W. S. Dorn

Generalizations of Horner's Rule for Polynomial Evaluation*

Abstract: Polynomials are generally evaluated by use of Horner's rule, sometimes referred to as the nesting rule. This rule is sequential and affords no opportunity for parallel computation, i.e., completion of several of the arithmetic operations simultaneously. Two generalizations of Horner's rule which allow for parallel computation are presented here.

Schedules and, in some cases, machine codes for evaluating a polynomial on a computer with several parallel arithmetic units are developed. Some advantages of the generalized rules in sequential computations on a computer with a single arithmetic unit are presented.

1. Introduction

The prospect of high-speed digital computers possessing several arithmetic units which may operate simultaneously requires a reappraisal of many of the standard methods and techniques of numerical analysis. Indeed, these methods were all developed with a sequential mode of operation in mind. Classical numerical methods were designed for an individual using either paper and pencil or a desk calculator. Even modern refinements were tailored for digital computers in which only one arithmetic operation could be performed at any given time.

Many of these classical methods should not be expected to be well adapted to computers having several parallel arithmetic units. The purpose of this article is to investigate one classical problem—that of the evaluation of a simple polynomial from this point of view. A complete analysis is given for this problem which is so prevalent in modern computing.

Polynomials are usually evaluated by Horner's rule, sometimes referred to as the *nesting rule*. This rule, however, is entirely sequential in the sense that none of its arithmetic operations may be performed simultaneously. After a brief review of Horner's rule (Section 2), two generalizations which allow for simultaneous arithmetic are derived in Section 3. Schedules which evaluate a polynomial in minimum time on computers with two, three or four arithmetic units are given in Section 4. Actual machine codes are written for a certain class of parallel computers, and estimates are given regarding the maximum number of arithmetic units which may be used efficiently.

Finally, Section 5 describes some advantages that the generalized Horner's rules provide even for sequential computers with a single arithmetic unit. In particular, the problem of integrating a rational function by use of Gauss quadrature is shown to require fewer arithmetic operations if the generalized rule is utilized in the computation.

2. Horner's rule

Consider a polynomial $p(x)$ of degree n

$$p(x) = a_0 + a_1x + \cdots + a_nx^n \tag{2.1}$$

and divide $p(x)$ by a linear factor $x - x_0$

$$p(x) = (x - x_0)(b_1 + b_2x + \cdots + b_nx^{n-1}) + b_0 . \tag{2.2}$$

The remainder, b_0, and the coefficients, $b_1, b_2, \cdots, b_n$, in the quotient are readily obtained by equating the coefficients of like powers of x in (2.1) and (2.2) as follows:

$$b_n = a_n \tag{2.3}$$

$$b_j = a_j + x_0b_{j+1} \qquad j = n-1, \cdots, 0 . \tag{2.4}$$

The b_j may be computed recursively from (2.3) and (2.4). Moreover, it follows from (2.2) that

$$p(x_0) = b_0 . \tag{2.5}$$

This method for evaluating the polynomial $p(x)$ at $x = x_0$ is *Horner's rule* and may be expressed alternatively by

$$p(x_0) = a_0 + x_0\{a_1 + x_0[a_2 + \cdots + x_0(a_n)\cdots]\} . \tag{2.6}$$

* Presented at the 16th National Conference of the Association for Computing Machinery, Los Angeles, California, September 5-8, 1961.

Horner's rule requires n multiplications and n additions for the evaluation of $p(x_0)$.

It can be shown that at least n additions are required to compute $p(x_0)$. It is also generally accepted that a total of $2n$ operations (additions and/or multiplications) are necessary to compute $p(x_0)$. No proof of this latter fact exists, however, except for $n \leqq 4$ (Ref. 1).

The minimum number of multiplications necessary to evaluate $p(x_0)$ is likewise an open question. Again for $n \leqq 4$ at least n multiplications are necessary.[1] Motzkin[2] has shown that for $n = 6$ only $n/2 = 3$ multiplications are required. Motzkin does not state the number of additions required by his algorithm.

It is clear from (2.4) that Horner's rule is sequential in the sense that for any j, b_j cannot be computed until all the b_i for $i = n, n-1, \cdots, j+1$ have been computed. It follows that none of the arithmetic operations may be performed in parallel. Thus the availability of a computer with several arithmetic units which can operate simultaneously would not decrease the time in which $p(x_0)$ could be calculated.

In the following section two generalizations of Horner's rule which allow for simultaneous operation of arithmetic units will be developed.

3. Generalizations of Horner's rule

To obtain Horner's rule the polynomial $p(x)$ was divided by a linear factor $x - x_0$, and the remainder was, therefore, $p(x_0)$. An obvious generalization is to divide $p(x)$ by a polynomial $q(x)$ which has x_0 as a root, i.e., $q(x_0) = 0$. Then the remainder, evaluated at $x = x_0$, is $p(x_0)$.

In particular, choose $q(x) = x^k - {x_0}^k$ where $k \geqq 1$.

$$p(x) = (x^k - {x_0}^k)(b_k + b_{k+1}x + \cdots + b_n x^{n-k}) + b_{k-1}x^{k-1} + \cdots + b_1 x + b_0 . \tag{3.1}$$

By equating the coefficients of like powers of x in (2.1) and (3.1) it follows that

$$b_j = a_j \qquad j = n, \cdots, n-k+1 \tag{3.2}$$

$$b_j = a_j + {x_0}^k b_{j+k} \qquad j = n-k, \cdots, 0 . \tag{3.3}$$

Moreover,

$$p(x_0) = b_{k-1}{x_0}^{k-1} + \cdots + b_1 x_0 + b_0 . \tag{3.4}$$

The computation of the b_j in (3.3) and the subsequent evaluation of $p(x_0)$ in (3.4) requires n additions and $n + k - 1$ multiplications. For $k = 1$ this reduces to Horner's Rule. The generalized rule given by (3.2) and (3.3) will be referred to as the k^{th} *order Horner's rule.*

Notice now that once the b_i have been computed for $i = n, n-1, \cdots, j$ (where $j \leqq n - k + 1$) then $b_{j-1}, b_{j-2}, \cdots, b_{j-k}$ can all be computed simultaneously. That is to say, k arithmetic units operating in parallel could compute k of the b_j in one addition time plus one multiplication time.

For large n, the time to compute $p(x_0)$ on a computer with k arithmetic units which operate in parallel is of the order of n/k multiplication times plus n/k addition times using the k^{th} order Horner's rule. A detailed analysis of the exact time requirements for several values of k is given in Section 4.

Another generalization of Horner's rule which allows for parallel computation has been given by Estrin.[3]

First compute

$$c_i^{(0)} = a_i + x_0 a_{i+1} \qquad i = 0, 2, \cdots, 2\lfloor n/2 \rfloor \tag{3.5}$$

where $\lfloor y \rfloor$ denotes the largest integer less than or equal to y. Then successively compute

$$\begin{aligned} c_i^{(1)} &= c_i^{(0)} + {x_0}^2 c_{i+2}^{(0)} && i = 0, 4, \cdots, 4\lfloor n/4 \rfloor \\ c_i^{(2)} &= c_i^{(1)} + {x_0}^4 c_{i+4}^{(1)} && i = 0, 8, \cdots, 8\lfloor n/8 \rfloor \\ &\vdots \qquad\qquad \vdots && \vdots \\ c_i^{(m)} &= c_i^{(m-1)} + {x_0}^{2^m} c_{i+2^m}^{(m-1)} && i = 0, 2^{m+1}, \cdots, 2^{m+1}\lfloor n/2^{m+1} \rfloor . \end{aligned} \tag{3.6}$$

The process will terminate when $m = \lfloor \log_2 n \rfloor$ and, moreover,

$$p(x_0) = c_0^{(m)} . \tag{3.7}$$

This procedure also may be expressed by

$$\begin{aligned} p(x_0) = a_0 &+ a_1 x_0 + {x_0}^2(a_2 + a_3 x_0) + {x_0}^4[a_4 + a_5 x_0 \\ &+ {x_0}^2(a_6 + a_7 x_0)] + {x_0}^8\{a_8 + a_9 x_0 \\ &+ {x_0}^2(a_{10} + a_{11}x_0) + {x_0}^4[a_{12} + a_{13}x_0 \\ &+ {x_0}^2(a_{14} + a_{15}x_0)]\} \\ &+ {x_0}^{16}\langle a_{16} + \{[\cdots + (a_{30} + a_{31}x_0)]\}\rangle \\ &+ \cdots \end{aligned} \tag{3.8}$$

Now notice that for each j all $c_i^{(j)}$ may be computed simultaneously in one addition time plus one multiplication time. Since there are $\lfloor \log_2 n \rfloor + 1$ values of j, the minimum time to compute $p(x_0)$ using this algorithm is

$$T = (\lfloor \log_2 n \rfloor + 1)(t_a + t_m) ,$$

where t_a is the time required for one addition, and t_m is the time for one multiplication.

In order to achieve this minimum time, however, it is necessary that the computer possess sufficient arithmetic units to compute all $c_i^{(j)}$ simultaneously for any j.

The maximum number of $c_i^{(j)}$ for any j occurs for $j = 0$, and there are $\lfloor n/2 \rfloor + 1$ of the $c_i^{(0)}$. If n is even the final $c_i^{(0)}$ is

$$c_n^{(0)} = a_n$$

and does not require any computation. The number

of $c_i^{(0)}$ which must be calculated, therefore, is given by $\lfloor (n+1)/2 \rfloor$. In order to calculate the $c_i^{(1)}$, however, x_0^2 must also be calculated. This can be done at the same time as the $c_i^{(0)}$ are computed. Thus $\lfloor (n+1)/2 \rfloor + 1 = N$ arithmetic units must be used initially. With N arithmetic units, all of $c_i^{(0)}$ and x_0^2 can be computed in one addition time and one multiplication time.

For $j \geqq 1$ the $c_i^{(j)}$ are fewer than $N-1$ in number, so all of the $c_i^{(j)}$ and $x_0^{2^{j+1}}$ can be calculated in one addition time and one multiplication time on N parallel arithmetic units.

If fewer than N arithmetic units are available a non-trivial scheduling problem arises if the computing time is to be minimized. This scheduling problem is discussed elsewhere[4] and will not be considered here.

4. Schedules for parallel computers

From the discussion of the previous section it appears that for a computer with k parallel arithmetic units the k^{th} order Horner's rule provides the fastest way to evaluate a polynomial. In general, it is to be expected that there will be relatively few arithmetic units available. Therefore, a detailed analysis for the case of two, three or four parallel arithmetic units will be given here. It will be assumed that all arithmetic units are identical.

A. *Second-order Horner's rule*

The second-order Horner's rule is

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} \\ b_j &= a_j + x_0^2 b_{j+2} \qquad j = n-2, \cdots, 0 \end{aligned} \tag{4.1}$$

and

$$p(x_0) = b_0 + b_1 x_0 \,. \tag{4.2}$$

This is equivalent to evaluating two polynomials in x^2 of degree $n/2$ by a first-order Horner's rule as follows

$$\begin{aligned} p(x_0) = \langle a_0 + x_0^2 \{ a_2 + x_0^2 [\cdots + x_0^2 (a_{2\lfloor n/2 \rfloor})] \} \rangle \\ + x_0 \langle a_1 + x_0^2 \{ a_3 + x_0^2 [+ x_0^2 \\ \times (a_{2\lfloor (n-1)/2 \rfloor} + 1)] \} \rangle \,. \end{aligned} \tag{4.3}$$

The formulation (4.3) was previously given by K. Ralston (see, e.g., Ref. 3).

If two parallel arithmetic units are available, the process is started by computing x_0^2 on one unit while the second sits idle. Then $x_0^2 a_n$ and $x_0^2 a_{n-1}$ are computed simultaneously followed by $b_{n-2} = a_{n-2} + (x_0^2 a_n)$ and $b_{n-3} = a_{n-3} + (x_0^2 a_{n-1})$. The complete schedule is given in Appendix I. Notice that the terminal steps vary depending on whether n is even or odd, but in either case a total of $n+2$ steps is necessary. The total time, T_2, is given by

$$T_2 = (\lfloor n/2 \rfloor + 1) t_a + (\lfloor (n+1)/2 \rfloor + 1) t_m \,, \tag{4.4}$$

where again t_a and t_m are the addition and multiplication times respectively. If the multiply and add times are equal ($t_m = t_a = t$) this reduces to

$$T_2 = (n+2)t \,. \tag{4.5}$$

The utilization U is defined to be

$$U = \frac{\text{total time all the arithmetic units are in use either individually or collectively}}{\text{total time all the arithmetic units are available}} \,.$$

The utilization then is a measure of how efficiently the arithmetic units are used. A utilization of 1 indicates no idle time on any unit. The utilization, U_2, of the schedule given in Appendix I is

$$U_2 = \frac{2n+1}{2(n+2)} \tag{4.6}$$

for the case $t_m = t_a$.

For large n, T_2 approaches nt, and U_2 approaches 1.

Notice that the foregoing discussion has neglected all hardware considerations. For example, no mention has been made of the number of memories the computer possesses or the access to these memories. Such considerations may significantly affect the validity of the computing time stated in (4.5). In order to determine the effect of these factors consider a mythical parallel computer of the type described in Appendix II with two arithmetic units. Suppose for the moment that $r = 1$, i.e., the arithmetic operations of addition and multiplication require 1 time cycle, as do the FETCH and STORE operations.

A program may be written for this computer (see Appendix III) which evaluates the n^{th} degree polynomial $p(x)$ in $2n + 9$ time cycles. The utilization is

$$U = \frac{4n+7}{4n+18} \,.$$

This program uses the second-order Horner's rule. For comparison, a similar computer with one arithmetic unit requires $4n + 2$ time cycles for a program based on the first-order Horner's rule. The utilization for this latter program is 1.

Thus for large n, the time required by two arithmetic units is one-half that required by one arithmetic unit, and the utilization in both cases is 1.

Additional arithmetic units, however, will not serve to further decrease the computing time. A justification and discussion of this fact will be deferred until the third- and fourth-order rules have been considered in detail.

B. *Third-order Horner's rule*

The third-order Horner's rule is

$$\begin{aligned} b_j &= a_j \qquad j = n, n-1, n-2 \\ b_j &= a_j + x_0^3 b_{j+3} \qquad j = n-3, \cdots, 0 \end{aligned} \tag{4.7}$$

which can be expressed as

$$p(x_0) = a_0 + x_0^3[a_3 + x_0^3(a_6 + \cdots)] + x_0\{a_1 + x_0^3[a_4 + x_0^3(a_7 + \cdots)]\} + x_0^2\{a_2 + x_0^3[a_5 + x_0^3(a_8 + \cdots)]\}. \quad (4.8)$$

For a computer with three parallel arithmetic units a schedule can be constructed which evaluates $p(x_0)$ in a total time of

$$T_3 = (n - \boxed{n/3} - \boxed{n/3} + 1)t_a + (\boxed{n/3} + 2)t_m, \quad (4.9)$$

where $\boxed{y}$ is the smallest integer greater than or equal to y and it is assumed that $t_m \geqq t_a$. If $t_m = t_a = t$ then

$$T_3 = (n - \boxed{n/3} + 3)t \quad \text{(See Appendix IV).} \quad (4.10)$$

The utilization in the latter case is

$$U_3 = \frac{2n+2}{3(n - \boxed{n/3} + 3)} \quad (4.11)$$

which approaches 1 for large n.

Consider now the parallel computer described in Section 4A but with *three* parallel arithmetic units ($r = 1$). The number of time cycles required to evaluate $p(x_0)$ is still of the order $2n$, and the maximum utilization for large n is 2/3. That is to say, the addition of the third arithmetic unit does not decrease the computing time. This is due to the fact that when one of the three arithmetic units has completed an addition or multiplication it must stand idle for one time cycle awaiting a memory access. This delay is necessary because there are now three units accessing the memory and only one may have access at any given time.

On the other hand, if $r = 2$ (addition and multiplication require two time cycles each) then the time requirement is approximately one-third that of a computer with a single arithmetic unit, and the limiting utilization is 1.

• C. *Fourth-order Horner's rule*

A similar analysis may be given for the fourth-order Horner's rule. The schedule using this rule on a computer with four parallel arithmetic units requires a computing time of

$$T_4 = (\boxed{n/4} + 2)t_a + (\boxed{n/4} + 2)t_m, \quad (4.12)$$

where again $t_m \geqq t_a$.

For $t_a = t_m = t$ this becomes

$$T_4 = (\boxed{n/4} + \boxed{n/4} + 4)t \quad (4.13)$$

and the utilization then is

$$U_4 = \frac{2n+3}{4(\boxed{n/4} + \boxed{n/4} + 4)}.$$

Again for the parallel computer described in Section 4A ($r = 1$) but with *four* parallel arithmetic units, the number of time cycles required to evaluate $p(x_0)$ is of the order of $2n$. In this case each unit is idle two time cycles after each arithmetic operation awaiting a memory access, and the maximum utilization is 0.5. Not until $r \geqq 3$ will it be profitable time-wise to add the fourth unit and use the fourth-order rule.

5. Use of sequential computers

The discussion thus far has been directed toward polynomial evaluation on parallel computers with multiple arithmetic units. The generalized Horner's rule also offers certain advantages for computation on sequential computers with a single arithmetic unit.

Notice from (3.1) that

$$p(\theta_j x_0) = r(\theta_j x_0), \quad (5.1)$$

where

$$r(x) = b_{k-1}x^{k-1} + \cdots + b_1 x + b_0 \quad (5.2)$$

and θ_j are the k^{th} roots of unity. Since θ_j, is in general, a complex number, additional $k - 1$ multiplications and $k - 1$ additions are all that are required to evaluate $p(\theta_j x_0)$ for any j once the coefficients $b_{k-1}, \cdots, b_0$ have been computed. All of the k values of $p(\theta_j x_0)$ can be obtained in $n + k(k-1)$ multiplications and $n + (k-1)^2$ additions. In contrast, k applications of the first-order rule for complex $\theta_j x_0$ would require $n + 2n(k-1)$ multiplications and a like number of additions.

Of particular interest is the case where $k = 2$. Then

$$p(-x_0) = b_0 - b_1 x_0. \quad (5.3)$$

Thus if $p(x_0)$ has been computed using the second-order Horner's rule, equations (4.1) and (4.2), then $p(-x_0)$ is obtained by one addition and no multiplications.

The evaluation of $p(x_0)$ *and* $p(-x_0)$ using the second-order Horner's rule requires a total of $n + 1$ multiplications and $n + 1$ additions. By way of comparison, the first-order Horner's rule requires $2n$ multiplications and $2n$ additions.

In this same connection consider the problem of evaluating the definite integral of a rational function by Gauss quadrature.

$$\int_{-1}^{1} \frac{p(x)}{q(x)}\,dx = \sum_{i=1}^{N} w_i \frac{p(x_i)}{q(x_i)}, \quad (5.4)$$

where x_i are the roots of the Legendre polynomial of degree N

$$P_N(x) = \frac{1}{2^N N!}\frac{d^N}{dx^N}(x^2 - 1)^N$$

and the weights w_i are defined as

$$w_i = \frac{1}{\prod_{\substack{j=1 \\ j\neq i}}^{N}(x_i - x_j)} \int_{-1}^{1} \prod_{\substack{j=1 \\ j\neq i}}^{N}(t - x_j)\,dt.$$

Here $p(x)$ and $q(x)$ will be assumed to be polynomials of degree n and m, respectively.

The roots x_i of the Legendre polynomial $P_N(x)$ are symmetrically placed about the origin. To evaluate the right-hand member of (5.4) then $p(x_i)$, $q(x_i)$ and $p(-x_i)$, $q(-x_i)$ are required. Using the first-order Horner's rule requires

$N(m + n + 1)$	multiplications
$N(m + n + 1) - 1$	additions
N	divisions.

On the other hand, using the second-order Horner's rule requires

$(N - \lfloor N/2 \rfloor)(m + n + 1) + 3 \cdot \lfloor N/2 \rfloor$	multiplications
$(N - \lfloor N/2 \rfloor)(m + n) + N - 1$	additions
N	divisions.

For example, for $m = n = 4$ and $N = 10$ the number of arithmetic operations are

	First-order rule	*Second-order rule*
Multiplications	90	60
Additions	89	49
Divisions	10	10

6. Conclusions and remarks

The schedules for the evaluation of a polynomial in minimum time may be classified as follows:

If the number of parallel arithmetic units, $k \geqq 1$, is relatively small compared with the degree of the polynomial, n, then the k^{th}-order Horner's rule should be used.

If the number of arithmetic units, k, is of the same order as the degree of the polynomial, n, then the generalization due to Estrin, equations (3.5) to (3.7), should be used.

These are, however, only general guides, and a detailed analysis of the schedule on the particular computer is necessary to assure efficient use. In particular, the timing of the memory accesses will dictate the number of parallel units which can be used advantageously, and hence the computational rule to be used. In the example considered in Section 4, only $r + 1$ arithmetic units could be used if the arithmetic operations required r times the time required by the memory accesses.

Regardless of the number of parallel arithmetic units, the k^{th}-order Horner's rule will complete some computations in minimum time. Evaluation of the definite integral of a rational function by Gauss quadrature, for example, can be most quickly computed by using an even-order Horner's rule as demonstrated in Section 5. Other problems requiring the evaluation of the same polynomial at values proportional to the k^{th} roots of unity can be similarly speeded up by use of the k^{th}-order rule.

Appendix I: Schedule for second-order Horner's rule

The schedule of operations on two identical arithmetic units which operate in parallel using the second-order Horner's rule is:

	Step Number	*Unit 1*	*Unit 2*
	1	$x_0 \cdot x_0$	Idle
	2	$x_0^2 \cdot a_n$	$x_0^2 \cdot a_{n-1}$
	3	$a_{n-2} + (a_n x_0^2) = b_{n-2}$	$a_{n-3} + (a_{n-1} x_0^2) = b_{n-3}$
	.	.	.
	.	.	.
	.	.	.
	.	.	.
For n even	$n - 1$	$a_2 + (b_4 x_0^2) = b_2$	$a_1 + (b_3 x_0^2) = b_1$
	n	$x_0^2 \cdot b_2$	$x_0 \cdot b_1$
	$n + 1$	$a_0 + (b_2 x_0^2) = b_0$	Idle
	$n + 2$	$b_0 + (b_1 x_0) = p(x_0)$	Idle
For n odd	$n - 1$	$x_0^2 \cdot b_3$	$x_0^2 \cdot b_2$
	n	$a_1 + (b_3 x_0^2) = b_1$	$a_0 + (b_2 x_0^2) = b_0$
	$n + 1$	$x_0 \cdot b_1$	Idle
	$n + 2$	$b_0 + (b_1 x_0) = p(x_0)$	Idle

Appendix II: Description of a parallel computer

Consider a computer with k identical arithmetic units which operate in parallel and with one memory which is available to all arithmetic units. Only one of the arithmetic units may have access to the memory at any given time. For example, if one unit is in the process of storing a word and a second unit then requests a word from memory, the second unit must wait until the store operation is completed.

The individual arithmetic units operate in the following way: Each unit possesses two registers, called A and B, which may be loaded from memory by the instruction FETCH (F) or stored into memory by the instruction STORE (S). The fetch instruction does not destroy the memory contents, nor does the store instruction destroy the register contents. The instruction MULTIPLY (M) forms the product of the contents of registers A and B and places the result in register A. Similarly, the instruction ADD (A) places the sum of the contents of the two registers in register A.

The STORE and FETCH instructions require one cycle of time, and the MULTIPLY and ADD instructions each require r cycles of time.

The following table of instructions will be used:

$F - A, m$	Fetch the contents of memory location m in memory to register A. Leave location m unaltered.
$F - B, m$	Fetch the contents of memory location m in memory register B. Leave location m unaltered.
$S - A, m$	Store the contents of register A in location m of memory. Leave register A unaltered.
$M, -$	Multiply the contents of registers A and B and place the product in register A.
$A, -$	Add the contents of registers A and B and place the sum in register A.

Appendix III: A program for a parallel computer with two arithmetic units

Consider a computer of the type described in Appendix II with two arithmetic units and with $r = 1$. The following program evaluates an n^{th} degree polynomial $p(x) = a_0 + a_1x + \cdots + a_nx^n$ using the second-order Horner's rule.

Let $a_0, a_1, \cdots, a_n$ be stored in memory locations A, $A + 1, \cdots, A + N$ respectively, and let x_0 be stored in memory location X.

Time cycle	*Unit 1*	*Unit 2*
1	$F - A, X$	Idle
2	$F - B, X$	Idle
3	$M, -$	Idle
4	$S - A, X2$	Idle
5	Idle	$F - A, X2$
6	Idle	$F - B, A + N - 1$
7	$F - B, A + N$	$M, -$
8	$M, -$	$F - B, A + N - 3$
9	$F - B, A + N - 2$	$A, -$
10	$A, -$	$F - B, X2$
11	$F - B, X2$	$M, -$
.	.	.
.	.	.

	Time cycle	*Unit 1*	*Unit 2*
For n even	$2n$	$M, -$	$F - B, A + 1$
	$2n + 1$	$F - B, A + 2$	$A, -$
	$2n + 2$	$A, -$	$F - B, X$
	$2n + 3$	$F - B, X2$	$M, -$
	$2n + 4$	$M, -$	$S - A, R$
	$2n + 5$	$F - B, A$	Idle
	$2n + 6$	$A, -$	Idle
	$2n + 7$	$F - B, R$	Idle
	$2n + 8$	$A, -$	Idle
	$2n + 9$	$S - A, R$	Idle
For n odd	$2n$	$A, -$	$F - B, X2$
	$2n + 1$	$F - B, X2$	$M, -$
	$2n + 2$	$M, -$	$F - B, A$
	$2n + 3$	$F - B, A + 1$	$A, -$
	$2n + 4$	$A, -$	$S - A, R$
	$2n + 5$	$F - B, X$	Idle
	$2n + 6$	$M, -$	Idle
	$2n + 7$	$F - B, R$	Idle
	$2n + 8$	$A, -$	Idle
	$2n + 9$	$S - A, R$	Idle

The value of $p(x_0)$ is stored in memory location R.

Appendix IV: Schedule for third-order Horner's rule

The schedule of operations on three identical arithmetic units which operate in parallel using the third-order Horner's rule is:

Step Number	*Unit 1*	*Unit 2*	*Unit 3*
1	$x_0 \cdot x_0$	Idle	Idle
2	$x_0 \cdot x_0^2$	Idle	Idle
3	$a_n \cdot x_0^3$	$a_{n-1} \cdot x_0^3$	$a_{n-2} \cdot x_0^3$
4	$a_{n-3} + (a_nx_0^3) = b_{n-3}$	$a_{n-4} + (a_{n-1}x_0^3) = b_{n-4}$	$a_{n-5} + (a_{n-2}x_0^3) = b_{n-5}$
.	.	.	.
.	.	.	.
.	.	.	.

Schedule for third-order Horner's rule *(continued)*

Step Number	*Unit 1*	*Unit 2*	*Unit 3*
Case 1, $n = 3m$			
$2m$	$a_3 + (b_6 x_0^3) = b_3$	$a_2 + (b_5 x_0^3) = b_2$	$a_1 + (b_4 x_0^3) = b_1$
$2m + 1$	$b_3 \cdot x_0^3$	$b_2 \cdot x_0^2$	$b_1 \cdot x_0$
$2m + 2$	$a_0 + (b_3 x_0^3) = b_0$	$(b_1 x_0) + (b_2 x_0^2)$	Idle
$2m + 3$	$b_0 + (b_1 x_0 + b_2 x_0^2) = p(x_0)$	Idle	Idle
Case 2, $n = 3m + 1$			
$2m$	$a_4 + (b_7 x_0^3) = b_4$	$a_3 + (b_6 x_0^3) = b_3$	$a_2 + (b_4 x_0^3) = b_2$
$2m + 1$	$b_4 \cdot x_0^3$	$b_3 \cdot x_0^3$	$b_2 \cdot x_0^2$
$2m + 2$	$a_1 + (b_4 x_0^3) = b_1$	$a_0 + (b_3 x_0^3) = b_0$	Idle
$2m + 3$	$b_1 \cdot x_0$	$b_0 + b_2 x_0^2$	Idle
$2m + 4$	$b_1 x_0 + (b_0 + b_2 x_0^2) = p(x_0)$	Idle	Idle
Case 3, $n = 3m + 2$			
$2m$	$a_5 + (b_8 x_0^3) = b_5$	$a_4 + (b_7 x_0^3) = b_4$	$a_3 + (b_6 x_0^3) = b_3$
$2m + 1$	$b_5 \cdot x_0^3$	$b_4 \cdot x_0^3$	$b_3 \cdot x_0^3$
$2m + 2$	$a_2 + (b_5 x_0^3) = b_2$	$a_1 + (b_4 x_0^3) = b_1$	$a_0 + (b_3 x_0^3) = b_0$
$2m + 3$	$b_2 \cdot x_0^2$	$b_1 \cdot x_0$	Idle
$2m + 4$	$(b_1 x_0) + (b_2 x_0^2)$	Idle	Idle
$2m + 5$	$b_0 + (b_1 x_0 + b_2 x_0^2) = p(x_0)$	Idle	Idle

In all three cases the total number of time steps is:
$n - \lfloor n/3 \rfloor + 3$.

References

1. A. M. Ostrowski, "On Two Problems in Abstract Algebra Connected With Horner's Rule," in *Studies in Mathematics and Mechanics Presented to Richard von Mises*, Academic Press, 1954, pp. 40-48.
2. T. S. Motzkin, "Evaluation of Polynomials," *Bull. Amer. Math. Soc.*, **61**, 163 (1955).
3. G. Estrin, "Organization of Computer Systems—The Fixed Plus Variable Structure Computer," *Proceedings Western Joint Computer Conference*, May, 1960, pp. 33-40.
4. W. S. Dorn, N. C. Hsu and T. J. Rivlin, *Some Mathematical Problems in Parallel Computation*, IBM Research Report (forthcoming).

Received September 20, 1961