Size Efficient Parallel Algebraic Circuits for Partial Derivatives*

Erich Kaltofen[1] *and Michael F. Singer*[2]

[1]Department of Computer Science, Rensselaer Polytechnic Institute,
Troy, NY 12180-3590; kaltofen@cs.rpi.edu

[2]Department of Mathematics, North Carolina State University,
Raleigh, NC 27695-8205; singer@matagh.ncsu.edu

Abstract. Given an algebraic circuit or straight-line program of depth d that computes multivariate rational function in l arithmetic operations (additions, multiplications, and divisions), we construct circuits that compute

(1) the first k partial derivatives in a single variable with depth $O(\log(k)(d+\log(k)))$ using $O(k\log(k)\log(\log\ k)\,d)$ arithmetic operations;

(2) all first partial derivatives in depth $O(d)$ using no more than $4l$ arithmetic operations.

Our first result is based on Taylor series expansion and essentially parallelizes the Leibniz formula. Our second result parallelizes a construction by Baur and Strassen. A crucial ingredient to the parallel solution is the fact that bounded fan-in computation graphs can be transformed to those where the fan-out is bounded as well while increasing the depth by no more than a constant factor.

1. Introduction

Transformations on structured computations of rational multivariate functions are a commonplace tool for sequential and parallel algebraic algorithm design and complexity analysis. One of the first results is Brent's (1974) transformation of an expression tree to one of logarithmic depth. That transformation generically produces parallel code for expression evaluation and has since then been studied extensively (Miller and Reif 1989), (Kosaraju and Delcher 1988), and (Cole and Vishkin 1988). Surprisingly, the tree contraction technique introduced by Miller and Reif can be generalized to reduce the depth of an algebraic computation graph or a straight-line program (introduced by Strassen (1972) as a model for algebraic computation) to poly-logarithmic in size and degree (Valiant et al 1984), (Miller et al 1986), and (Kaltofen 1988). Another transformation, inspired by the problem of reducing the inversion of a matrix to the computation of its determinant, was found by Baur and Strassen (1983) who show that if one is given a multivariate rational function by a straight-line program, then one can simultaneously compute all first order partial derivatives with only a constant factor growth in the length of the computation.

*This material is based on work supported in part by the National Science Foundation under Grant No. CCR-87-05363 and under Grant No. CDA-88-05910 (first author) and by the National Science Foundation under Grant No. DMS-8803109 (second author). This paper appears in the Proceedings of the IV International Conference on Computer Algebra in Physical Research, D. V. Shirkov, V. A. Rostovtsev, and V. P. Gerdt (eds.), World Scientific, Singapore, pp. 133-145 (1991).

The motivation of this article is to investigate how one might apply derivatives to a straight-line computation of small depth, such as the ones produced by the parallelization methods mentioned before, without substantially increasing the depth while keeping the size minimal.

Baur and Strassen's (loc. cit.) clever construction does not increase the size by more than a constant factor, but it can produce a straight-line program of depth as large as the number of instructions in the straight-line program. In this paper we give a modified transformation that increases the length of the program for all first order partial derivatives of the final function computed by a factor of no more than 4 compared to the original program. Much more significantly our construction retains the depth of the original program within a constant factor. Our construction makes use of another computation graph transformation by Hoover, Klawe, and Pippenger (1984) who show how a bounded fan-in, unbounded fan-out circuit can be transformed into one with bounded fan-out as well by increasing both size and depth by a constant factor only. Their construction essentially adds specially balanced trees of duplication nodes behind nodes of high fan-out. One key ingredient in our transformation is the observation that the circuit which accumulates all partial derivatives from a coarse view-point is a mirror image of the circuit for computing the function itself. Therefore, data flows into dual nodes of high fan-in whose preimage counterparts have high fan out. These nodes only need to perform additions, so we can do these additions in the nodes that in Hoover's et al. construction were just duplication nodes, and therefore do not increase the length at all.

Our second result deals with computing the k-th order derivative in a single variable. Our earlier solution (Kaltofen 1987) made use of the Leibniz formula, which not only increases the length of the program by a factor of $O(k^2)$, but also determines the i-th derivative from the $(i-1)$-st one and therefore multiplies the depth by a factor of $O(k \log(k))$. Our new solution is based on the simple observation that higher derivatives can be read off from the coefficients of the power series expansion in the variable to differentiated by (see (Aho et al. 1975)). In our case, we need to compute Taylor series expansions in a single variable of multivariate rational functions given by straight-line programs. This problem, however, has been studied thoroughly (Strassen 1973) and (Kaltofen 1988). Indeed, by pushing the divisions back to a final single one, we can compute all derivatives in a single variable up to order k by a program whose length has increased by a factor of order $O(k \log(k) \log(\log k))$ and whose depth by roughly a factor of order $O(\log(k))$.

A straight-line program of length l is a sequence of assignment

$$v_i \leftarrow v_i' \circ_i v_i'', \quad 1 \le i \le l,$$

where v_i is a new program variable,

$$v_i', v_i'' \in \mathsf{K} \cup \bigcup_{j=1}^{n} \{x_j\} \cup \bigcup_{k=1}^{i-1} \{v_k\},$$

and the operation $\circ_i$ is either $+$, $-$, $\times$, or $\div$. In this paper we assume that K is an arbitrary field. Every v_i therefore computes a rational function in $\mathsf{K}(x_1, \ldots, x_n)$ and it is not permitted to divide by a v_i'' that is identically 0. All our transformation necessarily must preserve this condition. The depth of each variable is defined recursively as

$$\mathrm{depth}(v_i) = 1 + \max\{\mathrm{depth}(v_i'), \mathrm{depth}(v_i'')\},$$

where depth of operands $v_i', v_i'' \in \mathsf{K} \cup \{x_1, \ldots, x_n\}$ is defined to be zero. The depth of a program is the maximum of the depths of the variables. The depth is a measure of the time it takes to evaluate a straight-line program in parallel. Given l processors, each of which evaluates a variable as soon as the values of their operands are known, the process of evaluating the entire program takes the depth of the straight-line program many rounds.

By $\mathrm{M}(n)$ we denote the number of arithmetic operations it takes to multiply two univariate polynomials over K of degree no more than n. By the result in (Cantor and Kaltofen 1987) there is a straight-line program without divisions of length $\mathrm{M}(n) = O(n \log(n) \log(\log n))$ and depth $O(\log(n))$ that can compute the coefficients of the product polynomial for this problem.

2. Differential Fields

The results in this paper deal with partial derivatives on multivariate rational functions. If we have a subfield of the complex numbers as the coefficient field, partial derivatives can be defined in an analytic setting by considering the functions as multivariate analytic complex functions. In particular, we obtain the Laurent series representation of such functions, as well as rules on differentiation such as the chain rule. However, we wish to include coefficient fields of finite characteristic in our results as well as use derivatives from a purely algebraic point of view with no analytic interpretation. Therefore, we will appeal to the algebraic theory of differential fields (Kaplansky 1957). Since we only need the very basics of that theory and since we wish to make this article somewhat self-contained, we discuss the necessary definitions and lemmas in this section.

Let R be a commutative ring (with unity). A *derivative* ∂ on R is a map from R into itself such that

$$\forall r, s \in R\colon \partial(r+s) = \partial(r) + \partial(s), \quad \partial(rs) = \partial(r)s + r\partial(s).$$

A *differential ring* is a ring together with a family of derivatives $\{\partial_i\}_{i\in I}$. It is surprising that most of the properties of function fields with derivatives can be modelled by this definition. We shall state the ones we need.

Let 1 be the unit in R. Then

$$\partial(1) = \partial(1^2) = 2 \cdot 1 \cdot \partial(1) = 0.$$

We define as the *ring of constants* of R

$$C_R = \{c \mid c \in R \text{ and } \forall i \in I\colon \partial_i(c) = 0\},$$

which thus is a subring of R.

Let $r \in R$ posses a multiplicative inverse r^{-1}. Then

$$0 = \partial(rr^{-1}) = \partial(r)r^{-1} + r\partial(r^{-1}), \quad \text{hence } \partial(r^{-1}) = -\partial(r)r^{-2}.$$

Therefore, $\partial(r^n) = nr^{n-1}\partial(r)$ for all such r and all $n \in \mathbb{Z}$.

Lemma 2.1. *Let the differential ring be an integral domain. Then any derivative ∂ extends uniquely to the quotient field of R,* $\mathrm{QF}(R)$.

Proof. The embedding of R into $\mathrm{QF}(R)$ requires us to define $\partial([r/1]) = [\partial(r), 1]$, where $r \in R$ and $[r/s]$ denotes the equivalence class for the pair r/s. Since for $r \neq 0$ the class $[1/r]$ is the multiplicative inverse of $[r/1]$, we must have $\partial([1/r]) = -[\partial(r)/1]\,[1/r^2]$. Hence we must define

$$\forall r, s \in R, s \neq 0\colon \partial([r/s]) = [(\partial(r)s - r\partial(s))/s^2].$$

One now can show easily that this definition is independent of the choice of representative of $[r/s]$, and in fact constitutes a derivative on $\mathrm{QF}(R)$. $\boxtimes$

We now extend derivatives to series rings constructed from R. The first would be the ring of polynomials over R, $R[x]$. It turns out that one can give $\partial(x)$ any value in $R[x]$ and then uniquely extend a derivative on R to one on $R[x]$. If R is an integral domain, the same holds true for $R(x)$,

the quotient field of $R[x]$. In our applications, R will always be a field K, possibly of positive characteristic.

It is useful to embed $\mathsf{K}(x)$ into the field of *extended formal power series*,

$$\mathsf{K}(\!(x)\!) := \bigcup_{k\geq 0} \frac{1}{x^k}\mathsf{K}[[x]],$$

where $\mathsf{K}[[x]]$ is the domain of formal power series over K. The embedding ι requires the identification of polynomial inverses with their power series, that is

$$\iota\left(\frac{1}{a_0 + a_1 x + \cdots + a_d x^d}\right) = b_0 + b_1 x + b_2 x^2 + \cdots + b_i x^i + \cdots, \quad a_0 \neq 0, b_0 = \frac{1}{a_0},$$

where with $a_l = 0$ for $l > d$ we have

$$\forall i \geq 1\colon b_i = \frac{1}{a_0}\left(-b_0 a_i - b_1 a_{i-1} - \cdots - b_{i-1} a_1\right).$$

We first discuss how to computed derivatives in $\mathsf{K}(\!(x)\!)$.

Lemma 2.2. *For any $g \in \mathsf{K}(\!(x)\!)$*

$$\partial\left(\sum_{i=-k}^{+\infty} a_i x^i\right) = -\frac{k a_{-k} g}{x^{k+1}} + \sum_{i=-k}^{+\infty}\Big(\partial(a_i) + (i+1)a_{i+1}g\Big)x^i, \quad a_i \in \mathsf{K}, k \geq 0,$$

defines a derivative on $\mathsf{K}(\!(x)\!)$ with $\partial(x) = g$. ⊠

Note that this lemma allows us the transfer of a derivative ∂' on $\mathsf{K}(x)$ to a derivative ∂ on $\mathsf{K}(\!(x)\!)$ by defining $\partial(x) = \iota(\partial'(x))$. These derivatives agree on K.

An important special case is that where $\partial(x) = 1$ and K is a subfield of the constant field, i.e., $\partial(a) = 0$ for all $a \in \mathsf{K}$. This is because for $\mathsf{K} = \mathbb{C}$, the field of complex numbers, this derivative agrees with the analytically defined one. In fact, one can by purely algebraic means establish the construction of multivariate MacLaurin series expansions.

Lemma 2.3. *Let $\mathsf{K}(x_1, \ldots, x_n)$ be endowed with a family of standard partial derivatives ∂_{x_1}, $\ldots, \partial_{x_n}$ such that*

$$\forall i\colon \partial_{x_i}(\mathsf{K}) = \{0\},\ \partial_{x_i}(x_i) = 1,\ \partial_{x_i}(x_j) = 0 \text{ for } j \neq i.$$

Consider the multivariate power series expansion for f/g where $f, g \in \mathsf{K}[x_1, \ldots, x_n]$, $g(0, \ldots, 0) \neq 0$,

$$\frac{f(x_1, \ldots, x_n)}{g(x_1, \ldots, x_n)} = \sum_{i_1 \geq 0, \ldots, i_n \geq 0} b_{i_1, \ldots, i_n} x_1^{i_1} \cdots x_n^{i_n}.$$

Then

$$\forall i_1 \geq 0, \ldots, i_n \geq 0\colon \left(\partial_{x_1}^{i_1}\partial_{x_2}^{i_2}\cdots\partial_{x_n}^{i_n}\frac{f}{g}\right)(0, \ldots, 0) = (i_1!)\cdots(i_n!)\, b_{i_1, \ldots, i_n},$$

where $\partial_{x_j}^i$ is applying ∂_{x_j} a sequence of i times.

Proof. First we note that the family of derivatives is also valid in $\mathsf{K}(\!(x_1, \ldots, x_n)\!)$. Hence by lemma 2.2 we have

$$\partial_{x_1}^{i_1}\partial_{x_2}^{i_2}\cdots\partial_{x_n}^{i_n}\left(\sum_{i_1 \geq 0, \ldots, i_n \geq 0} b_{i_1, \ldots, i_n} x_1^{i_1}\cdots x_n^{i_n}\right)$$
$$= (i_1!)\cdots(i_n!) b_{i_1, \ldots, i_n} + \sum_{l_1 > i_1, \ldots, l_n > i_n} \underline{l_1^{i_1}}\cdots\underline{l_n^{i_n}}\, b_{l_1, \ldots, l_n} x_1^{l_1 - i_1}\cdots x_n^{l_n - i_n},$$

where $\underline{l^i}$ denotes $l(l-1)\cdots(l-i+1)$. Plugging in 0 for x_j proves the lemma. ⊠

We finally prove the chain rule for rational functions.

Lemma 2.4. *Let the derivatives $\partial_{x_1}, \ldots, \partial_{x_n}$ on $\mathsf{K}[x_1, \ldots, x_n]$ be as in lemma 2.3. Similary, define the standard derivatives $\partial_{y_1}, \ldots, \partial_{y_m}$ on $\mathsf{K}[y_1, \ldots, y_m]$. Let $f \in \mathsf{K}(y_1, \ldots, y_m)$ and let $g_1, \ldots, g_m \in \mathsf{K}(x_1, \ldots, x_n)$. Assume that the denominator of f does not become zero in $\mathsf{K}(x_1, \ldots, x_n)$ by setting y_j to $g_j(x_1, \ldots, x_n)$ for all $1 \leq j \leq m$. Then*

$$\partial_{x_i}\Big(f(g_1, \ldots, g_m)\Big) = \sum_{j=1}^{m} (\partial_{y_j} f)(g_1, \ldots, g_m)\, \partial_{x_i}(g_j)$$

Note that by the quotient rule in the proof of lemma 2.1 no zero division occurs in any of the $\partial_{y_j}(f)(g_1, \ldots, g_m)$.

Proof. For $h \in \mathsf{K}(x_1, \ldots, x_n)$ consider

$$\bar{h}(z_1, \ldots, z_n) = h(X_1 + z_1, \ldots, X_n + z_n) \in \Big(\mathsf{K}(X_1, \ldots, X_n)\Big)(z_1, \ldots, z_n).$$

We first note that

$$\big(\partial_{z_i} \bar{h}\big)(0, \ldots, 0) = (\partial_{x_i} h)(X_1, \ldots, X_n).$$

Hence by lemma 2.3 we have

$$h(x_1 + z_1, \ldots, x_n + z_n) = h(x_1, \ldots, x_n) + \sum_{i=1}^{n} (\partial_{x_i} h) z_i + \cdots.$$

We now apply this expansion to both f and all g_j and obtain

$$f(y_1 + u_1, \ldots, y_m + u_m) = f(y_1, \ldots, y_m) + \sum_{j=1}^{m} \big(\partial_{y_j} f\big)\, u_j + \cdots$$

and for $j = 1, \ldots, n$,

$$g_j(x_1 + z_1, \ldots, x_n + z_n) = g_j(x_1, \ldots, x_n) + \sum_{i=1}^{n} (\partial_{x_i} g_j)\, z_i + \cdots.$$

Now we substitute g_j for y_j and $\sum_{i=1}^{n} (\partial_{x_i} g_j)\, z_i + \cdots$ for u_j in the expansion for f. This substitution is valid because for $z_1 = \cdots = z_n = 0$ no zero division occurs by assumption. Considering $h = f(g_1, \ldots, g_m) \in \mathsf{K}[x_1, \ldots, x_n]$ we need to collect the coefficient of z_i in

$$h(x_1 + z_1, \ldots, x_n + z_1) = f(g_1(x_1 + z_1, \ldots, x_n + z_n), \ldots, g_m(x_1 + z_1, \ldots, x_n + z_n))$$

to obtain $\partial_{x_i} h$. By the above substitution this coefficient is

$$\sum_{j=1}^{m} (\partial_{y_j} f)(g_1, \ldots, g_m)\, \partial_{x_i}(g_j),$$

which proves the lemma. $\boxtimes$

3. Higher Order Partial Derivatives in a Single Variable

Given $f \in \mathsf{K}(x_1, \ldots, x_n)$ by a straight-line program P, at issue is to construct a straight-line program Q that computes $\partial_{x_1}^k(f)$. Our first solution to this problem used the Leibniz formula for the higher order product rule (Kaltofen 1987). We briefly explain the approach. Let $w \leftarrow u \circ v$ be an instruction in P. The new program Q introduces variables $u^{(j)}$, $v^{(j)}$, $w^{(j)}$, $0 \le j \le k$, such that $w^{(j)}$ computes $\partial_{x_1}^j(w)$. Clearly, we then must have $w^{(0)} \leftarrow u^{(0)} \circ v^{(0)}$ and for $j \ge 1$ and

$$\begin{aligned}
\text{for } \circ = +\text{: } & w^{(j)} = u^{(j)} + v^{(j)}, \\
\text{for } \circ = \times\text{: } & w^{(j)} = \sum_{m=0}^{j} \binom{j}{m} u^{(j-m)} v^{(m)}, \\
\text{and for } \circ = \div\text{: } & w^{(j)} = \left(u^{(j)} - \sum_{m=1}^{j} \binom{j}{m} w^{(j-m)} v^{(m)} \right) \Big/ v^{(0)}.
\end{aligned}$$

Therefore, $w^{(j)}$ can be computed from

$$u^{(0)}, \ldots, u^{(j)}, v^{(0)}, \ldots, v^{(j)}, w^{(0)}, \ldots, w^{(j-1)}$$

in $O(j)$ assignments, compiling to a total of $O(k^2)$ assignments for $w^{(k)}$.

For $k = 1$, this approach is the best we know. In particular, the quotient rule given above only requires 3 additional instructions, leading to a program Q of length no more than $4l$, where l is the length of the original program P. For large k, this approach is, however, not very efficient. Not only does it experience quadratic growth in size, but also leads a program Q of depth $O(k \log(k)\, d)$, where d is the depth of the original program P. Our new approach is based on reading the derivatives off a Taylor series expansion (see lemma 2.3) and reduces both size and depth of Q. We have the following theorem.

Theorem 3.1. *Let $f \in \mathsf{K}(x_1, \ldots, x_n)$ be given by a straight-line program of length l and depth d. Then f and all derivatives $\partial_{x_1}(f), \partial_{x_1}^2(f), \ldots, \partial_{x_1}^k(f)$, $k \ge 1$, can be computed by a straight-line program Q of length $O(\mathrm{M}(k)\, l)$ and depth $O\big(\log(k)(d + \log(k))\big)$.*

Proof. Consider the straight-line program P_y that computes $f(x_1 + y, x_2, \ldots, x_n) \in \mathsf{K}(x_1, \ldots, x_n, y)$. By the algorithm Taylor Series Coefficients (Kaltofen 1988, Section 7) we can transform P_y into a straight-line program Q that computes the coefficients $b_j(x_1, \ldots, x_n) \in \mathsf{K}(x_1, \ldots, x_n)$, $0 \le j \le k$, of the expansion

$$f(x_1 + y, x_2, \ldots, x_n) = \sum_{j=0}^{\infty} b_j(x_1, \ldots, x_n) y^j.$$

By the argument given in the proof of lemma 2.4 we then get

$$\partial_{x_1}^j(f) = j!\, b_j(x_1, \ldots, x_n).$$

The program Q essentially keeps for each program variable w the truncated power series coefficients in y in additional variables. Clearly, if we have the assignment $w \leftarrow u \circ v$, then we can compute the coefficients of w from those of u and v. More precisely, let

$$u = \sum_{j=0}^{k} u^{(j)} y^j + O(y^{k+1}),\ v = \sum_{j=0}^{k} v^{(j)} y^j + O(y^{k+1}),\ \text{and } w = \sum_{j=0}^{k} w^{(j)} y^j + O(y^{k+1}).$$

We will have $w^{(0)} \leftarrow u^{(0)} \circ v^{(0)}$ and for $j \geq 1$ and

$$\text{for } \circ = +\colon\ w^{(j)} = u^{(j)} + v^{(j)},$$

$$\text{and for } \circ = \times\colon\ w^{(j)} = \sum_{m=0}^{j} u^{(j-m)} v^{(m)}.$$

For $\circ = \div$ we are reqired to perform power series inversion

$$\left(\sum_{j=0}^{k} v^{(j)} y^j + O(y^{k+1}) \right)^{-1},$$

which in this case can be done since $v^{(0)} = v \neq 0$, because the original program P divides by v. By polynomial multiplication both the case $\circ = \times$ and the case $\circ = \div$ can be done in $O(\mathrm{M}(k))$ instructions (Sieveking 1972), (Kung 1974). However, not for any field K can $O(\log(k))$ depth for power series inversion be easily accomplished while asymptotically retaining the instruction count.

We finally show how all but a division in the last instruction can be replaced by additions and multiplications. The idea is already discussed in (Strassen 1973) and introduces for each variable w in P two variables $w^{(1)}$ and $w^{(2)}$ such that both will compute polynomials in $\mathsf{K}[x_1, \ldots, x_n]$ and $w = w^{(1)}/w^{(2)}$. Given an instruction $w \leftarrow u \circ v$ we now can compute, e.g.,

$$\text{for } \circ = +\colon\ w^{(1)} = u^{(1)} v^{(2)} + v^{(1)} u^{(2)},\ w^{(2)} = u^{(2)} v^{(2)}.$$

This transformation does not asymptotically increase either length of depth. Now we apply our transformation to the new program with one division. By using Newton iteration to resolve the final truncated power series inversion we will then add $O(\mathrm{M}(k))$ length and $O((\log k)^2)$ depth to the program Q, whose preceding $O(\mathrm{M}(k)\, l)$ instructions perform $O(l)$ truncated power series additions and multiplications at a depth of $O(\log(k)\, d)$. $\boxtimes$

4. Multiple First Order Derivatives

Given be a straight-line program of length l and depth d that computes $f \in \mathsf{K}(x_1, \ldots, x_n)$. In the previous section we developed an algorithm that allows to compute all

$$\partial^k_{x_1}(f), \partial^k_{x_2}(f), \ldots, \partial^k_{x_n}(f)$$

by a straight-line program of length $O\big(nl\,\mathrm{M}(k)\big)$ and depth $O\big(\log(k)\,(d + \log(k)\big)$. In this section we treat the case $k = 1$ more closely. We will show that one can then remove the factor n from the length while retaining depth $O(d)$. Our construction is based on the result by Baur and Strassen (1983) who establish a transformation to compute all first order partial derivatives with an increase in length by only a scalar multiple. However, their transformation is not depth preserving. By analyzing their construction more closely, we can prove the following theorem.

Theorem 4.1. *Let $f \in \mathsf{K}(x_1, \ldots, x_n)$ be computed by a straight-line program P of length l and depth d. Then f and all derivatives $\partial_{x_1}(f), \partial_{x_2}(f), \ldots, \partial_{x_n}(f)$ can be computed by a straight-line program Q of length no more than $4\,l$ and depth $O(d\,\log(t))$, where t is the maximum number of times any variable is used as an operand in P, i.e., the maximum "fan-out" in the program P.*

Proof. Let $v_i \leftarrow v_{I_1(i)} \circ_i v_{I_2(i)}$, $n+1 \leq i \leq n+l$, be the $(i-n)$-th instruction in the program P. Here the function I_1 retrieves the index of the left operand of right-hand side expression, and the function I_2 the right operand. We set $v_i := x_i$ for $1 \leq i \leq n$, hence have a range for the operand indices of $1 \leq I_1(i), I_2(i) < i$. If the left or right operands are scalars, no such indexing will be needed. For $i > n$ the symbol v_i stands, strictly speaking, for a program variable in P, or a node in the computation DAG for f. However, we also use it to identify with it the rational function in $\mathsf{K}(x_1, \ldots, x_n)$ that is computed in this variable. Baur and Strassen's construction proceeds by viewing f as a sequence of functions

$$g_i(y_1, \ldots, y_i) \in \mathsf{K}(y_1, \ldots, y_i), \quad n+l \geq i \geq n.$$

We will have

$$g_i(v_1, \ldots, v_i) = f(x_1, \ldots, x_n) \text{ for all } n \leq i \leq n+l.$$

The interpretation of g_i is the function that gets computed in v_l if we cut off the instructions for $v_{n+1}, \ldots, v_i$ in the program P and set $v_1, \ldots, v_i$ to the variables $y_1, \ldots, y_i$ whenever the occur in the truncated program. In particular, we want to have

$$g_n(x_1, \ldots, x_n) = f(x_1, \ldots, x_n).$$

Now let

$$h_i(y_{I_1(i)}, y_{I_2(i)}) = y_{I_1(i)} \circ_i y_{I_2(i)} \in \mathsf{K}(y_1, \ldots, y_{i-1}), \quad n+l \geq i \geq n+1,$$

denote the rational function that gets formally computed by the $(i-n)$-th instruction in P. The functions g_i are therefore inductively defined to be related by

$$g_{i-1}(y_1, \ldots, y_{i-1}) := g_i\Big(y_1, \ldots, y_{i-1}, h_i(y_{I_1(i)}, y_{I_2(i)})\Big), \tag{$*$}$$

where initially $g_{n+l}(y_1, \ldots, y_{n+l}) := y_{n+l}$. The goal is to compute

$$\partial_{y_1}(g_n), \partial_{y_2}(g_n), \ldots, \partial_{y_n}(g_n).$$

This is done by using the inductive definition of g_i and the chain rule (lemma 2.4). We first have

$$\partial_{y_j}(g_{n+l}) = 0 \text{ for all } 1 \leq j \leq n+l-1, \quad \partial_{y_{n+l}}(g_{n+l}) = 1.$$

Now lets assume that at level $n+l-i$ we already compute the derivatives

$$\partial_{y_1}(g_i), \partial_{y_2}(g_i), \ldots, \partial_{y_i}(g_i).$$

From $(*)$ we get by the chain rule for $j_1 := I_1(i)$ and $j_2 := I_2(i)$ that

$$(\partial_{y_k} g_{i-1})(y_1, \ldots, y_{i-1}) = (\partial_{y_k} g_i)(y_1, \ldots, y_{i-1}, h_i(y_{j_1}, y_{j_2})) \text{ for } 1 \leq k \leq i-1, k \neq j_1, k \neq j_2$$

and

$$\begin{aligned}(\partial_{y_j} g_{i-1})(y_1, \ldots, y_{i-1}) = {} & (\partial_{y_j} g_i)(y_1, \ldots, y_{i-1}, h_i(y_{j_1}, y_{j_2})) \\ & + (\partial_{y_i} g_i)(y_1, \ldots, y_{i-1}, h_i(y_{j_1}, y_{j_2}))(\partial_{y_j} h_i)(y_{j_1}, y_{j_2})\end{aligned}$$

for $j = j_1$ or $j = j_2$.

The dynamics of these rules are displayed in figure 1. The substitution $h_i(y_{j_1}, y_{j_2})$ for y_i is accomplished by connecting the corresponding node to the nodes for y_{j_1} and y_{j_2} and performing the operation $\circ_i$ in the node. Then the derivatives $\partial_{y_j}(g_{i-1})$ are computed from those of $\partial_{y_j}(g_i)$ plus a value derived from

$$\overline{\partial_{y_i}(g_i)} = (\partial_{y_i} g_i)(y_1, \ldots, y_{i-1}, h_i(y_{j_1}, y_{j_2}))$$

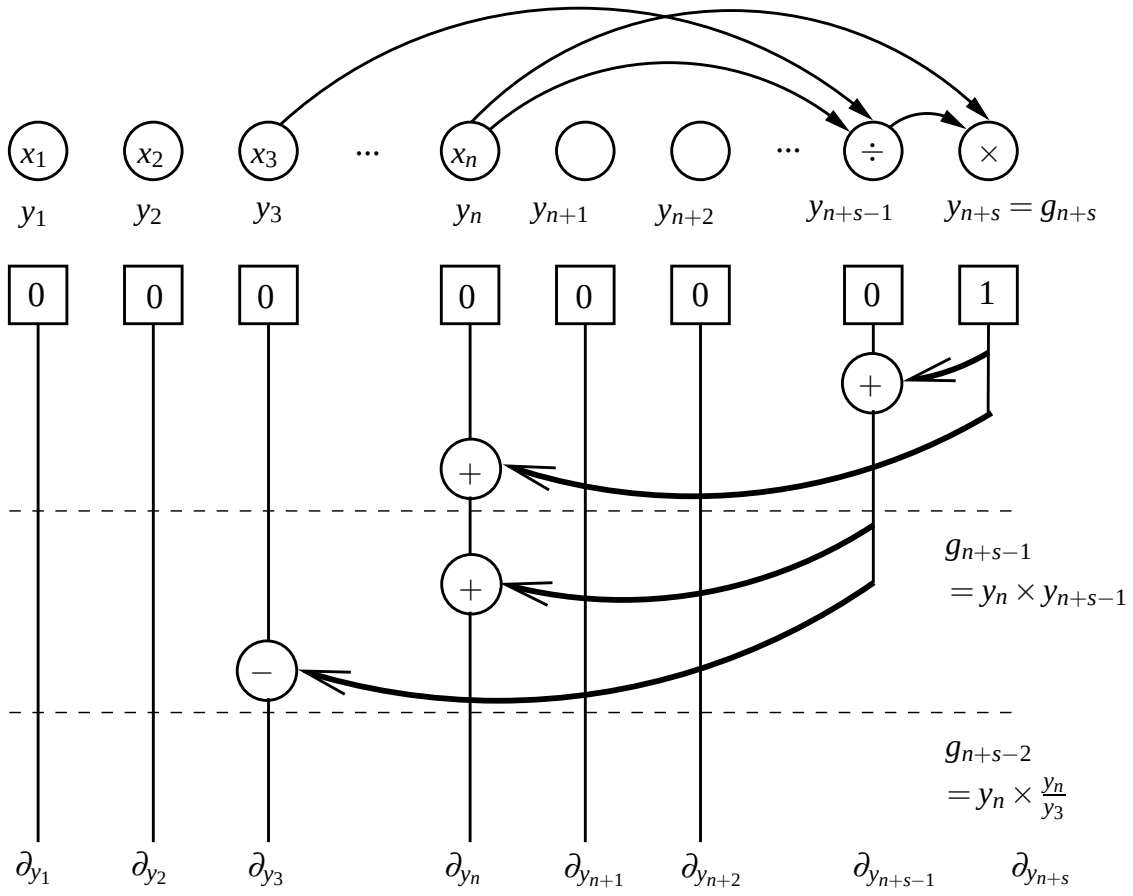

Figure 1: Coarse View of the Baur and Strassen Construction

and the derivatives of $h_i(y_{j_1}, y_{j_2})$. The latter are solely dependent on the nodes corresponding to the variables y_{j_1} and y_{j_2} and require constant work. In figure 1, this is indicated by a thick connection from the line for ∂_{y_i} to ∂_{y_j}. Let us for a moment consider the operation $\circ_i$ with the most costly work, namely division. For

$$h_i(y_{j_1}, y_{j_2}) = \frac{y_{j_1}}{y_{j_2}} \text{ we have } \partial_{y_{j_1}}(h_i) = \frac{1}{y_{j_2}} \text{ and } \partial_{y_{j_2}}(h_i) = -\frac{y_{j_1}}{y_{j_2}^2}.$$

The strategy is to divide $\overline{\partial_{y_i}(g_i)}$ by y_{j_2}, add that into the $\partial_{y_{j_1}}$ line, or multiply it with y_{j_1}/y_{j_2}, the value computed in the node for y_i, and then subtract that from the ∂_{y_2} line. In other words, if $\circ_i = \div$ one needs 4 additional operations to go to the next level. There is one more issue that needs to be settled in the division case. Later substitutions for y_{j_2} must not cause a division by

a function that is identical zero. This is true because the circuit computing the derivatives will only divide by quantities that the original program P divides by.

We now discuss how to accomplish the given length and depth measures. For each v_i in P, $n+1 \leq i \leq n+l$, we will introduce at most 5 instructions in our new program Q, one from the original program and at most 4 more to eliminate y_i. This leads to an upper bound of $5l$ for the length of Q, but l of these instructions either add the initial $\partial_{y_j}(g_{n+l}) = 0$ into $\overline{\partial_{y_i}(g_i)}\,(\partial_{y_j} h_i)(y_{j_1}, y_{j_2})$ or multiply $\partial_{y_{n+l}}(g_{n+l}) = 1$ into $\partial_{y_j}(h_{n+l})(y_{j_1}, y_{j_2})$. Since we have any instruction v_i participate in the computation of v_{n+l}, there are at least l trivial instructions that can be eliminated from such a Q. Note that if we only have subtractions on a line for ∂_{y_j} we pass the minus sign along to the level for the derivatives of g_{j-1}. On each line for ∂_{y_j}, $1 \leq j \leq n$, we might then have to negate the final result, costing us an additional instruction. However, we did not account for the savings at the starting level of those lines, and therefore we do not need more than $4\,l$ instructions overall. Lastly, we discuss how to accomplish the stated depth. First we observe that if we were to treat the lines in figure 1 on which we accumulate the ∂_{y_j} as single nodes, and if we were to treat the connections from ∂_{y_i} to $\partial_{y_{j_1}}$ and to $\partial_{y_{j_2}}$ as single edges, then the circuit to compute the derivatives would be a mirror image of the original circuit for f. Therefore, the depth of this view of the part of Q that implements the chain rules and which has "superedges" and "supernodes" is d. Now, on each "superedge" we only have a constant delay. Let t_j be the fan-out for v_j in P. Then in each supernode corresponding to the line for ∂_{y_j} we have exactly $t_j - 1$ addition and subtraction nodes. Separating the lines that get added from the ones that get subtracted, we can build with $t_j - 1$ nodes a tree that performs the same computation but which has $O(\log(t))$ depth (see figure 2). Hence the entire depth of Q can be made $O(\log(t)d)$. ⊠

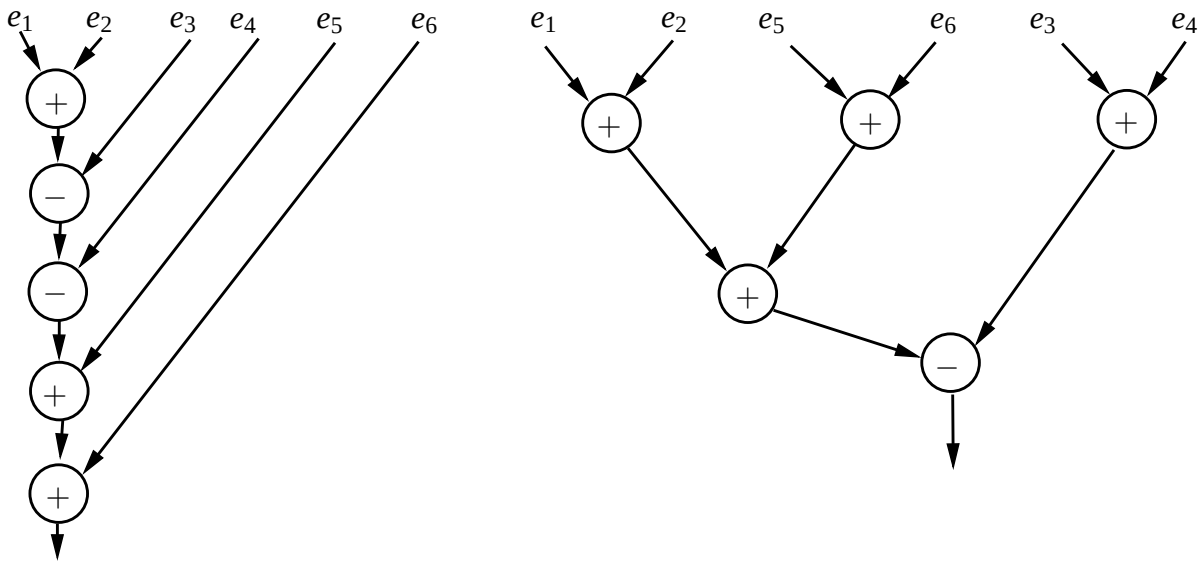

Figure 2: Balancing an Accumulation Tree

One can, at no expense in terms of length, reduce the depth of Q to $O(d)$. This follows from a construction by Hoover, Klawe, and Pippenger (1984) who show that for bounded fan-in circuits one can introduce duplication nodes behind the nodes of high fan out in such a way that both size and depth are preserved within a constant factor. In our case the situation is so special

that we can accomplish this introduction of duplication nodes without increasing the size of the circuit.

Theorem 4.2. *Let f, P, l, and d be as in theorem 4.1. Then f and all derivatives $\partial_{x_1}(f)$, $\partial_{x_2}(f), \ldots, \partial_{x_n}(f)$ can be computed by a straight-line program Q of length no more than $4\,l$ and depth $O(d)$.*

Proof. We apply the transformation of Hoover et al. (loc. cit.) in an implicit way to the circuit constructed in the proof of theorem 4.1. Consider the lower part of Q that is a mirror image of P (see figure 1). Furthermore, assume that the subtrees in Q which perform additions on the ∂_{y_j} lines are again contracted to "supernodes". We suppose that subtractions are already separated out, and subtraction nodes on those lines remain untouched. Nonetheless, the depth of this abstraction of the circuit is still $O(d)$, the extra factor of $\log(t)$ in theorem 4.1 coming from the delay in the supernodes. Now we apply the construction by Hoover et al. to this view of the lower part of Q, reversing the flow of information. That construction will insert behind the nodes of high fan-out a binary tree of duplication nodes whose root is that node and whose leaves are the targets of the arcs leaving that node. Hoover et al. then show that if one optimizes the structure of that tree with respect to the distance of the target nodes to the output node in such a way that target nodes from which there are long paths to the output node are close to the root, one can overall retain depth $O(d)$. Once such duplication trees are in place behind the supernodes, all we have to do is reverse the flow of information and perform additions in both supernodes and duplication nodes. ⊠

5. Conclusion

We have given size and depth efficient transformations from straight-line programs to such programs that compute either a higher order partial derivative in a single variable or all first order partial derivatives. These transformations are easily performed in polynomial time, but can also be executed in parallel themselves (for the tree optimization problem needed in the Hoover et al. fan-out reduction refer to (Teng 1987)). There are known computational limits to the derivation problem. The first discovered fact is that the formula size can grow exponentially if one allows exponentiation in the formula (Caviness and Epstein 1978). Even without this extra operator, the growth can be related to the $\#\mathcal{P}$-complete problem of evaluating the permanent. The example is the function given by the formula

$$f(y_1, \ldots, y_n, x_{1,1}, \ldots, x_{n,n}) = \prod_{i=1}^{n} \left(\sum_{j=1}^{n} y_j x_{i,j} \right)$$

of size $O(n^2)$, one of whose mixed iterated derivative is

$$\partial_{y_1} \cdots \partial_{y_n}(f) = \text{Permanent}(\left[x_{i,j}\right]_{1 \le i,j \le n}),$$

the generic permanent of the matrix in the $x_{i,j}$ (Valiant 1982). This means that if one had a solution with polynomial growth to transforming a straight-line program to one that computes iterated derivatives in different variables, then one would have a polynomial-time program for counting the number of satisfying assignments in a Boolean formula in random polynomial time, in particular one would have shown that $\mathcal{RP} = \#\mathcal{P}$. Applying Kronecker substitution to f one also can show that the k-th derivative in a single variable most likely cannot be computed in time polynomial in $\log(k)$ (Kaltofen 1987).

Following Baur and Strassen (1984) our result on simultaneously computing all first order partial derivatives was primarily motivated by the question on how to solve linear systems in

parallel using an optimal number of processors. Given be a straight-line program† with $O(n^3)$ instructions of depth $O((\log n)^2)$ that computes the determinant

$$\Delta(x_{1,1}, \ldots, x_{n,n}) = \det(A), \quad A = \left[x_{i,j}\right]_{1 \le i,j \le n}.$$

Then the inverse of A can also be computed by a program of such asymptotic length and depth, namely

$$A^{-1} = \left[(-1)^{i+j} \frac{\partial_{x_{j,i}}(\Delta)}{\Delta}\right]_{1 \le i,j \le n}.$$

Morgenstern (1985) is one of the first to observe that the Baur and Strassen result generalizes to straight-line programs that also allow instructions like $v \leftarrow \exp(u)$, $v \leftarrow \log(u)$, $v \leftarrow \sqrt{u}$, etc. At least for fields K of characteristic zero, all our results can also be generalized to such an extended model of straight-line programs.

Acknowledgement: Part of the results were conceived in June 1988, while the authors were visiting the Research Institute for Symbolic Computation in Linz, Austria.

6. Literature Cited

Aho, A. V., Steiglitz, K., and Ullman, J. D., "Evaluating polynomials at fixed sets of points," *SIAM J. Comp.* **4**, pp. 533–539 (1975).

Baur, W. and Strassen, V., "The complexity of partial derivatives," *Theoretical Comp. Sci.* **22**, pp. 317–330 (1983).

Brent, R. P., "The parallel evaluation of general arithmetic expressions," *J. ACM* **21**, pp. 201–208 (1974).

Cantor, D. G. and Kaltofen, E., "Fast multiplication of polynomials over arbitrary rings," *Tech. Report* **87-35**, Dept. Comput. Sci., Rensselaer Polytechnic Institute, December 1987.

Caviness, B. F. and Epstein, H. I., "A note on the complexity of algebraic differentiation," *Inf. Proc. Lett.* **7**, pp. 122–124 (1978).

Chistov, A. L., "Fast parallel calculation of the rank of matrices over a field of arbitrary characteristic," *Proc. FCT '85, Springer Lec. Notes Comp. Sci.* **199**, pp. 63–69 (1985).

Cole, R. and Vishkin, U., "Optimal parallel algorithms for expression tree evaluation and list ranking," *Proc. AWOC 88, Springer Lec. Notes Comp. Sci.* **319**, pp. 91–100 (1988).

Galil, Z. and Pan, V., "Parallel evaluation of the determinant and of the inverse of a matrix," *Inform. Process. Letters* **30**, pp. 41–45 (1989).

Hoover, H. J., Klawe, M. M., and Pippenger, N. J., "Bounding fan-out in logical networks," *J. ACM* **31**/1, pp. 13–18 (1984).

Kaltofen, E., "Single-factor Hensel lifting and its application to the straight-line complexity of certain polynomials," *Proc. 19th Annual ACM Symp. Theory Comp.*, pp. 443–452 (1987).

Kaltofen, E., "Greatest common divisors of polynomials given by straight-line programs," *J. ACM* **35**/1, pp. 231–264 (1988).

Kaplansky, I., *An Introduction to Differential Algebra*; Hermann, Paris, 1957.

Kosaraju, S. R. and Delcher, A. L., "Optimal parallel evaluation of tree-structured computations by raking," *Proc. AWOC 88, Springer Lec. Notes Comp. Sci.* **319**, pp. 101–110 (1988).

†To our knowledge, no such program is known at this time. The best field independent solution of depth $O((\log n)^2)$ appears to be Chistov's (1985) with length $O(n^4 \log(n))$, and the best solution for fields of characteristic zero appears to be Preparata's and Sarwate's (1978) with length $O(n^3\sqrt{n})$. In both cases fast matrix multiplication will improve the stated lengths (see also (Galil and Pan 1989)).

Kung, H. T., "On computing reciprocals of power series," *Numer. Math.* **22**, pp. 341–348 (1974).

Miller, G. L., Ramachandran, V., and Kaltofen, E., "Efficient parallel evaluation of straight-line code and arithmetic circuits," *SIAM J. Comput.* **17**/4, pp. 687–695 (1988).

Miller, G. L. and Reif, J. H., "Parallel tree contraction Part 1: Fundamentals," in *Randomness in Computation*, Advances in Computing Research **5**, edited by S. Micali; JAI Press Inc., Greenwich, CT., pp. 47–72, 1989.

Morgenstern, J., "How to compute fast a function and all its derivations," *SIGACT News* **16**/4, pp. 60–62 (1985).

Preparata, F. P. and Sarwate, D. V., "An improved parallel processor bound in fast matrix inversion," *Inform. Process. Letters* **7**/3, pp. 148–150 (1978).

Sieveking, M., "An algorithm for division of power series," *Computing* **10**, pp. 153–156 (1972).

Strassen, V., "Berechnung und Programm I," *Acta Inf.* **1**, pp. 320–335 (1972). In German.

Strassen, V., "Vermeidung von Divisionen," *J. reine u. angew. Math.* **264**, pp. 182–202 (1973). In German.

Teng, S.-H., "The construction of Huffman-equivalent prefix code in $\mathcal{NC}$," *SIGACT News* **18**/4, pp. 54–61 (1987).

Valiant, L., "Reducibility by algebraic projections," *L'Enseignement mathématique* **28**, pp. 253–268 (1982).

Valiant, L., Skyum, S., Berkowitz, S., and Rackoff, C., "Fast parallel computation of polynomials using few processors," *SIAM J. Comp.* **12**, pp. 641–644 (1983).