REPORT R-676 APRIL, 1975 UILU ENG 75-2211

CSL COORDINATED SCIENCE LABORATORY

RESTRUCTURING OF ARITHMETIC EXPRESSIONS FOR PARALLEL EVALUATION

DAVID E. MULLER
FRANCO P. PREPARATA

UNIVERSITY OF ILLINOIS - URBANA, ILLINOIS

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*
RESTRUCTURING OF ARITHMETIC EXPRESSIONS FOR PARALLEL EVALUATION

5. TYPE OF REPORT & PERIOD COVERED
Technical Report

6. PERFORMING ORG. REPORT NUMBER
R-676; UILU-ENG 75-2210

7. AUTHOR(s)
David E. Muller
Franco P. Preparata

8. CONTRACT OR GRANT NUMBER(s)
DAAB-07-72-C-0259

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Coordinated Science Laboratory
University of Illinois
Urbana, Illinois 61801

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Joint Services Electronics Program
Fort Monmouth, New Jersey

12. REPORT DATE
April 1975

13. NUMBER OF PAGES
28

14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*

15. SECURITY CLASS. *(of this report)*
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Evaluation of Arithmetic Expressions
Parallel Computation
Computation of Arithmetic Expressions
Computational Complexity
Division-free Expressions
General Arithmetic Expressions

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Let E be an arithmetic expression involving n variables, each of which appears just once, and the possible operations of addition, multiplication, and division. Although other cases are considered, when these three operations take unit time the restructuring algorithms presented in this paper yield evaluation times no greater than $2.88 \log_2 n + 1$ and $2.08 \log_2 n$ for general and division-free expressions, respectively. The coefficients are precisely given by $2/\log_2 \alpha \approx 2.88$ and $1/\log_2 \beta \approx 2.08$, where α and β are the positive real roots of

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

UNCLASSIFIED

20. Abstract (Continued)

the equations $z^2 = z + 1$ and $z^4 = 2z + 1$, respectively. While these times were known to be of order $\log_2 n$, the best previously known coefficients were 4 and 2.15 for the two cases.

We conjecture that the present coefficients are the best possible and we have exhibited expressions which seem to require these times within an additive constant.

We also present upper-bounds to the restructuring time of a given expression E and to the number of processors required for its parallel evaluation. We show that at most $O(n^{1.44})$ and $O(n^{1.817})$ operations are needed for restructuring general and division-free expressions, respectively. It is pointed out that, since the order of the compiling time is greater than n log n, the numbers of required processors exhibit the same rate of growth in n as the corresponding compiling times.

UILU-ENG 2211

RESTRUCTURING OF ARITHMETIC EXPRESSIONS FOR PARALLEL EVALUATION

by

David E. Muller and Franco P. Preparata

This work was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) under Contract DAAB-07-72-C-0259.

Restructuring of Arithmetic Expressions for Parallel Evaluation

David E. Muller* and Franco P. Preparata**
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Abstract

Let E be an arithmetic expression involving n variables, each of which appears just once, and the possible operations of addition, multiplication, and division. Although other cases are considered, when these three operations take unit time the restructuring algorithms presented in this paper yield evaluation times no greater than $2.88 \log_2 n + 1$ and $2.08 \log_2 n$ for general and division-free expressions, respectively. The coefficients are precisely given by $2/\log_2 \alpha \approx 2.88$ and $1/\log_2 \beta \approx 2.08$, where α and β are the positive real roots of the equations $z^2 = z + 1$ and $z^4 = 2z + 1$, respectively. While these times were known to be of order $\log_2 n$, the best previously known coefficients were 4 and 2.15 for the two cases.

We conjecture that the present coefficients are the best possible and we have exhibited expressions which seem to require these times within an additive constant.

We also present upper-bounds to the restructuring time of a given expression E and to the number of processors required for its parallel evaluation. We show that at most $O(n^{1.44})$ and $O(n^{1.817})$ operations are needed for restructuring general and division-free expressions, respectively. It is pointed out that, since the order of the compiling time is greater than n log n, the numbers of required processors exhibit the same rate of growth in n as the corresponding compiling times.

---

*Department of Mathematics, University of Illinois at Urbana-Champaign.

**Department of Electrical Engineering, University of Illinois at Urbana-Champaign
This work was supported by the Joint Services Electronics Program (U.S. Army, U.S. Navy, and U.S. Air Force) under Contract DAAB-07-72-C-0259.

1. Introduction.

In recent times several computing systems have been designed or conceived so that many arithmetic operations may be executed simultaneously; thus it is important to study ways of arranging computations to take best advantage of such capability. One aspect of this problem which has attracted the attention of many investigators is the restructuring of an algebraic expression by means of algebraic identities so as to yield a computation tree of minimum depth, and hence a minimum computation time. It is possible to either assume that the number of available processors is unlimited, or to regard the problem as a trade-off between the cost of additional processors and the advantage of greater speed. In this paper we make the first assumption, because the mathematical methods we have developed are applicable in this case, but we are mindful of the importance of the more general trade-off between number of processors and speed, and we feel that the methods used here may be adapted to the general case as well.

The early work by Baer and Bovet [1] used associativity and commutativity of arithmetic operations to achieve limited restructuring of the computation tree. Later workers such as Muraoka [2] and Brent, Kuck, and Maruyama [3] used distributivity as well; the latter group showed that any algebraic expression not involving division and containing n distinct variables called atoms could be evaluated in no more than $2.465 \log_2 n + O(1)$ steps. Later, Preparata and Muller [4] showed that the coefficient of $\log_2 n$ may be reduced to 2.1507. In the present paper a further reduction to 2.080 is achieved and it is conjectured that this is the minimum possible.

The special cases of polynomial evaluation was treated by Maruyama [5] and by Munro and Paterson [6] who showed that n-th degree polynomials could be evaluated in $\log_2 n + O(\sqrt{\log_2 n})$ steps. Also, Kuck and Maruyama [7] have shown, among other interesting results, that continued fractions with n terms require no more than $2 \log_2 n + O(1)$ steps.

The case of general arithmetic expressions, which might involve division as well as the other arithmetic operations, was treated by Winograd [8] and by Brent [9]. Brent's objective was the minimization of computation time with an unlimited number of processors, and so his results are more directly comparable to ours. His method involves the restructuring of a general expression into a rational form. This form can be evaluated by performing a single division at the end after the numerator and denominator have been computed without using division. He showed that any algebraic expression of n atoms could be evaluated in this way using no more than $4 \log_2 n + O(1)$ steps. In the present paper, the same method of "end division" is used and the coefficient of 4 is reduced to 2.880. We also conjecture that some algebraic expressions require this much time for their evluation, so that this is the minimum possible. It should be pointed out, however, that while Brent's method uses a number of processors which is proportional to n, the one described here uses a number which is proportional to $n^{1.44}$.

In the next section we shall analyze general arithmetic expressions and establish upper-bounds on the time for their parallel evaluation as well as on the number of required processors. An analogous analysis will be presented in Section 3 for the class of division-free arithmetic expressions.

2. General arithmetic expressions.

2.1. Evaluation time.

Let E be an expression involving numerical variables and the operations of addition, subtraction, multiplication, and division. We call E a primitive expression if each variable appears only once and we call the function it represents a primitive function. The variables appearing in a primitive expression will be assumed to be of two kinds: atomic variables $a_1, \ldots, a_r$ and free variables $x_1, \ldots, x_s$. The weight of E, written $|E|$, is defined as the number of atomic variables appearing in E.

As is well-known, an expression E may be transformed to a rational form P/P' representing the same function, where P and P' are polynomials in the variables and are relatively prime. Also, if E is primitive, then P and P' will not involve higher powers than the first in any of the variables.

Let P and P' be expressed as Maclaurin's series in the free variables writing $P = E_1\lambda_1 + \ldots + E_m\lambda_m$ and $P' = E'_1\lambda_1 + \ldots + E'_m\lambda_m$, where each expression λ_i is either 1 or a product of distinct free variables, and the coefficients E_i and E'_i are either 0 or are expressions involving atomic variables. These expressions for P and P' are unique except for possible order of the terms and algebraic transformations of the coefficients.

Define $t(E_i)$ as the minimum time required to compute E_i and similarly define $t(E'_i)$ for $i = 1, \ldots, m$. It is assumed that we are allowed to take advantage of any algebraic identities and that as many independent processors are available as are needed to carry out different types of

operations simultaneously. We let τ_A represent the time required for addition or subtraction, and τ_M the time required for multiplication, and we assume these times are known for purposes of calculating the bounds to $t(E_i)$ and $t(E_i')$. The time τ_D for division is not needed for these calculations, since E_i and E_i' do not involve division, and we shall assume that division will not be artificially introduced even if doing so speeds the calculation of E_i and E_i'.

Now, define $\hat{t}(E)$ as the maximum of all the times $t(E_i)$, ..., $t(E_n)$, $t(E_i')$, ..., $t(E_m')$. As remarked before, the sums P and P' are unique, so $\hat{t}(E)$ is thus uniquely defined when E is a primitive expression.

Let A and B be two primitive expressions with different variables and let x_i be any free variable in A. Then, we define the composition of A and B with respect to x_i, written A $\textcircled{x_i}$ B as the primitive expression obtained by substituting B for the free variable x_i in A. When the free variable x_i is understood, we shall simply write the composition as A o B. It is obvious that

$$|A \text{ o } B| = |A| + |B|.$$

Lemma 1: Let A and B be two primitive expressions with distinct variables and let t_1 and t_2 be constructively achievable upper-bounds to the computation times $\hat{t}(A)$ and $\hat{t}(B)$ respectively. Then the upper-bound $\hat{t}(A_1 \text{ o } B) \leq \tau_A + \tau_M + \max(t_1, t_2)$ is constructively achievable.

Proof: Let us tranform A into its rational form

$$\frac{A_1\lambda_1 + \ldots + A_m\lambda_m}{A_1'\lambda_1 + \ldots + A_m'\lambda_m} \text{ and B into its rational form } \frac{B_1\lambda_1' + \ldots + B_{m'}\lambda_{m'}'}{B_1'\lambda_1' + \ldots + B_{m'}'\lambda_{m'}'}.$$

Next, construct the rational form for A o B, the composition of A and B

with respect to some free variable x_i in A. Let λ_j and λ_k be two products in the rational form for A such that $\lambda_j = x_i\lambda_k$ and let λ'_ℓ be any product in the rational form for B. Then, there is a numerator term $(A_jB_\ell + A_kB'_\ell)\lambda_k\lambda'_\ell$ and a denominator term $(A'_jB_\ell + A'_kB'_\ell)\lambda_k\lambda'_\ell$ in the rational form for A o B. Since the time to compute any of the coefficients is bounded by the bound given in the lemma, the proof is complete.

Lemma 2: Let A and B be two primitive expressions with distinct variables and let t_1 and t_2 be constructively achievable upper-bounds to the computation times $\hat{t}(A)$ and $\hat{t}(B)$ respectively. Then the following upper-bounds are constructively achievable:

(i) $\hat{t}(A \pm B) \leq \tau_A + \tau_M + \max(t_1, t_2)$,

(ii) $\hat{t}(A \cdot B) \leq \tau_M + \max(t_1, t_2)$,

(iii) $\hat{t}(A / B) \leq \tau_M + \max(t_1, t_2)$.

Proof: Let rational forms be given for A and B which are written as in the proof of Lemma 1. Then, if λ_j and λ'_k are products in the rational forms for A and B respectively, we have the following constructions.

(i) There is a numerator term $(A_jB'_k \pm A'_jB_k)\lambda_j\lambda'_k$ and a denominator term $A'_jB'_k\lambda_j\lambda'_k$ in the rational form for $A \pm B$.

(ii) There is a numerator term $A_jB_k\lambda_j\lambda'_k$ and a denominator term $A'_jB'_k\lambda_j\lambda'_k$ in the rational form for $A \cdot B$.

(iii) There is a numerator term $A_jB'_k\lambda_j\lambda'_k$ and a denominator term $A'_jB_k\lambda_j\lambda'_k$ in the rational form for A/B.

We note that the times required to compute the coefficients of these terms are bounded as given by the statement of the lemma and the proof is complete.

We now cite a lemma which is a slight generalization of similar lemma appearing in [3] and [9], and which is written here using our notation, without proof.

Lemma 3: Let E be any primitive expression and q a real number in the range $1 \leq q \leq |E|$. Then E may be written in the form $A \textcircled{x_r} (B \theta C)$, where A, B, and C are primitive expressions with no common variables, where x_r is a free variable of A and the only variable of A, B, and C which does not appear in E, and where $B \theta C$ denotes one of the expressions $B + C$, $B \cdot C$, B / C, or C / B. Furthermore, A, B, and C and be algorithmically chosen so that $|B| \leq |C| < q$, while $|B| + |C| \geq q$.

Lemma 4: Let $\tau_A + \tau_M = 1$, and let α be the positive root of the equation $z^2 = z + 1$. If E is any primitive expression, the upper-bound $\hat{t}(E) \leq \log|E|/\log \alpha$ is algorithmically achievable.

Proof: Assume inductively that for some given integer n, the result holds whenever $|E| < n$. By constructing the first few cases, it is easy to show that the induction may be started with n no smaller than 4.

Now let E be some primitive expression satisfying $|E| = n$. Using Lemma 3 we choose primitive expressions A, B, and C such that $E = A \circ (B \theta C)$, where the composition and operation θ satisfy the conditions of Lemma 3 and where $|B| \leq |C| < \alpha^{-2}n$ but $|B| + |C| \geq \alpha^{-2}n$. Since $n \geq 4$ and $\alpha \approx 1.44$ we have $\alpha^{-2}n > 1$ and hence we can take $\alpha^{-2}n$ to be the number q of Lemma 3.

By our inductive assumption, since $|C| < n$ the rational form for C can be algorithmically constructed so that $\hat{t}(C) \leq \log |C|/\log \alpha < \log \alpha^{-2} n/\log \alpha = \log n/\log \alpha - 2$ is achieved. Since $|B| \leq |C|$, the same bound applies to $\hat{t}(B)$. Hence, by Lemma 2, the rational form for $B \theta C$ may be algorithmically constructed so that $\hat{t}(B \theta C) \leq \log n/\log \alpha - 1$ is achieved.

Also, $|A| = |E| - |B| - |C| \leq n - \alpha^{-2} n = \alpha^{-1} n$, by the defining equation for α. Again, $|A| < n$, so inductively $\hat{t}(A) \leq \log \alpha^{-1} n/\log \alpha = \log n/\log \alpha - 1$ may be achieved. Thus, by Lemma 1, there is an algorithm for obtaining $\hat{t}(E) = \hat{t}(A \circ (B \theta C)) \leq \log n/\log \alpha$, and the lemma is proved.

We wish to point out that the reason it is possible to use a larger root α in the proof of this lemma than was used by previous investigators (see [9]), is that we allow the free variables of E to appear anywhere in the expression, and do not constrain them to lie on a single path of the original computation tree of E, as was done in [9].

Theorem 1: Let E be a primitive expression containing no free variables and involving possibly the operations of addition, subtraction, multiplication, and division, requiring times τ_A, τ_M, and τ_D respectively. Then a constructively achievable upper-bound to the time t(E) required to compute E is given by

$$\frac{(\tau_A + \tau_M)\log |E|}{\log \alpha} + \tau_D \approx 1.44\ (\tau_A + \tau_M)\ \log_2 |E| + \tau_D.$$

Proof: Since E contains no free variables, its equivalent rational form P/P' is just E_1/E_1', because $\lambda_1 = 1$. We have $\hat{t}(E) = \max(t(E_1), t(E_1'))$, so that $t(E) \leq \hat{t}(E) + \tau_D$.

Now, Lemma 4 applies to all primitive expressions, so it applies to the special case in which E contains no free variables. In Lemma 4, the time scale was normalized to make $\tau_A + \tau_D = 1$. Hence, to achieve the present result we simply multiply the value of $\hat{t}(E)$ so normalized by $\tau_A + \tau_M$. This completes the proof of the theorem.

Although the proofs which have been given show that the upper-bound stated in Theorem 1 is achievable, we shall now exhibit two additional techniques which may be used for restructuring of general expressions and may improve the actual computation time in some cases. These techniques are described below.

1. The first technique is a variant of the algorithm described in the proof of Lemma 4. Rather than choosing the parameter q of Lemma 3 to be $\alpha^{-2}n$, we allow it to be selected anywhere in the range $\alpha^{-2}n \leq q \leq \alpha^{-1}n$. The validity of the algorithm will be proved by "assertions" following the individual steps.

Step 1. Choose a subexpression $B \theta C$ of E such that $E = A \circ (B \theta C)$, $|B| + |C| \geq q$, and $|B| \leq |C| < q$.

Step 2. If $|C| < \alpha^{-2}n$, construct E as $A \circ (B \theta C)$.

Assertion: Since $|B| \leq |C| < \alpha^{-2}n$, we have $\hat{t}(B), \hat{t}(C) \leq \log n/\log \alpha - 2$ giving $\hat{t}(B \theta C) \leq \log n/\log \alpha - 1$. Also, $|A| = |E| - |B| - |C| \leq n - q \leq n - \alpha^{-2}n = \alpha^{-1}n$ so $\hat{t}(A) \leq \log n/\log \alpha - 1$. Hence $\hat{t}(E) = \hat{t}(A \circ (B \theta C)) \leq \log n/\log \alpha$.

Step 3. If $\alpha^{-2}n \leq |C| < q$ then form $F = A \circ (B \theta x')$, where x' is a new free variable which does not appear in E. Construct E as $F \textcircled{x'} C$.

Assertion: F is primitive and $|F| = |E| - |C| \leq n - \alpha^{-2}n = \alpha^{-1}n$ so $\hat{t}(F) \leq \log n/\log \alpha - 1$. Also, $|C| < q \leq \alpha^{-1}n$ so $\hat{t}(C) \leq \log n/\log \alpha - 1$. Hence $\hat{t}(E) = \hat{t}(F \textcircled{x'} C) \leq \log n/\log \alpha$.

This flexibility in the choice of the bound q of Lemma 3 may enable one to consider several alternative stopping points in the algorithm described in the proof of that lemma and choose one which yields the shortest time. It may be noted in this connection that, according to Lemma 2, the operation θ may sometimes require less than unit time. In such an application, little additional compiling time is required if one bases one's time estimates on the weights $|A|$, $|B|$, and $|C|$.

2. A second technique for speeding the process described in Lemma 4 is to obtain tighter upper-bounds for starting the induction process. If for a given integer k it is possible to find a positive constant δ such that $\hat{t}(E) \leq \log |E|/\log \alpha - \delta$ whenever $|E|$ lies in the range $\alpha^{-2}k \leq |E| < k$, then the inductive process of Lemma 4 can be directly extended to show that it also holds when $|E| \geq k$.

One method which might be used to find such a constant δ is to use exhaustive methods to compute $\hat{t}(E)$ for all expressions E satisfying $|E| < k$ for some small value of k. Computer methods might be used to improve δ by increasing k. To carry out an algorithm for taking

advantage of such an improved value of δ, it would be necessary to tabulate the fastest forms for expressions of weight less than k.

We next point out that the upper-bound to the computing time given in Theorem 1 does not require that the processors be capable of executing independent programs, but it holds also for computing systems in which all processors perform the same operation at any given time. This occurs because each of the basic formulas used in Lemmas 1 and 2 is either a product or a sum of two products: so the computation sequence consists of alternating multiplication and (possibly dummy) addition-subtraction for the entire set of processors. Division is performed only once at the end of the entire calculation.

We conclude this section with the conjecture that there exist primitive expressions E which require at least time $(\tau_A + \tau_M) \log |E| / \log \alpha - C$ for their evaluation, where C is a constant. A possible method for constructing such an expression is by the inductive definition:

(i) $T_0 = a$, an atomic variable;

(ii) $T_{j+1} = T_{j-1} + \dfrac{a_j}{b_j + T_j}$.

Here, it is understood that the atomic variables appearing in T_j and T_{j-1} are distinct although they contain isomorphic parts and that the atomic variables a_j, b_j also appear nowhere else.

2.2. Number of processors and compiling time.

We now wish to obtain upper-bounds to the compiling time of a restructured expression and to the number of processors required for its parallel evaluation. For this and for subsequent related results

we require the following lemma, whose proof is omitted because it can be obtained by standard analytical techniques [10].

Lemma 5. Let f(x) be a convex-downward function of a real variable x and let $g = a_1 f(x_1) + \ldots + a_r f(x_r)$, where $x_1, \ldots, x_r$ are real non-negative variables and $a_1, \ldots, a_r$ are positive constants. If the domain of g is the convex hull defined by a set of extreme points, all the maxima of g occur at extreme points.

In the preceding subsection we have described two algorithms for restructuring of a general expression. The first of these, which was also the easiest to describe, was used in the proof of Lemma 4. The second algorithm involved considering two cases but allowed one to choose the parameter q anywhere in the range $\alpha^{-2}n \le q \le \alpha^{-1}n$, while the first required $q = \alpha^{-2}n$. In the following analyses we assume that the restructuring has been performed by the first algorithm; a detailed analysis of the second algorithm shows, however, that, in the worst case, the same upper-bound is obtained.

Let W(E) denote the number of operations performed by the restructuring algorithm in processing a given primitive expression E. For brevity, W(E) will be referred to as the "compiling work". It is convenient to distinguish two processes in compiling, although they are interleaved in the actual operation. The first process concerns the decomposition of a given expression E into three expressions A, B, and C so that $E = A \circ (B \theta C)$ (see Lemma 3), and we denote by $W_1(E)$ the corresponding work. The second process concerns the assignment of processors to carry out the operation θ in $B \theta C$ (see Lemma 2) and

the composition of A with (B θ C) (see Lemma 1); we let $W_2(E)$ denote the corresponding work.

We begin by analyzing $W_1(E)$. The corresponding algorithm (sketched in [3]) consists of two basic steps. In the first step, for each vertex of the tree T(E) representing E we compute the weight of the expression described by its subtree. In the second step, we trace a path from the root of T(E) following at each vertex the branch of larger weight until B and C are found. Together these two steps require work which is bounded above by $k_1|E|$, for some constant $k_1 > 0$. In fact, since free variables do not contribute to the weight, only those vertices of T(E) must be considered by the algorithm whose descendant subtrees have positive weights; the number of such vertices is $|E| - 1$.

The algorithm associated with Lemma 4 applies the previous algorithm recursively and we obtain

$$(1) \quad W_1(E) \leq W_1(A) + W_1(B) + W_1(C) + k_1|E|.$$

We now analyze $W_2(E)$. A basic operation is the assignment of operands to a processor. Referring to the algorithms associated with Lemmas 1 and 2, the number of such assignments is at most three times the number of numerator and denominator terms in the rational form of the result (since three assignments are needed to compute $wx + yz$ from operands w, x, y, z).

Let p, p_A, p_B, and p_C be the numbers of free variables in E, A, B, and C respectively. Thus, in forming B θ C, there are no more than $6 \cdot 2^{p_B + p_C}$ such assignments, and in forming the composition of A with

B θ C there are no more than $6 \cdot 2^p$. Hence, the corresponding compiling work is bounded above by $k_2(2^{p_B + p_C} + 2^p)$, where $k_2 > 0$ is some constant.

Again, the algorithm associated with Lemma 4 applies the previous two recursively and we obtain:

$$(2)\quad W_2(E) \le W_2(A) + W_2(B) + W_2(C) + k_2(2^{p_B + p_C} + 2^p).$$

Both inequalities (1) and (2) may be used to obtain upper-bounds to $W_1(E)$ and $W_2(E)$ by the application of Lemma 5.

In the case of $W_2(E)$, we assume inductively that for given $n > 1$, when $|E| < n$, then $W_2(E) \le k_2(2^p + 1)(|E|^\xi - 1)$, where $\xi \ge 1$ is a constant to be evaluated later. We note that $k_2(2^p + 1)(|E|^\xi - 1)$ is convex-downward and that the induction may be started by suitable choice of k_2. To complete the inductive step, we let $|E| = n$ and inequality (2) becomes

$$W_2(E) \le k_2(2^{p_A} + 1)(|A|^\xi - 1) + k_2(2^{p_B} + 1)(|B|^\xi - 1) +$$

$$k_2(2^{p_C} + 1)(|C|^\xi - 1) + k_2(2^{p_A + p_B} + 2^p).$$

We shall obtain an upper-bound to the right hand side of this inequality by treating $|A|$, $|B|$, and $|C|$ as real variables. In accordance with Lemma 5, its maximum can be shown to occur at the extreme point corresponding to $|A| = \alpha^{-1}n$, $|B| = 1$, $|C| = \alpha^{-2}n - 1$, and $p_A = p + 1$, $p_B = p_C = 0$. Thus, we obtain $W_2(E) \le k_2((2^{p+1} + 1)(\alpha^{-\xi}n^\xi - 1) + 2((\alpha^{-2}n - 1)^\xi - 1) + 2^p + 1)$. Now, ξ must be chosen so that $W_2(E) \le k_2(2^p + 1)(n^\xi - 1)$. If we replace $(\alpha^{-2}n - 1)^\xi$ by $(\alpha^{-2}n)^\xi$ in the right hand side of the preceding inequality, then clearly ξ need not be larger than is necessary to satisfy the equation:

$$k_2((2^{p+1} + 1)(\alpha^{-\xi} n^{\xi} - 1) + 2(\alpha^{-2\xi} n^{\xi} - 1) + 2^p + 1) = k_2(2^p + 1)(n^{\xi} - 1).$$

This equation is satisfied if $\alpha^{-\xi} = 1/2$, (which yields $\xi = 1/\log_2\alpha \approx 1.44$) and the inductive step is justified.

In the case to which Theorem 1 applies, E has no free variables; that is, $p = 0$. Then, we obtain

$$(3) \quad W_2(E) \leq 2k_2(|E|^{1.44} - 1) < 2k_2|E|^{1.44}.$$

Returning to $W_1(E)$, we make the inductive assumption that for given $n > 1$, when $|E| < n$, then $W_1(E) \leq k_3|E|\log_2|E|$, where k_3 is a constant to be evaluated later. Again, $|E|\log_2|E|$ is convex-downward and the induction may be started if k_3 is made large enough. To complete the inductive step, we assume $|E| = n$ and inequality (a) becomes

$$W_1(E) \leq k_3|A|\log_2|A| + k_3|B|\log_2|B| + k_3|C|\log_2|C| + k_1 n.$$

To prove that $W_1(E) \leq k_3|E|\log_2|E|$, k_3 must be chosen so that

$$k_3(|A|\log_2|A| + |B|\log_2|B| + |C|\log_2|C|) + k_1 n \leq k_3 n \log_2 n.$$

Treating $|A|$, $|B|$, and $|C|$ as real variables, in accordance with Lemma 5, it can be shown that the maximum of the left hand side of this inequality occurs at the extreme point corresponding to $|A| = \alpha^{-1}n$, $|B| = 1$, $|C| = \alpha^{-2}n - 1$. This yields:

$$W_1(E) \leq k_3(\alpha^{-1} n \log_2 \alpha^{-1} n + (\alpha^{-2} n - 1)\log_2(\alpha^{-2} n - 1)) + k_1 n.$$

The constant k_3 must be chosen so that the $W_1(E) \leq k_3 n \log n$. If we replace $(\alpha^{-2}n - 1)\log_2(\alpha^{-2}n - 1)$ by $\alpha^{-2} n \log_2 \alpha^{-2} n$ in the right hand side of the preceding inequality, then k_3 need be larger than is required to satisfy the equation:

$$k_3(\alpha^{-1}n \log \alpha^{-1}n + \alpha^{-2}n \log \alpha^{-2}n) + k_1 n = k_3 n \log n.$$

This equation is satisfied for all values of n if $k_3 = k_1/(1 + \alpha^{-2})\log_2\alpha$, and we have

(4) $W_1(E) \leq k_3|E|\log_2|E|$ for all E.

Combining the two results, we may write $W(E) = W_1(E) + W_2(E) \leq k_3|E|\log_2|E| + 2k_2|E|^{\xi}$. Since $\xi > 1$, the second term dominates as $|E|$ becomes large and W(E) grows as $|E|^{1.44}$.(1)

We now turn to the equally important problem of obtaining an upper-bound to the number of processors required for the evaluation of an expression E which has been restructured by applying the algorithm associated with Lemma 4. This problem is closely related to the previous one, and in particular to the evaluation of an upper-bound to $W_2(E)$. Certainly, the total number of processors cannot exceed the number $W_2(E)$ of processor assignments performed by the compiling algorithm. Hence, an upper-bound to the number of processors is $k_4|E|^{1.44}$ where k_4 is some constant.(2) This bound does not take advantage of the fact that a single processor may be used repeatedly, but it seems unlikely that this property can be used to reduce the order of the upper-bound, and in any case could do no more than divide it by the upper-bound to t(E), which grows only as $\log_2|E|$.

---

(1) It is worth pointing out that for the restructuring algorithm of general expressions described in [9], an analogous analysis shows that $\xi = 1$, whereby $W_1(E)$ becomes the dominating term and W(E) grows as $|E|\log_2|E|$.

(2) An analogous analysis can be developed for the restructuring algorithm of division-free expressions described in [3]. This analysis shows that the required number of processors grows no faster than $k|E|^{1.232}$, for some $k > 0$, whereas Brent et al. estimated this bound at $O(|E|^{1.71})$ (notice that $1.232.. \approx 1/2 \log_2\gamma$, where γ is the real positive root of $z^3 = z + 1$.)

3. Division-free arithmetic expressions.

3.1. Evaluation time.

In this section we shall consider the parallel evaluation of expressions involving numerical variables and the operations of addition, subtraction and multiplication. We shall call these expressions "division-free".

We shall use the nomenclature developed in Section 2. We know by Lemma 3 that any primitive expression E can be written in the form $A \textcircled{x} (B \theta C)$, where A, B, and C are primitive expressions with no common variable, x is a free variable of A not appearing in E, and θ represents either "+" or "·". The expression E can be expanded in Maclaurin's series with respect to x and x can be replaced by $(B \theta C)$. Since E is division-free we obtain

$$E = A'(B \theta C) + A''.$$

Notice that, differently from the general case, A' and A" are primitive expressions and that $|A'| \leq |A|$, $|A''| \leq |A|$. The notion of free variables is not essential to the following analysis, although the notation of composition (hereafter simplified by omitting the specification of the free variable involved) is quite convenient.

We shall now provide a constructive upper-bound to the time for parallel evaluation of division-free expressions. As in the general case, we shall describe an algorithm for restructuring a given primitive expression into an algebraically equivalent one, so that the computation tree of the latter has bounded depth. Unfortunately, as the reader will

notice, the tree of the restructured expression does not exhibit the alternation of addition and multiplication, as we found for general expressions. For this reason we shall assume that addition and multiplication require identical unit times, and express the bounds in terms of time units. For a given expression E, we let t(E) denote the minimum time required to compute E.

Lemma 6: Let β be the positive real root of the equation $z^4 = 2z + 1$ and let E be a primitive division-free expression. Then

$$t(E) \leq \log |E| / \log \beta .$$

Proof: We assume inductively that for given integer n the following hypotheses hold (they are seen to be true for $n \leq 4$):

P1. If $|E| < n$ then $t(E) \leq \log |E| / \log \beta$.

P2. Let E_1 and E_2 be primitive division-free expressions and define $r \triangleq \frac{\beta^3}{1+\beta} \max (|E_1|, \beta|E_2|)$. If $r < n$, then $t(E_1 + E_2) \leq \log r / \log \beta$.

The proof is constructive and is supplied by a procedure for restructuring division-free expressions. The procedure consists of two parts, Algorithms P1 and P2, which provide the proofs of the inductive extensions of P1 and P2, respectively. In each step, when it is shown that an expression satisfies the conditions of P1 or P2, it is assumed that the corresponding algorithm is recursively called to carry out the restructuring. The two algorithms mutually call each other, as we shall see below. We shall follow the same step-assertion format used in the preceding section. We begin by proving the inductive extension of P1.

Proof for P1: Let $|E| = n$

Algorithm P1

Step 1. Choose a subexpression $(B\ \theta\ C)$ of E such that $E = A\ o\ (B\ \theta\ C)$ $|B\ \theta\ C| \geq n(1 - \beta^{-2})$, $|B| \leq |C| < n(1 - \beta^{-2})$.

Assertion: We have $|A| = |E| - |B\ \theta\ C| \leq n - n(1 - \beta^{-2}) = n\beta^{-2}$. Moreover we have $|B| \leq |C| < n(1 - \beta^{-2}) < n\beta^{-2}$. Therefore, by P1, we obtain

(5) $t(A'), t(A''), t(B), t(C) \leq \log n/\log \beta - 2$.

Step 2. If $|C| \leq n\beta^{-3}$, then set $E \leftarrow A'(B\ \theta\ C) + A''$ and halt.

Assertion: $|B| \leq |C| \leq n\beta^{-3}$ implies $t(B\ \theta\ C) \leq \log n/\log \beta - 2$ by P1. This and (5) yield $t(A'(B\ \theta\ C)) \leq \log n/\log \beta - 1$ and $t(E) \leq \log n/\log \beta$.

Step 3. ($|C| > n\beta^{-3}$). If θ represents "+", then set $E \leftarrow A'C + A\ o\ B$ and halt.

Assertion: $|A\ o\ B| = |E| - |C| < n - n\beta^{-3} < n\beta^{-1}$, which by P1 implies $t(A\ o\ B) \leq \log n/\log \beta - 1$. This and (5) yield $t(E) \leq \log n/\log \beta$.

Step 4. ($|C| > n\beta^{-3}$ and θ represents "$\cdot$"). Choose a subexpression $(B_1\ \theta'\ C_1)$ of C such that $C = A_1\ o\ (B_1\ \theta'\ C_1)$, with $|B_1\ \theta'\ C_1| \geq n(1 - \beta^{-1})$, $|B_1| \leq |C_1| < n(1 - \beta^{-1})$ and set $E \leftarrow A'BA_1'\ (B_1\ \theta'\ C_1) + A\ o\ (BA_1'')$.

Assertion: Notice that $|A\ o\ BA_1''| = |E| - |B_1\ \theta'\ C_1| \leq n - n(1 - \beta^{-1}) = n\beta^{-1}$, which shows, by P1, that $t(A\ o\ BA_1'') \leq \log n/\log \beta - 1$. To complete the proof we must show that the product $E' \triangleq A'BA_1(B_1\ \theta'\ C_1)$ is computable

in time at most $\log n/\log \beta - 1$. We begin by transforming the expression $A_1'(B_1\ \theta'\ C_1)$ to a product of the form S_1S_2.

Step 5. If $|C_1| > n\beta^{-4}$ and θ represent "$\cdot$", set $S_1 \leftarrow A_1'B_1$ and $S_2 \leftarrow C_1$(Case 1); else set $S_1 \leftarrow A_1'$ and $S_2 \leftarrow (B_1\ \theta'\ C_1)$ (Case 2).

Assertion: We shall show that in all cases

(6) $t(S_1) \le \log n/\log \beta - 4$, $t(S_2) \le \log n/\log \beta - 3$.

(Case 1): $|A_1'B_1| \le |C| - |C_1| < n(1 - \beta^{-2}) - n\beta^{-4} < n\beta^{-4}$, which implies by P1, $t(A'B) \le \log n/\log \beta - 4$; we also have $|C_1| \le n(1 - \beta^{-1}) < n\beta^{-3}$, whence by P1, $t(C_1) \le \log n/\log \beta - 3$. (Case 2): Notice at first

$$|A_1| = |C| - |B_1\ \theta'\ C_1| < n(1 - \beta^{-2}) - n(1 - \beta^{-1}) = n(\beta^{-1} - \beta^{-2}) < n\beta^{-4},$$

which by P1 implies $t(A_1') \le \log n/\log \beta - 4$. We have now two subcases to consider:

(i) $|C_1| \le n\beta^{-4}$, which yields $t(B_1\ \theta'\ C_1) \le \log n/\log \beta - 3$, by P1;

(ii) $|C_1| > n\beta^{-4}$ and θ' represents +, in which case $|C_1| < n(1 - \beta^{-1}) < n\beta^{-6}(1 + \beta)$ and $|B_1| \le |C| - |C_1| < n(1 - \beta^{-2}) - n\beta^{-4} < n\beta^{-7}(1 + \beta)$; it follows that $[\beta^{-3}/(1 + \beta)]\max(|C_1|, \beta|B_1|) < n\beta^{-3}$, whence, by P2, $t(B_1 + C_1) \le \log n/\log \beta - 3$.

Step 6. If $|B| \le n\beta^{-4}$, set $E' \leftarrow A'((BS_1)S_2)$ (Case 1); if $n\beta^{-4} < |B| \le n\beta^{-3}$, set $E' \leftarrow (A'B)(S_1S_2)$ (Case 2); if $n\beta^{-3} < |B|$, set $E' \leftarrow ((A'S_1)S_2)B$ (Case 3) and halt.

Assertion: (Case 1): $|B| \le n\beta^{-4} \Rightarrow t(B) \le \log n/\log \beta - 4$, by P1; this, (5) and (6) yield the result $t(E') \le \log n/\log \beta - 1$. (Case 2): $n\beta^{-4} < |B| \le n\beta^{-3}$ imply $t(B) \le \log n/\log \beta - 3$ by P1 and

$|A| = |E| - |B| - |C| < n - n\beta^{-3} - n\beta^{-4} = n\beta^{-3}$; this in turn yields $t(E') \leq \log n/\log \beta - 3$, whence $t(E') \leq \log n/\log \beta - 1$. (Case 3) $|B| > n\beta^{-3}$ implies $|A| < n - 2n\beta^{-3} = n\beta^{-4}$, whence $t(A') \leq \log n/\log \beta - 4$; this, (5) and (6) yield $t(E') \leq \log n/\log \beta - 1$. This completes the proof for P1.

Proof for P2. We shall at first consider expression E_1 alone and show that it can be restructured as a sum $(F_1 + G_1)$ with the following properties: letting $|E_1| = r(1 + \beta)\beta^{-3} = r(\beta^{-2} + \beta^{-3})$, with $n \leq r < n + 1$, we have $t(F_1) \leq \log r/\log \beta - 1$ and $|G_1| \leq r\beta^{-5}(1 + \beta)$.

Algorithm P2

Step 1. Choose a subexpression $(B_1 \theta C_1)$ of E_1 such that

$$E_1 = A_1 \circ (B_1 \theta C_1), |B_1 \theta C_1| \geq r(\beta^{-2} + \beta^{-3}) - r(\beta^{-4} + \beta^{-5})$$

and $|B_1| \leq |C_1| < r(\beta^{-2} + \beta^{-3}) - r(\beta^{-4} + \beta^{-5})$.

Assertion: We have $|A_1| = |E_1| - |B_1 \theta C_1| \leq r(\beta^{-2} + \beta^{-3}) - r(\beta^{-2} + \beta^{-3}) + r(\beta^{-4} + \beta^{-5}) = r(\beta^{-4} + \beta^{-5}) = r\beta^{-5}(1 + \beta) < r\beta^{-2}$. Moreover, we have $|B_1| \leq |C_1| \leq r(\beta^{-2} + \beta^{-3} - \beta^{-4} - \beta^{-5}) < r\beta^{-2}$. Since $r\beta^{-2} < n$ by P1 we obtain

(7) $t(A_1'), t(A_1''), t(B_1), t(C_1) \leq \log r/\log \beta - 2$.

Step 2. If $|C_1| \leq r\beta^{-3}$, then set $F_1 \leftarrow A_1'(B_1 \theta C_1)$ and $G_1 \leftarrow A_1''$ and halt.

Assertion: $|B_1| \leq |C_1| \leq r\beta^{-3}$ implies that $t(B_1 \theta C_1) \leq \log r/\log \beta - 3$ by P1; this and (7) yield $t(F_1) = t(A'(B_1 \theta C)) \leq \log r/\log \beta - 1$. We showed above that $|G_1| \leq |A_1| \leq r\beta^{-5}(1 + \beta)$.

Step 3. $(|C_1| > r\beta^{-3})$. If θ represents ".", define $D \triangleq A_1'B_1$, set $F_1 \leftarrow DC_1$, $G_1 \leftarrow A_1''$ and halt.

Assertion: Notice that $|A_1'B| \le |E_1| - |C_1| < r(\beta^{-2} + \beta^{-3}) - r\beta^{-3} = r\beta^{-2}$; thus, by P1, $t(A_1'B_1) \le \log r/\log \beta - 2$. This and (7) yield $t(F_1) = t(DC_1) \le \log r/\log \beta - 1$.

Step 4. ($|C_1| > r\beta^{-3}$ and θ represent "+"). If $|B_1| \le r(\beta^{-5} + \beta^{-6})$, set $F_1 \leftarrow A_1'(B_1 + C_1)$, $G_1 \leftarrow A_1''$ and halt.

Assertion: Since $\beta^{-2} + \beta^{-3} < 2(\beta^{-4} + \beta^{-5})$ we have $|C_1| < r(\beta^{-4} + \beta^{-5}) = r(1 + \beta)\beta^{-5}$. Also, we have $|B_1| \le r(\beta^{-5} + \beta^{-6}) = (1 + \beta)\beta^{-6}$. Letting $r' \triangleq \frac{\beta^3}{\beta + 1} \max(|C_1|, \beta|B_1|)$ we have $r' \le \frac{\beta^3}{\beta + 1} r(1 + \beta)\beta^{-5} = r\beta^{-2}$, which, by P2, implies $t(B_1 + C_1) \le \log r/\log \beta - 2$ and, consequently, $t(F_1) = t(A_1'(B_1 + C_1)) \le \log r/\log \beta - 1$.

Step 5. ($|C_1| > r\beta^{-3}$, θ represents +, and $|\beta_1| > r(\beta^{-5} + \beta^{-6})$). Choose a subexpression $(B_2\ \theta\ C_2)$ of B_1 such that $B_1 = A_1 \circ (B_2\ \theta'\ C_2)$, $|B_2\ \theta'\ C_2| \ge r\beta^{-4}$, $|B_2| \le |C_2| < r\beta^{-4}$. Similarly find a subexpression $(B_3\ \theta''\ C_3)$ of C_1, such that $C_1 = A_3 \circ (B_3\ \theta''\ C_3)$, $|B_3\ \theta''\ C_3| \ge r\beta^{-4}$, $|B_3| \le |C_3| < r\beta^{-4}$. Define $D_2 = A_1'A_2'$ and $D_3 = A_1'A_3'$, set $F_1 \leftarrow (D_2(B_2\ \theta'\ C_2) + D_3(B_3\ \theta''\ C_3))$, $G_1 \leftarrow A_1 \circ (A_2'' + A_3'')$ and halt.

Assertion: Notice first that $|A_1'A_3'| \le |E_1| - |B_1| - |B_3\ \theta''\ C_3| < r(\beta^{-2} + \beta^{-3}) - r(\beta^{-5} + \beta^{-6}) - r\beta^{-4} < r\beta^{-3}$. It follows, by P1, that $t(D_3) \le \log r/\log \beta - 3$. Similarly, *a fortiori*, we obtain $t(D_2) \le \log r/\log \beta - 3$. Next, we notice that $|B_2| \le |C_2| < r\beta^{-4}$ imply, by P1, $t(B_2\ \theta'\ C_2) \le \log r/\log \beta - 3$; similarly, $t(B_3\ \theta''\ C_3) \le \log r/\log \beta - 3$ holds. This shows that $t(D_2(B_2\ \theta'\ C_2) + D_3(B_3\ \theta''\ C_3)) \le \log r/\log \beta - 1$. Finally, we observe that $|G_1| = |A_1 \circ (A_2'' + A_3'')| = |E_1| - |B_2\ \theta'\ C_2| - |B_3\ \theta''\ C_3| \le r(\beta^{-2} + \beta^{-3}) - 2r\beta^{-4} < r(\beta^{-4} + \beta^{-5}) = r\beta^{-5}(1 + \beta)$.

In all cases, E_1 can be restructured as a sum expression of the form $(F_1 + G_1)$ where $t(F_1) \leq \log r/\log \beta - 1$ and $|G_1| \leq r\beta^{-5}(1 + \beta)$. Similarly E_2 can be restructured as $(F_2 + G_2)$, with $t(F_2) \leq \log r/\log \beta - 2$ and $|G_2| \leq r\beta^{-6}(1 + \beta)$. Notice now that $\max(|G_1|, \beta|G_2|)\, \beta^3/(1 + \beta) \leq r\beta^{-2} \triangleq r''$. Thus $E_1 + E_2$ can be structured as

$$E_1 + E_2 = F_1 + (F_2 + (G_1 + G_2))$$

where $t(G_1 + G_2) \leq \log r''/\log \beta = \log r/\log \beta - 2$ by the inductive hypothesis P2, thereby yielding $t(E_1 + E_2) \leq \log r/\log \beta$. This completes the proof of the lemma.

As an immediate consequence of the preceding lemma, we have the following result.

Theorem 2. Let E be a primitive division-free arithmetic expression. Assuming that both addition and multiplication require unit time, E can be evaluated in parallel in at most $\log |E|/\log \beta \approx 2.0806 \log_2 |E|$ time units.

Finally, we formulate the conjecture that there exist primitive expressions E which require at least $\log |E|/\log \beta - c'$ time units for their evaluation. One possible family of such expressions is given by the following inductive construction:

(i) $T_0 = a$, an atomic variable;

(ii) $T_j = T_{j-3}(T'_{j-3}T_{j-4} + a_j) + b_j$,

where it is understood that a_j, b_j and all variables appearing in expressions T_{j-3}, T'_{j-3} and T_{j-4} are distinct.

3.2. Number of processors and compiling time.

We now seek an upper-bound to the number of processors required for the parallel evaluation of division-free expressions. For convenience, we shall say that an expression has been Pi-restructured if it has been processed by Algorithm Pi, where i = 1, 2.

We assume inductively that there are three constants $c_1 > 0$, $c_2 > 0$, and $\xi > 1$ such that:

1) If E is a primitive expression and $|E| < n$, the number of processors required for P1-restructuring of E is at most $c_1 \cdot |E|^{\xi}$;

2) If E is a primitive expression and $|E|\beta^3/(1 + \beta) < n$, the number of processors required for P2-restructuring of E is at most $c_2 \cdot |E|^{\xi}$.

The analysis here is considerably more complicated than that for general expressions (see preceding section), due to the large number of cases and to the interplay of the two algorithms. Therefore we shall omit the most tedious details and sketch the adopted approach.

We shall first consider Algorithm P1. This algorithm is characterized by several restructuring patterns for E. In Steps 2 and 3 two different patterns are given; Steps 5 and 6 deal with different subexpressions and each has three patterns corresponding to the three cases of Step 5 and the three cases of Step 6. Hence nine additional patterns arise from considering Steps 5 and 6. Each of these patterns is an expression $G(E_1, \ldots, E_s)$ whose terms $E_1, E_2, \ldots, E_s$ are themselves expressions; of these, without loss of generality, $E_1, \ldots, E_m$ are P1-restructured and $E_{m+1}, \ldots, E_s$ are P2-restructured. Moreover, there

are linear constraints on the nonnegative weights $|E_1|, \ldots, |E_s|$, so that the s-tuple $(|E_1|/|E|, \ldots, |E_s|/|E|)$ is restricted to the convex hull of a finite set of (extreme) points. Contrary to what we found for general expressions, the restructured computation trees of division-free expressions have no internal fan-out, i.e., in general the number of processors required by a term E_j, defined above, depends only on $|E_j|$ and no apparent advantage can be taken of the fact that E_j may share literals with other terms of the set $E_1, \ldots, E_s$. Let $N_i(E_j)$ be the number of processors required to evaluate one Pi-restructured expression E_j. For any given restructuring pattern the number $N_1(E)$ of processors is bounded as follows:

$$N_1(E) \leq \max(N_1(G), \sum_{j=1}^{m} N_1(E_j) + \sum_{i=m+1}^{s} N_2(E_i))$$

where G is the expression describing the pattern. According to our inductive assumptions $N_1(E_j) \leq c_1|E_j|^{\xi}$, $N_2(E_i) \leq c_2|E_i|^{\xi}$, since $|E_k| < n$ for $k = 1, \ldots, s$. Moreover, we shall assume for convenience that c_1 and c_2 are sufficiently large to guarantee

$N_1(G) \leq c_1 \sum_{j=1}^{m} |E_j|^{\xi} + c_2 \sum_{i=m+1}^{s} |E_i|^{\xi}$ even when $E_1, \ldots, E_s$ are individual variables. We note that the function $c_1 \sum_{j=1} |E_j|^{\xi} + c_2 \sum_{i=m+1} |E_i|^{\xi}$ achieves its maximum at an extreme point of the convex domain of the n-tuples $(|E_1|, \ldots, |E_s|)$ by Lemma 5, since $\xi > 1$, $c_1 > 0$, $c_2 > 0$. The constants ξ, c_1, and c_2 must be chosen such that $N_1(E) \leq c_1|E|^{\xi}$. Therefore, we shall select them so that

$$(8) \qquad c_1|E|^{\xi} = c_1 \sum_{j=1}^{m} |E_j|^{\xi} + c_2 \sum_{i=m+1}^{s} |E_i|^{\xi}.$$

Any restructuring pattern of Algorithm P1 yields a relation of this type.

Similarly, we analyze Algorithm P2. Here again, there are four restructuring patterns, exhibited respectively in Step 2, Step 3, Step 4, and Step 5. Each such pattern is an expression $H(F_1, \ldots, F_t)$, where $F_1, \ldots, F_t$ are expressions, $F_1, \ldots, F_p$ are P1-restructured and $F_{p+1}, \ldots, F_t$ are P2-restructured. Reasoning as above, any such patterns yields a relation of the type

$$(9) \qquad c_2|E|^{\xi} = c_1 \sum_{j=1}^{p} |F_j|^{\xi} + c_2 \sum_{i=m+1}^{t} |F_i|^{\xi} .$$

Suppose now we pair a relation of type (8) with a relation of type (9). We obtain two equations which can be solved for the unknowns c_1/c_2 and ξ at the extreme points of the convex domain of the real variables $(|E_1|/|E|, \ldots, |E_s|/|E|, |F_1|/|E|, \ldots, |F_t|/|E|)$: we shall retain the largest value of ξ as the solution corresponding to the equation pair.

In principle, this procedure should be carried out for each equation pair and the largest value of ξ should be retained; in practice, simple considerations allow disregarding a large number of equation pairs. We spare the reader the obvious but laborious details which lead to showing that the following pattern pair yields the largest value of ξ:

Algorithm P1: (Step 5, Case 1) (Step 6, Case 1), yielding the equation:

$$(10) \qquad |A|^{\xi} + |B|^{\xi} + |A \circ (A_1'' B)|^{\xi} + |A_1' B_1|^{\xi} + |C_1|^{\xi} = |E|^{\xi}$$

Algorithm P2: (Step 2) yielding the equation

$$(11) \qquad 2|A_1|^{\xi} + |B_1|^{\xi} + |C_1|^{\xi} = c_2/c_1|E|^{\xi}.$$

The extreme points yielding the maximum value of ξ is given by: $|A|/|E| = \beta^{-2}$, $|B|/|E| = 0$, $|A_1|/|E| = \beta^{-1} - \beta^{-2}$, $|B_1|/|E| = 0$, $|C_1|/|E| = 1 - \beta^{-1}$, for the variables appearing in (10), and by $|A_1|/|E| = 1 - \beta^{-2}$, $|B_1|/|E| = 0$ and $|C_1|/|E| = \beta^{-2}$ for the variables appearing in (11). It is worth pointing out that since c_2/c_1 results to be less than 1 the value of ξ is entirely determined by (10) and is given by $\xi \approx 1.817$. Thus the number N(E) of required processors grows as $|E|^{1.817}$.

This bound may be contrasted with the result $N(E) \leq O(|E|^{1.232})$ which we have shown to hold for the restructuring algorithm of Brent et al. (see footnote (2), Section 2.2). The reason for this difference may be traced to the fact that in the latter algorithm, as in the algorithm described in Section 2.1, the restructuring consists essentially of computing coefficients of the rational form, whereas in Algorithms P1 and P2 new expressions are being formed by composing subexpressions of the original expression (see, particularly Steps 3 and 4 of Algorithm P1 and Step 5 of Algorithm P2).

We conclude this section with the remark that an upper-bound to the compiling work can be obtained exactly along the lines described in Section 2.2, resulting in the conclusion that the compiling work has the same worst-case rate of growth $O(|E|^{1.817})$.

4. Concluding remarks.

In this paper we have presented improved upper-bounds to the time required for the parallel evaluation of general as well as division-free arithmetic expressions. We have shown that, assuming that all operations take one unit of time, an expression with n atoms can be evaluated in at most 2.88 log n steps or 2.080 log n steps, depending upon whether or not it involves division. We have also exhibited families of expressions which we conjecture to require times for their evaluation within additive constants of the corresponding upper-bounds.

We have also investigated the growth of the compiling time for restructured expressions and of the number of processors required for evaluation, and elucidated an interesting connection between these two quantities.

References

[1] J. L. Baer and D. P. Bovet, "Compilation of Arithmetic Expressions for Parallel Computations," Proc. of IFIP Congress, 1968, pp. 340-346.

[2] Y. Muraoka, "Parallelism Exposure and Exploitation in Programs," Ph.D. thesis, University of Illinois at Urbana-Champaign, Department of Computer Science Report No. 424, Feb. 1971.

[3] R. P. Brent, D. J. Kuck, and K. Maruyama, "The Parallel Evaluation of Arithmetic Expressions Without Division," IEEE Transactions on Computers, Vol. C-22, No. 5, May 1973, pp. 532-534.

[4] F. P. Preparata and D. E. Muller, "The time required to evaluate division-free arithmetic expressions," to appear in Information Processing Letters.

[5] K. Maruyama, "On the Parallel Evaluation of Polynomials," IEEE Transactions on Computers, Vol. C-22, No. 1, Jan. 1973, pp. 2-5, (originally, Report 437, Department of Computer Science, University of Illinois at Urbana, March 1971.)

[6] I. Munro and M. Paterson, "Optimal Algorithm for Parallel Polynomial Evaluation," Proc. IEEE Twelfth Annual Symposium on Switching and Automata Theory, Oct. 1971, pp. 132-139.

[7] D. J. Kuck and K. Maruyama, "Time bounds on the parallel evaluation of arithmetic expressions," to appear in SIAM Journal on Computing.

[8] S. Winograd, "On the parallel evaluation of certain arithmetic expressions," to appear in JACM (also available as IBM Report RC 4804, Yorktown Heights, N.Y., April 1974).

[9] R. P. Brent, "The Parallel Evaluation of General Arithmetic Expressions," Journal of the ACM, Vol. 19, No. 2, April 1974, pp. 201-206.

[10] D. Blackwell and M. A. Girshick, Theory of Games and Statistical Decisions, J. Wiley, New York, 1954.