Correspondence

The Parallel Evaluation of Arithmetic Expressions Without Division

RICHARD BRENT, DAVID KUCK, AND KIYOSHI MARUYAMA

Abstract—As computers become capable of executing more arithmetic operations simultaneously, the question of compiling for such machines becomes more important.

In this correspondence we consider arbitrary arithmetic expressions of n distinct variables with operations restricted to addition, subtraction, and multiplication. We first construct a scheme whereby any such expression can be evaluated in at most $3 \log_2 n + 0(1)$ steps if sufficiently many processors are available. We then improve this result and reduce $3 \log_2 n$ to $2.465 \log_2 n$. Finally, we deduce some results that apply when a fixed number of processors are available.

Index Terms—Arithmetic expression evaluation, number of processors, parallel computation, processing time upper bound, simultaneous operations, tree-height reduction.

I. Introduction

Many computers now exist that are capable of executing more than one arithmetic operation simultaneously. As parallel and pipeline arithmetic units continue to be developed, the question of how quickly arithmetic expressions can be evaluated becomes more interesting. For example, the sum of 2^k numbers can obviously be formed in k steps using 2^{k-1} processors. On the other hand a polynominal of degree 2^k written in the form of Horner's rule requires 2^{k+1} steps if evaluated in its given form. Thus it may be useful to have good algorithms for transforming given arithmetic expressions into forms in which they may be more quickly evaluated.

The problem of evaluating an arithmetic expression using as many independent processors as necessary has been studied by a number of people. Baer and Bovet [1] gave a comprehensive algorithm that takes advantage of the associativity and commutativity of arithmetic operations. Muraoka [6] studied the use of distributivity as well and this is also discussed in [3]. It was conjectured in [6] that an arithmetic expression of 2^k variables whose operations are + and $*$ can be evaluated in at most $2k$ steps. It was proved in [6] that such an expression with d levels of parenthesis nesting can be evaluated in at most $1 + 2d + k$ steps. Brent [2] has shown that arithmetic expressions of the form $a_0 + x_1(a_1 + x_2(a_2 + \cdots + x_n a_n) \cdots)$, where $n \leqslant 2^k$, can be evaluated in $k + \sqrt{8k} + 3$ steps. It has been shown by Maruyama [4] and by Munro and Paterson [5] that polynomials of degree n can be evaluated in $k + \sqrt{2k} + 0(1)$ steps.[1]

In this correspondence we study the problem of evaluating arithmetic expressions using sufficiently many independent processors, each of which is capable of performing an addition or multiplication on each step. First we show that an arithmetic expression of 2^k variables with operations + and $*$ can be evaluated in at most $3k - 4$ steps ($k > 2$). Our proof is given in the form of a constructive procedure for transforming a given expression into a form that satisfies this upper bound.

Manuscript received July 13, 1972; revised October 30, 1972. This work was supported in part by NSF Grant GJ-27446, AT(11-1)-2118 and in part by the IBM T. J. Watson Research Center.

R. Brent is with the Computer Centre, Australian National University, Canberra, Australia.

D. Kuck is with the Department of Computer Science, University of Illinois, Urbana, Ill. 61801.

K. Maruyama was with the Department of Computer Science, University of Illinois, Urbana, Ill. 61801. He is now with the IBM T. J. Watson Research Center, Yorktown Heights, N.Y.

[1] We use the standard notation for the order of magnitude of a function: $f(n) = 0(g(n))$ if there is a constant $r > 0$ such that $\limsup_{n\to\infty} (f(n)/g(n)) = r$.

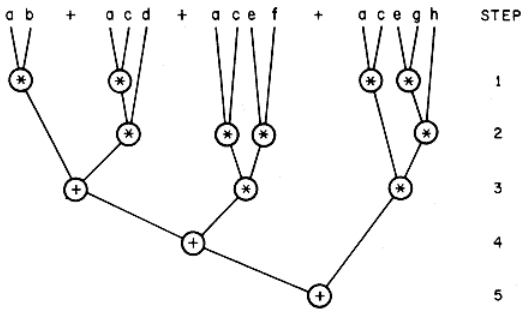

Fig. 1. Fully distributed tree.

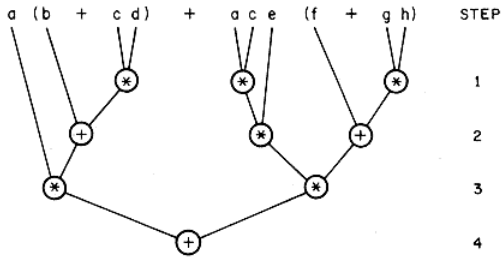

Fig. 2. Properly cut and distributed tree.

The proof uses only the associative, commutative, and distributive laws, so it applies for arithmetic expressions in any commutative ring (e.g., Boolean expressions). Also, the introduction of the (binary or unary) subtraction operator does not significantly alter the result. However, our proof does not hold if the division operator is allowed.

A slight modification of the argument shows that $2.465k + 0(1)$ steps are sufficient (see Theorem 2). We suspect that the number $2.46 \cdots$ can be reduced further, but this is an open question.

The results mentioned above apply if arbitrarily many processors are available. We also give (without proof) some results that apply under the more realistic assumption that a fixed number $p \geqslant 1$ of processors is available.

II. Example

In order to motivate our discussion of the problem and its solution we now present a simple example that illustrates our method. Consider the problem of evaluating the expression

$$E = a(b + c(d + e(f + gh)))$$

which requires seven operations and seven time steps as presented, assuming multiplication and addition each take one time step. By performing some "redundant" operations, we may speed up the evaluation process. Fig. 1 shows that by performing all possible distributions, E may be evaluated in five steps. However, by performing only selected distributions, Fig. 2 shows that it is possible to evaluate the expression in just four steps.

The form shown in Fig. 2 illustrates our method of tree-height reduction. In general our method proceeds to "cut in half" a given tree by distributing certain multiplication operations over additions. Then each half is again cut and the procedure continues from the root to the atoms of a tree.

Since E has eight atoms we perform a cut between the fourth and fifth atoms, namely, between d and e. Thus by distribution and association, E may be rewritten as

$$E' = a(b + cd) + ace(f + gh).$$

For this example, just one cut is sufficient and a tree for E' is shown in Fig. 2. It should be noted that while E required only seven operations, E' requires nine operations for its evaluation and the fully distributed expression of Fig. 1 requires 13 operations. Furthermore, while the evaluation of E requires at most one processor, E' may be evaluated using three processors and the expression of Fig. 1 requires four processors.

Thus the general idea of our approach is to introduce extra operations by distribution, in an attempt to form a tree that is of lower height than any tree for the presented form of a given expression. However, we must generally refrain from performing all possible distributions, because too many redundant operations will cause the tree height to be greater than necessary.

III. Definitions and Preliminary Results

An *atom* is a single variable or constant. We denote atoms by lowercase letters. We consider only the two standard binary arithmetic operations of addition and multiplication, denoted by + and $*$. Throughout this correspondence, by an *arithmetic expression* we mean any well-formed string of + and $*$ operators and atoms. We denote arithmetic expressions by uppercase letters and write $E(n)$ to denote an arithmetic expression E of at most n distinct[2] atoms. To single out particular atoms in an arithmetic expression we include them in the argument list. Thus $E(n-1, g)$ refers to an expression of at most n atoms, one of which is g. To denote the exact number of atoms in any expression E we write $|E|$.

It is well known that a well-formed arithmetic expression has one or more parse trees. We denote a tree for the expression E as T_E, and arbitrary trees as T_1, T_2, etc. We also let $|T_E|$ represent the number of atoms in the tree T_E. We say that two trees T_i and T_j are *joined* by an operator θ if the root nodes of T_i and T_j are both attached to a new root node labeled with the θ. Given any tree T_1 we define its *subtrees* as follows. Two subtrees of T_1, T_{11}, and T_{12}, are joined at the root of T_1. Similarly T_{11} and T_{12} may contain subtrees, each of which is also regarded as a subtree of T_1. This continues until finally all the atoms of T_1 are reached, and they also are regarded as subtrees of T_1.

Lemma 1: Using only the distributive, associative, and commutative properties of + and $*$, any arithmetic expression $E(n,g)$ can be rewritten in the form $E'(2n,g) = A(n)*g + B(n)$.

Proof: Consider any parse tree T_E for E. Without loss of generality, let g be joined to subtree T_1 of T_E by operator θ_1 as shown in Fig. 3. Assume that the subtree consisting of g, θ_1, and T_1 is joined to subtree T_2 by θ_2 and so on, with the root operator of T_E being denoted by θ_r, as shown in Fig. 3. Let the expression associated with T_i be denoted by E_i, $1 \leqslant i \leqslant r$.

By distribution we form $A(n) = \Pi_I E_i$, where $I = \{i \mid 1 \leqslant i \leqslant r \text{ and } \theta_i = *\}$. Since there are at most n atoms in $E(n)$ (excluding g), A can have no more than n atoms.

To find $B(n)$, let k be the smallest index of a + operator in Fig. 3, i.e., $\theta_k = +$ and $\theta_j = *$ for $j = k-1, k-2, \cdots, 1$. B can be found by deleting from T_E the subtree corresponding to $E_{k-1} * E_{k-2} \cdots * g * E_1$ if $k > 1$ or by simply deleting g if $k = 1$. The expression corresponding to the remaining tree is B. This is arithmetically equivalent to setting $g = 0$ in E. Since E has at most n atoms (excluding g), B has at most n atoms.

Lemma 2: Suppose $1 < m \leqslant n$, and let $E(n)$ be an arithmetic expression. If T_E is any parse tree for $E(n)$ then there is a pair of subtrees T_L and T_R of T_E such that T_L and T_R are joined to each other

$$|T_L| < m, \quad |T_R| < m, \quad |T_L| + |T_R| \geqslant m.$$

Proof: Let $T_P^{(0)} = T_E$. Consider the subtrees $T_P^{(1)}$ and $T_Q^{(1)}$ joined to the root node of $T_P^{(0)}$. Without loss of generality, $|T_P^{(1)}| \geqslant |T_Q^{(1)}|$.

[2] Throughout this correspondence we refer to expressions of a number of distinct atoms. By this we mean that each occurrence of each atom is counted so that, for example, $a + a + a$ and $a + b + c$ each have three atoms.

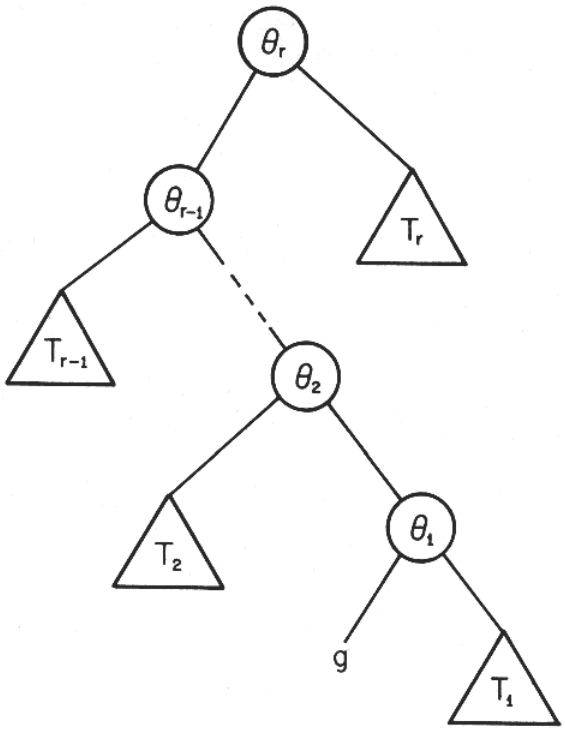

Fig. 3. Parse tree T_E.

If $|T_P^{(1)}| \geqslant m$, consider the subtrees $T_P^{(2)}$ and $T_Q^{(2)}$ joined to the root node of $T_P^{(1)}$. Continuing in this way, we eventually find subtrees $T_P^{(i-1)}$, $T_P^{(i)}$, and $T_Q^{(i)}$, such that

$$|T_P^{(i)}| + |T_Q^{(i)}| = |T_P^{(i-1)}| \geqslant m$$

and

$$1 \leqslant |T_Q^{(i)}| \leqslant |T_P^{(i)}| < m.$$

Thus, we may take $T_L = T_P^{(i)}$ and $T_R = T_Q^{(i)}$.

Theorem 1: For $k \geqslant 3$, any arithmetic expression $E(2^k)$ may be evaluated in $3k - 4$ steps.

Proof: By inspection, the theorem is true for $k = 3$. As an inductive hypothesis suppose that, for some $k \geqslant 3$, expressions with 2^k atoms can be evaluated in $3k - 4$ steps. We shall show that an expression $E(2^{k+1})$ can be evaluated in $3k - 1 = 3(k+1) - 4$ steps.

Let T_E be any parse tree for $E(2^{k+1})$. Find T_L and T_R (joined by θ) using Lemma 2 with $m = 2^k$ and $n = 2^{k+1}$. By the inductive hypothesis, the expressions L and R corresponding to T_L and T_R can be evaluated in $3k - 4$ steps, so $G = L\ \theta\ R$ can be evaluated in $3k - 3$ steps.

Let T_{E_1} be the tree formed by replacing the subtree T_G of T_E by an atom g. By Lemma 2, $|G| \geqslant 2^k$, so $|E_1| \leqslant 2^k + 1$, and we may write E_1 as $E_1(2^k, g)$. Applying Lemma 1, $E_1 = A(2^k) * g + B(2^k)$ for some expressions A and B, so $E = A * G + B$.

By the inductive hypothesis, A and B can be evaluated in $3k - 4$ steps. Since G can be evaluated in $3k - 3$ steps, $A * G$ can be evaluated in $3k - 2$ steps, and $E = A * G + B$ in $3k - 1$ steps. Thus, the result follows by induction on k.

IV. Improvement on Theorem 1

Let $\tau(n)$ be the number of steps required to evaluate an expression with n distinct atoms. By inspection, $\tau(n) = n - 1$ for $1 \leqslant n \leqslant 4$. Also, from Theorem 1,[3] $\tau(n) \leqslant 3\lceil \log_2 n \rceil - 4$ for $n \geqslant 5$, so

[3] Since Theorem 1 was proved for $n = 2^k$, it can be applied to any integer n by introducing the notation $\lfloor x \rfloor$ which for any real x denotes the integer such that $x \leqslant \lfloor x \rfloor < x + 1$.

$$\tau(n) \leqslant 3 \log_2 n + 0(1) \tag{1}$$

as $n \to \infty$. In this section we show that the factor 3 in (1) can be reduced to $\log_\lambda 2 = 2.4649$, where $\lambda = 1.3247\cdots$ is the (unique) real positive root of $z^3 = 1 + z$.

Lemma 3: Let $r_0 = 1, r_1 = 2, r_2 = 3$, and

$$r_{k+3} = 1 + r_k + r_{k+1} \tag{2}$$

for $k \geqslant 0$. Then $\tau(r_k) \leqslant k$.

Proof: The proof is similar to that of Theorem 1. As an inductive hypothesis suppose that $\tau(r_0) \leqslant 0, \tau(r_1) \leqslant 1, \cdots, \tau(r_{k+2}) \leqslant k + 2$. (By inspection, this is true for $k = 0$.) We shall show that $\tau(r_{k+3}) \leqslant k + 3$.

Let $E(r_{k+3})$ be an arithmetic expression with parse tree T_E. Find T_L and T_R (joined by θ) from Lemma 2 with $m = 1 + r_k$ and $n = r_{k+3}$. Let $G = L\theta R$, E_1 and g be as in the proof of Theorem 1. Since $|E_1| \leqslant 1 + r_{k+3} - |G| \leqslant r_{k+3} - r_k = 1 + r_{k+1}$, E_1 may be written as $E_1(r_{k+1}, g)$. Applying Lemma 1, $E_1 = A(r_{k+1}) * g + B(r_{k+1})$ for some expressions A and B, so $E = A * G + B$. By the inductive hypothesis, A and B can be evaluated in $k + 1$ steps. Also, $|L| \leqslant m - 1 \leqslant r_k$, so L can be evaluated in k steps, and similarly for R. Thus G can be evaluated in $k + 1$ steps, and E can be evaluated in $k + 3$ steps. Hence, the result follows by induction on k.

Lemma 4: If r_k and λ are as above, then

$$r_k \geqslant \left(\frac{3\lambda^2 + 4\lambda + 2}{2\lambda + 3}\right)\lambda^k - 1 - 2\lambda^{-k/2}\sqrt{\frac{\lambda^2 + \lambda - 3}{4\lambda^2 - 6\lambda + 5}}.$$

Proof: The general solution of the linear recurrence relation (2) is

$$r_k = \sum_{i=1}^{3} c_i \lambda_i^k - 1$$

where the λ_i are the roots of $z^3 = 1 + z$, and the c_i are arbitrary constants. Since $r_0 = 1$, $r_1 = 2$, and $r_2 = 3$, we find (using generating functions) that

$$c_i = \frac{3\lambda_i^2 + 4\lambda_i + 2}{2\lambda_i + 3}.$$

Suppose that $\lambda_1 = \lambda$ is real. Then

$$|\lambda_2| = |\lambda_3| = \lambda^{-1/2}$$

$$\lambda_2 = \bar{\lambda}_3 = -\frac{1}{2}(\lambda \pm i\sqrt{3\lambda^2 - 4})$$

and

$$|c_2| = |c_3| = \sqrt{\frac{\lambda^2 + \lambda - 3}{4\lambda^2 - 6\lambda + 5}}$$

so the result follows.

Theorem 2: For $n \geqslant 2$

$$\tau(n) \leqslant \lfloor \log_\lambda (\alpha n + \beta) \rfloor.$$

where[4]

$$\alpha = \lambda / c_1 = 0.5956\cdots,$$

$$\beta = 2\lambda|c_2|/c_1 = 0.1665\cdots,$$

and λ, c_1, and, c_2 are as in the proof of Lemma 4.

Proof: Let $k \geqslant 1$ be such that $r_{k-1} < n \leqslant r_k$. From Lemma 4,

$$n \geqslant r_{k-1} + 1 \geqslant c_1 \lambda^{k-1} - 2|c_2|$$

so

$$\lambda^k \leqslant \alpha n + \beta$$

giving

$$k \leqslant \log_\lambda (\alpha n + \beta).$$

However, from Lemma 3

$$\tau(n) \leqslant \tau(r_k) \leqslant k$$

so

$$\tau(n) \leqslant \log_\lambda (\alpha n + \beta).$$

Since $\tau(n)$ is an integer, the result follows.

V. Conclusion

Theorem 1 shows that expressions with 2^k atoms ($k > 2$) can be evaluated in $3k - 4$ steps if enough processors are available. It is easy to show, by induction on k, that 4^{k-2} processors are enough. (In the proof of Theorem 1, the four expressions A, B, L, and R must be evaluated simultaneously.) A more delicate argument shows that $0(3^k)$ processors suffice. Thus, an expression $E(n)$ can be evaluated in the number of steps given by (1) using $0(n^{1.58\cdots})$ processors, where $1.58\cdots = \log_2 3$. Similarly, $E(n)$ can be evaluated in the number of steps given by Theorem 2 using $0(n^{1.71\cdots})$ processors, where $1.71\cdots = \log_\lambda ((1 + \sqrt{5})/2)$. This follows from an extension of Lemma 3. It is easy to give a bound on the time required to evaluate an expression if a restricted number of processors are available. If $T_p(n)$ denotes the time required to evaluate $E(n)$ using p processors, then

$$T_p(n) \geqslant \lfloor n/p \rfloor - 1 + \lceil \log_2 p \rceil.$$

One application of this work could be in the area of logic design. For any given Boolean expression, assume that some procedure was used to minimize the number of literals. Then the methods described here could be used to design circuits with small overall delay.

If this work were extended to include the division operator, the results could be of interest in compiling arithmetic expressions for multi-arithmetic unit computers.

[4] For any real x we use $\lfloor x \rfloor$ to denote the integer such that $x - 1 < \lfloor x \rfloor \leqslant x$.

References

[1] J. L. Baer and D. P. Bovet, "Compilation of arithmetic expressions for parallel computations," in *Proc. IFIP Congr.*, 1968, pp. 340–346.

[2] R. Brent, "On the addition of binary numbers," *IEEE Trans. Comput.* (Corresp.), vol. C-19, pp. 758–759, Aug. 1970.

[3] D. J. Kuck, Y. Muraoka, and S. Chen, "On the number of operations simultaneously executable in Fortran-like programs and their resulting speedup," *IEEE Trans. Comput.*, vol. C-21, pp. 1293–1303, Dec. 1972.

[4] K. Maruyama, "Parallel methods and bounds of evaluating polynomials," Dep. Comput. Sci., Univ. Illinois, Urbana-Champaign, Mar. 1971; also "On the parallel evaluation of polynomials," *IEEE Trans. Comput.*, vol. C-22, pp. 2–5, Jan. 1973.

[5] I. Munro and M. Paterson, "Optimal algorithm for parallel polynomial evaluation," IBM Res. Rep. RC 3497, Aug. 1971; also in *IEEE-SWAT Proc.*, Oct. 1971.

[6] Y. Muraoka, "Parallelism exposure and exploitation in programs," Ph.D. dissertation, Dep. Comput. Sci., Univ. Illinois, Urbana-Champaign, Rep. 424, Feb. 1971.