Information Processing Letters 44 (1992) 91–94
North-Holland

19 November 1992

Non-associative parallel prefix computation

Rong-Jaye Chen and Yu-Song Hou

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 30050, Taiwan, ROC

Communicated by D. Gries
Received 13 May 1992
Revised 16 July 1992

Abstract

Chen, R.-J. and Y.-S. Hou, Non-associative parallel prefix computation, Information Processing Letters 44 (1992) 91–94.

Associativity of a binary operation allows many applications that are not possible with a non-associative binary operation. We propose a method to transform a binary expression into an associative one and present two related applications.

Keywords: Parallel algorithms; associative binary operation; parallel prefix computation; one-way finite automata

1. Introduction

Associativity is very useful in evaluating binary expressions. For example, without associativity, prefix computation is hard to implement in parallel [1,2,5,6], and a non-associative binary expression with reverse calculation order is difficult to evaluate using a one-way automata (see Section 4). If the corresponding operation is associative, these two examples become easy to compute because the order of computation does not affect the result.

We show how to translate a non-associative operator into an associative one. In Section 2, we define the prefix computation problem with a non-associative operator and present a method to solve it in parallel. Section 3 extends the method to the prefix computation problem with multiple operators. An application to a one-way finite automata design is discussed in Section 4.

Correspondence to: R.-J. Chen, Department of Computer Science and Information Engineering, National Chiao Tung University, 1001 TA Hsueh Road, Hsinchu 30050, Taiwan, ROC Email: rjchen@csunix.csie.nctu.edu.tw.

2. Parallel prefix computation with a non-associative operator

Let $[A, \otimes]$ be an algebraic structure, where A is a finite set and $\otimes$ is a (not necessarily associative) binary operator mapping $A \times A$ to A.

Definition 1. The binary expression $x_1 \otimes x_2 \otimes \cdots \otimes x_n$ is defined as

$$(\cdots((x_1 \otimes x_2) \otimes x_3) \cdots \otimes x_n),$$

for $x_1, x_2, \ldots, x_n \in A$.

The prefix computation problem is to evaluate all products $x_1 \otimes x_2 \otimes \cdots \otimes x_i$, for $i = 1, \ldots, n$, for a given sequence $x_1, x_2, \ldots, x_n$. In [6], a well-known parallel algorithm taking $2D + 1$ steps on a $(2n - 1)$-node tree circuit is presented, where D is the depth of the tree circuit. Prefix computations can be applied to carry-look-ahead addition

[1,5,6], linear recurrences [6, p. 241], and scheduling [2].

For the case of a non-associative operator $\otimes$, we define a function induced by an element in A as follows.

Definition 2. For $x \in A$, function $\otimes_x : A \to A$ is defined by $\otimes_x(y) = y \otimes x$, for every $y \in A$. And we define a constant function I_x as $I_x(y) = x$, for every $y \in A$.

The definition of $\otimes_x$ is similar to currying, as used in lambda calculus, type theories, and functional programming languages [7].

By the definition of $\otimes_x$, we have

$$\begin{aligned} & x_1 \otimes x_2 \otimes \cdots \otimes x_n \\ &= (\cdots((x_1 \otimes x_2) \otimes x_3) \cdots \otimes x_n) \\ &= \left(\cdots\left((\otimes_{x_2}(x_1) \otimes x_3\right) \cdots \otimes x_n\right) \\ &= \otimes_{x_n}\left(\cdots \otimes_{x_3}(\otimes_{x_2}(x_1)) \cdots\right) \\ &= \otimes_{x_n} \circ \cdots \circ \otimes_{x_3} \circ \otimes_{x_2} \circ I_{x_1}(-). \end{aligned}$$

The original binary expression $x_1 \otimes x_2 \otimes \cdots \otimes x_n$ can be regarded as $\otimes_{x_n} \circ \cdots \circ \otimes_{x_3} \circ \otimes_{x_2} \circ I_{x_1}$; and the prefix computation problem becomes a suffix computation problem. Since the composition operator $\circ$ is associative, the technique of parallel prefix computation previously discussed can be applied.

Example 3. $[\{a, b, c\}, \otimes]$ is an algebraic structure with a non-associative operator $\otimes$ whose operator table is defined below:

$\otimes$	a	b	c
a	a	a	c
b	c	a	b
c	b	c	a

Then

$$\begin{aligned} \otimes_a &= \{(a, a), (b, c), (c, b)\}, \\ \otimes_b &= \{(a, a), (b, a), (c, c)\}, \\ \otimes_c &= \{(a, c), (b, b), (c, a)\}, \\ I_a &= \{(a, a), (b, a), (c, a)\}, \\ I_b &= \{(a, b), (b, b), (c, b)\}, \\ I_c &= \{(a, c), (b, c), (c, c)\}. \end{aligned}$$

The prefixes of $a \otimes b \otimes c \otimes a$ are evaluated as follows,

$$\begin{aligned} & a \otimes b = \otimes_b \circ I_a(-) = a, \\ & a \otimes b \otimes c = \otimes_c \circ \otimes_b \circ I_a(-) = c, \\ & a \otimes b \otimes c \otimes a = \otimes_a \circ \otimes_c \circ \otimes_b \circ I_a(-) = b. \end{aligned}$$

We now address the time complexity of the parallel algorithm. Let $|A|$ denote the size of A. By modifying the parallel prefix one [6], we can get a parallel suffix algorithm that takes $O(D)$ steps when implemented on a binary tree circuit of depth D. To compute all the suffixes of $\otimes_{x_n} \circ \cdots \circ \otimes_{x_3} \circ \otimes_{x_2} \circ I_{x_1}$, we adopt Gries' view [3, Chapter 5] to use a one-dimensional array of size $|A|$ to represent a function $\otimes_x$. In accordance with the notation in [3], let $a[1..|A|]$, $b[1..|A|]$ and $c[1..|A|]$ be three arrays (functions). Then the composition $c = a \circ b$ can be defined as

$$(c;\ 1 : a[b[1]];\ 2 : a[b[2]];\ldots;|A| : a[b[|A|]]).$$

So it takes $O(|A|)$ time to compute a composition of two functions. Furthermore, an array (function) movement along an edge on a tree circuit takes $O(|A|)$ time. Since each step in the parallel suffix algorithm consists of at most a function composition and two array movements, we can get an $O(|A| \cdot D)$ parallel algorithm. The time complexity will be $O(|A| \log n)$ on a $(2n-1)$-node complete binary tree circuit (where $D = \log(2n-1)$), which is smaller than the time complexity $O(n)$ of the sequential algorithm when $|A|$ is fixed and n is large.

As for space complexity, each node needs a two-dimensional array of size $|A| \times |A|$ for the operation table and three arrays of size $|A|$ for induced functions, thus amounting to $O(n|A|^2)$. If common memory is available, the operation table can be shared and the space complexity becomes $|A|^2 + O(n|A|)$.

3. Parallel prefix computation with multiple operators

In this section, we extend the computations in Section 2 to a more general class of binary expressions. Assume $[A, \otimes^1, \ldots, \otimes^k]$ is an algebraic structure with k operators ($k \geqslant 1$). These operators need not be associative. The prefix computation problem is to evaluate all prefixes for a given expression

$$x_1 \oplus x_2 \oplus^2 \cdots \oplus^{n-1} x_n$$
$$= \left(\left(\cdots \left(x_1 \oplus^1 x_2 \right) \oplus^2 \cdots \oplus^{n-1} x_n \right) \right),$$

where $\oplus^i \in \{\otimes^1, \ldots, \otimes^k\}$ for all i. We define a family of induced functions as before.

Definition 4. For $x \in A$ an operator $\otimes_x^i$ is defined as $\otimes_x^i : A \to A$, with $\otimes_x^i(y) = y \otimes^i x$, for every $y \in A$.

Similarly,

$$x_1 \oplus^1 x_2 \oplus^2 \cdots \oplus^{n-1} x_n$$
$$= \oplus_{x_n}^{n-1} \circ \cdots \circ \oplus_{x_3}^2 \circ \oplus_{x_2}^1 \circ I_{x_1}(-).$$

The suffixes of $\oplus_{x_n}^{n-1} \circ \cdots \circ \oplus_{x_3}^2 \circ \oplus_{x_2}^1 \circ I_{x_1}$ can be computed in $O(|A| \cdot D)$ by using a $(2n-1)$-node tree circuit.

Applications include Boolean algebra (AND, OR), modular arithmetic (addition modulo m, multiplication modulo m), and recurrence equations. Two examples are shown as follows.

Example 5. $[\{0, 1, 2\}, \otimes^1, \otimes^2]$ is an algebraic structure with two operators, where $\otimes^1$ is addition modulo 3 and $\otimes^2$ is multiplication modulo 3. Their operation tables are shown below:

$\otimes^1$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

$\otimes^2$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Then

$$\otimes_0 = \{(0, 0), (1, 1), (2, 2)\},$$
$$\otimes = \{(0, 1), (1, 2), (2, 0)\},$$
$$\otimes_2 = \{(0, 2), (1, 0), (2, 1)\},$$
$$\otimes_0^2 = \{(0, 0), (1, 0), (2, 0)\},$$
$$\otimes_1^2 = \{(0, 0), (1, 1), (2, 2)\},$$
$$\otimes_2^2 = \{(0, 0), (1, 2), (2, 1)\}.$$

The prefixes of $1 \otimes^1 0 \otimes^2 2 \otimes^1 1$ are evaluated as follows,

$$1 \otimes^1 0 = \otimes_0^1 \circ I_1(-) = 1,$$
$$1 \otimes^1 0 \otimes^2 2 = \otimes_2^2 \circ \otimes_0^1 \circ I_1(-) = 2,$$
$$1 \otimes^1 0 \otimes^2 2 \otimes^1 1 = \otimes_1^1 \circ \otimes_2^2 \circ \otimes_0^1 \circ I_1(-) = 0.$$

Example 6. Consider the following recurrence relations:

$$z_{2i+1} = z_{2i} \otimes^1 c_i,$$
$$z_{2i} = a_i \otimes^2 z_{2i-1} \otimes^1 b_i \otimes^2 z_{2i-2}. \qquad \text{for } i \geqslant 1,$$

given z_0, z_1 are initial values. $\otimes^1$ is addition modulo m and $\otimes^2$ is multiplication modulo m for a positive integer m. Vector–matrix multiplication and vector–vector addition are defined as follows,

$$(a, b)\begin{pmatrix} c & d \\ e & f \end{pmatrix}$$
$$= (a \otimes^2 c \otimes^1 b \otimes^2 e,\ a \otimes^2 d \otimes^1 b \otimes^2 f),$$
$$(a, b) + (c, d) = (a \otimes^1 c,\ b \otimes^1 d).$$

We can rewrite the above recurrence relations as

$$(z_{2i+1}, z_{2i}) = (z_{2i}, z_{2i-1})\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} + (c_i, 0),$$
$$(z_{2i}, z_{2i-1}) = (z_{2i-1}, z_{2i-2})\begin{pmatrix} a_i & 1 \\ b_i & 0 \end{pmatrix}.$$

So

$$(z_{2i+1}, z_{2i}) = (z_{2i-1}, z_{2i-2})\begin{pmatrix} a_i & 1 \\ b_i & 0 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$$
$$+ (c_i, 0),$$

and

$$(z_{2i+1}, z_{2i})$$
$$= \left(\left(\cdots \left((z_1, z_0)\begin{pmatrix} a_1 & 1 \\ b_1 & 0 \end{pmatrix} \right) \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \right) \right.$$
$$\left. + (c_1, 0) \right) \left(\begin{pmatrix} a_2 & 1 \\ b_2 & 0 \end{pmatrix} \right) \left(\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \right)$$
$$+ (c_2, 0)) + \cdots + (c_i, 0)).$$

We compute $z_0, z_1, \ldots, z_{2n+1}$ by evaluating prefixes in the above equation. It takes $O(m^2 D)$ time and needs $O(nm^4)$ space on an $O(n)$-node depth D binary tree circuit.

4. An automata design for string acceptance

Hopcroft and Ullman [4, Exercise 2.8(b)] ask how to design a nondeterministic finite automata (NFA) that accepts the following language L:

"The set of all strings over the alphabet $\{a, b, c\}$ that have the same value when evaluated left to right as right to left by multiplying according to the table in Example 3."

It is easy to use a deterministic finite automata (DFA) to evaluate an expression from left to right. The bottleneck of this problem is evaluation right to left. Since the tape head of the one-way automata cannot rewind, a straightforward method is to use a one-way NFA that "guesses" all possible products of the entire string from right to left. However, the construction is complicated. Instead, we use the concept of the transformation in Section 2 to construct a simple DFA that accepts language L.

In accordance with the notation in [4], we give the definitions of the DFA below.

$\Sigma = \{a, b, c\}$ is an input alphabet.
$\mathfrak{F} = \{f \mid f : \Sigma \to \Sigma\}$.
$f_a = \{(a, a), (b, a), (c, c)\}$.
$f_b = \{(a, c), (b, a), (c, b)\}$.
$f_c = \{(a, b), (b, c), (c, a)\}$.
(Induced function f_x is defined as $f_x(y) = xy$.)

DFA $M = (\mathscr{Q}, \Sigma, \delta, q_0, F)$ is defined as follows.

$\mathscr{Q} = \{[l, f, r] \mid l, r \in \Sigma, f \in \mathfrak{F}\} \cup \{q_0\}$ is the set of states;

q_0 is the initial state;

$F = \{[v, f, v] \mid v \in \Sigma, f \in \mathfrak{F}\}$ is the set of final states;

δ is a transition function, where

$\delta(q_0, x) = [x, f_x, x]$ for $x \in \Sigma$, and

$\delta([x, f, y], z) = [xz, f \circ f_z, f(z)]$ for $x, y, z \in \Sigma$, $f \in \mathfrak{F}$.

State $[l, f, r]$ means that l is the current product from left to right, r is the current product from right to left, and f is the composition of induced functions.

Theorem 7. *DFA M accepts language L.*

Proof. By induction, we can prove that after running the input string $x_1 x_2 \ldots x_n$, M enters the state

$$\left[((\cdots(x_1x_2)\cdots)x_n),\ f_{x_1}\circ f_{x_2}\circ\cdots\circ f_{x_n},\ f_{x_1}\circ f_{x_2}\circ\cdots\circ f_{x_{n-1}}(x_n)\right].$$

Accordingly, $x_1 x_2 \ldots x_n \in L$ if and only if

$$\begin{aligned}&((\cdots(x_1x_2)\cdots x_{n-1})x_n)\\ &\quad = f_{x_1}\circ f_{x_2}\circ\cdots\circ f_{x_{n-1}}(x_n)\\ &\quad = (x_1(x_2\cdots(x_{n-1}x_n))\cdots).\qquad\square\end{aligned}$$

References

[1] R.P. Brent and H.T. Kung, A regular layout for parallel adders, *IEEE Trans. Comput.* **31** (1982) 260–264.

[2] E. Dekel and S. Sahni, Binary trees and parallel scheduling algorithms, *IEEE Trans. Comput.* **32** (1983) 307–315.

[3] D. Gries, *The Science of Programming* (Springer, Berlin, 1981).

[4] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).

[5] R.E. Ladner and M.J. Fischer, Parallel prefix computation, *J. ACM* **27** (1980) 831–838.

[6] T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes* (Morgan Kaufmann, Reading, MA, 1992).

[7] G. Revesz, *Lambda-Calculus, Combinators, and Functional Programming* (Cambridge University Press, Cambridge, 1988).