Chen, T. C., "Automatic Computation of Exponentials, Logarithms, Ratios and Square Roots," IBM T. Res. Develop., Vol. 16, pp. 380-8, July, 1972.

[CLE62] Clenshaw, C. W., "Chebyshev Series for Mathematical Functions," National Physical Laboratory Tables, 5, HMSO, London, 1962.

[DEL70] DeLugish, B. G., "A Class of Algorithms for Automatic Evaluation of Certain Elementary Functions in a Binary Computer," Ph.D. Dissertation, Department of Computer Science, University of Illinois, Urbana, Illinois, June, 1970.

[DEM73] Demidovich, B. P., I. A. Maron, Computational Mathematics, Moscow, Mir Publishers, 1973.

[ERC73] Ercegovac, M. D., "Radix-16 Evaluation of Certain Elementary Functions," IEEE Trans. Comput., Vol. C-22, pp. 561-566, June, 1973.

[FAD63] Faddeev, D. K. and V. N. Faddeeva, Computational Methods of Linear Algebra, San Francisco, W. H. Freeman and Co., 1963.

[FRA73] Franke, R., "An Analysis of Algorithms for Hardware Evaluation of Elementary Functions," Naval Post-graduate School, Monterey, California, AD-761519, May, 1973.

[HAR68] Hart, J. F., Computer Approximations, New York, John Wiley & Sons, Inc., 1968.

[HO73] Ho, I. T. 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, August, 1973.

[KUC73] Kuck, D. J., "Multioperation Machine Computational Complexity," Proceedings of Symposium on Complexity of Sequential and Parallel Numerical Algorithms, pp. 17-47, Academic Press, 1973.

[KUC74] Kuck, D. J., "On the Speedup and Cost of Parallel Computation," Proceedings on the Complexity of Computational Problem Solving, The Australian National University, December, 1974.

[KUN74] Kung, H. T., "New Algorithms and Lower Bounds for the Parallel Evaluation of Certain Rational Expressions," Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, Technical Report, February, 1974.

[ROB58] Robertson, J. E., "A New Class of Digital Division Methods," IRE Trans. Electron. Comput., Vol. EC-7, pp. 218-222, eptember, 1958.

[SPE65] Specker, W. H., "A Class of Algorithms for lnx, expx, sinx, cosx, $\tan^{-1}x$, and $\cot^{-1}x$," IEEE Trans. Electron. Comput. (Short Notes), Vol. EC-14, pp. 85-86, February, 1965.

[SVO63] Svoboda, A., "An Algorithm for Division," Information Processing Machines, Vol. 9, pp. 25-32, 1963.

[TUN68] Tung, C., "A Combinational Arithmetic Function Generation System," Ph.D. Dissertation, Department of Engineering, University of California, Los Angeles, California, June, 1968.

[TUN70] Tung, C. and A. Avizienis, "Combinational Arithmetic Systems for the Approximation of Functions," AFIPS Conference Proceedings, Spring Joint Computer Conference, pp. 95-107, 1970.

[VOL59] Volder, T. E., "The CORDIC Trigonometric Computing Technique," IEEE Trans. Electron. Comput., Vol. EC-8, pp. 330-334, September, 1959.

[WAL71] Walther, T. S., "A Unified Algorithm for Elementary Functions," AFIPS Conference Proceedings, Spring Joint Computer Conference, pp. 379-385, 1971.

VITA

Milos D. Ercegovac was born in Belgrade, Yugoslavia, in 1942. He received a Diploma in Electrical Engineering from the University of Belgrade in 1965, his M.S. and Ph.D. degrees in Computer Science from the University of Illinois at Urbana-Champaign, in 1972 and 1975, respectively.

From 1966 to 1970, Mr. Ercegovac was associated with the Institute for Automation and Telecommunications "Mihailo Pupin," Belgrade, where he was involved in the design and development of digital computing systems. From 1970 to 1975 he was employed as a graduate research assistant by the Department of Computer Science at the University of Illinois where he was a participant in the Digital Computer Arithmetic Group, headed by Professor James E. Robertson. He is an associate member of Sigma Xi.

Presently, he is an assistant professor in the Computer Science Department at the University of California, Los Angeles.