1973

A new upper bound on the complexity of derivative evaluation

H. T. Kung
Carnegie Mellon University

Follow this and additional works at: http://repository.cmu.edu/compsci
NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.
A NEW UPPER BOUND ON THE COMPLEXITY OF DERIVATIVE EVALUATION

H. T. Kung

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa.

September 1973

This research was supported in part by the National Science Foundation under Grant GJ32117 and the Office of Naval Research under Contract N00014-67-A-0314-0010, NR 044-422.
ABSTRACT

Let $T(n)$ denote the number of arithmetic operations needed to evaluate the normalized derivatives $P^{(i)}(t)/i!$, $i = 0,\ldots,n$, for an nth degree polynomial $p(t)$ over the field of complex numbers. We show

$$T(n) \leq \frac{1}{2} c n \log^2 n + \text{lower order terms}$$

where $c$ may be taken as 12.
1. INTRODUCTION

Let $k$ be the field of complex numbers, $t$ an indeterminate over $k$, and $P(t)$ an $n$th degree polynomial over $k$. Let $T(n)$ denote the number of arithmetic operations needed to evaluate the normalized derivatives $P^{(i)}(t)/i!$, $i = 0, \ldots, n$, at an arbitrary point in $k$. We are interested in upper bounds on $T(n)$.

By using the standard algorithm (iterated Horner algorithm), we can prove that

$$T(n) \leq n^2 + n, \forall n.$$ 

By using the special case of the Shaw-Traub family of algorithms with parameter $q = n+1$ [4, Section 2], referred to here for conciseness as the Shaw-Traub algorithm, we can prove that

$$T(n) \leq \frac{1}{2} n^2 + \frac{7}{2} n - 2, \forall n.$$ 

These upper bounds have been further improved for large $n$. (However, the Shaw-Traub algorithm is still the best known algorithm for small $n$.) Borodin and Munro [2, Chapter 3, Problem 5] observed that

$$(1.1) \quad T(n) \leq T_e(n) + T_i(n)$$

where $T_e(n)$ is the number of arithmetic operations needed to evaluate an $n$th degree polynomial at $n+1$ points and $T_i(n)$ is the number of arithmetic operations needed to construct an $n$th degree interpolating polynomial from $n+1$ pairs of points. (See also Kung [3].) Now, two $n$th degree polynomials can be multiplied in $(cn \log n + \text{lower order terms})$ arithmetic operations. (This can be done with the Fast Fourier Transform with $c = 12$. All logarithms in this note are to base 2.) A number of people showed independently that
\[ T_e(n) \leq O(n \log^2 n), \]
\[ T_1(n) \leq O(n \log^2 n). \]

(See the survey paper written by Borodin [1] and Kung [3].) For simplicity, in the following we assume that \( n = 2^r - 1 \) for some positive integer \( r \). Kung's algorithm gives the best previously known asymptotic constants:

\[ T_e(n) \leq \frac{3}{2} \, c \, n \log^2 n + \text{lower order terms}, \]
\[ T_1(n) \leq 2 \, c \, n \log^2 n + \text{lower order terms}. \]

Hence by (1.1),

\[ T(n) \leq \frac{7}{2} \, c \, n \log^2 n + \text{lower order terms}. \]

This is the best previously known upper bound on \( T(n) \) for large \( n \). In this note we show that

\[ T(n) \leq \frac{1}{2} \, c \, n \log^2 n + \text{lower order terms}. \]
2. MAIN RESULTS

Let \( P(t) = \sum_{i=0}^{n} a_i t^i \) and let \( t_0 \) be any point in \( k \). Suppose that we want to evaluate \( P^{(i)}(t)/i! \), \( i = 0, \ldots, n \), at \( t_0 \). It is equivalent to compute \( b_0, \ldots, b_n \) from \( a_0, \ldots, a_n \) and \( t_0 \) such that

\[
\sum_{i=0}^{n} b_i t^i = \sum_{i=0}^{n} a_0 (t+t_0)^i.
\]

Note that for \( j = 0, \ldots, r-1 \),

\[
\sum_{i=0}^{2^j+1-1} a_i (t+t_0)^i = \sum_{i=0}^{2^j-1} a_i (t+t_0)^i + (t+t_0)^{2^j-1} \sum_{i=0}^{2^{j+1}-1} a_i (t+t_0)^i.
\]

We first compute \( d_{j,i} \), \( i = 0, \ldots, 2^j \), \( j = 0, \ldots, r-1 \), such that

\[
(t+t_0)^{2^j} = \sum_{i=0}^{2^j-1} d_{j,i} t^i.
\]

It is easy to check that this can be done in \((c n \log n + \text{lower order terms})\) arithmetic operations. Then by (2.2) and by using Fast Fourier Transform for polynomial multiplication, we have

\[
T(2^{j+1}) \leq 2T(2^j) + c' j \cdot 2^j + \text{lower order terms},
\]

for \( j = 0, \ldots, r-1 \). Therefore we have shown the following

**Theorem 2.1**

\[
T(n) \leq \frac{1}{2} c n \log^2 n + \text{lower order terms},
\]

where \( n = 2^r - 1 \) for any positive integer \( r \).
Remarks.

The above algorithm is based on (2.2) which is obtained by the binary splitting of the summation in the left hand side. It is easy to see that the iterated Horner algorithm is based on the following splitting: for 
\[ j = n-1, n-2, \ldots, 0, \]

\[
\sum_{i=0}^{n-j} a_{j+i} (t+t_0)^i = a_j + (t+t_0) \sum_{i=0}^{n-j-1} a_{j+i+1} (t+t_0)^i.
\] 

Furthermore, let \( \tilde{b}_i = b_i t^i_0 \) and \( \tilde{a}_i = a_i t^i_0 \) for \( i = 0, \ldots, n \). Then by (2.1)

\[
\sum_{i=0}^{n} \tilde{b}_i t^i = \sum_{i=0}^{n} \tilde{a}_i (t+1)^i
\]

and by (2.3), for \( j = n-1, n-2, \ldots, 0, \)

\[
\sum_{i=0}^{n-j} \tilde{a}_{j+i} (t+1)^i = \tilde{a}_j + (t+1) \sum_{i=0}^{n-j-1} \tilde{a}_{j+i+1} (t+1)^i.
\]

By (2.4) and (2.5) we know that \( \tilde{b}_0, \ldots, \tilde{b}_n \) can be computed by the iterated Horner algorithm applied to the polynomial \( \sum_{i=0}^{n} \tilde{a}_i t^i \) with \( t_0=1 \). After \( \tilde{b}_1 \) has been computed \( b_1 \) is computed by \( b_1 = \tilde{b}_1 t_0^{-1} \). This algorithm for computing \( b_0, \ldots, b_n \) is exactly the Shaw-Traub algorithm.

ACKNOWLEDGMENTS

I would like to thank Professor J. F. Traub for his comments on this note.
References


