1. Statement of Problem

One generic parallel evaluation scheme for algebraic objects, that of evaluating algebraic computation trees or formulas, is presented by Miller in a preceding chapter of this book. However, there are basic algebraic functions for which the tree model of computation seems not sufficient to allow an efficient—even sequential—decision-free algebraic computation. The formula model essentially restricts the use of an intermediate result to a single place, because the parse tree nodes have fan-out one. If an intermediate result participates in the computation of several further nodes, in the tree model it must be recomputed anew for each of these nodes. It is a small formal change to allow node values to propagate to more than one node at a deeper level of the computation. Thus we obtain the *algebraic circuit model*, which is equivalent to the *straight-line program model.* Figure 1 exhibits a small example of such an algebraic computation. Note that the node v_1 propagates its value into the nodes v_3 and v_4. This distinguishes the algebraic circuit from an *expression tree (formula)*, where each node would be restricted to fan-out 1.

$$\begin{aligned}
v_1 &\leftarrow x_2 + x_3\\
v_2 &\leftarrow x_4 + x_5\\
v_3 &\leftarrow x_1 \times v_1\\
v_4 &\leftarrow v_1 \times x_4\\
v_5 &\leftarrow v_3 + v_4\\
v_6 &\leftarrow v_5 \times v_2
\end{aligned}$$

Figure 1: Sample algebraic circuit and corresponding straight-line program.

We shall formally define this model in §1.2. Before discussing the parallelization results that are known in this context, we present algebraic circuits for several well-known algebraic objects. Since the size of the circuit corresponds to the length of the program that computes the object, it is important to keep the size as small as possible. For sake of the discussion we take the parallel time of an algebraic circuit to be the depth of that circuit, that is, the longest path in the corresponding directed acyclic graph (DAG).

1.1. Algebraic Circuit Examples

Symmetric Functions: The n elementary symmetric functions σ_i, $1 \le i \le n$ in n indeterminates $x_1, \ldots, x_n$ can be defined by the coefficients of a polynomial of degree n whose roots are the (negated) indeterminates as follows:

$$\left.\begin{aligned}
(z + x_1)(z + x_2) \cdots (z + x_n) = z^n + \sigma_1(x_1, \ldots, x_n) z^{n-1} + {}&\\
\sigma_2(x_1, \ldots, x_n) z^{n-2} + \cdots + \sigma_n(x_1, \ldots, x_n).&
\end{aligned}\right\} \quad (1)$$

Another way of computing σ_i is as the sum of the products of all combinations of i of the n indeterminates,

$$\sigma_i(x_1, \ldots, x_n) = \sum_{1 \le j_1 < j_2 < \cdots < j_i \le n} x_{j_1} \cdots x_{j_i}. \tag{2}$$

One important property is that these functions generate the algebra of symmetric polynomials in $x_1, \ldots, x_n$, where a symmetric polynomial f is one that remains invariant under permutation of the subscripts of the indeterminates, i.e., $f(x_1, \ldots, x_n) = f(x_{\tau(1)}, \ldots, x_{\tau(n)})$ for all permutations τ on the n subscripts $1, \ldots, n$. The task is to compute the σ_i's from the x_j's. Clearly, by multiplying out the linear terms in (1) we can determine all σ_i's in $O(n^2)$ additions and multiplications. Let us assume here that the values for the x_j's come from a ring. The computation will be by an algebraic circuit, and the parallel time will be $O(\log(n)^2)$ if we perform the polynomial multiplications in a tree-like fashion.

The question arises how to compute the middle $\sigma_{\lfloor n/2 \rfloor}$ by an algebraic computation tree with polynomially (in n) many nodes. Clearly, the formula (2) for $i = \lfloor n/2 \rfloor$ contains exponentially many terms under the sum, approximately $2^n/\sqrt{\pi n/2}$. Our question is actually equivalent to the question of computing $\sigma_{\lfloor n/2 \rfloor}$ by an algebraic circuit of parallel time $O(\log n)$. This is, in fact, possible by using an evaluation/interpolation approach (see Exercise 3), but that method leads to a fairly large polynomial-sized formula for $\sigma_{\lfloor n/2 \rfloor}$. In general, the algebraic circuit model appears superior to the expression tree model. It is hard to imagine how one would devise a computation tree for $\sigma_{\lfloor n/2 \rfloor}$ with $O(n \log(n)^2 \log(\log n))$ nodes, which is the node count of the asymptotically smallest known algebraic circuit that computes this function (see (Aho et al. 1974)).

Determinants: The basic invariant for linear system solving is the determinant of an $n \times n$ matrix. Treating the entries as indeterminates $x_{j,k}$, we can define it as a polynomial in these n^2 variables, namely,

$$\Delta_n = \mathrm{Det}\left([x_{j,k}]_{1 \le j,k \le n} \right) = \sum_{\tau \text{ a permutation on } 1, \ldots, n} \mathrm{sign}(\tau)\, x_{1,\tau(1)} \cdots x_{n,\tau(n)}, \tag{3}$$

where $\mathrm{sign}(\tau)$ is $+1$ or -1, depending whether τ is generated by an even or odd number of transpositions (i.e., permutations that switch two indices but leave the rest the same), respectively. Considering this sum of $n!$ terms alone, it is not obvious at all how to construct an algebraic circuit of polynomial size in n that computes this determinant. The solution hinges on the Gaussian elimination algorithm for solving linear systems. Consider the matrix $A^{(0)}$ of elements $a_{j,k}^{(0)} = x_{j,k}$. If we triangularize this matrix by elementary row operations, the determinant does not change. From (3) we conclude that the determinant of the resulting triangular matrix is the product of the elements on the diagonal. Let us assume for a moment that the elements come from a field. We have

$$\begin{array}{l}
\textbf{for } i \leftarrow 1, \ldots, n-1 \textbf{ do} \\
\quad \textbf{for } j \leftarrow i+1, \ldots, n \textbf{ do} \\
\quad\quad \textbf{for } k \leftarrow i+1, \ldots, n \textbf{ do} \\
\quad\quad\quad a_{j,k}^{(i)} \leftarrow a_{j,k}^{(i-1)} - \dfrac{a_{j,i}^{(i-1)} a_{i,k}^{(i-1)}}{a_{i,i}^{(i-1)}} \\
\Delta_n \leftarrow a_{1,1}^{(0)} a_{2,2}^{(1)} \cdots a_{n,n}^{(n-1)}
\end{array} \tag{4}$$

where the $a_{j,k}^{(i)}$ denote distinct circuit nodes.

Clearly, the obtained algebraic circuit has $O(n^3)$ nodes and requires $\Theta(n)$ parallel time. Since it contains divisions, it does not allow the evaluation of the determinant for any matrix of field elements. At this point, numerical procedures introduce decision statements that select proper non-zero elements before dividing. In general, the introduction of control of flow elements into a model of computation makes it difficult to parallelize sequential algorithms in that model. Furthermore, it is desirable to obtain division-free circuits for polynomial functions. We shall mention in §2.8 a method by Strassen that generically allows the removal of divisions in such a situation. For the determinant example here we shall now give a somewhat simpler solution.

First we observe that for the variables in the above program we have

$$a_{j,k}^{(i)} = \frac{\mathrm{Det}(A^{(0)}(1,\ldots,i,j;\,1,\ldots,i,k))}{\mathrm{Det}(A^{(0)}(1,\ldots,i;\,1,\ldots,i))}, \tag{5}$$

where $A^{(0)}(j_1, j_2, \ldots;\, k_1, k_2, \ldots)$ denotes the submatrix obtained from $A^{(0)}$ (the original input) by selecting the rows $j_1, j_2, \ldots$ and the columns $k_1, k_2, \ldots$ only. This fact is easy to prove, see for example Gantmacher (1960, Chapter II). If the entries in the input matrix come from an integral domain[1] D such as the integers or univariate polynomials over a field, we can simulate the arithmetic in the field of quotients by recording the numerator and denominator of a rational element separately. Defining the numerators and denominators of the right-hand side of (5),

$$\mathrm{Num}(a_{j,k}^{(i)}) = \mathrm{Det}(A^{(0)}(1,\ldots,i,j;\,1,\ldots,i,k)),\ \mathrm{Den}(a_{j,k}^{(i)}) = \mathrm{Det}(A^{(0)}(1,\ldots,i;\,1,\ldots,i)),$$

we get

$$\frac{\mathrm{Num}(a_{j,k}^{(i)})}{\mathrm{Den}(a_{j,k}^{(i)})} = a_{j,k}^{(i)} = \frac{\mathrm{Num}(a_{j,k}^{(i-1)})\,\mathrm{Num}(a_{i,i}^{(i-1)}) - \mathrm{Num}(a_{j,i}^{(i-1)})\,\mathrm{Num}(a_{i,k}^{(i-1)})}{\mathrm{Den}(a_{i,i}^{(i-1)})\,\mathrm{Num}(a_{i,i}^{(i-1)})}.$$

We conclude that $\mathrm{Den}(a_{i,i}^{(i-1)})$ can be divided out from the numerator of the right-hand-side expression. This observation leads to an algorithm for computing the determinant that uses subtractions, multiplications, and divisions by known factors. The following assumes that $B^{(0)}$ is initialized to the integral entries of the input matrix, and that $b_{0,0}^{(-1)} = 1$. Here the variables $b_{j,k}^{(i)}$ correspond to $\mathrm{Num}(a_{j,k}^{(i)})$.

for $i \leftarrow 1, \ldots, n-1$ **do**
  for $j \leftarrow i+1, \ldots, n$ **do**
    for $k \leftarrow i+1, \ldots, n$ **do**
      $b_{j,k}^{(i)} \leftarrow \left(b_{j,k}^{(i-1)} b_{i,i}^{(i-1)} - b_{j,i}^{(i-1)} b_{i,k}^{(i-1)}\right) \big/ b_{i-1,i-1}^{(i-2)}$
$\Delta_n \leftarrow b_{n,n}^{(n-1)}$.

[1] i.e., a commutative ring D with the property that

$$\forall b, p, q \in \mathsf{D}\colon bp = bq \text{ and } b \neq 0 \Longrightarrow p = q,$$

that is if an element $a = bq$ is divisible by $b \neq 0$, then there exists a unique quotient q.

We still have the problem of how to guarantee that the now exact division is not by zero. However, zero division can be avoided by considering a 'characteristic' matrix $B^{(0)} = z\,I_n + A$ in place of A. Here z is a new indeterminate, and I_n is the $n \times n$ identity matrix. Essentially, we have switched from the integral domain D of entries to the domain $\mathsf{D}[z]$ of entries. The exact divisions necessary in the second version of the algorithm now become polynomial divisions by the polynomials

$$b_{i,i}^{(i-1)} = \mathrm{Det}(z\,I_i + A(1,\ldots,i;\,1,\ldots,i)) = z^i + \text{lower order terms}.$$

We can now record the coefficients of z of the intermediate polynomial numerators explicitly (and drop the usage of z altogether from the computation). The exact divisions will be computations of polynomial quotients using polynomial divisors whose lead term is always 1. Since in the polynomial division process (Knuth 1981, §4.6.1, Algorithm D) the only divisions are by the lead term of the divisor, the operations on the arising coefficient vector elements will not necessitate a division by a domain element. Hence these polynomial divisions can be carried out no matter what the entries of the original matrix A are. By (5) it is also easy to see that the $b_{j,k}^{(i)}$'s have degree in z no more than $i+1$, so the polynomial arithmetic for each individual assignment can be accomplished in $O(n^2)$ additions, subtractions, and multiplications in the domain D itself. The determinant of A finally is the constant coefficient of $b_{n,n}^{(n-1)}$. In summary, we have described a division-free algebraic circuit with $O(n^5)$ nodes that can find the determinant of a matrix over an integral domain, in fact over any ring, in parallel time of order no more than $O(n^2)$.

Optimal Matrix Chain Multiplication: In the previous two examples, the arithmetic was carried out over a ring. However, for certain optimization problems it is sometimes necessary to work over an even weaker algebraic structure, for example that of a commutative semiring. In a commutative semiring we have an associative and commutative addition $\oplus$ and multiplication $\otimes$, such that the multiplication distributes over addition, i.e., $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ for all semiring elements a, b, and c. The dynamic programming algorithm presented here supplies an example for an algebraic circuit over the semiring of nonnegative integers with the formal addition $a \oplus b = \min\{a, b\}$ and the formal multiplication $a \otimes b = a + b$. The problem is the following: We are given m matrices $A_1, \ldots, A_m$ with entries from a ring, such that A_i is of dimension $n_i \times n_{i+1}$ for all $1 \le i \le m$. We want to compute the product $A_1 A_2 \cdots A_m$ using standard matrix multiplication, but we wish to pick the order of multiplications in such a way that the total number of ring multiplications is minimized. For example, if $m = 4$, $n_1 = n_4 = 1$, and $n_2 = n_3$, then an optimal order is $(A_1 A_2)(A_3 A_4)$ costing $O(n_2^2)$ operations, while the order $A_1(A_2 A_3)A_4$ requires the full $n_2 \times n_2$ matrix product $A_2 A_3$. An optimal order for the general problem is found by dynamic programming as follows (see also Aho et al. (1974), §1, for a more detailed discussion of this technique): let $c_{i,j}$ denote the optimal number of operations to compute $A_i A_{i+1} \cdots A_j$ for $1 \le i \le j \le m$ with the initial values $c_{i,i} = 0$. For all $1 \le i < j \le m$ we have

$$c_{i,j} = \min\{c_{i,k-1} + c_{k,j} + n_i n_k n_j \mid \text{for all } k \text{ with } i < k \le j\}, \tag{6}$$

the interpretation being that we try all possible splits $(A_i \cdots A_{k-1})(A_k \cdots A_j)$. One can now evaluate $c_{1,m}$ by employing the recursive definition (6) together with *memoization* (see

Abelson and Sussman (1985) for the usage of this notion) that is one computes an individual $c_{i,j}$ only once and looks up its value if it is needed again. Usually, the resulting algebraic circuit is presented bottom-up:

$$\begin{array}{l} \textbf{for } l \leftarrow 1, \ldots, m-1 \textbf{ do} \\ \quad \textbf{for } i \leftarrow 1, \ldots, m-l \textbf{ do} \\ \qquad c_{i,i+l} \leftarrow \bigoplus_{k=i+1}^{i+l} c_{i,k-1} \otimes c_{k,i+l} \otimes (n_i n_k n_{i+l}). \end{array}$$

Notice that the integer $n_i n_k n_j$ corresponds to a circuit node containing a constant, so the algebraic circuit is actually over the semiring $(\mathbb{Z}_{\geq 0}, \oplus, \otimes)$. In fact, it is memoization that reduces the number of trial combinations from exponentially many to $O(n^3)$, corresponding exactly to reducing the exponential sized computation tree to a polynomial sized computation DAG. The bottom-up realization of this technique is referred to as *dynamic programming*

1.2. Definitions and Main Results

The three examples given above all exhibit algebraic circuits for important multivariate polynomials. Circuits for the determinant and the optimal matrix product seemingly require fan-out higher than one in order to have only polynomially many nodes. Only the circuit for the symmetric functions has parallel time $O(\log(n)^2)$. The goal of this article is to give a transformation for any algebraic computation DAG with a certain (necessary) degree restriction that results in a DAG computing the same functions, but in log-squared depth. We now define the model of an algebraic circuit and its formal degree precisely, and state the main result.

Definition 1. An *algebraic circuit* C over a semiring $\mathbb{R}$ is an edge-weighted directed acyclic graph with node set V and (directed) edges E satisfying the following conditions:

- Each edge has either an element from $\mathbb{R}$ as its weight, or the special symbol "unit-weight." For a (directed) edge $(v, w) \in E$, where v and w are nodes, we denote its weight by $W(v, w)$.
- Each node is labeled as one of three types: a *leaf*, an *addition node*, or a *multiplication node*.
- Any leaf has in-degree (*fan-in*) zero, i.e., there is no edge leading into the corresponding node, any addition node has in-degree no less than one, and any multiplication node has in-degree exactly two.
- Every leaf v is also assigned an element in $\mathbb{R}$, denoted by value(v).

A *formula* or *expression tree* is an algebraic circuit in which each node has at most one edge directed out of it. □

This definition is more general than what we have used so far in two ways: first, there are weights on the edges with the interpretation that a semiring value is multiplied with the weight on an edge before being fed to the target node along that edge. These edge-weights are merely a conceptual tool for the transformations presented later. Clearly, one can realize the multiplication by a weight as an additional multiplication node. The reason why we also need the special weight "unit-weight" is that in $\mathbb{R}$ we are not guaranteed to have a multiplicative unit. The second difference is that we allow arbitrarily many edges leading into an addition node. Such "super-nodes" must be ultimately realized by a balanced binary tree of addition nodes of fan-in two. Otherwise, one could compute an n-dimensional determinant in parallel-time $O(\log n)$ by employing a tree for (3) with exponentially many nodes, which is clearly unrealistic.

The set of *children* of a node v in the circuit C are all those nodes $u \in V$ such that there is an edge $(u, v) \in E$ from u to v. With this notion we can recursively define the value of each node:

Definition 2. The value of a leaf w is the associated value(w). Now let v be a non-leaf node of the circuit C, and let U be the set of all its children. We define value(v) in terms of the value(u) and the edge weights $W(u, v)$ for $u \in U$. Assume first that v is an addition node. Then we define

$$\text{value}(v) = \sum_{u \in U} \text{propagate}(u, v),$$

where

$$\text{propagate}(u, v) = \begin{cases} \text{value}(u) & \text{if } W(u, v) = \text{"unit-weight"}, \\ \text{value}(u) \times W(u, v) & \text{if } W(u, v) \text{ is a semiring element.} \end{cases}$$

If v is a multiplication node we have

$$\text{value}(v) = \prod_{u \in U} \text{propagate}(u, v).$$

Notice that in this case we have precisely two factors under the product. □

There is an almost 1-1 correspondence between algebraic circuits and straight-line programs (refer back to Figure 1). We shall not give a precise definition for the latter (see, e.g., Strassen (1972)), but we wish to point to some minor differences. In a straight-line program there exists a (somewhat artificial) total order of the intermediate variables that is not apparent in an algebraic circuit. Furthermore, in a straight-line program we are allowed to issue assignments of the form $v \leftarrow u \times u$, which are somewhat excluded in a circuit (there cannot be more than one edge from u to v). However, we clearly can simulate such an assignment by the pair of assignments $w \leftarrow u; v \leftarrow u \times w$, where w can be chosen to be an addition node with fan-in one in the corresponding algebraic circuit. This trick will be needed anyway in our results, since the transformation algorithm will be restricted to circuits that do not have any edges from a multiplication node to another multiplication node.

The *size* of an algebraic circuit is simply the number of its nodes, excluding the leaf nodes. The *depth* of a circuit is the longest path in its defining DAG. We observe that for an algebraic circuit of fan-in not higher than two, the size corresponds to the number of arithmetic operations in a straight-line program that computes the node values, and that the depth corresponds to the parallel time. The goal of our parallelizing transformations is to construct a new algebraic circuit that computes all values of a given circuit but that has much smaller depth. For instance, we will construct a fan-in bounded circuit for the determinant of an $n \times n$ matrix that has depth $O(\log(n)^2)$. However, it is easily shown that not all algebraic circuits can be parallelized with such a dramatic gain in depth. Consider the straight-line program

$$v_1 \leftarrow x \times x; v_2 \leftarrow v_1 \times v_1; \ldots; v_n \leftarrow v_{n-1} \times v_{n-1};$$

the variable v_n computes the value x^{2^n}. Now the optimal algebraic circuit with respect to depth that computes the same function x^{2^n} must have depth $\Omega(n)$. For we can partition the nodes of that circuit into levels, where the ith level contains the nodes for which the maximum length path originating in any leaf node is of length exactly i. It follows by induction on i that the degrees of the values computed by the nodes on level i as polynomials in indeterminates replacing the leaf values is not higher than 2^i. Thus any algebraic circuit computing x^{2^n} must have depth at least n. Note that this argument is only properly valid for semirings in which the function x^{2^n} cannot be realized by a lower degree polynomial function in x. It is thus certainly true for any infinite integral domain. With this simple lower bound in mind, we will restrict our transformation to circuits that compute functions of polynomially bounded degree.

We now define the *formal degree* of a node in an algebraic circuit:

Definition 3. The formal degree of a leaf node in an algebraic circuit is defined as the integer 1. The formal degree of an interior node v is defined recursively from the formal degrees of its children:

$$\text{degree}(v) = \begin{cases} \max\{\text{degree}(u) \mid u \text{ a child of } v\} & \text{if } v \text{ is an addition node,} \\ \sum_{u \text{ a child of } v} \text{degree}(u) & \text{if } v \text{ is a multiplication node.} \end{cases}$$

Finally, the formal degree of the entire circuit is the maximum of the formal degree of any node. □

Essentially, the degree of a node is the value that the node has if we replace all leaf values in the circuit by 1, and convert all additions to taking the maximum and all multiplications to integer addition. As in the matrix chain product example above, we still have a circuit over a semiring at hand. It should be realized that the formal degree of a node might be higher than the degree of its algebraic value as a polynomial in the leaf values; this is because leading terms of the children of an addition node can cancel one another: if u_1 has value x^2 and u_2 has value $x - x^2$, then $v \leftarrow u_1 + u_2$ has formal degree 2, but actually its value is x. Note that over an arbitrary ring such "loss of leading terms" (to paraphrase the loss of significant digits in numerical computing) can also occur for multiplication nodes.

We now can formulate the main result of this article. Given is the description of an algebraic circuit in which no addition node has fan-in higher than two. All leaf values and edge weights are represented as distinct symbols. The circuit has size n and formal degree d. We shall use the function $M(n)$ that denotes the asymptotic circuit size for $n \times n$ matrix multiplication over the semiring that is simultaneously of depth $O(\log n)$. The standard algorithm gives $M(n) = O(n^3)$ for any semiring, and the best fast method over a ring has $M(n) = O(n^{2.3755})$ (Coppersmith and Winograd 1990). The result is the following: One can compute, on a parallel random access machine with $M(n)$ processors running in time $O(\log(n) \log(dn))$ the description of a circuit of

$$\text{depth} = O(\log(n) \log(dn)) \text{ and size} = O(M(n) \log(dn)),$$

a subset of whose nodes attain all values of the input circuit. A similar result can be obtained for other models of parallel algebraic complexity, e.g., the parallel algebraic PRAM or arithmetic Boolean circuits discussed in von zur Gathen's chapter. However, since we do not formally introduce these models, we have confined our discussion to the standard PRAM and circuit models.

1.3. Exercises

Exercise 1. Determine the formal degree of the algebraic circuit over the semiring ($\mathbb{Z}_{\geq 0}$, $\min, +$) for computing the optimal matrix chain product, which is discussed in §1.1.

Exercise 2. Show that the formal degree of the division-free algebraic circuit for computing the determinant of an $n \times n$ matrix over a ring described in §1.1 is of order $O(n)$. [Hint: the exact division circuit over $\mathsf{D}[z]$ computes polynomials that are homogeneous in $x_{j,k}$, the entries of A, and in z, the new indeterminate. Hence the coefficients of z^m have lower degree as polynomials in $x_{j,k}$ if m is larger.]

Exercise 3*.[2] Construct a circuit of depth $O(\log n)$ that computes $\sigma_{\lfloor n/2 \rfloor}(x_1, \ldots, x_n)$ over any ring. [Hint (see (Eberly 1989)): if the ring is the complex numbers, and ω is a primitive 2^k-th root of unity, $2^{k-1} < n+1 \leq 2^k$, one can interpolate the polynomial $(z+x_1) \cdots (z+x_n)$ at the points $1, \omega, \omega^2, \ldots, \omega^{2^k-1}$ in $O(\log n)$ time by the discrete fast Fourier transform.]

Exercise 4. Given is an algebraic circuit over a semiring with n nodes and with depth $O(\log n)$. Show that there exists a formula with $n^{O(1)}$ nodes that computes the values of each node of the circuit.

1.4. History of Solutions

The theory on parallelizing transformations of algebraic programs starts with Brent's (1974) seminal paper on parallelizing the parse tree of an expression. A polynomial-sized, poly-logarithmic depth circuit for the determinant and inverse of an $n \times n$ matrix over a field of characteristic 0 goes back to Csanky (1976). Hyafil (1979) obtained poly-logarithmic depth circuits for the general problem, but his construction needed super-polynomially many nodes. Valiant and Skyum (1981) made the construction also work simultaneously in polynomial size (see also (Valiant et al. 1983)). With the recasting of Brent's problem as a dynamic question, i.e., by requiring the transformation itself to be a parallel algorithm, the techniques discussed in this chapter were introduced. Miller and Reif's (1989) tree contraction process of 1985 solved the dynamic formula evaluation problem, and the dynamic circuit evaluation algorithm followed (Miller, Ramachandran, and Kaltofen 1988). Both algorithms have found their way into the textbooks (Gibbons and Rytter 1988), (Leighton 1991). Follow-up papers containing additional results with respect to the circuit compression algorithm are (Miller and Teng 1987) and (Mayr 1987).

Circuits or straight-line programs have been used for a long time as a structured model of algebraic computation. We refer to Strassen's (1984) article for an excellent survey of the known results and open problems, as well as to Karp and Ramachandran's (1990) excellent survey on parallel algorithms in general. The problem of dealing with divisions will be discussed in detail in §3.1, and we delay the citation of work to that sub-section. We will also introduce other related results in §3. The application of circuit compression to dynamic programming goes back to Valiant et al. (1983), where the context free language recognition problem is solved in poly-logarithmic time using that technique. We consider the application of this algorithm to the optimal matrix chain product circuit presented here as part of the folklore of that method.

[2] Starred exercises are difficult, and may contain/constitute publishable research results.

2. Algebraic Circuit Compression

We now describe the algorithm by Miller et al. (1988) and present its complexity analysis. This algorithm was developed by attempting to combine the ideas of Miller and Reif (1985) for tree compression (see also the chapter by Miller in this book) with the ideas by Valiant et al. (1983) for straight-line code evaluation. However, in its final form it appears quite distinctive and even introduces a graph transformation (called the "shunt operation") that has found its way back into the simple and elegant solution to tree compression by Abrahamson et al. (1989) and by Kosaraju and Delcher (1988).

2.1. The Compression Algorithm

We assume that the input circuit has no edges from multiplication nodes to multiplication nodes. As pointed out in §1.2, if this is not the case, then we add unary addition nodes along those edges that connect multiplication nodes. The algorithm will apply transformations to the DAG. These transformations can affect the circuit in the following ways: they may add a new edge with a certain weight between nodes previously unconnected, they may remove an edge, they may change an addition or multiplication node into a leaf node with a certain value, or they may change the weight on an existing edge. There are three types of transformations, the so-called *Matrix-Multiply* transformation, the *Rakes*, and the *Shunts*. A combination of these three transformations applied in that order is called a *Phase*. The algorithm repeats Phases until every interior node is turned into a leaf. The crucial theorem in its analysis is that this takes no more than $O(\log(nd))$ Phases, where n is the number of nodes and d is the formal degree of the input circuit. Let us now define the individual transformations. For this it is useful to label the circuit nodes in order as $v_1, \ldots, v_n$.

Procedure *Matrix-Multiply:* Consider the following $n \times n$ matrices derived from the edge-weights $W(v_i, v_j)$ on the nodes $v_1, \ldots, v_n$ of the circuit.

$$W[\![+,+]\!]_{i,j} = \begin{cases} W(v_i, v_j) & \text{if } v_i \text{ and } v_j \text{ are addition nodes,} \\ 0 & \text{otherwise;} \end{cases}$$

$$W[\![\,.\,,+]\!]_{i,j} = \begin{cases} W(v_i, v_j) & \text{if } v_j \text{ is an addition node,} \\ 0 & \text{otherwise;} \end{cases}$$

$$W[\![\,.\,,\,.\,]\!]_{i,j} = \begin{cases} W(v_i, v_j) & \text{if } \underline{\text{not}} \text{ both } v_i \text{ and } v_j \text{ are addition nodes,} \\ 0 & \text{otherwise.} \end{cases}$$

For simplicity, we shall assume that the semiring associated with the circuit has an additive zero, denoted by 0, and furthermore, that it also has a multiplicative unit, denoted by 1. We thus need no symbolic edge-weights "unit-weight" (see Exercise 5 below for removing these restrictions). Now, we compute the matrix product and sum

$$W' \leftarrow W[\![\,.\,,+]\!] \cdot W[\![+,+]\!] + W[\![\,.\,,\,.\,]\!] \tag{7}$$

and replace all edge-weights $W(v_i, v_j)$ by $W'_{i,j}$. Notice, that this action not only replaces weights on edges, but it may introduce a new edge (in case $W'_{i,j} \neq 0$ but (v_i, v_j) is not an edge in the original circuit) or remove an old edge (in case $W'_{i,j} = 0$ but (v_i, v_j) is an edge). Furthermore, in case all edges into an addition node are removed, that node is converted into a leaf of value 0. The mechanics of this procedure are illustrated in Figure 2.

$$\begin{pmatrix} 0 & a & b & c \\ 0 & 0 & \gamma & \alpha \\ 0 & 0 & 0 & \beta \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \gamma & \alpha \\ 0 & 0 & 0 & \beta \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & a & b & c \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & a & a\gamma + b & a\alpha + b\beta + c \\ 0 & 0 & 0 & \beta\gamma \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Figure 2: Matrix-Multiply example.

Essentially, the matrix multiplication in (7) collects the combined weights from any node to a second level addition node through intermediate addition nodes. We add to that weight the weight that already is on the edge between those nodes. However, if those two nodes are addition nodes, no weight must be added, since that weight will be moved upward to a connection from an earlier node to the target node, explaining the definition of $W[\![\,.\,,\,.\,]\!]$. Note that this procedure does not alter the fan-in into a multiplication node, and the formal Definition 1 for an algebraic circuit remains valid for the resulting circuit.

Procedure *Rake:* Whenever all children of an interior node have become leaves, we can compute the value of that interior node. This is done as follows:

> **for all** addition nodes v all of whose children are leaves **do**
> { set v to a leaf node;
> value(v) $\leftarrow \sum_{u \text{ a child of } v} W(u,v) \cdot$ value(u);
> remove the edges (u,v) for all children u of v; }
> **for all** multiplication nodes v all of whose children are leaves **do**
> { set v to a leaf node;
> value(v) $\leftarrow \prod_{u \text{ a child of } v} W(u,v) \cdot$ value(u);
> remove the edges (u,v) for all children u of v. }

The procedure essentially removes leaves from active participation in the computation, hence the name 'rake.'

Procedure *Shunt:* This procedure processes those multiplication nodes one of whose children is a leaf. Then one can pass those leaf values together with the edge weights through to the edge between the non-leaf child and the parents of the multiplication node under consideration. Figure 3 exhibits this action on a small circuit, while the following code defines this procedure formally:

for all multiplication nodes v_j that have exactly one leaf child v_l and one child v_i that is an interior node **do**
  for all parents v_k of v_j **do**
    $P_{i,j,k} \leftarrow W(v_i, v_j) \cdot W(v_j, v_k) \cdot W(v_l, v_j) \cdot \text{value}(v_l)$;
for all pairs (i, k) considered previously **do**
  { $W'_{i,k} \leftarrow \sum_{\text{all considered intermediate } j} P_{i,j,k}$; and
  $W(v_i, v_k) \leftarrow W(v_i, v_k) + W'_{i,k}$;
  set the weight of the edge (v_i, v_k) to $W(v_i, v_k)$, unless $W(v_i, v_k) = 0$, in which case the edge is removed; }
for all (j, k) considered previously **do**
  remove the edge (v_j, v_k).

Figure 3: Shunt example.

The **procedure** *Phase* is the consecutive execution of these three procedures in the order with which we presented them. The **algorithm** *Circuit Compression* now performs a sequence of phases until every node in the circuit has been converted into a leaf.

We first observe that the value of each node in the input circuit (see Definition 2) is not changed by any of our circuit transformation procedures. The formal proof is done by induction on the nodes $v_1, \ldots, v_n$ going from lowest level (the leaves) to the highest level. It is in this argument that we appeal to the axioms of a commutative semiring. Especially, the commutativity of multiplication plays a significant role; in fact, it is an open problem how to do without commutativity while remaining equally efficient (see the conclusion). Second, we note that the algorithm eventually obtains the value of every node. In fact, it will complete this quite quickly. We will prove next that after $O(\log(nd))$ phases all nodes are leaves, but it should be clear that the algorithm makes some progress towards completion in every one of the phases.

We have given a high level formulation of our algorithm as it would run on an algebraic random access machine (RAM) with the capability of performing semiring arithmetic. However, in the introduction we have emphasized the circuit transformation aspect of our result.

Hence, one has to visualize an algorithm that does not actually perform the semiring arithmetic stated in our procedures, but that hardwires the corresponding algebraic subcircuits together for the new compressed circuit.

2.2. Analysis of the Compression Algorithm

We will carry out the run-time analysis in two steps. First, we are going to show that each individual procedure in a phase can be carried out efficiently on a PRAM. Second, we will show that the algorithm needs no more than $O(\log(nd))$ phases to complete, where n is the number of nodes and d is the formal degree of the circuit.

Our first goal is quite straightforward. The reader may consider a PRAM that has the circuit by means of a vector of leaf values and the matrix of weights W symbolically represented in its shared memory. In carrying out the Matrix Multiplication procedure using $M(n)$ processors, it stores the necessary circuit fragments in a place of shared memory that builds the compressed new circuit. The time needed is $O(\log n)$, as is in the matrix multiplication algorithm used. The details are somewhat technically involved, and we will not expand on them here. Exercise 7 below deals with the precise argument, and a reference to the solution of a very similar problem can be found there. The Rakes can be encoded by no more than n balanced binary addition or multiplication trees of no more than n leaves, which can be done by an n^2-processor PRAM in $O(\log n)$ time. A similar processor count/time measure yields the Shunts: The key observation is that every $P_{i,j,k}$ corresponds to exactly one edge (v_j, v_k), because the multiplication node v_j has fan-in two. Thus, there are $O(n^2)$ elements $P_{i,j,k}$ to be considered for the summations to obtain $W'_{i,k}$. One uses a PRAM sorting procedure (say the one by Cole (1988), also in the chapter on sorting in this book) with the pairs of subscripts (i,k) as the keys. Then one sums up all $P_{i,j,k}$ with the same key, which are now located in adjacent memory locations. In summary, we can carry out procedure Shunt on a PRAM with n^2 processors in $O(\log n)$ parallel time.

We now estimate the number of phases necessary to evaluate every node in the input circuit. In order to obtain a sufficiently precise count, we measure the progress made in each such phase. A first idea is to express this progress in terms of the formal degrees of the circuits before and after each phase. Clearly, the procedures Shunt and Rake on multiplication nodes affect the formal degree of some nodes. But the procedure Rake on addition nodes and, more importantly, the procedure Matrix Multiply may not, especially along long chains of addition nodes. For this reason, we introduce an artificial new formal degree, which we will name the *height* and which is similar to the formal degree (see Definition 2), but which also grows along paths containing only addition nodes.

Definition 4. The height of a leaf is defined as the integer 1. The height of an interior node v is defined recursively from the heights of all its children: if v is an addition node we define

$$\text{height}(v) = \max\left\{a + \tfrac{1}{2}, m\right\} \text{ where } \begin{cases} a = \max\{\text{height}(u) \mid u \text{ an addition child of } v\}, \\ m = \max\{\text{height}(u) \mid u \text{ a multiplication child or} \\ \qquad\qquad \text{a leaf child of } v\}; \end{cases}$$

and if v is a multiplication node we define

$$\text{height}(v) = \sum_{u \text{ a child of } v} \text{height}(u).$$

Finally, we define the height of the entire circuit as the maximum of the heights of any node. □

The following two lemmas are the key to the analysis of the algorithm. The proofs for the lemmas, which are based on induction on the structure of the circuit, are given at the end of this subsection. The first lemma estimates the progress made in each phase with respect to the height measure.

Lemma 1. *Consider a node v in an algebraic circuit C. We apply a single Phase to the circuit, obtaining a compressed circuit C' with the node v' corresponding to v. Now assume that the node v' in the circuit C' is not a leaf node or a node with fan-out zero, a so-called output node. Then the heights of the node before and after the single Phase compression are related by*

$$\mathrm{height}(v') \le \tfrac{1}{2}\,\mathrm{height}(v).$$

The second lemma relates the height of a circuit to its formal degree:

Lemma 2. *Consider an algebraic circuit: Let d be its formal degree, h its height, and $E[\![+,+]\!]$ the subset of its edges that go from an addition to an addition node. Then*

$$h \le \tfrac{1}{2}\,d\,e + d \quad \text{where } e \text{ is the cardinality of } E[\![+,+]\!].$$

From these lemmas we can easily deduce an upper bound for the running time of our algorithm.

Theorem 1. *The Circuit Compression algorithm completes in no more than*

$$O(\log(n) + \log(d))$$

consecutive applications of the procedure Phase, where n is the number of nodes and d is the formal degree of the input circuit.

Proof. The number e of addition-to-addition edges is $O(n^2)$, hence the height of the input circuit (and by its definition the height of any of its nodes) is by Lemma 2 of order $O(n^2 d)$. Now, by Lemma 1 each Phase reduces the height of an interior node that does not become an output node by a divisor of at least 2. Hence after $O(\log_2(n^2 d))$ Phases there can be only leaves and output nodes left in the circuit. An additional Phase will by virtue of the procedure Rake evaluate all output nodes. The number of Phases necessary in the Compression algorithm is therefore bounded by $c \log_2(n^2 d) + 1 = O(\log(nd))$, where c is constant. ⊠

There are several corollaries that immediately follow from this theorem and the processor count given at the beginning of this subsection. The first corollary shows that all polynomials of polynomial formal degree—even quasi-polynomial, as we will state it—can be computed in poly-logarithmic depth.

Corollary 1. *Given is a multivariate polynomial $f(x_1, \dots, x_m)$ with coefficients from a commutative semiring. Suppose that f is the value of an n-node algebraic circuit with leaf values $x_1, \dots, x_m$, whose formal degree is*

$$n^{O(\log(n)^c)} \quad \text{for a constant } c \geq 0,$$

i.e., quasi-polynomial in n. Then f is the value of an algebraic circuit with leaf values $x_1, \dots, x_m$ that has bounded fan-in and

$$n^{O(1)} \text{ many nodes and } O(\log(n)^{c+2}) \text{ depth.} \quad \boxtimes$$

The second corollary addresses the problem of computing all values of an algebraic circuit in parallel. Since we have not formally introduced the notion of an algebraic PRAM here, we shall formulate the corollary for integers as our semiring.

Corollary 2. *Consider an exclusive-read/exclusive-write PRAM that has the description of an n-node algebraic circuit over the integers stored in its shared memory. Let d be the formal degree of the circuit, and assume that the PRAM has $M(n)$ many processors. Than the values of all nodes of the circuit can be computed in $O(\log(n)\log(nd))$ arithmetic parallel PRAM steps, where arithmetic refers to the fact that we will count an integer arithmetic operation as a single step independently of how many bits the individual integer operands have.* $\boxtimes$

Notice that this corollary shows how to find the formal degree of an algebraic circuit dynamically. One replaces the leaf values by the integer 1, addition nodes by the maximum function, and multiplication nodes by integer addition. The Circuit Compression algorithm executed on a PRAM will complete in $O(\log(n)\ \log(dn))$ time, where d is the wanted formal degree. Note that d appears in the run-time estimate, but nothing needs to be known for it (cf. Theorems 2 and 3 below).

We conclude this subsection with the proofs of the Lemmas 1 and 2. Both proofs will be by induction on the size of the circuit. We will consider the subcircuit C_v for a circuit C and one of its nodes v, that we define as the circuit of all those nodes and edges of C that participate in the determination of the value of v (see Definition 4). We will also use the notion of a *dominant child* of an addition node v. If among the children of v there is a multiplication or leaf node u with $\text{height}(u) = \text{height}(v)$, we call u a dominant child. If there is no such node, we call a child u with $\text{height}(v) = \text{height}(u) + \frac{1}{2}$ a dominant child. Essentially, a dominant child determines the height of v.

Proof of Lemma 1. We consider the node v' in the circuit C'. The proof proceeds by induction on the depth of the subcircuit $C'_{v'}$. The basis case is where all children of v' are leaves. We argue this case first for v' being an addition node. We must show that the height of v is no less than 2, where v is the node corresponding to v' before the application of a Phase. Assume to the contrary that $\text{height}(v) \leq 3/2$. By the definition of height, the depth of C_v can be no more than 2 and cannot contain a multiplication node. Clearly the procedures Matrix-Multiply and Rake would convert v into a leaf node in C', in contradiction to our assumption that v' is not a leaf.

The second part of the basis case is when v' is a multiplication node. It suffices to show that both children of v have height at least 2. Suppose to the contrary that a child u has height$(u) < 2$. After raking addition nodes, this child will be a leaf. If the second child is also a leaf at that point, raking the multiplication nodes will turn v' into a leaf. Otherwise, the procedure Shunt will disconnect v' from the rest of C', thus making v' an output node. Both are in contradiction to our assumption that v' is not a leaf or an output node.

We now present the inductive argument. The easier case here is when v' is a multiplication node. If none of its children u_1' and u_2' are leaves, we have by the induction hypothesis

$$\mathrm{height}(v') = \mathrm{height}(u_1') + \mathrm{height}(u_2') \le \tfrac{1}{2}\left(\mathrm{height}(u_1) + \mathrm{height}(u_2)\right) = \tfrac{1}{2}\,\mathrm{height}(v).$$

If one of its children u_i' is a leaf, we note that height$(u_i) \ge 2$, for otherwise the node v would have participated in a Shunt and been turned into an output node, as was argued in the basis case. Thus the same inequality holds, leaving only the case where v' is an addition node.

Let u' be a dominant child of the addition node v'. If u' is a multiplication node, then

$$\mathrm{height}(v') = \mathrm{height}(u') \le \tfrac{1}{2}\,\mathrm{height}(u) \le \tfrac{1}{2}\,\mathrm{height}(v)$$

by the induction hypothesis and the fact that the height function does not decrease along paths. The node u' cannot be a leaf, because otherwise that case would be the basis case. Thus we are left with the possibility that u' is an addition node. It suffices to establish in this last case that

$$\mathrm{height}(u) \le \mathrm{height}(v) - 1, \tag{8}$$

for then we have by induction hypothesis

$$\mathrm{height}(v') = \mathrm{height}(u') + \tfrac{1}{2} \le \tfrac{1}{2}\,\mathrm{height}(u) + \tfrac{1}{2} \le \tfrac{1}{2}\,\mathrm{height}(v).$$

It remains to show (8). Note that both u and v are addition nodes. If there is a path from u to v containing an additional node, (8) follows from Definition 2. Finally, suppose that the only path from u to v in C is a single edge. This configuration is impossible, however, since the procedure Matrix-Multiply will remove that edge, and no edge (u', v') can be introduced because there is no other path from u to v. ⊠

Proof of Lemma 2. The proof is by induction on the number of nodes in C. Clearly, for a single node circuit, which must be a single leaf, the estimate is true with $e = 0$ and $h = d = 1$. Suppose now that the theorem is true for all circuits with less than n nodes, and let C be a circuit with n nodes. For every non-output node u we have by induction hypothesis

$$\mathrm{height}(u) \le \mathrm{degree}(u)\left(\frac{e_u}{2} + 1\right) \le d\left(\frac{e}{2} + 1\right), \tag{9}$$

where e_u is the number of addition to addition edges in C_u, which satisfies $e_u \le e$, because the circuit C_u is a subcircuit of C. It thus remains to prove the estimate for every output node v in C. First let v be a multiplication node, and let u_1 and u_2 be its children. Then by (9) we have

$$\mathrm{height}(v) = \mathrm{height}(u_1) + \mathrm{height}(u_2) \le (\mathrm{degree}(u_1) + \mathrm{degree}(u_2))\left(\frac{e}{2} + 1\right) \le d\left(\frac{e}{2} + 1\right).$$

Second, assume that v is an addition node. If the dominant child is a multiplication or leaf child, height(v) will be equal to the height of that child, which by (9) already satisfies the estimate. So we are finally left with the situation that the dominant child u of v is an addition child. We then have

$$\text{height}(v) = \text{height}(u) + \tfrac{1}{2} \le \text{degree}(u)\left(\frac{e_u}{2} + 1\right) + \tfrac{1}{2} \le d\left(\frac{e-1}{2} + 1\right) + \tfrac{1}{2}, \qquad (10)$$

because $d \ge \text{degree}(v) \ge \text{degree}(u)$ and $e_u + 1 \le e$, the latter since the subcircuit C_u does not contain the edge (u, v). Expanding out the right-hand side of (10) and observing that $d \ge 1$ completes the proof. ⊠

2.3 Exercises

Exercise 5. Modify the description of the Circuit Compression algorithm so that it is valid over a semiring without a multiplicative unit. [Hint: the problem is that the Matrix-Multiply procedure may generate the addition $1 + 1$ for the formal unit edge weight; introduce an additional integer weight on each edge with the interpretation that $2 \cdot a = a + a$ for any semiring element a.]

Exercise 6. Show how to accomplish circuit compression over a fixed finite non-commutative ring. [See Miller and Teng (1987) for a solution.]

Exercise 7. This exercise relates the problem of building description of circuits on a standard PRAM to evaluating circuits on an algebraic PRAM. Given is a $P(n)$-processor algebraic PRAM that evaluates an algebraic circuit in $O(T(n))$ steps without testing an element of the semiring for zero. Show that there exists a standard $P(n)$-processor PRAM that on the same input constructs in $O(T(n))$ time the formal description of a circuit of size $O(P(n)\,T(n))$ and depth $O(T(n))$ that computes all values of the input circuit. [The solution to the sequential counterpart of this problem can be found in (Kaltofen 1988, Theorem 4.1); the notion of an algebraic RAM is also defined there.]

Exercise 8. Consider an $n \times n$ matrix with polynomial entries of degree no more than n and coefficients from a ring. Construct a circuit of size $n^{O(1)}$ and depth $O(\log(n)^2)$ that computes the coefficients of the determinant polynomial of the matrix.

Exercise 9. Exhibit an n-node circuit of formal degree polynomially bounded in n, such that the Circuit Compression algorithm requires $\Omega(\log(n)^2)$ parallel steps.

Exercise 10*. Given are $n \times n$ matrices $A_1, A_2 \ldots$ and an n-dimensional column vector b over a ring. Consider the sequential circuit of size $O(n^2 m)$ for computing $A_1 \cdots A_m b$ by multiplication from right to left. Give a sharper than $O(M(n^2 m))$ estimate for the number of nodes in the circuit produced by the Circuit Compression algorithm. [Open problem: Is there a circuit of depth $O(\log(n)\log(m))$ and size $O(n^2 m)$ that computes that matrix chain product?]

Exercise 11*. Given is a non-singular $n \times n$ matrix over a field. Find a circuit for the Gram-Schmidt orthogonalization of that matrix that has $n^{O(1)}$ nodes and $O(\log(n)^2)$ depth (Kozen 1987).

The following exercises are taken from (Leighton et al. 1988, Problem Set 1).

Exercise 12. Show that for a ring, the Compression Algorithm can produce a parallel circuit of depth $O(\log(n)\log(d))$, where n is the number of nodes of the input circuit and d is the formal degree. [Hint: eliminate all plus-plus edges in the input circuit by applying the procedure Matrix Multiply repeatedly before performing the full compression.]

Exercise 13*. Consider an algebraic circuit C with n nodes and formal degree d over the integers. Construct a Boolean circuit that evaluates C for l-bit integers, and that has $(dnl)^{O(1)}$ many gates and $O(\log(dnl)\log(dn))$ delay. [Hint: one can add N L-bit integers in $O(\log(LN))$ delay (Muller and Preparata 1975). The value of each node v, $\text{value}(v) \in \mathbb{Z}[x_1, \ldots, x_m]$, where the x_i are indeterminates for the leaf values. Since $\|\text{value}(v)\|_1 \leq 2^{d+n}$ (cf. Lemma 2), where $\|\text{value}(v)\|_1$ denotes the sum of the absolute values of the integral coefficients of value(v) as a polynomial in $x_1, \ldots, x_m$, all node values are bounded absolutely by 2^{dl+d+n}. It therefore suffices to compute the node values modulo 2^L with $L = dl + d + n + 1$. Now the procedures Matrix Multiply, Rake, and Shunt add up at most $O(n^2)$ integers with L bits, which by the above reference is doable in $O(\log(dln))$ delay. The remaining multiplications modulo 2^L are doable in $O(\log L)$ delay (Wallace 1964), hence each Phase can be done in delay $O(\log(dnl)$.] Note that this exercise shows that one can compute the determinant of an $n \times n$ matrix with the entries being n-bit integers on a Boolean circuit of depth $O(\log(n)^2)$; this specific result was proven in (Borodin et al. 1983).

3. Related Questions

In this section we survey results and problems that are related to the Circuit Compression algorithm. We shall not give complete proofs for the theorems stated here, but we confine ourselves to sketching the ideas and/or giving the reference to the literature.

3.1. Circuits with Division

In this subsection we only consider circuits over a field K. It is natural to permit division nodes in such circuits. Note that a division node has fan-in two, and that the in-edge for the dividend is distinguished from the in-edge for the divisor. The Gaussian elimination circuit for computing the determinant described in §1.1 is an example of a circuit with divisions. There is the possibility of introducing a division by zero into such circuits. By definition, we exclude circuits that always divide by zero, no matter what input values the leaves assume. For example, the circuit corresponding to the program

$$v_1 \leftarrow x;\ v_2 \leftarrow v_1 - x;\ v_3 \leftarrow 1 \div v_2;$$

is excluded from consideration, since v_2 always assumes the value 0. As values for leaves we consider elements from the algebraic closure of K.[3] Thus, any such circuit must not divide by zero when retaining the leaf values as indeterminates $x_1, \ldots, x_m$, i.e., when evaluating over the field of rational functions $\mathsf{K}(x_1, \ldots, x_m)$. Note that for finite fields K the problem of deciding whether a circuit divides by zero for all leaf values from K is co-$\mathcal{NP}$-complete (Ibarra and Moran 1983).

The first problem concerns circuits with divisions that are used to compute selected outputs $f_i(x_1, \ldots, x_m)$, $1 \le i \le k$, that are polynomials in $\mathsf{K}[x_1, \ldots, x_m]$. Again, the Gaussian elimination determinant circuit falls into this category. A fundamental result, due to Strassen (1973), shows how to avoid the use of divisions for computing all f_i. We first state the theorem and then give an idea for its proof.

Theorem 2. *Let $f_i(x_1, \ldots, x_m) \in \mathsf{K}[x_1, \ldots, x_m]$, $n - k < i \le n$, be k selected polynomial values of an n-node circuit with divisions and that has as leaf values the indeterminates $x_1, \ldots, x_m$. Suppose we are given a bound δ for the algebraic degrees of all f_i, and the values of all nodes for a specific input $x_1 \leftarrow a_1 \in \mathsf{K}, \ldots x_m \leftarrow a_m \in \mathsf{K}$. Note that the occuring divisors must not evaluate to zero at those input points $a_1, \ldots, a_m$. Then we can (on a PRAM) construct the description of a circuit without divisions that also computes all f_i, $n - k < i \le n$, that has*

$$O(n\,\delta \log(\delta) \log(\log \delta)) \text{ nodes,}$$

and that is of formal degree no more than δ. ⊠

The idea behind this construction is similar to our characteristic matrix trick for eliminating divisions from the Gaussian elimination circuit. For $1 \le i \le n$ let $f_i(x_1, \ldots, x_m) \in \mathsf{K}(x_1, \ldots, x_m)$ be the rational function values of all nodes. We consider the auxiliary functions

$$g_i(y_1, \ldots, y_m, z) = f_i(y_1 z + a_1, \ldots, y_m z + a_m). \tag{11}$$

[3] If K is the field of rational numbers, the algebraic closure of K is contained in the field of complex numbers.

Since $g_i(0, \ldots, 0, z) = f_i(a_1, \ldots, a_m) \in \mathsf{K}$, the coefficient $c_{i,0}$ of the constant term of g_i as an extended power series in z over $\mathsf{K}(y_1, \ldots, y_m)$,

$$g_i(y_1, \ldots, y_m, z) = \sum_{j=-l_i}^{+\infty} c_{i,j}(y_1, \ldots, y_m) z^j, \quad l_i \geq 0, \tag{12}$$

is an element in K and is given as input. Furthermore, if the i-th node is the divisor of another node, then $c_{i,j} \neq 0$, for otherwise the circuit evaluated at $a_1, \ldots, a_m$ would divide by zero. We also remark that for $n - k < i \leq n$ the coefficients $c_{i,j}$ are equal 0 for all $j < 0$ or $j > \delta$, because the corresponding f_i are polynomials of degree no more than δ and the extended power series representation (12) is a canonical, thus unique, form.

It now follows easily by induction on the depth of the input circuit that no g_i has terms in z of negative order. That is, for all $j < 0$ we have $c_{i,j} = 0$; and it follows that for all $j > 0$ the $c_{i,j}(y_1, \ldots, y_m)$ are polynomials in $\mathsf{K}[y_1, \ldots, y_m]$. The division-free output circuit of the above theorem computes the $c_{i,j}(y_1, \ldots, y_m)$ for $1 \leq j \leq \delta$ by truncated power series arithmetic. Finally, the original output polynomials f_i, $n - k < i \leq n$, are computed as

$$f_i(x_1, \ldots, x_m) = f_i(a_1, \ldots, a_m) + \sum_{j=1}^{\delta} c_{i,j}(x_1 - a_1, \ldots, x_m - a_m),$$

which undoes the translations in (11) by setting $z = 1$ and $y_i = x_i - a_i$. The exact circuit transformation operations are descibed in (Kaltofen 1988, §7).

Strassen used this result to prove that divisions do not aid in the construction of asymptotically fast $n \times n$ matrix multiplication circuits. In that setting the f_i are the n^2 row-column inner products, whose algebraic degree is two. In light of the Circuit Compression algorithm, which excludes division nodes, the method has been scrutinized again. Take the Gaussian elimination circuit (4), for instance. A point at which the circuit avoids zero-division is the identity matrix I_n. We certainly know the values of all $a_{j,k}^{(i)}$ for that matrix. Thus, by combining Theorem 2 and Corollary 1 we have found a division-free circuit for the $n \times n$ determinant that has depth $O(\log(n)^2)$ and $n^{O(1)}$ many nodes.[4] Note that the method generates a circuit of formal degree no more than δ (cf. Exercise 2). This transformation also shows that if one is given a bound for the algebraic degree of a polynomial computed by a circuit of much higher formal degree, that circuit can be transformed into one whose formal degree is no more than the actual degree bound of the polynomial.

There are two obvious questions that one may ask at this point: How does one find, in general, a degree bound δ or an appropriate point set $a_1 \ldots, a_m$ and the corresponding node values? And, what if the circuit computes a rational function in $\mathsf{K}(x_1, \ldots, x_m)$ rather than a polynomial? The problem of finding a point that avoids zero-division in a circuit is intimately related to the problem of avoiding the zeros of a multivariate polynomial. A sequential Monte-Carlo randomized polynomial-time algorithm, based on a lemma by Schwartz (1980), (see also (Zippel 1979)) is described in detail in (Kaltofen 1988, Lemma 4.2). We note that

[4] n exponent 'big-oh' of 1 indeed! Plugging in the upper bounds from both constructions we get $O(M(n^4 \log(n) \log(\log n)))$ many nodes in the circuit. This ring-independent construction is due to Borodin et al. (1982).

for finite fields K of small cardinality one can simulate evaluation of the input circuit at values from a sufficiently large algebraic extension by arithmetic in K itself (Kaltofen 1987, Lemma 1) thus keeping the methods of Theorem 2 valid for circuits over any field K. The maximum degree of all output polynomials f_i can be determined by a sequential Monte-Carlo polynomial-time algorithm as well (see Kaltofen (1988, §5), Kaltofen and Trager (1990, §2, Footnote)). However, all known solutions to the first question are sequential, as they require the evaluation of the input circuit at random leaf values.

The second question concerns the division free computation of the polynomial numerator and denominator of the rational function value of a circuit. It is assumed the the numerators and denominators are reduced, i.e., that they do not share a polynomial common factor. This problem was first posed by Strassen in 1973, and was settled by the author of this article in 1985. Note that the solution is needed if the Circuit Compression algorithm were to be applied to circuits computing rational functions rather than polynomials. Using power series expansions in an auxiliary indeterminate, as we did above for polynomial outputs, and Padé approximations to such series, one can obtain the following theorem (see Kaltofen (1988, §8) and Kaltofen and Trager (1990, §3)).

Theorem 3. *Let $f, g \in \mathsf{K}[x_1, \ldots, x_m]$ be two relatively prime polynomials over the infinite field K. Given is an n-node algebraic circuit that computes from leaf values $x_1, \ldots, x_m$ the rational function f/g, and given is an upper bound δ for the degrees of f and g. Then there exists a circuit with*

$$O(n\,\delta + \delta\,\log(\delta)^2 \log(\log\,\delta))$$

many nodes that computes the polynomials cf and g/c where c is a non-zero element of K. $\boxtimes$

As for the first set of problems, this construction of the circuit for f and g can again be realized by a sequential polynomial-time Monte-Carlo randomized algorithm, and remains valid for small finite fields but with a worse asymptotic node count. Moreover, one can also determine the degrees of f and g by a sequential Monte-Carlo polynomial-time algorithm (see the references to the theorem). Now, we can use the methods in Theorem 2 to remove the divisions in the circuit constructed in Theorem 3, and then use the Circuit Compression algorithm to obtain the following far-reaching corollary.

Corollary 3. *Let $f, g \in \mathsf{K}[x_1, \ldots, x_m]$ be two relatively prime polynomials of degree no more than d with coefficients from the field K such that f/g is computed by an algebraic circuit with n nodes. Then f/g can be also computed by an algebraic circuit with $(nd)^{O(1)}$ many nodes and with depth $O(\log(nd)^2)$.* $\boxtimes$

3.2. Reducing the Fan-Out

In §2.1 we constructed algebraic circuits of bounded fan-in and shallow depth. There are situations where one wishes to control the maximum fan-out as well. A simple idea is to replace a node of high fan-out by a binary tree of duplication nodes, say addition nodes with fan-in 1 and fan-out 2. As the fan-in is assumed to be bounded, it is easy to see that the number of nodes in the new circuit stays within a constant factor of the original number of nodes. Choosing balanced binary trees might, however, increase the depth of the new circuit asymptotically by a log factor of the maximum original fan-out. Hoover et al. (1984) select non-balanced binary trees for the duplication process in such a way that a distance measure from the source node to all possibly reachable output nodes is optimized (see Figure 4). They can then prove the following theorem.

Theorem 4. *Given is an algebraic circuit with n nodes and of depth t such that no node has fan-in higher than 2. Then there is an algebraic circuit with $O(n)$ nodes and depth $O(t)$ where no node has fan-out higher than 2, and a subset of whose nodes determines the values of all nodes of the input circuit.* $\boxtimes$

Unbounded fan-in Bounded fan-in

Figure 4: Fan-out reduction.

3.3. Partial Derivatives

The circuit transformation of the previous subsection preserved size and depth within a constant factor. We now give an algebraic transformation that also shares this property. Again, let us consider circuits with divisions over a field K. By labeling the input nodes with the indeterminates $x_1, \ldots, x_n$, a selected output node has a rational function $f(x_1, \ldots, x_m) \in \mathsf{K}(x_1, \ldots, x_m)$ as its value. At issue is to construct a circuit that has all the first order partial derivatives

$$\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_m} \tag{13}$$

as a subset of its values. One certainly can compute the value of a single derivative D (defined on $\mathsf{K}(x_1, \ldots, x_m)$) along with the value of each node by inductively applying the addition, product, and quotient rules for D. For example, for the assignment $v \leftarrow u_1 \div u_2$ we construct the circuit fragment for $v' \leftarrow (u_1' - v \times u_2')/u_2$, where the nodes u_1', u_2', and

v' attain the values $D(\text{value}(u))$, $D(\text{value}(v))$, and $D(\text{value}(v))$, respectively. The resulting circuit is of size no more than 4 times the input circuit that also computes $D(f)$. It is Baur's and Strassen's (1983) accomplishment to establish that all partial derivatives (13) of f can be computed in essential the same number of operations. Using their construction in conjunction with Theorem 4, we can actually show that the depth is preserved as well (Kaltofen and Singer 1990).

Theorem 5. *Given be an n node algebraic circuit of depth t over the field K that computes in one of its output nodes the rational function $f \in \mathsf{K}(x_1, \ldots, x_m)$, where $x_1, \ldots, x_m$ are the leaf values. Then there is a circuit with no more than $4n$ nodes and with depth $O(t)$ that computes in a subset of its output nodes f and all first order partial derivatives $\partial f/\partial x_i$, $1 \le i \le m$.* ⊠

Baur's and Strassen's original motivation is still useful in the setting of parallel algebraic complexity. Consider a circuit that computes the determinant $\Delta(x_{1,1}, \ldots, x_{n,n})$ of an $n \times n$ matrix $A = [x_{i,j}]_{1 \le i,j \le n}$ with leaf values $x_{i,j}$. Then by Theorem 5, the inverse of A can be computed by a circuit of the same asymptotic size and depth, namely

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

In fact, one can also establish for Theorem 5 that the circuit for the partial derivatives avoids any zero-division for all those inputs for which the input circuit for f does not divide by zero.

3.4. Exercises

Exercise 14*. Show that the construction by Hoover et al. (1984) for Theorem 4 can be carried out on a PRAM in $O(\log(n)^2)$ time [Hint: see Atallah et al. (1989) for a parallel construction of the optimal duplication trees.]

Exercise 15. Develop a theorem similar to Theorem 5 for circuits that allow fan-in 1 nodes of the form $v \leftarrow \sqrt{u}$, or $v \leftarrow \exp(u)$, or $v \leftarrow \log(u)$ (Morgenstern 1985).

Exercise 16*. Show that if a circuit over the complex numbers of size n with divisions and nodes of the form $v \leftarrow \sqrt{u}$ computes a polynomial f in the leaf values and that has degree d, then there exists a circuit of size $(dn)^{O(1)}$ without divisions and squareroots that also computes f (Lickteig 1990). An example of such a circuit is the method of computing the determinant of an $n \times n$ real matrix A by orthogonalizing the matrix to U, computing the squares of the lengths of the basis vectors as $D = UU^{\mathrm{T}}$, and then finding the determinant as $\mathrm{Det}(A) = \sqrt{\mathrm{Det}(D)}$. Note that the program is valid in an open neighborhood of the identity matrix in Euclidean n^2-space.

4. Recent Progress

Within the last two years, several problems related to the topic discussed here have been solved. The first is the question on the use of commutativity of the multipicative operation in the circuit compression algorithm. It turns out that in non-commutative rings, it is not possible to achieve poly-logarithmic speedup. Nisan (1991) has shown that any circuit C_n that evaluates the function

$$\begin{aligned} f_n(x_1, x_2) &= \sum_{\chi:\{1,\ldots,n\}\to\{1,2\}} x_{\chi(1)} x_{\chi(2)} \cdots x_{\chi(n)} x_{\chi(n)} \cdots x_{\chi(2)} x_{\chi(1)} \\ &= x_1 f_{n-1}(x_1, x_2) x_1 + x_2 f_{n-1}(x_1, x_2) x_2 \end{aligned}$$

of formal degree $2n$ has depth $\Omega(n)$. Kosaraju (1990) gives a similar family of functions over the semiring $\mathcal{L}$ of languages over $\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ with $\oplus$ to be language union and $\otimes$ to be language concatenation, i.e.,

$$L_1 \oplus L_2 := \{w \mid w \in L_1 \text{ or } w \in L_2\}, \quad L_1 \otimes L_2 := \{w \mid w = uv \text{ with } u \in L_1 \text{ and } v \in L_2\}.$$

Any circuit computing the function

$$f_n(x) = \{\mathsf{a}\} \otimes f_{n-1}(x) \otimes \{\mathsf{a}\} \ \oplus \ \{\mathsf{b}\} \otimes f_{n-1}(x) \otimes \{\mathsf{b}\}, \quad f_0(x) = x,$$

over $\mathcal{L}$ must have depth no less than n.

A second problem that has been solved in part is the question of constructing $\log(n)^{O(1)}$ deep circuits with $O(M(n))$ many arithmetic nodes that can compute the determinant of an $n \times n$ matrix; by Theorem 4 one then also can compute matrix inverses and solve linear systems. V. Pan and the author (1991) prove that there exists a randomized algebraic circuit with n^2 inputs, $O(n)$ nodes that denote random (input) elements, and of

$$O(n^\omega \log(n)) \text{ size and } O(\log(n)^2) \text{ depth},$$

with the following property. If the inputs are the entries of non-singular matrix $A \in \mathsf{K}^{n\times n}$, where K is a field of characteristic zero or greater than n, and if the random nodes uniformly select field elements in $S \subset \mathsf{K}$, then with probability no less than $1 - 3n^2/\mathrm{card}(S)$ the circuit outputs the determinant of A. On the other hand, if the random choices are unlucky or if the input matrix is singular, the circuit divides by zero. On non-singular inputs zero-divisions occur with probability no more than $3n^2/\mathrm{card}(S)$. A similar construction can certify the matrix to be singular.

A major set of open problems is concerned with the smallest depth possible to compute selected polynomials. For instance, is it possible to compute the power A^n of an $n \times n$ matrix A by a circuit of size $n^{O(1)}$ and depth $o(\log(n)^2)$, i.e., asymptotically better than $\log(n)^2$? Or, can one establish a lower bound $\Omega(t(n))$, where $\lim_{n\to\infty} \log(n)/t(n) = 0$, for the minimum depth of any polynomial sized circuit that computes A^n? In fact, we know not a single such family of functions over a ring of polynomially bounded algebraic degree that would not be computable by formulas of polynomial size, while being computable by polynomial sized circuits. Some related lower bound references are: (Strassen 1973), (Jerrum and Snir 1982), (Kalorkoti 1985); see also the chapter by von zur Gathen for $\log(n)$-reductions of related problems such as the determinant to matrix power.

Acknowledgement: I wish to express my gratitude to Jeff Ullman and Vijaya Ramachandran for their detailed remarks about this paper, to an anonymous referee and to several of my students in my Design and Analysis of Algorithms Course in the Fall 1990 for spotting misprints, and to my wife Hoang for preparing the figures on MacDraw.

Literature Cited

Abelson, H. and Sussman, G. J., *Structure and Interpretation of Computer Program*; MIT Press, Cambridge, MA, 1985.

Abrahamson, K., Dadoun, N., Kirkpatrick, D. G., and Przytycka, T., "A simple parallel tree contraction algorithm," *J. Algorithms* **10**, pp. 287–302 (1989).

Aho, A., Hopcroft, J., and Ullman, J., *The Design and Analysis of Algorithms*; Addison and Wesley, Reading, MA, 1974.

Atallah, M. J., Kosaraju, S. R., Larmore, L. L., Miller, G. L., and Teng, S.-H., "Constructing trees in parallel," *Proc. 1989 ACM Symp. Parallel Algorithms and Architectures*, pp. 421–431 (1989).

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

Borodin, A., Cook, S. A., and Pippenger, N., "Parallel computations for well-endowed rings and space-bounded probabilistic machines," *Information Control* **58**/1–3, pp. 113–136 (1983).

Borodin, A., von zur Gathen, J., and Hopcroft, J. E., "Fast parallel matrix and GCD computations," *Inf. Control* **52**, pp. 241–256 (1982).

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

Cole, R., "Parallel merge sort," *SIAM J. Comput.* **17**/4, pp. 770–785 (1988).

Coppersmith, D. and Winograd, S., "Matrix multiplication via arithmetic progressions," *J. Symbolic Comput.* **9**/3, pp. 251–280 (1990).

Csanky, L., "Fast parallel matrix inversion algorithms," *SIAM J. Comput.* **5**/4, pp. 618–623 (1976).

Eberly, W., "Very fast parallel polynomial arithmetic," *SIAM J. Comput.* **18**/5, pp. 955–976 (1989).

Gantmacher, F. R., *The Theory of Matrices, Vol. 1*; Chelsea Publ. Co., New York, N. Y., 1960.

Gibbons, A. and Rytter, W., *Efficient Parallel Algorithms*; Cambridge Univ. Press, Cambridge, 1988.

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

Hyafil, L., "On the parallel evaluation of multivariate polynomials," *SIAM J. Comp.* **8**, pp. 120–123 (1979).

Ibarra, O. H. and Moran, S., "Probabilistic algorithms for deciding equivalence of straight-line programs," *J. ACM* **30**, pp. 217–228 (1983).

Jerrum, M. and Snir, M., "Some exact complexity results for the straight-line computations over semi-rings," *J. ACM* **29**/3, pp. 874–897 (1982).

Kalorkoti, K. A., "A lower bound for the formula size of rational functions," *SIAM J. Comp.* **14**, pp. 678–687 (1985).

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).

Kaltofen, E. and Pan, V., "Processor efficient parallel solution of linear systems over an abstract field," in *Proc. 3rd Ann. ACM Symp. Parallel Algor. Architecture*; ACM Press, pp. 180–191, 1991.

Kaltofen, E. and Singer, M. F., "Size efficient parallel algebraic circuits for partial derivatives," *Tech. Report* **90-32**, Dept. Comput. Sci., Rensselaer Polytechnic Inst., Troy, N.Y., October 1990.

Kaltofen, E. and Trager, B., "Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators," *J. Symbolic Comput.* **9**/3, pp. 301–320 (1990).

Karp, R. M. and Ramachandran, V., "Parallel algorithms for shared-memory machines," in *Handbook of Theoretical Computer Science, Algorithms and Complexity (Volume A)*, edited by J. van Leeuwen; Elsevier Science Publ., Amsterdam, pp. 869–941, 1990.

Knuth, D. E., *The Art of Programming, Vol. 2, Semi-Numerical Algorithms, Ed. 2*; Addison Wesley, Reading, MA, 1981.

Kosaraju, S. R., "On the parallel evaluation of classes of circuits," in *Foundations of Software Technology and Theoretical Computer Science*, Springer Lect. Notes Comput. Sci. **472**, edited by Nori, K. V. and Veni Madhavan, C. E.; pp. 232–237, 1990.

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).

Kozen, D., "Fast parallel orthogonalization," *SIGACT News* **18**/2, p. 47 (1987).

Leighton, F. T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees & Hypercubes*; Morgan Kaufmann Publ., San Mateo, California, 1991.

Leighton, T., Leiserson, C. E., Maggs, B., Plotkin, S., and Wein, J., *Advanced parallel and VLSI computation*; Research Seminar Series **RSS 1**; Lab. Comp. Sci., MIT, March 1988.

Lickteig, T. M., "On semialgebraic decision complexity," *Tech. Report* **TR-90-052**, Internat. Computer Sci. Inst., Berkeley, California, September 1990. Habilitationsschrift.

Mayr, E. W., "The dynamic tree expression problem," in *Concurrent Computations Algorithms, Architecture, and Technology*, edited by Tewksbury, S. K., Dickinson, B. W., and Schwartz, S. C.; Plenum Press, New York, pp. 157–179, 1987.

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.

Miller, G. L. and Teng, S.-H., "Dynamic complexity of computational circuits," *Proc. 19th Annual ACM Symp. Theory Comp.*, pp. 254–263 (1987).

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

Muller, D. E. and Preparata, F. P., "Bounds to complexities of networks for sorting and switching," *J. ACM* **22**/2, pp. 195–201 (1975).

Nisan, N., "Lower bounds for non-commutative computation," in *Proc. 23rd Ann. ACM Symp. Theory Comput.*; ACM Press, pp. 410–418, 1991.

Schwartz, J. T., "Fast probabilistic algorithms for verification of polynomial identities," *J. ACM* **27**, pp. 701–717 (1980).

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.

Strassen, V., "Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten," *Numer. Math.* **20**, pp. 238–251 (1973). In German.

Strassen, V., "Algebraische Berechnungskomplexität," in *Anniversary of Oberwolfach 1984*, Perspectives in Mathematics; Birkhäuser Verlag, Basel, pp. 509–550, 1984.

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).

Valiant, L. G. and Skyum, S., "Fast parallel computation of polynomials using few processors," *Proc. ICALP 81, Springer Lect. Notes Comput. Sci.* **118**, pp. 132–139 (1981).

Wallace, C. S., "A suggestion for a fast multiplier," *IEEE Trans. Electronic Comput.* **EC-13**, pp. 14–17 (1964).

Zippel, R. E., "Probabilistic algorithms for sparse polynomials," *Proc. EUROSAM '79, Springer Lec. Notes Comp. Sci.* **72**, pp. 216–226 (1979).