JOURNAL OF COMPUTER AND SYSTEM SCIENCES 7, 189–198 (1973)

# Optimal Algorithms for Parallel Polynomial Evaluation

IAN MUNRO

*Department of Applied Analysis and Computer Science, University of Waterloo, Waterloo, Ontario, Canada*

AND

MICHAEL PATERSON

*Department of Computer Science, University of Warwick, Coventry, Warwicks CV4 7AL, England*

Received February 3, 1972

Algorithms for the evaluation of polynomials on a hypothetical computer with $k$ independent arithmetic processors are presented. It is shown that, provided the degree of the polynomial to be evaluated exceeds $k\lceil \log_2 k \rceil$, an algorithm given is within one time unit of optimality.

## I. INTRODUCTION

The usual algorithm for evaluating a polynomial is Horner's rule, where to calculate

$$p(x) = a_n x^n + \cdots + a_0$$

we compute successively

$$p_n = a_n ,$$
$$p_i = p_{i+1} \cdot x + a_i \qquad \text{for} \quad i = n - 1,..., 0.$$

Then $p(x) \equiv p_0$. This method is essentially sequential in that for each arithmetic operation at least one of its arguments is computed only at the immediately preceding step. In this paper we shall investigate algorithms for polynomial evaluation which allow a large amount of parallelism.

This problem has previously been reported on by Estrin [1] and Dorn [2] and Muraoka [3]. The work of Estrin and Dorn is summarized by Knuth [4].

Estrin's algorithm computes $p(x)$ as

$$p(x) = q(x)\, x^{(n/2)+1} + r(x),$$

where

$$q(x) = a_n x^{n/2} + \cdots + a_{(n/2)+1}$$

and

$$r(x) = a_{n/2} x^{n/2} + \cdots + a_0$$

and then computes $q(x)$ and $r(x)$ similarly by a binary splitting. Thus it starts by computing

$$a_1 x + a_0\,,\ a_3 x + a_2\,, \ldots$$

and then

$$(a_3 x + a_2)\, x^2 + (a_1 x + a_0), \ldots, \text{ etc.}$$

If an unlimited number of processors are available, this algorithm runs in time about 2 log $n$.

Dorn gives an algorithm for the "$k$-th order Horner's rule" which computes

$$\begin{aligned} q_0(x^k) &= a_0 + a_k x^k + a_{2k} x^{2k} + \cdots, \\ q_1(x^k) &= a_1 + a_{k+1} x^k + \cdots, \\ &\vdots \\ q_{k-1}(x^k) &= a_{k-1} + a_{k+k-1} x^k + \cdots, \end{aligned}$$

and then

$$p(x) = q_0(x^k) + x q_1(x^k) + \cdots + x^{k-1} q_{k-1}(x^k).$$

With $k$ processors this algorithm takes time at least $2n/k + 2 \log k$. Dorn also makes the point that if the parallel computer has only one memory and only one of the arithmetic units may have access to the memory at any given time, then this is a serious limitation on the number of processors which can be usefully employed.

## II. Abstract Formulation

We shall assume that we have available $k$ identical arithmetic processors, each of which can perform any one of the binary operations of $+$, $-$, $\times$, or $\div$ in unit time. For this study we ignore all problems of memory access and programming to work within the following abstract framework.

A $k$-computation from $S_0$, where $S_0$ is a set of real numbers, is a sequence of sets $S_1, S_2, \ldots$ such that for each $i \geqslant 0$,

$$S_{i+1} = S_i \cup \{y_{i_1}, \ldots, y_{i_k}\},$$

where each $y_{i_j} \in (S_i * S_i) = \{z_1 * z_2 \mid * = +, -, \times, \text{ or } \div, z_1 \in S_i, z_2 \in S_i\}$. A set $S$ is $k$-computable (from $S_0$) in time $t$ if there is a $k$-computation from $S_0$ such that $S \subseteq S_t$. For an appropriate investigation of the evaluation of polynomials we consider computations of

$$p(x) = a_n x^n + \cdots + a_0$$

from $\{x, a_0, ..., a_n\} \cup A$, where $x, a_0, ..., a_n$ are algebraically independent reals and $A$ is the set of algebraic reals. This is what we shall mean when we talk about the evaluation of a *general* $n$-th degree polynomial. Thus there can be no "short-cuts" and the formal theory should correspond to our intuitive ideas about polynomial evaluation.

Let $T_k(n)$ be the least running time of all algorithms which evaluate a general $n$-th degree polynomial. $T_\infty(n) = \min_k T_k(n)$ and corresponds to an unlimited number of processors.

## III. Preliminary Results

Lemma 1. *$x^n$ is 2-computable from $\{x\}$ in time $\lceil \log n \rceil$*[1].

*Proof.* One processor successively computes $x^2, x^4, x^8, ..., x^{2^r}, ...$, while the other processor accumulates as a product the appropriate powers of $x$ as they are generated. There will be one term in the product corresponding to each 1 in the binary notation of $n$.

Lemma 2. *$\{x, x^2, ..., x^n\}$ is $n$-computable from $\{x\}$ in time $\lceil \log n \rceil$ with only $n - 1$ operations being performed.*

*Proof.* Trivial.

Estrin's method of polynomial evaluation takes time approximately $2 \log n$. This time is achieved also by an algorithm which computes $\{x, ..., x^n\}$ in $\log n$ steps, then $\{a_0, a_1 x, ..., a_n x^n\}$ in one more step and finally combines these by pairwise additions in a further $\log n$ steps. A simple improvement of Estrin's algorithm to a binary splitting in the golden (Fibonacci) ratio instead of in halves can be shown to compute in time $t$ any polynomial of degree less than $F_{t+1}$ where $F_i$ is the $i$-th number in the Fibonacci sequence.

$$1, 1, 2, 3, 5, 8, 13, 21, ... .$$

Hence,

$$T_\infty(n) \leqslant \alpha \cdot \log n + o(\log n),$$

[1] All logarithms are taken to base 2.

where

$$\alpha = 1/\log(\tfrac{1}{2}(\sqrt{5}+1)) \simeq 1.44.$$

This algorithm was discovered independently by Muraoka [3] and has the merit of simplicity, but an asymptotically faster method is described in the next section.

## IV. Main Results

Theorem 1. *Suppose the computation of a single quantity $Q$ requires $q \geqslant 1$ binary arithmetic operations. Then the shortest $k$-computation of $Q$ is at least*

$$\lceil((q+1) - 2^{\lceil\log k\rceil})/k\rceil + \lceil\log k\rceil \quad \textit{steps}$$

*if*

$$q \geqslant 2^{\lceil\log k\rceil},$$

*and*

$$\lceil\log(q+1)\rceil \qquad \textit{otherwise.}$$

*Proof.* Let $t$ be the computation time for some algorithm which computes $Q$. At time $t$ at most 1 processor is usefully employed. At time $t-1$ at most 2 processors are usefully employed. Indeed, for all $r \geqslant 0$, at time $t-r$, at most $\min(k, 2^r)$ processors can compute values to be used in the computation of $Q$.

If

$$q > 2^{\lceil\log k\rceil} - 1 = \sum_{i=1}^{\lceil\log k\rceil} 2^{i-1}$$

(i.e., the boundedness of the parallelism affects the computation), then

$$t > \lceil\log k\rceil,$$

and so the total number of useful operations which the algorithm can perform is

$$\underbrace{1+2+4+\cdots+2^{\lfloor\log k\rfloor}+k+\cdots+k}_{\longleftarrow\; t \text{ terms} \;\longrightarrow}$$
$$= k(t - \lceil\log k\rceil) + 2^{\lceil\log k\rceil} - 1.$$

Hence

$$q \leqslant k(t - \lceil\log k\rceil) + 2^{\lceil\log k\rceil} - 1$$

which implies

$$t \geqslant \lceil(q + 1 - 2^{\lceil\log k\rceil})/k\rceil + \lceil\log k\rceil.$$

On the other hand, if the boundedness of the parallelism does not interfere with the computation, that is $t \leqslant \lceil \log k \rceil$, or $q \leqslant 2^{\lceil \log k \rceil} - 1$, the maximum number of useful operations which can be performed in $t$ steps is

$$\sum_{i=0}^{t-1} 2^i = 2^t - 1.$$

Thus

$$q \leqslant 2^t - 1$$

or

$$t \geqslant \lceil \log(q + 1) \rceil.$$

COROLLARY 1. *If* $n \geqslant k/2$, $T_k(n) \geqslant \lceil (2n + 2 - 2^{\lceil \log k \rceil})/k \rceil + \lceil \log k \rceil$.

*Proof.* Winograd [5] and others have shown that at least $2n$ arithmetic operations are required to evaluate a general $n$-th degree polynomial. Borodin [6] has shown that Horner's rule is the only way in which to perform the computation in as few arithmetics. It is obvious that Horner's rule cannot be used for parallel schemes, and so at least 1 more arithmetic operation must be performed. From this and the theorem, the corollary follows. This bound, although precise, is very awkward. At the risk of saying the obvious we note that the bound is roughly

$$\lceil (2n + 2)/k \rceil + \lceil \log k \rceil - 1.$$

In the same manner the result is obtained for the case of unbounded (which in this case basically means $k > n$) parallelism.

COROLLARY 2. $T_\infty(n) \geqslant \lceil \log(n + 1) \rceil + 1$.

We now return to algorithms for polynomial evaluation and show that the coefficient of $\alpha$ obtained in Section III can be improved to 1.

THEOREM 2. $T_\infty(n) \leqslant \log n + O((\log n)^{1/2})$.

*Proof.* We describe informally an algorithm to achieve this bound.

*Algorithm* A

Let $D_r = \frac{1}{2} r(r + 1) + 1$. We define a recursive evaluation procedure for polynomials of degree $n$. Let $p = \lceil \log(n + 1) \rceil$ and suppose $D_{r-1} < p \leqslant D_r$. The polynomial may be expressed in the form

$$p(x) = q_0(x) + q_1(x) x^{2^{p-r}} + q_2(x) x^{2 \times 2^{p-r}} + \cdots + q_{2^r - 1}(x) x^{(2^r - 1) 2^{p-r}},$$

where the $q_i(x)$ are polynomials of degree $< 2^{p-r}$.

So to compute $p(x)$, we compute the $q$'s, multiply by the appropriate power of $x$ at the next step and then use $r$ further steps to combine the $2^r$ terms by binary addition.

To prove inductively that the algorithm takes at most $p + r + 1$ steps, suppose the result is true for all degrees less than $n$, then the $q$'s can be computed in time

$$(p - r) + (r - 1) + 1 = p,$$

since $p - r \leqslant D_r - r = D_{r-1}$. The powers of $x$ required are also available in this time. So the algorithm indeed runs in time $p + r + 1$.

For the basis of the induction ($n = 1$), we observe that $r = 0$, $p = 1$, and that a polynomial of degree 1 can be computed in two time steps. Since

$$r(r - 1)/2 < p,$$

we have show that

$$T_\infty(n) \leqslant \log n + (2 \log n)^{1/2} + O(1).$$

This algorithm may be improved somewhat at the cost of complication, for example, Table II illustrates the evaluation of polynomials of degree 21. Indeed, for polynomials of low degree, the Fibonacci splitting method is in some cases better. Table I shows the degree of polynomials which may be computed at time $t$.

TABLE I

| $t =$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Fibonacci | 1 | 2 | 4 | 7 | 12 | 20 | 33 | 54 | 88 | 143 | 232 | 376 | 609 | 986 | 1590 |
| Algorithm A | 1 | 1 | 3 | 3 | 7 | 15 | 15 | 31 | 63 | 127 | 127 | 255 | 511 | 1023 | 2047 |
| Best known (Modification of Algorithm A, see also Table II) | 1 | 2 | 4 | 7 | 12 | 21 | 37 | 63 | 107 | 187 | 327 | 578 | 1010 | 1764 | 3124 |

Greatest degree of general polynomial computable in time $T_\infty(n) = t$

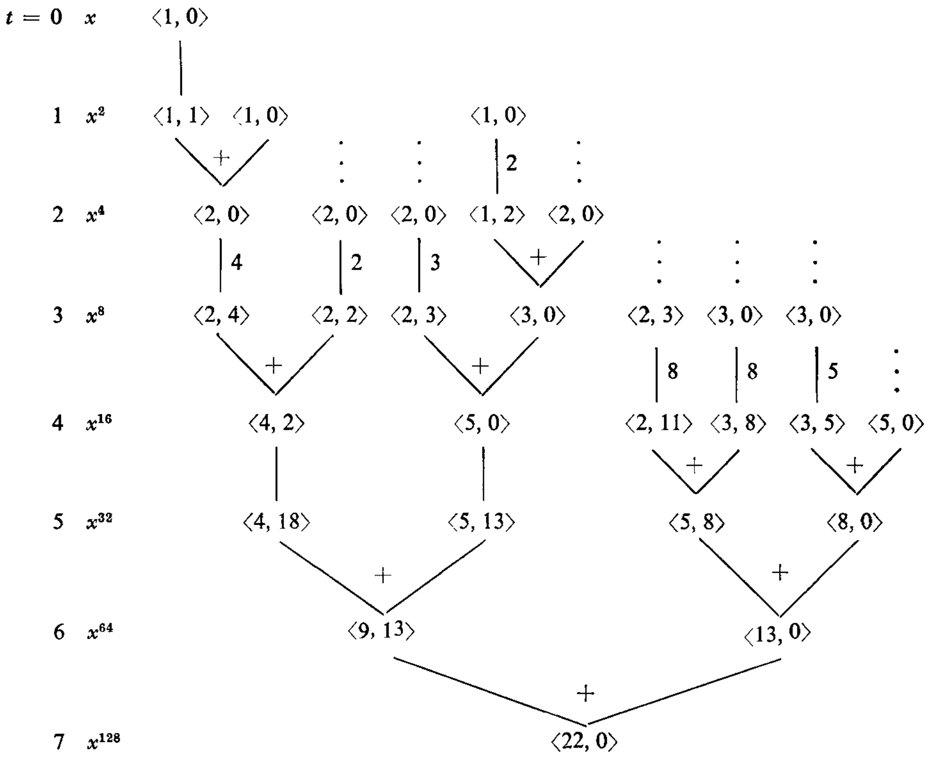
Algorithm A has also been discovered independently and essentially simultaneously by Maruyama [7].

It can be seen easily that this algorithm does not need more than $n$ processors for the evaluation of polynomials of degree $n$. We now show how the algorithm may be modified for $k < n$.

THEOREM 3. $T_k(n) \leqslant 2n/k + \log k + O((\log k)^{1/2})$ if $k \geqslant O(\log n)$.

TABLE II

Illustrating $T_\infty(21) \leqslant 7$


t = 0 x ⟨1, 0⟩
1 x² ⟨1, 1⟩ ⟨1, 0⟩ ⟨1, 0⟩
2 x⁴ ⟨2, 0⟩ ⟨2, 0⟩ ⟨2, 0⟩ ⟨1, 2⟩ ⟨2, 0⟩
3 x⁸ ⟨2, 4⟩ ⟨2, 2⟩ ⟨2, 3⟩ ⟨3, 0⟩ ⟨2, 3⟩ ⟨3, 0⟩ ⟨3, 0⟩
4 x¹⁶ ⟨4, 2⟩ ⟨5, 0⟩ ⟨2, 11⟩ ⟨3, 8⟩ ⟨3, 5⟩ ⟨5, 0⟩
5 x³² ⟨4, 18⟩ ⟨5, 13⟩ ⟨5, 8⟩ ⟨8, 0⟩
6 x⁶⁴ ⟨9, 13⟩ ⟨13, 0⟩
7 x¹²⁸ ⟨22, 0⟩
+
2
4
3
8
5

$\langle m, n\rangle$ denotes an arbitrary polynomial of the form

$$c_{m-1}x^{m+n} + \cdots + c_0 x^n,$$

$|n$ denotes a multiplication by $x^n$,
$\vdots$ means it has been done similarly elsewhere in the diagram.
Remember that $\langle m, 0\rangle$ denotes a polynomial of degree $m - 1$.

*Algorithm* A′

If we have only $k$ processors, $k < n$, for an $n$-th degree polynomial $p(x)$, we express

$$p(x) \equiv A_0(x) + A_1(x)X + \cdots + A_{k-1}(x)\,X^{k-1},$$

where $X = x^{n/k}$ and the $A_i$ are polynomials of degree $n/k$. We start the evaluation by computing all the $A_i(x)$'s and also $X$. The computation of $X$ takes at most $2\lceil(\log n - \log k)\rceil$ operations and indeed $\lceil(\log n - \log k)\rceil$ steps. By using processor $i$

compute $A_i$ by Horner's rule but pre-empting at most 2 processors (spreading the load) in each of the first $\lceil(\log n - \log k)\rceil$ steps to compute $X$, all processors may be gainfully employed till the final step. Since, at most $2n + 2\lceil\log n - \log k\rceil$ operations are performed this initial stage is completed in time

$$2n/k + 2(\log n)/k + O(1).$$

Next, Algorithm A is employed starting from $\{X, A_0, ..., A_{k-1}\}$. The total running time is thus

$$2n/k + 2(\log n)/k + \log k + O((\log k)^{1/2})$$

from which the theorem follows.

One would expect that as the number of processors $k$ decreases compared with $n$, it would be possible to use them more and more "efficiently," though of course the computation time must increase. We now turn our attention to polynomial evaluation for the cases in which $k = o(\log n)$, that is, when computing powers such as $x^n$ would seem wasteful. This approach yields an algorithm which is within one step of being optimal even if $k$ is as large as $n/\log n$.

*Algorithm* B

(1) Compute $p_i = a_{2i} + a_{2i+1}x$ for $i = 0, ..., \lfloor n/2 \rfloor$ and $x^2, x^4, ..., x^{2k-2}, x^{2k}$.

(2) Using processor $j (j = 1, ..., k)$, compute

$$p_j{}^* = \sum_{i=0}^{\lfloor n/2k \rfloor} p_{ik+j-1} (x^{2k})^i$$

by Horner's rule.

(3) Finally, evaluate

$$p(x) = \sum_{j=1}^{k} p_j{}^* \, x^{2(j-1)}$$

by forming the $p_j{}^* x^{2(j-1)}$'s and then adding them in obvious fan-in way. Note that unless $2k$ exactly divides $n + 1$ some of the terms mentioned above vanish, with a consequent saving in arithmetic operations.

To determine the number of steps required by this algorithm, first observe that all processors may be fully utilized until the final fan-in, provided the computing of powers of $x$ can be interlaced with the $p_i$'s (i.e., $n \geqslant k \log k$). Hence the number of steps used in this case is basically that obtained as the optimum in Theorem 1. A quick inspection of the algorithm shows that $n$ additions and $n + k$ multiplications, or a total of $2n + k$ operations are performed. Therefore

$$\lceil((2n + k + 1) - 2^{\lceil\log k\rceil})/k\rceil + \lceil\log k\rceil$$

steps are used provided $n \geqslant k\lceil\log k\rceil$.

From this algorithm and Corollary 1 we have now shown the following:

THEOREM 4.

$$\lceil((2n+2)-2^{\lceil \log k\rceil})/k\rceil + \lceil \log k\rceil \leqslant T_k(n)$$
$$\leqslant \lceil((2n+k+1)-2^{\lceil \log k\rceil})/k\rceil + \lceil \log k\rceil \qquad \text{provided} \qquad n \geqslant k\lceil \log k\rceil.$$

This means, of course, that Algorithm B is within 1 step of being optimal under the given condition. It is conjectured that it is actually optimal for $k > 1$ when the given condition holds.

## V. "PREPROCESSING"

If a polynomial is to be evaluated iteratively at a large number of points, it may be appropriate to compute certain functions of the coefficients first. These may be used in evaluations, thus reducing the number of operations needed for the computation. Such techniques are referred to as preconditioning or preprocessing of coefficients.

We can use the results of Motzkin [8] and Belaga [9] to establish a lower bound on the evaluation time. They show that at least $\lceil 3n/2\rceil$ operations are required to evaluate a polynomial of degree $n$, even if preconditioning is not counted. Hence by Theorem 1, the running time with $k$ processors is at least

$$\lceil 3n/2k\rceil + \lceil \log k\rceil - 2.$$

There is a very simple algorithm which is within a constant number of steps of being optimal for this case.

*Algorithm* C

The preprocessed form that we choose for a polynomial of degree $n$ is as a product of $k$ polynomials each of degree at most $\lceil n/k\rceil + 1$

$$p(x) = r_1(x) \times r_2(x) \times \cdots \times r_k(x).$$

Since $p(x)$ can be expressed as a product of quadratic and linear factors with real coefficients, we can choose the $r_i$ to be real. Each $r_i(x)$ can be computed separately with distinct processors and then they can be combined by binary multiplication in time $\lceil \log k\rceil$. Where we have an unbounded number of processors available we use just $n$.

Hence if $T^*$ corresponds to $T$ with preprocessing:

THEOREM 5.

$$T_k^*(n) = 3n/2k + \log k + O(1) \qquad \text{if} \quad n \geqslant k$$
$$T_\infty^*(n) = \log n + O(1).$$

## VI. Conclusion

We have investigated a model of arithmetic computation permitting parallelism, and presented several algorithms for the evaluation of polynomials. In addition, we have proved that the computation times of these algorithms are, in many cases, within a constant of optimality for this model.

## Acknowledgment

We should like to thank Ray Miller and Allan Borodin for helpful discussions and encouragement during the preparation of this paper.

This work was supported in part by the Advanced Research Projects Agency under ONR contract Number N00014-70-A-0362-002, International Business Machines Corporation and the National Research Council of Canada.

## References

1. G. Estrin, "Organization of Computer System—The Fixed Plus Variable Structure Computer," Proceedings Western Joint Computer Conference, May, 1960, AFIPS Press, Montvale, NJ, pp. 33–40.
2. W. S. Dorn, "Generalizations of Horner's Rule for Polynomial Evaluation," Vol. 6, IBM J. Res. Dev. (1962), 239–245.
3. Y. Muraoka, "Parallelism Exposure and Exploitation in Programs," Report No. 424, Department of Computer Science, University of Illinois, Urbana-Champaign, IL, 1971.
4. D. E. Knuth, "The Art of Computer Programming: II. Seminumerical Algorithms," pp. 422–444, Addison–Wesley, Reading, MA, 1969.
5. S. Winograd, "On the Number of Multiplications Required to Compute Certain Functions," *Proc. Nat. Acad. Sci. U.S.A.* **58** (1967), 1840–1842.
6. A. Borodin, "Horner's Rule is Uniquely Optimal," Proceedings International Symposium on the Theory of Machines and Computation, Haifa, August, 1971, (Z. Kohavi and A. Paz, eds.), Academic Press, NY, pp. 45–48.
7. K. Maruyama, "Parallel Methods and Bounds of Evaluating Polynomials," Report No. 427, Department of Computer Science, University of Illinois, Urbana-Champaign, IL, 1971.
8. T. S. Motzkin, "Evaluation of Polynomials and Evaluation of Rational Functions," *Bull. Amer. Math. Soc.* **61** (1955), 163.
9. E. G. Belaga, "On Computing Polynomials in One Variable with Initial Conditioning of the Coefficients," *Problemy Kibernet.* **5** (1961), 7–15.