Proceedings of the 6th WSEAS International Conference on Applied Informatics and Communications, Elounda, Greece, August 18-20, 2006 (pp280-285)

Computation-Efficient Parallel Prefix

YEN-CHUN LIN
Department of Computer Science and Information Engineering
National Taiwan University of Science and Technology
43 Keelung Road, Section 4, Taipei 106
TAIWAN

Abstract: - We are interested in solving the prefix problem of n inputs using $p < n$ processors on completely connected distributed-memory multicomputers (CCDMMs). This paper improves a previous work in three respects. First, the communication time of the previous algorithm is reduced significantly. Second, we show that $p(p + 1)/2 < n$ is required for the new algorithm and the original one to be applicable. Third, we argue that for the new algorithm to be faster than other algorithms run on CCDMMs, $n > p^3$ is required. The new algorithm can achieve linear speedup and is cost-optimal when $n = \Omega(p^2 \log p)$.

Key-Words: - Completely connected multicomputer, cost-optimal, half-duplex, linear speedup, parallel algorithms, prefix computation

1 Introduction

Given n inputs $x_1, x_2, \ldots, x_n$ and an associative binary operation, denoted by o, the prefix computation problem, or simply the prefix problem, is to compute the n prefixes

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

In this paper, we will use x_i's and y_i's to represent inputs and outputs, respectively. Prefix computation has been extensively studied for its broad applications [1-3, 8, 10, 12, 14, 21-25, 46, 48, 50, 51]; it is used, for example, in biological sequence comparison, cryptography, loop parallelization, the solution of linear recurrences, carry-look-ahead addition, polynomial evaluation and interpolation, and digital filtering. The binary operation o can be a simple Boolean operation or a time-consuming multiplication of matrices [11]. Because of its importance, prefix computation has been proposed as a basic operation [4]. In fact, many parallel prefix algorithms have been proposed [1, 7, 9, 17, 22, 24, 26, 27, 33, 38, 39, 41-43], and many parallel prefix circuits have also been designed [3, 5, 6, 13, 16, 19, 23, 24, 27-30, 34-37, 40, 44, 49-51]. In particular, Egecioglu and Kog give a computation-efficient parallel prefix algorithm (henceforth named EK) for the completely connected distributed-memory multicomputer (CCDMM) model with $p < n$ processing elements (PEs) [11].

With half-duplex communication, each PE of a CCDMM can only send a message to or receive a message from any other PE in a communication step. The half-duplex model of communication is very important [20]. Although a PE of a modern multicomputer can send and receive in the same step, it usually takes longer to send and receive in a step than to send only or receive only due to the inherent hardware capability and software overhead [18, 31, 45]. On a p-PE system, the half-duplex communication ensures that no more than $p/2$ messages are communicated in a communication step and thus a communication step will not take too much time.

In this paper, we also solve the prefix problem of n input items on the same CCDMM. This paper first presents an algorithm that improves on the communication time of Algorithm EK on the same half-duplex CCDMM model. Then, we show how the communication time can be further reduced with a stronger communication capability. The communication time is reduced from $\Theta(n) = \Omega(p^3)$ of Algorithm EK to $\Theta(p \log p)$. These algorithms each take $2n(p + 1)/(p(p + 1) + 2) - 1$ computation steps. We then take two different approaches to show that merely $p < n$ is not enough for these algorithms to be applicable; exactly, $p(p + 1)/2 < n$ is required. We also show that for these algorithms to be faster than the other CCDMM algorithms, $n > p^3$ is required. Thus, the communication time is reduced from $\Theta(n) = \Omega(p^3)$ to $\Theta(p \log p)$.

Section 2 presents the new prefix algorithm for the half-duplex CCDMM. Section 3 uses the broadcast and scatter collective communication operations to further reduce the communication time. Section 4 shows that Algorithm EK and our new algorithm require a stricter condition than $p < n$, and derives a much stronger condition of $p(p + 1)/2 < n$.

Section 5 compares the new algorithm with other CCDMM algorithms. Section 6 concludes this paper.

2 Parallel Prefix for the CCDMM

In this section, we describe a parallel algorithm that solves the prefix problem of n inputs using $p < n$ PEs (exactly, as will be shown in a later section, $n > p(p + 1)/2$). The p PEs will be represented by $P_1, P_2, \dots$, and P_p. To simplify the presentation, $l{:}m$ is used to represent the result of computing $x_l \text{ o } x_{l+1} \text{ o} \dots \text{o } x_m$, where $l \le m$. Like Algorithm EK, it is assumed that the time required to perform a binary operation is greater than that required to transfer a message between PEs. The algorithm has two phases. In the first phase, we partition the inputs $x_1, x_2, \dots, x_n$ into two parts $L_1 = (x_1, x_2, \dots, x_v)$ and $L_2 = (x_{v+1}, x_{v+2}, \dots, x_n)$, where $v = \lceil \alpha_p n \rceil$ and $0 < \alpha_p < 1$. The value of v is chosen to make the number of computation steps required by using the first $p - 1$ PEs to concurrently compute the prefixes of the first v inputs in L_1 equal to the number of computation steps required by the last PE, P_p, to compute the prefixes of the $n - v$ inputs in L_2. For ease of presentation, we will ignore the ceiling function.

The same algorithm is used recursively for the first $p - 1$ PEs to compute the prefixes of L_1 in parallel; in the mean time, we use the last PE to compute the prefixes of L_2 sequentially. In the second phase, initially $y_v = 1{:}v$ is sent from P_{p-1} to all the other PEs. Then, we equally distribute the values $z_{v+i} = (v + 1){:}(v + i)$, $i = 1, 2, \dots, n - v$, in P_p among all the p PEs; thus, each PE contains $(n - v)/p$ values. Finally, the p PEs compute $y_{v+i} = y_v \text{ o } z_{v+i}$, $i = 1, 2, \dots, n - v$, in $(n - v)/p$ parallel computation steps.

Two measures are used to evaluate the algorithm. The first measure, denoted by $C_p(n)$, is the number of computation steps required by using p PEs to compute the prefixes of n inputs. The second measure, denoted by $R_p(n)$, is the number of communication steps required. To help understand the algorithm and derive $C_p(n)$ and $R_p(n)$, we will consider the cases $p = 2$ and $p = 3$.

First, consider the case $p = 2$. In the first phase, we assign $L_1 = (x_1, x_2, \dots, x_v)$ to P_1 and $L_2 = (x_{v+1}, x_{v+2}, \dots, x_n)$ to P_2. Then, P_1 and P_2 compute the prefixes of L_1 and L_2 independently. By the rule of choosing v, we make the number of computation steps required by P_1 and P_2 to compute the prefixes of L_1 and L_2, respectively, equal to each other. That is, $v - 1 = (n - v) - 1$. Therefore, $v = n/2$. After this phase, we will have all the prefixes of the L_1 and L_2 in P_1 and P_2, respectively. Let

$$z_{v+i} = (v + 1){:}(v + i),\ i = 1, 2, \dots, n/2,$$

be the $n/2$ prefixes computed in P_2.

In the second phase, we send $y_v = 1{:}v$ from P_1 to P_2 in a communication step, and send the first half of the prefixes computed in P_2, i.e., $z_{v+1}, z_{v+2}, \dots, z_{3n/4}$, from P_2 to P_1 in another communication step. Then, P_1 can use y_v and $z_{v+1}, z_{v+2}, \dots, z_{3n/4}$, and P_2 can use y_v and $z_{3n/4+1}, z_{3n/4+2}, \dots, z_n$ to compute the final prefixes $y_{v+1}, y_{v+2}, \dots, y_n$. Since the first phase takes $n/2 - 1$ computation steps and the second phase takes $n/4$ computation steps, the total number of computation steps is

$$C_2(n) = n/2 - 1 + n/4 = 3n/4 - 1. \qquad (1)$$

Clearly, the number of communication steps is

$$R_2(n) = 1 + 1 = 2.$$

Now consider the case $p = 3$. In the first phase, we assign $L_1 = (x_1, x_2, \dots, x_v)$ to P_1 and P_2, and assign $L_2 = (x_{v+1}, x_{v+2}, \dots, x_n)$ to P_3. Then P_1 and P_2 use the algorithm just described for $p = 2$ to compute the prefixes of L_1 in parallel; they take $C_2(v)$ computation steps. At the same time, P_3 computes the prefixes of L_2 sequentially, taking $(n - v) - 1$ computation steps. By the rule of choosing v,

$$C_2(v) = (n - v) - 1. \qquad (2)$$

Using Eqs. (1) and (2), we have

$$v = 4n/7.$$

Note that the prefixes computed in P_3 are

$$z_{4n/7+i} = (4n/7 + 1){:}(4n/7 + i),\ i = 1, 2, \dots, 3n/7.$$

In the second phase, we first send $y_{4n/7} = 1{:}4n/7$ from P_2 to both P_1 and P_3 in 2 communication steps. Then, we equally distribute the $3n/7$ prefixes in P_3 among all the three PEs; in other words, it takes 2 more communication steps to send the prefixes $z_{4n/7+1}, z_{4n/7+2}, \dots, z_{5n/7}$ from P_3 to P_1, and send the prefixes $z_{5n/7+1}, z_{5n/7+2}, \dots, z_{6n/7}$ from P_3 to P_2. Subsequently, these three PEs can use $y_{4n/7}$ and the distributed prefixes to compute the final prefixes concurrently in $n/7$ computation steps. Thus,

$$C_3(n) = C_2(4n/7) + n/7 = 3n/7 - 1 + n/7 = 4n/7 - 1,$$
$$R_3(n) = R_2(\alpha_3 n) + 2 + 2 = 6.$$

It is straightforward to obtain the following two equations from [11]:

$$C_p(n) = 2n(p + 1)/(p(p + 1) + 2) - 1, \qquad (3)$$
$$v = n(p(p - 1) + 2)/(p(p + 1) + 2). \qquad (4)$$

The values of $C_p(n)$ and v are the same as those of Algorithm EK. In contrast, the number of communication steps $R_p(n)$ is reduced. Note that $R_p(n)$ is the sum of the following: (1) the number of communication steps, $R_{p-1}(v)$, required by the first $p - 1$ PEs to compute the prefixes of v inputs, (2) the number of communication steps required by P_{p-1} to send $y_v = 1{:}v$ to the other $p - 1$ PEs, and (3) the number of communication steps taken to equally distribute among all the p PEs the $n - v$ prefixes computed by P_p. So,

$$\begin{aligned} R_p(n) &= R_{p-1}(v) + (p-1) + (p-1) \\ &= R_{p-1}(\alpha_p n) + 2p - 2 \\ &= R_{p-2}(\alpha_{p-1}\alpha_p n) + 2(p-1) - 2 + 2p - 2 \\ &= R_2(\alpha_3\alpha_4 \ldots \alpha_p n) + \sum_{i=3}^{p}(2i-2). \end{aligned}$$

Since $R_2(n) = 2$ for $n > 1$, we have $R_p(n) = p(p - 1)$.

Let us compare $R_p(n)$ with

$$R_{EK} = np(p-1)/(p(p+1)+2) + p(p-1)/2,$$

which is the number of communication steps taken by Algorithm EK. As we will see in Section 4, the two algorithms can only be applicable under the condition $n \geq p(p + 1)/2 + 1$. If $n = p(p + 1)/2 + 1$, then $R_p(n) = R_{EK}$. If $n > p(p + 1)/2 + 1$, then $R_p(n) < R_{EK}$; the new algorithm is faster than Algorithm EK. In Section 3, we will see further improvement of the communication time.

3 Further Time Reduction

For portable parallel programming, the Message-Passing Interface (MPI) [15] has been common. For ease of programming, the MPI includes collective communication primitives, which involve more than two communicating PEs in a single instruction. Using a single collective communication primitive can be experimentally faster than using a sequence of point-to-point operations to achieve the same function [47].

Thus, in the second phase of our algorithm, we can replace the $p - 1$ transfers of y_v from P_{p-1} to all other PEs with a single broadcast operation to achieve the same effect. The broadcast primitive takes $\Theta(\log p)$ steps to transfer a message to p PEs on a multicomputer, and we can assume that it takes $c_1 \log p$ steps to broadcast y_v, where c_1 is a constant and $0 < c_1 \leq 1$ [47].

Similarly, the $n - v$ prefixes computed by P_p can be equally distributed among all the PEs by a single scatter operation to further reduce the communication time. With an appropriate implementation, a scatter operation can take $\Theta(\log p)$ time, which is equivalent to $c_2 \log p$ steps, where c_2 is a constant and $0 < c_2 \leq 1$ [47]. Let $c = c_1 + c_2$. Thus,

$$\begin{aligned} R_p(n) &= R_{p-1}(\alpha_p n) + c \log p \\ &= R_{p-2}(\alpha_{p-1}\alpha_p n) + c \log(p-1) + c \log p \\ &= R_2(\alpha_3\alpha_4 \ldots \alpha_p n) + c \sum_{i=3}^{p} \log i. \end{aligned}$$

Since $R_2(n) = 2$ for $n > 1$, and $\sum_{i=3}^{p} \log i = \Theta(p \log p)$, we have

$$R_p(n) = \Theta(p \log p).$$

Therefore, the algorithm takes $\Theta(n/p + p \log p)$ time. If $n = \Omega(p^2 \log p)$, then $n/p = \Omega(p \log p)$, and thus $\Theta(n/p + p \log p) = \Theta(n/p)$. Since the sequential solution for the prefix problem takes $\Theta(n)$ time, the new algorithm achieves linear speedup and is cost-optimal when $n = \Omega(p^2 \log p)$.

4 Precondition of the Algorithms

Recall that Eq. (3) gives the number of computation steps required by Algorithm EK and our new algorithm. In this section, we show that $p < n$ is not sufficient for Eq. (3) to be valid, and a stronger relation of p and n is required.

As an example, when $p = 128$ and $n = 256$, from Eq. (3) we have $C_{128}(256) < 3$. Clearly, it is impossible to compute the prefixes of 256 inputs in 3 or fewer computation steps. This is confirmed by Snir's finding [44]. Snir has proved that the number of computation steps needed when using p PEs to compute the prefixes of $n > p$ inputs must satisfy

$$C_p(n) \geq (2n-2)/(p+1).$$

Consequently, $C_{128}(256) \geq 4$. Therefore, Eq. (3) is not valid under this situation.

In fact, the algorithms are not applicable when $p = 128$ and $n = 256$. If we try to use any of the algorithms in this case, then $v = 253$ inputs are assigned to the first 127 PEs and $n - v = 3$ inputs to the last PE. Applying the same algorithm recursively to $p = 127$ and $n = 253$ will lead to the situation that many PEs are not even assigned any input to work. Furthermore, in the second phase, when trying to equally distribute the 3 prefixes obtained by P_{128} among the PEs, 125 out of 128 PEs are actually not assigned any value. Thus, the algorithms should not be used in this case.

The situation improves when there are $n - v \geq p$ inputs assigned to the last PE. This ensures that in the second phase each PE can be distributed at least one of the prefixes computed in the first phase by the last PE. From $n - v \geq p$ and Eq. (4), we obtain the precondition for using the algorithms: $n > p(p + 1)/2$.

Therefore, the algorithms are applicable only when $n > p(p + 1)/2$. For example, if $n = 128$, the algorithms are applicable when $p \leq 15$; on the other hand, if $p = 256$, then $n > 32896$ is required.

5 Comparison with Other Algorithms

In this section, we compare the new algorithm with other algorithms that run on CCDMMs. On the basis of a parallel prefix algorithm presented by Kruskal, Rudolph, and Snir for the exclusive-read exclusive-write parallel random-access machine (PRAM) [22], a corresponding algorithm for the 1-port CCDMM with $p \leq n$ PEs requires $C_p(n) = 2n/p + \log_2 p - 2$ and $R_p(n) = \log_2 p + 1$ [11]. With k-port communication, each PE has k input ports and k output ports to communicate with other PEs in one communication step, for some $k \geq 1$. Lin and Lin present a parallel prefix algorithm named PLL for the half-duplex CCDMM [32]; with p PEs, where $10 \leq p < n$, PLL requires $C_p(n) = 2n/p + 1.44 \log_2 p - 1$ and $R_p(n) = 1.44 \log_2 p + 1$. On a k-port CCDMM of $p < n$ PEs, parallel algorithms requiring $C_p(n) = 2n/p + (k + 1) \log_{k+1} p - 2$ and $R_p(n) = \log_{k+1} p$ have also been presented [38].

Clearly, our new algorithm requires more communication steps than any of the other algorithms. Thus, the new algorithm must have a $C_p(n)$ that is small enough for it to be faster than the other algorithms. Because each of the others needs a few more than $2n/p$ computation steps, the new algorithm may be faster only if the difference between $2n/p$ and the $C_p(n)$ of Eq. (3) is large enough to compensate for its larger $R_p(n)$. Let $D = 2n/p - 2n(p + 1)/(p(p + 1) + 2) = 4n/(p^3 + p^2 + 2p)$. The value of D should be large to make our algorithm faster.

Thus, it is reasonable to say that $n > p^3$ is required for the new algorithm to be competitive. Note that the ratio of the time required by a computation step to the time required by a communication step also has an impact on whether the new algorithm is faster. If the ratio is large enough, which may be especially true when the binary operation o is matrix multiplication, the new algorithm can be faster when $n > p^3$; on the other hand, if the ratio is small, $n >> p^3$ may be required for the new algorithm to be faster.

6 Conclusion

We have presented a parallel algorithm for the CCDMM to solve the prefix problem on n inputs using p PEs, where $n > p(p + 1)/2$. The new algorithm can achieve linear speedup and is cost-optimal when $n = \Omega(p^2 \log p)$. It takes much less communication time than Algorithm EK. We have also proved that these algorithms should only work under the condition $n > p(p + 1)/2$. They require less computation time than other algorithms for CCDMMs. The new algorithm can be faster than the others when $n > p^3$.

It should be noted that the new algorithm readily yields an algorithm of time complexity $\Theta(n/p)$ for the concurrent-read exclusive-write PRAM. The two collective operations in the second phase now translate to concurrent reads. Clearly, the algorithm achieves linear speedup and is cost-optimal.

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

References:

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

[2] S. Aluru, N. Futamura, K. Mehrotra, Parallel biological sequence comparison using prefix computaions, *J. Parallel Distrib. Comput.*, Vol. 63, No. 3, 2003, pp. 264-272.

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

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

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

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

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

[8] R. Cole, U. Vishkin, Faster optimal parallel prefix sums and list ranking, *Infom. Contr.*, Vol. 81, 1989, pp. 334-352.

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

[10] C. Efstathiou, H.T. Vergos, D. Nikolos, Fast parallel-prefix modulo $2^n + 1$ adders, *IEEE*

Trans. Comput., Vol. 53, No. 9, 2004, pp. 1211-1216.

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

[12] A. Ferreira, S. Ubeda, Parallel complexity of the medial axis computation, in Proc. Int. Conf. on Image Processing, vol. 2, Washington, D.C., 1995, pp. 105-108.

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

[14] A.L. Fisher, 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.

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

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

[17] D.R. Helman, J. JaJa, Prefix computations on symmetric multiprocessors, *J. Parallel Distrib. Comput.*, Vol. 61, 2001, pp. 265-278.

[18] Inmos, *The Transputer Databook*, Inmos, 1992.

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

[20] D.W. Krumme, G. Cybenko, K.N. Venkataraman, Gossiping in minimal time, *SIAM J. Comput.*, Vol. 21, No. 1, 1992, pp. 111-139.

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

[22] C.P. Kruskal, L. Rudolph, M. Snir, The power of parallel prefix, *IEEE Trans. Comput.*, Vol. C-34, No. 10, 1985, pp. 965-968.

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

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

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

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

[27] 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.

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

[29] Y.-C. Lin, J.-W. Hsiao, A new approach to constructing optimal parallel prefix circuits with small depth, *J. Parallel Distrib. Comput.*, Vol. 64, No. 1, 2004, pp. 97-107.

[30] Y.-C. Lin, Y.-H. Hsu, C.-K. Liu, Constructing *H*4, a fast depth-size optimal parallel prefix circuit, *J. Supercomput.*, Vol. 24, No. 3, 2003, pp. 279-304.

[31] Y.-C. Lin, H.Y. Lai, Perfectly overlapped generation of long runs on a transputer array for sorting, *Microprocessors Microsystems*, Vol. 20, No. 5, 1997, pp. 529-539.

[32] Y.-C. Lin, C.M. Lin, Efficient parallel prefix algorithms on fully connected message-passing computers, in Proc. 3rd Int. Conf. on High Performance Computing, IEEE Computer Society Press, Trivandrum, India, 1996, pp. 316-321.

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

[34] Y.-C. Lin, C.-K. Liu, Finding optimal parallel prefix circuits with fan-out 2 in constant time, *Inform. Process. Lett.*, Vol. 70, No. 4, 1999, pp. 191-195.

[35] Y.-C. Lin, 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.

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

[37] Y.-C. Lin, C.-Y. Su, Faster optimal parallel prefix circuits: New algorithmic construction, *J. Parallel Distrib. Comput.*, Vol. 65, No. 12, 2005, pp. 1585-1595.

[38] Y.-C. Lin, C.-S. Yeh, Efficient parallel prefix algorithms on multiport message-passing systems, *Inform. Process. Lett.*, Vol. 71, No. 2, 1999, pp. 91-95.

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

[40] J. Liu, S. Zhou, H. Zhu, C.-K. Cheng, An algorithmic approach for generic parallel adders, in Proc. ICCAD, San Jose, CA, 2003, pp. 734-740.

[41] R. Manohar, J.A. Tierno, Asynchronous parallel prefix computation, *IEEE Trans. Comput.*, Vol. 47, No. 11, 1998, pp. 1244-1252.

[42] Y. Pan, S.Q. Zheng, K. Li, H. Shen, An improved generalization of mesh-connected computers with multiple buses, *IEEE Trans. Parallel Distributed Syst.*, Vol. 12, No. 3, 2001, pp. 293-305.

[43] E.E. Santos, Optimal and efficient algorithms for summing and prefix summing on parallel machines, *J. Parallel Distrib. Comput.*, Vol. 62, 2002, pp. 517-543.

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

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

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

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

[48] F. Zhou, P. Kornerup, Computing moments by prefix sums, *J. VLSI Signal Process. Systems*, Vol. 25, No. 1, 2000, pp. 5-17.

[49] H. Zhu, C.-K. Cheng, R. Graham, On the construction of zero-deficiency parallel prefix adders, in Proc. 13th Int. Workshop on Logic and Synthesis, Temecula, CA, 2004, pp. 280-286.

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

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