October 1988

UILU-ENG-88-2253
ACT-101

COORDINATED SCIENCE LABORATORY
College of Engineering
Applied Computation Theory

PARALLEL RESTRUCTURING AND EVALUATION OF EXPRESSIONS

D. E. Muller
F. P. Preparata

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-88-2253 ACT #101	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois	6b. OFFICE SYMBOL (If applicable) N/A
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Ave. Urbana, IL 61801	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research and National Science Foundation
	7b. ADDRESS (City, State, and ZIP Code) Office of Naval Research & Nat. Science Found Arlington, VA 22217 1800 G St., N.W. Washington, DC 205
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Joint Services Electronics Program and National Science Foundation	8b. OFFICE SYMBOL (If applicable)
8c. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217 & Washington, DC 20550	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER JSEP - N00014-84-C-0149 NSF - CCR-87-03807
	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

"Parallel Restructuring and Evaluation of Expressions"

12. PERSONAL AUTHOR(S)

Muller, D. E. and Preparata, F. P.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	October 1988	27

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	evaluation of expressions, semiring computations, restructuring of expressions, computational complexity, parallel computation, minimum depth networks

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In this paper we describe a boolean network of size $O(N^2 \log N)$ which accepts a fully parenthesized N-variable expression over a given semiring and produces its value in $O(\log N)$ time. The network consists of two components: a preprocessor and a universal evaluator. The preprocessor computes the destinations of the expression terms and routes them to the correct input terminals of the universal evaluator.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Parallel Restructuring and Evaluation of Expressions[1]

D. E. Muller and F. P. Preparata
University of Illinois at Urbana-Champaign

Abstract

In this paper we describe a boolean network of size $O(N^2 \log N)$ which accepts a fully parenthesized N-variable expression over a given semiring and produces its value in $O(\log N)$ time. The network consists of two components: a preprocessor and a universal evaluator. The preprocessor computes the destinations of the expression terms and routes them to the correct input terminals of the universal evaluator.

1. Introduction

The evaluation of tree-structured expressions is a fundamental computation encountered in several problems. The feasibility of parallel computing has attracted considerable research interest to the restructuring of expressions – typically arithmetic expressions – to speed up their evaluation. While restructuring for parallel evaluation has been the main objective for some time[BB68,M71,MP71,B73,BKM73,B74,KM75,MP76,PM76], more recently attention has focussed on the parallelization of the actual evaluation starting from the original expression itself. Several algorithms have been recently proposed for implementation on the P-RAM model[BV85,MR85,GR86,CV87, CV88, KD88]; the most efficient of these algorithms achieve time $O(\log N)$ for an N-variable expression, and are either optimal, $O(N/\log N)$, or near-optimal, $O(N)$, in the number of processors used.

[1]This work was supported in part by NSF Grant CCR-87-03807 and by the Joint Services Electronics Program under Contract N00014-84-C-0149.

The evaluation of an expression in parallel could be carried out as the combined execution of the restructuring and evaluation tasks. Indeed, in this paper we propose a method consisting of two cascaded phases, i.e. the restructuring of the expression followed by its evaluation. The adopted framework is the boolean network model. Specifically, our network consists of two components: *a universal evaluator*, i.e. a network designed to carry out the evaluation of any expression with at most N variables, and a *preprocessor*, designed to compute the assignments to the terminals of the universal evaluator of the variables and connectives of the given expression. Our results, which combine Theorems 1, 4, 5, and 6 in this paper, are summarized by the following theorem:

Theorem A: *An N-variable expression E over a semiring can be restructured and evaluated in time $O(\log N)$ by an $O(N^2 \log N)$-size boolean network.*

Although the term "semiring" implies that the two operations "+" and "·" may have no inverses, the present scheme permits the inclusion of their inverses "−" and "÷" in the calculation if they do exist.

A result with analogous time performance, but based on an entirely different approach, has been developed by Buss, Cook, Gupta, and Ramachandran [private communication].

2. Expressions

Let E be an expression over a semiring, where all variables are assumed to be distinct. Such an expression will be called a *primitive* expression. Thus, we require only that the algebraic structure to which the variables belong has two associative operations, conventionally "+" and "·", such that "·" distributes over "+", and that there is the additive identity 0. Obviously, this class includes rings, fields, distributive lattices, and boolean algebras.

For concreteness of presentation, we assume E to be an expression over the field of rationals (i.e. the operators are "+" and "·" and their inverses), but specializations to other cases are straightforward (nowhere will commutativity be invoked).

Expression E is thought of as defining a computation tree $T(E)$, and is given as a fully parenthesized string, where (i) a variable a is an *atomic* expression (a), and (ii) given two expressions E_1 and E_2, the string $(E_1 \gamma E_2)$ is an expression with $\gamma \in \{+, \cdot, -, \div\}$.

Example: $((((a_1)\gamma_1(a_2))\gamma_2(a_3))\gamma_3((a_4)\gamma_4(a_5)))\gamma_5(a_6))\gamma_6(a_7))$ is a fully parenthesized expression, with variables $\{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$ and operators $\{\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5, \gamma_6\}$. It is a trivial exercise to show that an expression with N variables (and $N-1$ operators) has $4N-2$ parentheses, i.e. a total of $6N-3$ symbols.

A *term* of an expression is either a variable or an operator. Note that a variable occurs between two facing parentheses "( )" and an operator between two opposing parentheses ") (". The *label* $\lambda(a)$ of a term a is its level in $T(E)$, i.e. the number of edges in the path between the root and the node of the term itself (thus, the root has label 0). It is easily seen that the label of a term is given by: (number of left parentheses to its left) − (number of right parentheses to its left) − 1. Thus, if we associate the integers $+1$ and -1 with each left and right parenthesis of E, respectively, then the labels of the terms are obtained by subtracting 1 from the prefix sums over the subsequence of parentheses of E. It is well-known (see, e.g.[LF80]) that such prefix sums can be computed for an N-variable expression by an $O(N)$-node tree network in time $O(\log N)$.

3. The Universal Evaluator.

The universal evaluator network is based on a restructuring scheme due

to Brent[B73], which we now review.

The variables of an expression are assumed to be of two kinds: *atomic* variables $a_1, \ldots, a_r$ and *free* variables $x_1, \ldots, x_s$. An expression is referred to as an A- or E-expression depending upon whether or not it contains free variables, respectively. Normally we will use the letter "E" for E-expressions and the letter "A" for A-expressions. The *weight* "$|\ |$" of an expression is the number of atomic variables it contains. The symbols θ and ϕ will be used to represent the operator that occurs at level 0 in the tree corresponding to an E- and an A-expression, respectively.

Given a E-expression E with N variables, $2^{J-1} \leq N \leq 2^J - 1$, the *breakpoint* of E is a unique node v of $T(E)$, such that the expression associated with the subtree rooted at v is $(E'\theta E'')$, with $\max(|E'|, |E''|) \leq 2^{J-1} - 1$, and $|E'| + |E''| \geq 2^{J-1}$. If we excise $T(E'\theta E'')$ from $T(E)$ and replace it with a free variable x (see Figure 1), we obtain the tree of a A-expression $A \circ x$ with $|A| = |E| - (|E'| + |E''|) \leq N - 2^{J-1} \leq 2^{J-1} - 1$. We call $A \circ (E'\theta E'')$ the canonical decomposition of E.

Analogously, given a A-expression $A \circ x$ with N atomic variables, $2^{J-1} \leq N \leq 2^J - 1$, the *breakpoint* of $A \circ x$ (a ϕ-breakpoint) is a unique node v of $T(A \circ x)$ such that the expression associated with the tree rooted at v is either $((A' \circ x)\phi E)$ or $(E\phi(A' \circ x))$, with $|A'| \leq 2^{J-1} - 1$, and $|A'| + |E| \geq 2^{J-1}$. Again, if we excise from $T(A \circ x)$ the subtree rooted at v and replace it with a free variable y (see Figure 1), we obtain an A-expression $(A'' \circ y)$, with $|A''| = |A| - (|A'| + |E|) \leq N - 2^{J-1} \leq 2^{J-1} - 1$ (see Figure 1b). We call $A'' \circ ((A' \circ x)\phi E)$, or $A'' \circ (E\phi(A' \circ x))$ as appropriate, the *canonical* decomposition of $A \circ x$.

Brent's scheme is based on the following standard forms for E- and A-expressions:

$$E = \frac{E_1}{E_2}, \quad A \circ x = \frac{A_{11}x + A_{12}}{A_{21}x + A_{22}},$$

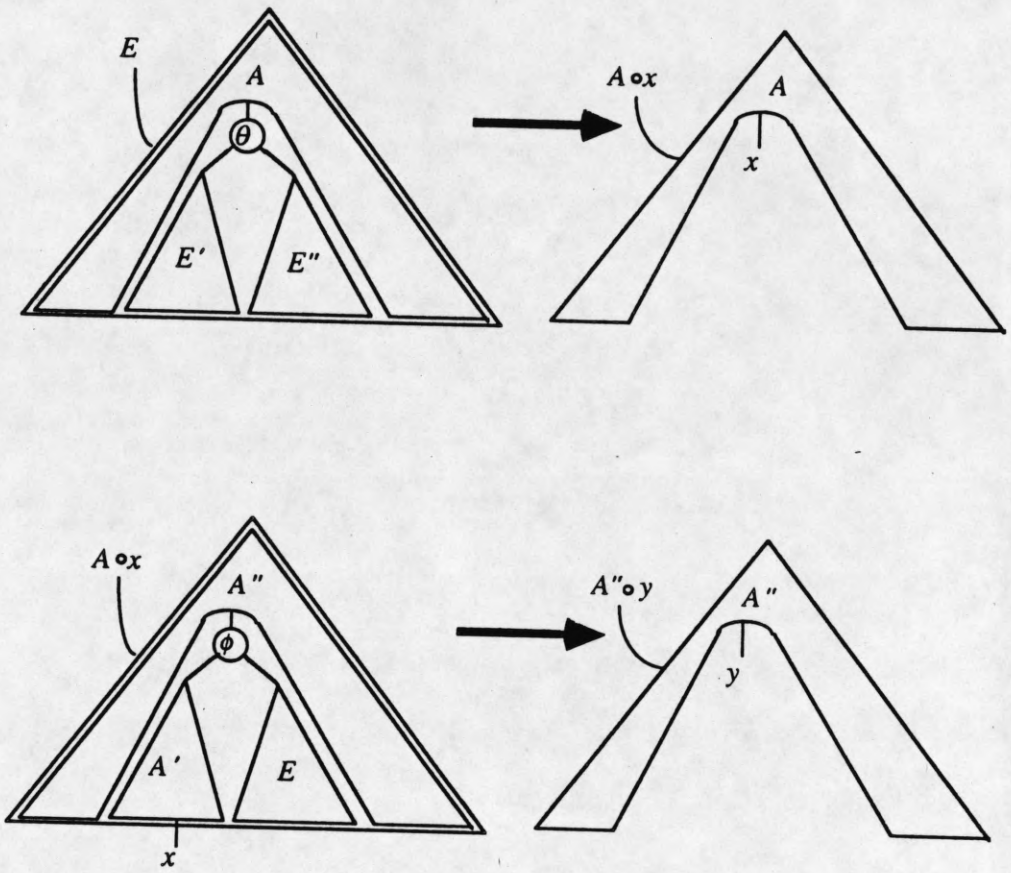

Figure 1: Canonical decompositions of E- and A-expressions.

where $E_1, E_2, A_{11}, A_{12}, A_{21}, A_{22}$ are division-free expressions. An E-expression E is given by the pair (E_1, E_2) and an A-expression $A \circ x$ by the quadruple $(A_{11}, A_{12}, A_{21}, A_{22})$. (Notice that for division-free arithmetic expressions, $E_2 = A_{22} = 1$ and $A_{21} = 0$). The canonical decomposition $E = A \circ (E' \theta E'')$ yields:

$$\begin{aligned} E_1 &= \text{numerator of } \left(A_{11} \left(\frac{E_1'}{E_2'} \theta \frac{E_1''}{E_2''} \right) + A_{12} \right) \\ E_2 &= \text{numerator of } \left(A_{21} \left(\frac{E_1}{E_2} \theta \frac{E_1''}{E_2''} \right) + A_{22} \right) \end{aligned} \tag{1}$$

with $|E| \leq 2^J - 1$ and $|E'|, |E''|, |A'| \leq 2^{J-1} - 1$.
Similarly, the canonical decomposition $A \circ x = A'' \circ ((A' \circ x) \phi E)$ yields:

$$\begin{aligned} A_{11}x + A_{12} &= \text{numerator of } \left(A_{11}'' \left(\frac{A_{11}'x + A_{12}'}{A_{21}'x + A_{22}'} \phi \frac{E_1}{E_2} \right) + A_{12}'' \right) \\ A_{21}x + A_{22} &= \text{numerator of } \left(A_{21}'' \left(\frac{A_{11}'x + A_{12}'}{A_{21}'x + A_{22}'} \phi \frac{E_1}{E_2} \right) + A_{22}'' \right) \end{aligned} \tag{2}$$

with $|A| \leq 2^J - 1$ and $|A'|, |A''|, \leq 2^{J-1} - 1, |E| \leq 2^J - 1$. Notice that Relations (1) and (2) involve no division; this and relation $E = E_1/E_2$ indicate that the evaluation of E involves a single division at the end.

These conditions on the weights of the terms of the decomposition readily establish that the depth of the restructured trees for both E and A are $O(J)$. Moreover, the above decompositions yield the structures of universal evaluators. We shall use script letters "$\mathcal{E}$" and "$\mathcal{A}$" to denote universal evaluators for E- and A-expressions, respectively. (Note the distinction between an expression – letters "E" and "A" – and the corresponding universal evaluator networks – letters "$\mathcal{E}$" and "$\mathcal{A}$".) It must be underscored that a universal evaluator network is itself described by an expression; however, the universal

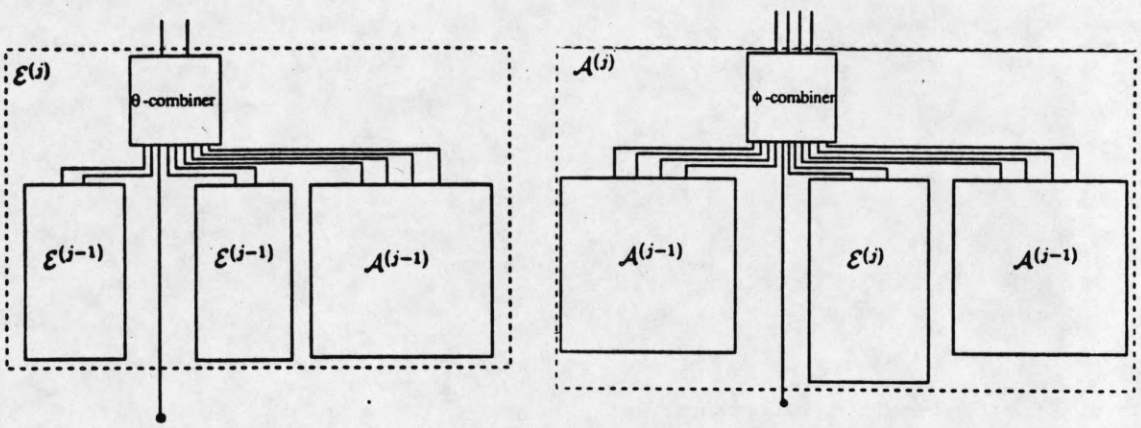

Figure 2: Recursive structures of universal evaluators of E- and A-expressions.

evaluator suited for all expressions of weight not exceeding a fixed bound B has substantially larger weight than B, as we shall see shortly. Specifically, let symbol $\mathcal{E}^{(j)}$ denote the universal evaluator suited for an expression E such that $2^{j-1} \leq |E| \leq 2^j - 1$. Analogously, we define $\mathcal{A}^{(j)}$. The structures of $\mathcal{E}^{(j)}$ and $\mathcal{A}^{(j)}$ are shown in Figure 2, where θ- and ϕ-combiners are respectively fixed-size modules implementing computations (1) and (2). Denoting by e_j and a_j the numbers of input terminals of networks $\mathcal{E}^{(j)}$ and $\mathcal{A}^{(j)}$, respectively, we readily have the following recurrence relations:

$$\begin{cases} e_j &= 2e_{j-1} + a_{j-1} \quad +1 \\ a_j &= 2a_{j-1} + e_j \quad +1 \end{cases}$$

which, combined with the initial values $e_1 = 1$ and $a_1 = 2$ yield

$$e_j = \frac{(4^j - 1)}{3}, \quad a_j = 2\frac{(4^j - 1)}{3} \text{ for } j = 1, 2, \ldots, . \tag{3}$$

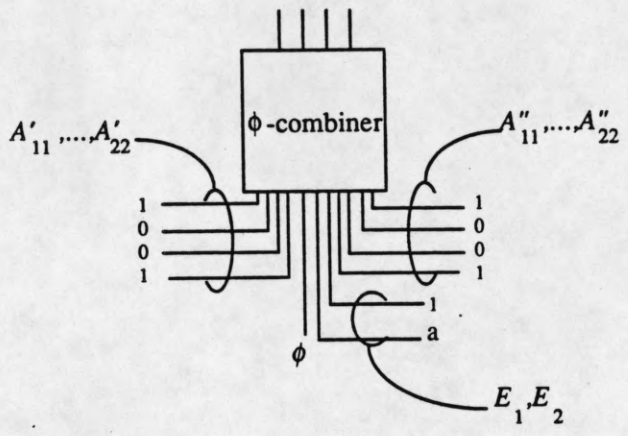

Figure 3: Biasing of $\mathcal{A}^{(1)}$ to evaluate $(x\phi a)$.

Letting $J = \lceil \log_2 N \rceil$, $e_J = (4^{\lceil \log_2 N \rceil} - 1)/3 = \Theta(N^2)$ is the number of input terminals of the universal evaluator for N-variable expressions, with $2^{J-1} \leq N \leq 2^J - 1$.

It is a simple exercise to verify that $\mathcal{E}^{(1)}$ for an input variable a is a trivial circuit consisting of a straight wire carrying the variable and of a second output with the constant bias 1. Network $\mathcal{A}^{(1)}$ for an A-expression $(x\phi a)$, consisting of a single atomic variable a and an operator ϕ, is realized by the biasing of the standard ϕ-combiner shown in Figure 3. It is immediate to verify that the default biasings of the terminals of the universal evaluator are value "0" for a variable terminal and operator "+" for an operator terminal.

We now consider the latency (i.e., the computation time) of the universal evaluator. Let $\delta(\mathcal{E}^{(j)})$ and $\delta(\mathcal{A}^{(j)})$ respectively denote the latencies of $\mathcal{E}^{(j)}$ and of $\mathcal{A}^{(j)}$ and let τ_ϕ and τ_θ respectively denote the delays of the ϕ- and θ-combiners. From the schemes of Figure 2 we deduce:

$$\begin{cases} \delta(\mathcal{E}^{(j)}) & = \max(\delta(\mathcal{E}^{(j-1)}), \delta(\mathcal{A}^{(j-1)})) & +\tau_\theta \\ \delta(\mathcal{A}^{(j)}) & = \max(\delta(\mathcal{A}^{(j-1)}), \delta(\mathcal{E}^{(j-1)})) & +\tau_\phi. \end{cases}$$

We readily have $\delta(\mathcal{A}^{(j)}) = \max(\delta(\mathcal{A}^{(j-1)}),\ \delta(\mathcal{E}^{(j)}) + \tau_\theta,\ \delta(\mathcal{A}^{(j-1)}) + \tau_\theta) + \tau_\phi = \max(\delta(\mathcal{E}^{(j-1)}),\ \delta(\mathcal{A}^{(j-1)})) + \tau_\theta + \tau_\phi$, which yields $\delta(\mathcal{A}^{(j)}) = \delta(\mathcal{E}^{(j)}) + \tau_\phi$ and

$\delta(\mathcal{E}^{(j)}) = \delta(\mathcal{A}^{(j-1)}) + \tau_\theta = \delta(\mathcal{E}^{(j-1)}) + \tau_\phi + \tau_\theta$. From the condition $\delta(\mathcal{E}^{(1)}) = 0$ we have

$$\delta(\mathcal{E}^{(J)}) = (J-1)(\tau_\phi + \tau_\theta) = (\lceil \log N \rceil - 1)(\tau_\phi + \tau_\theta).$$

(Here and in what follows, all logarithms are assumed to be base 2.) We also note that delays can be trivially inserted so that $\mathcal{E}^{(J)}$ can be used in pipeline fashion.

We summarize the preceding discussion with the following theorem:

Theorem 1. The universal evaluator for an N-variable expression has depth $O(\log N)$ and size $O(N^2)$.

4. Parallel restructuring

In this section we discuss the overall scheme whereby the terms of a given expression E (variables and operators) are applied to the terminals of the universal evaluator $\mathcal{E}^{(\lceil \log |E_w| \rceil)}$. A flow-diagram of the process outlined in the introduction is given in Figure 4. The expression E is applied at first to the *preprocessor*, which computes for each term a of E an integer $\rho(a)$, called the assignment of a. Subsequently, for each term a, the pair $(\rho(a), a)$ is formed. The router directs term a to the terminal of index $\rho(a)$ of the universal evaluator. For brevity, the same denotation will be used for a term of E and for the corresponding node of $T(E)$.

Definition. Let w be a node of $T(E)$, and E_w the expression represented by the subtree of $T(E)$ rooted at w. In $T(E)$ a *left* (resp. *right*) ancestor of a node v is any node such that v belongs to its right (resp. left) subtree. For v in $T(E_w)$, we define $\rho(v|w)$ – the *assignment of v relative to w* – as the index of the terminal of $\mathcal{E}^{(\lceil \log |E_w| \rceil)}$ to which term v should be applied. We

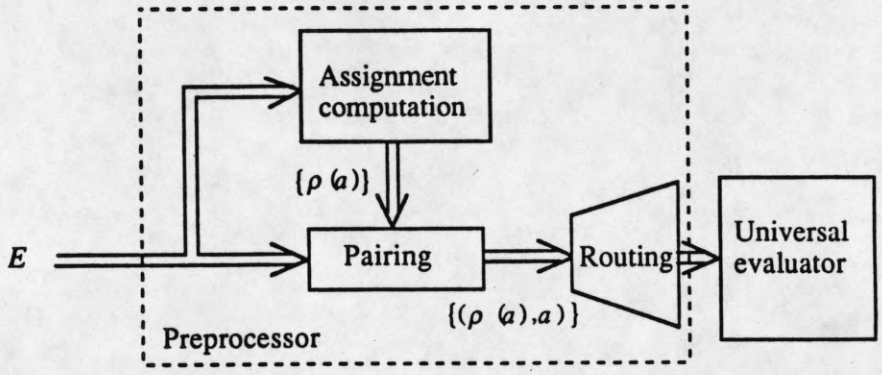

Figure 4: Flow diagram of the overall restructuring/evaluation process.

also define $\rho(v|root(T(E))) \triangleq \rho(v)$, the *(absolute) assignment* of v.

We now analyze the calculation of $\rho(\quad)$. The structure of the universal evaluator $E^{(J)}$ for an expression with at most $(2^J - 1)$ variables exhibits the following properties:[2]

(i). In the canonical decomposition $A \circ (E'\theta E'')$, the component expressions appear in the left-to-right order $E', E'', A \circ x$. Moreover, we assume $|E'| \geq |E''|$.

(ii). In the canonical decomposition $A'' \circ (E\phi(A' \circ x))$ or $A'' \circ ((A' \circ x)\phi E)$, the component expressions appear in the left-to-right order $A' \circ x, E$, $A'' \circ y$.

Lemma 1. If the orders of the positions of the terms in E and of their assignments to the terminals of $\mathcal{E}^{(J)}$ are to be consistent, then the original

[2]If the operations are not commutative, then we introduce new denotations for the original operations with the operands in reverse order.

computation tree $T(E)$ must be rearranged to satisfy the following conditions:

1. Each free variable is the left child of its parent;
2. At each ϕ-breakpoint the A-expression is associated with the left subtree.
3. At each ϕ-breakpoint the heavier subexpression E' is associated with the left subtree.

Proof: We say that a free variable is *terminal* if it was generated by the removal of an expression of the form $(E'\theta E'')$. *Proof of (1)*: If the free variable x is terminal, then by Property (i) the expression $(E'\theta E'')$ is to the left of $A \circ x$, which implies that each term of A is to the right of $(E'\theta E'')$, i.e., x is a left child. If the free variable y is not terminal, then by Property (ii), the expression $E\phi(A' \circ x)$ – or $(A' \circ x)\phi E$ – appears to the left of $A'' \circ y$, which, again implies that y is a left child. *Proof of (2)*: An immediate consequence of Property (ii), by which $(A' \circ x)$ appears to the left of E. *Proof of* (3): Trivial, by Property (i). □

We claim that the tree $T'(E)$ obtained by rearranging the original $T(E)$ in order to comply with Conditions (1), (2), and (3) above, *the left subtree of each* internal node is at least as heavy as the right subtree. This is trivially so for a ϕ-breakpoint. For a ϕ-breakpoint, the left subtree is, by (2), an A-expression of the form $A' \circ x$. We distinguish two cases: (a) $|A'| = 0$, i.e., the A-expression reduces to a free variable x. In this case x is terminal and originated by a decomposition of the form $A \circ (E'\theta E'')$ with $|E'| + |E''| > |A|$. Since the left subtree of the parent of x in $T'(E)$ has weight at most $|A|$, the claim holds. (b) $|A'| > 0$. In this case we have the situation illustrated in

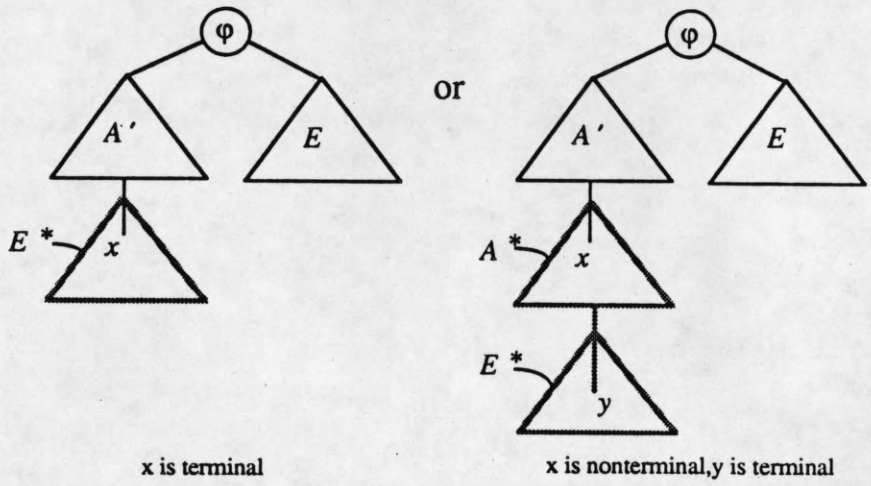

Figure 5: Illustration of the chaining of free variables.

Figure 5. If x is terminal, then the expression of the tree rooted at node ϕ is $(A' \circ E^*)\phi E$, with $|E^*| \geq |A'|+|E|$ (by the selection of θ-breakpoints) and thus $|A'| + |E^*| > |E|$. If x is not terminal, then there is a A-expression $(A^* \circ y)$, with y terminal, such that the expression of the tree rooted at node ϕ is $A' \circ (A^* \circ E^*)\phi E$, with $|E^*| \leq |A'|+|A^*|+|E|$, and thus $|A'|+|A^*|+|E^*| > |E|$. This establishes the claim.

Referring to the rearranged tree $T'(E)$, for each node v we define two companion nodes $r(v)$ and $l(v)$ as follows:

$r(v)$: the farthest right ancestor of v such that each node in the path from v to $r(v)$ is a right ancestor of v. (Note, $r(v)$ always exists and may coincide with v.)

$l(v)$: the closest left ancestor of v, i.e. $l(v)$ is the parent of $r(v)$. (Note, $l(v)$ may not exist, in which case we say $l(v) = \epsilon$, the empty node.)
We then have

$$\rho(v) = \rho(v|r(v)) + \rho(l(v)).$$

Indeed v is on the leftmost path of the computation tree $T(E_{r(v)})$, and

$\rho(v|r(v))$ is the assignment term v would receive in $T(E_{r(v)})$; moreover, v is in the right subtree of $l(v)$, so that the assignment of v must be offset by $\rho(l(v))$. Iterating the above formula we obtain

$$\rho(v) = \rho(v|r(v)) + \sum_{w \in \Lambda(v)} \rho(w|r(w)) \qquad (4)$$

where $\Lambda(v)$ is the set of left ancestors of v in $T'(E)$.

Formula (4) reduces our problem to the calculation of $\rho(v|r(w))$ for an arbitrary v in E. The value of $\rho(v|r(v))$ depends exclusively on the weights $L(v)$ and $R(v)$ of the left and right subtrees of v, respectively. Indeed, in the semiclosed interval $(\max(L(v), R(v)), L(v) + R(v)]$ there is a unique integer of the form $p2^q$ with largest value of q. Let $1b_{s-2} \ldots b_{q+1}10 \ldots 0$ be the binary spelling of $p2^q$.

Suppose at first that $p = 1$ (i.e., $s - 1 = q$). Term v is a θ-breakpoint, and its terminal of the universal evaluator occurs immediately to the right of a subnetwork $\mathcal{E}^{(s-1)}$, i.e., $\rho(v) = e_{s-1} + 1$.

When $p > 1$, term v is a ϕ-breakpoint, and its terminal of the universal evaluator occurs after the following sequence of subnetworks (refer to Figure 2):

1. $\mathcal{E}^{(s-1)}$, a terminal for a ϕ-operator, $\mathcal{E}^{(s-1)}$ (for a total of $2e_{s-1} + 1$ terminals).

2. For $j = s - 2$ down to $q + 1$: $\mathcal{A}^{(j)}$, a terminal for a ϕ-operator, $\mathcal{E}^{(j+1)}$, (for a total of $a_j + e_{j+1} + 1$ terminals) if and only if $b_j = 1$.

3. $\mathcal{A}^{(q)}$.

Therefore

$$\rho(v|r(v)) = (2e_{s-1} + 1) + \sum_{j=q+1}^{s-2} b_j(a_j + e_{j+1} + 1) + a_q + 1$$

Using equalities (3) we obtain

$$\begin{aligned}
\rho(v|r(v)) &= 2\cdot\frac{4^{s-1}-1}{3}+1+\sum_{j=q+1}^{s-2} b_j 2\cdot 4^j + 2\cdot\frac{4^q-1}{3}+1 \\
&= 2\sum_{j=0}^{s-2}4^j + \sum_{j=q+1}^{s-2} 2b_j\cdot 4^j + 2\sum_{j=0}^{q-1}4^j + 2 \\
&= 4^{s-1} - \sum_{j=0}^{s-2} 4j - 1 + \sum_{j=q+1}^{s-2} 2b_j\cdot 4^j + 2\sum_{j=0}^{q-1}4^j + 2 \\
&= 4^{s-1} + \sum_{j=q+1}^{s-2}(2b_j-1)4^j - 4^q + \sum_{j=0}^{q-1}4^j + 1 \\
&= \sum_{j=0}^{s-1}(2c_j-1)4^j + 1 \qquad (5)
\end{aligned}$$

where $c_{s-1}c_{s-2}\ldots c_0$ is the binary spelling of $p2^q-1$.

From a computational viewpoint we note: If $m_{r-1}\ldots m_0$ and $s_{r-1}\ldots s_0$ are the binary spellings of $\max(L(v),R(v))$ and of $L(v)+R(v)$, respectively, then $(s-1)$ is the largest value of j for which $m_j = s_j = 1$, and q is the largest value of j for which $m_j = 0$ and $s_j = 1$. Thus in time $O(\log N)$ we can obtain $c_{s-1}c_{s-2}\ldots c_0$. To obtain $\rho(v|r(v))$ from this number we perform the following transformation

$$c_j \rightarrow d^{(j)}_{2j+2}d^{(j)}_{2j+1}d^{(j)}_{2j} = \begin{cases} 1\ 1\ 1 & \text{if } c_j = 0 \\ 0\ 0\ 1 & \text{if } c_j = 1 \end{cases}$$

and add modulo- 2 bit-by-bit the equally-indexed bits of the corresponding s binary numbers.

The calculation of the subtree weights $L(v)$ and $R(v)$ is the topic of Section 5. In Section 6 we shall address the question of the distribution of the *offsets* $\rho(w|r(w))$ for $w \in \Lambda(v)$. Finally, Section 7 will discuss the routing of v to the terminal of $\mathcal{E}^{(j)}$ indexed $\rho(v)$.

5. Calculation of Subtree Weights

The expression E consists of a sequence of parentheses, variables, and operators. We shall write $\hat{E}$ to denote the subsequence of E formed by erasing the parentheses, leaving just the operators and variables. We note that the members of $\hat{E}$ are in one-to-one correspondence with the nodes of the tree $T(E)$, and we have seen at the end of Section 2 that a label $\lambda(v)$, representing its level in $T(E)$, may be found for each such node v.

Let v be a variable in the expression E, so that v corresponds to a leaf of $T(E)$. There is a unique path from the root of $T(E)$ to v. We write this path as $v_0, v_1, \ldots, v_p$ where v_0 is the root of $T(E)$ and $v_p = v$. For each node v_i in the path $\lambda(v_i) = i$, and if i is less than p, then v_i is an operator.

We now wish to investigate the properties of the subsequence of $\hat{E}$ corresponding to the path $v_0, v_1, \ldots, v_p$ of $T(E)$. We begin with the following:

Lemma 2: In the sequence $\hat{E}$, for any $i < p$ there is no element between v_i and v_p whose level is less than i. Furthermore, if x is any operator or variable of some level $i < p$ such that every element between x and v_p is of level greater than i, then x must be v_i.

Proof: The variable v_p is either in the right or the left subtree of $T(E_{v_i})$ (cf. Definition 1). Let us assume first that it is in the left subtree so that v_p is to the left of v_i in E. Then there is a subsequence of $\hat{E}$ of the form $(E' v_i E'')$ such that v_p occurs in E'. All the operators and variables in E' have level greater than i so we may conclude that all elements of $\hat{E}$ between v_p and v_i have level greater than i. Furthermore, if we assume that x is some variable or operator to the left of v_p in $\hat{E}$ which has level no greater than i, and such that every element of $\hat{E}$ between x and v_p is of level greater than i, then x must be the first operator immediately to the left of $(E' v_i E'')$. Since it must have level less than i it cannot be v_i, and the conclusion follows. Second,

if we assume that v_p is in the right subtree a precisely analogous argument gives the same result. □

Now, starting with v_p we define the *first forward subsequence* $FS(v_p)$ *of* $\hat{E}$, *starting with* v_p *with monotone decreasing levels* to consist of elements $a_1, a_2, \ldots, a_r$ such that (i) $a_1 = v_p$, (ii) for each a_i before the last, the element a_{i+1} is the first member of $\hat{E}$ to the right of a_i having a smaller level than a_i, and (iii) there is no element of $\hat{E}$ to the right of a_r having a smaller level than a_r.

A similar definition describes the *first backward subsequence* $BS(v_p)$ *of* $\hat{E}$, *starting with* v_p *with monotone decreasing levels.* The only difference is that the word "right" replaces the word "left" in all places in the definition. The corresponding subsequence has the form $b_s, b_{s-1}, \ldots, b_1$ and is such that (i) $b_1 = v_p$, (ii) b_{i+1} is always the first member of $\hat{E}$ to the left of b_i having smaller level than b_i, (iii) there is no element of $\hat{E}$ to the left of b_s with smaller level than b_s.

Theorem 2: A path $v_0, v_1, \ldots, v_p$ on $T(E)$ from the root v_0 to a leaf v_p corresponds to $FS(v_p)$ and to $BS(v_p)$.

Proof: Using the previous lemma, we see that the members of the path are precisely the elements satisfying the conditions for being in either the first forward or first backward subsequence starting with v_p. □

If v is a node of the tree $T(E)$ which is not a leaf, then it represents an operator. The subtree $T(E_v)$ of $T(E)$ associated with v corresponds to an expression of the form $(E'vE'')$. The weights $L(v) = |E'|$ and $R(v) = |E''|$ are the weights of the left and right subtrees of $T(E_v)$ respectively. Our objective is to develop a procedure to determine $L(v)$ and $R(v)$ for each operator node

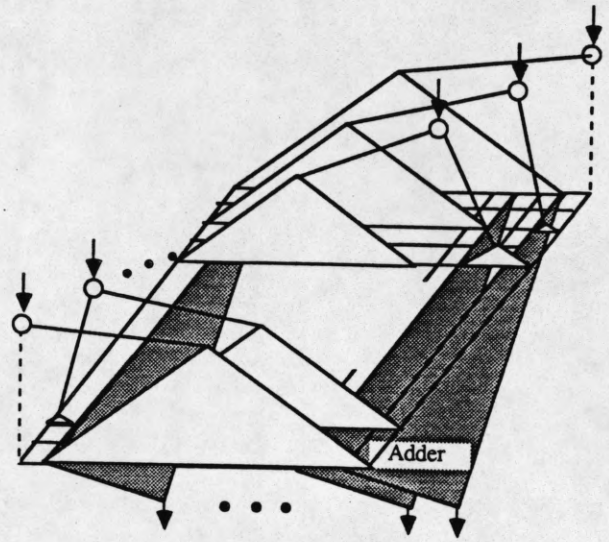

Figure 6: Circuits to compute $(L(v), R(v))$.

v of $T(E)$ which can be carried out in time $O(\log N)$.

The circuit to compute $L(v)$ and $R(v)$ contains a binary tree corresponding to each variable w in E, a typical one of which we shall call T_w. The edges in the tree are serial transfer paths and the leaves receive the members a of $\hat{E}$, with the exception of w, in their given order (see Figure 6). The label $\lambda(a)$ of each member a of $\hat{E}$ is also applied to each corresponding leaf of T_w. These labels are also transmitted to the internal nodes of T_w in such a way that each internal node μ receives the minimum label of all the leaves in its subtree. This is accomplished by sending messages from the leaves to the root of T_w so as to give each internal node μ the minimum label received by its two children. This operation can clearly be carried out in time $O(\log N)$ giving each node μ of T_w a label $\delta = \lambda(\mu)$ by endowing each node with a one-bit comparator and feeding the labels most-significant-bit first.

Two messages called *tokens* are sent from the root of T_w along the left and right edges of T_w connected to the root. These tokens have the form (L, δ) and (R, δ), where L signifies "left" and R "right" and δ is the label $\lambda(w)$. We

now trace the behavior of the left token (L, δ) as it travels along the edges of T_w towards the leaves. An analogous behavior will hold for (R, δ), but with "left" and "right" interchanged.

When starting out from the root, if the first node encountered by (L, δ) (i.e. the left child of the root) has label no less δ, then the token is erased, otherwise, it proceeds as follows.

After leaving the root of T_w, the token (L, δ) passes on a path along the edges of T_w encountering various nodes until it finally comes to a leaf, where it stops. We shall see that this leaf of T_w must correspond to an operator v such that w is in the right subtree of $T(E_v)$. It will therefore contribute 1 to the quantity $R(v)$. The way that the token (L, δ) chooses the correct path is now described.

Since T_w is a binary tree, all its internal nodes are of degree 3 except the root which is of degree 2. When the token (L, δ) reaches an internal node μ of degree 3 it takes either the left or the right descending branch according to the following rule, where the labels on the left and right children of μ are δ_L and δ_R, respectively.

Rule for left-token propagation

1.**begin if** $\delta_R < \delta$ **then**
2. **begin** (L, δ) proceeds on the right edge;
3. **if** $\delta_L < \delta_R$ **then** (L, δ_R) is created and sent on the left edge
4. **end**
5. **else** (L, δ) proceeds on the left edge
end

Note that token (L, δ_R) created at Line 3 follows the same rule as the former token except that, since it has a different label (namely δ_R instead of δ), its interactions will be correspondingly different. An analogous Rule

holds for (R, δ) with the following substitutions: $\delta_R \rightarrow \delta_L, R \rightarrow L$, left $\rightarrow$ right , right $\rightarrow$ left .

Naturally, all the generated tokens as well as the original two tokens are strings of $O(\log N)$ bits that propagate simultaneously in T_w in bit-serial fashion. Since they all move on paths of the binary tree, they do not retrace their paths and hence the time required is determined by the sum of the length of their representation, which is $O(\log N)$, and of the maximum path length from the root to a leaf of the binary tree, which is also $O(\log N)$.

It remains to prove the following.

Theorem 3: The leaves of T_w which receive tokens are exactly those which correspond to operators on the path of $T(E)$ from w to it root. Furthermore, those receiving a left token have w in their right subtrees (are left ancestors of w), and those receiving a right token have w in their left subtrees (are right ancestors of w).

Proof: We shall show that those leaves in the left subtree of T_w which receive left tokens correspond to just the members of $BS(w)$. An analogous result applies to the right subtree. Then, using Theorem 2 the result is proved.

There are three parts to the proof. First, we show that the original token (L, δ) goes to the leaf of T_w corresponding to the first member of $\hat{E}$ to the left of w having level less than δ. To show this, we imagine that each edge of T_w has a label $\lambda(a)$ which is the same as the label of the child node a to which it connects. We now trace the path from the root of T_w in the left subtree taken by the original token (L, δ). It can never follow an edge with label as large as δ, but otherwise it will always go to the right if possible until it reaches a leaf. All parts of the subtree which are to the left of this path must have labels which are greater than or equal to δ, by the mechanism that assign labels to the internal nodes of T_w. On the other hand, the label

of the leaf reached by the token must be less than δ since it lies on the path. This proves our first assertion.

Second, we show that any new token (L, δ') generated in the process must go to one of the members of $BS(w)$. At the point of its initiation, the token (L, δ') cannot have passed to the right of any part of the left subtree with label less than δ', because the tokens from which it was generated had labels greater than δ'. Following its initiation, the token (L, δ') follows a path such that all parts of the subtree which are to the right of this path must have labels at least as large as δ'. Thus, the leaf it reaches is the first one with label less than δ' and is thus on $BS(w)$.

Third, we show that every member of $BS(w)$ must receive a token. Let the operator a with level λ be a member of this subsequence. From the definition, we know that $\lambda < \delta$ and that all leaves of the left subtree of T_w which occur to the right of a have level greater than λ. Tracing a path on T_w from a to the root, we see that this entire path has labels at most λ, but that tributaries to the right of this path have labels which are greater than λ. Now, consider a token starting out on this path. In the beginning it has label δ and as long as the path goes to the right it will retain this label and follow the path. Now, if the path goes left, it either retains the label δ or else obtains a new label $\delta' < \delta$. However, $\delta' > \lambda$ because the path is ultimately connected to the leaf of a. Thus, a left token will ultimately reach a.

Now, by Theorem 2: we see that the leaves of T_w reached by tokens are exactly those corresponding to those operators on the path of $T(E)$ from its root to the leaf corresponding to w. Also, those receiving left tokens have w in their right subtrees and those receiving right tokens have w in their left subtrees. □

The final steps for computing $L(v)$ and $R(v)$ for each operator v of E

must be carried out by adding the numbers of left and right tokens received by each v. This can be done by using a separate pair of adders for each operator. Such an adder is in effect a parallel counter using N single-bit inputs corresponding to the N variables, and can be constructed as a binary tree of depth $\log_2(N)$. The computation time is $O(\log N)$. In Figure 6 we have a global illustration of the machinery implementing the computation of $L(v)$ and $R(v)$ for all internal nodes v of $T(E)$.

Combining the foregoing discussion with the results of Section 4 on the conversion of $L(v)$ and $R(v)$ to $\rho(v|r(v))$, we have the following theorem:

Theorem 4. The computation of the relative assignments $\rho(v|r(v))$ for each term v of N-term expression E can be done by a boolean network of size $O(N^2)$ in time $O(\log N)$.

6. Distribution of the Offsets

The last step needed to calculate the second term in the expression for $\rho(v)$, (formula (4)), for each vertex v of $T(E)$, requires forming the sum $\sum \rho(w|r(w))$ over all the ancestors w of v in $T(E)$ such that v is in the lighter subtree of w.

To carry out this calculation we can use the same structure that was used to calculate the weights $L(v)$ and $R(v)$ and illustrated in Figure 6. Again, we let $\delta \triangleq \lambda(w)$. Tokens of the form (L, δ) and (R, δ) are sent from the root of each tree T_w, but in this case the rules obeyed by the tokens are different from those described in Section 4. (Notice that the token labels L and R are used here to aid the explanation but need not be implemented.) We also assume that the functions $L(w)$ and $R(w)$ have already been calculated.

Now, for each node v, we wish to add the offset of w provided that v is in the lighter subtree of w. We begin by comparing $L(w)$ with $R(w)$ to

determine which of these two numbers is smaller and in this way decide into which of the two subtrees of the root of T_w to send the token.

In order to send the token to only those leaves of T_w which correspond to descendants of w in $T(E)$, we must choose the rules suitably. For concreteness, let us assume that $R(w) < L(w)$, in which case the token (R, δ) enters the right subtree of the root of T_w. Then all the leaves of T_w corresponding to nodes v of $T(E)$ that must receive the offset $\rho(w|r(w))$ occur in a consecutive sequence at the left of this subtree. Specifically, this sequence of leaves is bounded on the right by a leaf μ corresponding to a (right) ancestor of w, which is associated with a label $\lambda' < \delta$. It follows that the token must be distributed exactly to the left subtrees of the leaf-to-root path in T_w originating in μ. Therefore we have the following rules:

(i). If the descending token (R, δ) enters a vertex of T_w whose label is less then δ, then the token follows the branch to the left child.

(ii). If the descending token enters a vertex whose label is not less than δ, then the token is duplicated and both children receive (R, δ).

(Note: (1) It is not possible for the label of a vertex reached by this process to be the same as that of w. (2) If the leaf μ exists, there must be at least one leaf to its left in the right subtree of T_w. Therefore, μ can never receive a token.)

An entirely analogous set of rules applies to the original token (L, δ), which is created when $L(w) \leq R(w)$. In this case (L, δ) is sent into the left subtree and the above rules are used with "right" and "left" interchanged in all places.

Finally, all the tokens will reach leaves of T_w which represent vertices in the lighter subtree of $T(E)$ whose root is w. To each of these leaves we

attach the offset $\rho(w|r(w))$ calculated by the method described in Section 4 and given by Formula (5).

The last step consists of adding, for each term v of $\hat{E}$, the offsets obtained from all the trees T_w. This may be done, again in time $O(\log N)$, by the same adders which were used for computing the functions $L(v)$ and $R(v)$, as described in Section 5. (Note that in this case each adder tree functions as a full-fledged adder of $O(N)$ integers of $O(\log N)$ bits.) The result is the assignment value of the $\rho(v)$ according to formula (4).

We summarize the discussion as follows:

Theorem 5. The computation of the (absolute) assignment $\rho(v)$ for each term v of an N-term expression E can be done by a boolean network of size $O(N^2)$ in time $O(\log N)$.

We next see how these assignments are used to accomplish the routing of the terms of $\hat{E}$ to the universal evaluator.

7. Routing to the universal evaluator

Once the set of N integers $\{\rho(a) : a \text{ a term of } E\}$ is available, the pairs $(\rho(a), a)$ are formed and supplied to a routing network, where $\rho(a)$ functions as the *address* of *record* a. As usual, $J = \lceil \log N \rceil$.

Let $\mathcal{B}_{2^s}$ (for an integer s) denote the 2^s-input/2^s-output butterfly network. The terminals of $\mathcal{B}_{2^s}$ are numbered from 0 to $2^s - 1$ from left to right and the stages of $\mathcal{B}_{2^s}$ are numbered from $s-1$ to 0 from input to output. Given an integer $r \in [0, 2^s - 1]$, we let $\mathrm{BIT}_j(r)$ be the coefficient of 2^j in the binary representation of r. Suppose that the (address, record) pair (r, R) is applied at any input terminal of $\mathcal{B}_{2^s}$; we say that R is *obliviously* routed to output terminal r if at stage j record R is routed on the right or on the left outgoing branch depending upon whether $\mathrm{BIT}_j(r) = 1$ or 0, respectively. We have

the following lemma:

Lemma 3. Let $(r_0, r_2, \ldots, r_{p-1}), p \leq 2^s$, be a sequence of distinct integers in the range $[0, 2^s - 1]$, sorted in ascending order. Pair (r_i, R_i) is applied to input terminal $(c + i) \bmod 2^s$ of $\mathcal{B}_{2^s}$ (for some fixed $c \in [0, 2^s - 1]$), and R_i is obliviously routed. Then the routing paths of the p records are vertex disjoint.

Proof: Sequence $(r_0, \ldots, r_{p-1})$ applied to B_{2^s} as in the statement of the Lemma is said to be well-positioned in $\mathcal{B}_{2^s}$. To prove the lemma, it suffices to show that the oblivious routing through Stage $(s-1)$ is free of collisions and yields two sequences $(r_0, \ldots, r_k)$ and $(r_{k+1}, \ldots, r_{p-1})$, with $r_k < 2^{s-1} \leq r_{k+1}$, which are respectively well-positioned in the left subnetwork $\mathcal{B}_{2^{s-1}}$ and right subnetwork $\mathcal{B}_{2^{s-1}}$ that are obtained by removing Stage $(s-1)$ from B_{2^s}.

A collision may occur only between two elements of the input sequence applied to two terminals of $\mathcal{B}_{2^s}$ situated 2^{s-1} positions apart. It is immediately realized that no collision occurs for $p \leq 2^{s-1}$, so we consider $p > 2^{s-1}$. If $c \leq 2^s - p - 1$, then $\mathrm{BIT}_{s-1}(r_i) \neq \mathrm{BIT}_{s-1}(r_{i+2^{s-1}})$ (for any $i = 0, \ldots, p-1$), for, otherwise $\mathrm{BIT}_{s-1}(r_i) = \mathrm{BIT}_{s-1}(r_{i+1}) = \ldots = \mathrm{BIT}_{s-1}(r_{i+2^{s-1}})$ because $(r_0, \ldots, r_{p-1})$ is sorted. But this implies that there are at least $2^{s-1} + 1$ distinct integers in $[0, 2^s - 1]$ with identical most-signficant bit, which is false. An analogous argument holds when $c > 2^s - p - 1$, thus establishing the first part of the lemma.

To prove the second part, we consider the case $c + p - 1 \leq 2^s$, the other case being analogous. If neither interval $[c, c + k]$ or $[c + k + 1, c + p - 1]$ contains $2^{s-1} - 1$, then $(r_0, \ldots r_k)$ and $(r_{k+1}, \ldots r_{p-1})$ are each applied as a single segment in the left and right half, respectively. Otherwise, one of them, say $(r_0, \ldots r_k)$, is split into segments $(r_0, \ldots, r_{2^{s-1}-c-1})$ and $(r_{2^{s-1}-c}, \ldots, r_k)$, which are jointly applied to form a well-positioned sequence in the left half;

the other sequence is applied as a single segment in the right half. In all cases we obtain well-positioned sequences in the left and right $\mathcal{B}_{2^{s-1}}$ subnetworks. □

To carry out the routing, we could sort the set $\{(\rho(a), a) : a \text{ a term of } E\}$ in ascending order by $\rho(a)$ and apply the sorted sequence to the leftmost segments of inputs of the appropriate butterfly network for oblivious routing. The latter is $\mathcal{B}_{2^{2J-1}}$, since, for $J \geq 1, 2^{2J-2} < (4^J - 1)/3 < 2^{2J-1}$ and therefore each $\rho(\ )$ is in the range $[0, 2^{2J-1} - 1]$. However, some pruning of $\mathcal{B}_{2^{2J-1}}$ is possible, since at most $2^{J+1} - 3$ terminals are used at the input of Stage $(2J-2)$, $2(2^{J+1} - 3)$ at the input of Stage $(2J-3)$, and so on, until we reach Stage J, where more than 2^{2J-2} inputs are used. In Stages $(2J-2), (2J-3), \ldots, J$ we will remove from $\mathcal{B}_{2^{2J-1}}$ all nodes that are not reachable by any of the input terminals with index $\geq 2^{J+1} - 4$. In the subsequent Stages $J-1, J-2, \ldots, 0$ we will remove all nodes that are not reachable by any of the output terminals with index $\geq e_J = (4^J - 1)/3$. We leave it as an exercise to show that the number of branching nodes of the pruned $B_{2^{2J-1}}$ is $((3J-1)2^{2J} + 1)/9 + 2^{2J} - 3.2^{J-1} - 6 = O(N^2 \log N)$. The routing is obviously accomplished in time $O(\log N)$. Since the preliminary sorting can be done in time $O(\log N)$ by a mesh-of-trees [MP75,L84] with $O(N^2)$ leaves we conclude:

Theorem 6. Routing of the expression terms to the terminals of the universal evaluator can be done in time $O(\log N)$ with equipment of size $O(N^2 \log N)$.

References

[B73] R.P. Brent, "The parallel evaluation of arithmetic expression in logarithmic time," in *Complexity of Sequential and Parallel Nu-*

merical Algorithms, 83-102, Academic Press, N.Y., 1973.

[B74] R.P. Brent, "The parallel evaluation of general arithmetic expressions," *J. ACM*, 21, 2 (April 1974) 201-206.

[BB68] J.L. Baer and D.P. Bovet, "Compilation of arithmetic expressions for parallel computation," *Proc. IFIP Cong. 1968*, North-Holland Pub. Co., Amsterdam, 340-346.

[BKM73] R.P. Brent, D.J. Kuck, and K. Maruyama, "The parallel evaluation of arithmetic expression without division," *IEEE Trans. on Computers*, C-22, 5, (May 1973) 532-534.

[BV85] I. Bar-On and U. Vishkin, "Optimal Parallel Generation of a Computation Tree Form," *ACM Trans. Prog. Lang. and Sys.*, 7, 348-357, 1985.

[CV87] R. Cole and U. Vishkin, "The accelerated centroid decomposition technique for optimal parallel tree evaluation in logarithmic time," *TR*, Courant Institute, June, 1987.

[CV88] R. Cole and U. Vishkin, "Optimal parallel algorithms for expression tree evaluation and list ranking," *Proc. AWOC 88, VLSI Algorithms and Architectures*, Corfu, Greece (Springer-Verlag)(1988), 91-100.

[GR86] A. Gibbons and W. Rytter, "An optimal parallel algorithm for dynamic expression evaluation and its applications," *RR 77*, Dept. of Computer Sci., Univer. of Warwick, April, 1986.

[KD88] S.R. Kosaraju and A.L. Delcher, "Optimal parallel evaluation of tree-structured computation by raking," *Proc. AWOC 88, VLSI Algorithms and Architectures*, Corfu, Greece (Springer-Verlag) (1988), 101-110.

[KM75] D.J. Kuck and K. Maruyama, "Time bounds on the parallel evaluation of arithmetic expressions," *SIAM J. Comput.* 4, 2, (June 1975), 147-162.

[L84] F.T. Leighton, "New lower bound techniques for VLSI," *Math. System Theory*, 17 (January 1984), 47-70.

[LF80] R.E. Ladner and M.J. Fischer, "Parallel prefix computations," *J. ACM*, 27, 4(October 1980), 831-838.

[M71] K. Maruyama, "On the parallel evaluation of polynomials," *IEEE Trans. on Computers*, C-22, 1, (Jan. 1973), pp. 2-5.

[MP71] I. Munro and M. Paterson, "Optimal algorithm for parallel polynomial evaluation," *Proc. IEEE Twelfth Annual Symp. on Switching and Automata Theory*, Oct. 1971, 132-139.

[MP75] D.E. Muller and F.P. Preparata, "Bounds to complexity of networks for sorting and for switching," *J.ACM*, 22, 2, (April 1975), 195-201.

[MP76] D.E. Muller and F.P. Preparata, "Restructuring of arithmetic expressions for parallel evaluation," *J. ACM* 23, 3, (July 1976), 534-543.

[MR85] G.L. Miller and J.H. Reif, "Parallel tree contraction and its applications," *Proc. 26th IEEE Symposium on Foundations of Computer Science*, 478-489, 1985.

[PM76] F.P. Preparata and D.E. Muller, "Efficient parallel evaluation of boolean expressions," *IEEE Trans. on Computers*, C-25, 5 (May 1976), 548-549.