Scalable Hardware-Algorithms for Binary Prefix Sums

R. Lin, *Member*, *IEEE Computer Society*, K. Nakano, *Member*, *IEEE Computer Society*, S. Olariu, *Member*, *IEEE Computer Society*, M.C. Pinotti, *Member*, *IEEE Computer Society*, J.L. Schwing, *Member*, *IEEE*, and A.Y. Zomaya, *Senior Member*, *IEEE*

Abstract—In this work, we address the problem of designing efficient and scalable hardware-algorithms for computing the sum and prefix sums of a w^k-bit, $(k \geq 2)$, sequence using as basic building blocks linear arrays of at most w^2 shift switches, where w is a small power of 2. An immediate consequence of this feature is that in our designs broadcasts are limited to buses of length at most w^2. We adopt a VLSI delay model where the "length" of a bus is proportional with the number of devices on the bus. We begin by discussing a hardware-algorithm that computes the sum of a w^k-bit binary sequence in the time of $2k-2$ broadcasts, while the corresponding prefix sums can be computed in the time of $3k-4$ broadcasts. Quite remarkably, in spite of the fact that our hardware-algorithm uses only linear arrays of size at most w^2, the total number of broadcasts involved is less than three times the number required by an "ideal" design. We then go on to propose a second hardware-algorithm, operating in pipelined fashion, that computes the sum of a kw^k-bit binary sequence in the time of $3k + \lceil \log_w k \rceil - 3$ broadcasts. Using this design, the corresponding prefix sums can be computed in the time of $4k + \lceil \log_w k \rceil - 5$ broadcasts.

Index Terms—Hardware-algorithms, shift switching, binary prefix sums, binary counting, scalable architectures, pipelining.

1 Introduction

Recent advances in VLSI have made it possible to implement algorithm-structured chips as building blocks for high-performance computing systems.

Given an n-bit sequence $a_1, a_2, \cdots, a_n$, it is customary to refer to $p_j = a_1 + a_2 + \cdots + a_j$ as the jth *prefix sum*. The BPS problem is to compute all the prefix sums $p_1, p_2, \cdots, p_n$ of A. The task of computing binary prefix sums (BPS, for short) is a fundamental computing problem for its solution is the principle ingredient in arithmetic expression evaluation, storage and data compaction, processor allocation, load balancing, ATM switch management, and routing, among many others [1], [2], [3], [4], [5], [6], [7], [10], [11], [12], [13], [16], [27], [30], [31], [33]. In this article, we address the problem of designing efficient and scalable hardware-algorithms for the BPS problem.

Motivation for this work was provided by three sources. On the one hand, in his fundamental paper [3], Blelloch argued convincingly that *scan* operations—that boil down to parallel prefix computation—should be considered primitive parallel operations and should be, whenever possible, implemented in hardware. Blelloch argues that if scan operations find an efficient solution, then correspondingly faster solutions will become available for an entire slew of applications. Not surprisingly, scans have been implemented in hardware in the Thinking Machines CM-5 [32]. Second, Lakshmivarahan and Dhall [13] have dedicated an entire volume to showing that a vast array of important computational problems can be solved elegantly using parallel prefix computation. Thus, providing fast hardware implementations for parallel prefix computation becomes important. Third, it is well-known that one of the major drawbacks of conventional arithmetic algorithms is that they are hard to parallelize due to complications of carry propagation. One way to alleviate this problem is to transform arithmetic operations (addition in particular) into a residue code domain and then perform the computation in this domain [15], [27], [28]. In the residue code arithmetic, each binary number is represented as a set of residue digits and the arithmetic operation is performed on these residue digits in parallel. This affords *carry-free* realization of binary addition (and other operations as well). However, these realizations require different hardware structures and designs [15], [28]. One of the goals of our work was to provide such a hardware structure, specially tailored for binary prefix computation.

We adopt a VLSI delay model where the "length" of a bus is proportional with the number of devices (e.g., gates, switches) on the bus. In this context, the main contribution of this work is to show that we can use short buses in conjunction with the shift switching technique introduced recently by Lin and Olariu [16] to design a scalable hardware-algorithm for BPS. As a byproduct, we also

- *R. Lin is with the Department of Computer Science, SUNY Geneseo, Geneseo, NY 14454.*
- *K. Nakano is with the Department of Electrical and Computer Engineering, Nagoya Institute of Technology, Showa-ku, Nagoya 466-8555, Japan. E-mail: nakano@elcom.nitech.ac.jp.*
- *S. Olariu is with the Department of Computer Science, Old Dominion University, Norfolk, VA 23529. E-mail: olariu@cs.odu.edu.*
- *M.C. Pinotti is with I.E.I., C.N.R., Pisa, Italy.*
- *J.L. Schwing is with the Department of Computer Science, Central Washington University, Ellensburg, WA 98926.*
- *A.Y. Zomaya is with the Parallel Computing Research Laboratory, Department of Electrical and Electronic Engineering, The University of Western Australia, Perth, Australia 6907, Australia.*

Manuscript received 10 June 1998; revised 14 July 1999; accepted 14 Feb. 2000.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 107507.

obtain a scalable hardware-algorithm for a parallel counter, that is, for computing the sum of a binary sequence (BS).

Given the importance of parallel prefix computation, a number of parallel prefix hardware designs have been proposed in the literature [4], [10], [12]. As pointed out in [27], most of these designs turn out to have a prohibitive cost. The parallel prefix problem has also been addressed on the reconfigurable mesh architecture [14], [18], [19], [23], [20], [21], [24]. Unfortunately, it has been argued recently that the reconfigurable mesh is unrealistic as the broadcasts involved occur on buses of arbitrary length. Therefore, in practical designs its use should be avoided whenever possible.

The basic building block of our hardware-algorithms is a linear array of shift switches in which each switch can cyclically shift an incoming signal depending on the contents of a one-bit control register. Importantly, our designs do not use processors. Instead, our processing elements have a minimal control, being very close in spirit to simple circuits.

Assuming that buses of length no larger than w^2 are available, our first design computes the sum and the prefix sums of w^k bits in $2k-2$ and $3k-4$ broadcasts, respectively. Interestingly, the total number of broadcasts involved are less than three times the number required by the "ideal" designs.

We then go on to propose a second hardware-algorithm, operating in pipelined fashion, that computes the sum of a kw^k-bit binary sequence in the time of $3k+\lceil \log_w k \rceil - 3$ broadcasts. Using this design, the corresponding prefix sums can be computed in the time of $4k+\lceil \log_w k \rceil - 5$ broadcasts.

The remainder of the paper is organized as follows: Section 2.1 discusses basic data representations used in our design, Section 2.2 reviews the concept of shift switching, and Section 2.3 introduces linear arrays of shift switches and discusses the delay model.. This naive design is improved upon in two stages. In Section 2.4, we develop an efficient design under the assumption that broadcasts over long buses are allowed. This assumption is relaxed in Sections 3 and 4. Specifically, Section 3 presents an efficient design for computing the sum. Section 4 extends the ideas put forth in Section 3 for the purpose of designing an efficient architecture for the prefix sums. Sections 5 and 6 improve the architecture in Sections 3 and 4 by using pipelining. Finally, Section 7 offers concluding remarks and open problems.

2 Terminology and Background

The principle goal of the section is to fill in the reader on terminology and background material used in the remainder of this work.

2.1 Data Representation

An integer z will be represented *positionally* as $x_n x_{n-1} \cdots x_2 x_1$ in various ways as described below. Specifically, for every i, $(1 \le i \le n)$,

Binary: $x_i = \left\lfloor \frac{z}{2^{i-1}} \right\rfloor \bmod 2$;
Unary: $x_i = 1$ if $i = z+1$ and 0 otherwise;
Filled unary: $x_i = 1$ if $1 \le i \le z$ and 0 otherwise;
Distributed: x_i is either 0 or 1 and $\sum_{i=1}^{n} x_i = z$;
Base w: Given an integer w, $(w \ge 2)$, $x_i = \left\lfloor \frac{z}{w^{i-1}} \right\rfloor \bmod w$;
Unary base w: Exactly like the base w representation except that x_i is represented in unary form.

It is clear that, except for the distributed representation, all the others are uniquely defined. The filled unary representation is a particular instance of the distributed representation, where the z low order bits are 1s, the others 0s. The task of converting back and forth between these representation is straightforward and can be readily implemented in VLSI [5], [11], [30], [31].

2.2 Shift Switches—The Workhorse of Our Designs

Lin and Olariu [16] have recently proposed to endow unidirectional broadcast buses with a new feature called shift switches. The novelty of their design was to enable switches to cyclically permute an incoming signal during broadcasting.

At the heart of the design lies a simple device that we call a *switching element* which is, essentially, a *CMOS transmission gate* implemented by two transistors—a *p*MOS and an *n*MOS transistor in parallel. Just as the transmission gate, the switching element is used as steering logic in our design. It is worth pointing out that numerous articles in recent literature have shown significant advantages of pass-transistor-based realizations over conventional designs in terms of both speed and power consumption [9], [17], [25], [26], [27], [29], [35].

Consider a positive integer w, typically a small power of 2. Referring to Fig. 1, a *shift switch* $S(w)$ is an array of w identical switching elements with the state changes controlled by a one-bit state register.

The input to the $S(w)$ consists of an integer b, $(0 \le b \le w-1)$, in unary form and of a bit a. The output is an integer c, $(0 \le c \le w-1)$, in unary form, along with a bit d, such that

$$c = (a+b) \bmod w, \text{ and } d = \left\lfloor \frac{a+b}{w} \right\rfloor. \tag{1}$$

Suppose that $b_w b_{w-1} \ldots b_1$ and $c_w c_{w-1} \ldots c_1$ are the unary representation of b and c, respectively. The $S(w)$ cyclically shifts b if a is asserted; otherwise b is not changed. Notice that this corresponds to the modulo w addition of a to the unary representation of b. Consequently, $d = 1$ if and only if $b_w = 1$ and $a = 1$. For this reason d is usually referred to as the *rotation bit*.

2.3 Linear Arrays of Shift Switches and Delay Model

Before we show how to harness the power or shift switches, we feel that it is appropriate to briefly discuss the VLSI *delay model* that we adopt in this work.

Although physical length of wires is very important, we define the "length" of a bus as the number of devices (gates, switches) on the bus. The basic motivation for adopting this delay model is that future improvements in technology could shrink the length of wires, while gate delays may not shrink proportionately. Moreover, it appears that in buses where there are several taps on the medium, the number of taps has a larger impact on the bus delay than its physical

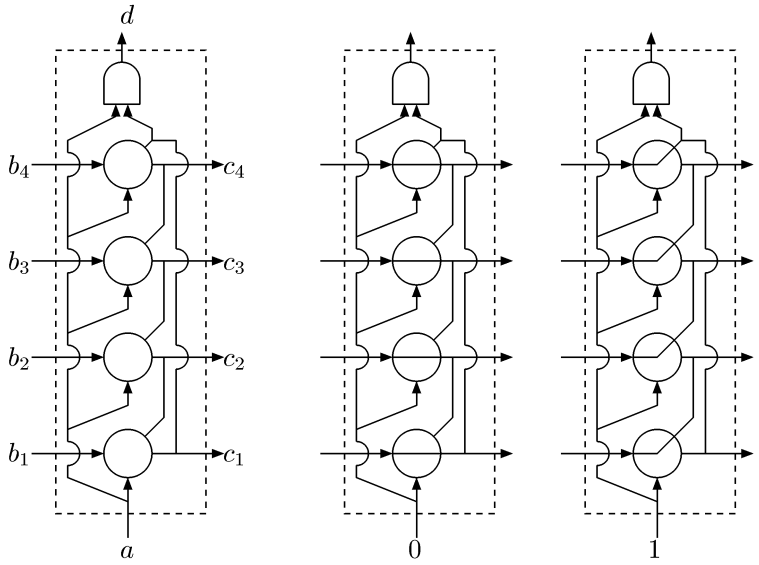


Fig. 1. Illustrating the $S(w)$.

length. For example, order of nsec delay (reasonable approximation for gate delays) corresponds to order of inches in terms of propagation delay on a wire. We note that, in fact, the classical "wire length" and our delay models coincide for two-dimensional meshes as well as other well-known regular architectures.

To further clarify our model, we note that most experts will agree that a 1 cm long wire in 0.25 technology is a "long wire" whose effect on delay should be taken into consideration, while a 1 mm wire could be ignored except for high frequency designs. In translating this "rule of thumb" to the problem of computing the prefix sums of w^k input bits, one may say that the delay on wires becomes important for the number of input bits is in the order of few billions if not more. But, surely, one will reach the pin number bound way before that.

In order to exploit the power and elegance of the $S(w)$, we construct a *linear array*$S(w,m)$ consisting of m $S(w)$s as illustrated in the left part of Fig. 2. The implementation of the $S(w,m)$ is featured in Fig. 3. Notice that

- The 1-bit input and 1-bit output of the ith $S(w)$ in left-to-right order is a_i and d_i, respectively;
- The input to the leftmost $S(w)$ and the output of the rightmost $S(w)$ of the $S(w,m)$ are denoted, respectively, by b and c.
- We refer to m, the number of $S(w)$s involved in the $S(w,m)$ as the *length of the bus*.

Suppose that $b_w b_{w-1}\cdots b_1$ and $c_w c_{w-1}\cdots c_1$ are the unary representations of b and c, respectively. Clearly, (1) guarantees that

$$c = (a_1 + a_2 + \cdots + a_m + b) \bmod w \tag{2}$$

and that

$$d = d_1 + d_2 + \cdots + d_m = \left\lfloor \frac{a_1 + a_2 + \cdots + a_m + b}{w} \right\rfloor. \tag{3}$$

In the remainder of this work, when it comes to performing the sum or the prefix sums of a binary sequence $a_1, a_2, \cdots, a_m$, we will supply this sequence as the bit input of some suitable linear array $S(w,m)$ and will insist that $b = 0$, or, equivalently, $b_w b_{w-1}\cdots b_2 b_1 = 00\cdots 01$. It is very important to note that, in this case, (2) and (3) are computing, respectively,

$$c = (a_1 + a_2 + \cdots + a_m) \bmod w, \text{ and}$$
$$d = \left\lfloor \frac{a_1 + a_2 + \cdots + a_m}{w} \right\rfloor.$$

Note that the collection of rotation bits $d_1, d_2, \ldots, d_m$ of the $S(w,m)$ is sparse in the sense that among any group of consecutive w bits in $d_1, d_2, \ldots, d_m$, there is at most one 1. In other words, with m' standing for $\lceil \frac{m}{w} \rceil$, we have for every i, $(1 \le i < m')$,

$$\begin{aligned} &d_{(i-1)\cdot w+1} + d_{(i-1)\cdot w+2} + \cdots + d_{i\cdot w} \\ &\quad = d_{(i-1)\cdot w+1} \vee d_{(i-1)\cdot w+2} \vee \cdots \vee d_{i\cdot w} \end{aligned} \tag{4}$$

and, similarly,

$$\begin{aligned} &d_{(m'-1)\cdot w+1} + d_{(m'-1)\cdot w+2} + \cdots + d_{m'\cdot w} \\ &\quad = d_{(m'-1)\cdot w+1} \vee d_{(m'-1)\cdot w+2} \vee \cdots \vee d_m. \end{aligned} \tag{5}$$

Equations (4) and (5) imply that by using $\lceil \frac{m}{w} \rceil$ OR gates with fan-in w, $d_1 + d_2 + \cdots + d_m$ is converted to an equivalent $\lceil \frac{m}{w} \rceil$-bit distributed representation. Equipped with $\lceil \frac{m}{w} \rceil$ OR gates as described above, the linear array $S(w,m)$ will be denoted by $T(w,m)$. We refer the reader to Fig. 2 for an illustration.

Notice that the computation of b and d specified in (2) and (3) above can be carried out by block $T(w,m)$ in the time of one broadcast operation. We refer to m, the number of $S(w)$s involved in the linear array $S(w,m)$, and therefore in block $T(w,m)$, as the *length of the bus*. It is clear that the broadcast operation above occurs on buses of length m. Consequently, we have the following result:

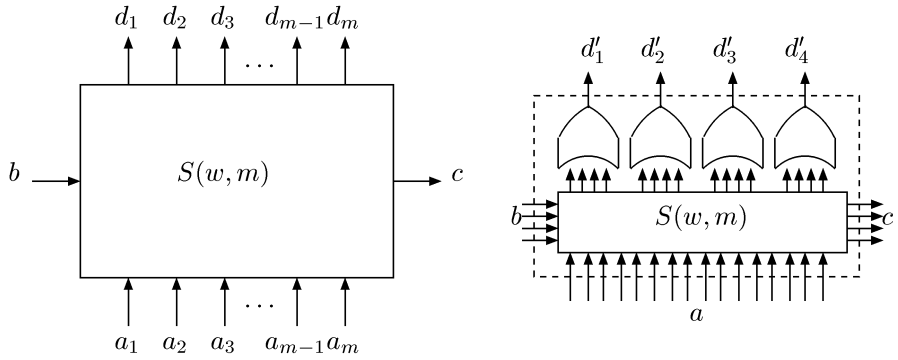


Fig. 2. A functional view of the array $S(w,m)$ and of block $T(w,m)$.

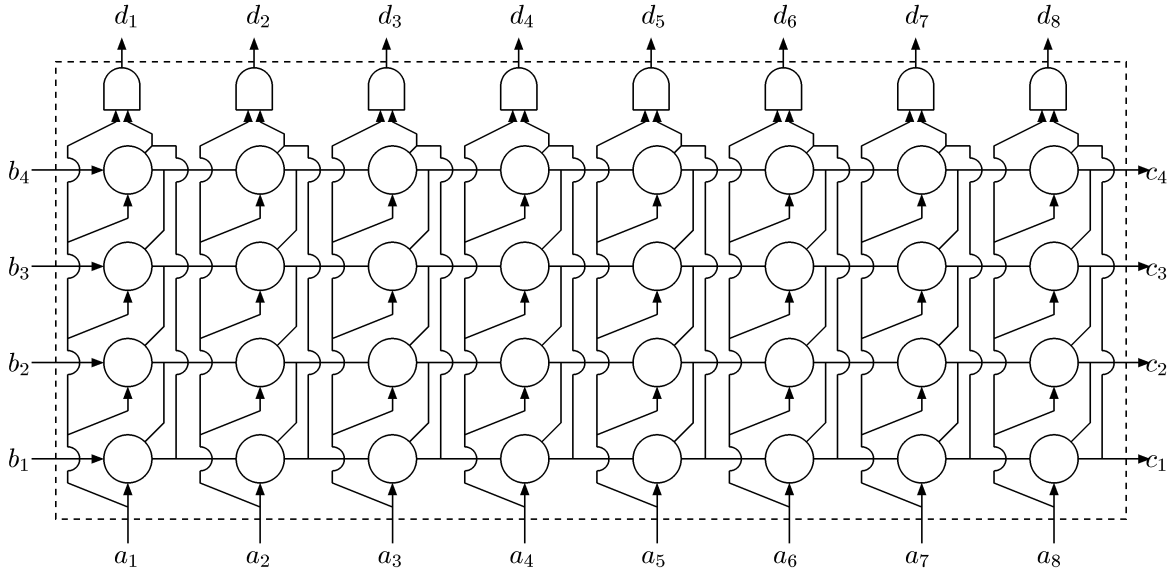


Fig. 3. Illustrating the shift switch implementation of the $S(w,m)$.

Remark 2.1. *Given the w-bit unary representation of b and the m-bit distributed representation of a, a block $T(w,m)$ can compute the w-bit unary representation of $c = (a+b) \bmod w$ and the $\lceil \frac{m}{w} \rceil$-bit distributed representation of $d = \lfloor \frac{a+b}{w} \rfloor$ in one broadcast operation over buses of length m.*

2.4 Computing Sums and Prefix Sums on Long Buses

The main goal of this subsection is to derive a simple hardware-algorithm for computing the sum and the prefix sums of an w^k-bit binary sequence, $k \geq 1$, under the assumption that blocks $T(w,m)$ for $w \leq m \leq w^k$ are available to us. We assume that broadcasting on a bus of length m takes unit time. Although this is not realistic in case m is moderately large, nonetheless, the results derived in this section will be used to motivate the improvements in the subsequent sections.

Consider a w^k-bit binary sequence $a_1, a_2, \cdots, a_{w^k}$ and write $a = a_1 + a_2 + \cdots + a_{w^k}$. Write a in unary base w representation as $A_{k+1}A_k \cdots A_1$. $A_1, A_2, \ldots, A_{k+1}$ can be computed using $D_1, D_2, \ldots, D_{k-1}$ as follows:

$$A_1 = a \bmod w \text{ and } D_1 = \left\lfloor \frac{a}{w} \right\rfloor,$$

and for every i, $(2 \leq i \leq k)$,

$$A_i = D_{i-1} \bmod w \text{ and } D_i = \left\lfloor \frac{D_{i-1}}{w} \right\rfloor.$$

Further,

$$A_{k+1} = D_k.$$

Lemma 2.1 guarantees that a block $T(w,w^k)$ computes the unary representation of A_1 as well as the w^{k-1}-bit distributed representation of D_1 in the time of one broadcast. Similarly, a block $T(w,w^{k-1})$ can compute the unary representation of A_2 and the w^{k-2}-bit distributed representation of D_2 by using as input the w^{k-1}-bit distributed representation of D_1. In general, for every i, $(2 \leq i \leq k)$, a block $T(w,w^{k-i+1})$ computes the unary representation of A_i and the w^{k-i}-bit distributed representation of D_i, using the w^{k-i+1}-bit distributed representation of D_{i-1}. Thus, collectively, the k blocks $T(w,w), T(w,w^2), \ldots, T(w,w^k)$ compute the unary base w representation of $A_{k+1}A_k \cdots A_1$ in the time of k broadcasts. We refer the reader to Fig. 4 for an illustration of this design. Here, the unary representation of $b = 0$ (i.e., 0000...01) is input to the ports of the leftmost $S(w)$ of each block $T(w,w^i)$. We summarize this discussion by stating the following result.

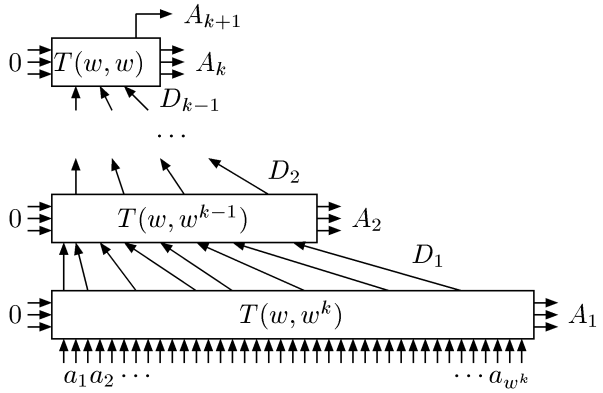


Fig. 4. Using k blocks $T(w,w), T(w,w^2), \ldots, T(w,w^k)$ to compute the sum of w^k bits.

Lemma 2.2. *The unary base w representation of the sum of w^k bits can be computed by k blocks $T(w,w), T(w,w^2), \ldots, T(w,w^k)$ in the time of k broadcasts, each over buses of length at most w^k.*

Consider again the binary sequence $a_1, a_2, \cdots, a_{w^k}$ and write for every j, $(1 \le j \le w^k)$, $p_j = a_1 + a_2 + \cdots + a_j$. Further, let $P_{k+1}^j P_k^j \cdots P_1^j$ be the unary base w representation of p_j. Notice that $P_{k+1}^1, P_{k+1}^2, \ldots, P_{k+1}^{w^k-1}$ are always 0 and that $P_{k+1}^{w^k} = 1$ only if $c_j = 1$ for all $1 \le j \le w^k$. Since this condition can be checked for very easily, from now on we shall ignore P_{k+1}^j for all j, $1 \le j \le w^k$.

Referring to Fig. 5, assume that the sequence $a_1, a_2, \cdots, a_{w^k}$ is supplied in left-to-right order to the $S(w)$s of the bottom-most $S(w,w^k)$ and that the unary representation of $b = 0$, (i.e., $000\cdots01$), is input to the ports of the leftmost $S(w)$ of this $S(w,w^k)$. Clearly, the output of the jth, $(1 \le j \le w^k)$, $S(w)$ of this $S(w,w^k)$ corresponds to P_1^j. Further, the prefix sums of the rotation bits $d_1^1, d_1^2, \ldots, d_1^{w^k}$ of this $S(w,w^k)$ correspond to the distributed representation of $D_1^1, D_1^2, \ldots, D_1^{w^k}$. Hence, it is clear that the array $S(w,w^k)$ can compute the unary representation of $P_1^1, P_1^2, \ldots, P_1^{w^k}$ and the distributed representation of $D_1^1, D_1^2, \ldots, D_1^{w^k}$ in the time of one broadcast.

In general, let $d_i^1, d_i^2, \ldots, d_i^{w^k}$ denote the rotation bits of the i-th $S(w,w^k)$. We can write

$$P_1^j = p_j \bmod w \text{ and } D_1^j = d_1^1 + d_1^2 + \cdots + d_1^j = \left\lfloor \frac{p_j}{w} \right\rfloor.$$

It is a straightforward task to confirm that for every i, $(2 \le i \le k)$,

$$P_i^j = (d_{i-1}^1 + d_{i-1}^2 + \cdots + d_{i-1}^j) \bmod w = D_{i-1}^j \bmod w$$

and that

$$D_i^j = d_i^1 + d_i^2 + \cdots + d_i^j = \left\lfloor \frac{d_{i-1}^1 + d_{i-1}^2 + \cdots + d_{i-1}^j}{w} \right\rfloor = \left\lfloor \frac{D_{i-1}^j}{w} \right\rfloor.$$

In a perfectly similar way, using $D_1^1, D_1^2, \ldots, D_1^{w^k}$ as input, the second $S(w,w^k)$ computes the unary representation of $P_2^1, P_2^2, \ldots, P_2^{w^k}$ as well as the distributed representation of $D_2^1, D_2^2, \ldots, D_2^{w^k}$. Continuing similarly, k linear arrays $S(w,w^k)$ are sufficient to compute all the P_i^j, $(1 \le i \le k, 1 \le j \le w^k)$. Thus, we have the following result.

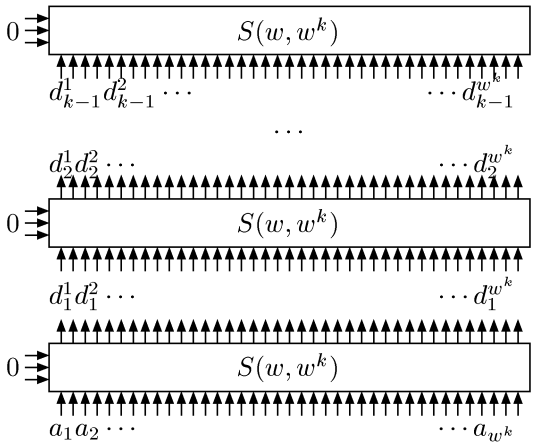


Fig. 5. Computing the prefix sums of w^k bits using k $S(w,w^k)$s.

Remark 2.3. *The unary base w representation of the prefix sums of a w^k-*bit *binary sequence can be computed by k linear arrays $S(w,w^k)$ in the time of k broadcasts, each over buses of length w^k.*

Notice that, in fact, there is no need for k arrays $S(w,w^k)$. By feeding the rotation bits back as input, the same computation can be performed by one $S(w,w^k)$, as shown in [16]. Thus, we have the following result:

Remark 2.4. *The unary base w representation of the prefix sums of a w^k-*bit *binary sequence can be computed by a linear array $S(w,w^k)$ in the time of k broadcasts, each over buses of length w^k.*

3 Computing Sums on Short Buses

In spite of their simplicity and intuitive appeal, the hardware-algorithm discussed in the previous section is less than perfect. The problem, of course, is that the broadcasts occur on buses of length w^k. The main goal of this section is to propose a hardware-algorithm with broadcasts restricted to short buses only. To be more precise, the building blocks used by the new design are linear arrays $S(w,m)$ and blocks $T(w,m)$ with $m \le w^2$.

Notice that, in principle, one can simulate a block $T(w,w^k)$ in the time of w^{k-2} broadcasts by using the concatenation of w^{k-2} blocks $T(w,w^2)$. Thus, the design in Fig. 4 can be implemented using $w^{k-2} + w^{k-3} + \cdots + w^0 + 1$ blocks $T(w,w^2)$ and one block $T(w,w)$. Unfortunately, this construction needs $O(w^{k-2})$ broadcasts, which is much larger than k. The remainder of this section is devoted to the task of reducing the number of broadcasts to $O(k)$.

Our design relies on a new block $U(w,w^k)$, $(k \ge 1)$, whose input is a w^k-bit binary sequence $a_1, a_2, \cdots, a_{w^k}$ and whose output is the unary base w representation $A_{k+1}A_k \cdots A_1$ of the sum $a_1 + a_2 + \cdots + a_{w^k}$. As before, we

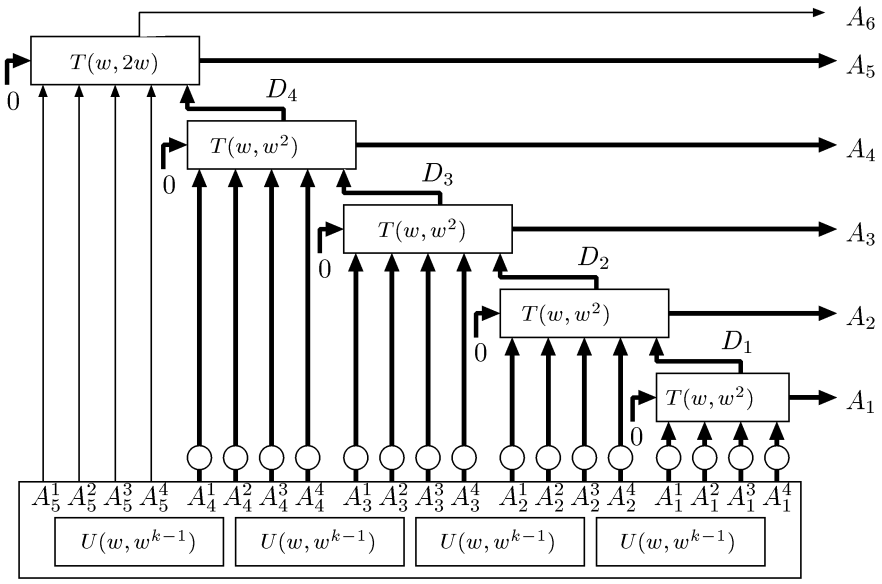


Fig. 6. Illustrating the recursive construction of $U(w, w^k)$.

note that A_{k+1} is 1 only if $a_1 = a_2 = \cdots = a_{w^k} = 1$. Recall that for every i, $(1 \le i \le k)$, $0 \le A_i \le w - 1$.

The block $U(w, w^k)$ is defined recursively as follows: $U(w, w^1)$ is just $T(w, w)$. $U(w, w^2)$ is implemented using blocks $T(w, w)$ and $T(w, w^2)$, as illustrated in Fig. 4. Let $k \ge 3$ and assume that the block $U(w, w^{k-1})$ has already been constructed. We now detail the construction of the block $U(w, w^k)$ using w blocks $U(w, w^{k-1})$. The reader is referred to Fig. 6 for an illustration of this construction in the case of $w = 4$ and $k = 5$.

For an arbitrary i, $(1 \le i \le w)$, let $A_k^i A_{k-1}^i \cdots A_1^i$ denote the base w representation of the output of the ith block $U(w, w^{k-1})$. We have

$$A_1 = (A_1^1 + A_1^2 + \cdots + A_1^w) \bmod w \text{ and}$$
$$D_1 = \left\lfloor \frac{A_1^1 + A_1^2 + \cdots + A_1^w}{w} \right\rfloor.$$

An easy argument asserts that for every i, $(2 \le i \le k)$,

$$A_i = (A_i^1 + A_i^2 + \cdots + A_i^w + D_{i-1}) \bmod w \text{ and that}$$
$$D_i = \left\lfloor \frac{A_i^1 + A_i^2 + \cdots + A_i^w + D_{i-1}}{w} \right\rfloor.$$

Similarly,

$$A_{k+1} = D_k.$$

The equations above confirm that the design in Fig. 6 computes the unary base w representation of the sum $a_1 + a_2 + \cdots + a_{w^k}$; here, the circles indicate circuitry to convert from unary to filled unary representation. Specifically, the rightmost $T(w, w^2)$ computes A_1. It receives the unary representation of 0 (i.e., $00\cdots01$) from the left and adds the filled unary representation of $A_1^1, A_1^2, \ldots, A_1^w$ to A_1^1. The output emerging from the right is the unary representation of A_1 and the rotation bits are exactly the distributed representation of D_1. The next $T(w, w^2)$ computes A_2 in the same manner: it receives the unary representation of 0 from the left, and adds to it the filled unary representation of $A_2^1, A_2^2, \ldots, A_2^w$ as well as the distributed representation of D_1. Clearly, the output emerging from the right is the unary representation of A_2. Since the unary representation of each $A_2^1, A_2^2, \ldots, A_2^w$ has $w - 1$ bits and D_1 has w bits, the total input bits from the bottom is $w(w-1) + w = w^2$. Thus, $T(w, w^2)$ has sufficient input terminals. Continuing in the same fashion, $k - 1$ blocks $T(w, w^2)$ compute $A_1, A_2, \ldots, A_{k-1}$. Since $A_k^1, A_k^2, \ldots, A_k^w$ consist of 1 bit each and since D_{k-1} involves w bits, a block $T(w, 2w)$ is used to compute A_k. Notice that the output, D_k, has 2 bits. As noted before, their logical OR is exactly A_{k+1}.

At this time it is appropriate to estimate the number of broadcasts involved in the above computation.

Lemma 3.1. *For every k, $(k \ge 2)$, and for every i, $(1 \le i \le k)$, the block $U(w, w^k)$ returns the unary base w representation of A_i at the end of $k + i - 2$ broadcasts and that of A_{k+1} at the end of $2k - 2$ broadcasts. Moreover, all the broadcasts take place on buses of length at most w^2.*

Proof. The proof is by induction on k. The basis is easy: the construction of $U(w, w^2)$ in Fig. 4 returns A_1 in one broadcast and A_2 and A_3 in two broadcasts. Thus, the lemma holds for $k = 2$.

For the inductive step, assume that the lemma holds for $U(w, w^{k-1})$. Our construction of $U(w, w^k)$ guarantees that, after receiving $A_1^1, A_1^2, \ldots, A_1^w$, A_1 and D_1 are available in one further broadcast. Since $A_1^1, A_1^2, \ldots, A_1^w$ are computed in $k - 2$ broadcasts, it follows that A_1 is computed in $k - 1$ broadcasts.

Similarly, A_2 and D_2 are computed in one broadcast after receiving $A_2^1, A_2^2, \ldots, A_2^w$, and D_1. Since all of them are computed in $k - 1$ broadcasts, A_2 and D_2 are computed in k broadcasts. Continuing similarly, it is easy to see that A_i, $(1 \le i \le k)$, is computed in $k + i - 2$ broadcasts and that A_{k+1} is computed in $2k - 2$ broadcasts, as claimed. □

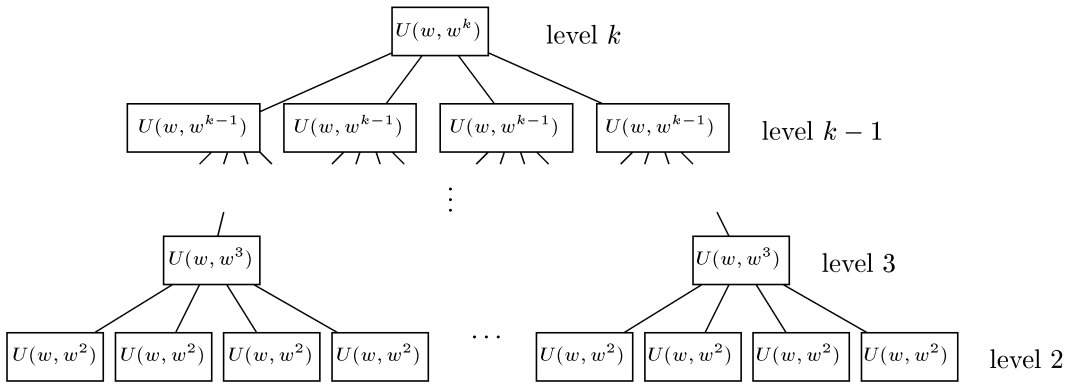


Fig. 7. Illustrating the w-ary tree structure of $U(w, w^k)$ for $w = 4$.

Lemma 3.2. *The construction of block $U(w, w^k)$, $(k \geq 2)$, involves a total of $2w^{k-2} + O(w^{k-3})$ blocks $T(w, m)$ with $m \leq w^2$.*

Proof. Let $g(k)$ be the number of blocks used to construct $U(w, w^k)$. $U(w, w^2)$ uses 2 blocks $T(w, w)$ and $T(w, w^2)$. Thus, $g(2) = 2$. As illustrated in Fig. 9, $U(w, w^k)$ uses w $U(w, w^{k-1})$s, $k-1$ $T(w, w^2)$s, and one $T(w, 2w)$. Hence, $g(k) = w \cdot g(k-1) + k$ holds for $k \geq 2$. By solving this recurrence we obtain

$$\begin{aligned} g(k) &= 2w^{k-2} + \frac{3w^{k-1} - 2w^{k-2} - (k+1)w + k}{(w-1)^2} \\ &= 2w^{k-2} + O(w^{k-3}), \end{aligned}$$

thus completing the proof of the lemma. □

Now Lemmas 3.1 and 3.2 combined imply the following important result.

Theorem 3.3 *The sum of an w^k-bit binary sequence can be computed in $2k-2$ broadcasts using $2w^{k-2} + O(w^{k-3})$ blocks $T(w, m)$, $m \leq w^2$, with all the broadcasts taking place on buses of length at most w^2.*

Notice that in order to compute the sum of w^{k-2} sequences of length w^2 using w^{k-2} $U(w, w^2)$s involves $2w^{k-2}$ blocks. Thus, the construction $U(w, w^k)$ uses $O(w^{k-3})$ additional blocks connecting w^{k-2} blocks $U(w, w^2)$. Since a block uses at most w^3 shift switches, a total of $w^{k+1} + O(w^k)$ switches are used. Thus, our design uses $w + O(1)$ switches per input bit, which is close to optimal in the shift switch implementation.

4 Computing Prefix Sums on Short Buses

The main goal of this section is to propose an efficient hardware-algorithm for BPS computation on short buses. We begin by discussing two extreme cases of such a design. The first one uses a large number of blocks but few broadcasts, while the second uses fewer blocks but many broadcasts.

Given a w^k-bit binary sequence as input, at the one end of the spectrum one can generate all w^k prefix sequences of length $1, 2, \ldots, w^k$ and then compute the sum of each prefix sequence independently, in parallel. It is easy to see that this approach requires only $2k-2$ broadcasts but uses blocks $U(w, 1), U(w, 2), \ldots, U(w, w^k)$, for a total of roughly $w^{2k-2} + O(w^{2k-3})$ blocks.

At the other end of the spectrum we have the design illustrated in Fig. 7. Clearly, the array $S(w, w^k)$ can be simulated, in w^{k-2} broadcasts, by using w^{k-2} arrays $S(w, w^2)$. Thus, the design in Fig. 5 can be simulated by kw^{k-2} linear arrays $S(w, w^2)$. However, the computation requires $O(kw^{k-2})$ broadcasts.

We propose to strike a balance between the two approaches outlined above. The goal is to use almost as few broadcasts as the first alternative, while keeping the number of blocks used close to what the second approach would require. Unlike the design in the previous section that was bottom-up, we shall describe our design in a top-down fashion.

To outline our idea, observe that each block $U(w, w^k)$ has, essentially, the structure of a w-ary tree as illustrated in Fig. 7. The tree has nodes with $k-1$ levels from 2 to k. For an arbitrary node v of level h of this w-ary tree, let $a_{s+1}, a_{s+2}, \ldots, a_{s+w^h}$ be the input offered at the leaves of the subtree rooted at v and refer to Fig. 8. Let $t(v)$ and $p(v)$ denote the *sum of* v and the *prefix sum of* v defined as follows:

$$\begin{aligned} t(v) &= a_{s+1} + a_{s+2} + \cdots + a_{s+w^h}, \\ p(v) &= a_1 + a_2 + \cdots + a_s. \end{aligned}$$

The blocks corresponding to v can compute $t(v)$ locally, but not $p(v)$. Let $v_1, v_2, \ldots, v_w$ be the w children of v specified in left-to-right order. Clearly, for every j, $(1 \leq j \leq w)$, we have

$$p(v_j) = p(v) + t(v_1) + t(v_2) + \cdots + t(v_{j-1}).$$

Hence, by computing $p(v), t(v_1), t(v_2), \ldots, t(v_{w-1})$, we obtain the prefix sum of each of the children of v. This computation proceeds from the root down to the leaves. Once this is done, by computing the local prefix sums at the leaves, we can get the resulting prefix sums of the input sequence.

We now show how this idea can be implemented efficiently. We begin by using a block $U(w, w^k)$ to compute the unary base w representation of $t(v)$ for every node v in the w-ary tree. For this purpose, let $P_{k+1}P_kP_{k-1}\cdots P_1$ be the unary base w representation of the prefix sum $p(v)$ of node v. As before, let $v_1, v_2, \ldots, v_w$ be the children of v and assume

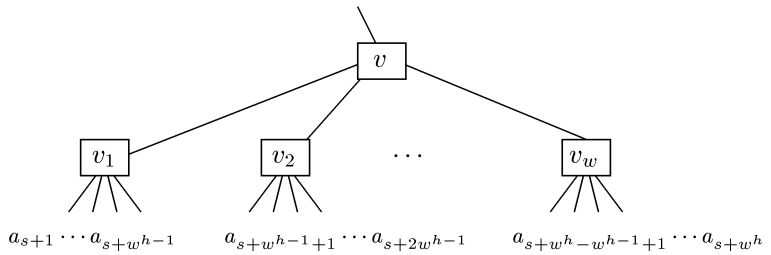


Fig. 8. A node v and its w children $v_1, v_2, \ldots, v_w$.

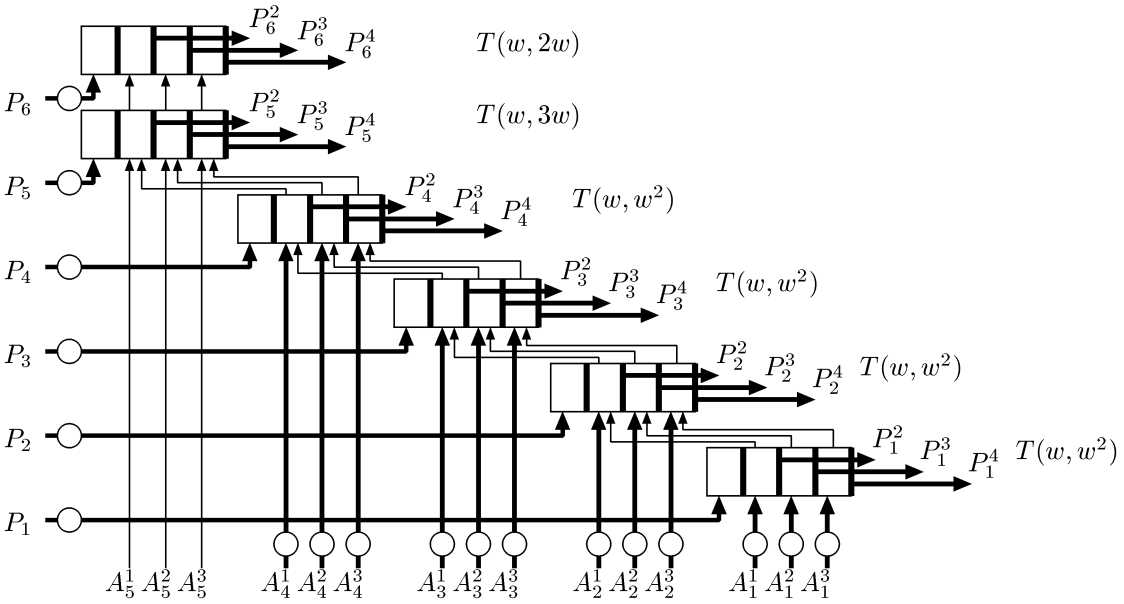


Fig. 9. Illustrating a design for computing the prefix sums $P_k^j P_{k-1}^j \cdots P_1^j$.

that the subtree rooted at v has w^h input bits. Since the subtree rooted at each v_j has w^{h-1} input bits, let each $A_h^j A_{h-1}^j \cdots A_1^j$, $(1 \leq j \leq w)$ denote the unary base w representations of the sum $t(v_j)$. Note that the most significant digit A_h^j is either 0 or 1. Further, let $P_{k+1}^j P_k^j P_{k-1}^j \cdots P_1^j$, $(1 \leq j \leq w)$ be the unary base w representations the prefix sum $p(v_j)$ of v_j. As explained in Section 2.4, for all prefix sums except, perhaps, the last one, the most significant digits P_{k+1} and P_{k+1}^j are always 0 and will be ignored. Our design will have, therefore, to solve the following problem:

Input: The prefix sum $p(v) = P_k P_{k-1} \cdots P_1$ of v and the sum $t(v_j) = A_h^j A_{h-1}^j \cdots A_1^j$ of each child v_j, $(1 \leq j \leq w-1)$ of v;
Output: The prefix sum $p(v_j) = P_k^j P_{k-1}^j \cdots P_1^j$ of each child v_j, $(2 \leq j \leq w)$, of v.

Note that we do not have to compute $p(v_1) = P_k^1 P_{k-1}^1 \cdots P_1^1$ because $p(v_1) = p(v)$. It is easy to confirm that, once this task is completed, the same task can be performed for the children $v_1, v_2, \ldots, v_w$ in order to compute the prefix sums of the grandchildren of v. By continuing in the same fashion until the leaves are reached, we get the final prefix sums.

We refer the reader to Fig. 9 that illustrates this task for $w = 4$, $h = 5$, and $k = 6$. We now provide a detailed explanation of why our design in Fig. 9 works. Clearly, for all j, $(2 \leq j \leq w)$,

$$P_1^j = (P_1 + A_1^1 + A_1^2 + \cdots + A_1^{j-1}) \bmod w \text{ and}$$
$$D_1^j = \left\lfloor \frac{P_1 + A_1^1 + A_1^2 + \cdots + A_1^{j-1}}{w} \right\rfloor.$$

An easy inductive argument shows that for every i, $(2 \leq i \leq h)$,

$$P_i^j = (P_i + A_i^1 + A_i^2 + \cdots + A_i^{j-1} + D_{i-1}^j) \bmod w \quad (6)$$

$$D_i^j = \left\lfloor \frac{P_i + A_i^1 + A_i^2 + \cdots + A_i^{j-1} + D_{i-1}^j}{w} \right\rfloor \quad (7)$$

and that for every i, $(h+1 \leq i \leq k)$,

$$P_i^j = (P_i + D_{i-1}^j) \bmod w \quad (8)$$

$$D_i^j = \left\lfloor \frac{P_i + D_{i-1}^j}{w} \right\rfloor. \quad (9)$$

Recall that each D_h^j, $(1 \leq j \leq w)$, is just one bit. The construction involves $h-1$ blocks $T(w, w^2)$ that are used to compute $P_1^j, P_2^j, \ldots, P_{h-1}^j$ as well as $D_1^j, D_2^j, \ldots, D_{h-1}^j$. Also, one block $T(w, 3w)$ is used to compute P_h^j and D_h^j, and $k-h$ blocks $T(w, 2w)$ are used to compute $P_{h+1}^j, P_{h+2}^j, \ldots, P_k^j$ and $D_{h+1}^j, D_{h+2}^j, \ldots, D_{k-1}^j$. Fig. 10 illustrates the details of the rightmost two $T(w, w^2)$s to compute P_1^j, P_2^j, D_1^j and D_2^j for

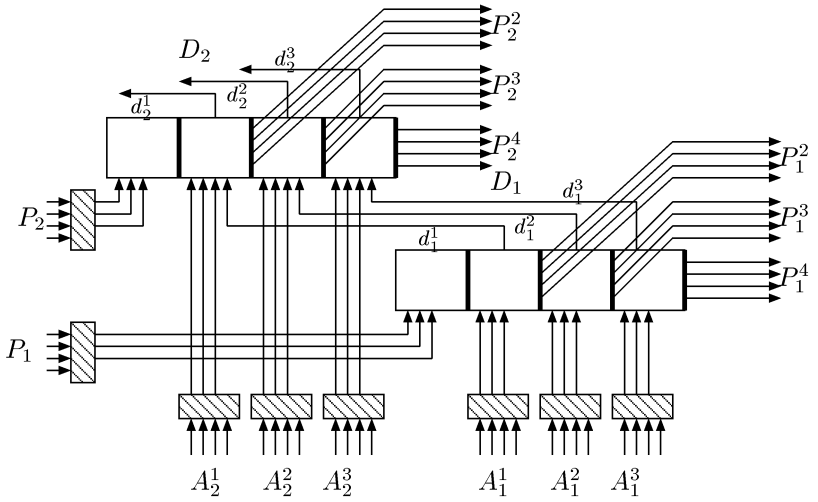


Fig. 10. Illustrating the details of the prefix sums design of Fig. 9.

$w = 4$. In the figure, each of the eight shaded boxes corresponds to the logic circuit for converting from unary to filled unary. The first $T(w, w^2)$ receives the filled unary representation of $P_1, A_1^1, A_1^2, \ldots, A_1^{w-1}$, and computes the prefix sums of $P_1, A_1^1, A_1^2, \ldots, A_1^{w-1}$. It returns the unary representation of $P_1^2, P_1^3, \ldots, P_1^w$ from the $2w\text{th}, 3w\text{th}, \ldots, w^2\text{th}$ $S(w)$s of $T(w, w^2)$, respectively. The output from the top denoted by $D_1 = \{d_1^1, d_1^2, \ldots, d_1^w\}$ involves w bits. The first $w-1$ bits of the prefix sums of D_1 correspond to $D_1^1, D_1^2, \ldots, D_1^{w-1}$. Thus, the second $T(w, w^2)$ computing $P_2^2, P_2^3, \ldots, P_2^w$ receives $A_2^1, d_1^1, A_2^2, d_1^2, \ldots, A_2^{w-1}, d_1^{w-1}$ from the bottom, where each A_1^j $(1 \le j \le w-1)$ is converted to filled unary representation. It also receives P_2 from the left. Since $D_1^j = d_1^1 + d_1^2 + \cdots + d_1^j$, we have the following relations for $2 \le j \le w$ from (6) and (7).

$$P_2^j = (P_2 + A_2^1 + d_1^1 + A_2^2 + d_1^2 + \cdots + A_2^{j-1} + d_1^{j-1}) \bmod w \tag{10}$$

$$D_2^j = \left\lfloor \frac{P_2 + A_2^1 + d_1^1 + A_2^2 + d_1^2 + \cdots + A_2^{j-1} + d_1^{j-1}}{w} \right\rfloor. \tag{11}$$

Hence, the second $T(w, w^2)$ outputs the unary representation of $P_2^1, P_2^2, \ldots, P_2^{w-1}$ from the $2w\text{th}, 3w\text{th}, \ldots, w^2\text{th}$ $S(w)$ of $T(w, w^k)$, respectively. Continuing similarly, we can compute all the P_i^js.

The construction for leaves is different. Since a leaf node has w^2 input bits, the naive approach illustrated in Fig. 5 can be applied. The k blocks of $T(w, w^2)$ are laid as in Fig. 5 and the unary base w of the prefix sum for the leaf node is input from the left. The w^2 input bits are given from the bottom. Then, the unary base w of the prefix sums of the input bits are obtained in each column of the blocks.

Let us now estimate the number of broadcasts used by our design. Each child v_j of the root r of the w-ary tree corresponding to the $U(w, w^k)$ outputs its unary $t(v_j) = A_k^j A_{k-1}^j \cdots A_1^j$ in $2k-4$ broadcasts. Specifically, by Lemma 3.1 A_i^j, $(1 \le i \le k-1)$, is obtained in $k+i-3$ broadcasts, while A_k^j is returned in $2k-4$ broadcasts. By the construction in Fig. 9, P_i^j, $(1 \le i \le k)$, of the child v_i of the root r is computed in $k+i-2$ broadcasts. Similarly, each of the P_i^j in the children of the root, i.e., nodes at level $k-1$, is computed in $k+i-1$ broadcasts. Finally, the unary base w representation of P_i^j in the leaves, i.e., nodes at level 0, is computed in $2k+i-4$ broadcasts. Thus, we have the following result:

Lemma 4.1. *Our prefix sums design returns P_i^j, $(1 \le i \le k; 1 \le j \le w)$, in at most $2k+i-4$ broadcasts.*

Thus, all the prefix sums can be computed in $3k-4$ broadcasts. Next, we estimate the number of blocks used. Since each node uses k blocks, and the tree has $w^{k-2} + w^{k-3} + \cdots + w^0$ nodes, this construction adds $kw^{k-2} + O(kw^{k-3})$ blocks. Since by Theorem 3.3 $U(w, w^k)$ uses $2w^{k-2} + O(w^{k-3})$, a total number of $(k+2)w^{k-2} + O(kw^{k-3})$ blocks are used. Thus we have the following result.

Theorem 4.2. *The prefix sums of an w^k-bit binary sequence can be computed in $3k-4$ broadcasts using $(k+2)w^{k-2} + O(kw^{k-3})$ blocks, with all the broadcasts taking place on buses of length at most w^2.*

5 Computing Sums Using Pipelining

The main goal of this section is to introduce our pipelining technique and to show how it can be used to design an improved hardware-algorithm for computing the sum of kw^k bits.

Suppose that r input sequences $\alpha_1, \alpha_2, \ldots, \alpha_r$ of w^k bits each are given. Let $s(\alpha_i)$, $(1 \le i \le r)$, denote the sum of the w^k bits in α_i. Recall that by Theorem 3.3 a block $U(w, w^k)$ can compute the sum of w^k bits in $2k-2$ broadcasts. Therefore, a single block $U(w, w^k)$ can compute in $r(2k-2)$ broadcasts, proceeding sequentially, $s(\alpha_1), s(\alpha_2), \ldots, s(\alpha_r)$.

Notice that the $U(w, w^k)$ uses each of its blocks and gates once and does not reuse them. Moreover, Lemma 3.1 asserts that with a w^k-bit binary sequence as input, the $U(w, w^k)$ returns the unary base w representation of the corresponding sum in k broadcasts starting with the $k-1\text{th}$ broadcast and continuing until the $2k-2-\text{th}$ broadcast.

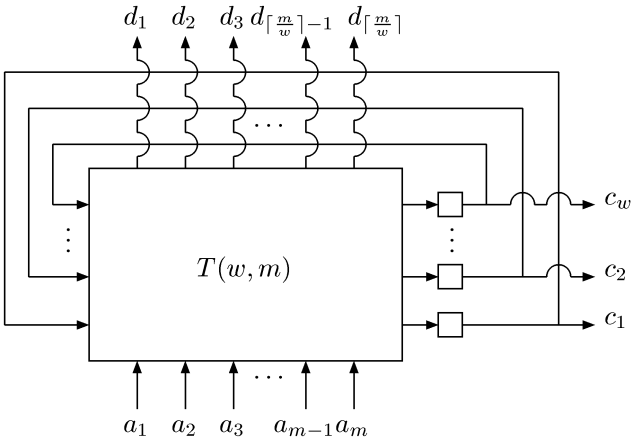


Fig. 11. Illustrating a block $V(w,m)$.

This is the key observation that allows us to pipeline the sequences $\alpha_1, \alpha_2, \ldots, \alpha_r$ through the $U(w,w^k)$ to speed up the computation of $s(\alpha_1), s(\alpha_2), \ldots, s(\alpha_r)$. We begin by supplying α_1 as input, followed by α_2, and so on. As noted above, $s(\alpha_1)$ will emerge in broadcasts $k-1$ through $2k-2$. After one broadcast delay, $s(\alpha_2)$ emerges in the k broadcasts ranging from the kth to the $2k-1$th broadcasts. Continuing similarly, $s(\alpha_r)$ is returned in k broadcasts ranging from the $k+r-2$th to the $2k+r-3$th. Thus, we have the following result.

Lemma 5.1. *The sum of r sequences, each of w^k bits can be computed, individually, in $2k+r-3$ broadcasts using altogether $2w^{k-2}+O(w^{k-3})$ blocks $T(w,m)$ with $m \leq w^2$.*

To implement the pipelining strategy discussed above, we now introduce the block $V(w,m)$ illustrated in Fig. 11. In essence, a $V(w,m)$ consists of a block $T(w,m)$ with w 1-bit registers (alternatively, a one w-bit register) and *feedback* lines as shown in Fig. 11. The w-bit output of the $T(w,m)$ is stored in the w-bit register; in turn, using the feedback lines, the contents of this w-bit is fed back into the $T(w,m)$. The block $V(w,m)$ functions as follows: Suppose that the w-bit register stores the unary representation of zero and the m-bit distributed representation of input sequences $a_1, a_2, \ldots, a_m$ are input to $V(w,m)$ one by one. After a_i, $(1 \leq i \leq m)$, is supplied, one broadcast stores the unary representation of

$$a_1 + a_2 + \cdots + a_i \bmod w,$$

in the w-bit register of $V(w,m)$. Further, at this moment, the concatenation of the bits output from the top of $V(w,m)$ corresponds to the distributed representation of

$$\left\lfloor \frac{a_1 + \alpha_2 + \cdots a_i}{w} \right\rfloor.$$

We are now interested in computing the sum of w^{k+l} bits partitioned into w^l sequences $\alpha_1, \alpha_2, \ldots \alpha_{w^l}$ each of w^k bits. Notice that the unary base w representation of the sum has $k+l+1$ digits. We ignore the most significant digit, because it is 1 only if the all input bits is 1, and otherwise 0. We first assume that blocks $V(w,m)$ for any m are available, and use the cascading architecture illustrated in

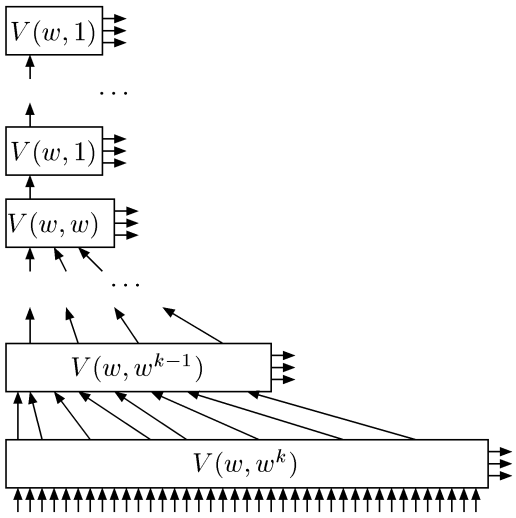


Fig. 12. Cascading $V(w,w^k), V(w,w^{k-1}), \ldots, V(w,1)$.

Fig. 12. The output of each $V(w,w^j)$ $(2 \leq j \leq k)$ is supplied to $V(w,w^{j-1})$. In general, the jth block, $(1 \leq j \leq k+l)$, from the bottom is used to compute the jth digit of the unary of base w representation of the sum. Suppose that every register is initialized by the unary representation of 0 and $\alpha_1, \alpha_2, \ldots \alpha_{w^l}$ are supplied to $V(w,w^k)$ in the interval of one broadcast. It should be clear that the register of $V(w,w^k)$ preserves the least significant digit of the unary base w representation of $(s(\alpha_1)+s(\alpha_2)+\cdots+s(\alpha_i)) \bmod w$, $(1 \leq i \leq w^k)$, after i broadcasts. Thus, the register of jth $(1 \leq j \leq k+l)$ block preserves the jth digit of $(s(\alpha_1)+s(\alpha_2)+\cdots+s(\alpha_i)) \bmod w$ after $i+j-1$ broadcasts. Therefore, the sum $s(\alpha_1)+s(\alpha_2)+\cdots+s(\alpha_{w^l})$ can be computed in $w^l+k+l-1$ broadcasts.

Next, we assume that only blocks $V(w,m)$ with $m \leq w^2$ are available. Suppose that $\alpha_1, \alpha_2, \ldots, \alpha_{w^l}$ of w^k bits are supplied to the architecture in Theorem 3.3 illustrated in Fig. 6. Then, the blocks corresponding to the root of w-ary tree output $s(\alpha_1), s(\alpha_2), \ldots, s(\alpha_{w^l})$ in turn. Our new idea is to compute the sum of these outputs using a variant of the design in Fig. 12. Specifically, we replace the blocks $T(w,2w)$ and $T(w,w^2)$ by $V(w,2w)$ and $V(w,w^2)$, respectively, and remove the logical OR gates. A block $V(w,2)$ is used to receive the output of $V(w,2w)$ and the cascading of $l-1$ $V(w,1)$s is attached to it. These changes only apply to the root of the w-ary structure of $U(w,w^k)$, the blocks corresponding to the other nodes being unaffected. It is easy to see that after a suitable number of broadcasts, the register of the jth block preserves the jth digit of the unary base w representation of the sum. Clearly, this design uses only l additional blocks.

To estimate the number of broadcasts, note that by Lemma 3.1 the least significant digit of $s(\alpha_1)$ is obtained in $k-1$ broadcasts. Thus, the least significant digit of the sum $s(\alpha_1)+s(\alpha_1)+\cdots+s(\alpha_{w^l})$ is obtained in $k+w^l-2$ broadcasts. Since the most significant digit of the sum is computed using $k+l-1$ additional broadcasts, $2k+w^l+l-3$ broadcasts are sufficient to complete the computation. Thus, we have the following result.

Theorem 5.2. *The sum of an w^{k+l}-bit binary sequence can be computed in $2k + w^l + l - 3$ broadcasts using a total of $2w^{k-2} + l + O(w^{k-3})$ blocks $T(w,m)$ and $V(w,m)$ with $m \leq w^2$. Moreover, all the broadcasts take place on buses of length at most w^2.*

Note that each $V(w,m)$ corresponding to the root outputs one of the digit of the unary base w representation of the prefix sum $s(\alpha_1) + s(\alpha_2) + \cdots + s(\alpha_i)$ of each i $(1 \leq i \leq w^k)$ in turn. As a corollary we have:

Corollary 5.3. *The sum of a kw^k-bit binary sequence can be computed in $3k + \lceil \log_w k \rceil - 3$ broadcasts using a total of $2w^{k-2} + O(w^{k-3})$ blocks $T(w,m)$ and $V(w,m)$ with $m \leq w^2$.*

6 Computing Prefix Sums Using Pipelining

The main goal of this section is to show that by using pipelining we can design a hardware-algorithm for BPS computation involving large input sets.

Suppose that the input of w^{k+l} bits is partitioned into w^l subsequences $\alpha_1, \alpha_2, \ldots \alpha_{w^l}$ of w^k bits each. Let us observe the behavior of the prefix sums architecture in Theorem 4.2 when the input $\alpha_1, \alpha_2, \ldots \alpha_{w^l}$ are supplied in the interval of one broadcast. Clearly, it outputs the "local" prefix sums of each of $\alpha_1, \alpha_2, \ldots, \alpha_{w^l}$ in turn from blocks corresponding to the leaves illustrated in Fig. 7. Our idea is to compute the "global" prefix sums of $s(\alpha_1), s(\alpha_2), \ldots, s(\alpha_{w^l})$ and then add them to local prefix sums to get correct prefix sums. Remember that the structure of these blocks is the cascading of $S(w,w^2)$s as illustrated in Fig. 5 for $k = 2$. We use these blocks to perform this addition as follows: Suppose that some block corresponding to the leaves is computing some digit of the prefix sums of input α_i. At this moment, if the same digit of the prefix sum $s(\alpha_1) + s(\alpha_2) + \cdots + s(\alpha_{i-1})$ (i.e., the sum of the first $(i-1)w^k$ bits) is supplied from the left of the block, then it outputs the same digit of the "global" prefix sums instead of the "local" prefix sums. Actually, we can compute the "global" prefix sum $s(\alpha_1) + s(\alpha_2) + \cdots + s(\alpha_{i-1})$ using the improved sum architecture of Theorem 5.2.

We now show how to implement the above idea. The improved prefix sums architecture is similar to the architecture in Theorem 4.2. We use the improved sum architecture in Theorem 5.2 instead of $U(w,w^k)$, in order to compute the prefix sum $s(\alpha_1) + s(\alpha_2) + \cdots + s(\alpha_{i-1})$. Each leaf of the w-ary tree of the architecture in Theorem 4.2 (not for $U(w,w^k)$) to compute the prefix sums has the cascading

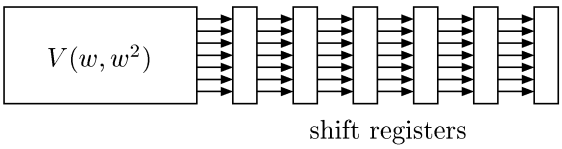


Fig. 13. $V(w,w^2)$ connected with shift registers.

of $k+l$ $S(w,w^2)$s with the same structure of Fig. 5. The additional l $S(w,w^2)$s are used not to lose significant l digits of the unary base w representation of the prefix-sums of w^{k+l} bits. The remaining problem we need to investigate is to adjust the timing to supply the prefix sums to blocks corresponding to the leaves, because the prefix sum that a fixed block wants to receive changes. To maintain the timing, we use the following properties:

- The hth $(1 \leq h \leq k+l)$ block from the bottom in all cascading structure always computes the hth digit of the unary base w representation, and always wants to receive the hth digit of the prefix sum $s(\alpha_1) + s(\alpha_2) + \cdots + s(\alpha_{i-1})$ for some i.
- If a block has to receive the prefix sum $s(\alpha_1) + s(\alpha_2) + \cdots + s(\alpha_{i-1})$ at some moment, then it has to receive $s(\alpha_1) + s(\alpha_2) + \cdots + s(\alpha_i)$ after one broadcast.

Thus, the *shift registers* can be used to adjust the timing. The output of each $V(w,w^2)$ that computes the "global" prefix sums is connected to the shift registers, as illustrated in Fig. 13.

To estimate the number of broadcasts, note that by Lemma 4.1, the least significant digit of the prefix sums of α_1 is computed in $2k-3$ broadcasts. Thus, its most significant digit is computed in $k+l-1$ additional broadcasts, that is, in $3k+l-4$ broadcasts. The most significant digit is computed in turn, so that of α_{w^l} is computed in $w^l - 1$ additional broadcasts. Therefore, all the prefix sums can be computed in the time of $3k + l + w^l - 5$ broadcasts.

We now estimate the number of blocks used. Each leaf must have l additional blocks in order not to lose the l most significant digits. Thus, in all, lw^{k-2} blocks are used for this purpose. Recall that Theorem 4.2 guarantees that the improved hardware-algorithm for BPS requires $(k+l+2)w^{k-2} + O(kw^{k-3})$ blocks. Thus, we have the following result.

Theorem 6.1. *The prefix sums of an w^{k+l}-bit binary sequence can be computed in $3k + l + w^l - 5$ broadcasts using a total of*

TABLE 1
A Summary of Our Results

problem	input bits	max block	# blocks	# broadcasts
BS	w^k	w^k	k	k
BPS	w^k	w^k	k	k
BS	w^k	w^2	$2w^{k-2} + O(w^{k-3})$	$2k-2$
BPS	w^k	w^2	$(k+2)w^{k-2} + O(kw^{k-3})$	$3k-4$
BS	kw^k	w^2	$2w^{k-2} + O(w^{k-3})$	$3k + \lceil \log_w k \rceil - 3$
BPS	kw^k	w^2	$(k + \lceil \log_w k \rceil + 2)w^{k-2} + O(kw^{k-3})$	$4k + \lceil \log_w k \rceil - 5$

$(k+l+2)w^{k-2}+O(kw^{k-3})$ *blocks* $T(w,m)$ *and* $V(w,m)$ *with* $m \leq w^2$.

As a corollary, we have the following result.

Corollary 6.2. *The sum of a* kw^k-bit *binary sequence can be computed in* $4k+\lceil \log_w k \rceil - 5$ *broadcasts using a total of* $(k+\lceil \log_w k \rceil + 2)w^{k-2}+O(kw^{k-3})$ *blocks* $T(w,m)$ *and* $V(w,m)$ *with* $m \leq w^2$.

7 Concluding Remarks

The main contribution of this work is to propose a number of efficient hardware-algorithms for the classical task of computing the sum and the prefix sums of a w^k-bit, $k \geq 2$, binary sequence. We use as basic building blocks linear arrays of at most w^2 shift switches. An immediate consequence of this feature is that in our designs broadcasts are limited to buses of length at most w^2 making them of practical relevance. Table 1 summarizes our results.

Our first hardware-algorithm computes the sum of a w^k-bit binary sequence in the time of $2k-2$ broadcasts, while the corresponding prefix sums can be computed in the time of $3k-4$ broadcasts.

Our second hardware-algorithm operates in pipelined fashion and computes the sum of a kw^k-bit sequence in the time of $3k+\lceil \log_w k \rceil - 3$ broadcasts. Using this design, the corresponding prefix sums can be computed in the time of $4k+\lceil \log_w k \rceil - 5$ broadcasts.

We begin by proposing a first design that allows to compute the sum of a w^k-bit binary sequence in the time of $2k-2$ broadcasts, using $2w^{k-2}+O(w^{k-3})$ blocks. The corresponding prefix sums can be computed in $3k-4$ broadcasts using $(k+2)w^{k-2}+O(kw^{k-3})$ blocks. It would be interesting to see if similar designs can be obtained for a number of other fundamental problems including computing inner products, convolutions, the Fast Fourier Transform (FFT) among many others. This promises to be an exciting avenue for further investigation.

Acknowledgments

The authors wish to thank four anonymous referees for their constructive criticism and many useful comments. These comments helped us clarify the implicit assumptions we made about the VLSI delay model in which we described our results. We also thank R. Vaidyanathan and Hossam ElGindy for useful comments on our delay model and for reading carefully an earlier draft of the paper. This work was supported by US NSF Grants CCR-9522092 and MIP-9630870, by NASA Grant NAS1-19858, by US ONR Grants N00014-95-1-0779 and N00014-97-1-0562, and by ARC Grant 04/15/412/001.

References

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

[2] H. Alnuweiri, M. Alimuddin, and H. Aljunaidi, "Switch Models and Reconfigurable Networks: Tutorial and Partial Survey," *Proc. Workshop on Reconfigurable Architectures,* pp. 1–10, Apr. 1994.

[3] G. Blelloch, "Scans As Primitives Parallel Operations," *IEEE Trans. Computers,* vol. 18, pp. 1,526–1,538, 1989.

[4] R.P. Brent and H.T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. Computers,* vol. 31, pp. 260–264, 1982.

[5] J.J.F. Cavanaugh, *Digital Computer Arithmetic Design and Implementation.* New York: McGraw-Hill, 1984.

[6] L. Dadda and V. Piuri, "Pipelined Adders," *IEEE Trans. Computers,* vol. 45, pp. 348–356, 1996.

[7] F. Halsall, *Data Communications, Computer Networks and Open Systems.* Addison-Wesley, 1996.

[8] R.H. Katz, *Contemporary Logic Design.* Benjamin/Cummings Publishing, 1994.

[9] U. Ko, P. T. Balsara, and W. Lee, "Low-Power Design Techniques for High-Performance CMOS Adders," *IEEE Trans. VLSI Systems,* vol. 3, pp. 327–333, 1995.

[10] P.M. Kogge and H.S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrences," *IEEE Trans. Computers,* vol. 22, pp. 786–793, 1973.

[11] H.T. Kung and C.E. Leiserson, "Algorithms for VLSI Processor Arrays," *Introduction to VLSI Systems,* C. Mead and L. Conway, eds. Reading, Mass.: Addison-Wesley, 1980.

[12] R.E. Ladner and M.J. Fischer, "Parallel Prefix Computation," *J. ACM,* vol. 27, pp. 831–838, 1980.

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

[14] H. Li and M Maresca, "Polymorphic-Torus Network," *IEEE Trans. Computers,* vol. 38, pp. 1,345–1,351, 1989.

[15] M.-B. Lin and A.Y. Oru, "The Design of an Optoelectric Arithmetic Processor Based on Permutation Networks," *IEEE Trans. Computers,* vol. 46, pp. 142–153, 1997.

[16] R. Lin and S. Olariu, "Reconfigurable Buses with Shift Switching—Architectures and Applications," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, pp. 93–102 1995.

[17] N. Lindert, T. Sugii, S. Tang, C. Hu, "Dynamic Threshold Pass-Transistor Logic for Improved Delay at Lower Power Supply Voltages," *IEEE J. Solid-State Circuits,* vol. 34, pp. 85–89, 1999.

[18] M. Maresca, "Polymorphic Processor Arrays," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, pp. 490–506, 1993.

[19] R. Miller, V.K.P. Kumar, D. Reisis, and Q. F. Stout, "Parallel Computations on Reconfigurable Meshes," *IEEE Trans. Computers,* vol. 42, pp. 678–692, 1993.

[20] K. Nakano, "An Efficient Algorithm for Summing up Binary Values on a Reconfigurable Mesh," *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences,* vol. E77-A, no. 4, pp. 652–657, 1994.

[21] K. Nakano, "Prefix-Sums Algorithms on Reconfigurable Meshes," *Parallel Processing Letters,* vol. 5, pp. 23-35, 1995.

[22] K. Nakano and S. Olariu, "An Efficient Algorithm for Row Minima Computations on Basic Reconfigurable Meshes," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, pp. 561–569, 1998

[23] K. Nakano, "A Bibliography of Published Papers on Dynamically Reconfigurable Architectures," *Parallel Processing Letters,* vol. 5, pp. 111-124, 1995.

[24] S. Olariu, J.L. Schwing, and J. Zhang, "Fundamental Data Movement Algorithms for Reconfigurable Meshes," *Int'l J. High Speed Computing,* vol. 6, pp. 311–323, 1994.

[25] T.-H. Liu, M.K. Ganai, A. Aziz, and J.L. Burns, "Performance Driven Synthesis for Pass-Transistor Logic," *Proc. 12th IEEE Int'l Conf. VLSI Design,* pp. 372–377, 1999.

[26] W.-H. Paik, S.-W. Kim, "Sum-Selector Generation Algorithm Based 64-Bit Adder Using Dynamic Chain Architecture," *Proc. Fourth IEEE Int'l Conf. Electronics, Circuits and Systems,* vol. 3, pp. 1,020-1,024, 1997.

[27] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford Univ. Press, 2000.

[28] T. Stourakis, S.W. Kim, and A. Skavantzos, "Full-Adder Based Arithmetic Units in Finite Fields," *IEEE Trans. Circuits and Systems II,* vol. 40, pp. 741–745, 1993.

[29] M. Suzuki, K. Shimbo, T. Yamanaka, A. Shimizu, K. Sasaki, and Y. Nakangone, "A 1.5ns CMOS ALU in Double Pass-Transistor Logic," *IEEE J. Solid-State Circuits,* vol. 28, pp. 1,145–1,151, 1993.

[30] E.E. Swartzlander, Jr.,, *Computer Arithmetic,* vol. 1. IEEE CS Press, 1990.

[31] E.E. Swartzlander, Jr.,, "Parallel Counters," *IEEE Trans. Computers,* vol. 22, pp. 1,021–1,024, 1973.

[32] Thinking Machines Corporation, "Connection Machine Parallel Instruction Set (PARIS)," July 1986.

[33] J.D. Ullman, *Computational Aspects of VLSI.* Rockville, Md.: Computer Science Press, 1984.

[34] N.H.E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, a Systems Perspective,* p. 525. Addison-Wesley, 1993.
[35] R. Zimmermann and W. Fichtner, "Low-Power Logic Styles: CMOS Versus Pass-Transistor Logic," *IEEE J. Solid-State Circuits,* vol. 32, pp. 1,079–1,090, 1997.

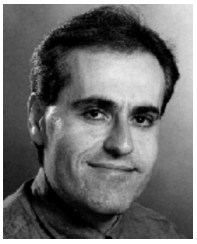
Rong Lin received the BS degree in mathematics from Peking University, Beijing China, the MS degree in computer science from Beijing Polytechnic University, Beijing China, and PhD degree in computer science from Old Dominion University, Norfolk, Virginia, in 1989. He Joined the faculty of State University of New York at Geneseo in September 1989, where he now is a professor and the chair of the computer science department. Dr. Lin's current research interests include parallel architectures, VLSI arithmetic circuits, run-time-reconfigurable digital circuits, and parallel algorithm designs. He is a member of the IEEE Computer Society.

Koji Nakano received the BE, ME, and PhD degrees from Osaka University, Japan in 1987, 1989, and 1992, respectively. From 1992-1995, he was a research scientist at Advanced Research Laboratory, Hitachi Ltd. Since 1995, he has worked at Nagoya Institute of Technology, Japan. He is currently an associate professor with the Department of Electrical and Computer Engineering. His research interests includes parallel algorithms and architectures, computational complexity, and graph theory. He is a member of the IEEE Computer Society.

Stephan Olariu received the MSc and PhD degrees in computer science from McGill University, Montreal in 1983 and 1986, respectively. In 1986, he joined Old Dominion University where he is a professor of computer science. Dr. Olariu has published extensively in various journals, book chapters, and conference proceedings. His research interests include image processing and machine vision, parallel architectures, design and analysis of parallel algorithms, computational graph theory, computational geometry, and mobile computing. Dr. Olariu serves on the editorial board of the *IEEE Transactions on Parallel and Distributed Systems*, the *Journal of Parallel and Distributed Computing, VLSI Design, Parallel Algorithms and Applications, International Journal of Computer Mathematics*, and the *International Journal of Foundations of Computer Science*. He is a member of the IEEE Computer Society.

M. Cristina Pinotti received the PhD degree cum laude in computer science from the Univesity of Pisa, Italy, in 1986. Since 1987 she has been a Researcher with the National Council of Research at the Istituto di Elaborazione dell'Informazione, Pisa. In 1994 and 1995 she was a Research Associate at the Department of Computers Sciences, University of North Texas. In 1997 she visited the Department of Computer Scienve, Old Dominion University. Her research interests include computer arithmetic, residue number systems, VLSI special purpose architectures, design and analysis of parallel algorithms, parallel data structures, distributed data structures and multiprocessor interconnection networks. She is a member of the IEEE Computer Society.

James L. Schwing (M '83) received the BS in mathematics from Worcester Polytechnic Institute in 1970 and the PhD in mathematics from the University of Utah in 1976. He has recently joined the faculty at Central Washington University as chair of the Computer Science Department. Dr. Schwing's research interests include parallel algorithms, human-computer interaction, computer graphics, and computer-aided design. Dr. Schwing serves on the editorial board of the *International Journal of Computer Mathematics*. He is a member of the IEEE and the ACM.



Albert Y. Zomaya received the PhD from the Department of Automatic Control and Systems Engineering, Sheffield University, United Kingdom. He is a professor and deputy-head in the Department of Electrical and Electronic Engineering at the University of Western Australia, where he also leads the Parallel Computing Research Laboratory. Also, he held visiting positions at Waterloo University (Canada) and The University of Missouri-Rolla (USA). He is the author/coauthor of five books, more than one hundred publications in technical journals, collaborative books, and conferences, and the editor of three volumes and three conference proceedings. He is currently an associate editor for the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Systems, Man, and Cybernetics* (Parts A, B, and C), the *Journal of Parallel Algorithms and Applications*, the *Journal of Interconnection Networks*, the *International Journal on Parallel and Distributed Systems and Networks*, the *Journal of Future Generation of Computer Systems*, and the *International Journal of Foundations of Computer Science*. He is also the founding editor of the Wiley Book Series on parallel and distributed computing. He is the editor-in-chief of the *Parallel and Distributed Computing Handbook* (McGraw-Hill, 1996). Professor Zomaya was elected in July 1999 to chair the IEEE Technical Committee on Parallel Processing. He is also a board member of the International Federation of Automatic Control (IFAC) committee on Algorithms and Architectures for Real-Time Control, and serves on the executive board of the IEEE Task Force on Cluster Computing. Professor Zomaya is a chartered engineer and a senior member of the IEEE, a member of the ACM, and a member of the Institute of Electrical Engineers (UK), the New York Academy of Sciences, the Association for the Advancement of Science, and Sigma Xi. His research interests are in the areas of high performance computing, parallel and distributed algorithms, scheduling and load-balancing, computational machine learning, scientific computing, adaptive computing systems, mobile computing, data mining, and cluster-, network-, and mega-computing.