On the Parallel Evaluation of Polynomials

KIYOSHI MARUYAMA

Abstract—**If an unlimited number of processors is available, then for any given number of steps s, $s>1$, polynomials of degree as large as $C2^{s-\delta}$ can be evaluated, where $C=\sqrt{2}$ and $\delta\approx\sqrt{2s}$. This implies polynomials of degree can be evaluated in $\log_2 n+\sqrt{2\log_2 n}+0(1)$ steps. Various techniques for the evaluation of polynomials in a "reasonable number" of "steps" are compared with the known lower bounds.**

Index Terms—**Parallel algorithms, polynomial evaluation.**

I. Introduction

THE evaluation of polynomials has been studied for many years. It has been shown by Pan [7], Winograd [8], and others that $2n$ operations are required to evaluate a general polynomial of degree n. Thus, for a serial machine, Horner's rule is optimal. However, it is easy to see that on a fully parallel machine it requires only $\lceil\log_2(n+1)\rceil$ steps to evaluate all of the terms, and $\lceil\log_2(n+1)\rceil$ steps to add those terms. Thus by introducing some "redundant" operations one can obtain the result in $2\lceil\log_2(n+1)\rceil$ steps (assuming multiplication and addition times are equal). This is a crude bound for a multiarithmetic unit machine because some additions can be performed before the final multiplications are completed.

In this paper we shall investigate algorithms for polynomial evaluation that allow parallelism. We first assume that arbitrarily many processors are available. Two previously known methods are Estrin's [3] and the Kth order of Horner's rule [2], which require at least $2\lceil\log_2(n+1)\rceil$ and $\lceil\log_2 n\rceil+\lceil\log_2(n+1)\rceil+1$ steps, respectively. Two other methods, a "tree" method and a "folding" method,[1] have been developed by Muraoka [6]. It has been shown by Munro and Paterson [5] that at least $\lceil\log_2 n\rceil+1$ steps are required to evaluate a general nth degree polynomial.

We write an nth degree polynomial as

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \qquad (1)$$

where we assume $x, a_0, a_1, \cdots, a_n$ are algebraically independent reals.

The *dual problem* to the problem of finding the minimum number of steps s required to evaluate $P_n(x)$ is the problem of finding the maximum N for which the polynomial $P_N(x)$ may be evaluated for a particular given method in s steps. This dual problem will be considered later.

II. Bounds for Parallel Evaluation of Polynomials

$N(s)$ will denote the degree of polynomial that can be evaluated in s steps.

Theorem 1: If an unlimited number of processors is available, then $N(D_r)\geq 2^{D_{r-1}}$ holds, where $D_r=r(r+1)/2+i$, for $r\geq 2$, $r-1\geq i\geq -1$.

Proof: We prove by induction that polynomials of degree as large as $2^{D_{r-1}}$ can be evaluated in D_r steps.[2] The proof of this theorem, for $i=0$, can be found in Munro and Paterson [5].

The result is true for $r=2$, since $N(3+i)\geq 2^{1+i}$ for $1\geq i\geq -1$. We assume that the result is true for $r\leq k$; we prove it for $r=k+1$.

By induction hypothesis, at time D_k, we have available any polynomial of degree less than or equal to $2^{D_{k-1}}$, and we also have all powers of x up to and including 2^{D_k}. An arbitrary polynomial of degree less than or equal to 2^{D_k} may be expressed in the form

$$P(x) = \sum_{j=0}^{2^k-1} Q_j(x)x^{j2^{D_{k-1}}}$$

where $Q_j(x)$ are polynomials of degree less than or equal to $2^{D_{k-1}}$. It is clear, however, that $P(x)$ can be evaluated in D_{k+1} steps, from which the theorem follows. Q.E.D.

Corollary 1 (Dual Property): For any given $s>1$, polynomials of degree as large as $C2^{s-\delta}$ can be evaluated in s steps, where $C=\sqrt{2}$ and $\delta\approx\sqrt{2s}$.

Proof: Easily derived from Theorem 1. Q.E.D.

We now revert to the problem of evaluating a polynomial of degree n in as few steps as possible. From Theorem 1 and Corollary 1 we deduce that[3]

$$T_\infty(n) \leq \log_2 n + \sqrt{2}\log_2 n/\sqrt{\log_2 n - 1} + 0(1)$$

when

$$\log_2 n = r(r-1)/2 + i, r\geq 2, r-1\geq i\geq -1.$$

Manuscript received July 21, 1971; revised August 28, 1972. This work was supported in part by the Department of Computer Science, University of Illinois, Urbana, Ill., and in part by NSF Grant GJ-328.

The author is with the Department of Computer Science, University of Illinois, Urbana, Ill. 61801.

[1] Muraoka's folding method requires at least $C\log_2 n+0(1)$ steps, where $C=1/\log_2((\sqrt{5}+1)/2)\approx 1.46$. We use the standard notation for the order of magnitude of a function: $f(n)=0(g(n))$ if there is a constant $k>0$ such that $\lim_{n\to\infty}\sup(f(n)/g(n))=k$.

[2] It is interesting to note here that in Brent [1, eq. (4), p. 758] the equation may be modified to evaluate polynomials and interpreted so that polynomials of degrees $2^{r(r-1)/2}-1$ can be evaluated in $r(r+1)/2$ steps.

[3] $T_K(n)$ denotes the least running time of all algorithms that evaluate a general nth degree polynomial. $T_\infty(n)=\min_K T_K(n)$ and corresponds to an unlimited number of processors. It is easy to see that $T_\infty(n)$ can be achieved with at most n processors.

Hence, we have the following theorem.

Theorem 2 (Primal Property):

$$T_\infty(n) \leq \log_2 n + \sqrt{2 \log_2 n} + 0(1).$$

Theorem 3 (Munro and Patterson): If we have only K processors, $0(K) = \sqrt{n}$, then

$$T_K(n) \leq 2n/K + \log_2 K + \sqrt{2 \log_2 K} + 0(1).$$

Proof: We write

$$P_n(x) = A_0(x) + A_1(x)X + \cdots + A_{K-2}(x)X^{K-2}$$

where A_i are polynomials of degree $n/(K-1)$ and $X = x^{n/(K-1)}$. We first evaluate $A_0, A_1, \cdots, A_{K-2}$ in $2n/(K-1)$ steps using Horner's rule for each A_i. Simultaneously, we also evaluate X. Finally, we get the result in another $\log_2(K-2) + \sqrt{2\log_2(K-2)} + 0(1)$ steps using Theorem 2. The total computation time is thus $2n/(K-1) + \log_2(K-2) + \sqrt{2\log_2(K-2)} + 0(1)$, from which the theorem follows. Q.E.D.

III. Computation Trees

This section is provided to show algorithms that satisfy the bound given by Theorem 2.

$P_n(x)$ may be divided into q *segments*, each segment consisting of a subpolynomial multiplied by a power of x (i.e., $Q_{n_i}(x)x_i^m$). Thus we may write

$$P_n(x) = \sum_{i=1}^{q-1} Q_{n_i}(x)x^{m_i} + P_m(x) \qquad (2)$$

where $n = m + \sum_{i=1}^{q-1}(n_i+1)$ and $m_i = m + i + \sum_{j<i} n_j$. Such a segmentation of a polynomial is called a *q-cut*. To compute a segment in s' steps, both the subpolynomial and a power of x should be evaluated in $(s'-1)$ steps, i.e., $s'-1 \geq T_\infty(n_i)$ and $s'-1 \geq \lceil \log_2 m_i \rceil$, for all i. A segment is said to be *consistent* if both conditions are satisfied.

It is convenient to consider (2) as a computation tree to evaluate $P_n(x)$. The subcomputation tree to evaluate the $\sum$ summation of (2) is called the left-hand tree (LHT), while the subcomputation tree for $P_m(x)$ is called the right-hand tree (RHT).

Examples of two computation trees, a 2-cut and a 3-cut, which respectively evaluate polynomials of degree 33 and 36 at step 8, are illustrated in Fig. 1. As can be seen, it is possible to increase the degree of polynomials that may be evaluated at a given step by an appropriate selection of the number of cuts.

In general, for such a computation tree, if $2^k < q-1 \leq 2^{k+1}$, $q \geq 3$, the degree of a polynomial that can be evaluated in $(s+1)$ steps, $N(s+1)$, by a q-cut is given by

$$N(s+1) = N(s) + \sum_{i=0}^{M+L} (N(s-k-1+M-i)+1)A_i \qquad (3)$$

where we have the following.

1) $q-1 = \sum_{i=0}^{M+L} A_i$, A_i is the number of segments that require $(s-k+M-i)$ number of steps and A_{M+L} is a multiple of 2.

2) $s-k-L$ is the least number of steps required to evaluate a segment in the LHT.

3) $s-k+M$ is the most number of steps required to evaluate a segment in the LHT.

The details of (3) can be found in Maruyama [4].

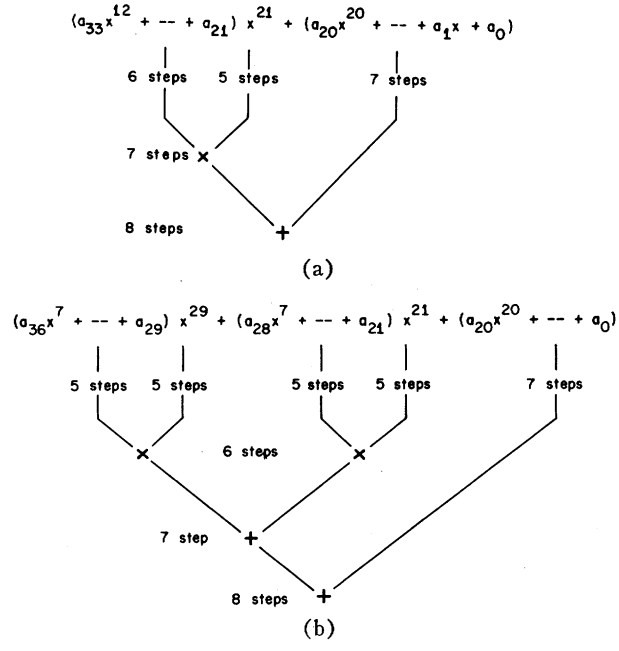

Fig. 1. Examples of computation trees at step 8. (a) The computation tree for Muraoka's folding method, a 2-cut at step 8, which evaluates a polynomial of degree 33. (b) The computation by a 3-cut at step 8, which evaluates a polynomial of degree 36.

A computation tree of (3) is said to be *q-consistent* if each segment in the LHT is consistent. A lower order first (LOF) computation tree is a balanced binary tree, i.e., $q-1 = A_0 + A_1$, such that the minimum power of x of segments in A_0 is greater than the maximum power of x of segments in A_1 for $2^k < q-1 < 2^{k+1}$. If $q-1 = 2^{k+1}$, then an LOF computation tree is a completely balanced tree. It is important to notice that for a given s and q, the LOF computation tree is unique.

It is shown in [4] that the LOF computation tree is better than the usual balance tree to solve the dual problem. Hereafter we consider the LOF computation tree for evaluating polynomials of degree n, as illustrated in Fig. 2.

Theorem 4: For $q \geq 3$ such that $2^k < q-1 \leq 2^{k+1}$, if the q-consistency condition

$$s - k - 2 \geq \lceil \log_2 (N(s) + (2q - 2^{k+1} - 3) \cdot (N(s-k-2)+1) + 1) \rceil \qquad (4)$$

for the LOF computation tree at step $(s+1)$ holds, then

$$N(s+1) = N(s) + 2(q-1-2^k)(N(s-k-2)+1) + (2^{k+1} - q + 1)(N(s-k-1)+1). \qquad (5)$$

For $q=2$, $N(s+1) = N(s) + N(s-1) + 1$.

Proof: It is easily derived by an inspection of the

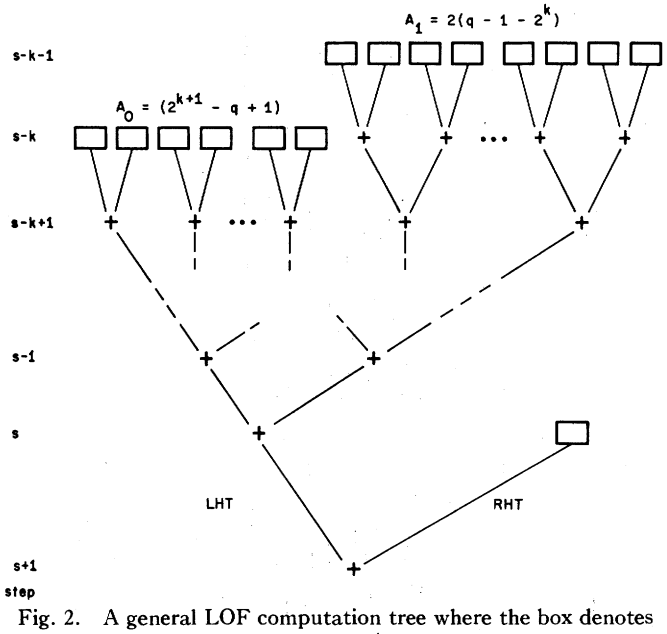

Fig. 2. A general LOF computation tree where the box denotes a segment.

LOF computation tree and the definition of q-consistency. Q.E.D.

Corollary 2: If an LOF computation tree is q-consistent, then $N(s+1)$ given by a q-cut is greater than $N(s+1)$ given by a $(q-1)$-cut, for $1 \leq i \leq q-3$.

Proof: A q-consistent LOF computation tree is clearly $(q-i)$-consistent for $1 \leq i \leq q-2$. It is sufficient to show that $N(s+1)$ given by a q-cut is greater than $N(s+1)$ given by a $(q-1)$-cut. Q.E.D.

Corollary 2 implies that to maximize $N(s+1)$ we should choose the maximum q such that the LOF computation tree is q-consistent. This *multifolding* approach has also been discovered (independently and essentially simultaneously) by Munro and Paterson [5]. The q-consistency condition of (4) for an LOF computation tree contains information to increase the degree of (5) by forcing β,

$$\beta = 2^{s-k-2} - N(s) - (2q - 2^{k+1} - 3)(N(s - k - 2) + 1) - 1$$

to be zero and achieving $(q+1)$-consistency. We call this approach the *modified multifolding* method. It is shown in [4] that the modified multifolding method is at least as good as the multifolding method.

Table I shows the maximum degree of polynomials evaluated by Muraoka's folding method and the modified multifolding method. It is easy to see that each $N(s)$ in the table satisfies the bound given by Theorem 1. The amplification factor $\alpha(s)$ for the multifolding method, defined by the ratio between $N(s)$ and $N(s-1)$, approaches 2 as s increases. After the seventh step, the multifolding method becomes superior to Muraoka's folding method. The greatest known degrees of polynomials $B(s)$ that may be evaluated at step s and the theoretical lower bound $\lceil \log_2 B(s) \rceil + 1$ are also shown in the table. To the author's knowledge a slight improvement on $B(s)$ for $s \geq 15$ can be expected, however, it will not affect the lower bound.

TABLE I
DEGREES OF POLYNOMIALS THAT CAN BE EVALUATED IN GIVEN STEPS

s	Folding Method	Multifolding Method N(s)	q	Best Known B(s)	q	$\lceil \log B(s) \rceil + 1$
1	0	0	-	0*	-	
2	1	1	2	1*	2	2
3	2	2	2	2*	2	3
4	4	4	2	4*	2	4
5	7	7	2	7*	2	4
6	12	12	2	12*	2	5
7	20	20	2	21*	-	6
8	33	36	3	37*	3	7
9	54	62	3	63*	3	7
10	88	104	3	107*	3	8
11	143	183	4	187*	-	9
12	232	320	4	327	4	10
13	376	572	5	575	5	11
14	609	992	5	1,007	5	11
15	986	1,728	5	1,759	5	12
16	1,596	3,059	6	3,119	6	13
17	2,583	5,489	7	5,575	7	14
18	4,180	9,767	7	9,895	7	15
19	6,764	17,454	8	17,703	8	16
20	10,945	31,286	9	31,783	9	16
21	17,710	55,915	10	56,743	9	17
22	28,656	101,095	11	102,111	11	18
23	46,367	182,875	12	185,047	12	19
24	75,024	330,839	13	335,031	13	20
25	121,392	602,873	15	607,423	14	21
26	196,417	1,096,807	16	1,109,143	16	22
27	317,810	1,991,463	17	2,017,047	17	22
28	514,228	3,619,735	18	3,662,215	18	23
29	832,039	6,603,699	20	6,680,511	20	24
30	1,346,268	12,071,699	22	12,216,343	22	25
31	2,178,308	22,129,325	24	22,373,607	24	26
32	3,524,577	40,738,153	27	41,071,247	26	27
33	5,702,886	75,027,401	29	75,758,895	29	28

Note: Those degrees marked with an asterisk have been found by Munro and Paterson [5].

IV. Conclusion

The author has found a simple computation tree called LOF (or the multifolding method) that is a generalization of Muraoka's folding method. It evaluates polynomials of degree n in a number of steps that are close to the lower bound for large n.

It would seem that the preceding results could be extended to the case in which divisions are permitted. It is felt, however, that the multifolding method is a reasonable model of computation permitting parallelism, even for the case in which only a limited number of processors is available.

Acknowledgment

The author wishes to thank Dr. D. Kuck, Professor of Computer Science at the University of Illinois,

Urbana-Champaign, for suggesting the problem and his comments. He also wishes to thank Dr. Y. Muraoka for his assistance and helpful discussion.

References

[1] R. Brent, "On the addition of binary numbers," *IEEE Trans. Comput.* (Short Notes), vol. C-19, pp. 758–759, Aug. 1970.
[2] W. Dorn, "Generalizations of Horner's rule for polynomial evaluation," *IBM J. Res. Develop.*, vol. 6, pp. 239–245, Apr. 1962.
[3] G. Estrin, "Organization of computer systems—The fixed plus variable structure computer," in *Proc. Western Joint Comput. Conf.*, May 1960, pp. 33–40.
[4] K. Maruyama, "Parallel methods and bounds of evaluating polynomials," Dep. Comput. Sci., Univ. Illinois, Urbana, Rep. 437, Mar. 1971.
[5] I. Munro and M. Paterson, "Optimal algorithms for parallel polynomial evaluation," in *Proc. IEEE 12th Annu. Symp. Switching and Automata Theory*, Oct. 1971, pp. 132–139.
[6] Y. Muraoka, "Parallelism exposure and exploitation in programs," Ph.D. dissertation, Dep. Comput. Sci., Univ. Illinois, Urbana, 1971, pp. 33–41.
[7] V. Pan, "Methods of computing values of polynomials," *Russ. Math. Surv.*, vol. 21, pp. 105–136, Jan.–Feb. 1966.
[8] S. Winograd, "On the number of multiplications required to compute certain functions," *Proc. Nat. Acad. Sci. U. S.*, vol. 58, pp. 1840–1842, 1968.

Kiyoshi Maruyama was born in Japan, on December 5, 1945. He received the B.S. degree in engineering from Nihon University, Tokyo, Japan, in 1968 and the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana, in 1970 and 1972, respectively.

He is currently with the Department of Computer Science, University of Illinois. His current areas of interest are parallel algorithms and picture processing.

Dr. Maruyama is a member of the Association for Computing Machinery.

Error Correction in Redundant Residue Number Systems

STEPHEN SIK-SANG YAU AND YU-CHENG LIU

***Abstract*—Two error-correcting algorithms for redundant residue number systems are presented, one for single residue-error correction and the other for burst residue-error correction. Neither algorithm requires table lookup, and hence their implementation needs a memory space which is much smaller than that required by existing methods. Furthermore, the conditions which the moduli of the redundant residue number systems must satisfy for single residue-error correction are less restrictive than that of existing methods. Comparison of the approach on which these two algorithms are based and that of existing methods is given.**

***Index Terms*—Algorithms, burst residue errors, conditions for moduli, error correction, memory requirement, redundant residue number systems, single residue errors, speed.**

Introduction

A MAJOR ADVANTAGE for using residue number systems in computer systems is that it has error detection and correction capability by adding some redundant residues [1]. Such a number system is called a *redundant residue number system*. Cheney [2] first proposed a method for error correction in redundant residue number systems which can correct any single bit error when the residues are encoded in a binary code. But his method is very time consuming and the condition required for single bit correction is very restrictive. Szabo and Tanaka [3] presented a method for correcting single residue error which seems to be too complicated for implementation. Watson [4] established a method for single residue-error correction which needs a correction table. He also stated the necessary and sufficient conditions which the moduli must satisfy in order to apply this correction method. In this paper we will present a single residue-error correction algorithm which does not need a correction table, and hence requires a much smaller memory space for implementation than the existing methods. Furthermore, the conditions which the moduli must satisfy are less restrictive. An algorithm for burst residue-error correction and the sufficient conditions for a set of positive integers to be used as its moduli of the redundant residue number system will also be given.

Manuscript received September 8, 1971; revised April 19, 1972. This paper was presented at the IEEE Symposium on Computer Arithmetic, University of Maryland, College Park, Md., May 1972.

S. S.-S. Yau is with the Departments of Electrical Engineering and Computer Sciences and the Biomedical Engineering Center, Northwestern University, Evanston, Ill. 60201.

Y.-C. Liu was with the Department of Electrical Engineering and the Biomedical Engineering Center, Northwestern University, Evanston, Ill. He is now with the Department of Electrical Engineering, University of Texas at El Paso, El Paso, Tex. 79968.

Single Residue-Error Correction

Let $m_1, m_2, \cdots, m_k$ be k positive integers, each m_i is called a *modulus*. For any integer X, the least positive remainder of dividing X by m_i is represented as $|X|_{m_i}$ and called the *residue* of X modulo m_i (or X mod m_i). We call the k tuple $(|X|_{m_1}, |X|_{m_2}, \cdots, |X|_{m_k})$ as the *residue representation* of the positive number X with the