New Families of Computation-Efficient Parallel Prefix Algorithms

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
yclin747@gmail.com
http://faculty.csie.ntust.edu.tw/~yclin/yclin.htm

LI-LING HUNG
Department of Computer Science and Information Engineering
Aletheia University
32 Chen-Li St., Tamsui, Taipei County 251
TAIWAN
llhung@mail.au.edu.tw

Abstract: - New families of computation-efficient parallel prefix algorithms for message-passing multicomputers are presented. The first family improves the communication time of a previous family of parallel prefix algorithms; both use only half-duplex communications. Two other families adopt collective communication operations to reduce the communication times of the former two, respectively. The precondition of the presented algorithms is also given. These families each provide the flexibility of either fewer computation time steps or fewer communication time steps to achieve the minimal running time depending on the ratio of the time required by a communication step to the time required by a computation step. Relative merits and drawbacks of parallel prefix algorithms are described and illustrated to provide insights into when and why the presented algorithms can be best used.

Key-Words: - Collective communication, Computation-efficient parallel prefix, Half-duplex communication, Message-passing multicomputer, Parallel algorithm, Precondition, Prefix computation

1 Introduction

Prefix computation is considered as a primitive operation because of its importance and usefulness [4]. It can be defined as follows: given n inputs x_1, x_2,…, x_n, and an associative binary operator $\oplus$, compute $y_i = x_1 \oplus x_2 \oplus \ldots \oplus x_i$, for $1 \le i \le n$. For ease of presentation, unless otherwise stated, this study uses x_i's and y_i's as inputs and outputs, respectively, and assumes that 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, 10, 13, 15, 24-27, 29, 54, 56, 58]. The binary operation $\oplus$ can be as simple as a Boolean operation or an extremely time-consuming operation, such as multiplication of matrices [11].

Numerous parallel prefix algorithms for various parallel computing models have been proposed [1, 7, 9, 12, 18-20, 25, 27, 30, 31, 39, 44, 45, 47, 49], and many parallel prefix circuits have also been designed [3, 5, 6, 14, 17, 22, 26-28, 31, 33-35, 37, 38, 40-43, 46, 48, 50, 51, 57, 58]. Additionally, prefix computation is a built-in operation for Message-Passing Interface parallel programming [16], and is implemented in hardware in the Thinking Machines CM-5 [53].

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 $p < n$ [11]. 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 [23]. 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 [21, 52]. 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.

Lin proposes a parallel prefix algorithm, henceforth named L, whose communication time is less than that of EK on the same model [32]. L has been generalized to become a family of parallel algorithms named A [19], which allow multiple combinations of the computation time and communication time; L is just an extreme member of the family. The other members of the family take less computation time and may also take less communication time than L does. One can take the exact time of performing $\oplus$ and that of communicating a message into account to obtain a member that requires the minimal running time on a specific multicomputer.

In this paper, new families of computation-efficient parallel prefix algorithms for message-passing parallel computers with p PEs, where $p < n$, are presented. The communication time of the A family can be decreased to result in another family. We then use collective communications to further improve the communication time, and derive one more family of algorithms from each of the first two families. Collective communication operations, such as broadcast and scatter, can be better than an equivalent sequence of send or receive operations for ease of programming and execution efficiency [55].

The rest of this paper is organized as follows. Section 2 reviews the A family of parallel prefix algorithms for half-duplex message-passing multicomputers, including the computation time and communication time, as well as other properties, of this family. Section 3 shows how the communication time of the A family can be decreased to become the second family. Section 4 reduces the communication time by using collective communications to obtain two more families of algorithms. Section 5 shows that the four families of algorithms are not always effective when $p < n$, and derives a much stronger precondition of the algorithms. Section 6 compares prefix algorithms for multicomputers. Conclusions are finally drawn in Section 7.

2 Review of the A family of parallel prefix algorithms

In this section, we review the A family of parallel algorithms for solving the prefix problem of n inputs with p PEs, where $p < n$, on the half-duplex multicomputer model. The p PEs are represented by $P_1, P_2,\ldots, P_p$. For ease of presentation, $i{:}j$ is used to represent the result of computing

$$x_i \oplus x_{i+1} \oplus \ldots \oplus x_j, \text{ where } i \le j.$$

Algorithm A(n, p, k) {Solving the prefix problem of n inputs, $x_1, x_2,\ldots, x_n$, using p PEs to generate $y_1, y_2,\ldots, y_n$, where $p = kq + 1$, $k \ge 1$, $q \ge 1$. In phase 1, k PEs are assigned to perform only computations; all the other PEs need to communicate among themselves, except when $p - k = 1$. To use this algorithm effectively, $n \ge (p^2 + kp + k + 1)/2$ is required. 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 $0 < v < n$. The value of v will be explained shortly. If $p = k + 1$, then P_1 uses N_1 to compute outputs $y_1, y_2,\ldots, y_v$ sequentially; otherwise, $P_1, P_2,\ldots, P_{p-k}$ use N_1 to compute $y_1, y_2,\ldots, y_v$ by invoking A(v, $p - k$, k) recursively. In the mean time, N_2 is first distributed evenly among the other k PEs, $P_{p-k+1}, P_{p-k+2},\ldots, P_p$; each of the PEs holds $c = (n - v)/k$ input values. These k PEs then take $c - 1$ parallel computation steps to compute $z_1 = (z_{1,1}, z_{1,2},\ldots, z_{1,c})$, $z_2 = (z_{2,1}, z_{2,2},\ldots, z_{2,c}),\ldots, z_k = (z_{k,1}, z_{k,2},\ldots, z_{k,c})$, 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}.

(Figure 1 should help the reader understand Phase 1.)

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 perform

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

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

(Figure 2 should help the reader understand Phase 2.)

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 perform

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

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

(Figure 3 should help the reader understand Phases 3 through $k + 1$.)

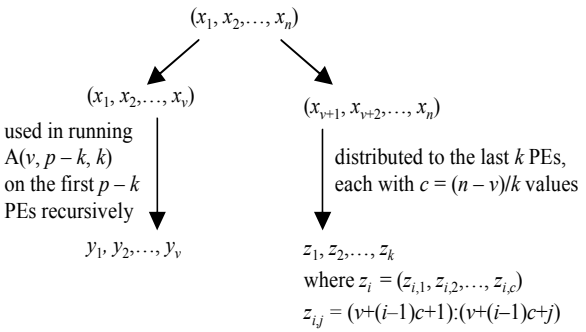

Fig. 1. Phase 1 of Algorithm A(n, p, k) when $p > k + 1$.

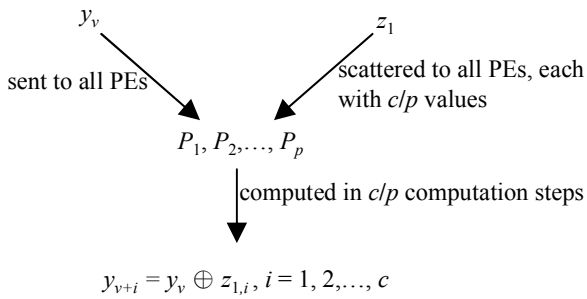

Fig. 2. Phase 2 of Algorithm A(n, p, k).

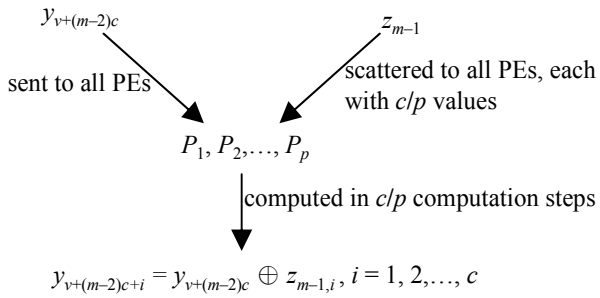

Fig. 3. Phase m (m = 3, 4,..., k + 1) of A(n, p, k).

Let $C(n, p, k)$ denote the number of computation steps required by Algorithm A(n, p, k), and $R(n, p, k)$ denote the number of communication steps. As in the two previous papers [11, 32], the initial input data loading time is not taken into account. To help the reader understand the algorithm, we give two examples in the following.

First, consider the case when $p = 4$, $k = 3$.

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, and P_4. The prefixes of N_1 are computed by P_1, and N_2 is processed 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, $(n - v)/3 - 1$. Hence, $v = n/4$, and it takes $n/4 - 1$ computation steps in this phase. After phase 1 has completed, P_1 obtains $y_1, y_2, \ldots, y_{n/4}$; P_2 obtains $z_{1,1}, z_{1,2}, \ldots, z_{1,n/4}$; P_3 obtains $z_{2,1}, z_{2,2}, \ldots, z_{2,n/4}$; and P_4 obtains $z_{3,1}, z_{3,2}, \ldots, z_{3,n/4}$.

Phase 2: P_1 initially sends $y_{n/4}$ to P_2, P_3, P_4 in 3 communication steps. Then, P_2 sends 1/4 of the $n/4$ values obtained in phase 1 to each of the other three PEs in 3 communication steps. That is, $z_{1,1}$ through $z_{1,n/16}$, $z_{1,n/8+1}$ through $z_{1,3n/16}$, and $z_{1,3n/16+1}$ through $z_{1,n/4}$ are sent to P_1, P_3, P_4, respectively. Subsequently, the four PEs compute $n/4$ outputs $y_{n/4+1}, y_{n/4+2}, \ldots, y_{n/2}$ in $n/16$ parallel computation steps. At the end, P_4 has $y_{n/2}$.

Phase 3: P_4 initially sends $y_{n/2}$ to the other PEs in 3 communication steps. Then, P_3 sends 1/4 of the $n/4$ values obtained in phase 1 to each of the other three PEs in 3 communication steps. That is, $z_{2,1}$ through $z_{2,n/16}$, $z_{2,n/16+1}$ through $z_{2,n/8}$, and $z_{2,3n/16+1}$ through $z_{2,n/4}$ are sent to P_1, P_2, P_4, respectively. Subsequently, the four PEs compute $n/4$ outputs $y_{n/2+1}, y_{n/2+2}, \ldots, y_{3n/4}$ concurrently in $n/16$ computation steps. At the end, P_4 has $y_{3n/4}$.

Phase 4: P_4 sends $y_{3n/4}$ and 1/4 of the $n/4$ values obtained in phase 1 to each of the other three PEs in 3 communication steps. That is, $z_{3,1}$ through $z_{3,n/16}$, $z_{3,n/16+1}$ through $z_{3,n/8}$, and $z_{3,n/8+1}$ through $z_{3,3n/16}$ are sent to P_1, P_2, P_3, respectively. Subsequently, the four PEs concurrently compute $n/4$ outputs $y_{3n/4+1}$, $y_{3n/4+2}, \ldots, y_n$ in $n/16$ computation steps.

Therefore, the total number of computation steps is

$$C(n, 4, 3) = (n/4 - 1) + n/16 + n/16 + n/16 = 7n/16 - 1. \quad (1)$$

The total number of communication steps is

$$R(n, 4, 3) = 3 \times 2 + 3 \times 2 + 3 = 15.$$

Next, consider the case when $p = 7$, $k = 3$.

Phase 1: Assign $N_1 = (x_1, x_2, \ldots, x_v)$ to the first four PEs, and assign $N_2 = (x_{v+1}, x_{v+2}, \ldots, x_n)$ to the last three PEs. From Eq. (1), we know that P_1, P_2, P_3, P_4 can compute the prefixes of N_1 in $C(v, 4, 3) = 7v/16 - 1$ computation steps. In the mean time, P_5, P_6, P_7 share N_2 evenly and compute their respective prefixes concurrently, taking $(n - v)/3 - 1$ computation steps. By the rule of deciding v,

$$7v/16 - 1 = (n - v)/3 - 1,$$
$$v = 16n/37.$$

Consequently, each of the last 3 PEs has $(n - v)/3 = 7n/37$ input values. Thus, the values $z_{1,1}$ through $z_{1,7n/37}$ are obtained in P_5, $z_{2,1}$ through $z_{2,7n/37}$ in P_6, and $z_{3,1}$ through $z_{3,7n/37}$ in P_7. Note that P_4 obtains $y_v = y_{16n/37}$, and $C(16n/37, 4, 3) = 7n/37 - 1$.

Phase 2: P_4 initially sends $y_{16n/37}$ to the other six PEs in 6 communication steps. Then, P_5 sends 1/7 of the $7n/37$ values obtained in phase 1 to each of the other six PEs in 6 communication steps. Subsequently, the seven PEs compute $7n/37$ outputs $y_{16n/37+1}, y_{16n/37+2}, \ldots, y_{23n/37}$ in $n/37$ parallel computation steps. At the end, P_7 has $y_{23n/37}$.

Phase 3: P_7 initially sends $y_{23n/37}$ to the other six PEs in 6 communication steps. Then, P_6 sends 1/7 of the $7n/37$ values obtained in phase 1 to each of the other six PEs in 6 communication steps. Subsequently, the seven PEs concurrently compute $7n/37$ outputs $y_{23n/37+1}$, $y_{23n/37+2}$,…, $y_{30n/37}$ in $n/37$ computation steps. At the end, P_7 has $y_{30n/37}$.

Phase 4: P_7 initially sends $y_{30n/37}$ and 1/7 of the $7n/37$ values obtained in phase 1 to each of the other six PEs in 6 communication steps. The seven PEs then concurrently compute $7n/37$ outputs $y_{30n/37+1}$, $y_{30n/37+2}$,…, y_n in $n/37$ computation steps.

Therefore, the total number of computation steps is

$$C(n, 7, 3) = 7n/37 - 1 + (n/37) \times 3 = 10n/37 - 1.$$

The total number of communication steps is

$$R(n, 7, 3) = R(16n/37, 4, 3) + 6 \times 5 = 45.$$

$C(n, p, k)$ and $R(n, p, k)$ in the general case, as well as other properties of Algorithm A, such as the values of p and k that lead to the smallest $C(n, p, k)$ and $R(n, p, k)$, are given in the following theorems.

Theorem 1. When $p = k + 1$, $v=n/p$; otherwise,

$$v = \frac{p^2-kp+k+1}{p^2+kp+k+1} n.$$

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

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 $k \geq 2$.

Theorem 5. If $d > a$, then $R(n, d, k) > R(n, a, k)$. If

$$p \geq 2k^2 - 3k + 1 \text{ and } k \geq 2,$$

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,

$$R(n, p, 1) < R(n, p, k) < R(n, p, k + 1).$$

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

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

Note that Theorems 4 and 7 reveal that the communication time is independent of n when $n \geq (p^2 + kp + k + 1)/2$. Like Algorithms EK and L, Algorithm A is 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 a time-consuming floating-point matrix multiplication or a series of matrix multiplications.

3 Algorithm B with reduced communication time

When $p \geq 2k + 1$ and $k \geq 2$, Algorithm A can be modified to become a faster algorithm named B. Recall that in phase 1 of Algorithm A, the number of computation steps required by the first $p - k$ PEs equals the number of computation steps required by the other k PEs. In addition, the first $p - k$ PEs communicate among themselves in phase 1, but the last k PEs perform no communication operations; thus, the last k PEs are idle for the amount of time that the first $p - k$ PEs take to communicate.

To reduce this idle time, we note that in Algorithm A there are communications among the last k PEs in phases 2 through $k + 1$, which can be performed in phase 1, but the communications involving the first $p - k$ PEs in phases 2 through $k + 1$ should not be moved to phase 1. In each of these k phases, the original communications that can be moved to phase 1 are message transfers from one of the last k PEs to the other $k - 1$ PEs. Thus, at most $k(k - 1)$ communication steps can be moved to phase 1.

Note that we still need to make sure whether all or only part of the $k(k - 1)$ communication steps should be moved. Comparing $k(k - 1)$ with the number of communication steps performed in phase 1 of Algorithm A, which is $R(v, p - k, k)$, can clarify this. From Theorem 4,

$$R(v, p - k, k) = (2k - 1)(p - k - 1)(p - 1)/2k.$$

Since $p \geq 2k + 1$, we have

$$R(v, p - k, k) \geq (2k - 1)k(2k)/2k = k(2k - 1).$$

Thus,

$$R(v, p - k, k) - k(k - 1) \geq k^2 > 0.$$

That is, $R(v, p - k, k) > k(k - 1)$. Therefore, all the $k(k - 1)$ communication steps can be performed in phase 1 without increasing the communication time needed in phase 1. Clearly, Algorithm B(n, p, k) takes less communication time than Algorithm A(n, p, k). Note that the two algorithms take the same amount of computation time.

Let $S(n, p, k)$ denote the number of communication steps required by Algorithm B(n, p, k). Thus, $S(n, p, k)$ is the sum of the following components:

(i) $S(v, p-k, k)$ communication steps required by the first $p-k$ PEs when invoking $B(v, p-k, k)$ in phase 1.
(ii) $p-1$ communication steps required by P_{p-k} to send y_v to the other $p-1$ PEs in phase 2.
(iii) $(k-1)(p-1)$ 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) $k(p-k)$ communication steps taken to distribute part of z_i, for $1 \le i \le k$, evenly among $P_1, P_2, \ldots P_{p-k}$ in phases 2 through $k+1$.

Note that in phase $k+1$, since the communications that contribute to component (iii) and those that contribute to component (iv) are all sent out by P_p, the latter communications can be merged to the former ones, eliminating $p-k$ communication steps in this phase. Let

$$p = k(i+1)+1, \text{ where } i \ge 1.$$

It can be shown that [36]

$$S(n, p, k) = \frac{2k-1}{2k}p^2 - \frac{1}{2}p + k^2 - 3k + 2.$$

Clearly, $S(n, p, k) = \Theta(p^2)$. Therefore, the discussion of cost optimality of Algorithm A applies to Algorithm B. That is, Algorithm B is also cost optimal when $n = \Omega(p^3)$.

4 Using Collective Communications

In this section, we propose to reduce the communication time of Algorithms A and B when collective communications are available. Their computation parts remain the same.

We first consider modifying Algorithm A to obtain a new one named E. In phase $i+2$, $0 \le i \le k-1$, of Algorithm A, the $p-1$ transfers of y_{v+ic} to all the other PEs can be replaced by a single broadcast to achieve the same effect; in addition, scattering of z_{i+1} can be done with a single scatter operation.

With a suitable implementation, either a broadcast or scatter operation takes $\Theta(\log p)$ time to send to p PEs on a multicomputer [48]. Thus, broadcasting the k values $y_v, y_{v+c}, \ldots, y_{v+(k-1)c}$ is equivalent to taking $c_1 k \log p$ point-to-point communication steps, where c_1 is a constant, and k scatter operations are equivalent to taking $c_2 k \log p$ point-to-point communication steps, where c_2 is a constant. Let $c = c_1 + c_2$ and $p = k(i+1)+1$, where $i \ge 0$**.** We use $T(n, p, k)$ to denote the number of communication steps required by Algorithm $E(n, p, k)$. It can be shown that $T(n, p, k) = O(p \log p)$ [36].

Algorithm E takes the same the number of computation steps as the former two families do. Since Theorem 2 gives $C(n, p, k) = \Theta(n/p)$, Algorithm E takes $\Theta(n/p) + O(p \log p)$ time. If $n = \Omega(p^2 \log p)$, then $n/p = \Omega(p \log p)$, and thus $\Theta(n/p) + O(p \log p) = \Theta(n/p)$. Since the sequential solution for the prefix problem takes $\Theta(n)$ time, Algorithm E is cost optimal when $n = \Omega(p^2 \log p)$.

The modification made to Algorithm B by using broadcast and scatter can be done similarly, and results in the same communication time complexity. Thus, the resulting algorithm, named F, is also cost optimal when $n = \Omega(p^2 \log p)$. Note that Algorithm F takes the same number of computation steps as the other three families do.

5 Precondition of algorithms

Theorem 7 gives that the relation

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

is required to use Algorithm A effectively. We can see that the same is also true for the other algorithms introduced in this paper. Before going into the general case, we first use two examples to shed some light.

Suppose $n = 1024$, $p = 256$, and $k = 85$. From Theorem 2, we have

$$C(1024, 256, 85) < 7.$$

However, it is even impossible to compute the sum of 1024 inputs in 7 computation steps, let alone the prefix computation of 1024 inputs. Therefore, Theorem 2 does not hold under this situation. That is, the four families of algorithms cannot complete the prefix computation in fewer than 7 computation steps.

As a more general case, suppose $n = 1024$, $p = 256 = kq + 1$, and $q \ge 1$. From Theorem 1, $v = 1024(256^2 - 256k + k + 1)/(256^2 + 256k + k + 1)$ inputs are assigned to the first $256 - k$ PEs, and

$$\begin{aligned} n - v &= 2kpn/(p^2 + kp + k + 1) \\ &= 512 \times 1024k/(256^2 + 256k + k + 1) \\ &< 8k \end{aligned}$$

inputs are assigned to the last k PEs. That is, each of the last k PEs has at most 8 inputs. Then, in phase 2, P_{257-k} sends at most 8 values to at most 8 of the 256 PEs for further computation; that is, at least 248 PEs are idle while the other PEs are busy. This ineffective use of PEs also exists in phases 3 through $k+1$ of all four families.

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

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

Therefore, this is the precondition for running the presented algorithms effectively.

6 Comparison of algorithms

Lin and Lin present a parallel prefix algorithm named PLL for the half-duplex multicomputer [39]; 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 $10 \le p < n$. The number of computation steps of any of our algorithms is less than that of PLL, but the number of communication steps is greater than that of 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. From Theorem 2, a larger k leads to less computation time. As for the communication time, from Theorem 5, $R(n, p, k) \le R(n, p, 1)$ when $p \ge 2k^2 - 3k + 1$ and $k \ge 2$; otherwise, $R(n, p, k) > R(n, p, 1)$. That is, A(n, p, k) is definitely faster than L when $p \ge 2k^2 - 3k + 1$ and $k \ge 2$. However, when $p < 2k^2 - 3k + 1$, we need to know the ratio of the time required by a communication step to the time required by a computation step, τ, to decide which algorithm is faster.

Algorithm B can also be faster than Algorithm L. To compare their communication times, we see that when $k \ge 2$,

$R(n , p, 1) - S(n, p, k) = (p^2 - kp - 2k^3 + 6k^2 - 4k)/2k$.

Recall that Algorithm B exists only when $p \ge 2k + 1$ and $k \ge 2$. Therefore, when $p^2 - kp - 2k^3 + 6k^2 - 4k > 0$, $p \ge 2k + 1$, and $k \ge 2$, Algorithm B is faster than Algorithm L; otherwise, we need to know the value of τ to decide which is faster.

To help the reader understand the relative merits and drawbacks of these algorithms, we give some example figures in the following. Fig. 4 shows the numbers of computation steps required when n = 8192 and p = 13. Note that k represents the number of PEs that perform no communications in phase 1 of our algorithms. Since PLL is a very different algorithm, it should not appear in the figures; however, for easy comparison, we show its related data in every figure as if k = 1. Clearly, Algorithms A and E each take the fewest number of computation steps when k = 12. Note that when $2 \le k \le (p - 1)/2$, i.e., when Algorithms B and F are defined, the two algorithms each take the same number of computation steps as Algorithm A. Algorithm PLL takes the most, 1265, computation steps.

Figure 5 shows the numbers of communication steps required by four algorithms. We assume that each collective communication involving p PEs requires the same amount of time as $2 \log_2 p$ point-to-point communications. Algorithm E takes the fewest communication steps among the four algorithms. As already mentioned, although the exact number of communication steps of Algorithm F is not derived, Algorithm F, when defined, needs slightly less communication time than Algorithm E. Moreover, EK takes 3550 communication steps, which is too large to fit in the figure; PLL takes 7 communication steps, which is too small to fit in the figure.

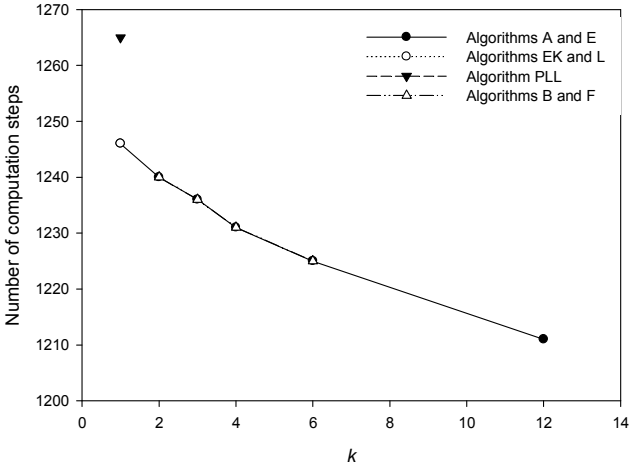

Fig. 4. The numbers of computation steps required when n = 8192 and p = 13.

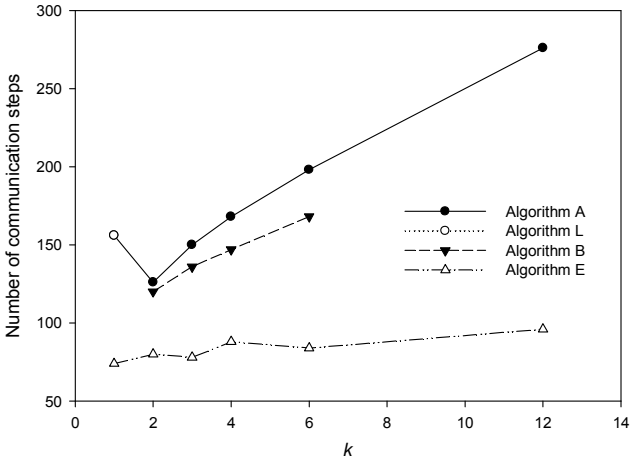

Fig. 5. The numbers of communication steps required when n = 8192 and p = 13.

Note that τ has an impact on the total running time. If τ is small, which may be true when the binary operation $\oplus$ is matrix multiplication, our new algorithms are more probable to be faster than previous ones. For example, Figs. 6 and 7 show the running times of algorithms when n = 8192 and p =

13, for $\tau = 1$ and 0.1, respectively. When $\tau = 1$, Fig. 6 shows that Algorithm E is faster than A, B, and L; however, Algorithm PLL is the fastest. Algorithm F, which is not shown in the figure, should be faster than E, but might be slower than PLL.

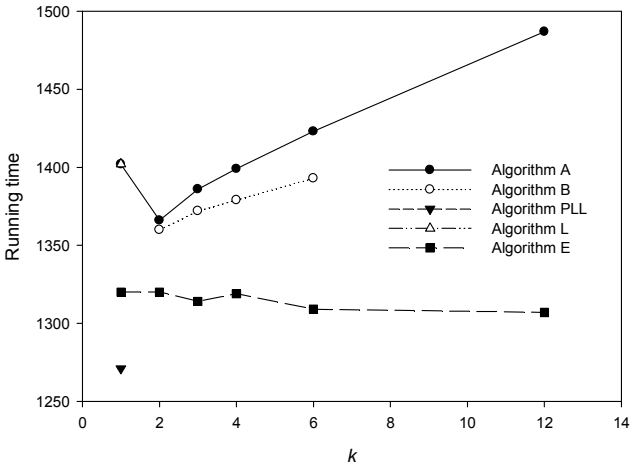

Fig. 6. Running times in number of computation steps when $n = 8192$, $p = 13$, and $\tau = 1$.

In contrast, as shown in Fig. 7, when $\tau = 0.1$, Algorithm B is generally faster than A and L, but A(8192, 13, 12) is faster than B(8192, 13, 6). Note that $k = 12$ is not valid for Algorithm B. Algorithm E is faster than the others, and E(8192, 13, 12) is the fastest. Also note that since F(8192, 13, 6) is only slightly faster than E(8192, 13, 6), E(8192, 13, 12) should be faster than F(8192, 13, 6). Note that Algorithm PLL becomes the slowest.

To contrast the running times when $n = 4096$ with those when $n = 8192$, Fig. 8 is given. The figure shows that Algorithm E is faster than the other algorithms, and E(4096, 13, 12) is the fastest.

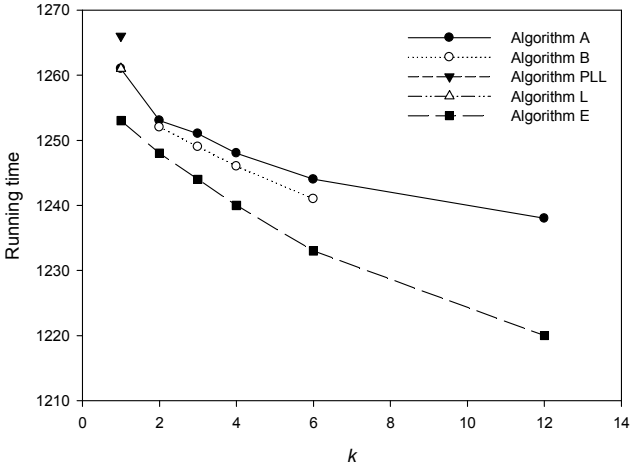

Fig. 7. Running times in number of computation steps when $n = 8192$, $p = 13$, and $\tau = 0.1$.

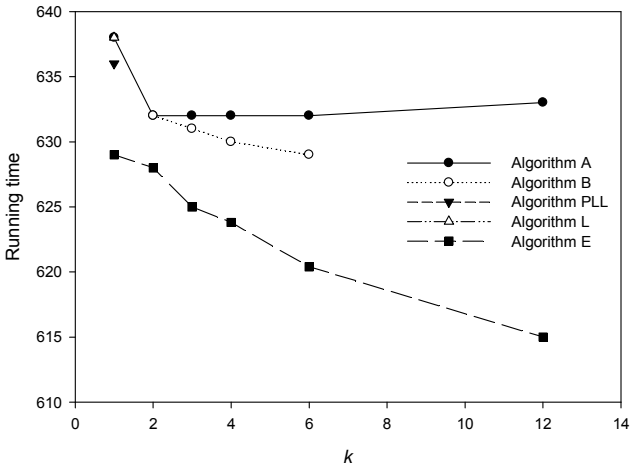

Fig. 8. Running times in number of computation steps when $n = 4096$, $p = 13$, and $\tau = 0.1$.

7 Conclusions

We have presented new algorithms to speed up A(n, p, k), a family of computation-efficient parallel algorithms 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 $n \geq (p^2 + kp + k + 1)/2$. This family is cost optimal when $n = \Omega(p^3)$. Algorithm A has been modified to become another family, Algorithm B, which may run faster than A. Either A or B can be transformed into another new family by adopting broadcast and scatter operations to reduce the communication time. The resulting two families are cost optimal when $n = \Omega(p^2 \log p)$.

Each of the four families of algorithms provides the flexibility of using either fewer computation time steps (and more communication time steps) or fewer communication time steps (and more computation time steps) to achieve the minimal running time. The key in considering this computation-communication trade-off and, more importantly, whether the running time is less than other prefix algorithms hinge on the ratio of the time required by a communication step to the time required by a computation step.

For the presented algorithms, the last k PEs are idle for some amount of time in phase 1. Clearly, in phase 1, if we assign fewer than v inputs to the first $p - k$ PEs and thus more than $n - v$ inputs to the last k PEs, then the first $p - k$ PEs do fewer computations and the last k PEs do more. Therefore, the idle time can be eliminated, and the running time reduced. How many inputs should be assigned to the first $p - k$ PEs is still open.

Although we have used the multicomputer model to present the algorithms, they can be adapted to suit shared-memory multiprocessors. The multiprocessor

usually needs less communication time than the multicomputer. Thus, the multiprocessor should also be a good platform for the presented algorithms.

Acknowledgment
This research was supported in part by the National Science Council of Taiwan under contract NSC 98-2221-E-011-072.

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 and 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 and 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 and B. Sugla, Limited width parallel prefix circuits, *Journal of Supercomputing,* Vol. 4, No. 2, 1990, pp. 107-129.

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

[8] R. Cole and U. Vishkin, Faster optimal parallel prefix sums and list ranking, *Information and Computation,* 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] 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.

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

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

[13] A. Ferreira and S. Ubeda, Parallel complexity of the medial axis computation, in *Proc. Int. Conf. on Image Processing*, Washington, D.C., 1995, pp. 105-108.

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

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

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

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

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

[19] L.-L. Hung and Y.-C. Lin, Parallel prefix algorithms on the multicomputer, *WSEAS Transactions on Computer Research,* Vol. 3, No. 4, 2008, pp. 229-239.

[20] L.-L. Hung and Y.-C. Lin, Two families of parallel prefix algorithms for multicomputers, in *Proc. 7th WSEAS International Conference on Telecommunications and Informatics*, Istanbul, Turkey, 2008, pp. 37-43.

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

[22] P. M. Kogge and 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.

[23] 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.

[24] C. P. Kruskal, T. Madej, and L. Rudolph, Parallel prefix on fully connected direct connection machines, in *Proc. Int. Conf. on Parallel Processing*, St. Charles, IL, 1986, pp. 278-284.

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

[26] R. E. Ladner and M. J. Fischer, Parallel prefix computation, *Journal of the ACM,* Vol. 27, No. 4, 1980, pp. 831-838.

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

[28] 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 *Proc. Int. Conf. on Parallel Processing*, St. Charles, IL, 1987, pp. 58-65.

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

[30] 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.

[31] 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.

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

[33] Y.-C. Lin and 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.

[34] Y.-C. Lin and 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.

[35] 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.

[36] Y.-C. Lin and 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.

[37] Y.-C. Lin and L.-L. Hung, Fast problem-size-independent parallel prefix circuits, *Journal of Parallel and Distributed Computing,* Vol. 69, No. 4, 2009, pp. 382-388.

[38] Y.-C. Lin and L.-L. Hung, Straightforward construction of depth-size optimal, parallel prefix circuits with fan-out 2, *ACM Transactions on Design Automation of Electronic Systems,* Vol. 14, No. 1, 2009, Article 15.

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

[40] Y.-C. Lin and 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.

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

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

[43] Y.-C. Lin and 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.

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

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

[46] J. Liu, S. Zhou, H. Zhu, and C.-K. Cheng, An algorithmic approach for generic parallel adders, in *Proc. Int. Conf. on Computer-Aided Design*, San Jose, CA, 2003, pp. 734-740.

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

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

[49] 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.

[50] M. Sheeran and 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.

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

[52] 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.

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

[54] 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.

[55] Z. Xu and 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.

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

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

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