Parallel Prefix Algorithms on the Multicomputer

LI-LING HUNG YEN-CHUN LIN
Department of Computer Science and Information Engineering
National Taiwan University of Science and Technology
43 Keelung Road, Sec. 4, Taipei 106
TAIWAN
D9115004@mail.ntust.edu.tw yclin747@gmail.com
http://faculty.csie.ntust.edu.tw/~yclin/yclin.htm

Abstract: - A family of computation-efficient parallel prefix algorithms for message-passing multicomputers are presented. The family generalizes a previous algorithm that uses only half-duplex communications, and thus can improve the running time. Several properties of the family are derived, including the number of computation steps, the number of communication steps, and the condition for effective use of the family. The family can adopt collective communication operations to reduce the communication time, and thus becomes a second family of algorithms. These algorithms provide the flexibility of choosing either less computation time or less communication time, depending on the characteristics of the target machine, to achieve the minimal running time.

Key-Words: - Computation-efficient, Cost optimality, Half-duplex, Message-passing multicomputers, Parallel algorithms, Prefix computation

1 Introduction

The prefix problem, or prefix computation, is defined as follows: given n inputs $x_1, x_2, \ldots, x_n$, and an associative binary operator $\oplus$, compute

$$y_i = x_1 \oplus x_2 \oplus \ldots \oplus x_i, \qquad \text{for } 1 \le i \le n.$$

For ease of presentation, unless otherwise stated, this study assumes that x_i's and y_i's represent inputs and outputs, respectively, and the number of inputs is n. Prefix computation has been extensively studied for its wide application in fields such as biological sequence comparison, cryptography, design of silicon compilers, job scheduling, image processing, loop parallelization, polynomial evaluation, processor allocation, and sorting [1-3, 8, 11, 14, 16, 23-26, 28, 51, 52, 54, 56, 57]}. The binary operation $\oplus$ can be as simple as a Boolean operation or an extremely time-consuming multiplication of matrices [12].

Because of its importance and usefulness, prefix computation has been proposed as a primitive operation [4]. In fact, prefix computation is a built-in operation for Message-Passing Interface (MPI) parallel programming [17], and is implemented in hardware in the Thinking Machines CM-5 [50]. Additionally, many parallel prefix algorithms for various parallel computing models have been proposed [1, 7, 9, 13, 19, 24, 26, 29, 30, 36, 41, 42, 44, 46, 51], and many parallel prefix circuits have also been designed [3, 5, 6, 10, 15, 18, 21, 25-27, 30, 32-34, 37-40, 43, 45, 47, 48, 51, 55-57].

In particular, Egecioglu and Koc present a computation-efficient parallel prefix algorithm, henceforth named EK, for the half-duplex multicomputer model with p processing elements (PEs), where $p < n$ [12]. Lin proposes an algorithm, henceforth named L, to reduce the communication time on the same model [31]. Half-duplex communication is the weakest communication model of message-passing multicomputers, with which each PE of a multicomputer can only send or receive a message in a communication step. This model of communication is basic and important [22].

Although a PE of a modern multicomputer can send and receive in the same step, it usually takes a longer time to send and receive than to send or receive only due to the inherent hardware capability and software overhead [20, 49]. On a p-PE system, the half-duplex communication ensures that no more than $p/2$ messages are transferred in a communication step and thus a communication step will not take too much time.

In this paper, computation-efficient parallel prefix algorithms for multicomputers with p PEs, where $n \ge (p^2 + kp + k + 1)/2$, are presented. We first present a family of algorithms that generalize Algorithm L such that they represent multiple combinations of the computation time and

communication time. Algorithm L is at one extreme of the family. The others of the family take less computation time than Algorithm L, but may take more or less communication time. Users can thus take into account both the exact time of performing $\oplus$ and that of communicating a message to choose an algorithm that requires the minimal running time.

Like Algorithms EK and L, the proposed algorithms are more practical when the amount of time required to perform a binary operation $\oplus$ is greater than that required to transfer a message between two PEs. This situation may happen, for example, when the binary operation is time-consuming matrix multiplication.

The rest of this paper is organized as follows. Section 2 presents a family of parallel prefix algorithms for half-duplex message-passing multicomputers. Section 3 gives various properties of the family, including the computation time, communication time, and the condition for effective use of algorithms. Section 4 compares the new algorithms with previous ones for multicomputers. Section 5 discusses possible ways to improve the running time. Conclusions are finally drawn in Section 6.

2 A Family of Parallel Prefix Algorithms

In this section, we describe a family of parallel algorithms for solving the prefix problem on the half-duplex multicomputer model. The p PEs are represented by P_1, P_2,…, P_p. For ease of presentation, $i{:}j$ is used to represent the result of computing $x_i \oplus x_{i+1} \oplus \ldots \oplus x_j$, where $i \le j$.

Algorithm A(n, p, k) {Solving the prefix problem of n inputs, x_1, x_2,…, x_n, using p PEs to generate y_1, y_2,…, y_n, where $n \ge (p^2 + kp + k + 1)/2$, $p = kq + 1$, $k \ge 1$, $q \ge 1$. For ease of presentation, assume that all numerical values are integers.}

Phase 1: Partition the inputs into two parts $N_1 = (x_1, x_2,\ldots, x_v)$ and $N_2 = (x_{v+1}, x_{v+2},\ldots, x_n)$, where $0 < v < n$. How the value of v is determined will be explained shortly. If $p = k + 1$, then P_1 uses N_1 to compute outputs y_1, y_2,…, y_v sequentially; otherwise, P_1, P_2,…, P_{p-k} use N_1 to compute y_1, y_2,…, y_v by invoking A(v, $p - k$, k) recursively. In the mean time, N_2 is first distributed evenly among the other k PEs, P_{p-k+1}, P_{p-k+2},…, P_p; each of the PEs holds $c = (n - v)/k$ input values. These k PEs then concurrently compute

$$z_1 = (z_{1,1}, z_{1,2},\ldots, z_{1,c}),$$

$$z_2 = (z_{2,1}, z_{2,2},\ldots, z_{2,c}),$$

$$\vdots$$

$$z_k = (z_{k,1}, z_{k,2},\ldots, z_{k,c}),$$

respectively, where

$$z_{i,j} = (v + (i - 1)c + 1){:}(v + (i - 1)c + j).$$

The value of v is chosen to make the total number of computation steps in this phase required by the first $p - k$ PEs equal to that required by the other k PEs, and it is given later. Note that y_v is obtained by P_{p-k}.

Phase 2: Initially, P_{p-k} sends y_v to all the other PEs. Next, P_{p-k+1} scatters, i.e., partitions and distributes, z_1 among all the PEs evenly, each PE having c/p of the c values. All the PEs then concurrently compute

$$y_{v+i} = y_v \oplus z_{1,i}, \qquad i = 1, 2,\ldots, c$$

in c/p computation steps. Note that y_{v+c} is computed by P_p.

Phase *m* ($m = 3, 4,\ldots, k + 1$)**:** Initially, P_p sends $y_{v+(m-2)c}$ to all the other PEs. Next, $P_{p-k+m-1}$ scatters z_{m-1} among all the PEs evenly, each PE having c/p values. All the PEs then concurrently compute

$$y_{v+(m-2)c+i} = y_{v+(m-2)c} \oplus z_{m-1,i}, \qquad i = 1, 2,\ldots, c$$

in c/p computation steps. Note that $y_{v+(m-1)c}$ is computed by P_p.

To evaluate Algorithm A(n, p, k), let $C(n, p, k)$ denote the number of computation steps required, and $R(n, p, k)$ denote the number of communication steps. As in the two previous papers [12, 31], the initial input data loading time is not taken into account in this paper. To help understand the algorithm, we give two examples in the following.

First, consider the case when $p = 5$, $k = 4$.

Phase 1: Assign $N_1 = (x_1, x_2,\ldots, x_v)$ to P_1 and $N_2 = (x_{v+1}, x_{v+2},\ldots, x_n)$ to P_2, P_3, P_4, P_5. The prefixes of N_1 are computed by P_1, and those of N_2 by the other PEs. By the rule of deciding v, the number of computation steps required by P_1, $v - 1$, equals the number of parallel computation steps required by the other three PEs, $(n - v)/4 - 1$. Hence, $v = n/5$. After phase 1 has completed, P_1 obtains y_1, y_2,…, $y_{n/5}$; P_2 obtains $z_{1,1}$, $z_{1,2}$,…, $z_{1,n/5}$; P_3 obtains $z_{2,1}$, $z_{2,2}$,…, $z_{2,n/5}$; P_4 obtains $z_{3,1}$, $z_{3,2}$,…, $z_{3,n/5}$; and P_5 obtains $z_{4,1}$, $z_{4,2}$,…, $z_{4,n/5}$. It takes $n/5 - 1$ computation steps.

Phase 2: P_1 initially sends $y_{n/5}$ to P_2, P_3, P_4, P_5 in 4 communication steps. Then, P_2 sends 1/5 of the $n/5$ prefixes computed in phase 1 to each of the other four PEs in 4 communication steps. That is, $z_{1,1}$ through $z_{1,n/25}$, $z_{1,2n/25+1}$ through $z_{1,3n/25}$, $z_{1,3n/25+1}$

through $z_{1,4n/25}$, and $z_{1,4n/25+1}$ through $z_{1,\,n/5}$ are sent to P_1, P_3, P_4, P_5, respectively. Subsequently, the five PEs compute $n/5$ outputs $y_{n/5+1}, y_{n/5+2},\ldots, y_{2n/5}$ in $n/25$ parallel computation steps. At the end, P_5 has $y_{2n/5}$.

Phase 3: P_5 initially sends $y_{2n/5}$ to the other PEs in 4 communication steps. Then, P_3 sends 1/5 of the $n/5$ prefixes computed in phase 1 to each of the other four PEs in 4 communication steps. That is, $z_{2,1}$ through $z_{2,n/25}$, $z_{2,n/25+1}$ through $z_{2,2n/25}$, $z_{2,3n/25+1}$ through $z_{2,4n/25}$, and $z_{2,4n/25+1}$ through $z_{2,n/5}$ are sent to P_1, P_2, P_4, P_5, respectively. Subsequently, the five PEs compute $n/5$ outputs $y_{2n/5+1}, y_{2n/5+2},\ldots, y_{3n/5}$ concurrently in $n/25$ computation steps. At the end, P_5 has $y_{3n/5}$.

Phase 4: P_5 initially sends $y_{3n/5}$ to the other PEs in 4 communication steps. Then, P_4 sends 1/5 of the $n/5$ prefixes computed in phase 1 to each of the other four PEs in 4 communication steps. That is, $z_{3,1}$ through $z_{3,n/25}$, $z_{3,n/25+1}$ through $z_{3,2n/25}$, $z_{3,2n/25+1}$ through $z_{3,3n/25}$, and $z_{3,4n/25+1}$ through $z_{3,n/5}$ are sent to P_1, P_2, P_3, P_5, respectively. Subsequently, the five PEs compute $n/5$ outputs $y_{3n/5+1}, y_{3n/5+2},\ldots, y_{4n/5}$ concurrently in $n/25$ computation steps. At the end, P_5 has $y_{4n/5}$.

Phase 5: P_5 sends $y_{4n/5}$ and 1/5 of the $n/5$ prefixes computed in phase 1 to each of the other four PEs in 4 communication steps. That is, $z_{4,1}$ through $z_{4,n/25}$, $z_{4,n/25+1}$ through $z_{4,2n/25}$, $z_{4,2n/25+1}$ through $z_{4,3n/25}$, and $z_{4,3n/25+1}$ through $z_{4,4n/25}$ are sent to P_1, P_2, P_3, P_4, respectively. Subsequently, the five PEs concurrently compute $n/5$ outputs $y_{4n/5+1}, y_{4n/5+2},\ldots, y_n$ in $n/25$ computation steps.

Therefore, the total number of computation steps is

$$C(n, 5, 4) = (n/5 - 1) + n/25 + n/25 + n/25 + n/25 = 9n/25 - 1. \quad (1)$$

The total number of communication steps is

$$R(n, 5, 4) = 4 \times 2 + 4 \times 2 + 4 \times 2 + 4 = 28.$$

Next, consider the case when $p = 9$, $k = 4$.

Phase 1: Assign $N_1 = (x_1, x_2,\ldots, x_v)$ to the first five PEs, and assign $N_2 = (x_{v+1}, x_{v+2},\ldots, x_n)$ to the last four PEs. From Eq. (1), we know that P_1, P_2, P_3, P_4, P_5 can compute the prefixes of N_1 in $C(v, 5, 4) = 9v/25 - 1$ computation steps. In the mean time, P_6, P_7, P_8, P_9 share N_2 evenly and compute their respective prefixes concurrently, taking $(n - v)/4 - 1$ computation steps. By the rule of deciding v,

$$9v/25 - 1 = (n - v)/4 - 1,$$
$$v = 25n/61.$$

Consequently, each of the last four PEs has $c = (n - v)/4 = 9n/61$ input values. Thus, the prefixes $z_{1,1}$ through $z_{1,9n/61}$ are computed in P_6, $z_{2,1}$ through $z_{2,9n/61}$ in P_7, $z_{3,1}$ through $z_{3,9n/61}$ in P_8, and $z_{4,1}$ through $z_{4,9n/61}$ in P_9. Note that P_5 obtains $y_v = y_{25n/61}$, and

$$C(25n/61, 5, 4) = 9n/61 - 1.$$

Phase 2: P_5 initially sends $y_{25n/61}$ to the other eight PEs in 8 communication steps. Then, P_6 sends 1/9 of the $9n/61$ prefixes computed in phase 1 to each of the other eight PEs in 8 communication steps. Subsequently, the nine PEs compute $9n/61$ outputs $y_{25n/61+1}, y_{25n/61+2},\ldots, y_{34n/61}$ in $n/61$ parallel computation steps. At the end, P_9 has $y_{34n/61}$.

Phase 3: P_9 initially sends $y_{34n/61}$ to the other eight PEs in 8 communication steps. Then, P_7 sends 1/9 of the $9n/61$ prefixes computed in phase 1 to each of the other eight PEs in 8 communication steps. Subsequently, the nine PEs concurrently compute $9n/61$ outputs $y_{34n/61+1}, y_{34n/61+2},\ldots, y_{43n/61}$ in $n/61$ computation steps. At the end, P_9 has $y_{43n/61}$.

Phase 4: P_9 initially sends $y_{43n/61}$ to the other eight PEs in 8 communication steps. Then, P_8 sends 1/9 of the $9n/61$ prefixes computed in phase 1 to each of the other eight PEs in 8 communication steps. Subsequently, the nine PEs concurrently compute $9n/61$ outputs $y_{43n/61+1}, y_{43n/61+2},\ldots, y_{52n/61}$ in $n/61$ computation steps. At the end, P_9 has $y_{52n/61}$.

Phase 5: P_9 initially sends $y_{52n/61}$ and 1/9 of the $9n/61$ prefixes computed in phase 1 to each of the other eight PEs in 8 communication steps. The nine PEs then concurrently compute $9n/61$ outputs $y_{52n/61+1}, y_{52n/61+2},\ldots, y_n$ in $n/61$ computation steps.

Therefore, the total number of computation steps is

$$C(n, 9, 4) = 9n/61 - 1 + (n/61) \times 4 = 13n/61 - 1.$$

The total number of communication steps is

$$R(n, 9, 4) = R(25n/61, 5, 4) + 8 \times 7 = 84.$$

3 Properties of Algorithm A

In this section, we present the various properties of Algorithm A, including the values of v, $C(n, p, k)$, and $R(n, p, k)$. How to achieve the minimum running time is also investigated. Let $v = \alpha_{p,k}\, n$, where $0 < \alpha_{p,k} < 1$.

Theorem 1. When $p = k + 1$, $\alpha_{p,k} = 1/p$; otherwise,

$$\alpha_{p,k} = \frac{p^2 - kp + k + 1}{p^2 + kp + k + 1}.$$

Proof. We consider two cases separately.

Case 1: $p = k + 1$. In phase 1 the number of computation steps required by P_1 equals the number of parallel computation steps required by the other k PEs. That is,

$$\alpha_{p,k} n - 1 = \frac{n - \alpha_{p,k} n}{k} - 1.$$

Thus,

$$\alpha_{p,k} = \frac{1}{k+1} = \frac{1}{p}. \tag{2}$$

Case 2: $p = kq + 1$, where $q \geq 2$. The number of computation steps in phase 1 required by P_1 through P_{p-k} equals that required by P_{p-k+1} through P_p. That is,

$$C(\alpha_{p,k} n, p-k, k) = \frac{n - \alpha_{p,k} n}{k} - 1. \tag{3}$$

Phases 2 through $k + 1$ require a total of $(n - \alpha_{p,k} n)/p$ computation steps. Hence,

$$\begin{aligned} C(n,p,k) &= C(\alpha_{p,k} n, p-k, k) + \frac{n - \alpha_{p,k} n}{p} \\ &= \frac{n - \alpha_{p,k} n}{k} - 1 + \frac{n - \alpha_{p,k} n}{p} \\ &= \frac{n(p+k)(1-\alpha_{p,k})}{kp} - 1. \end{aligned} \tag{4}$$

Substituting n and p in Eq. (4) with $\alpha_{p,k} n$ and $p - k$, respectively, we obtain

$$C(\alpha_{p,k} n, p-k, k) = \alpha_{p,k} n \frac{p(1-\alpha_{p-k,k})}{k(p-k)} - 1. \tag{5}$$

From Eqs. (3) and (5) we have

$$\frac{n - \alpha_{p,k} n}{k} - 1 = \frac{p(\alpha_{p,k} n - \alpha_{p-k,k} \alpha_{p,k} n)}{k(p-k)} - 1,$$

$$\alpha_{p,k} = \frac{p-k}{2p - k - p\alpha_{p-k,k}}. \tag{6}$$

Then, let

$$r_i = ip - ki(i+1)/2$$

and

$$s_i = (i+1)p - ki(i+1)/2.$$

We prove by induction on i that

$$\alpha_{p,k} = \frac{r_i - r_{i-1}\alpha_{p-ki,k}}{s_i - s_{i-1}\alpha_{p-ki,k}}, \qquad \text{for } i \geq 1. \tag{7}$$

Base step: Since $r_0 = 0$, $s_0 = p$, $r_1 = p - k$, and $s_1 = 2p - k$, we have

$$\frac{r_1 - r_0\alpha_{p-k,k}}{s_1 - s_0\alpha_{p-k,k}} = \frac{p-k}{2p - k - p\alpha_{p-k,k}}.$$

From Eq. (6), we have

$$\alpha_{p,k} = \frac{r_1 - r_0\alpha_{p-k,k}}{s_1 - s_0\alpha_{p-k,k}}.$$

Induction step: Assume

$$\alpha_{p,k} = \frac{r_t - r_{t-1}\alpha_{p-kt,k}}{s_t - s_{t-1}\alpha_{p-kt,k}}. \tag{8}$$

We are to show

$$\alpha_{p,k} = \frac{r_{t+1} - r_t\alpha_{p-k(t+1),k}}{s_{t+1} - s_t\alpha_{p-k(t+1),k}}. \tag{9}$$

Substituting p in Eq. (6) with $p - kt$, we obtain

$$\alpha_{p-kt,k} = \frac{p - kt - k}{2(p-kt) - k - (p-kt)\alpha_{p-k(t+1),k}}.$$

Thus, Eq. (8) can be rewritten

$$\alpha_{p,k} = \frac{r_t - r_{t-1}\dfrac{p-kt-k}{2p-2kt-k-(p-kt)\alpha_{p-k(t+1),k}}}{s_t - s_{t-1}\dfrac{p-kt-k}{2p-2kt-k-(p-kt)\alpha_{p-k(t+1),k}}}$$

$$= \frac{r_t(2p-2kt-k-(p-kt)\alpha_{p-k(t+1),k}) - r_{t-1}(p-kt-k)}{s_t(2p-2kt-k-(p-kt)\alpha_{p-k(t+1),k}) - s_{t-1}(p-kt-k)}$$

$$= \frac{r_t(2p-2kt-k) - r_{t-1}(p-kt-k) - r_t(p-kt)\alpha_{p-k(t+1),k}}{s_t(2p-2kt-k) - s_{t-1}(p-kt-k) - s_t(p-kt)\alpha_{p-k(t+1),k}} \tag{10}$$

By definition, we have

$$r_{t+1} = (t+1)p - k(t+1)(t+2)/2,$$
$$r_t = tp - kt(t+1)/2,$$
$$r_{t-1} = (t-1)p - k(t-1)t/2.$$

These lead to

$$r_{t-1} = r_t - p + kt,$$
$$r_{t+1} = r_t + p - kt - k.$$

Thus,

$$\begin{aligned} &r_t(2p - 2kt - k) - r_{t-1}(p - kt - k) \\ &= r_t(2p - 2kt - k) \\ &\quad - (r_t - p + kt)(p - kt - k) - (p - kt)(r_t + p - kt - k) \\ &= (p - kt)r_{t+1}. \end{aligned} \tag{11}$$

By definition, we have

$$s_{t+1} = (t+2)p - k(t+1)(t+2)/2,$$
$$s_t = (t+1)p - kt(t+1)/2,$$

$$s_{t-1} = tp - k(t-1)t/2.$$

These lead to

$$s_{t-1} = s_t - p + kt,$$
$$s_{t+1} = s_t + p - kt - k.$$

Thus,

$$s_t(2p - 2kt - k) - s_{t-1}(p - kt - k)$$
$$= s_t(2p - 2kt - k) - (s_t - p + kt)(p - kt - k)$$
$$= (p - kt)(s_t + p - kt - k)$$
$$= (p - kt)s_{t+1}. \quad (12)$$

Using Eqs. (11) and (12), we see that Eq. (10) can be rewritten

$$\alpha_{p,k} = \frac{(p-kt)r_{t+1} - r_t(p-kt)\alpha_{p-k(t+1),k}}{(p-kt)s_{t+1} - s_t(p-kt)\alpha_{p-k(t+1),k}}.$$

This can be reduced to Eq. (9), and thus proves Eq. (7).

Setting $i = (p - k - 1)/k$ for Eq. (7), we have

$$\alpha_{p,k} = \frac{r_i - r_{i-1}\alpha_{k+1,k}}{s_i - s_{i-1}\alpha_{k+1,k}}, \qquad i = (p - k - 1)/k.$$

From Eq. (2),

$$\alpha_{k+1,k} = 1/(k + 1).$$

In addition, the definition of r_i implies

$$r_i = r_{i-1} + p - ki,$$

and the definition of s_i implies

$$s_i = s_{i-1} + p - ki.$$

Thus,

$$\alpha_{p,k} = \frac{r_{i-1} + p - ki - \frac{r_{i-1}}{k+1}}{s_{i-1} + p - ki - \frac{s_{i-1}}{k+1}}$$
$$= \frac{\frac{k}{k+1}r_{i-1} + k + 1}{\frac{k}{k+1}s_{i-1} + k + 1}, \qquad i = (p - k - 1)/k. \quad (13)$$

Since $i - 1 = (p - 2k - 1)/k$, from the definitions of r_i and s_i, we have

$$r_{i-1} = \frac{p-2k-1}{k}p - \frac{k}{2}\frac{p-2k-1}{k}\frac{p-k-1}{k},$$

$$s_{i-1} = \frac{p-k-1}{k}p - \frac{k}{2}\frac{p-2k-1}{k}\frac{p-k-1}{k}.$$

Thus, Eq. (13) can be written

$$\alpha_{p,k} = \frac{\frac{k}{k+1}(\frac{p-2k-1}{k}p - \frac{k}{2}\frac{p-2k-1}{k}\frac{p-k-1}{k}) + k + 1}{\frac{k}{k+1}(\frac{p-k-1}{k}p - \frac{k}{2}\frac{p-2k-1}{k}\frac{p-k-1}{k}) + k + 1}$$
$$= \frac{\frac{k}{k+1}\frac{p^2 - pk - 2k^2 - 3k - 1}{2k} + k + 1}{\frac{k}{k+1}\frac{p^2 + pk - 2k^2 - 3k - 1}{2k} + k + 1}$$
$$= \frac{\frac{p^2 - pk - 2k^2 - 3k - 1}{2(k+1)} + k + 1}{\frac{p^2 + pk - 2k^2 - 3k - 1}{2(k+1)} + k + 1}$$
$$= \frac{p^2 - pk - 2k^2 - 3k - 1 + 2(k+1)(k+1)}{p^2 + pk - 2k^2 - 3k - 1 + 2(k+1)(k+1)}$$
$$= \frac{p^2 - pk + k + 1}{p^2 + pk + k + 1}. \quad \text{Q.E.D.}$$

Theorem 2. $C(n, p, k) = \frac{2n(p+k)}{p^2+kp+k+1} - 1.$

Proof. We consider two cases separately.

Case 1: $p = k + 1$. In phase 1 of Algorithm A(n, p, k), P_1 takes $\alpha_{p,k}\, n - 1$ computation steps to sequentially compute the prefixes of the $\alpha_{p,k}\, n$ inputs assigned, and the number of computation steps required by the other k PEs is also $\alpha_{p,k}\, n - 1$. In phases 2 through $k + 1$, totally $(n - \alpha_{p,k}\, n)/p$ computation steps are required to compute $n - \alpha_{p,k}\, n$ values, precisely y_{v+1}, y_{v+2},…, y_n, by all the p PEs concurrently. Hence,

$$C(n, p, k) = \alpha_{p,k}n - 1 + \frac{n - \alpha_{p,k}n}{p}.$$

From Theorem 1, $\alpha_{p,k} = \frac{1}{p}$; thus,

$$C(n, p, k) = \frac{n}{p} - 1 + \frac{n - n/p}{p} = n\frac{2p-1}{p^2} - 1.$$

Since $p = k + 1$, we have

$$C(n, p, k) = 2n\frac{2p-1}{2p^2} - 1$$
$$= 2n\frac{p+k}{p^2+p(k+1)} - 1$$
$$= \frac{2n(p+k)}{p^2+kp+k+1} - 1.$$

Case 2: $p = kq + 1$, where $q \geq 2$. The number of computation steps in phase 1 required by P_{p-k+1} through P_p is

$$\frac{n-\alpha_{p,k}n}{k}-1.$$

As already mentioned in case 1, phases 2 through $k+1$ require $(n-\alpha_{p,k}n)/p$ computation steps. Hence, totally

$$C(n,p,k)=(\frac{n-\alpha_{p,k}n}{k}-1)+\frac{n-\alpha_{p,k}n}{p}$$

$$=\frac{n(p+k)(1-\alpha_{p,k})}{kp}-1.$$

By Theorem 1, we then have

$$C(n,p,k)=\frac{n(p+k)(1-\frac{p^2-kp+k+1}{p^2+kp+k+1})}{kp}-1$$

$$=\frac{2n(p+k)}{p^2+kp+k+1}-1. \qquad \text{Q.E.D.}$$

We can then examine what values of p and k can achieve the minimal computation time. Let $a = bq_1 + 1$, $d = eq_2 + 1$, and $b, e, q_1, q_2 \geq 1$. Then

$$C(n,a,b)-C(n,d,e)$$

$$=\frac{2n(a+b)}{a^2+ab+b+1}-\frac{2n(d+e)}{d^2+de+e+1}$$

$$=2n\frac{(a+b)(d^2+de+e+1)-(a^2+ab+b+1)(d+e)}{(a^2+ab+b+1)(d^2+de+e+1)}.$$

The numerator above can be rewritten

$$ad^2+ade+ae+a+bd^2+bde+be+b$$
$$-(a^2d+abd+bd+d+a^2e+abe+be+e)$$
$$=ad(d-a)+ad(e-b)+(ae-bd)+(a-d)$$
$$+(bd^2-a^2e)+be(d-a)+(b-e).$$

If $p = a = d$, the numerator becomes

$$a^2(e-b)+a(e-b)+a^2(b-e)+(b-e)$$
$$=(e-b)(a-1).$$

Clearly, if $e > b$, then the numerator is positive, and thus $C(n, p, b) > C(n, p, e)$; i.e., A(n, p, e) takes less computation time than A(n, p, b).

On the other hand, if $k = b = e$, the numerator becomes

$$ad(d-a)+b(a-d)+(a-d)+b(d^2-a^2)$$
$$+b^2(d-a)$$
$$=(d-a)(ad-b-1+ab+bd+b^2).$$

Clearly, if $d > a$, then the numerator is positive, and thus $C(n, a, k) > C(n, d, k)$; i.e., A(n, d, k) takes less computation time than A(n, a, k).

We now summarize the effect of p and k on the computation time as follows. Using as many PEs as possible and using the maximal k, which equals $p-1$, can achieve the minimal computation time. This can be expressed as the following theorem.

Theorem 3. If $d > a$ and $e > b$, then $C(n, a, b) > C(n, d, b) > C(n, d, e)$.

Theorem 4. $R(n, p, 1) = p(p-1)$;

$R(n, p, k) = (2k-1)(p-1)(p+k-1)/2k$ for $k \geq 2$.

Proof. From Algorithm A, $R(n, p, k)$ is the sum of the following four components:

(i) The number of communication steps, $R(v, p-k, k)$, when the first $p-k$ PEs perform A$(v, p-k, k)$ in phase 1.
(ii) The number of communication steps required by P_{p-k} to send y_v to the other $p-1$ PEs in phase 2.
(iii) The number of communication steps required by P_p to send y_{v+ic} to the other $p-1$ PEs in phase $i+2$, for $i = 1, 2,\ldots, k-1$.
(iv) The number of communication steps taken to distribute a total of kc, or $n-v$, values evenly among all the p PEs in phases 2 through $k+1$. Note that the kc values are $z_{i,j}$ for $i = 1, 2,\ldots, k$, and $j = 1, 2,\ldots, c$, which are obtained in phase 1 of Algorithm A.

The value of $R(n, p, k)$ depends on the values of p and k. Thus, we consider the following three cases.

Case 1: $k = 1$. The algorithm has only two phases, and it degenerates into Algorithm L. It has been shown [31]

$$R(n, p, 1) = p(p-1).$$

Case 2: $k \geq 2$ and $p = k+1$. Component (i) is 0; component (ii) is $p-1$; component (iii) is $(k-1)(p-1)$, and component (iv) is $k(p-1)$. Note that in phase $k+1$, component (iii) has the same communication source and destinations as component (iv). These two components can become one in phase $k+1$; that is, $p-1$ communication steps can be reduced. Hence,

$$R(n,p,k)$$
$$=0+(p-1)+(k-1)(p-1)+k(p-1)-(p-1)$$
$$=(p-1)(2k-1), \qquad (14)$$

which is equal to $(2k-1)(p-1)(p+k-1)/2k$ since $p = k+1$.

Case 3: $k \geq 2$ and $p = k(i + 1) + 1$, where $i \geq 1$. Component (i) is $R(v, p - k, k)$. Components (ii), (iii), and (iv) are the same as those in case 2. Thus, using Eq. (14), we have

$$\begin{aligned}
&R(n, p, k)\\
&= R(\alpha_{p,k}\, n, p - k, k) + (p - 1)(2k - 1)\\
&= R(\alpha_{p-k,k}\, \alpha_{p,k}\, n, p - 2k, k) + (p - k - 1)(2k - 1)\\
&\quad + (p - 1)(2k - 1)\\
&\vdots\\
&= R(\alpha_{p-(i-1)k,k} \ldots \alpha_{p-k,k}\, \alpha_{p,k}\, n, p - ik, k)\\
&\quad + (2k - 1)[(p - (i - 1)k - 1) + ... + (p - k - 1)\\
&\quad + (p - 1)]\\
&= R(\alpha_{p-(i-1)k,k} \ldots \alpha_{p-k,k}\, \alpha_{p,k}\, n, p - ik, k)\\
&\quad + (2k - 1)[i(p - 1) - (k + 2k + \ldots + (i - 1)k)]\\
&= R(\alpha_{2k+1,k} \ldots \alpha_{p-k,k}\, \alpha_{p,k}\, n, k + 1, k)\\
&\quad + (2k - 1)[i(p - 1) - \frac{ki(i-1)}{2}]\\
&= R(\alpha_{2k+1,k} \ldots \alpha_{p-k,k}\, \alpha_{p,k}\, n, k + 1, k)\\
&\quad + (2k - 1)[\frac{p-k-1}{k}(p - 1) - (p - k - 1)\frac{p-2k-1}{2k}].
\end{aligned}$$

Since Eq. (14) can be rewritten

$$R(n, k + 1, k) = k\,(2k - 1),$$

which is independent of the value of n, we obtain

$$\begin{aligned}
R(n, p, k) &= k\,(2k - 1)\\
&\quad + (2k - 1)[\frac{p-k-1}{k}(p - 1) - (p - k - 1)\frac{p-2k-1}{2k}]\\
&= (2k - 1)\frac{p^2-2p+kp-k+1}{2k}\\
&= (2k - 1)(p - 1)(p + k - 1)/2k. \qquad \text{Q.E.D.}
\end{aligned}$$

Note that the communication time is independent of n. We now use Theorem 4 to investigate what values of p and k can achieve the minimal communication time. Clearly, a smaller p results in less communication time.

To see the effect of k, we first note that when $k \geq 2$, a smaller k leads to less communication time. Next, we compare $R(n, p, 1)$ and $R(n, p, k)$, where $k \geq 2$. From Theorem 4,

$$\begin{aligned}
&R(n, p, 1) - R(n, p, k)\\
&= (p^2 - p) - (2k - 1)(p - 1)(p + k - 1)/2k\\
&= (p - 1)(p - (2k^2 - 3k + 1))/2k.
\end{aligned}$$

Therefore, when $p \geq 2k^2 - 3k + 1$, $R(n, p, k) \leq R(n, p, 1)$; otherwise, $R(n, p, 1) < R(n, p, k)$.

We summarize the effect of k on the communication time as follows. If $p \geq 2k^2 - 3k + 1$ and $k \geq 2$, then $R(n, p, k) \leq R(n, p, 1)$, and $R(n, p, 2)$ is the minimal number of communication steps; otherwise, $R(n, p, k) > R(n, p, 1)$, and $R(n, p, 1)$ is the minimal number of communication steps. Together with the effect of p on the communication time, we have the following theorem.

Theorem 5. If $d > a$, then $R(n, d, k) > R(n, a, k)$. If $p \geq 2k^2 - 3k + 1$ and $k \geq 2$, then $R(n, p, k) \leq R(n, p, 1)$ and $R(n, p, 2) \leq R(n, p, k) < R(n, p, k + 1)$; otherwise, $R(n, p, 1) < R(n, p, k) < R(n, p, k + 1)$.

Let τ be the ratio of the time required by a communication step to the time required by a computation step. Thus, the total running time of the algorithm is equivalent to the time required to perform $C(n, p, k) + \tau\, R(n, p, k)$ computation steps.

From Theorem 2, $C(n, p, k) = \Theta(n/p)$; from Theorem 4, $R(n, p, k) = \Theta(p^2)$. Totally, Algorithm A takes

$$C(n, p, k) + \tau\, R(n, p, k) = \Theta(n/p) + \Theta(p^2)$$

time. When $n = \Omega(p^3)$, $n/p = \Omega(p^2)$; thus,

$$C(n, p, k) + \tau\, R(n, p, k) = \Theta(n/p).$$

Since the sequential solution for the prefix problem takes $\Theta(n)$ time, and $\Theta(n/p) \times p = \Theta(n)$, we have the following theorem.

Theorem 6. Algorithm A is cost optimal when $n = \Omega(p^3)$.

Let us use two examples to gain more insight into Algorithm A. Suppose $n = 2048$, $p = 511$, and $k = 255$. From Theorem 2, we have $C(2048, 511, 255) < 8$. However, it is even impossible to compute the sum of 2048 inputs in 8 computation steps. Therefore, Theorem 2 does not hold under this situation. We need to dig further to understand and solve this problem.

As a more general case, suppose $n = 2048$, $p = 511 = kq + 1$, and $q \geq 1$. From Theorem 1,

$$v = \frac{2048(511^2 - 511k + k + 1)}{(511^2 + 511k + k + 1)}$$

inputs are assigned to the first $511 - k$ PEs, and

$$n - v = 2kpn/(p^2 + kp + k + 1)$$

ISSN: 1991-8755 Issue 4, Volume 3, April 2008

$$= 1022\times 2048k/(511^2 + 511k + k + 1)$$
$$< 9k$$

inputs are assigned to the last k PEs. That is, each of the last k PEs has at most 9 inputs. Then, in phase 2, P_{512-k} scatters at most 9 values to at most 9 of the 511 PEs for further computation, and thus at least 502 PEs are idle while the others are communicating and computing. This ineffective use of PEs happens in every phase except for phase 1.

Thus, in phase 1 at least kp inputs should be assigned to the last k PEs, which guarantees that in any later phase each PE can be assigned at least one value to compute. Using $n - v \geq kp$ and Theorem 1, we obtain

$$n - n(p^2 - kp + k + 1)/(p^2 + kp + k + 1) \geq kp,$$
$$n \geq (p^2 + kp + k + 1)/2.$$

Therefore, we have following theorem.

Theorem 7. To use Algorithm A(n, p, k) effectively, it is required that $n \geq (p^2 + kp + k + 1)/2$.

The same condition can be derived by another approach. Snir [49] has proved that the number of computation steps needed when using p PEs to compute the prefixes of n inputs, where $n > p$, must satisfy

$$C(n, p, k) \geq (2n - 2)/(p + 1).$$

Thus, after assigning v inputs to the first $p - k$ PEs, the number of computation steps needed to recursively use the $p - k$ PEs to solve a prefix problem of v inputs is

$$C(v, p - k, k) \geq (2v - 2)/(p - k + 1)$$
$$= \frac{2n\frac{p^2 - kp + k + 1}{p^2 + kp + k + 1} - 2}{p - k + 1}.$$

In addition, by the rule of choosing v, the number of computation steps, $C(v, p - k, k)$, required by the first $p - k$ PEs is equal to the number of computation steps, $(n - v)/k - 1$, required by the last k PEs. That is,

$$C(v, p - k, k) = (n - v)/k - 1$$
$$= 2pn/(p^2 + kp + k + 1) - 1.$$

Therefore, from the above relations, once more we obtain

$$n \geq (p^2 + kp + k + 1)/2.$$

4 Comparisons

Lin and Lin present a parallel prefix algorithm named PLL for the half-duplex multicomputer [36]; PLL requires $2n/p + 1.44 \log_2 p - 1$ computation steps and $1.44 \log_2 p + 1$ communication steps when using p PEs, where $10 \leq p < n$. The number of computation steps of Algorithm A is less than that of PLL, but the number of communication steps is greater than that of PLL. When τ is small, A may be faster than PLL.

Since Algorithm L is a special case of A(n, p, k), precisely A(n, p, 1), it takes $C(n, p, 1)$ computation steps and $R(n, p, 1)$ communication steps. We have shown that a larger k results in less computation time, i.e., $C(n, p, 1) > C(n, p, k)$ for $k \geq 2$. As for the communication time, we have shown that $R(n, p, k) \leq R(n, p, 1)$ when $p \geq 2k^2 - 3k + 1$; otherwise, $R(n, p, k) > R(n, p, 1)$. Therefore, A(n, p, k) is definitely faster than L when $p \geq 2k^2 - 3k + 1$ and $k \geq 2$. However, when $p < 2k^2 - 3k + 1$, we must know the value of τ to decide which algorithm is faster.

5 Discussion

It is more important to decide the best values of p and k that can achieve the least running time than to obtain the minimum computation time or communication time. As already mentioned, a larger k or p leads to less computation time. However, a smaller p or k results in less communication time, except when $p \geq 2k^2 - 3k + 1$ and $k \geq 2$. When $p \geq 2k^2 - 3k + 1$ and $k \geq 2$, A(n, p, k) is definitely faster than A(n, p, 1), but it is difficult to determine analytically the values of p and k that result in the minimal running time. Fortunately, we can determine the minimal running time by choosing appropriate p and k values.

Note that Algorithm A(n, p, k) requires that $p = kq + 1$, $k \geq 1$, and $q \geq 1$. Thus, when $p = 98$, for example, k can only be 1 or 97. That is, when $p - 1$ is prime, there are only two combinations of k and q. However, when $p = 97$, k can be 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, or 96. Since the running time of A(n, p, k) depends on the values of n, p, k, and τ, A(n, 97, k) seems to have a better chance to be faster than A(n, 98, k). Thus, in some cases, we may want to use fewer PEs to run faster. It is desirable that an algorithm guarantees faster execution using all available PEs than using fewer PEs.

It may be possible to reduce the communication time of Algorithm A by using collective communications, such as broadcast and scatter. Collective communication operations can be easier to program and run faster than a sequence of send or

receive operations [53]. We can thus use broadcast and scatter to replace the primitive send and receive in Algorithm A to improve the communication time, and obtain one more family of algorithms.

With the above changes, which involve only communications, most of the properties of Algorithm A presented in Section 3 are not affected. The communication time and running time can be shorter. The resulting family of algorithms are cost optimal when $n = \Omega(p^2 \log p)$ [35].

6 Conclusion

We have presented a family of parallel prefix algorithms A(n, p, k) run on half-duplex multicomputers with p PEs to solve the prefix problem of n inputs, where $p = kq + 1$, $k \geq 1$, $q \geq 1$, and $n \geq (p^2 + kp + k + 1)/2$. The numbers of computation steps and communication steps have been derived. We can determine the minimal running time by choosing appropriate p and k values. The new algorithms are cost optimal when $n = \Omega(p^3)$. When $p \geq 2k^2 - 3k + 1$ and $k \geq 2$, A(n, p, k) is definitely faster than the previous Algorithm L. Otherwise, whether the proposed algorithms are faster than other prefix algorithms hinges on the ratio of the time required by a communication step to the time required by a computation step.

A(n, p, k) can be modified to be another family using broadcasts and scatters to reduce the communication time. The resulting family is cost optimal when $n = \Omega(p^2 \log p)$.

Acknowledgment

This research was supported in part by the National Science Council of Taiwan under contract NSC 91-2218-E-011-002.

References:

[1] S. G. Akl, *Parallel Computation: Models and Methods*, Prentice-Hall, 1997.

[2] S. Aluru, N. Futamura, and K. Mehrotra, Parallel biological sequence comparison using prefix computations, *Journal of Parallel and Distributed Computing,* Vol. 63, No. 3, 2003, pp. 264-272.

[3] A. Bilgory, D. D. Gajski, A heuristic for suffix solutions, *IEEE Transactions on Computers,* Vol. C-35, No. 1, 1986, pp. 34-42.

[4] G. E. Blelloch, Scans as primitive operations, *IEEE Transactions on Computers,* Vol. 38, No. 11, 1989, pp. 1526-1538.

[5] R. P. Brent, H. T. Kung, A regular layout for parallel adders, *IEEE Transactions on Computers,* Vol. C-31, No. 3, 1982, pp. 260-264.

[6] D. A. Carlson, B. Sugla, Limited width parallel prefix circuits, *Journal of Supercomputing,* Vol. 4, No. 2, 1990, pp. 107-129.

[7] L. Cinque, G. Bongiovanni, Parallel prefix computation on a pyramid computer, *Pattern Recognition Letters,* Vol. 16, No. 1, 1995, pp. 19-22.

[8] R. Cole, U. Vishkin, Faster optimal parallel prefix sums and list ranking, *Information and Control,* Vol. 81, No. 3, 1989, pp. 334-352.

[9] A. Datta, Multiple addition and prefix sum on a linear array with a reconfigurable pipelined bus system, *Journal of Supercomputing,* Vol. 29, No. 3, 2004, pp. 303-317.

[10] G. Dimitrakopoulos, D. Nikolos, High-speed parallel-prefix VLSI Ling adders, *IEEE Transactions on Computers,* Vol. 54, No. 2, 2005, pp. 225-231.

[11] C. Efstathiou, H. T. Vergos, and D. Nikolos, Fast parallel-prefix modulo $2^n + 1$ adders, *IEEE Transactions on Computers,* Vol. 53, No. 9, 2004, pp. 1211-1216.

[12] O. Egecioglu, C. K. Koc, Parallel prefix computation with few processors, *Computers and Mathematics with Applications,* Vol. 24, No. 4, 1992, pp. 77-84.

[13] S. C. Eisenstat, O(log* n) algorithms on a Sum-CRCW PRAM, *Computing,* Vol. 79, No. 1, 2007, pp. 93-97.

[14] A. Ferreira, S. Ubeda, Parallel complexity of the medial axis computation, in *Proceedings of International Conference on Image Processing*, Washington, D.C., 1995, pp. 105-108.

[15] F. E. Fich, New bounds for parallel prefix circuits, in *Proceedings of 15th Symposium on the Theory of Computing*, 1983, pp. 100-109.

[16] A. L. Fisher, A. M. Ghuloum, Parallelizing complex scans and reductions, in *Proceedings of ACM SIGPLAN '94 Conference on Programming Language Design and Implementation*, Orlando, FL, 1994, pp. 135-146.

[17] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, 1994.

[18] T. Han, D. A. Carlson, Fast area-efficient VLSI adders, in *Proceedings of 8th Computer Arithmetic Symposium*, Como, Italy, 1987, pp. 49-56.

[19] D. R. Helman, J. JaJa, Prefix computations on symmetric multiprocessors, *Journal of Parallel and Distributed Computing,* Vol. 61, 2001, pp. 265-278.

[20] Inmos, *The Transputer Databook*, 3rd ed., Inmos, 1992.

[21] P. M. Kogge, H. S. Stone, A parallel algorithm for the efficient solution of a general class of recurrence equations, *IEEE Transactions on Computers,* Vol. C-22, No. 8, 1973, pp. 783-791.

[22] D. W. Krumme, G. Cybenko, and K. N. Venkataraman, Gossiping in minimal time, *SIAM Journal on Computing,* Vol. 21, No. 1, 1992, pp. 111-139.

[23] C. P. Kruskal, T. Madej, and L. Rudolph, Parallel prefix on fully connected direct connection machines, in *Proceedings of International Conference on Parallel Processing*, St. Charles, IL, 1986, pp. 278-284.

[24] C. P. Kruskal, L. Rudolph, and M. Snir, The power of parallel prefix, *IEEE Transactions on Computers,* Vol. C-34, 1985, pp. 965-968.

[25] R. E. Ladner, M. J. Fischer, Parallel prefix computation, *Journal of the Association for Computing Machinery,* Vol. 27, No. 4, 1980, pp. 831-838.

[26] S. Lakshmivarahan, S. K. Dhall, *Parallel Computing Using the Prefix Problem*, Oxford University Press, 1994.

[27] S. Lakshmivarahan, C. M. Yang, and S. K. Dhall, On a new class of optimal parallel prefix circuits with (size + depth) = $2n - 2$ and $\lceil \log n \rceil \leq$ depth $\leq (2 \lceil \log n \rceil - 3)$, in *Proceedings of International Conference on Parallel Processing*, St. Charles, IL, 1987, pp. 58-65.

[28] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, 1992.

[29] R. Lin, K. Nakano, S. Olariu, M. C. Pinotti, J. L. Schwing, and A. Y. Zomaya, Scalable hardware-algorithms for binary prefix sums, *IEEE Transactions on Parallel and Distributed Systems,* Vol. 11, No. 8, 2000, pp. 838-850.

[30] Y.-C. Lin, Optimal parallel prefix circuits with fan-out 2 and corresponding parallel algorithms, *Neural, Parallel & Scientific Computations,* Vol. 7, No. 1, 1999, pp. 33-42.

[31] Y.-C. Lin, A family of computation-efficient parallel prefix algorithms, *WSEAS Transactions on Computers,* Vol. 5, No. 12, 2006, pp. 3060-3066.

[32] Y.-C. Lin, J.-N. Chen, *Z*4: A new depth-size optimal parallel prefix circuit with small depth, *Neural, Parallel & Scientific Computations,* Vol. 11, No. 3, 2003, pp. 221-235.

[33] Y.-C. Lin, J.-W. Hsiao, A new approach to constructing optimal parallel prefix circuits with small depth, *Journal of Parallel and Distributed Computing,* Vol. 64, No. 1, 2004, pp. 97-107.

[34] Y.-C. Lin, Y.-H. Hsu, and C.-K. Liu, Constructing *H*4, a fast depth-size optimal parallel prefix circuit, *Journal of Supercomputing,* Vol. 24, No. 3, 2003, pp. 279-304.

[35] Y.-C. Lin, L.-L. Hung, Four Families of Computation-Efficient Parallel Prefix Algorithms for Multicomputers, Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, Technical Report: NTUST-CSIE-08-01, February 2008.

[36] Y.-C. Lin, C. M. Lin, Efficient parallel prefix algorithms on multicomputers, *Journal of Information Science and Engineering,* Vol. 16, No. 1, 2000, pp. 41-64.

[37] Y.-C. Lin, C.-K. Liu, Finding optimal parallel prefix circuits with fan-out 2 in constant time, *Information Processing Letters,* Vol. 70, No. 4, 1999, pp. 191-195.

[38] Y.-C. Lin, C.-C. Shih, Optimal parallel prefix circuits with fan-out at most 4, in *Proceedings of 2nd IASTED International Conference on Parallel and Distributed Computing and Networks*, Brisbane, Australia, 1998, pp. 312-317.

[39] Y.-C. Lin, C.-C. Shih, A new class of depth-size optimal parallel prefix circuits, *Journal of Supercomputing,* Vol. 14, No. 1, 1999, pp. 39-52.

[40] Y.-C. Lin, C.-Y. Su, Faster optimal parallel prefix circuits: New algorithmic construction, *Journal of Parallel and Distributed Computing,* Vol. 65, No. 12, 2005, pp. 1585-1595.

[41] Y.-C. Lin, C.-S. Yeh, Efficient parallel prefix algorithms on multiport message-passing systems, *Information Processing Letters,* Vol. 71, No. 2, 1999, pp. 91-95.

[42] Y.-C. Lin, C.-S. Yeh, Optimal parallel prefix on the postal model, *Journal of Information Science and Engineering,* Vol. 19, No. 1, 2003, pp. 75-83.

[43] J. Liu, S. Zhou, H. Zhu, and C.-K. Cheng, An algorithmic approach for generic parallel adders, in *Proceedings of International Conference on Computer-Aided Design*, San Jose, CA, 2003, pp. 734-740.

[44] R. Manohar, J. A. Tierno, Asynchronous parallel prefix computation, *IEEE Transactions on Computers,* Vol. 47, No. 11, 1998, pp. 1244-1252.

[45] J. H. Park, H. K. Dai, Reconfigurable hardware solution to parallel prefix computation, *Journal of Supercomputing,* Vol. 43, No. 1, 2008, pp. 43-58.

[46] E. E. Santos, Optimal and efficient algorithms for summing and prefix summing on parallel machines, *Journal of Parallel and Distributed Computing,* Vol. 62, 2002, pp. 517-543.

[47] M. Sheeran, I. Parberry, A New Approach to the Design of Optimal Parallel Prefix Circuits, Department of Computer Science and Engineering, Chalmers University of Technology, Goteborg, Sweden, Technical Report: 2006:1, 2006.

[48] M. Snir, Depth-size trade-offs for parallel prefix computation, *Journal of Algorithms,* Vol. 7, 1986, pp. 185-201.

[49] M. Snir, P. Hochschild, D. D. Frye, and K. J. Gildea, The communication software and parallel environment of the IBM SP2, *IBM Systems Journal,* Vol. 34, No. 2, 1995, pp. 205-221.

[50] Thinking Machines, *Connection Machine Parallel Instruction Set (PARIS)*, Thinking Machines, 1986.

[51] S. Vanichayobon, S. K. Dhall, S. Lakshmivarahan, and J. K. Antonio, Power-speed trade-off in parallel prefix circuits, *Journal of Circuits, Systems, and Computers,* Vol. 14, No. 1, 2005, pp. 65-98.

[52] H. Wang, A. Nicolau, and K. S. Siu, The strict time lower bound and optimal schedules for parallel prefix with resource constraints, *IEEE Transactions on Computers,* Vol. 45, No. 11, 1996, pp. 1257-1271.

[53] Z. Xu, K. Hwang, Modeling communication overhead: MPI and MPL performance on the IBM SP2, *IEEE Parallel & Distributed Technology,* Vol. 4, No. 1, 1996, pp. 9-23.

[54] F. Zhou, P. Kornerup, Computing moments by prefix sums, *Journal of VLSI Signal Processing Systems,* Vol. 25, No. 1, 2000, pp. 5-17.

[55] H. Zhu, C.-K. Cheng, and R. Graham, Constructing zero-deficiency parallel prefix circuits of minimum depth, *ACM Transactions on Design Automation of Electronic Systems,* Vol. 11, No. 2, 2006, pp. 387-409.

[56] R. Zimmermann, Non-heuristic optimization and synthesis of parallel-prefix adders, in *Proceedings of International Workshop on Logic and Architecture Synthesis*, Grenoble, France, 1996, pp. 123-132.

[57] R. Zimmermann, Binary Adder Architectures for Cell-Based VLSI and Their Synthesis, Ph.D. thesis, Swiss Federal Institute of Technology (ETH), Zurich, 1997.