JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 16, 41-64 (2000)

DOI:10.1688/JISE.2000.16.1.3

Efficient Parallel Prefix Algorithms on Multicomputers

YEN-CHUN LIN* AND C. M. LIN
Department of Electronic Engineering
National Taiwan University of Science and Technology
Taipei, Taiwan 106, R.O.C.
**E-mail: yclin@et.ntust.edu.tw*

Given n values $v(0)$, $v(1)$,..., $v(n-1)$ and an associative binary operation, denoted by o, the prefix computation problem, or simply the prefix problem, is to compute the n prefixes $v(0)$ o $v(1)$ o ... o $v(i)$, $0 \le i \le n-1$. We are interested in performing prefix computation on message-passing completely connected multicomputers with the weakest communication capability in as few communication steps as possible. An efficient algorithm is presented to solve the prefix problem on a system of n processors. To explore the possibility of obtaining a faster algorithm, a class of algorithms is then presented. It is shown that the algorithm in this class requiring the fewest communication steps is equivalent to the first algorithm presented.

Keywords: completely connected multicomputers, message-passing, parallel algorithms, prefix computation, send-or-receive

1. INTRODUCTION

Given n values $v(0)$, $v(1)$,..., $v(n-1)$ and an associative binary operation, denoted by o, the prefix computation problem, or simply the prefix problem, is to compute the n prefixes $v(0)$ o $v(1)$ o ... o $v(i)$, $0 \le i \le n-1$. Prefix computation has been extensively studied due to its numerous applications [1-8]; for example, it is used in loop parallelization, the evaluation of polynomials, and the solution of linear recurrences. Because of its importance, prefix computation has been considered to be a primitive operation [9]. Indeed, many parallel prefix algorithms have been proposed [2, 5, 6, 10-12], and many prefix circuits have also been designed and studied [6, 12-18]. This paper presents algorithms that solve the prefix problem on a message-passing completely connected multicomputer. Each processing element (PE) in the multicomputer can only perform a send or receive operation with another PE in a communication step. This send-or-receive model of communication is the most important one of the four communication models presented in [19]. Because it is the weakest communication model on a completely connected multicomputer, the upper bounds of this model apply to other communication models, and the lower bounds of this model are the upper bounds for the lower bounds of other models. Algorithms developed for this model are suited to implementation on some leading-edge message-passing multicomputers, such as the SP2 [20], nCUBE 2 [21], CM-5 [22], and Paragon [23], and to implementation in environments, such as Express [24], Zipcode [25], and PICL [26]. Although a PE of a modern message-passing parallel computer can send a message to a directly connected PE and, in the same step, receive a message from the other directly con-

Received September 5, 1998; revised March 5, 1999; accepted April 9, 1999.
Communicated by Jang-Ping Sheu.

nected PE, it usually takes longer to send and receive in a single step than to send only or receive only due to the inherent hardware capability and software overhead [27-29]. For example, on the SP2, the unidirectional data rate can be as high as 35.5 MB/sec; however, when data are transferred in both directions, the rate in either direction can reach only 24.1 MB/sec [29]. As another example, the unidirectional bandwidth on transputers is 1.8 MB/sec; however, the bandwidth in either direction is only 1.2 MB/sec when data are transferred in both directions [27]. On the other hand, while the SP2 can be programmed as a completely connected computer, the multistage interconnection may cause it to take longer to pass more messages in a single communication step [30]. On an n-PE system, the send-or-receive model ensures that no more than $\lfloor n / 2 \rfloor$ messages are communicated in a single communication step; thus, one communication step will not take too much time.

The completely connected model has some advantages in algorithm development [31]. First, this model is flexible because no specific PE interconnection is assumed; for example, algorithms may be portable between computers that can dynamically allocate PEs and that can tolerate PE faults. In addition, because when designing algorithms for the completely connected model we do not consider tedious details related to message routing, insight can be gained into developing algorithms on computers with a particular interconnection among PEs. Third, if the communication requirement of an algorithm run on the completely connected model can be efficiently mapped to another target model, then that algorithm can be transformed into a new one for the target model.

In this paper, we are interested in prefix computation on send-or-receive completely connected multicomputers in as few communication steps as possible. Section 2 presents an efficient algorithm for solving the prefix problem on a system of n PEs in $\lceil 1.44 \log_2 n \rceil$ or $\lceil 1.44 \log_2 n \rceil + 1$ communication steps. To explore the possibility of obtaining a faster algorithm, section 3 gives a class of algorithms for the prefix problem also on a system of n PEs. The algorithm that requires the fewest communication steps is shown to be equivalent to the one presented in section 2. Section 4 gives related research. Section 5 concludes this paper.

2. AN ALGORITHM

This section presents a parallel algorithm that solves the prefix problem on a system of n PEs. The n PEs are numbered 0 through $n - 1$. Each PE i, $0 \le i < n$, has a memory location, $v(i)$, which stores its initial value and a location, $c(i)$, which keeps its final result as well as temporary values. No other memory locations are needed. To simplify the presentation, $x{:}y$ is used to represent the result of computing $v(x)$ o $v(x+1)$ o ... o $v(y)$, where $x \le y$. PE 0 initially contains the result $v(0)$; for other PEs, we want to obtain $c(i) = 0{:}i$. We will use the Fibonacci sequence defined by $F_0 = 0$, $F_1 = 1$, and $F_{k+2} = F_{k+1} + F_k$ for $k \ge 0$. For ease of presentation, the communication operation "transfer the value of $c(i)$ from PE i to PE j" will be abbreviated to "transfer $c(i)$ to PE j". Further, we will use the operation "$c(j)$:=o $c(i)$" as a shortened form of "$c(j)$:= $c(i)$ o $c(j)$ in PE j", i.e., assigning $c(j)$ in PE j with the result of performing $c(i)$ o $c(j)$. More definitions used throughout this paper for ease of presentation are as follows:

$N = \lfloor n / 2 \rfloor$;
$L_k = F_{k+3} + F_{k+1}$;

$$G(x,y) = \begin{cases} F_y, & \text{if } x = y; \\ 2F_y + \sum_{j=0}^{x-y-2} (F_{y+1} + jF_{y-1}), & \text{if } x > y; \\ 0, & \text{otherwise;} \end{cases}$$

$\phi = (1+\sqrt{5}) / 2$;
$\varphi = (1-\sqrt{5}) / 2$.

Algorithm LL(*n*). /* $n \geq 10$ */

Stage A:

Step 0: Every PE computes $m = \min\{i \mid L_i \geq N\}$ or looks it up from a precomputed table. Note that $L_m \geq N > L_{m-1}$.

Step 1: For all i, $0 \leq i < N$, transfer $v(2i)$ from PE $2i$ to PE $(2i + 1)$, and then compute $c(2i + 1) := v(2i)$ o $v(2i + 1)$ in PE $(2i + 1)$.

Step 2: For all i, $0 \leq i < N - 1$, transfer $c(2i + 1)$ to PE $2i$, and then perform $c(2i) := c(2i + 1)$ in PE $2i$.

Stage B: Step k $(3 \leq k \leq m)$.

Case (a): m is even.

If k is odd, for each i, $0 \leq i < N - F_k$,
(1) transfer $c(2i + 1)$ to PE $2(i + F_{k-2})$,
(2) perform $c(2(i + F_{k-2})) :=$o $c(2i + 1)$.

If k is even, for each i, $0 \leq i < N - F_{k-2}$,
(1) transfer $c(2i)$ to PE $(2(i + F_{k-2}) + 1)$,
(2) perform $c(2(i + F_{k-2}) + 1) :=$o $c(2i)$.

Case (b): m is odd.

If k is odd, for each i, $0 \leq i < N - F_{k-2}$,
(1) transfer $c(2i)$ to PE $(2(i + F_{k-2}) + 1)$,
(2) perform $c(2(i + F_{k-2}) + 1) :=$o $c(2i)$.

If k is even, for each i, $0 \leq i < N - F_k$,
(1) transfer $c(2i + 1)$ to PE $2(i + F_{k-2})$,
(2) perform $c(2(i + F_{k-2})) :=$o $c(2i + 1)$.

Stage C: Step $m + 1$.

For each i, $0 \leq i < F_m$,
(1) transfer $c(2i + 1)$ to PE $(2(i + F_m) + 1)$,
(2) perform $c(2(i + F_m) + 1) :=$o $c(2i + 1)$.

For each i, $0 \leq i < N - 2F_m$,
(1) transfer $c(2(i + F_m))$ to PE $(2(i + 2F_m) + 1)$,
(2) perform $c(2(i + 2F_m) + 1) :=$o $c(2(i + F_m))$.

Stage D: Step $m + 2$.

For each i, $0 \leq i < F_{m+1}$,
(1) transfer $c(2(i + F_{m-2}) + 1)$ to PE $(2(i + 2F_m) + 1)$,

(2) perform $c(2(i + 2F_m) + 1) :=o\ c(2(i + F_{m-2}) + 1)$.

For each i, $0 \le i < N - 2Fm - F_{m+1}$,

(1) transfer $c(2(i + 2F_m))$ to PE $(2(i + 2F_m + F_{m+1}) + 1)$,

(2) perform $c(2(i + 2F_m + F_{m+1}) + 1) :=o\ c(2(i + 2F_m))$.

Stage E: Step $m + 3$.

For each i, $0 \le i < N - 2F_m - F_{m+1}$,

(1) transfer $c(2(i + 2F_m - F_{m-1}) + 1)$ to PE $(2(i + 2F_m + F_{m+1}) + 1)$,

(2) perform $c(2(i + 2F_m + F_{m+1}) + 1) :=o\ c(2(i + 2F_m - F_{m-1}) + 1)$.

Stage F: Step $m + 4$.

If n is even (respectively, odd) for each i, $0 \le i < N - 1$ (respectively, $0 \le i < N$), transfer $c(2i + 1)$ to PE $(2i + 2)$, and then compute $c(2i + 2) := c(2i + 1)\ o\ v(2i + 2)$ in PE $(2i + 2)$.

As an example, Fig. 1 helps explain how and when the final results are generated for $n = 16$ and, therefore, $N = 8$ and $m = 3$. Every vertical line represents a PE whose identification number is above the line. An oblique line drawn between PE i and PE j with a dot at the lower end on PE j represents a data transfer from PE i to PE j. The symbol x:y shown on the right side of the dot denotes a final or intermediate result obtained in the same step as the data transfer. For example, PE 3 produces 2:3 and 0:3 in steps 1 and 3, respectively.

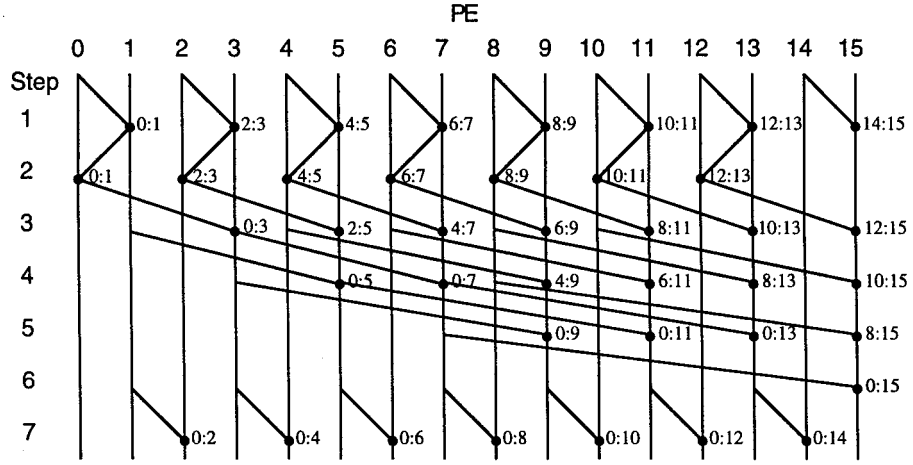

Fig. 1. Execution of LL(16).

Let us briefly point out some characteristics of the above algorithm to gain more insight. After stage A, $c(2i) = c(2i + 1) = 2i$:$(2i + 1)$ for $0 \le i \le N - 2$, and $c(2N - 1) = (2N - 2)$:$(2N - 1)$. During stage B, the algorithm executes $m - 2$ steps. If $m \ge 3$ and m is even (respectively, odd), then in step 3, $c(i)$ values in the first $N - F_3$ odd-numbered (respectively, the first $N - F_1$ even-numbered) PEs are transferred to even-numbered (respectively, odd-numbered) PEs. Then, in step 4, $c(i)$ values in the first $N - F_2$ even-numbered (respectively, the first $N - F_4$ odd-numbered) PEs are transferred to odd-numbered (respectively, even-numbered) PEs, and so on, until step m.

In steps $m + 1$ and $m + 2$, while a portion of the even-numbered PEs transfer their values to odd-numbered PEs, respectively, a portion of the odd-numbered PEs also transfer their $c(i)$ values to other odd-numbered PEs, respectively. In step $m + 3$, only odd-numbered PEs transfer their values to other odd-numbered PEs, respectively. After step $m + 3$, all odd-numbered PEs obtain their respective final results. That is, $c(2i + 1) = 0{:}(2i + 1)$ for $0 \le i < N$. Then, in the last step, each $c(2i + 1)$, where $0 \le i < N$ and $2i + 1 < n - 1$, is transferred to PE $(2i + 2)$ so that the receiver PE can compute the final result $0{:}(2i + 2)$.

Lemma 1. After executing stage E of Algorithm LL, $c(2i + 1) = 0{:}(2i + 1)$ for $0 \le i < N$; that is, odd-numbered PEs contain their respective final results.

Proof: Because the proof is very lengthy, it is given in Appendix A.

Theorem 1. Algorithm LL solves the prefix problem in $\lceil \log_\phi n \rceil = \lceil 1.44 \log_2 n \rceil$ or $\lceil \log_\phi n \rceil + 1$ communication steps.

Proof: From Lemma 1, after step $m + 3$, all odd-numbered PEs obtain their respective final results. Then, after step $m + 4$, all even-numbered PEs also contain their respective final results. That is, only $m + 4$ communication steps are needed to compute all the prefixes.

Let $q = m + 2$; then, $L_m = l_{m+2} = l_q$, where $l_q = F_{q+1} + F_{q-1}$ is the Lucas number [32]. Using the closed form $l_q = \phi^q + \varphi^q$ and the condition $L_m \ge N > L_{m-1}$, we have

$$\phi^q + \varphi^q \ge \lfloor n / 2 \rfloor > \phi^{q-1} + \varphi^{q-1}.$$

Thus,

$$\begin{aligned}
&\phi^q + \varphi^q \ge (n-1)/2 \text{ and } n/2 > \phi^{q-1} + \varphi^{q-1},\\
&\phi^q \ge (n - 2\varphi^q - 1)/2 \text{ and } (n - 2\varphi^{q-1})/2 > \phi^{q-1},\\
&q \ge \log_\phi((n - 2\varphi^q - 1)/2) \text{ and } \log_\phi((n - 2\varphi^{q-1})/2) > q - 1,\\
&q + 2 \ge \log_\phi((n - 2\varphi^q - 1)/2) + 2 \text{ and } \log_\phi((n - 2\varphi^{q-1})/2) + 3 > q + 2,\\
&\log_\phi((n - 2\varphi^q - 1)/(2n)) + \log_\phi n + 2 \le q + 2 < \log_\phi((n - 2\varphi^{q-1})/(2n)) + \log_\phi n + 3.
\end{aligned}$$

Because $n \ge 10$, or $N \ge 5$, and $L_m \ge N > L_{m-1}$, we have $m \ge 2$; thus, $q \ge 4$. Therefore, $|\varphi^q| < 0.2$ and $|\varphi^{q-1}| < 0.3$. In addition, it is assumed that $n \ge 10$; therefore,

$$\log_\phi(0.5 - (0.4 + 1)/20) + \log_\phi n + 2 \le q + 2 < \log_\phi(0.5 + 0.3/10) + \log_\phi n + 3.$$

Because

$$\log_\phi(0.5 - (0.4 + 1)/20) > -2 \text{ and } \log_\phi(0.5 + 0.3/10) < -1,$$

we thus have

$$\begin{aligned}
&\log_\phi n < q + 2 < \log_\phi n + 2,\\
&\lceil \log_\phi n \rceil \le q + 2 \le \lceil \log_\phi n \rceil + 1,\\
&\lceil \log_\phi n \rceil \le m + 4 \le \lceil \log_\phi n \rceil + 1.
\end{aligned}$$

Q.E.D.

3. A CLASS OF ALGORITHMS

Recall that in Algorithm LL, odd-numbered PEs start to send values to other odd-numbered PEs in step $m + 1$, where $m = \min\{i \mid L_i \geq N\}$. A question arises naturally: Can we change the value of m and still have an algorithm for prefix computation? If we can, is there a better algorithm than LL in the sense that it takes fewer communication steps? This section presents a class of parallel prefix algorithms in which the value of m can be any integer greater than 1. However, we shall show that no algorithms are better than LL.

Algorithm CL(n, m). /* $n \geq 10$ and $m \geq 2$ */

Stage A:

Step 0: Every PE computes $t = \min\{i \mid G(i, m) \geq N\}$ or looks it up from a precomputed table. Note that $G(t, m) \geq N > G(t - 1, m)$.

Step 1: For all i, $0 \leq i < N$, transfer $v(2i)$ from PE $2i$ to PE $(2i + 1)$, and then compute $c(2i + 1) := v(2i)$ o $v(2i + 1)$ in PE $(2i + 1)$.

Step 2: For all i, $0 \leq i < N - 1$, transfer $c(2i + 1)$ to PE $2i$, and then perform $c(2i) := c(2i + 1)$ in PE $2i$.

Stage B: Step k $(3 \leq k \leq m)$.

Case (a): m is even.

If k is odd, for each i, $0 \leq i < N - F_k$,
(1) transfer $c(2i + 1)$ to PE $2(i + F_{k-2})$,
(2) perform $c(2(i + F_{k-2})) :=$o $c(2i + 1)$.

If k is even, for each i, $0 \leq i < N - F_{k-2}$,
(1) transfer $c(2i)$ to PE $(2(i + F_{k-2}) + 1)$,
(2) perform $c(2(i + F_{k-2}) + 1) :=$o $c(2i)$,
(3) if $F_k \geq N$, go to stage F.

Case (b): m is odd.

If k is odd, for each i, $0 \leq i < N - F_{k-2}$,
(1) transfer $c(2i)$ to PE $(2(i + F_{k-2}) + 1)$,
(2) perform $c(2(\mathrm{i} + F_{k-2}) + 1) :=$o $c(2i)$,
(3) if $F_k \geq N$, go to stage F.

If k is even, for each i, $0 \leq i < N - F_k$,
(1) transfer $c(2i + 1)$ to PE $2(i + F_{k-2})$,
(2) perform $c(2(i + F_{k-2})) :=$o $c(2i + 1)$.

Stage C: Step $m + 1$.

For each i, $0 \leq i < F_m$,
(1) transfer $c(2i + 1)$ to PE $(2(i + F_m) + 1)$,
(2) perform $c(2(i + F_m) + 1) :=$o $c(2i + 1)$.

For each i, $0 \leq i < N - 2F_m$,
(1) transfer $c(2(i + F_m))$ to PE $(2(i + 2F_m) + 1)$,
(2) perform $c(2(i + 2F_m) + 1) :=$o $c(2(i + F_m))$.

Stage D: Step k ($m + 2 \le k \le t - 1$).
For each i, $0 \le i < F_{m+1} + (k - m - 2)F_{m-1}$,
(1) transfer $c(2(i + G(k - 1, m) - F_{m+1} - (k - m - 2)F_{m-1}) + 1)$ to PE $(2(i + G(k - 1, m)) + 1)$,
(2) perform $c(2(i + G(k - 1, m)) + 1)$ $:=$o $c(2(i + G(k - 1, m) - F_{m+1} - (k - m - 2)F_{m-1}) + 1)$.
For each i, $0 \le i < N - G(k, m)$,
(1) transfer $c(2(i + G(k - 1, m)))$ to PE $(2(i + G(k, m)) + 1)$,
(2) perform $c(2(i + G(k, m)) + 1)$ $:=$o $c(2(i + G(k - 1, m)))$.

Stage E: If $t < m + 2$, go to stage F; otherwise,
for each i, $0 \le i < N - G(t - 1, m)$,
(1) transfer $c(2(i + G(t - 1, m) - F_{m+1} - (t - m - 2)F_{m-1}) + 1)$ to PE $(2(i + G(t - 1, m)) + 1)$,
(2) perform $c(2(i + G(t - 1, m)) + 1)$ $:=$o $c(2(i + G(t - 1, m) - F_{m+1} - (t - m - 2)F_{m-1}) + 1)$.

Stage F: If n is even (respectively, odd), for each i, $0 \le i < N - 1$ (respectively, $0 \le i < N$), transfer $c(2i + 1)$ to PE $(2i + 2)$, and then compute $c(2i + 2) := c(2i + 1)$ o $v(2i + 2)$ in PE $(2i + 2)$.

Note that Algorithm CL shares the following with LL: steps 1 and 2, stages C and F. In addition, if CL does not jump from stage B to stage F, then CL and LL also pass identical messages between the same pairs of PEs and perform identical o operations and assignments in the same PEs in stage B.

Let us examine the possible executions of CL when $n \ge 10$ and $m \ge 2$. When $m = 2$, except for stage B, all stages will be executed. When $m \ge 3$, stage B will be executed, but other stages may be skipped, as presented in the following. If $F_m \ge N$, the execution will jump from stage B to stage F, skipping stages C, D, and E. If $2F_m \ge N > F_m$ or $G(m + 1, m) \ge N > G(m, m)$, then $t = m + 1$; thus, stages D and E will be skipped. If $L_{m-1} \ge N > 2F_m$ or $2F_m + F_{m+1} = G(m + 2, m) \ge N > G(m + 1, m)$, then $t = m + 2$; thus, only stage D will be skipped. If $N > L_{m-1}$, then $t \ge m + 3$; thus, no stage will be skipped. Note that when CL uses the same m value as LL does, satisfying $L_m \ge N > L_{m-1}$, CL is equivalent to LL in the sense that they pass identical messages between the same pairs of PEs and perform identical operations and assignments in the same PEs in each step except step 0.

Lemma 2. Before executing stage F of Algorithm CL, we have $c(2i + 1) = 0{:}(2i + 1)$ for $0 \le i < N$; that is, odd-numbered PEs contain their respective final results.

Proof: Because the proof is very lengthy, it is given in Appendix B.

Theorem 2. Consider solving the prefix problem with Algorithm CL. If $N > F_m$, it takes $t + 1$ communication steps, where $G(t, m) \ge N > G(t - 1, m)$. If $F_m \ge N$, it takes $j + 1$ communication steps, where $F_j \ge N > F_{j-1}$, and j is even if and only if m is even.

Proof: From Lemma 2 and Algorithm CL, all even-numbered PEs obtain their final results one step later than all odd-numbered PEs. If $N > 2F_m$, then, as already noted, $t \geq m + 2$, and from Lemma 2, whether stage D is skipped or not, all odd-numbered PEs obtain their respective final results after stage E, or step t. Thus, it takes $t + 1$ communication steps to solve the prefix problem. If $2F_m \geq N > F_m$, after step $m + 1 = t$, or stage C, and before stage F, all odd-numbered PEs contain their respective final results; therefore, it also takes $t + 1$ communication steps to solve the prefix problem.

If $F_m \geq N$ and m is even (respectively, odd), it takes j communication steps to execute stages A and B for all odd-numbered PEs to obtain their respective final results, where j is also even (respectively, odd) and $F_j \geq N > F_{j-2}$. This implies that when $F_j \geq N > F_{j-1}$, it also takes j communication steps for all odd-numbered PEs to obtain their respective final results. Therefore, it takes $j + 1$ communication steps to solve the prefix problem. Q.E.D.

To better understand the effect of the relationship between n and m on the number of communication steps needed to solve the prefix problem, Table 1 lists the ranges of n in which Algorithm CL can manage to solve the prefix problem in 4 through 10 communication steps for three possible relationships of N and m. For example, when $F_m \geq N > F_{m-1}$ and $12 \leq n \leq 17$, it takes 7 communication steps. Note that although we do not expect to solve the prefix problem with Algorithm CL when $n < 10$, CL is actually applicable when $4 \leq n \leq 9$ and $2F_m \geq N > F_{m-1}$. Table 1 shows that, for a fixed value of $n \geq 10$, Algorithm CL with $L_m \geq N > L_{m-1}$ takes no more communication steps than it does with the other two relationships. In fact, we shall now prove that CL takes the fewest steps when $L_m \geq N > L_{m-1}$.

Table 1. Ranges of *n* that Algorithm CL can manage in solving the prefix problem in 4 to 10 communication steps for three different relationships of *n* and *m*.

Number of steps	$F_m \geq N > F_{m-1}$	$2F_m \geq N > F_m$	$L_m \geq N > L_{m-1}$
4	4-5	4-5	N/A
5	6-7	6-9	N/A
6	8-11	10-13	10-15
7	12-17	14-21	16-23
8	18-27	22-33	24-37
9	28-43	34-53	38-59
10	44-69	54-85	60-95

Theorem 3. Algorithm CL takes the fewest steps when $L_m \geq N > L_{m-1}$.

Proof: Suppose $L_{m_0} \geq N > L_{m_0-1}$ and $m_0 \geq 2$; thus,

$$2F_{m_0} + 2F_{m_0+1} + F_{m_0-1} \geq N > 2F_{m_0-1} + 2F_{m_0} + F_{m_0-2},$$

$$2F_{m_0} + \sum_{j=0}^{1}(F_{m_0+1} + jF_{m_0-1}) \geq N > 2F_{m_0} + \sum_{j=0}^{0}(F_{m_0+1} + jF_{m_0-1}),$$

$$G(m_0 + 3,\ m_0) \geq N > G(m_0 + 2,\ m_0).$$

Let $t_0 = m_0 + 3$. From Theorem 2, CL(n, m_0) takes $t_0 + 1$ communication steps. We shall consider three cases.

(i) If $N > F_{m1}$, $m_0 > m_1 \geq 2$, and $G(t_1, m_1) \geq N > G(t_1 - 1, m_1)$, then from Theorem 2, CL(n, m_1) takes $t_1 + 1$ communication steps. We need to show that $t_0 \leq t_1$.

From $G(t_1, m_1) \geq N$ and $N > G(m_0 + 2, m_0) = G(t_0 - 1, m_0)$, we have $G(t_1, m_1) > G(t_0 - 1, m_0)$.

Let $m_0 = m_1 + i$, where $i \geq 1$; then,

$$G(t_1, m_0 - i) > G(t_0 - 1, m_0);$$

that is,

$$2F_{m_0-i} + \sum_{j=0}^{t_1-m_0+i-2}(F_{m_0-i+1} + jF_{m_0-i-1}) > 2F_{m_0} + F_{m_0+1}.$$

We shall now show that the above inequality does not hold when $t_1 \leq m_1 + i + 2$, thereby, obtaining $t_1 \geq m_1 + i + 3 = m_0 + 3 = t_0$. We will prove this by induction on i. The claim can be checked for $i = 1$. Assume that the claim holds for $i = k$. Thus, when $t_1 = m_1 + k + 2$,

$$2F_{m_0-k} + \sum_{j=0}^{k}(F_{m_0-k+1} + jF_{m_0-k-1}) \leq 2F_{m_0} + F_{m_0+1}.$$

When $i = k + 1$, $m_0 = m_1 + k + 1$ and $t_1 \leq m_1 + k + 3$, and we have

$$\begin{aligned}
&2F_{m_0-k-1} + \sum_{j=0}^{t_1-m_0+k-1}(F_{m_0-k} + jF_{m_0-k-2}) \\
&\leq 2F_{m_0-k-1} + \sum_{j=0}^{k+1}(F_{m_0-k} + jF_{m_0-k-2}) \\
&= 2F_{m_0-k-1} + (k+2)F_{m_0-k} + \sum_{j=0}^{k+1} jF_{m_0-k-2} \\
&= 2F_{m_0-k} + kF_{m_0-k} + 2F_{m_0-k-1} + \sum_{j=0}^{k+1} jF_{m_0-k-2} \\
&= 2F_{m_0-k} + (k+1)F_{m_0-k+1} - F_{m_0-k} - (k-1)F_{m_0-k-1} + \sum_{j=0}^{k+1} jF_{m_0-k-2} \\
&= 2F_{m_0-k} + (k+1)F_{m_0-k+1} - (k+1)F_{m_0-k-2} - kF_{m_0-k-3} + \sum_{j=0}^{k+1} jF_{m_0-k-2} \\
&= 2F_{m_0-k} + (k+1)F_{m_0-k+1} - kF_{m_0-k-3} + \sum_{j=0}^{k} jF_{m_0-k-2} \\
&< 2F_{m_0-k} + (k+1)F_{m_0-k+1} + \sum_{j=0}^{k} jF_{m_0-k-1} \\
&= 2F_{m_0-k} + \sum_{j=0}^{k}(F_{m_0-k+1} + jF_{m_0-k-1}).
\end{aligned}$$

Consequently,

$$2F_{m_0-k-1} + \sum_{j=0}^{t_1-m_0+k-1}(F_{m_0-k} + jF_{m_0-k-2}) < 2F_{m_0} + F_{m_0+1}.$$

That is, the claim holds when $i = k + 1$. Therefore, we have proved that $t_1 \geq t_0$.

(ii) If $N > F_{m2}$, $m_2 > m_0$, and $G(t_2, m_2) \geq N > G(t_2 - 1, m_2)$, then from Theorem 2, CL(n, m_2) takes $t_2 + 1$ communication steps. We need to show that $t_2 \geq t_0$.

From $N > F_{m_2}$ and $G(t_2, m_2) \geq N$, we have $G(t_2, m_2) > F_{m_2}$. If $t_2 = m_2$, then $G(t_2, m_2) = F_{m_2}$. Thus, $t_2 \geq m_2 + 1$ must hold for $G(t_2, m_2) > F_{m_2}$ to hold.

If $t_2 = m_2 + 1$, from $G(t_2, m_2) \geq N$ and $N > G(t_0 - 1, m_0)$, we have

$$G(t_2, m_2) > G(t_0 - 1, m_0),$$
$$2F_{m_2} > 2F_{m_0} + F_{m_0+1},$$
$$2F_{m_2} > F_{m_0} + F_{m_0+2} \geq 2F_{m_0+1};$$

thus,

$$m_2 > m_0 + 1,$$
$$m_2 + 1 \geq m_0 + 3,$$
$$t_2 \geq t_0.$$

If $t_2 = m_2 + i$, $i \geq 2$, then from the assumption that $m_2 > m_0$, we get $t_2 \geq m_0 + i + 1 \geq t_0$.

(iii) If $F_{m3} \geq N$, from Theorem 2, CL(n, m_3) takes $j + 1$ communication steps when $F_j \geq N > F_{j-1}$, and j is even if and only if m is even. We need to show that $j \geq t_0$. Because

$$F_j \geq N \text{ and } N > L_{m_0-1},$$

we have

$$F_j > F_{m_0+2} + F_{m_0} > F_{m_0+2},$$
$$j > m_0 + 2,$$
$$j \geq m_0 + 3 = t_0.$$ Q.E.D.

Corollary 1. Algorithm LL takes no more communication steps than does Algorithm CL.

4. RELATED RESEARCH

Parallel prefix algorithms on a message-passing completely connected multicomputer may be derived from parallel prefix circuits and from prefix algorithms on the parallel random-access machine (PRAM) model. A parallel prefix circuit with fan-out 2 denoted by $L(n)$ has been presented, and it can be mapped to a prefix algorithm requiring $2\lfloor \log_2 n \rfloor - 1$ or $2\lfloor \log_2 n \rfloor$ communication steps on a send-or-receive completely connected multicomputer [15, 33]. $L(n)$ is depth-size optimal, meaning that its depth plus its size equals the lower bound $2n - 2$, and the corresponding algorithm of $L(n)$ is optimal in the sense that the number of communication steps plus the total number of messages equals $2n - 2$.

The lower bound of the time complexity of prefix computation for all PRAMs is $\Omega(\log n)$. A well-known time-optimal prefix algorithm on a concurrent-read exclusive-write (CREW) PRAM with $n/2$ PEs [1, 34] can be mapped to a prefix algorithm on a send-or-receive completely connected multicomputer with n PEs. The latter algorithm requires $2\lceil \log_2 n \rceil$ communication steps. Clearly, Algorithm LL takes fewer communication steps than do the other two algorithms on the same model.

The prefix sums problem is a special case of the prefix problem: the associative binary operation is the arithmetic addition. The fast algorithm used to solve the prefix sums problem on a concurrent-read concurrent-write (CRCW) PRAM as presented in [2] does not work for the more general prefix problem. When the n initial values are O(log n)-bit numbers, the algorithm takes O(log n / log log n) time using n log log n/log n PEs.

On an exclusive-read exclusive-write (EREW) PRAM, Ladner and Fischer solved the prefix problem in $\Theta(\log n)$ time using $\Theta(n/\log n)$ PEs [12]. On a concurrent-read exclusive-write (CREW) PRAM with $p < n$ PEs, prefix computation takes $\Theta(n/p + \log p)$ time, which is time-optimal and cost-optimal when $p = \Theta(n/\log n)$ [34].

On the k-port ($k \geq 1$) communication model of a completely connected message-passing machine, each PE can send k distinct messages to k PEs and receive k messages from k PEs in one communication step. On such a machine with n PEs, it needs at least $\lceil \log_{k+1} n \rceil$ communication steps to solve the prefix problem [35].

The gossip operation is used to send a value in each PE to all the other PEs. On the k-port model, it takes at least $\lceil \log_{k+1} n \rceil$ communication steps to perform the gossip operation [31]. Thus, the prefix operation and gossip operation have the same lower bound for communication complexity. In addition, the gossip operation needs at least $\lceil \log_\phi n \rceil \approx \lceil 1.44 \log_2 n \rceil$ communication steps on the send-or-receive model [19]. Therefore, we conjecture that $\lceil \log_\phi n \rceil$ is also a communication lower bound for the prefix problem. Entringer and Slater have presented an algorithm for the gossip operation on an n-PE send-or-receive machine in $\lceil \log_\phi n \rceil + 4$ communication steps [19, 36].

5. CONCLUSIONS

We are interested in solving the prefix problem in as few communication steps as possible on message-passing completely connected multicomputers with only send-or-receive communication requirement. The lower bound on the communication complexity for the prefix problem on a system of n PEs is conjectured to be $\lceil \log_\phi n \rceil$ communication steps. Our Algorithm LL solves the prefix problem on a system of n PEs in $\lceil \log_\phi n \rceil$ or $\lceil \log_\phi n \rceil + 1$ communication steps. To find a faster algorithm, we have designed a class of algorithms CL. The algorithm that requires the fewest communication steps in CL is equivalent to LL. This not only implies that there may not be any algorithm requiring fewer communication steps, but also increases our confidence that at least $\lceil \log_\phi n \rceil$ communication steps are required.

As a final remark, we note that LL can be modified to solve the prefix problem by using only p PEs, where $10 \leq p < n$. For simplicity, assume that $q = n/p$ is an integer, and that PE i initially contains $v(iq), v(iq + 1),..., v((i + 1)q - 1)$, for that $0 \leq i < p$. We can obtain $v(j) = 0{:}j$, for $0 \leq j < n$. The reader is referred to [10] for a description of the modified algorithm. The algorithm requires $\Theta(n/p + \log p)$ computation time and $\Theta(\log p)$ communication time. If $p = \Omega(n/\log n)$, then $n/p = O(\log n)$. In addition, $\log p = O(\log n)$; thus, the execution time is $\Theta(n/p + \log p) = O(\log n)$, which is optimal. On the other hand, if $p = O(n/\log n)$, then $n/p = \Omega(\log n) = \Omega(\log p)$; thus, the time complexity $\Theta(n/p + \log p) = \Theta(n/p)$, and the algorithm achieves linear speedup and is cost-optimal. Therefore, when $p = \Theta(n/\log n)$, the algorithm

is time-optimal and cost-optimal. It should be noted that when $n = \Omega(p \log p)$, we have $n/p = \Omega(\log p)$, $\Theta(n/p + \log p) = \Theta(n/p)$; thus, the algorithm also achieves linear speedup and is cost-optimal.

APPENDIX A

Proof of Lemma 1: Because $n \geq 10$, or $N \geq 5$, and $L_m \geq N > L_{m-1}$, we have $m \geq 2$. Note that stage B is executed for at least one step only when $m \geq 3$. Therefore, we must consider both the situation where stage B is executed and that where stage B is skipped.

(i) First, we will show that during case (*a*) of stage B (*m* is even), after step *k*, if *k* is odd, then

$$c(2i) = 0{:}(2i + 1), \quad \text{for } 0 \leq i < F_k;$$
$$c(2i) = 2(i - F_k + 1){:}(2i + 1) \quad \text{for } F_k \leq i < N - F_{k-1}.$$

Otherwise, if *k* is even,

$$c(2i + 1) = 0{:}(2i + 1), \quad \text{for } 0 \leq i < F_k;$$
$$c(2i + 1) = 2{:}(i - F_k + 1){:}(2i + 1), \quad \text{for } F_k \leq i < N.$$

We shall prove by induction on *k*.

The claim can be checked for $k = 3, 4$. Specifically, after step $k = 3$,

$$c(2i) = 0{:}(2i + 1), \quad \text{for } 0 \leq i < 2;$$
$$c(2i) = (2i - 2){:}(2i + 1), \quad \text{for } 2 \leq i < N - 1.$$

After step $k = 4$,

$$c(2i + 1) = 0{:}(2i + 1), \quad \text{for } 0 \leq i < 3;$$
$$c(2i + 1) = 2(i - 2){:}(2i + 1), \quad \text{for } 3 \leq i < N.$$

Assume the claim holds for $k \leq s < m$; we can show that it also holds for $k = s + 1$.

Case (1): *s* is even.

By the induction hypothesis, after step $k = s - 1$, we have

$$c(2i) = 0{:}(2i + 1), \quad \text{for } 0 \leq i < F_{s-1}; \tag{1}$$
$$c(2i) = 2(i - F_{s-1} + 1){:}(2i + 1), \quad \text{for } F_{s-1} \leq i < N - F_{s-2}. \tag{2}$$

In addition, after step $k = s$, we have

$$c(2i + 1) = 0{:}(2i + 1), \quad \text{for } 0 \leq i < F_s; \tag{3}$$
$$c(2i + 1) = 2(i - F_s + 1){:}(2i + 1), \quad \text{for } F_s \leq i < N. \tag{4}$$

In fact, Eq. (2) is equivalent to

$$c(2(i + F_{s-1})) = 2(i + 1){:}2(i + F_{s-1}) + 1, \quad \text{for } 0 \leq i < N - F_s. \tag{5}$$

In step $s + 1$, the first $N - F_{s+1}$ values in Eqs. (3) and (4) are involved in operations; thus, instead of using Eqs. (3) and (4), we will use Eq. (3) and

$$c(2i + 1) = 2(i - F_s + 1):(2i + 1), \qquad \text{for } F_s \le i < N - F_{s+1}. \tag{6}$$

Therefore, we use Eqs. (3), (6), and (5) in performing

$$c(2(i + F_{s-1})) :=\text{o } c(2i + 1), \qquad \text{for } 0 \le i < N - F_{s+1},$$

to obtain

$$c(2(i + F_{s-1})) = 0:(2(i + F_{s-1}) + 1), \qquad \text{for } 0 \le i < F_s;$$
$$c(2(i + F_{s-1})) = 2(i - F_s + 1):(2(i + F_{s-1}) + 1), \qquad \text{for } F_s \le i < N - F_{s+1}.$$

The above two equations are, respectively, equivalent to

$$c(2i) = 0:(2i + 1), \qquad \text{for } F_{s-1} \le i < F_{s+1}; \tag{7}$$
$$c(2i) = 2(i - F_{s+1} + 1):(2i + 1), \qquad \text{for } F_{s+1} \le i < N - F_s. \tag{8}$$

Combining Eqs. (1) and (7), we get

$$c(2i) = 0:(2i + 1), \qquad \text{for } 0 \le i < F_{s+1}. \tag{9}$$

Thus, by Eqs. (9) and (8), the claim holds for $k = s + 1$.

Case (2): s is odd.

By the induction hypothesis, after step $k = s - 1$, we have

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 0 \le i < F_{s-1}; \tag{10}$$
$$c(2i + 1) = 2(i - F_{s-1} + 1):(2i + 1), \qquad \text{for } F_{s-1} \le i < N. \tag{11}$$

In addition, after step $k = s$, we have

$$c(2i) = 0:(2i + 1), \qquad \text{for } 0 \le i < F_s; \tag{12}$$
$$c(2i) = 2(i - F_s + 1):(2i + 1), \qquad \text{for } F_s \le i < N - F_{s-1}. \tag{13}$$

Eq. (11) is equivalent to

$$c(2(i + F_{s-1}) + 1) = 2(i + 1):2(i + F_{s-1}) + 1, \qquad \text{for } 0 \le i < N - F_{s-1}. \tag{14}$$

In step $s + 1$, the $N - F_{s-1}$ values in Eqs. (12) and (13) are involved in operations; thus, we use Eqs. (12), (13), and (14) in performing

$$c(2(i + F_{s-1}) + 1) := \text{o } c(2i), \qquad \text{for } 0 \le i < N - F_{s-1},$$

to obtain

$$c(2(i + F_{s-1}) + 1) = 0:2(i + F_{s-1}) + 1, \qquad \text{for } 0 \le i < F_s;$$

$$c(2(i + F_{s-1}) + 1) = 2(i - F_s + 1):2(i + F_{s-1}) + 1, \qquad \text{for } F_s \le i < N - F_{s-1}.$$

The above two equations are, respectively, equivalent to

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } F_{s-1} \le i < F_{s+1}; \tag{15}$$
$$c(2i + 1) = 2(i - F_{s+1} + 1):(2i + 1), \qquad \text{for } F_{s+1} \le i < N. \tag{16}$$

Combining Eqs. (10) and (15), we get

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 0 \le i < F_{s+1}. \tag{17}$$

Thus, by Eqs. (17) and (16), the claim holds for $k = s + 1$.

(ii) Next, we will prove that for case (*b*) of stage B (*m* is odd), after step *k*, if *k* is even, then

$$c(2i) = 0:(2i + 1), \qquad \text{for } 0 \le i < F_k;$$
$$c(2i) = 2(i - F_k + 1):(2i + 1), \qquad \text{for } F_k \le i < N - F_{k-1}.$$

Otherwise, if *k* is odd,

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 0 \le i < F_k;$$
$$c(2i + 1) = 2(i - F_k + 1):(2i + 1), \qquad \text{for } F_k \le i < N.$$

If $m = 3$, the claim can be easily checked for $k = 3$. If $m \ge 5$, the proof is carried out by induction on *k* and is almost the same as that for (i) above. The major difference is in the details of the basis part or in checking for $k = 3, 4$. Specifically, after step $k = 3$,

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 0 \le i < 2;$$
$$c(2i + 1) = (2i - 2):(2i + 1), \qquad \text{for } 2 \le i < N.$$

After step $k = 4$,

$$c(2i) = 0:(2i + 1), \qquad \text{for } 0 \le i < 3;$$
$$c(2i) = 2(i - 2):(2i + 1), \qquad \text{for } 3 \le i < N - 2.$$

The induction part is exactly the same as that in (i) above, except that the two words "odd" and "even" should be interchanged. Specifically, now, case (1) is for odd *s*, and case (2) is for even *s*.

(iii) Then, we can show that after stage C, or after step $m + 1$,

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 0 \le i < 2F_m; \tag{18}$$
$$c(2i + 1) = 2(i - F_{m+1} + 1):(2i + 1), \qquad \text{for } 2F_m \le i < N. \tag{19}$$

If $m = 2$, that is, stage B is not executed, then after stage A and before stage C, we have

$$c(2i) = 0:(2i + 1), \qquad \text{for } 0 \le i < F_{m-1}; \tag{20}$$

$$c(2i) = 2(i - F_{m-1} + 1):(2i + 1), \qquad \text{for } F_{m-1} \le i < N - F_{m-2} \tag{21}$$

and

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 0 \le i < F_m; \tag{22}$$
$$c(2i + 1) = (2(i - F_m + 1)):(2i + 1), \qquad \text{for } F_m \le i < N. \tag{23}$$

If $m \ge 3$, that is, stage B is executed, from the above proof, after step $m - 1$, we also have Eqs. (20) and (21) whether m is even or odd. In addition, after step m, we have Eqs. (22) and (23) whether m is even or odd. Eq. (23) is equivalent to

$$c(2(i + F_m) + 1) = 2(i + 1):(2(i + F_m) + 1), \qquad \text{for } 0 \le i < N - F_m.$$

It can be partitioned into two portions:

$$c(2(i + F_m) + 1) = 2(i + 1):(2(i + F_m) + 1), \qquad \text{for } 0 \le i < F_m; \tag{24}$$
$$c(2(i + F_m) + 1) = 2(i + 1):(2(i + F_m) + 1), \qquad \text{for } F_m \le i < N - F_m. \tag{25}$$

Using Eqs. (22) and (24) in performing

$$c(2(i + F_m) + 1) :=\text{o } c(2i + 1), \qquad \text{for } 0 \le i < F_m,$$

in stage C, we have

$$c(2(i + F_m) + 1) = 0:(2(i + F_m) + 1), \qquad \text{for } 0 \le i < F_m.$$

The above equation is equivalent to

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } F_m \le i < 2F_m.$$

Combining the above equation and Eq. (22), we have Eq. (18).

The $N - 2F_m$ values in Eq. (21),

$$c(2i) = 2(i - F_{m-1} + 1):(2i + 1), \qquad \text{for } F_m \le i < N - F_m,$$

can be expressed as

$$c(2(i + F_m)) = 2(i + F_{m-2} + 1):(2(i + F_m) + 1), \qquad \text{for } 0 \le i < N - 2F_m. \tag{26}$$

Also note that Eq. (25) is equivalent to

$$c(2(i + 2F_m) + 1) = 2(i + F_m + 1):(2(i + 2F_m) + 1), \quad \text{for } 0 \le i < N - 2F_m. \tag{27}$$

Therefore, we use Eqs. (26) and (27) in performing

$$c(2(i + 2F_m) + 1) :=\text{o } c(2(i + F_m)), \qquad \text{for } 0 \le i < N - 2F_m,$$

in stage C to obtain

$$c(2(i + 2F_m) + 1) = 2(i + F_{m-2} + 1):(2(i + 2F_m) + 1), \text{ for } 0 \le i < N - 2F_m. \quad (28)$$

The above equation is equivalent to Eq. (19). Therefore, the claim holds.

(iv) Now, we will prove that after stage D, or after step $m + 2$,

$$c(2i + 1) = 0:(2i + 1), \quad \text{for } 0 \le i < 2F_m + F_{m+1}; \quad (29)$$
$$c(2i + 1) = 2(i - F_{m+1} - F_{m-1} + 1):(2i + 1), \quad \text{for } 2F_m + F_{m+1} \le i < N. \quad (30)$$

Eq. (28) can be partitioned into two portions:

$$c(2(i + 2F_m) + 1) = 2(i + F_{m-2} + 1):(2(i + 2F_m) + 1), \quad \text{for } 0 \le i < F_{m+1}; \quad (31)$$
$$c(2(i + 2F_m) + 1) = 2(i + F_{m-2} + 1):(2(i + 2F_m) + 1), \quad \text{for } F_{m+1} \le i < N - 2F_m. \quad (32)$$

The F_{m+1} values in Eq. (18),

$$c(2i + 1) = 0:(2i + 1), \quad \text{for } F_{m-2} \le i < 2F_m,$$

can be expressed as

$$c(2(i + F_{m-2}) + 1) = 0:(2(i + F_{m-2}) + 1), \quad \text{for } 0 \le i < F_{m+1}. \quad (33)$$

Thus, using Eqs. (31) and (33) in performing

$$c(2(i + 2F_m) + 1) :=\text{o } c(2(i + F_{m-2}) + 1), \quad \text{for } 0 \le i < F_{m+1},$$

we have

$$c(2(i + 2F_m) + 1) = 0:(2(i + 2F_m) + 1), \quad \text{for } 0 \le i < F_{m+1},$$

which is equivalent to

$$c(2i + 1) = 0:(2i + 1), \quad \text{for } 2F_m \le i < 2F_m + F_{m+1}.$$

Combining the above equation and Eq. (18), we obtain Eq. (29).

To perform

$$c(2(i + 2F_m + F_{m+1}) + 1) :=\text{o } c(2(i + 2F_m)), \quad \text{for } 0 \le i < N - 2F_m - F_{m+1},$$

the $N - 2F_m - F_{m+1}$ values in Eq. (21),

$$c(2i) = 2(i - F_{m-1} + 1):(2i + 1), \quad \text{for } 2F_m \le i < N - F_{m+1},$$

are employed, which can be rewritten as

$$c(2(i + 2F_m)) = 2(i + 2F_m - F_{m-1} + 1):(2(i + 2F_m) + 1), \text{ for } 0 \le i < N - 2F_m - F_{m+1}. \quad (34)$$

All the values in Eq. (32) are also employed in the operations, which can be expressed as

$$c(2(i + 2F_m + F_{m+1}) + 1) = 2(i + 2F_m + 1):(2(i + 2F_m + F_{m+1}) + 1), \quad \text{for } 0 \le i < N - 2F_m - F_{m+1}. \quad (35)$$

Thus, using Eqs. (34) and (35) in the operations, we have

$$c(2(i + 2F_m + F_{m+1}) + 1) = 2(i + 2F_m - F_{m-1} + 1):(2(i + 2F_m + F_{m+1}) + 1), \quad \text{for } 0 \le i < N - 2F_m - F_{m+1}.$$

The above equation is equivalent to Eq. (30). Therefore, the claim holds.

(v) Finally, we will show that after stage E, or after step $m + 3$,

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 0 \le i < N.$$

Eq. (30) is equivalent to

$$c(2(i + 2F_m + F_{m+1}) + 1) = 2(i + 2F_m - F_{m-1} + 1):(2(i + 2F_m + F_{m+1}) + 1), \quad \text{for } 0 \le i < N - 2F_m - F_{m+1}. \quad (36)$$

Furthermore, the $N - 2F_m - F_{m+1}$ values in Eq. (29),

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 2F_m - F_{m-1} \le i < N - F_{m-1} - F_{m+1},$$

are equivalent to

$$c(2(i + 2F_m - F_{m-1}) + 1) = 0:(2(i + 2F_m - F_{m-1}) + 1), \quad \text{for } 0 \le i < N - 2F_m - F_{m+1}. \quad (37)$$

Thus, using Eqs. (37) and (36) in performing

$$c(2(i + 2F_m + F_{m+1}) + 1) :=\text{o } c(2(i + 2F_m - F_{m-1}) + 1), \qquad 0 \le i < N - 2F_m - F_{m+1},$$

we have

$$c(2(i + 2F_m + F_{m+1}) + 1) = 0:(2(i + 2F_m + F_{m+1}) + 1), \quad \text{for } 0 \le i < N - 2F_m - F_{m+1},$$

which is equivalent to

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 2F_m + F_{m+1} \le i < N.$$

Combining the above equation and Eq. (29), we get

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 0 \le i < N. \qquad \text{Q.E.D.}$$

APPENDIX B

Proof of Lemma 2: We distinguish five cases.

(i) First, consider the case where $m \geq 3$ and $N > L_{m-1}$. As already mentioned, no stage is skipped. If step 0 is not considered, then stages A, B, and C of Algorithm CL are the same as those of Algorithm LL except that in CL, it is possible to jump from stage B to stage F. Because no stage is skipped, as in Algorithm LL, after stage B, as well as after stage C, we have Eq. (21), which is copied here for convenience:

$$c(2i) = 2(i - F_{m-1} + 1):(2i + 1), \qquad \text{for } F_{m-1} \leq i < N - F_{m-2}. \tag{38}$$

We will now show that, after step k in stage D,

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 0 \leq i < G(k, m); \tag{39}$$
$$c(2i + 1) = 2(i - F_{m+1} - (k - m - 1)F_{m-1} + 1):(2i + 1), \quad G(k, m) \leq i < N. \tag{40}$$

This can be proved by induction on k as follows. When $k = m + 2$, step $m + 2$ is equivalent to step $m + 2$ of Algorithm LL. Thus, after step $m + 2$, from the proof of Lemma 1, we have Eqs. (29) and (30), which can be rewritten as

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 0 \leq i < 2F_m + F_{m+1} = G(k, m);$$
$$c(2i + 1) = 2(i - F_{m+1} - F_{m-1} + 1):(2i + 1), \qquad \text{for } G(k, m) \leq i < N.$$

Because stage D begins with step $m + 2$, the basis holds.

As the induction hypothesis, assume that, after step $k = m + s - 1 < t - 1$, where $s \geq 3$, we have

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 0 \leq i < G(m + s - 1, m); \tag{41}$$
$$c(2i + 1) = 2(i - F_{m+1} - (s - 2)F_{m-1} + 1):(2i + 1), \quad \text{for } G(m + s - 1, m) \leq i < N. \tag{42}$$

We can partition Eq. (42) into two portions

$$c(2i + 1) = 2(i - F_{m+1} - (s - 2)F_{m-1} + 1):(2i + 1), \text{ for } G(m + s - 1, m) \leq i < G(m + s, m);$$

and

$$c(2i + 1) = 2(i - F_{m+1} - (s - 2)F_{m-1} + 1):(2i + 1), \quad \text{for } G(m + s, m) \leq i < N.$$

The above two equations are, respectively, equivalent to

$$c(2(i + G(m + s - 1, m)) + 1)$$
$$= 2(i + G(m + s - 1, m) - F_{m+1} - (s - 2)F_{m-1} + 1):2(i + G(m + s - 1, m)) + 1,$$
$$\text{for } 0 \leq i < F_{m+1} + (s - 2)F_{m-1} \tag{43}$$

and

$$c(2(i + G(m + s, m)) + 1)$$
$$= 2(i + G(m + s - 1, m) + 1):2(i + G(m + s, m)) + 1,$$
$$\text{for } 0 \le i < N - G(m + s, m) \quad (44)$$

In step $k = m + s$, the last $F_{m+1} + (s - 2)F_{m-1}$ values in Eq. (41),

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } G(m + s - 1, m) - F_{m+1} - (s - 2)F_{m-1} \le i < G(m + s - 1, m),$$

are employed in operations. These values can be rewritten as

$$c(2(i + G(m + s - 1, m) - F_{m+1} - (s - 2)F_{m-1}) + 1)$$
$$= 0:2(i + G(m + s - 1, m) - F_{m+1} - (s - 2)F_{m-1}) + 1, \quad \text{for } 0 \le i < F_{m+1} + (s - 2)F_{m-1}. \quad (45)$$

Using Eqs. (45) and (43) in performing

$$c(2(i + G(m + s - 1, m)) + 1) :=\text{o } c(2(i + G(m + s - 1, m) - F_{m+1} - (s - 2)F_{m-1}) + 1),$$
$$\text{for } 0 \le i < F_{m+1} + (s - 2)F_{m-1},$$

in stage D, we have

$$c(2(i + G(m + s - 1, m)) + 1) = 0:2(i + G(m + s - 1, m)) + 1,$$
$$\text{for } 0 \le i < F_{m+1} + (s - 2)F_{m-1},$$

which is equivalent to

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } G(m + s - 1, m) \le i < G(m + s, m).$$

Combining the above equation and Eq. (41), we get

$$c(2i + 1) = 0:(2i + 1), \qquad \text{for } 0 \le i < G(m + s, m). \quad (46)$$

Also in step $k = m + s$, the $N - G(m + s, m)$ values in Eq. (38),

$$c(2i) = 2(i - F_{m-1} + 1):(2i + 1), \qquad \text{for } G(m + s - 1, m) \le i < N - F_{m+1} - (s - 2)F_{m-1},$$

are employed in other operations. These values can be rewritten as

$$c(2(i + G(m + s - 1, m)))$$
$$= 2(i - F_{m-1} + G(m + s - 1, m) + 1):2(i + G(m + s - 1, m)) + 1,$$
$$\text{for } 0 \le i < N - G(m + s, m). \quad (47)$$

Using Eqs. (47) and (44) in performing

$$c(2(i + G(m + s, m)) + 1) :=\text{o } c(2(i + G(m + s - 1, m))),$$
$$\text{for } 0 \le i < N - G(m + s, m),$$

in stage D, we have

$$c(2(i + G(m + s, m)) + 1)$$
$$= 2(i - F_{m-1} + G(m + s - 1, m) + 1):2(i + G(m + s, m)) + 1, \quad \text{for } 0 \le i < N - G(m + s, m).$$

The above equation is equivalent to

$$c(2i + 1) = 2(i - F_{m+1} - (s - 1)F_{m-1} + 1):(2i + 1), \quad \text{for } G(m + s, m) = G(k, m) \le i < N. \tag{48}$$

Thus, by Eqs. (46) and (48), the claim holds for $k = m + s$. That is, during stage D, Eqs. (39) and (40) hold after step k, where $m + 2 \le k \le t - 1$. Therefore, after step $t - 1$ and before stage E, we have

$$c(2i + 1) = 0:(2i + 1), \quad \text{for } 0 \le i < G(t - 1, m); \tag{49}$$
$$c(2i + 1) = 2(i - F_{m+1} - (t - m - 2)F_{m-1} + 1):(2i + 1), \quad \text{for } G(t - 1, m) \le i < N. \tag{50}$$

Now, consider the effect of executing stage E. The $N - G(t - 1, m)$ values in Eq. (49),

$$c(2i + 1) = 0:(2i + 1),$$
$$\text{for } G(t - 1, m) - F_{m+1} - (t - m - 2)F_{m-1} \le i < N - F_{m+1} - (t - m - 2)F_{m-1}.$$

can be rewritten as

$$c(2(i + G(t - 1, m) - F_{m+1} - (t - m - 2)F_{m-1}) + 1)$$
$$= 0:(2(i + G(t - 1, m) - F_{m+1} - (t - m - 2)F_{m-1}) + 1), \quad \text{for } 0 \le i < N - G(t - 1, m). \tag{51}$$

In addition, Eq. (50) is equivalent to

$$c(2(i + G(t - 1, m)) + 1)$$
$$= 2(i + G(t - 1, m) - F_{m+1} - (t - m - 2)F_{m-1} + 1):(2(i + G(t - 1, m)) + 1), \quad \text{for } 0 \le i < N - G(t - 1, m). \tag{52}$$

Using Eqs. (51) and (52) in performing

$$c(2(i + G(t - 1, m)) + 1) :=o\ c(2(i + G(t - 1, m) - F_{m+1} - (t - m - 2)F_{m-1}) + 1), \quad \text{for } 0 \le i < N - G(t - 1, m),$$

during stage E, we get

$$c(2(i + G(t - 1, m)) + 1) = 0:(2(i + G(t - 1, m)) + 1), \quad \text{for } 0 \le i < N - G(t - 1, m).$$

The above equation is equivalent to

$$c(2i + 1) = 0:(2i + 1), \quad \text{for } G(t - 1, m) \le i < N.$$

Combining the above equation and Eq. (49), we have

$$c(2i + 1) = 0{:}(2i + 1), \qquad \text{for } 0 \le i < N,$$

before executing stage F.

(ii) Second, consider the case where $m \ge 3$ and $L_{m-1} \ge N > 2F_m$, in which only stage D is skipped. In this case, Algorithm CL performs the same operations as LL does in step 1 through stage C. Thus, after stage C, we have Eqs. (18) and (19), which are copied here for convenience:

$$c(2i + 1) = 0{:}(2i + 1), \qquad \text{for } 0 \le i < 2F_m; \tag{53}$$
$$c(2i + 1) = 2(i - F_{m+1} + 1){:}(2i + 1), \qquad \text{for } 2F_m \le i < N. \tag{54}$$

Eq. (53) is equivalent to

$$c(2(i - F_{m+1}) + 1) = 0{:}(2(i - F_{m+1}) + 1), \qquad \text{for } F_{m+1} \le i < L_{m-1}. \tag{55}$$

Note that stage E is executed in step $t = m + 2$. Thus, using Eq. (54) and the $N - 2F_m$ values in Eq. (55),

$$c(2(i - F_{m+1}) + 1) = 0{:}(2(i - F_{m+1}) + 1), \qquad \text{for } 2F_m \le i < N,$$

to perform

$$c(2(i + 2F_m) + 1) :=\text{o } c(2(i + 2F_m - F_{m+1}) + 1), \qquad \text{for } 0 \le i < N - 2F_m,$$

or equivalently,

$$c(2i + 1) :=\text{o } c(2(i - F_{m+1}) + 1), \qquad \text{for } 2F_m \le i < N,$$

in stage E, we obtain

$$c(2i + 1) = 0{:}(2i + 1), \qquad \text{for } 2F_m \le i < N.$$

Combining the above equation and Eq. (53), we have

$$c(2i + 1) = 0{:}(2i + 1), \qquad \text{for } 0 \le i < N. \tag{56}$$

(iii) Third, consider the case where $m \ge 3$ and $2F_m \ge N > F_m$, in which stages D and E are skipped. In proving Lemma 1, we have shown that after stage C, Eq. (18) holds. Since $2F_m \ge N$, following the same approach used in proving Lemma 1, we can have Eq. (56) after stage C as well as just before executing stage F.

(iv) Consider the case where $m \ge 3$ and $F_m \ge$ N, in which stages C, D, and E are skipped. In the proof of Lemma 1, we have shown that if m is even (respectively odd) and k is even (respectively odd), then after step k in stage B, we can obtain

$$c(2i + 1) = 0{:}(2i + 1), \qquad \text{for } 0 \le i < F_k.$$

Now, after executing stage B and before stage F, we have $F_k \geq N$; therefore, following the same approach used in proving Lemma 1, we can obtain Eq. (56) before executing stage F.

(v) Finally, consider the case where $m = 2$, in which only stage B is not executed. Since $N > 4 = L_1$, we have $N > L_{m-1}$, which also holds in case (i) of this proof. We have seen in the proof of Lemma 1 that regardless of the value of m, after stage C, we have Eqs. (18), (19), (20), and (21), which are also used in case (i) of this proof. Therefore, as in case (i), after executing stage E and before stage F, we have Eq. (56). Q.E.D.

ACKNOWLEDGMENT

This research was supported in part by the National Science Council of the R.O.C. under contract NSC85-2213-E-011-017. The authors thank the reviewers for valuable comments and suggestions.

REFERENCES

1. S. G. Akl, *Parallel Computation: Models and Methods*, Prentice-Hall, Upper Saddle River, NJ, 1997.
2. R. Cole and U. Vishkin, "Faster optimal parallel prefix sums and list ranking," *Information and Control*, Vol. 81, 1989, pp. 334-352.
3. A. L. Fisher and A. M. Ghuloum, "Parallelizing complex scans and reductions," in *Proceedings of ACM SIGPLAN '94 Conference on Programming Language Design and Implementation*, 1994, pp. 135-146.
4. C. P. Kruskal, L. Rudolph and M. Snir, "The power of parallel prefix," *IEEE Transactions on Computers*, Vol. C-34, No. 10, 1985, pp. 965-968.
5. C. P. Kruskal, T. Madej and L. Rudolph, "Parallel prefix on fully connected direct connection machines," in *Proceedings of International Conference on Parallel Processing*, 1986, pp. 278-284.
6. S. Lakshmivarahan and S.K. Dhall, *Parallel Computing Using the Prefix Problem*, Oxford University Press, Oxford, UK, 1994.
7. F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, San Mateo, CA, 1992.
8. 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.
9. G. E. Blelloch, "Scans as primitive operations," *IEEE Transactions on Computers*, Vol. 38, No. 11, 1989, pp. 1526-1538.
10. Y. C. Lin and C. M. Lin, "Parallel prefix on a fully-connected machine with the weakest communication capability," Technical Report NTIT-ET-TR95008, Department of Electronic Engineering, National Taiwan Institute of Technology, Taipei, R.O.C., 1995.
11. Y. C. Lin and C. M. Lin, "Efficient parallel prefix algorithms on fully connected message-passing computers," in *Proceedings of 3rd International Conference on High Performance Computing*, 1996, pp. 316-321.

12. R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *Journal of the Association for Computing Machinery*, Vol. 27, No. 4, 1980, pp. 831-838.
13. D. A. Carlson and B. Sugla, "Limited width parallel prefix circuits," *The Journal of Supercomputing*, Vol. 4, No. 2, 1990, pp. 107-129.
14. F. E. Fich, "New bounds for parallel prefix circuits," in *Proceedings of 15th Symposium on the Theory of Computing*, 1983, pp. 100-109.
15. Y. C. Lin, "Optimal prefix circuits with fan-out 2," in *Proceedings of International Conference on Algorithms*, 1996, pp. 175-181.
16. Y. C. Lin and C. C. Shih, "A new class of depth-size optimal parallel prefix circuits," *The Journal of Supercomputing*, Vol. 14, No. 1, 1999, pp. 39-52.
17. Y. C. Lin and 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*, 1998, pp. 312-317.
18. M. Snir, "Depth-size trade-offs for parallel prefix computation," *Journal of Algorithms*, Vol. 7, 1986, pp. 185-201.
19. 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.
20. T. Agerwala, J. L. Martin, J. H. Mirza, D. C. Sadler, D. M. Dias and M. Snir, "SP2 system architecture," *IBM Systems Journal*, Vol. 34, No. 2, 1995, pp. 152-184.
21. *nCUBE, nCUBE 2 Supercomputers: Systems, nCUBE*, Beaverton, Oregon, 1990.
22. Thinking Machines, *The Connection Machine CM-5 Technical Summary*, Thinking Machines, Cambridge, MA, 1992.
23. W. Groscup, "The Intel Paragon XP/S supercomputer," in *Proceedings of 5th ECMWF Workshop on the Use of Parallel Processors in Meteorology*, 1993, pp. 173-187.
24. Parasoft, *Express 3.0 Introductory Guide*, Parasoft, Pasadena, CA, 1990.
25. A. Skjellum and A. P. Leung, "Zipcode: A portable multicomputer communication library atop the reactive kernel," in *Proceedings of 5th Distributed Memory Computing Conference*, 1990, pp. 328-337.
26. G. A. Geist, M. T. Heath, B. W. Peyton and P. H. Worley, *A User's Guide to PICL: A Portable Instrumented Communication Library*, ORNL Technical Report ORNL/TM-11616, 1990.
27. Inmos, *The Transputer Databook*, 3rd ed., Inmos, Almondsbury, Bristol, UK, 1992.
28. Y. C. Lin and H. Y. Lai, "Perfectly overlapped generation of long runs on a transputer array for sorting," *Microprocessors and Microsystems*, Vol. 20, No. 5, 1997, pp. 529-539.
29. 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.
30. C. B. Stunkel, D. G. Shea, B. Abali, M. G. Atkins, C. A. Bender, D. G. Grice, P. H. Hochschild, D. J. Josoph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao and P. R. Varker, "The SP2 High-Performance Switch," *IBM Systems Journal*, Vol. 34, No. 2, 1995, pp. 185-204.
31. J. Bruck, C. T. Ho, S. Kipnis, E. Upfal and D. Weathersby, "Efficient algorithms for all-to-all communications in multiport message-passing systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 11, 1997, pp. 1143-1155.
32. R. L. Graham, D. E. Knuth and O. Patashnik, *Concrete Mathematics*, Addison-Wesley,

Reading, MA, 1989.
33. Y. C. Lin, "Optimal parallel prefix circuits with fan-out 2 and corresponding parallel algorithms," *Neural, Parallel and Scientific Computations*, Vol. 7, No. 1, 1999, pp. 33-42.
34. M. J. Quinn, *Parallel Computing: Theory and Practice*, McGraw-Hill, New York, 1994.
35. C. S. Yeh, *Efficient Parallel Prefix Algorithms on the Multiport and Multiport Postal Models*, Master Thesis, Department of Electronic Engineering, National Taiwan University of Science and Technology, 1998.
36. R. C. Entringer and P. J. Slater, "Gossips and telegraph," *Journal of Franklin Institute*, Vol. 307, 1979, pp. 353-360.

Yen-Chun Lin (林彥君) received his Ph.D. degree in electrical engineering from National Taiwan University in 1988. Since 1988, he has been on the faculty of the National Taiwan Institute of Technology, which was renamed National Taiwan University of Science and Technology in August 1997. He has been a Professor in the Department of Electronic Engineering since 1993. He was a Senior Software Engineer at Wang Computer (Taiwan) Ltd. from 1984 to 1985 and was an Associate Development Engineer at IBM Taiwan Corporation from 1985 to 1988. He was a Visiting Scientist at the IBM Almaden Research Center, San Jose, CA, from 1993 to 1994. His research interests include parallel computing. Dr. Lin is a member of the IEEE and the ACM.

C. M. Lin (林崇銘) received his bachelor degree in information and computer engineering from Chung Yuan Christian University, Chung-Li, Taiwan, in 1994 and received his M.S. degree in electronic engineering from the National Taiwan Institute of Technology, Taipei in 1996. His research interests include parallel computing.