Efficient Parallel Evaluation of Straight-line Code and Arithmetic Circuits*

Gary L. Miller†
Mathematical Sciences Research Institute and
Department of Computer Science
University of Southern California
Los Angeles, CA 90089-0782

Vijaya Ramachandran‡
Mathematical Sciences Research Institute and
University of Illinois
Coordinated Science Laboratory
Urbana, IL 61801-3082

Erich Kaltofen§
Mathematical Sciences Research Institute and
Rensselaer Polytechnic Inst.
Computer Science Department
Troy, NY 12181

Abstract

A new parallel algorithm is given to evaluate a straight line program. The algorithm evaluates a program over a commutative semi-ring R of degree d and size n in time $O(\log n(\log nd))$ using $M(n)$ processors, where $M(n)$ is the number of processors required for multiplying $n \times n$ matrices over the semi-ring R in $O(\log n)$ time.

*Appears in *SIAM J. Comput.*, **17**/4, pp. 687–695 (1988). Preliminary version of this paper appeared in [6].

†Research supported in part by National Science Foundation Grant MCS-800756 A01.

‡Research supported by NSF under ECS-8404866, the Semiconductor Research Corporation under RSCH 84-06-049-6, and by an IBM Faculty Development Award.

§Research Supported in part by NSF Grant DCR-8504391 and by an IBM Faculty Development Award.

1 Introduction

In this paper we consider the problem of dynamic evaluation of a straight line program in parallel. This is a generalization of the result of Valiant et al [10]. They consider the problem of taking a straight line program and transforming it into a program of 'shallow' depth. Their transformation is performed by a sequential polynomial time algorithm. We show how to construct this 'shallow' program with slightly smaller size and the same time bounds on-line, no preprocessing, as their off-line algorithm.

We consider two basically equivalent models of evaluation over a semi-ring: straight line programs and arithmetic circuits. In the introduction we will restrict our discussion to the former model while most of the rest of the paper will deal with the latter model. A *straight line program* over a commutative semi-ring $R = (R, +, \times, 0, 1)$ is a sequence of assignment statements of the form $a \leftarrow b + c$ or $a \leftarrow b \times c$ where b and c are either elements of R or previously assigned variables. The value of a variable is the natural one. We will assume that the semi-ring operations can be performed in unit time. Let $M(n)$ denote the number of processors required to multiply two $n \times n$ matrices in $\log n$ time over the semi-ring R [1, 4].

A special case of a straight line program is a Boolean circuit. Ladner has shown that the Boolean circuit evaluation problem is P-Complete [5]. It is therefore believed that this evaluation problem is not in NC [3]. In this paper, we show that circuits of degree d and size n (we define these terms in Definition 2.3) can be evaluated in time $O(\log n(\log nd))$ using $M(n)$ processors. The crucial difference between this result and the result in Valiant et al. [10] is that our algorithm need not know the degree of the circuit in advance. As a nontrivial application of our procedure we can also compute the degree of a circuit in the above time and processor bounds. This follows because the operations of maximum and sum form a commutative semi-ring over the nonnegative integers. We know of no other parallel algorithm for computing the degree that satisfies the above time and processor bounds.

2 Preliminaries

We view a straight line program as a special case of a more general object, an arithmetic circuit. Our results are more easily applied to arithmetic circuits:

Definition 2.1 *An* arithmetic circuit *is a edge-weighted directed acyclic graph (DAG) (where the weights on the edges are from the semi-ring R) satisfying the following conditions:*

- *Each node is labeled as one of three types: a leaf, a multiplication node, or an addition node.*
- *Leaves are assigned a value in R, denoted $value(v)$ for a leaf v.*

- *The indegree of a leaf node is zero, a multiplication node is two, and an addition node is nonzero.*
- *All edges are directed away from leaves.*
- *There are no edges from multiplication nodes to multiplication nodes.*

Note that any circuit can be modified to satisfy the last condition by simply adding a dummy addition node of indegree and outdegree 1 in the middle of each edge that connects two multiplication nodes. We say an edge is a *plus-plus* edge if it connects two addition nodes. The *size* of an arithmetic circuit U is the number of nodes in U. The *subcircuit evaluating* v, denoted by U_v, is the subcircuit induced by all nodes that are contained on some path to v. A node w is a *child* of v if there exists an edge from w to v. A node of outdegree 0 are called an *output node.*

Definition 2.2 *We define the* value *of each node v in an arithmetic circuit U_v by induction on the size of U_v. The value for a leaf is given by the definition of an arithmetic circuit. If the node v is an addition node with children $v_1, \ldots, v_k$ then the value of v is defined by:*

$$value(v) = \sum_{i=1}^{k} value(v_i) \cdot U(v_i, v)$$

where $U(v_i, v)$ is the weight on the edge from v_i to v. If, on the other hand, v is an multiplication node with children v_1 and v_2, then

$$value(v) = value(v_1) \cdot value(v_2) \cdot U(v_1, v) \cdot U(v_2, v).$$

We will restrict our attention to circuits where any edge entering a multiplication node has weight 1. All the algorithms in this paper preserve this restriction. Thus, the value of the multiplication node v is $value(v_1) \cdot value(v_2)$. The value *of a circuit is a vector of all its node values.*

Given a straight-line program, we obtain its arithmetic circuit by constructing a node for each statement and for each input variable, and an edge from node i to node j if j is a statement that uses the variable evaluated at statement i. All edge weights are set to 1, and nodes corresponding to input variables are given values assigned to the corresponding variables.

Definition 2.3 *The* (algebraic) degree *of a node in an arithmetic circuit is defined inductively: a leaf has degree 1, an addition node has degree equal to the maximum degree of it children, and a multiplication node has degree equal to the sum of the degree of its children. The* degree *of an arithmetic circuit is the maximum over the degree of its nodes.*

3 The Algorithm

In this section we describe our algorithm for arithmetic circuit evaluation. The value of the circuit will be obtained by repeated application of a procedure called *Phase*. This procedure takes as input an arithmetic circuit and returns a new circuit with the same nodes such that every node will have the same value as before. Repeated application of *Phase* will eventually return with the value of the circuit.

In a natural way an arithmetic circuit can be viewed as an upper triangular matrix U with zero diagonal where the entry U_{ij} is the weight on the edge from node v_i to node v_j if the edge exists and it is zero otherwise. We need three submatrices derived from U:

$$U(+,+)_{ij} = \begin{cases} U_{ij} & \text{if } v_i \text{ and } v_j \text{ are addition nodes} \\ 0 & \text{otherwise} \end{cases}$$

$$U(X,+)_{ij} = \begin{cases} U_{ij} & \text{if } v_j \text{ an addition node} \\ 0 & \text{otherwise} \end{cases}$$

$$U(X,X)_{ij} = \begin{cases} U_{ij} & \text{if } v_i \text{ or } v_j \text{ is } \textit{not} \text{ an addition node} \\ 0 & \text{otherwise} \end{cases}$$

The matrix $U(+,+)$ corresponds to the subcircuit containing only plus-plus edges, while $U(X,+)$ corresponds to the subcircuit containing any edge terminating at an addition node, and similarly for $U(X,X)$. We can now define the procedure Matrix Multiply (MM). The procedure uses one matrix multiplication and one matrix addition over the semi-ring R. Thus, it can be performed in $O(\log n)$ time using $O(n^{2.49})$ processors for many semi-rings.

Procedure $MM(U)$
$U \leftarrow U(X,+) \cdot U(+,+) + U(X,X)$

We need two more procedures called Plus Evaluate ($Eval_+$, see Figure 1), and Multiplication Evaluate or Shunt ($Eval_\times$, see Figure 2). The first of these procedures simply evaluates an addition node if all its children have been evaluated. The first part of the second procedure evaluates multiplication nodes if both its children have been evaluated. The new idea is the second part of the procedure which we call *Shunt*. Here we do partial evaluation of a multiplication node when only one of its two arguments has been evaluated, see Figure 3 for an example.

The procedures $Eval_+$, $Eval_\times$ and MM can all be performed on a PRAM in in $O(\log n)$ time. The processor count for MM is the number of processors required for matrix multiplication for the particular semiring. of the circuit. Procedures $Eval_+$ and $Eval_\times$ need only $O(n^2)$ processors. To see that $Eval_\times$ can be performed with $O(n^2)$ processors note that the number of terms in line (*) is at most the number of edges. Thus we simply sort these terms on their key (l, i) using say parallel bucket sort [8] and then sum the terms using parallel list-ranking [7, 11].

It is interesting to point out a strong analogy between the procedures Rake and Compress used to evaluate expression trees, see [7], and our new procedures. One can view $Eval_+$ and $Eval_\times$ as removing the leaves of an arithmetic circuit, i.e., Rake; while Matrix Multiplication, MM, 'compresses' addition chains, a natural generalization of Compress [7].

Procedure $Eval_+(U)$
  for all addition nodes v_j whose children are leaves **do**
    $value(v_j) \leftarrow \sum_{i=1}^{n} value(v_i) \cdot U_{ij}$
    Set v_j to a leaf
    $U_{ij} \leftarrow 0$ for $i \in \{1, \ldots, n\}$
  od

Figure 1: The Procedure Plus Evaluation

Procedure $Eval_\times(U)$
  for all multiplication nodes v_j with children v_k and v_l both of which are leaves
  do
    $value(v_j) \leftarrow value(v_k) \cdot value(v_l)$
    Set v_j to a leaf
    $U_{kj} \leftarrow 0$ and $U_{lj} \leftarrow 0$
  od
  for all U_{ji} where v_j is a multiplication node with children v_k and v_l
      and v_k is a leaf and v_l is not **do**
    $W_{li} \leftarrow \sum_j value(v_k) \cdot U_{ji}$ (*)
    $U_{li} \leftarrow U_{li} + W_{li}$
    $U_{ji} \leftarrow 0$
  od

Figure 2: The Procedure Multiplication Evaluation or Shunt

We combine these three procedures, MM, $Eval_+$, and $Eval_\times$, into a single procedure $Phase$ that we will repeatedly apply until the value of the arithmetic circuit is returned:

Procedure $Phase(U)$
  do
    $U \leftarrow MM(U)$
    $U \leftarrow Eval_+(U)$
    $U \leftarrow Eval_\times(U)$
  od

To show that Phase is correct (sound) it will suffice to prove the following Lemma.

Lemma 3.1 *The procedures MM, $Eval_+$, and $Eval_\times$ applied to an arithmetic circuit return a new circuit with the same value.*

The proof of the Lemma follows by a straightforward proof by induction on the size of U, using the associative, commutative, and distributive properties of R.

In Figure 4 we show the effect of applying the different procedures to a circuit. We represent leaves by square boxes and addition or multiplication nodes by circles. All isolated nodes have been deleted and edge weights have been ignored. We start with the circuit (a) and apply procedure MM obtaining circuit (b). To which circuit (b) we apply procedure $Eval_+$ obtaining circuit (c); which we then apply $Eval_\times$ obtaining circuit (d).

Figure 3: An Arithmetic Circuit after an Application of Procedure $Eval_\times$

4 The Height of an Arithmetic Circuit

In this section we define the height of a node. This notion is the main tool we shall use to analyse the procedure *Phase*. We will show that every application of *Phase* reduces the height of the circuit by a factor of approximately one half. In Theorem 4.2 we will prove an upper bound on the height in terms of the size and the degree of a circuit.

Definition 4.1 *The* **height** *of a node is defined inductively:*

Figure 4: An arithmetic circuit after successive application of the procedures: MM, $Eval_+$, and $Eval_\times$.

1. *A leaf has height 1.*

2. *A multiplication node has height equal to the sum of the heights of its children.*

3. *If v is an addition node then the height of v equals $max(a+1/2, m)$ where a equals the maximum height of any child of v which is an addition node, and m equals the height of any child which is either a leaf or a multiplication node.*

The height *of a circuit U is the maximum height of any node in U.*

We say a child w of an addition node v is *dominant* if either w is a multiplication node and $h(v) = h(w)$ or it is an addition node and $h(v) = h(w) + 1/2$, i. e., the height of w determines the height of v. We can now prove the upper bound on the height of a circuit.

Theorem 4.2 *If U is an arithmetic circuit of degree d and e is the number of plus-plus edges then the height of $U \leq (1/2e \cdot d + d)$.*

Proof: The proof is by induction on the number of nodes n in the subcircuit U_v. We start with subcircuits of size one, leaves. The height of a leaf is one which is clearly less than or equal to $e+1$. Suppose the theorem is true for subcircuits of size $\leq n$. We show the theorem holds for circuits of size $n+1$. Let U_v be a subcircuit with $n+1$ nodes. Let $v_1, \ldots, v_k$ be the children of v having degrees $d_1, \ldots, d_k$ and heights $h_1, \ldots, h_k$, respectively. The subcircuits evaluating $v_1, \ldots, v_k$ are of size $\leq n$. Therefore, by induction $h_i \leq 1/2e'd_i + d_i$, for $1 \leq i \leq k$, where e' is the number of plus-plus edges in U_{v_i}. There are two cases: v is either an addition node or a multiplication node. We treat the two cases separately.

First, suppose that v is a multiplication node. The degree d of v equals $d_1 + \cdots + d_k$ and the height, by induction, is $\leq \sum_{i=1}^{k}(1/2)e'd_i + d_i$ which is equal to $(1/2)e'd + d$. Thus the theorem holds in this case, since $e' \leq e$. Second, suppose that v is an addition node. Again, there are two cases: either a dominant child is an addition node or it is a multiplication node. The most interesting case is the first case. Suppose that v_1 is a dominant addition node, i.e., $h_1 \geq h_i,\ 1 \leq i \leq k$. Here the degree d of v will be greater than or equal to d_1, while the height $h = h_1 + 1/2 \leq (1/2)e'd_1 + d_1 + 1/2 \leq 1/2e'd + d + 1/2$. Since we have at least one new plus-plus edge we know that $e' \leq e - 1$. Thus, $h \leq 1/2(e-1)d + d + 1/2\ = (1/2)ed - (1/2)d + d + 1/2$. Using the fact that $d \geq 1$ we get the desired estimate, $h \leq 1/2ed + d$. □

5 Analysis of the Algorithm

In this section we use the height of a circuit to analyse the number of applications of *Phase* needed to evaluate a circuit of height h. We start by stating and proving the main technical lemma from which the main theorem will follow. Recall that all procedures defined so far take circuits to circuits. They modify the edge structure but map nodes to nodes in a one-to-one way. Thus, we may view the procedures as maps of circuits to circuits which are themselves surjective on nodes. Throughout this section let U be a circuit and U' its image under the transformation *Phase*. Similarly, if v is a node of U then its image under *Phase* will be denoted by v'.

Lemma 5.1 *If U and U' are arithmetic circuits as above and v' is a node of U' which is not a leaf and not an output node then the height of v is at least twice the height of v'.*

Proof: Let v' be a node of U' which is neither a leaf nor an output node. The proof will be by induction on the size of the subcircuit $U'_{v'}$. We begin with the case when all the children of v' are leaves. There are two subcases: either v' is an addition node or it is a multiplication node. First, suppose that v' is an addition node. We must show that height of v is at least 2, where v is the preimage of v'. Suppose by way of a contradiction that the height of v is <2. Now, v cannot be of height 1 because a height 1 node must either be a leaf or all its children are leaves. Thus, one application of $Eval_+$ will transform v into a leaf, a contradiction. If, on the other hand, the height is 3/2 then all the dominant children of v are addition nodes whose children are leaves. Thus, after MM and $Eval_+$ the node v will be a leaf and hence v' will be a leaf. This proves the case when v' is an addition node of height 1.

We next consider the more interesting case when v' is a multiplication node with both its children leaves. It will suffice to show that both children of v have height at least 2. Suppose that one child w has height less than 2. In this case, after MM and $Eval_+$ the node w will be a leaf. Thus after $Eval_\times$ the vertex v will either be a leaf or an output node, depending on whether the other child of v is a leaf or not after $Eval_+$, a contradiction. This proves the initial cases of the induction.

The inductive case for multiplication nodes is rather straight forward. The only difficulty arises when one of the two children of v' is a leaf. We handle this by noting that in the last paragraph we actually proved something slightly stronger. Namely, if v' is a multiplication node which is not an output node and w' is a child of v' which is a leaf then the height of w is at least 2. Thus, induction for the multiplication nodes follows. We have only to prove the induction for addition nodes.

Suppose that v' is an addition node. Let w' be a dominant child of v'. If w' is a multiplication node the theorem follows easily. Thus, we may assume that w' is an addition node. It will suffice to prove the following claim:

Claim: The height of w is $\leq$ the height of v minus 1, i.e., $h(w) \leq h(v) - 1$.

Proof of Claim: Note that both v and w are addition nodes. If there is a path in U from w to v containing two or more edges then the claim follows by the definition of height. Thus the only path from w to v is a singleton edge. But this is a contradiction since procedure MM will then remove this edge and $Eval$ cannot replace it since there is now no paths from w to v. This proves the claim and the Theorem. □

By the last Theorem after $\lceil \log_2 h \rceil$ applications of *phase* to a circuit of height h the resulting circuit will contain only leaves and output nodes. Thus, in one more application of *Phase* (only $Eval_+$ and $Eval_\times$ are needed) all nodes will be leaves; the circuit has been evaluated. With a slightly more careful analysis the number of applications can be bounded by $\lfloor log_2 h \rfloor + 1$. We state this fact as a theorem:

Theorem 5.2 *If U is an arithmetic circuit with height h then after $\lfloor log_2 h \rfloor + 1$ applications of Phase all nodes of U are evaluated.*

The upper bounds given in Theorem 5.2 are optimal for our procedure *Phase*. In Figure 5 we exhibit a circuit C_k, for $k \geq 2$, of height $2^k - 1/2$ which requires 2^k applications of

Phase. It is not hard to see that C_2 requires 2 applications of *Phase*; and the subcircuit evaluating v contained in $Phase(C_{k+1})$ equals C_k, for $k \geq 2$.

Figure 5: The Arithmetic Circuit C_k; A Worst Case Example for *Phase*

We can now prove the main theorem of the paper:

Theorem 5.3 *If U is an arithmetic circuit of degree d and size n then the value can be computed in parallel in time $O(\log n(\log nd))$ using at most $M(n)$ processors.*

Proof: By Theorem 5.2 procedure *Phase* need only be applied $\lfloor logh \rfloor + 1$ times, where h is the height of U. By Theorem 4.2, $h = O(e \cdot d)$. Thus, *Phase* is applied at most $O(\log nd)$ times. Now, each application of *Phase* requires only $\log n$ parallel time. The processor expensive step is the matrix multiplication in MM, which can be performed using $O(M(n))$ processors, at least for sufficiently large n. $\square$

6 Open Questions

We know of no similar results for noncommutative rings. We note that for arithmetic circuits over the ring of $n \times n$ matrix one can expand the matrices operations into the underlying commutative ring operations and apply the methods of this paper.

Extension of this work to rings with division would also be interesting.

References

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] R.P. Brent. The parallel evaluation of general arithmetic expressions. *Journal Assoc. Computing Machinery*, 21(2):201–208, April 1974.

[3] S. A. Cook. Towards a complexity theory of synchronous parallel computation. In *Internationales Symposium uber Logik und Algorithmik zu Enren von Professor Hort Specker*, page , February 1980.

[4] D. Coppersmith and S. Winograd. On the asymptotic complexity of matrix multiplication. *SIAM J. Comput.*, 11(3):472–492, August 1982.

[5] R. E. Ladner. The circuit value problem is log space complete for p. *SIGACT News*, 7(1):18–20, 1975.

[6] Gary L. Miller, Vijaya Ramachandran, and Erich Kaltofen. *Efficient Parallel Evaluation of Straight-Line Code*, pages 236–245. Volume 227 of *Lecture Notes in Computer Science*, Springer-Verlag, 1986.

[7] Gary L. Miller and John H. Reif. Parallel tree contraction and its applications. In *26th Symposium on Foundations of Computer Science*, pages 478–489, IEEE, Portland, Oregon, 1985.

[8] John H. Reif. An optimal parallel algorithm for integer sorting. In *26th Annual Symposium on Foundations of Computer Science*, pages 496–504, ACM, 1985.

[9] L. G. Valiant and S. Skyum. *Fast Parallel Computation of Polynomials Using Few Processors*, pages 132–139. Volume 118 of *Lecture Notes in Computer Science*, Spinger-Verlag, New York, 1981.

[10] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, November 1983.

[11] U. Vishkin. Randomized speed-ups in parallel computation. In *Proc. of the 16th Annual ACM Symp. on Theory of Computing*, pages 230–239, ACM, Washington, D.C., April 1984.